# LinkPool LiquidSDIndexPool Audit Report

Version 2.0

*Cyfrin.io*

March 10, 2023

# Cyfrin LiquidSDIndexPool Mitigation Audit Report

Cyfrin.io

March 7, 2023

## LinkPool LiquidSDIndexPool Audit Report

Version 1.0

Prepared by: Cyfrin Lead Auditors:

- Patrick Collins
- Ben Sacchetti

Assisting Auditors:

- Giovanni Di Siena
- Hans

## Table of Contents

- [G-3] Using bools for storage incurs overhead
- [G-4] Cache array length outside of loop
- [G-5] State variables should be cached in stack variables rather than re-reading them from storage
- [G-6] Use calldata instead of memory for function arguments that do not get mutated
- [G-7] Use Custom Errors
- [G-8] Don't initialize variables with default value
- [G-9] Long revert strings
- [G-10] Functions guaranteed to revert when called by normal users can be marked `payable`
- [G-11] `++i` costs less gas than `i++`, especially when it's used in **`for`**-loops (`--i`/`i--` too)
- [G-12] Use != 0 instead of > 0 for unsigned integer comparison

### Disclaimer

The Cyfrin team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed to two weeks, and the review of the code is solely on the security aspects of the solidity implementation of the contracts.

## Protocol Summary

The LinkPool LiquidSDIndexPool protocol allows users to deposit liquid staking derivative tokens (LSDs) like Rocket Pool ETH (rETH) & Lido ETH (stETH) and, by doing so, receive a token that represents holding a basket of these assets in return. The protocol makes a fee on withdrawls.

This product intends to provide exposure to ETH Staking by averaging rate of the interest across multiple staked ETH derivative protocols.

## Audit Details

### Scope Of Audit

Between Februrary 6th 2023 - Feb 17th 2023, the Cyfrin team conducted an audit on the `liquidSDIndex` folder of their staking-contracts-v2 repository. The scope of the audit was as follows:

1. Full audit of the single folder of contracts in the git repository specified by linkpool
    1. Commit hash: 7084a32 of staking-contracts-v2
    2. Contracts in the `liquidSDIndex` folder: `staking-contracts-v2/contracts/liquidSDIndex/`
2. Out of scope
    1. The test folder & test contracts in `liquidSDIndex` folder

### Severity Criteria

- High: Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).
- Medium: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.
- Low: Low impact and low/medium likelihood events where assets are not at risk (or a trivia amount of assets are), state handling might be off, functions are incorrect as to natspec, issues with comments, etc.
- Informational / Non-Critial: A non-security issue, like a suggested code improvement, a comment, a renamed variable, etc. Auditors did not attempt to find an exhaustive list of these.

- Gas: Gas saving / performance suggestions. Auditors did not attempt to find an exhaustive list of these.

### Summary Of Findings

We highly recommend writing fuzz & invariant tests to catch these issues moving forward.

High - 2

Medium - 5

Low - 10

*Key: Ack == Acknowledged*

| Finding | Severity | Status |
|---|---|---|
| H-1 Protocol fees become unrecoverable | High | Closed |

| Finding | Severity | Status |
|---|---|---|
| H-2 RocketPoolRETHAdapter rate is reversed | High | Closed |
| M-1 Hardcoded Lido exchange rate potentially creates MEV, arbitrage, and remove value from protocol | Medium | Ack |
| M-2 Reentrancy risk in `deposit` function | Medium | Resolved |
| M-3 No tolerance check in initialization | Medium | Resolved |
| M-4 Loss of precision circumvents protocol fees | Medium | Closed |
| M-5 Centralization risk for trusted owners | Medium | Ack |
| L-1 Lack of events make data migrations & use of indexing services difficult | Low | Resolved |
| L-2 Transfer allowance of adapters could be 0 in the distant future | Low | Ack |
| L-3 Shadow declaration of local variables | Low | Resolved |
| L-4 Calling `getWithdrawalAmounts` with more than the protocol has deposited panics | Low | Resolved |
| L-5 Loss of precision in `getWithdrawalAmounts` | Low | Closed |
| L-6 Getters can revert | Low | Resolved |
| L-7 Empty function body - consider commenting why | Low | Ack |
| L-8 Initializers could be front-run | Low | Ack |
| L-9 Protect against changing storage layout | Low | Resolved |
| L-10 Revert on zero deposit | Low | Resolved |
| I-1 Use predefined constants instead of arbitrary numbers for code readbility | Info | Resolved |
| I-2 `totalDeposits` is used as an overloaded term, consider renaming variables | Info | Ack |
| I-3 Fuzz testing (and invariant testing) | Info | Ack |
| I-4 Use Internal Function for Code Reuse | Info | Ack |
| I-5 LiquidSDIndexPool totalSupply doesn't follow the ERC20 standard | Info | Ack |
| I-6 Functions not used internally could be marked external | Info | Ack |
| I-7 Return values of `approve()` not checked | Info | Ack |
| I-8 Missing checks for `address(0)` when assigning values to address state variables | Info | Ack |
| I-9 Mitigate fee rounding errors in `updateRewards` | Low | Ack |

### Tools used

- Slither
- 4naly3er
- foundry
- Hardhat
- Solodit

## High Findings

### [H-1] Protocol fees become unrecoverable

The protocol takes fees from users when they withdraw, keeping them locked in the contract. However, withdrawal fees are unable to be removed from the protocol and so can become unrecoverable.

#### Description

The issue starts here

```
totalDeposits -= _amount - _getWithdrawalFeeAmount(_amount);
```

`totalDeposits` keeps track of the amount of user funds in the protocol. However, on this line, the protocol is still counting the withdrawl fee as user funds, even though all the user's iETH receipts have been burned. So no one user can take the funds, but the fee holders don't have a claim on them either.

Additionally, even after adjusting this you will find that the fees are still unrecoverable.

Scenario: - Fees are 5% - User deposits 1000 stETH for 1000 iETH - User withdraws 950 stETH for 1000 iETH
- There are 50 stETH left in the protocol as a fee - Attempt to claim the 50 stETH, the protocol thinks they are user deposits and won't withdraw

Foundry Test Example:

```
function test_unreachableWithdrawalFees() public {
        vm.startPrank(owner);
        pool.setWithdrawalFee(MAX_WITHDRAWAL_FEE);
        pool.setCompositionEnforcementThreshold(10000e18);
        vm.stopPrank();

        vm.startPrank(user);
```

```
        lsdTokenA.mintShares(user, 1000e18);
        lsdTokenA.approve(address(pool), type(uint256).max);
        pool.deposit(address(lsdTokenA), 1000e18);

        pool.withdraw(1000e18);
        vm.stopPrank();

        vm.startPrank(owner);
        uint256 basisPointsToAdd = MAX_FEE_BASIS_POINTS - pool.
            _totalFeesBasisPointsPublic();
        pool.addFee(owner, basisPointsToAdd);
        vm.stopPrank();

        pool.updateRewards();

        LiquidSDIndexPool.Fee[] memory feeHolders = pool.getFees();
        for (uint256 index = 0; index < feeHolders.length; index++) {
            vm.startPrank(feeHolders[index].receiver);
            pool.withdraw(pool.balanceOf(feeHolders[index].receiver));
            vm.stopPrank();
        }

        assert(pool.totalShares() < 1e18);
        assert(pool.totalSupply() < 1e18);
    }
```

You can find the above test in our forked test suite.

#### Mitigation

Adjust the code so `totalDeposits` is accuratly updated and consider tracking fees separately from deposits. Then, solve the above test based on how you'd like to see fees processed. One suggestion would be to keep track of the withdrawl fees.

*Recommended: Write fuzz/invariant tests to catch these. We have a minimal example you can use as a base-line with a test that already fails to test against.*

#### c0877e0 Resolution

Linkpool states that to withdraw all funds, the admin must set the withdrawal fee to 0 before users can withdraw their underlying assets at a higher rate. Cyfrin has confirmed this using stateful fuzz tests. We note that this means the admin team has full discretion on when the fees could be taken.

### [H-2] RocketPoolRETHAdapter exchange rate is reversed

#### Description

The getUnderlyingByLsd function requires the Underlying/LSD price to be accurate. For the case of RocketPool, that would be ETH/rETH and the adpater calls the following:

```
function getExchangeRate() public view override returns (uint256) {
    return IRocketPoolRETH(address(token)).getExchangeRate();
}
```

But looking at RocketTokenRETH.sol it returns the rETH/ETH rate instead.

```
rETH/ETH != ETH/rETH
```

This is backwards.

#### Mitigation

Fix the adapter to return the correct reversed rate.

*Recommended: Write fuzz tests that account for this variable changing. We have a minimal example you can use as a base-line with a test that already fails to test against.*

#### c0877e0 Resolution

Cyfrin and Linkpool have concluded that the original code is correct. The rocketpool contract returns the `ETH/rETH` price feed and not the `rETH/ETH` price feed.

## Medium Findings

### [M-1] Hardcoded Lido exchange rate potentially creates MEV, arbitrage, and remove value from protocol

#### Description

Lido is currently behind a DAO/Proxy where they could enable withdrawals. The `LidoSTETHAdapter.sol` currently hard codes the `stETH` -> `ETH` exchange rate.

```
    function getExchangeRate() public view override returns (uint256) {
        return 1 ether;
    }
```

However, if/when Lido enables withdrawals, this assumption may not hold, creating an arbitrage opportunity at the expense of the protocol.

For example:

1. `stETH` and `rETH` (RocketPool) are the LSDToken integrated with `LiquidSDIndexPool.sol` with a hard coded exchange rate of 1 ETH = 1 stETH
    1. Composition is 50/50, tolerance is 50%, and there are 5,000 stETH and 4,000 rETH in the protocol
2. Lido enables withdrawals and makes stETH an exchange rate based token as opposed to a rebasing token, similar to Compound
3. The true exchange rate is 1 ETH = 1.1 stETH, but the protocol is still assuming 1 ETH = 1 stETH
4. There is now a race to get all the stETH out of the protocol.
5. Validators can move transactions around so `withdraw` transactions benefit them.

Or, if the true exchange rate is in the other direction (1 stETH = 0.9 ETH for example), the protocol could be exploited by moving stETH in and withdrawing rETH.

#### Mitigation

To ensure this doesn't happen, either:

1. Disallow withdrawls until Lido allows withdrawls
2. Disallow Lido as a valid LSD for the protocol
3. Acknowledge the risk and move forward

#### c0877e0 Resolution

This was acknowledged by the LinkPool team.

### [M-2] Reentrancy Risk in `deposit` function

#### Description

The deposit function in `LiquidSDIndexPool.sol` violates CEI (Checks, Effects, Interactions), and due to this is a reentrancy risk:

```
1     function deposit(address _lsdToken, uint256 _amount) external
          tokenIsSupported(_lsdToken) notPaused {
2         require(getDepositRoom(_lsdToken) >= _amount, "Insufficient
              deposit room for the selected lsd");
3         ILiquidSDAdapter lsdAdapter = lsdAdapters[_lsdToken];
4         IERC20Upgradeable(_lsdToken).safeTransferFrom(msg.sender,
              address(lsdAdapter), _amount); // @reentrancy
5         uint256 underlyingAmount = lsdAdapter.getUnderlyingByLSD(
              _amount);
6         _mint(msg.sender, underlyingAmount);
7         totalDeposits += underlyingAmount;
8     }
```

#### Mitigation

Reorganize function to prevent reentrancy and conform to CEI:

```
1     function deposit(address _lsdToken, uint256 _amount) external
          tokenIsSupported(_lsdToken) notPaused {
2         // Checks
3         require(getDepositRoom(_lsdToken) >= _amount, "Insufficient
              deposit room for the selected lsd");
4
5         // Effects
6         ILiquidSDAdapter lsdAdapter = lsdAdapters[_lsdToken];
7         uint256 underlyingAmount = lsdAdapter.getUnderlyingByLSD(
              _amount);
8         totalDeposits += underlyingAmount;
9         _mint(msg.sender, underlyingAmount);
10
11        // Interactions
12        IERC20Upgradeable(_lsdToken).safeTransferFrom(msg.sender,
              address(lsdAdapter), _amount);
13    }
```

#### c0877e0 Resolution

The `deposit` function now correctly updates the state of the contract before making an external call. It should be noted that an event is still emitted *after* an external function call, which is a potential issue if the contract upgrades by way of contract migration by replaying events.

```
1     function deposit(address _lsdToken, uint256 _amount) external
          tokenIsSupported(_lsdToken) notPaused {
2         require(getDepositRoom(_lsdToken) >= _amount, "Insufficient
              deposit room for the selected lsd");
3
```

```
4           ILiquidSDAdapter lsdAdapter = lsdAdapters[_lsdToken];
5
6           uint256 underlyingAmount = lsdAdapter.getUnderlyingByLSD(
                _amount);
7           require(underlyingAmount != 0, "Deposit amount too small");
8
9           _mint(msg.sender, underlyingAmount);
10          totalStaked += underlyingAmount;
11
12          IERC20Upgradeable(_lsdToken).safeTransferFrom(msg.sender,
                address(lsdAdapter), _amount);
13
14          emit Deposit(msg.sender, _lsdToken, _amount); // <-
15      }
```

### [M-3] No tolerance check during initialization

#### Description

In the `LiquidSDIndexPool.sol` initializer, there is no composition tolerance check.

```
1 compositionTolerance = _compositionTolerance;
```

This allows the protocol to have a tolerance above 100%, impacting funds downstream.

#### Mitigation

Use the `setCompositionTolerance` fuction, which performs the check, in the initializer.

#### c0877e0 Resolution

The initializer now correctly uses the `setCompositionTolerance` function.

```
1  function initialize(
2          string memory _derivativeTokenName,
3          string memory _derivativeTokenSymbol,
4          uint256 _compositionTolerance,
5          uint256 _compositionEnforcementThreshold,
6          Fee[] calldata _fees,
7          uint256 _withdrawalFee
8      ) public initializer {
9          __StakingRewardsPool_init(address(0), _derivativeTokenName,
               _derivativeTokenSymbol);
10         setCompositionTolerance(_compositionTolerance);
```

```
11          setCompositionEnforcementThreshold(
                _compositionEnforcementThreshold);
12          setWithdrawalFee(_withdrawalFee);
13          for (uint256 i = 0; i < _fees.length; i++) {
14              fees.push(_fees[i]);
15          }
16          require(_totalFeesBasisPoints() <= 5000, "Total fees must be <=
                50%");
17      }
```

### [M-4] Loss of precision circumvents protocol fees

#### Description

Often in the contract, division by `10000` is performed as a way to represent 100%. However, this can lead to loss of precision. For example, if a user has `10000` LSD deposited into the protocol and goes to withdraw `9999`, they will be charged a fee of `0`.

```
1       // One could write these lines in liquid-id-index-pool.test.ts and
            see the output
2       await pool.connect(signers[1]).deposit(lsd1.address, 10000)
3       // this line prints out 0, circumventing protocol fees
4       console.log((await pool.getWithdrawalAmounts(9999)).toString())
5       // withdrawing 9999 and then 1 will result in a fee-free withdrawal
            of all 10000
6       console.log("Starting balance: ", (await lsd1.balanceOf(await
            signers[1].getAddress())).toString())
7       await pool.connect(signers[1]).withdraw(9999)
8       await pool.connect(signers[1]).withdraw(1)
9       console.log("Ending balance: ", (await lsd1.balanceOf(await signers
            [1].getAddress())).toString())
```

#### Mitigation

Require a minimum deposit/withdrawal of `10000`, and use `10000` as the smallest unit of precision.

#### c0877e0 Resolution

The fuzz test was incorrectly configured (using 0 for a `withdrawalfee`), resulting in missing fees. This issue is closed, with a note that potential misconfiguration is possible.

### [M-5] Centralization Risk for trusted owners

#### Description

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

*Instances (10)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

313:     function addLSDToken(address _lsdToken, address _lsdAdapter, uint256[] calldata _compositionTargets) external onlyOwner {

337:     ) external onlyOwner tokenIsSupported(_lsdToken) {

369:     function setCompositionTargets(uint256[] memory _compositionTargets) external onlyOwner {

388:     function setCompositionTolerance(uint256 _compositionTolerance) external onlyOwner {

401:     function setCompositionEnforcementThreshold(uint256 _compositionEnforcementThreshold) external onlyOwner {

409:     function setWithdrawalFee(uint256 _withdrawalFee) external onlyOwner {

419:     function addFee(address _receiver, uint256 _feeBasisPoints) external onlyOwner {

430:     function updateFee(uint256 _index, address _receiver, uint256 _feeBasisPoints) external onlyOwner {

448:     function setPaused(bool _isPaused) external onlyOwner {
```

```
File: ./contracts/liquidSDIndex/base/LiquidSDAdapter.sol

67:     function _authorizeUpgrade(address) internal override onlyOwner {}
```

#### Mitigation

Acknowledge, or rewrite protocol to remove centralized controls.

*This finding was found by 4naly3er.*

#### c0877e0 Resolution

Acknowledged.

## Low Findings

### [L-1] Lack of events makes data migrations & use of indexing services difficult

#### Description

In the event of a data migration to a new contract, upadating storage mappings in new contracts is substantially more difficult without events.

**Important: Depending on the desired integration with other web3 services, this could be a Medium finding.**

#### Mitigation

Add events to make a future data migration easier, especially when updating mappings.

To make it easier for indexing services to track the protocol, also add events when updating arrays and storage variables.

As a rule of thumb, emit an event any time a storage value changes.

#### c0877e0 Resolution

Events have been added and now are being correctly emitted.

```
1     event Deposit(address indexed account, address indexed token,
          uint256 amount);
2     event Withdraw(address indexed account, address[] tokens, uint256[]
           amounts);
3     event UpdateRewards(address indexed account, uint256 totalStaked,
          int rewardsAmount, uint256 totalFees);
4     event AddLSDToken(address indexed lsdToken, address lsdAdapter,
          uint256[] compositionTargets);
5     event RemoveLSDToken(address indexed lsdToken, uint256[]
          compositionTargets);
6     event SetCompositionTargets(uint256[] compositionTargets);
7     event SetCompositionTolerance(uint256 compositionTolerance);
8     event SetCompositionEnforcementThreshold(uint256
          compositionEnforcementThreshold);
```

```
9       event SetWithdrawalFee(uint256 withdrawalFee);
10      event AddFee(address indexed receiver, uint256 feeBasisPoints);
11      event UpdateFee(address indexed receiver, uint256 feeBasisPoints);
```

Note that one line should be updated to conform with code style (`int256` instead of **`int`**):

```
event UpdateRewards(address indexed account, uint256 totalStaked, int
    rewardsAmount, uint256 totalFees);
```

to

```
event UpdateRewards(address indexed account, uint256 totalStaked,
    int256 rewardsAmount, uint256 totalFees);
```

### [L-2] Transfer allowance of adapters could be 0 in the distant future

#### Description

Many ERC20 tokens reduce the allowance of a spender after every transfer. The adapter contracts are given the maximum allowance during initialization, but that's the only time allowance is set.

```
1       function __LiquidSDAdapter_init(address _token, address _indexPool)
            public onlyInitializing {
2           token = IERC20Upgradeable(_token);
3           token.approve(_indexPool, type(uint256).max);
4       .
5       .
6       }
```

A contract like stETH reduces allowance after every transfer.

```
1       function transferFrom(address _sender, address _recipient, uint256
            _amount) public returns (bool) {
2           uint256 currentAllowance = allowances[_sender][msg.sender];
3           require(currentAllowance >= _amount, "
                TRANSFER_AMOUNT_EXCEEDS_ALLOWANCE");
4           _transfer(_sender, _recipient, _amount);
5           _approve(_sender, msg.sender, currentAllowance.sub(_amount));
6           return true;
7       }
```

At some point in the distant future, if enough people use the protocol, the allowance could be 0, freezing the protocol.

#### Mitigation

Add a function that anyone can call to the adapter base contract to set the allowance to the maximum value.

```
    function updateAllowance() public {
        token.approve(indexPool, type(uint256).max);
    }
```

#### c0877e0 Resolution

Acknowledged.

### [L-3] Shadow declaration of local variables

In `LiquidSDIndexPool.sol` the following code is inside a for loop:

```
uint256 deposits = lsdAdapters[lsdToken].getTotalDeposits();
.
.
.
uint256 compositionTarget = compositionTargets[_lsdToken];
```

And then additionally outside of it.

#### Mitigation

Rename one of the variables to something more specific.

#### c0877e0 Resolution

The variables have been renamed.

```
        uint256 depositTokenCompositionTarget = compositionTargets[
            _lsdToken];
        uint256 depositTokenDeposits = lsdAdapters[_lsdToken].
            getTotalDeposits();
```

### [L-4] Calling `getWithdrawalAmounts` with more than the protocol has deposited panics

#### Description

If the protocol has 0 deposits and a user calls `getWithdrawalAmounts` with a non-zero amount, the protocol will panic.

```
Error: VM Exception while processing transaction: reverted with panic
    code 0x11 (Arithmetic operation underflowed or overflowed outside of
     an unchecked block)
     at LiquidSDIndexPool.getWithdrawalAmounts (contracts/liquidSDIndex/
        LiquidSDIndexPool.sol:280)
```

#### Mitigation

Handle gracefully, for example, if 0 deposits, return 0.

#### c0877e0 Resolution

`require` step added to ensure that users are not withdrawing more than the protocol has.

```
require(_amount <= totalStaked, "Cannot withdraw more than total staked
    amount");
```

### [L-5] Loss of precision in `getWithdrawalAmounts`

#### Description

With small amounts of funds, precision can be lost. This is a low severity finding, rather than medium, because it is mitigated later in the function.

```
uint256 newTargetDepositsOfToken = (newDepositsTotal *
    compositionTargets[lsdTokens[i]]) / 10000;
```

Example: If `newDepositsTotal` is 9999 and `compositionTargets[lsdTokens[i]]` is 1, `newTargetDepositsOfToken` will incorrectly be `0`.

And:

```
uint256 minThreshold = (compositionEnforcementThreshold *
    compositionTarget) / 10000;
```

Example: If `compositionEnforcementThreshold` is `1` and `compositionTarget` is `1`, `minThreshold` will incorrectly be `0`.

#### Mitigation

Require minimum deposit of `10000` and use `10000` as the smallest unit of precision. Have `compositionEnforcementThreshold` have a minimum of `10000`.

#### c0877e0 Resolution

Cyfrin and Linkpool have concluded that the tests were not accurate, and precision loss is insignificant.

### [L-6] Getters can revert

#### Description

The following invariant test can fail:

```
function invariant_gettersShouldNeverRevert() public {
    pool.compositionEnforcementThreshold();
    pool.compositionTolerance();
    pool.getComposition();
    pool.getCompositionTargets();
    pool.getFees();
    pool.getLSDTokens();
    pool.getRewards();
}
```

`getComposition` can fail, if there are 0 deposits.

```
composition[i] = (deposits * 10000) / totalDeposits;
```

Acknowledge `getRewards` can fail if a token rebases too high, or handle gracefully if `_totalDeposits()` returns more than the `int256` max size.

#### Mitigation

Have `getComposition` check for 0 divisor.

Acknowledge that `getRewards` can fail if a token rebases too high.

#### c0877e0 Resolution

`getComposition` now has the following line:

```
if (totalDeposits == 0) return composition;
```

`getRewards` was acknowledged.

### [L-7] Empty function body - consider commenting why

#### Description

*Instances (1)*:

```
File: ./contracts/liquidSDIndex/base/LiquidSDAdapter.sol

67:     function _authorizeUpgrade(address) internal override onlyOwner
    {}
```

#### Mitigation

Consider commenting why.

*This finding was found by 4naly3er.*

#### c0877e0 Resolution

Acknowledged.

### [L-8] Initializers could be front-run

#### Description

Initializers could be front-run, allowing an attacker to either set their own values, take ownership of the contract, and in the best case forcing a re-deployment

*Instances (15)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

46:     function initialize(

```

```
53:     ) public initializer {

55:         __StakingRewardsPool_init(address(0), _derivativeTokenName,
     _derivativeTokenSymbol);
```

```
File: ./contracts/liquidSDIndex/adapters/LidoSTETHAdapter.sol

16:     function initialize(address _token, address _indexPool) public
    initializer {

16:     function initialize(address _token, address _indexPool) public
    initializer {

17:         __LiquidSDAdapter_init(_token, _indexPool);
```

```
File: ./contracts/liquidSDIndex/adapters/RocketPoolRETHAdapter.sol

17:     function initialize(address _token, address _indexPool) public
    initializer {

17:     function initialize(address _token, address _indexPool) public
    initializer {

18:         __LiquidSDAdapter_init(_token, _indexPool);
```

```
File: ./contracts/liquidSDIndex/base/LiquidSDAdapter.sol

19:     function __LiquidSDAdapter_init(address _token, address
    _indexPool) public onlyInitializing {

23:         __Ownable_init();

24:         __UUPSUpgradeable_init();
```

```
File: ./contracts/liquidSDIndex/test/LiquidSDAdapterMock.sol

13:     function initialize(

17:     ) public initializer {

18:         __LiquidSDAdapter_init(_token, _indexPool);
```

**Mitigation**

Options: - Acknowledge this is intended. - Use the constructor to initialize non-proxied contracts. - For initializing proxy contracts deploy contracts using a factory contract that immediately calls initialize after deployment or make sure to call it immediately after deployment and verify the transaction

succeeded.

*This finding was found by 4naly3er.*

#### c0877e0 Resolution

Acknowledged.

### [L-9] Protect against changing storage layout

#### Description

Empty storage gaps are used to reserve space for future versions of a contract to add new variables without shifting down storage in the inheritance chain.

```
1 uint256[10] private __gap; //upgradeability storage gap
```

#### Mitigation

This storage gap declaration should be moved to the end of the contract to avoid accidentally declaring a state variable within the contract body and messing up the storage layout.

See OpenZeppelin upgradeable contracts as an example.

#### c0877e0 Resolution

Storage gap correctly moved to end of the contract.

### [L-10] Revert on zero deposit

#### Description

When a user deposits a given `lsdToken`, the underlying amount is calculated by calling `LiquidSDAdapter::getUnderlyingByLSD` on the corresponsing adapter.

#### Mitigation

If this value returned is zero, `LiquidSDIndexPool::deposit` should revert to mitigate the case where no shares are minted for a non-zero transfer amount.

**c0877e0 Resolution**

Added a require step to ensure the deposit is not too small.

## Informational / Non-Critical Findings

Note: Informational / Non-Critical Findings findings are not exhaustive, and not attempted to be exhaustive. These are included in the report because they were found by auditors, and we wanted to share them with you.

### [I-1] Use predefined constants instead of arbitrary numbers for code readbility

Example:

```
composition[i] = (deposits * 10000) / depositsTotal;
```

**Mitigation:**

Example:

```
uint256 private constant PRECISION = 10000;
.
.
.
composition[i] = (deposits * PRECISION) / depositsTotal;
```

Another suitable alternative could be `COMPOSITION_TARGET_TOTAL`.

**c0877e0 Resolution**

Included a constant called BASIS_POINTS_TOTAL

### [I-2] `totalDeposits` is used as an overloaded term, consider renaming variables

As a larger overhaul, to be more specific, it is possible to generalize the types of tokens involved in the protocol.

- index token (ie: iETH)
- collateral token (ie: stETH)

- reference token (ie: ETH)

With this in mind, some potential suggestions are: - `LiquidSDIndexPool.sol`: `uint256 private totalDeposits;` could be renamed to `totalStaked`, `totalUnderlyingDeposits`, `totalReferenceTokens`, `totalAccountedReferenceDeposits`.
- This is the value in the reference token (ie: ETH for stETH & rETH) that the procotol has accounted for. This does not include rewards/interest from the collateral. - `LiquidSDIndexPool.sol`: `function _totalDeposits()` seems accurate, since it's the total number of deposits of the LSD tokens. To be extra verbose, consider `function _totalUnderlyingDepositsIncludingRewards()`. - `LiquidSDAdapater.sol`: `function getTotalDeposits()` could be renamed to `getReferenceTokenDeposits()` or `getReferenceTokenValueOfDeposits()`.

#### Additional renaming suggesgtions

- amount -> could be renamed to `withdrawalAmount`.

#### c0877e0 Resolution

Acknowledged.

### [I-3] Fuzz testing (and invariant testing)

Fuzz testing is a testing mechanism used to input random or semi-random data into a system. We recommend adding fuzz testing to the protocol to find edge cases that should hold. Since the repo uses Hardhat, you could add Echidna as your fuzz tester of choice. You can find a minimal fuzz test example here.

For both Echidna and Foundry, there is a subcategory of fuzz tests often referred to as invariant tests, or stateful fuzz tests, which we would also recommend. It is quite possible that adding invariant tests will aid in finding additional vulnerabilities.

We've started a repo for you here. To get started, please read the `CYFRIN_README.md` file.

#### c0877e0 Resolution

Acknowledged.

### [I-4] Use internal function for code reuse

`LiquidSDAdapter::_totalDeposits` is used by `LiquidSDAdapter::getRewards` to calculate the total rewards amount; however, `LiquidSDAdapter::updateRewards` inlines this logic and should make use of the internal helper function instead.

### [I-5] LiquidSDIndexPool totalSupply doesn't follow the ERC20 standard

`totalShares` is the actual total supply of the `iETH` token.

`totalSupply` represents the total underlying LSDs, whereas the ERC20 standard states that `totalSupply` should represent the total number of tokens in existence.

#### c0877e0 Resolution

Acknowledged.

### [I-5] Functions not used internally could be marked external

*Instances (11)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

46:     function initialize(
```

```
File: ./contracts/liquidSDIndex/adapters/LidoSTETHAdapter.sol

16:     function initialize(address _token, address _indexPool) public
    initializer {

24:     function getExchangeRate() public view override returns (
    uint256) {
```

```
File: ./contracts/liquidSDIndex/adapters/RocketPoolRETHAdapter.sol

17:     function initialize(address _token, address _indexPool) public
    initializer {

25:     function getExchangeRate() public view override returns (
    uint256) {
```

```
File: ./contracts/liquidSDIndex/base/LiquidSDAdapter.sol

```

```
19:     function __LiquidSDAdapter_init(address _token, address
    _indexPool) public onlyInitializing {

31:     function getTotalDeposits() public view returns (uint256) {

39:     function getTotalDepositsLSD() public view returns (uint256) {

57:     function getLSDByUnderlying(uint256 _underlyingAmount) public
    view returns (uint256) {
```

*This finding was found by 4naly3er.*

#### c0877e0 Resolution

Acknowledged.

### [I-7] Return values of `approve()` not checked

Not all IERC20 implementations `revert()` when there's a failure in `approve()`. The function signature has a boolean return value and they indicate errors that way instead. By not checking the return value, operations that should have marked as failed may potentially go through without actually approving anything.

*Instances (1)*:

```
File: ./contracts/liquidSDIndex/base/LiquidSDAdapter.sol

21:         token.approve(_indexPool, type(uint256).max);
```

*This finding was found by 4naly3er.*

#### c0877e0 Resolution

Acknowledged.

### [I-8] Missing checks for `address(0)` when assigning values to address state variables

*Instances (1)*:

```
File: ./contracts/liquidSDIndex/base/LiquidSDAdapter.sol

22:         indexPool = _indexPool;
```

*This finding was found by 4naly3er.*

#### c0877e0 Resolution

Acknowledged.

### [I-9] Mitigate fee rounding errors in `updateRewards`

#### Description

When `updateRewards` is called, the function updates and distributes rewards based on the balance deltas of adapters. This includes the distribution of fees which can be improved to avoid issues in rounding fee amounts.

Currently:

```
if (totalRewards > 0) {
  uint256[] memory feeAmounts = new uint256[](fees.length);

  for (uint256 i = 0; i < fees.length; i++) {
      feeAmounts[i] = (uint256(totalRewards) * fees[i].basisPoints) / 10000;
      totalFeeAmounts += feeAmounts[i];
  }

  if (totalFeeAmounts > 0) {
      uint256 sharesToMint = (totalFeeAmounts * totalShares) / (totalDeposits - totalFeeAmounts);
      _mintShares(address(this), sharesToMint);

      for (uint256 i = 0; i < fees.length; i++) {
          if (i == fees.length - 1) {
              transferAndCallFrom(address(this), fees[i].receiver, balanceOf(address(this)), "0x00");
          } else {
              transferAndCallFrom(address(this), fees[i].receiver, feeAmounts[i], "0x00");
          }
      }
  }
}
```

#### Mitigation

Prevent discrepancy in fee distribution due to rounding errors with something like the following, please note this exact code isn't correct and has not been tested.

Recommended:

```
if (totalRewards > 0) {
  uint256 totalFeeBasisPoints = _totalFeeBasisPoints();
  uint256 totalFeeAmounts = totalFeeBasisPoints * totalRewards;
  uint256 feeAmount;
  uint256 feeWeight;

  if (totalFeeAmounts > 0) {
      uint256 sharesToMint = (totalFeeAmounts * totalShares) / (
         totalDeposits - totalFeeAmounts);
      _mintShares(address(this), sharesToMint);

      for (uint256 i = 0; i < fees.length; i++) {
        feeWeight += fees[i].basisPoints;
        uint256 currentFeeAmount = (totalFeeAmounts * feeWeight) / (
           totalFeeBasisPoints - feeAmount);
        feeAmount += currentFeeAmount;
        transferAndCallFrom(address(this), fees[i].receiver,
           currentFeeAmount, "0x00");
      }
  }
}
```

#### c0877e0 Resolution

Acknowledged.

## Gas Findings

Gas findings are not exhaustive, and not attempted to be exhaustive. These are included in the report because they were found by auditors, and we wanted to share them with you.

### [G-1] Initializing the `LiquidSDIndexPool` contract doesn't need a staking rewards pool token

#### Description

```
__StakingRewardsPool_init(address(0), _derivativeTokenName,
   _derivativeTokenSymbol);
```

This contract could just as easily use an inherited contract that doesn't require a token address instead of just defaulting to the zero address.

#### Mitigation

Create a base class without the token parameter.

#### c0877e0 Resolution

Acknowledged.

## Automated Gas Findings

The following were found by 4naly3er.

#### c0877e0 Resolution

These have been acknowledged by the Linkpool team.

#### [G-2] Use assembly to check for `address(0)`

*Saves 6 gas per instance*

*Instances (2)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

69:         require(address(lsdAdapters[_lsdToken]) != address(0), "
    Token is not supported");

314:         require(address(lsdAdapters[_lsdToken]) == address(0), "
    Token is already supported");
```

#### [G-3] Using bools for storage incurs overhead

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from ‘false’ to ‘true’, after having been ‘true’ in the past. See source.

*Instances (1)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

37:     bool public isPaused;
```

### [G-4] Cache array length outside of loop

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

*Instances (18)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

60:         for (uint256 i = 0; i < _fees.length; i++) {

100:         for (uint256 i = 0; i < lsdTokens.length; i++) {

115:         for (uint256 i = 0; i < composition.length; i++) {

134:         for (uint256 i = 0; i < lsdTokens.length; i++) {

204:         for (uint256 i = 0; i < targetDepositDiffs.length; i++) {

215:         for (uint256 i = 0; i < targetDepositDiffs.length; i++) {

240:         for (uint256 i = 0; i < withdrawalAmounts.length; i++) {

259:             for (uint256 i = 0; i < fees.length; i++) {

273:         for (uint256 i = 0; i < lsdTokens.length; i++) {

287:             for (uint256 i = 0; i < fees.length; i++) {

296:                 for (uint256 i = 0; i < fees.length; i++) {

321:         for (uint256 i = 0; i < _compositionTargets.length; i++) {

342:         for (uint256 i = 0; i < lsdTokens.length; i++) {

349:         for (uint256 i = index; i < lsdTokens.length - 1; i++) {

357:         for (uint256 i = 0; i < _compositionTargets.length; i++) {

373:         for (uint256 i = 0; i < _compositionTargets.length; i++) {

468:         for (uint256 i = 0; i < lsdTokens.length; i++) {

481:         for (uint i = 0; i < fees.length; i++) {
```

### [G-5] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replaces each Gwarmaccess (100 gas) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

*Saves 100 gas per instance*

*Instances (1)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

280:             totalDeposits = uint256(int256(totalDeposits) +
   totalRewards);
```

### [G-6] Use calldata instead of memory for function arguments that do not get mutated

Mark data types as `calldata` instead of `memory` where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies `memory` storage.

*Instances (4)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

47:         string memory _derivativeTokenName,

48:         string memory _derivativeTokenSymbol,

51:         Fee[] memory _fees,

369:     function setCompositionTargets(uint256[] memory
   _compositionTargets) external onlyOwner {
```

### [G-7] Use Custom Errors

Source Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

*Instances (19)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

63:         require(_totalFeesBasisPoints() <= 5000, "Total fees must be <= 50%");

64:         require(_withdrawalFee <= 500, "Withdrawal fee must be <= 5%");

69:         require(address(lsdAdapters[_lsdToken]) != address(0), "Token is not supported");

74:         require(!isPaused, "Contract is paused");

179:         require(getDepositRoom(_lsdToken) >= _amount, "Insufficient deposit room for the selected lsd");

314:         require(address(lsdAdapters[_lsdToken]) == address(0), "Token is already supported");

315:         require(_compositionTargets.length == lsdTokens.length + 1, "Invalid composition targets length");

326:         require(totalComposition == 10000, "Composition targets must sum to 100%");

338:         require(_compositionTargets.length == lsdTokens.length - 1, "Invalid composition targets length");

339:         require(lsdAdapters[_lsdToken].getTotalDeposits() < 1 ether, "Cannot remove adapter that contains deposits");

362:         require(totalComposition == 10000, "Composition targets must sum to 100%");

370:         require(_compositionTargets.length == lsdTokens.length, "Invalid composition targets length");

378:         require(totalComposition == 10000, "Composition targets must sum to 100%");

389:         require(_compositionTolerance < 10000, "Composition tolerance must be < 100%");

410:         require(_withdrawalFee <= 500, "Withdrawal fee must be <= 5%");

421:         require(_totalFeesBasisPoints() <= 5000, "Total fees must be <= 50%");

431:         require(_index < fees.length, "Fee does not exist");
```

```
36
37 441:         require(_totalFeesBasisPoints() <= 5000, "Total fees must
       be <= 50%");
38
39 449:         require(_isPaused != isPaused, "This pause status is
       already set");
```

### [G-8] Don't initialize variables with default value

*Instances (17)*:

```
1  File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol
2
3  60:         for (uint256 i = 0; i < _fees.length; i++) {
4
5  100:         for (uint256 i = 0; i < lsdTokens.length; i++) {
6
7  115:         for (uint256 i = 0; i < composition.length; i++) {
8
9  134:         for (uint256 i = 0; i < lsdTokens.length; i++) {
10
11 204:         for (uint256 i = 0; i < targetDepositDiffs.length; i++) {
12
13 215:         for (uint256 i = 0; i < targetDepositDiffs.length; i++) {
14
15 240:         for (uint256 i = 0; i < withdrawalAmounts.length; i++) {
16
17 259:             for (uint256 i = 0; i < fees.length; i++) {
18
19 273:         for (uint256 i = 0; i < lsdTokens.length; i++) {
20
21 287:             for (uint256 i = 0; i < fees.length; i++) {
22
23 296:                 for (uint256 i = 0; i < fees.length; i++) {
24
25 321:         for (uint256 i = 0; i < _compositionTargets.length; i++) {
26
27 342:         for (uint256 i = 0; i < lsdTokens.length; i++) {
28
29 357:         for (uint256 i = 0; i < _compositionTargets.length; i++) {
30
31 373:         for (uint256 i = 0; i < _compositionTargets.length; i++) {
32
33 468:         for (uint256 i = 0; i < lsdTokens.length; i++) {
34
35 481:         for (uint i = 0; i < fees.length; i++) {
```

### [G-9] Long revert strings

Revert strings are stored in the contract bytecode and loaded into memory before being returned. As such, longer revert strings result in increased deployment and runtime costs.

*Instances (9)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

179:         require(getDepositRoom(_lsdToken) >= _amount, "Insufficient deposit room for the selected lsd");

315:         require(_compositionTargets.length == lsdTokens.length + 1, "Invalid composition targets length");

326:         require(totalComposition == 10000, "Composition targets must sum to 100%");

338:         require(_compositionTargets.length == lsdTokens.length - 1, "Invalid composition targets length");

339:         require(lsdAdapters[_lsdToken].getTotalDeposits() < 1 ether, "Cannot remove adapter that contains deposits");

362:         require(totalComposition == 10000, "Composition targets must sum to 100%");

370:         require(_compositionTargets.length == lsdTokens.length, "Invalid composition targets length");

378:         require(totalComposition == 10000, "Composition targets must sum to 100%");

389:         require(_compositionTolerance < 10000, "Composition tolerance must be < 100%");
```

### [G-10] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided.

*Instances (10)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

313:     function addLSDToken(address _lsdToken, address _lsdAdapter, uint256[] calldata _compositionTargets) external onlyOwner {
```

```
369:     function setCompositionTargets(uint256[] memory
   _compositionTargets) external onlyOwner {

388:     function setCompositionTolerance(uint256 _compositionTolerance
   ) external onlyOwner {

401:     function setCompositionEnforcementThreshold(uint256
   _compositionEnforcementThreshold) external onlyOwner {

409:     function setWithdrawalFee(uint256 _withdrawalFee) external
   onlyOwner {

419:     function addFee(address _receiver, uint256 _feeBasisPoints)
   external onlyOwner {

430:     function updateFee(uint256 _index, address _receiver, uint256
   _feeBasisPoints) external onlyOwner {

448:     function setPaused(bool _isPaused) external onlyOwner {
```

```
File: ./contracts/liquidSDIndex/base/LiquidSDAdapter.sol

19:     function __LiquidSDAdapter_init(address _token, address
   _indexPool) public onlyInitializing {

67:     function _authorizeUpgrade(address) internal override onlyOwner
    {}
```

#### [G-11] `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

*Saves 5 gas per loop*

*Instances (18)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

60:         for (uint256 i = 0; i < _fees.length; i++) {

100:         for (uint256 i = 0; i < lsdTokens.length; i++) {

115:         for (uint256 i = 0; i < composition.length; i++) {

134:         for (uint256 i = 0; i < lsdTokens.length; i++) {

204:         for (uint256 i = 0; i < targetDepositDiffs.length; i++) {

215:         for (uint256 i = 0; i < targetDepositDiffs.length; i++) {

```

```
240:         for (uint256 i = 0; i < withdrawalAmounts.length; i++) {

259:             for (uint256 i = 0; i < fees.length; i++) {

273:         for (uint256 i = 0; i < lsdTokens.length; i++) {

287:             for (uint256 i = 0; i < fees.length; i++) {

296:                 for (uint256 i = 0; i < fees.length; i++) {

321:         for (uint256 i = 0; i < _compositionTargets.length; i++) {

342:         for (uint256 i = 0; i < lsdTokens.length; i++) {

349:         for (uint256 i = index; i < lsdTokens.length - 1; i++) {

357:         for (uint256 i = 0; i < _compositionTargets.length; i++) {

373:         for (uint256 i = 0; i < _compositionTargets.length; i++) {

468:         for (uint256 i = 0; i < lsdTokens.length; i++) {

481:         for (uint i = 0; i < fees.length; i++) {
```

**[G-12] Use != 0 instead of > 0 for unsigned integer comparison**

*Instances (4)*:

```
File: ./contracts/liquidSDIndex/LiquidSDIndexPool.sol

242:             if (amount > 0) {

292:             if (totalFeeAmounts > 0) {
```