



LinkPool LiquidSDIndexPool Audit Report

Version 2.0

Cyfrin.io

March 22, 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

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 withdrawals.

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 February 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 trivial amount of assets are), state handling might be off, functions are incorrect as to natspec, issues with comments, etc.
- Informational / Non-Critical: 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.

Tools used

- Slither
- 4naly3er
- foundry
- Hardhat
- Solodit

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 ActivePool._rebalance() does not take into account the case when the vault's strategy gets loss	H	Open
H-1 Users would lose some shares during withdrawal in <code>ReaperVaultV2._withdraw()</code> .	H	Open
M-1 "Dust" collaterals/shares are not cleared in ActivePool._rebalance()	M	Open

[H-1] ActivePool._rebalance() does not take into account the case when the vault's strategy gets loss

ActivePool._rebalance() does not consider the case when the vault's strategy gets loss

<https://github.com/code-423n4/2023-02-ethos/blob/73687f32b934c9d697b97745356cdf8a1f264955/Ethos-Core/contracts/ActivePool.sol#L251> <https://github.com/code-423n4/2023-02-ethos/blob/73687f32b934c9d697b97745356cdf8a1f264955/Ethos-Core/contracts/ActivePool.sol#L282> <https://github.com/code-423n4/2023-02-ethos/blob/73687f32b934c9d697b97745356cdf8a1f264955/Ethos-Core/contracts/ActivePool.sol#L288>

Impact

The `_rebalance()` reverts if a strategy gets loss. Because `_rebalance()` is called on all important workflows, this leads to insolvency of the protocol.

Proof of Concept

The protocol uses `ReaperVaultERC4626` to manage the collateral assets and farm profit. The vaults are connected to whitelisted strategies.

But it is not guaranteed that the strategies earn profit all the time.

On the other hand, in several places of `ActivePool._rebalance()`, the protocol assumes that it can get more than deposits all the time.

At L251, where the protocol calculates the profit by subtracting the stored `yieldingAmount` from the `sharesToAssets`, this will revert if the strategy got loss.

```
1 ActivePool.sol
2 242:          // how much has been allocated as per our internal records
   ?
3 243:          vars.currentAllocated = yieldingAmount[_collateral];//
   @audit-info current yield deposit
4 244:
5 245:          // what is the present value of our shares?
6 246:          vars.yieldGenerator = IERC4626(yieldGenerator[_collateral
   ]);
7 247:          vars.ownedShares = vars.yieldGenerator.balanceOf(address(
   this));
8 248:          vars.sharesToAssets = vars.yieldGenerator.convertToAssets(
   vars.ownedShares);
9 249:
10 250:          // if we have profit that's more than the threshold,
   record it for withdrawal and redistribution
11 251:          vars.profit = vars.sharesToAssets.sub(vars.
   currentAllocated);//@audit-issue profit from the farming, this can
   revert!
12 252:          if (vars.profit < yieldClaimThreshold[_collateral]) {
13 253:              vars.profit = 0;
14 254:          }
```

At L282 where the protocol withdraws specifying the collateral amount to receive but it will revert if the strategy lost.

```
1 ActivePool.sol
2 276:          // + means deposit, - means withdraw
3 277:          vars.netAssetMovement = int256(vars.toDeposit) - int256(
   vars.toWithdraw) - int256(vars.profit);
```

```
4 278:         if (vars.netAssetMovement > 0) {
5 279:             IERC20(_collateral).safeIncreaseAllowance(
yieldGenerator[_collateral], uint256(vars.netAssetMovement));
6 280:             IERC4626(yieldGenerator[_collateral]).deposit(uint256(
vars.netAssetMovement), address(this));
7 281:         } else if (vars.netAssetMovement < 0) {
8 282:             IERC4626(yieldGenerator[_collateral]).withdraw(uint256
(-vars.netAssetMovement), address(this), address(this)); // @audit-
info coll received, this can revert
9 283:         }
```

Because `_rebalance()` is called on all important workflows, this leads to insolvency of the protocol.

Tools Used

Manual Review

Recommended Mitigation Steps

Do not assume `sharesToAssets > yieldingAmount` at all places mentioned and handle appropriately.

[H-1] Users would lose some shares during withdrawal in `ReaperVaultV2._withdraw()`.

Users would lose some shares during withdrawal in `ReaperVaultV2._withdraw()`.

<https://github.com/code-423n4/2023-02-ethos/blob/73687f32b934c9d697b97745356cdf8a1f264955/Ethos-Vault/contracts/ReaperVaultV2.sol#L401>

Impact

`ReaperVaultV2._withdraw()` burns 100% of shares even if the vault balance is less than the required underlying amount.

As a result, users would lose some shares during withdrawal.

Proof of Concept

Users can receive underlying tokens by burning their shares using `_withdraw()`.

If the vault doesn't have enough underlying balance, it withdraws from strategies inside `withdrawalQueue`.

```
1 File: ReaperVaultV2.sol
2 359:     function _withdraw(
3 360:         uint256 _shares,
4 361:         address _receiver,
5 362:         address _owner
6 363:     ) internal nonReentrant returns (uint256 value) {
7 364:         require(_shares != 0, "Invalid amount");
8 365:         value = (_freeFunds() * _shares) / totalSupply();
9 366:         _burn(_owner, _shares);
10 367:
11 368:         if (value > token.balanceOf(address(this))) {
12 398:             ....
13 399:             vaultBalance = token.balanceOf(address(this));
14 400:             if (value > vaultBalance) {
15 401:                 value = vaultBalance; //@audit should reduce
            shares accordingly
16 402:             }
17 403:
18 404:             require(
19 405:                 totalLoss <= ((value + totalLoss) *
            withdrawMaxLoss) / PERCENT_DIVISOR,
20 406:                 "Withdraw loss exceeds slippage"
21 407:             );
22 408:         }
23 409:
24 410:         token.safeTransfer(_receiver, value);
25 411:         emit Withdraw(msg.sender, _receiver, _owner, value,
            _shares);
26 412:     }
```

After withdrawing from the strategies of `withdrawalQueue`, it applies the max cap at L401.

But as we can see from `setWithdrawalQueue()`, `withdrawalQueue` wouldn't contain all of the active strategies and the above condition at L400 will be true.

In this case, users will get fewer underlying amounts after burning the whole shares that they requested.

As a reference, it recalculates the shares for the above case in Yearn vault.

```
1     if value > vault_balance:
2         value = vault_balance
3         # NOTE: Burn # of shares that corresponds to what Vault has on-
            hand,
4         # including the losses that were incurred above during
            withdrawals
5         shares = self._sharesForAmount(value + totalLoss)
```

Tools Used

Manual Review

Recommended Mitigation Steps

We should recalculate the shares and burn them rather than burn all shares.

[M-1] “Dust” collaterals/shares are not cleared in `ActivePool._rebalance()`

Dust collaterals/shares are not cleared in `ActivePool._rebalance()`

<https://github.com/code-423n4/2023-02-ethos/blob/73687f32b934c9d697b97745356cdf8a1f264955/Ethos-Core/contracts/ActivePool.sol#L252> <https://github.com/code-423n4/2023-02-ethos/blob/73687f32b934c9d697b97745356cdf8a1f264955/Ethos-Core/contracts/ActivePool.sol#L267>

Impact

The unclaimed “dust” profit will be locked in the vault. Because the affected amount will be not substantial and it will occur only for edge cases, evaluate the severity to Med.

Proof of Concept

The protocol uses `yieldClaimThreshold` to prevent unnecessary transfer of dust collateral profit (maybe to save gas?).

```
1 ActivePool.sol
2 252:         if (vars.profit < yieldClaimThreshold[_collateral]) {
3 253:             vars.profit = 0; //@audit-issue check how the dust
           remaining in the vault are processed in the end
4 254:         }
```

And if `_amountLeavingPool==collAmount[_collateral]`, i.e. for the “last” withdrawal from the vault, the profit under the threshold is not claimed while the protocol considers it does not have any collaterals left in the vault.

As a result, the unclaimed “dust” profit will be locked in the vault. Because the affected amount will be not substantial and it will occur only for edge cases, evaluate the severity to Med.

Tools Used

Manual Review

Recommended Mitigation Steps

At L266, check if `vars.finalBalance==0` and add the profit to the target withdraw amount (or redeem the whole owned shares). The redeemed profit will be distributed by the following logic.