LinkPool LiquidSDIndexPool Audit Report

Version 2.0

Cyfrin.io

March 28, 2023

Cyfrin LiquidSDIndexPool Mitigation Audit Report

Cyfrin.io

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 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.

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-2 Users would lose some shares during withdrawal in `ReaperVaultV2._withdraw()`.	H	Open
M-1 "Dust" collaterals/shares are not cleared in ActivePool._rebalance()	M	Open
NC-1 Non-standard storage packing	NC	Open
NC-2 EIP-1967 second pre-image best practice	NC	Open
NC-3 Remove experimental ABIEncoderV2 pragma	NC	Open
NC-4 Inconsistent use of decimal/hex notation in inline assembly	NC	Open
NC-5 Unused imports and errors	NC	Open
NC-6 Inconsistency in LibMath comments	NC	Open
NC-7 FIXME and TODO comments	NC	Open

Finding	Severity	Status
NC-8 Use correct NatSpec tags	NC	Open
NC-9 Poorly descriptive variable & function names in `GeoEmaAndCumSmaPump` are difficult to read	NC	Open
NC-10 Remove TODO Check if bytes shift is necessary	NC	Open
NC-11 Use _ prefix for internal functions	NC	Open
NC-12 Missing test coverage for a number of functions	NC	Open
NC-13 Use `uint256` over `uint`	NC	Open
NC-14 Use constant variables in place of inline magic numbers	NC	Open
NC-15 Insufficient use of NatSpec and comments on complex code blocks	NC	Open
NC-16 Precision loss on large values transformed between log2 scale and the normal scale	NC	Open
NC-17 Emit events prior to external interactions	NC	Open
G-1 Simplify modulo operations	G	Open
G-2 Branchless optimization	G	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/73687f32b934c9d697b977453
Core/contracts/ActivePool.sol#L282 https://github.com/code-423n4/2023-02-ethos/blob/73687f32b934c9d697b977453
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.

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

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

```
ActivePool.sol
276:         // + means deposit, - means withdraw
277:         vars.netAssetMovement = int256(vars.toDeposit) - int256(
   vars.toWithdraw) - int256(vars.profit);
278:         if (vars.netAssetMovement > 0) {
279:             IERC20(_collateral).safeIncreaseAllowance(
   yieldGenerator[_collateral], uint256(vars.netAssetMovement));
280:             IERC4626(yieldGenerator[_collateral]).deposit(uint256(
   vars.netAssetMovement), address(this));
281:         } else if (vars.netAssetMovement < 0) {
```

```
282:             IERC4626(yieldGenerator[_collateral]).withdraw(uint256
    (-vars.netAssetMovement), address(this), address(this));//@audit-
    info coll received, this can revert
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-2] 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`.

```
File: ReaperVaultV2.sol
359:     function _withdraw(
360:         uint256 _shares,
361:         address _receiver,
362:         address _owner
363:     ) internal nonReentrant returns (uint256 value) {
364:         require(_shares != 0, "Invalid amount");
365:         value = (_freeFunds() * _shares) / totalSupply();
366:         _burn(_owner, _shares);
367:
368:         if (value > token.balanceOf(address(this))) {
398:             ....
399:             vaultBalance = token.balanceOf(address(this));
400:             if (value > vaultBalance) {
401:                 value = vaultBalance; //@audit should reduce shares accordingly
402:             }
403:
404:             require(
405:                 totalLoss <= ((value + totalLoss) * withdrawMaxLoss) / PERCENT_DIVISOR,
406:                 "Withdraw loss exceeds slippage"
407:             );
408:         }
409:
410:         token.safeTransfer(_receiver, value);
411:         emit Withdraw(msg.sender, _receiver, _owner, value, _shares);
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.

```
    if value > vault_balance:
        value = vault_balance
        # NOTE: Burn # of shares that corresponds to what Vault has on-hand,
        #       including the losses that were incurred above during withdrawals
        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/73687f32b934c9d697b977453
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?).

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

[NC-1] Non-standard storage packing

Per the Solidity docs, the first item in a packed storage slot is stored lower-order aligned; however, manual packing in `LibBytes` does not follow this convention. Modify the `storeUint128` function to store the first packed value at the lower-order aligned position.

[NC-2] EIP-1967 second pre-image best practice

When calculating custom EIP-1967 storage slots, as in Well.sol::RESERVES_STORAGE_SLOT, it is best practice to add an offset of −1 to the hashed value to further reduce the possibility of a second pre-image attack.

[NC-3] Remove experimental ABIEncoderV2 pragma

ABIEncoderV2 is enabled by default in Solidity 0.8, so two instances can be removed.

[NC-4] Inconsistent use of decimal/hex notation in inline assembly

For readability and to prevent errors when working with inline assembly, decimal notation should be used for integer constants and hex notation for memory offsets.

[NC-5] Unused imports and errors

In `LibMath`: - OpenZeppelin SafeMath is imported but not used - `PRBMath_MulDiv_Overflow` error is declared but never used

[NC-6] Inconsistency in LibMath comments

There is inconsistent use of `x` in comments and `a` in code within the `nthRoot` and `sqrt` functions of `LibMath`.

[NC-7] FIXME and TODO comments

There are several FIXME and TODO comments that should be addressed.

[NC-8] Use correct NatSpec tags

Uses of `@dev See {IWell.fn}` should be replaced with `@inheritdoc IWell` to inherit the NatSpec documentation from the interface.

[NC-9] Poorly descriptive variable & function names in GeoEmaAndCumSmaPump are difficult to read

For example, in `update`: - `b` could be renamed `returnedReserves`. - `aN` could be renamed `alphaN` or `alphaRaisedToTheDeltaTimeStamp`.

Additionally, `A`/`_A` could be renamed `ALPHA` and `readN` could be renamed `readNumberOfReserves`.

[NC-10] Remove TODO Check if bytes shift is necessary

In `LibBytes16::readBytes16`, the following line has a `TODO`:

```
mstore(add(reserves, 64), shl(128, sload(slot)))//TODO: Check if byte
 shift is necessary
```

Since two reserve elements' worth of data is stored in a single slot, the left shift is indeed needed. The following test shows how these are different:

```
1  function testNeedLeftShift() public {
2      uint256 reservesSize = 2;
3      uint256 slotNumber = 12345;
4      bytes32 slot = bytes32(slotNumber);
5
6      bytes16[] memory leftShiftreserves = new bytes16[](reservesSize);
7      bytes16[] memory noShiftreserves = new bytes16[](reservesSize);
8
```

```
 9      // store some data in the slot
10      assembly {
11          sstore(
12              slot,
13              0
                    x00000000000000000000000000007b00000000000000000000000000000011

14          )
15      }
16
17      // left shift
18      assembly {
19          mstore(add(leftShiftreserves, 32), sload(slot))
20          mstore(add(leftShiftreserves, 64), shl(128, sload(slot)))
21      }
22
23      // no shift
24      assembly {
25          mstore(add(noShiftreserves, 32), sload(slot))
26          mstore(add(noShiftreserves, 64), sload(slot))
27      }
28      assert(noShiftreserves[1] != leftShiftreserves[1]);
29  }
```

[NC-11] Use _ prefix for internal functions

For functions such as `getSlotForAddress`, it is more readable to have this function be named `_getSlotForAddress` so readers know it is an internal function. A similarly opinionated recommendation is to use `s_` for storage variables and `i_` for immutable variables.

[NC-12] Missing test coverage for a number of functions

Consider adding tests for `GeoEmaAndCumSmaPump::getSlotsOffset`, `GeoEmaAndCumSmaPump::getDeltaTimestamp` and `_getImmutableArgsOffset` to increase test coverage and confidence that they are working as expected.

[NC-13] Use `uint256` over `uint`

`uint` is an alias for `uint256` and is not recommended for use. The variable size is not immediately clear and this can also cause issues when encoding data with selectors if the alias is mistakenly used within the signature string.

[NC-14] Use constant variables in place of inline magic numbers

When using a number in the procotol, it should be made clear what the number represents by storing it as a constant variable.

For example, in `Well.sol`, the calldata location of the pumps is given by the following:

```
uint dataLoc = LOC_VARIABLE + numberOfTokens() * 32 +
    wellFunctionDataLength();
```

Without additional knowledge, it may be difficult to read and so it is recommend to assign variables such as:

```
uint256 constant ONE_WORD = 32;
uint256 constant PACKED_ADDRESS = 20;
...
uint dataLoc = LOC_VARIABLE + numberOfTokens() * ONE_WORD +
    wellFunctionDataLength();
```

The same recommendation can be applied to inline assembly blocks which perform shifts such that numbers like `248` and `208` have some verbose meaning.

Additionally, when packing values in immutable data/storage, the code would benefit from a note explicitly stating where this is the case, e.g. pumps.

[NC-15] Insufficient use of NatSpec and comments on complex code blocks

Many low-level functions such as `WellDeployer::encodeAndBoreWell` are missing NatSpec documentation. Additionally, many of the math-heavy contracts and libraries can be difficult to understand without NatSpec and supporting comments.

[NC-16] Precision loss on large values transformed between log2 scale and the normal scale

In `GeoEmaAndCumSmaPump.sol::_init`, the reserve values are transformed into log2 scale:

```
byteReserves[i] = reserves[i].fromUIntToLog2();
```

This transformation implies a precision loss, particularly for large `uint256` values as demonstrated by the following test:

```
function testUIntMaxToLog2() public {
uint x = type(uint).max;
bytes16 y = ABDKMathQuad.fromUIntToLog2(x);
```

```
4  console.log(x);
5  console.logBytes16(y);
6  assertEq(ABDKMathQuad.fromUInt(x).log_2(), ABDKMathQuad.fromUIntToLog2(
       x));
7  uint x_recover = ABDKMathQuad.pow_2ToUInt(y);
8  console.log(ABDKMathQuad.toUInt(y));
9  console.log(x_recover);
10 }
```

Consider explicit limiting of the reserve values to avoid precision loss.

[NC-17] Emit events prior to external interactions

To strictly conform to the Checks Effects Interactions pattern, it is recommended to emit events prior to any external interactions. Implementation of this pattern is generally advised to ensure correct migration through state reconstruction, which in this case should not be affected given all instances in `Well.sol` are protected by the `nonReentrant` modifier, but it is still good practice.

[G-1] Simplify modulo operations

In `LibBytes::storeUint128` and `LibBytes::readUint128`, `reserves.lenth % 2 == 1` and `i % 2 == 1` can be simplified to `reserves.length & 1 == 1` and `i & 1 == 1`.

[G-2] Branchless optimization

The `sqrt` function in `MathLib` and related comment should be updated to reflect changes in Solmate's `FixedPointMathLib` which now includes the branchless optimization `z := sub(z, lt(div(x, z), z))`.