Ender Labs Security Review

Auditors

hash

1 Executive Summary

Over the course of 1 days in total, Ender Labs engaged with hash to review Ender Protocol V1 (Update). In this period of time a total of 6 issues were found.

Summary

Project Name	Ender Labs
Code Under Review	173063cabc...
Type of Project	Yield, Restaking
Audit Timeline	8 June 2024 - 9 June 2024

Issues Found

High Risk	2
Medium Risk	1
Low Risk	2
Informational	1
Total Issues	6

2 Findings

2.1 High

2.1.1 Incorrect depositReturn calculation due to improper updation of fundsInfo mapping

`depositReturn` is calculated as `(totalReturn * totalDeposit) / fundsInfo[_stEthAddress]`

```
    function calculateDepositReturn(address _stEthAddress) public returns (uint256 depositReturn) {

        .....

        } else {
            uint256 totalDeposit = IEnderBond(enderBond).totalDeposit();
            if (totalDeposit == 0 || fundsInfo[_stEthAddress] == 0) {
                depositReturn = 0;
            } else {
=>              depositReturn = (totalReturn * totalDeposit) / fundsInfo[_stEthAddress];
            }
        }
    }
```

The fundsInfo mapping is reduced whenever a user withdraws from the bond with the correponding `depositReturn` amount. The idea is that this will still leave the assets associated with `bondFee` in the fundsInfo mapping thereby partitioning the received yield to `treasuryReturn` and `depositReturn`

```
    function withdraw(EndRequest memory param) external onlyBond {
        if (param.account == address(0)) revert ZeroAddress();
        if (fundsInfo[param.stakingToken] < param.tokenAmt) revert InsufficientAmount();
=>      fundsInfo[param.stakingToken] -= param.tokenAmt;
```

But fundsInfo is not updated when the assets are withdrawn from instadapp through the withdrawAvaialblePOL function. This causes incorrect partitioning of the yield (rebase vs treasury) ie. even when the assets generating the yield is solely deposit assets, the depositReturn will still only be a fraction of the totalReturn

Recommendation Updae the fundsInfo mapping on POL withdrawals

Status Acknowledged. The issue of staker's receiving incorrect/lowered rebase reward is considered acceptable for now

2.1.2 Rounding in Instadapp will cause reverts

When assets are deposited or withdrawn, `IInstadappLiteV2(receiptToken).convertToAssets(receiptTokenAmount)` will be returning a rounded down value

```
    function calculateTotalReturn() internal view returns (uint256 totalReturn) {
        address receiptToken = instadapp;
        uint256 receiptTokenAmount = IInstadappLiteV2(receiptToken).balanceOf(address(this));
        if (receiptTokenAmount > 0) {
            totalReturn =
=>              IInstadappLiteV2(receiptToken).convertToAssets(receiptTokenAmount) +
                instaDappWithdrawlValuations -
                instaDappDepositValuations -
                instaDappLastValuation;
        }
```

But the calculations don't factor this and expect the amount to be greater than or equal ie. on a deposit, the final actual value in instadapp `IInstadappLiteV2(receiptToken).convertToAssets(receiptTokenAmount)` must be

greater than or equal to `instaDappDepositValuations + instaDappLastValuation`. Hence it will underflow and cause all functionalities that eventually calls `stakeRebasingReward` to revert. The share value in instadapp will only increase when the exchangePrice is updated till which this revert will continue to occur. Although a negative yield is not expected, the underflow will be amplified in such a case. Similar problem exists here. If the initialDeposit was made without any bondFee, then the totalDepositReturn could be greater than convertToAssets and this will cause a underflow

Recommendation Check for the underflow condition / keep a int variable and return 0 incase value < 0

Status Fixed in PR95

2.2 Medium

2.2.1 Share inflation attack

The kept checks of 1e16 amounts doesn't guard against share inflation attack.

```
    function stake(uint256 amount) external stakingEnabled stakingContractPaused {
=>      if (amount < 1e16) revert InvalidAmount();
        _epochStakingReward(stEth);


        uint256 refractionFeePercentage = IEndToken(endToken).refractionFeePercentage();
        uint256 fee = (amount * refractionFeePercentage) / 10000;
        uint256 sEndAmount = calculateSEndTokens(amount-fee);
        ISEndToken(sEndToken).mint(msg.sender, sEndAmount);
        ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);


        emit Stake(msg.sender, amount);
    }

    function unstake(uint256 amount) external unstakeEnabled stakingContractPaused {
        if (amount == 0) revert InvalidAmount();
        if (ISEndToken(sEndToken).balanceOf(msg.sender) < amount) revert InvalidAmount();


        _epochStakingReward(stEth);


        uint256 reward = claimRebaseValue(amount);
=>      if (reward < 1e16) revert InvalidAmount();


        // transfer token
        ISEndToken(endToken).safeTransfer(msg.sender, reward);
        ISEndToken(sEndToken).burn(msg.sender, amount);


        emit UnStake(msg.sender, amount);
    }
```

Eg: The user can now stake 1e17 tokens, unstake (1e17-1) tokens and then inflate the value of the share to > 1e16. This will cause the future stakers to loose their assets unless they make deposits in multiple of the shareValue

Recommendation Keep a check for the minimum shares minted / donate the initial set of sEnd tokens to a trusted address

Status The protocol will work around this by launching with a non-zero bondRewardPercentage and hence expects the initial set of shares to be distributed across multiple user's

2.3 Low

2.3.1 Deposits/withdrawals occuring after the instadapp `updateExchangePrice` call can revert

Inside `_epochStakingReward`, `rebaseEndAmountPerDay` is calculated as `rebaseEndAmount * SECONDS_IN_DAY / (block.timestamp - latestRebaseUpdateTime)` if the rebaseReward is non-zero

```
    function _epochStakingReward(address _asset) internal {
        if (_asset != stEth) revert NotAllowed();
        latestRebaseUpdateTime = IEnderBond(enderBond).latestRebaseUpdateTime();
        int256 totalReward = IEnderTreasury(enderTreasury).stakeRebasingReward(_asset);
        lastRebaseReward = totalReward;


        if (totalReward > 0) {
            rebaseRefractionReward = (uint256(totalReward) * bondRewardPercentage) / 10000;
            uint256 rebaseEndAmount = uint256(totalReward) - rebaseRefractionReward;
=>          rebaseEndAmountPerDay = rebaseEndAmount * SECONDS_IN_DAY / (block.timestamp -
↪  latestRebaseUpdateTime);
```

The exchange price and hence the instadappValuation will be updated after the `updateExchangePrice` function is invoked by the rebalancer in instadapp. And hence after this call is when the `rebaseReward` will become non-zero too. If there was a `stakeRebasingReward` call in the same block but before the `updateExchangePrice` call, then the `latestRebaseUpdateTime` will be same as `block.timestamp`. This will cause further deposits/withdrawals in the same block to revert due to division by 0

Status Fixed in PR93

2.3.2 getWithdrawAmountForPOL will return incorrectly

The `getWithdrawAmountForPOL` function is supposed to return the amount of assets that can be withdrawn as POL. But since the contract is not in the latest state, ie. `epochStakingReward` is not invoked, the returned value will be inflated

Status Fixed in PR94. The view function was changed to a state changing function which invokes `epochStakingReward` before calculating the POL amount

2.4 Informational

2.4.1 Instadapp withdrawal fees not considered

Instadapp charges a fee on withdrawals. This causes the receiver to receive less than `withdrawAmount` when withdrawing from Instadapp. For POL withdrawals, this causes the totalPOLDepositAmount to not account for this fees

Status Fixed in PR97 for POL withdrawals while user's will continue to receive lowered assets (after the fees)