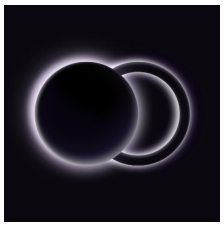
Ender Protocol

Security Review



June 4, 2024

Conducted by:
Blckhv, Independent Security Researcher
Slavcheww, Independent Security Researcher

Contents

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

A liquid staking protocol using LST-bonds to enable concentrated yield tokens and liquid staking power. Through a novel bond-based "yield compression" and yield-splitting mechanism, Ender Protocol enables users to capture liquid staking yield into a token, **END**, backed by the staking rewards from the bond deposits, in order to allow for concentrated yields in liquid staking rewards.

Overview

Project	Ender Protocol
Repository	https://github.com/enderprotocol/ender-v1
Commit Hash	6b138844c62b974b1976765c2cc6eee2a1273d5e
Resolution	75449d9763538d961a4db9b9e3fc70c42c64f6b2
Date	May 23 – May 29, 2024

Scope

EnderBond.sol
EnderStaking.sol
EnderTreasury.sol
ERC20/EndToken.sol
ERC20/EnderStakeEth.sol
ERC20/SEndToken.sol

Issues Found

High Risk	2
Medium Risk	5
Low/Info Risk	9

5. Findings

5.1. High severity

5.1.1. Strategy deposits will fail because of edge case in `stETH`

Severity: High Risk

Context: EnderBond.sol#L362-L367

Description: The current architecture of Ender has 2 subsequent transfers of `stETH` tokens, one from the user to the `EnderTreasury` and then **the same amount** is transferred to the `Instadapp` strategy. This is problematic because of the transfer functionality of `stETH` - https://docs.lido.fi/guides/lido-tokens-integration-guide/#1-2-wei-corner-case. The issue will happen because tokens are converted to shares and most of the time `principal - 1–2 wei` amount will be transferred to the treasury, then the whole `principal` will be deposited into strategy and there will be an insufficient amount of `stETH` in the `EnderTreasury`.

As a result most of the bond deposits will fail, unless `EnderTreasury` has `stETH` balance greater than the principal which will be taken.

Recommendation: In order to avoid reverts of the deposits, record the balance of the transfers before/after and use only the difference.

Resolution: Fixed

5.1.2. Temporary DoS of staking/unstaking functionality

Severity: High Risk

Context: EnderStaking.sol#L184-L187

Description: When there is no reward from the strategy `lastRebaseReward` is set to 0, stake/unstake functions in EnderStrategy will be temporarily blocked because of underflow in the `calculateRebaseIndexPerSecond` function.

We can see that only `lastRebaseReward` is updated in this scenario:

```
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;
...
}
```

Then function will be reverting since `rebaseRefractionReward` is the bondholders rewards from the previous epoch and will remain unchanged, whereas the `lastRebaseReward` will be 0:

```
function calculateRebaseIndexPerSecond() internal {
    uint256 sEndTotalSupply = ISEndToken(sEndToken).totalSupply();
    if (sEndTotalSupply == 0) {
        rebasingIndexPerSecond = 0;
    } else {
        uint256 rebaseEndAmount = uint256(lastRebaseReward) - rebaseRefractionReward;//@audit 0 - rebaseRefractionReward
        if (block.timestamp > latestRebaseUpdateTime) {
            rebasingIndexPerSecond =
                (rebaseEndAmount * 1e18) /
                (sEndTotalSupply * (block.timestamp - latestRebaseUpdateTime));
        } else {
            rebasingIndexPerSecond = (rebasingIndexPerSecond * oldSEndBalance) / sEndTotalSupply;
        }
    }
}
```

The issue will occur until there is no positive yield from the strategy and `totalRewards` become greater than 0.

Recommendation: `calculateRebaseIndexPerSecond` can be safely removed, since it is from the older version of the protocol and is not being used now.

Resolution: Fixed

5.2. Medium severity

5.2.1. `linearInterpolate` is constant for the period of 31-59 maturity days

Severity: Medium Risk

Context: EnderBond.sol#L289

Description: When maturity is between **31-59** days **maturityModifier** will be constant, since *`y1 = 85 and y0 = 85`* and the `linearInterpolate` will always return 85:

```
function getInterest(uint256 maturity) public view returns (uint256 rate) {
    uint256 maturityModifier;
    if (maturity < 15) {
        maturityModifier = linearInterpolate(7, 15, 70, 80, maturity);
    } else if (maturity < 30) {
        maturityModifier = linearInterpolate(15, 30, 80, 85, maturity);
    } else if (maturity < 60) {
        maturityModifier = linearInterpolate(30, 60, 85, 85, maturity);
    } else if (maturity < 90) {
        maturityModifier = linearInterpolate(60, 90, 90, 100, maturity);
    ....
 }
```

```
function linearInterpolate(uint256 x0, uint256 x1, uint256 y0, uint256 y1, uint256 x) internal pure returns (uint256) {
    return y0 + (x - x0) * (y1 - y0) / (x1 - x0);
        // 85 + (59 - 30) * (85 - 85) / (60 - 30) = 85
        // 85 + (31 - 30) * (85 - 85) / (60 - 30) = 85
}
```

Recommendation: Adjust it to be *`y0 = 85, y1 = 90`* to cover until the next maturity period.

Resolution: Fixed

5.2.2. Deposits may account more `stETH` when user deposit with NATIVE

Severity: Medium Risk

Context: EnderBond.sol#L361

Description: When a user deposits into a bond with `NATIVE`, their ethers will be sent to the `lido` contract and `EnderBond.sol` will receive the corresponding `stETH` in return, then will transfer those `stETH` to the treasury. However, it will send the whole amount, and if there are any `stETH` in the contract, it will account them for the current user, which is wrong, as he will have deposited less.

```
   function deposit(
       address user,
       uint256 principal,
       uint256 maturity,
       uint256 bondFee,
       address token
   ) public payable nonReentrant depositEnabled bondPaused returns (uint256 tokenId) {
       ...
       // token transfer
       if (token == address(0)) {
           if (msg.value != principal) revert InvalidAmount();
           (bool suc, ) = payable(lido).call{value: msg.value}(
               abi.encodeWithSignature("submit(address)", address(this))
           );
           require(suc, "lido eth deposit failed");
           stEthFromDeposit = IERC20(stEth).balanceOf(address(this));
           IERC20(stEth).safeTransfer(address(endTreasury), stEthFromDeposit); <---------- here it gets all stETH balance instead
           endTreasury.depositTreasury(IEnderBase.EndRequest(user, stEth, stEthFromDeposit));
           tokenId = _deposit(user, stEthFromDeposit, maturity, stEth, bondFee);
       } ...
   }
```

Recommendation: Add a check to deposit only the `stETH` equivalent of the user's `NATIVE` deposit.

Resolution: Fixed

5.2.3. `EnderTreasury::treasuryMint` called in `quarter = 5` will block the function forever

Severity: Medium Risk

Context: EnderTreasury.sol#L479

Description: In case `treasuryMint` is not called in the 90 days of quarter 4, quarter then will be 5 which will block the function forever, because of the if statement on line 479.

```
473 function treasuryMint() external onlyRole(DEFAULT_ADMIN_ROLE) {
474         uint256 currentTime = block.timestamp;
475         uint256 mintRate;
476         uint256 currentQuarter = ((currentTime - lastMintTime) / SECONDS_IN_DAY) / 90;
477         uint256 spentDate = ((currentTime - lastUpdateTime) / SECONDS_IN_DAY);
478 
479         if (currentQuarter == 0 || currentQuarter > 4 || spentDate < 90) {
480             return;
481         }
482         if (currentQuarter < 4) {
483             mintRate = mintYearRate / 3;
484             remainedMintRate -= mintRate;
485             if (remainedMintRate < mintRate) {
486                 mintRate += remainedMintRate;
487                 remainedMintRate = 0;
488             }
489             lastUpdateTime = currentTime;
490         } else {
491             mintRate = remainedMintRate;
492             remainedMintRate = 0;
493             lastMintTime = currentTime;
494             lastUpdateTime = currentTime;
495             if (mintYearRate > 100) {
496                 mintYearRate -= 100;
497             }
498             remainedMintRate = mintYearRate;
499         }
500         uint256 mintAmount = (totalEndSupply * mintRate) / 10000;
501         IEndToken(endToken).mint(address(this), mintAmount);
502         totalAddedMintAmount += mintAmount;
503     }
```

Recommendation: Re-implement the function to start a new quarter when a user miss the 4th one.

Resolution: Fixed

5.2.4. EnderStaking::setBondRewardPercentage should update the rewards

Severity: Medium Risk

Context: EnderStaking.sol#L132

Description: Updating the `bondRewardPercentage` without first claiming the rebase rewards will make the accumulated yield from the **previous epoch** be distributed with the new percentages potentially favoring the stakers or bondholders more, depending on if it is being increased or decreased.

Recommendation: Call `_epochStakingReward` before updating the fee to ensure the rewards for the current epoch are claimed.

Resolution: Fixed

5.2.5. Ender architecture is not robust enough and will not support the desired assets and strategies

Severity: Medium Risk

Context: *

Description: The current architecture of the Ender contracts is not robust enough and will not be able to support the desired tokens and strategies in the future. Issues arise from the following implementation decisions:

- `EnderTreasury::_depositInStrategy` handles only `stETH` deposits in `Instadapp` and there is no functionality for the other strategies.
- `EnderTreasury::withdrawFromStrategy` can withdraw only from `Instadapp`.
- `EnderTreasury::getTreasuryTVL` checks only for the `stETH` balance and does not take into account other tokens.
- `EnderTreasury::withdraw` passes only `stETH` address to the withdrawFromStrategy.

Despite the decision of the protocol team to use only `Instadapp` and `stETH` for V1, the desired other yield strategies should be supported also, since `EnderTreasury` migration will not be possible in the future.

Recommendation: Support for the missing assets should be added by either making the important functions `virtual` so they can be overridden in the future by upgrading the version of the contracts or implementing the needed functionality to address the limitations.

Resolution: Acknowledged

5.3. Low/Info severity

5.3.1. No way to withdraw the received ETH

Severity: Low Risk

Context: EnderBond.sol, EnderTreasury.sol

Description: There are defined `receive` functions in both `EnderBond` and `EnderTreasury` contracts, but they don't have the functionality to be claimed, so all the native tokens sent to these contracts will be locked forever.

Recommendation: Consider whether to remove the `receive` functions completely or add an admin-restricted function to claim the tokens.

Resolution: Fixed

5.3.2. `EnderTreasury::withdrawBondFee` lacks `activeBondFee` check and can decrease the backing

Severity: Low Risk

Context: EnderTreasury.sol#L377

Description: The `withdrawBondFee` function lacks check whether the withdrawn fee is more than the currently available bond fee and because of an admin mistake it can decrease the backing by withdrawing more than the available:

```
function withdrawBondFee(uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE) {
    uint256 endETHBalance = IERC20(enderStakeEth).balanceOf(address(this));
    if (endETHBalance < amount) revert InsufficientAmount();

    IERC20(enderStakeEth).safeTransfer(msg.sender, amount);
    totalRewardFeeAmount += amount;
}
```

Recommendation:

```
  function withdrawBondFee(uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE) {
      uint256 endETHBalance = IERC20(enderStakeEth).balanceOf(address(this));
      if (endETHBalance < amount) revert InsufficientAmount();
+     if (amount > IEnderBond(enderBond).activeBondFee()) revert InsufficientActiveFee();

      IERC20(enderStakeEth).safeTransfer(msg.sender, amount);
      totalRewardFeeAmount += amount;
  }
```

Resolution: Acknowledged

5.3.3. Admin setter functions are not managing the roles

Severity: Low Risk

Context: EndToken.sol#L95

Description: Currently, admin setter functions are not revoking the old admin role, nor giving the new admin role. This will lead to mixing the roles - currently, the old owner will have a role and the new one won't and they should be manually given.

Recommendation: Rewrite the setter functions to revoke the admin role of the old admin and give it to the new admin, but make sure to not allow passing the same address as the existing owner:

```
    function setAdmin(address _admin) external onlyRole(DEFAULT_ADMIN_ROLE) {
          if (_admin == address(0)) revert ZeroAddress();
  +       if (_admin == admin) revert AlreadySet();
  +       _grantRole(DEFAULT_ADMIN_ROLE, _admin);
  +       _revokeRole(DEFAULT_ADMIN_ROLE, admin);
          admin = _admin;
      }
```

Resolution: Fixed

5.3.4. Wrong if check assuming the contract is paused

Severity: Low Risk

Context: EnderStaking.sol#L90

Description: `EnderStaking.sol` allows its admins to pause the contract via `stakingContractPause`. However, due to a wrong check in the `stakingContractPaused` modifier, when *`stakingContractPause = true,`* the contract will not be paused, but will be paused when set to **`false`**.

```
modifier stakingContractPaused() {
    if (stakingContractPause != true) revert NotAllowed();
    _;
}
```

Recommendation: Switch the check to be stakingContractPause == true

Resolution: Fixed

5.3.5. Ender upgradeable contracts lack `_disableInitializers()` call

Severity: Low Risk

Context: EnderBond.sol, EnderTreasury.sol, EnderStaking.sol

Description: All of the Ender contracts don't invoke the `_disableInitializers()` function in the constructor and can lead to the attacker initializing the implementation, rather than the proxy itself.

Recommendation: Modify the code, as per OZ suggestion - https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable

Resolution: Fixed

5.3.6. Wrong formula in comments

Severity: Information Risk

Context: EnderBond.sol#L563

Description: `calculateRefractionData()` NatSpec shows an example of the formula in the function, but the formula in the comments is wrong compared to the implementation:

Comment: `1 + rateOfChange * (1 + bondFee) * maturity`

Implementation: *`1000000 + (rateOfChange * (10000 + _bondFee) * _maturity)`*

Recommendation: The implementation is correct, but the comment should be - *`1 + rateOfChange * (1 + bondFee * 0.01) * maturity`*

5.3.7. EnderBond::getPrivateAddress can be removed

Severity: Information Risk

Context: EnderBond.sol#L231-L239

Description: The view function `getPrivateAddress` to check the private addresses is unnecessary since they can simply be made public, avoiding the unnecessary complexity added to the contracts.

```
function getPrivateAddress(uint256 _type) external view returns (address addr) {
  if (_type == 0) revert ZeroAddress();

  if (_type == 1) addr = address(endTreasury);
  else if (_type == 2) addr = address(bondNFT);
  else if (_type == 3) addr = address(endStaking);
  else if (_type == 4) addr = address(depositContract);
  else revert InvalidAddress();
}
```

Recommendation: Remove the `getPrivateAddress` function and make the addresses public.

Resolution: Fixed

5.3.8. EnderBond::calcEpochBondAmount currentTime is not needed

Severity: Information Risk

Context: EnderBond.sol#L547-L554

Description: The `calcEpochBondAmount` function expects a `currentTime` argument, which is only being passed as `block.timestamp` and is not needed, because the `EndToken` rebase rewards are only calculated from the current time.

```
function calcEpochBondAmount(uint256 currentTime) external onlyTreasury returns (uint256 endMintAmount) {
  if (currentTime <= latestRebaseUpdateTime) {
      endMintAmount = 0;
  } else {
      endMintAmount = totalEpochBondPrincipal * 1000 * (currentTime - latestRebaseUpdateTime);
      latestRebaseUpdateTime = currentTime;
  }
}
```

Recommendation: Rewrite the `calcEpochBondAmount` by removing the argument.

Resolution: Fixed

5.3.9. Unused functions and variables only increase the complexity of the contracts

Severity: Information Risk

Context: EnderBond.sol, EnderTreasury.sol, EnderStaking.sol

Description: The following functions, variables, and calculations are unused and only increase the complexity of the contracts:

- EnderStaking:
  - rebasingIndexPerSecond
  - oldSEndBalance
  - rebaseEndAmountPerDay
  - totalRebaseFeeAmount
  - contractSigner
  - calculateRebaseIndexPerSecond
- EnderTreasury:
  - bondYieldBaseRate
  - balanceLastEpoch
  - startTime
  - setBondYieldBaseRate
  - totalEndSupply
- EnderBond:
  - contractSigner
  - isSet

Recommendation: Unless they are being used on the frontend they can be safely removed, since they are not being used anywhere.

Resolution: Fixed