# Competitive Security Assessment

## Ender_V1

Jun 4th, 2024

Secure3

secure3.io

# Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3′s Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.
The comprehensive examination and auditing scope includes:
• Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
• Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
• Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
• Verify the code base is compliant with the most up-to-date industry standards and security best practices.
• Comprehensive line-by-line manual code review of the entire codebase by industry experts.
The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

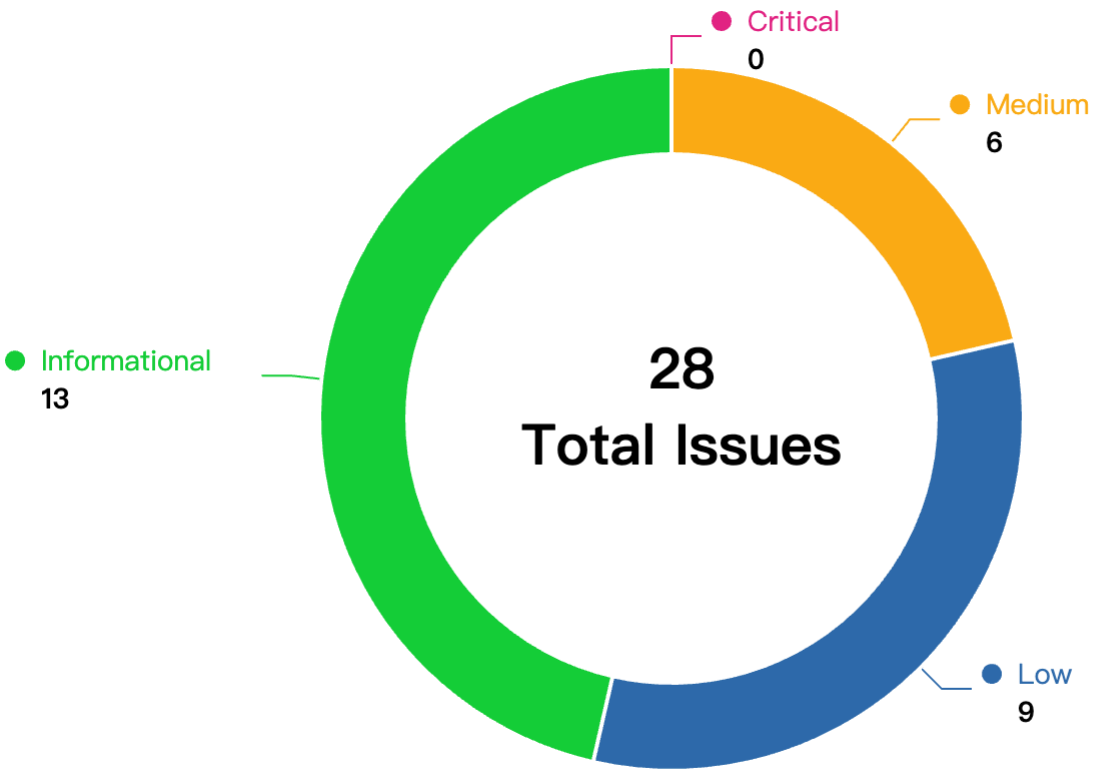
# Overview

| | |
|---|---|
| Project Name | Ender_V1 |
| Language | Solidity |
| Codebase | • https://github.com/enderprotocol/ender-v1/tree/audit<br>• audit version - 40cd7bbada9fcebbf5b858e7fa242287498f35a5<br>• final version - 1418ffbb770391b41dc1fd6718a03da8186fa91c |
| Audit Methodology | • Audit Contest<br>• Business Logic and Code Review<br>• Privileged Roles Review<br>• Static Analysis |

# Audit Scope

| File | SHA256 Hash |
| --- | --- |
| contracts/EnderBond.sol | f31d74819c34668474719ffd72502688936d81187e7f520315b2e19ecf935eb8 |
| contracts/EnderTreasury.sol | 5c88f7d9d546aa944e7b45eefd2fa84d06ac0ebf06e65984e6cb3e26dcc5cec3 |
| contracts/EnderStaking.sol | 1084af8a4f74afc811aef0bc23287b9dbe9cd51778257a98bb4af17089dbf32a |
| contracts/ERC20/EndToken.sol | a997720b2009a04d6f6c0ef52edea08d901e9e79a3c69664563e2933a6daadea |
| contracts/ERC20/SEndToken.sol | f1e8bdc430ecc4b800797b9434a03f375dc43acb67aee0f4593d031df9149c33 |

# Code Assessment Findings




| ID | Name | Category | Severity | Client Response | Contributor |
|---|---|---|---|---|---|
| ED1-1 | Wrong token address used | Logical | High | Fixed | biakia, danielt |
| ED1-2 | Unauthorized Issuance | Privilege Related | High | Fixed | Saaj, grep-er, Cara, danielt, biakia, shealtielanz, BradMoonUESTC, Bauer, n16h7m4r3, Yaodao, 0xac |
| ED1-3 | When depositing ETH into a bond, the principal is incorrect | Logical | Medium | Acknowledged | biakia |
| ED1-4 | Users will receive more sEndAmount when they stake | Logical | Medium | Fixed | Bauer |
| ED1-5 | Transferring funds before calculating shares will result in user losses | Logical | Medium | Fixed | Bauer |
| ED1-6 | Should initialize the reentrancy module. | Language Specific | Medium | Fixed | Saaj, 0xffchain |

| | | | | | |
|---|---|---|---|---|---|
| ED1-7 | First depositor can inflate share price and steal funds from other users | Logical | Medium | Fixed | Bauer |
| ED1-8 | Donation attack | Logical | Medium | Fixed | danielt |
| ED1-9 | Wrong initialization for `SECONDS_IN_DAY` | Logical | Low | Acknowledged | biakia, 0xac, 0xWeb3boy, shealtielanz, danielt, Saaj, Cara |
| ED1-10 | Use upgradeable instead of contract variants of OpenZeppelin | Language Specific | Low | Fixed | Saaj |
| ED1-11 | Use `disableInitializers` to prevent front-running on the initialize function | Language Specific | Low | Fixed | Saaj, biakia, danielt |
| ED1-12 | Role not revoked. | Privilege Related | Low | Fixed | n16h7m4r3 |
| ED1-13 | Pragma non-specification can lead to non-functional / corrupted contract when deployed on Arbitrum | Language Specific | Low | Acknowledged | Saaj |
| ED1-14 | Incorrect interest calculation | Logical | Low | Fixed | Bryce, biakia |
| ED1-15 | If the `_asset` is `USDT`, the approve may fail | Logical | Low | Fixed | biakia, Saaj |
| ED1-16 | Fee sanity check is missing | Logical | Low | Fixed | Saaj |
| ED1-17 | Contract does not account fee on transfer. | Logical | Low | Fixed | n16h7m4r3 |
| ED1-18 | Unsafe Casting can lead to over/underflow | Logical | Informational | Fixed | Saaj |
| ED1-19 | Test data leave in the contract | Logical | Informational | Fixed | danielt |
| ED1-20 | Risky approve | Logical | Informational | Fixed | danielt |
| ED1-21 | Restricted user can transfer | Logical | Informational | Acknowledged | csanuragjain |
| ED1-22 | Redundant state update | Code Style | Informational | Fixed | Bryce, 0xac, biakia, n16h7m4r3, Cara |
| ED1-23 | No prevention for setting old value to again set as new one | Logical | Informational | Fixed | Saaj |

| | | | | | |
|---|---|---|---|---|---|
| ED1-24 | Missing event trigger | Logical | Informational | Fixed | Cara, Bryce |
| ED1-25 | Missing Current Stake Limit check will revert the whole `deposit()` function | Logical | Informational | Acknowledged | Bauer |
| ED1-26 | Lack of reasonable upper boundary | Logical | Informational | Fixed | Yaodao, grep-er, biakia, csanuragjain, 0xac, danielt |
| ED1-27 | Lack of logic for status `3` | Logical | Informational | Fixed | Yaodao, Saaj |
| ED1-28 | Incorrectly Named Modifier for Staking Contract Pause Check | Code Style | Informational | Fixed | 0xac |
| ED1-29 | Incorrect event notification | Logical | Informational | Fixed | Cara, 0xac, danielt, grep-er |
| ED1-30 | Incompatibility With Deflationary Tokens | Language Specific | Informational | Fixed | Saaj, biakia, danielt |

# ED1-1:Wrong token address used

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | High | Fixed | biakia, danielt |

## Code Reference

- code/contracts/EnderTreasury.sol#L184-L190
- code/contracts/EnderTreasury.sol#L307-L326

```
184: function depositTreasury(EndRequest memory param) external onlyBond {
185:         unchecked {
186:             fundsInfo[param.stakingToken] += param.tokenAmt;
187:         }
188:         _depositInStrategy(stEth, priorityStrategy, param.tokenAmt);
189:         emit TreasuryDeposit(param.stakingToken, param.tokenAmt);
190:     }
```

```
307: function withdraw(EndRequest memory param) external onlyBond {
308:         if (param.account == address(0)) revert ZeroAddress();
309:         if (fundsInfo[param.stakingToken] < param.tokenAmt) revert InsufficientAmount();
310:         fundsInfo[param.stakingToken] -= param.tokenAmt;
311:         // bond token transfer
312:         uint256 balanceTreasury = address(this).balance;
313:         if (param.stakingToken != address(0)) {
314:             balanceTreasury = IERC20(param.stakingToken).balanceOf(address(this));
315:         }
316:
317:         if (balanceTreasury >= param.tokenAmt) {
318:             _transferFunds(param.account, param.stakingToken, param.tokenAmt);
319:         } else {
320:             if (balanceTreasury > 0) {
321:                 _transferFunds(param.account, param.stakingToken, balanceTreasury);
322:             }
323:             withdrawFromStrategy(param.account, stEth, priorityStrategy, param.tokenAmt - balan
ceTreasury);
324:         }
325:         emit TreasuryWithdraw(param.stakingToken, param.tokenAmt);
326:     }
```

## Description

**biakia:** In `EnderBond`, the user can call `deposit` to deposit a specified token into a bond. It will call `endTreasury.depositTreasury` to deposit the staked tokens to the treasury:

```
endTreasury.depositTreasury(IEnderBase.EndRequest(user, token, principal));
```

However, in `EnderTreasury`, the function `depositTreasury` is using incorrect staked token `stEth` instead of `param.stakingToken`:

```
function depositTreasury(EndRequest memory param) external onlyBond {
        unchecked {
            fundsInfo[param.stakingToken] += param.tokenAmt;
        }
        _depositInStrategy(stEth, priorityStrategy, param.tokenAmt);
        emit TreasuryDeposit(param.stakingToken, param.tokenAmt);
    }
```

As a result, any deposit using tokens that are not stEth will fail because there is not enough `**stETH**` in the contract `**EnderTreasury**`.

**danielt:** The `**withdraw**` function withdraws the `**stakingToken**` to the account:

```
    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;
        // bond token transfer
        uint256 balanceTreasury = address(this).balance;
        if (param.stakingToken != address(0)) {
            balanceTreasury = IERC20(param.stakingToken).balanceOf(address(this));
        }

        if (balanceTreasury >= param.tokenAmt) {
            _transferFunds(param.account, param.stakingToken, param.tokenAmt);
        } else {
            if (balanceTreasury > 0) {
                _transferFunds(param.account, param.stakingToken, balanceTreasury);
            }
            withdrawFromStrategy(param.account, stEth, priorityStrategy, param.tokenAmt - balanceT
reasury);
        }
        emit TreasuryWithdraw(param.stakingToken, param.tokenAmt);
    }
```

But, note that, two tokens are intended to be withdrawn to the account if `**balanceTreasury < param.tokenAmt**` and `**balanceTreasury > 0**`:

```
        if (balanceTreasury >= param.tokenAmt) {
            _transferFunds(param.account, param.stakingToken, param.tokenAmt);
        } else {
            if (balanceTreasury > 0) {
                _transferFunds(param.account, param.stakingToken, balanceTreasury);
            }
            withdrawFromStrategy(param.account, stEth, priorityStrategy, param.tokenAmt - balanceT
reasury);
        }
```

Firstly, the `balanceTreasury` amount `stakingToken` token are transferred to the account, then, the `param.tokenAmt - balanceTreasury` amount of `stEth` are intended to be transferred to the account.
Note that, if the `stEth` is expensive than the `stakingToken`, malicious users can steal `stEth` from the contract by staking and withdrawing operations.

## Recommendation

**biakia:** Consider following fix:

```
function depositTreasury(EndRequest memory param) external onlyBond {
        unchecked {
            fundsInfo[param.stakingToken] += param.tokenAmt;
        }
        address asset = param.stakingToken == address(0)? stEth : param.stakingToken;
        _depositInStrategy(asset, priorityStrategy, param.tokenAmt);
        emit TreasuryDeposit(param.stakingToken, param.tokenAmt);
    }
```

**danielt:** Updating the token address `stEth` to the `param.stakingToken`.

## Client Response

client response for biakia: Fixed. commit-c393f8f1da5731ee3a6fe1c556eac98a5a0e2e7a
EnderTreasury depositTreasury() function is called with stETH token in the deposit() function of EnderBond contract

```
if (token == address(0)) {
            if (msg.value != principal) revert InvalidAmount();
            uint256 stEthBalBefore = IERC20(stEth).balanceOf(address(this));
            (bool suc, ) = payable(lido).call{value: msg.value}(
                abi.encodeWithSignature("submit(address)", address(this))
            );
            require(suc, "lido eth deposit failed");
            uint256 stEthBalAfter = IERC20(stEth).balanceOf(address(this));
            stEthFromDeposit = stEthBalAfter - stEthBalBefore;
            IERC20(stEth).safeTransfer(endTreasuryAddr, stEthFromDeposit);
            uint256 afterBalance = IERC20(stEth).balanceOf(endTreasuryAddr);
            stEthFromDeposit = afterBalance - beforeBalance;
            IEnderTreasury(endTreasuryAddr).depositTreasury(IEnderBase.EndRequest(user, stEth, stE
thFromDeposit));
            tokenId = _deposit(user, stEthFromDeposit, maturity, stEth, bondFee);
        } else {
            // send directly to the ender treasury
            IERC20(token).safeTransferFrom(msg.sender, endTreasuryAddr, principal);
            uint256 afterBalance = IERC20(stEth).balanceOf(endTreasuryAddr);
            stEthFromDeposit = afterBalance - beforeBalance;
            IEnderTreasury(endTreasuryAddr).depositTreasury(IEnderBase.EndRequest(user, token, stE
thFromDeposit));
            tokenId = _deposit(user, stEthFromDeposit, maturity, token, bondFee);
        }
```

If user deposits the ETH, it is converted the stETH and deposited to the treasury contract, specifically to the instaDappLite strategy.
If user deposits the stETH, it's deposited to the treasury contract directly.
client response for danielt: Fixed. commit-c393f8f1da5731ee3a6fe1c556eac98a5a0e2e7a
Currently, user receives the bondNFT with stETH as staked token even though user deposits the ETH to the EnderBond contract.
Also, user receives the stETH token when user withdraw the bondNFT even if user deposits the ETH to the EnderBond contract.

# ED1-2:Unauthorized Issuance

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Privilege Related | High | Fixed | Saaj, grep-er, Cara, da nielt, biakia, shealtiela nz, BradMoonUESTC, Bauer, n16h7m4r3, Yao dao, 0xac |

## Code Reference

- code/contracts/ERC20/SEndToken.sol#L68
- code/contracts/ERC20/SEndToken.sol#L68-L70
- code/contracts/ERC20/SEndToken.sol#L68-L80
- code/contracts/ERC20/SEndToken.sol#L68-70
- code/contracts/ERC20/SEndToken.sol#L78
- code/contracts/ERC20/SEndToken.sol#L78-L80
- code/contracts/ERC20/SEndToken.sol#L78-80

```
68: function burn(address from, uint256 value) public {
```

```
68: function burn(address from, uint256 value) public {
69:         super._burn(from, value);
70:     }
```

```
68: function burn(address from, uint256 value) public {
69:         super._burn(from, value);
70:     }
71:
72:     function whitelist(address _whitelistingAddress, bool _action) external onlyRole(DEFAULT_ADM
IN_ROLE) {
73:         isWhitelisted[_whitelistingAddress] = _action;
74:         emit WhitelistChanged(_whitelistingAddress, _action);
75:     }
76:
77:     ///for testing purpose
78:     function mint(address to, uint256 amount) public {
79:         _mint(to, amount);
80:     }
```

```
68: function burn(address from, uint256 value) public {
69:         super._burn(from, value);
70:     }
```

```
78: function mint(address to, uint256 amount) public {
```

```
78: function mint(address to, uint256 amount) public {
79:         _mint(to, amount);
80:     }
```

```
78: function mint(address to, uint256 amount) public {
79:         _mint(to, amount);
80:     }
```

## Description

**Saaj:**

## Vulnerability Details

`burn` function in ` SEndToken` contract allows anyone to burn other tokens that can lead to direct loss.
The context is due to using ````ERC20Upgradeable`, `_burn`function by calling with public visibility in` burn``` function without any restriction that leads to exploiting other funds.

```
function burn(address from, uint256 value) public {
        super._burn(from, value);
    }
```

## Impact

The publicly visible ```burn`` function allows an attacker or any malicious user to burn token of others leading to direct loss and damage the whole protocol integrity.

## POC

The below test demonstrate burning of tokens by an authorized user.
The test first mint 100 tokens to a user ` random_One` the burn function is then called by another user ` random_Two `.
The ` random_Two` user calls the `burn` function of `SEndToken` contract and provide input `(random_One, 70e18) ` i.e., burn 70 tokens of ` random_One`.

```
// forge t --mt test_burn -vv
    function test_burn() external {
        vm.prank(random_One); // starting call with 1s random caller
        SEndToken.mint(random_One, 100e18); // calling mint function
        uint256 balance_beforeBurn = SEndToken.balanceOf(random_One); // caching  balance of 1st u
ser
        console.log("balance_beforeBurn: %e", balance_beforeBurn); // logging balance before burn

        vm.assertEq(balance_beforeBurn, 100e18); // asserting balance equal 100e18

        vm.prank(random_Two); // starting call with 1s random caller
        SEndToken.burn(random_One, 70e18); // calling burn function
        uint256 balance_AfterBurn = SEndToken.balanceOf(random_One); // caching  balance of 1st us
er after burn
        console.log("balance_AfterBurn: %e", balance_AfterBurn); // logging balance after burn
        vm.assertEq(balance_AfterBurn, 30e18); // asserting balance after burn
    }
```

The test passes and shows in the end balance of ` **random_One**` which is `**3e19 or 30e18**` means `**70e18**` tokens are successfully burned.

```
[PASS] test_burn() (gas: 72386)
Logs:
  balance_beforeBurn: 1e20
  balance_AfterBurn: 3e19

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.57ms (245.94µs CPU time)

Ran 1 test suite in 40.51ms (1.57ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

**Saaj:**

## Vulnerability Details

`**mint**` function in `**SEndToken**` can be used by a malicious attacker to mint unlimited amount of tokens that can lead to draining of protocol.
As the comment describe it is for testing purpose, then it should be commented out to avoid accidentally left during time of deployment that opens potential of minting tokens without any restriction.

## POC

The below test is use to demonstrate if the code is left in deployed code then it can be used by anyone especially a malicious attacker to mint tokens amount according to desire.

```
        // forge t --mt test_randomMint -vv
    function test_randomMint() external {
        vm.startPrank(random_One); // starting call
        SEndToken.mint(random_Two, 10e18); // calling mint with params

        vm.assertEq(SEndToken.balanceOf(random_Two), 10e18); // asserting balance after minting
    }
```

```
[PASS] test_randomMint() (gas: 62884)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.69ms (177.64µs CPU time)

Ran 1 test suite in 71.27ms (1.69ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

**grep-er:** ## Summary
The `burn` function allows an address to burn a specified amount of tokens from another address. However, there is a critical access control vulnerability in this function that allows any caller to burn tokens from any address without any restrictions or permissions, leading to potential abuse and loss of user funds.

# Vulnerability Details

## Function Overview

The `burn` function is designed to destroy (burn) a specified amount of tokens from a given address, reducing the total supply. The function currently uses the `_burn` function from a parent contract to perform the burning operation:

```
    function burn(address from, uint256 value) public {
        super._burn(from, value);
    }
```

## Vulnerability

The critical issue here is the lack of access control. The function is marked as `public`, allowing any caller to invoke it. Without proper checks or restrictions, any user can call this function to burn tokens from any address, leading to the following risks:

1. **Unauthorized Token Burning**: Malicious users can arbitrarily burn tokens from any address, causing loss of funds for legitimate users.
2. **Total Supply Manipulation**: Attackers can exploit this function to disrupt the token economy by burning significant amounts of tokens from multiple addresses.

## Potential Exploitation

An attacker could exploit this vulnerability by calling the `burn` function with the address and value of their choice, effectively destroying tokens without the owner's consent. For example:

`burn(victimAddress, amount);`
This call would burn the specified `amount` of tokens from `victimAddress`, leading to unauthorized and potentially irrecoverable token loss.
**Cara:** Any user can call the `mint` and `burn` functions in the `SEndToken` contract to mint and destroy SEnd tokens, which can lead to significant asset loss to project and participants. For example, a user could mint a large number of SEnd tokens for themselves, then go to the `EnderStaking` contract, call the `unstake` function, and receive end token as a reward.

```
function burn(address from, uint256 value) public {
    super._burn(from, value);
}

///for testing purpose
function mint(address to, uint256 amount) public {
    _mint(to, 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);

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

    calculateRebaseIndexPerSecond();

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

**danielt:** In the `SEndToken` contract, any user can burn any user's `sEnd` token, which is critical issue to the protocol.

```
    function burn(address from, uint256 value) public {
        super._burn(from, value);
    }
```

**danielt:** In the `SEndToken` contract, any user can mint `sEnd` token, which is critical issue to the protocol.

```
    function mint(address to, uint256 amount) public {
        _mint(to, amount);
    }
```

Malicious user can mint tons of `sEnd` with the `mint` function and dump them for profit.

**biakia:** The `mint` function is missing permission checks:

```
function mint(address to, uint256 amount) public {
        _mint(to, amount);
    }
```

Anyone can call the `mint` function to mint any number of tokens for himself.
The `burn` function is missing permission checks:

```
function burn(address from, uint256 value) public {
        super._burn(from, value);
    }
```

Anyone can call the `burn` function to burn tokens from any account.
**shealtielanz:** The `burn` function in SEndToken.sol lacks any access control and is set to public, therefore it can be called by anyone to burn tokens for any userleading to loss of funds

```
      function burn(address from, uint256 value) public {
        super._burn(from, value);
    }
```

**shealtielanz:** The `mint` function in SEndToken.sol lacks any access control and is set to public, therefore it can be called by anyone to mint tokens for himself, allowing the user to withdraw tokens not deposited leading to loss of funds

```
    function mint(address to, uint256 amount) public {
        _mint(to, amount);
```

**BradMoonUESTC:** This vulnerability is present in the burn function of a smart contract, which lacks proper access control mechanisms such as onlyOwner or require(msg.sender == from). This oversight allows any user to destroy tokens belonging to any address specified by the from parameter. The vulnerability allows unauthorized external parties to permanently decrease the token balance of any address, thereby potentially manipulating the token supply and economy or causing destructive griefing attacks
**Bauer:** The `SEndToken.mint()` function lacks access control, allowing anyone to call this function and mint tokens for themselves.

```
  function mint(address to, uint256 amount) public {
        _mint(to, amount);
    }
```

**n16h7m4r3:** The function `mint()` is unrestricted allowing any wallet to mint arbitrary amount of tokens.
**Yaodao:** The functions `mint()` and `burn()` are used to mint tokens and burn tokens. However, the two functions lack privilege control and can be called by anyone.
As a result, anyone can mint tokens or burn others' tokens without limit.
Furthermore, the `totalSupply` of `sEndToken` will be used in the `EnderStaking` to calculate the `rebasingIndex()`. And the `mint()/burn()` functions will update the value of `totalSupply`.
**0xac:** The `burn` and `mint` functions lack proper access control, allowing any user to potentially mint or burn tokens. This poses a high security risk as unauthorized users could manipulate the token supply.

## Recommendation

**Saaj:** The recommendation is made to use already inherited `ERC20Burnable`, `burn` function instead of `ERC20Upgradeable` that allows only the caller to burn their own tokens only.

```
-    function burn(address from, uint256 value) public {
+    function burn(uint256 amount) public virtual {
-        super._burn(from, value);
+        super._burn(_msgSender(), amount);
     }
```

**Saaj:** Remove or comment out mint() as it can lead to unauthorized mint
**grep-er:** ### Mitigation
To mitigate this vulnerability, the function should include access control mechanisms to ensure that only authorized entities can burn tokens. One approach is to restrict the function to only allow the token owner or an approved operator to burn tokens:

```
function burn(address from, uint256 value) public {
++    require(msg.sender == from || isApprovedOperator(msg.sender, from), "Unauthorized burn attem
pt");
    super._burn(from, value);
}
```

**Cara:** It is recommended to add access controls to the `mint` and `burn` functions.
**danielt:** Adding access control to the `burn` function.
**danielt:** Adding access control on the `mint` function
**biakia:** Consider adding permission checks for `mint` and `burn` functions.
**shealtielanz:** The proper access modifier should be added to this function allowing only addresses with the `burner` role to call it.
**shealtielanz:** The proper access modifier should be added to this function allowing only addresses with the minter role to call it.
**BradMoonUESTC:** To mitigate this vulnerability, it is essential to implement robust access control checks within the burn function. Adding conditions such as onlyOwner or verifying msg.sender == from can ensure that only the token owner or authorized individuals can execute token burning. This change will safeguard user assets against unauthorized access and manipulation, aligning with the fundamental principles of asset security and control within blockchain systems. Prompt patching and auditing of smart contracts with similar functionalities are advised to prevent exploitation.
**Bauer:** The recommended fix is to add access control.
**n16h7m4r3:** Consider restricting the functionallity based on protocol requirements.
**Yaodao:** Recommend adding privilege control logic.
**0xac:** Implement role-based access control for the burn and mint functions. Typically, you would require that the caller has a specific role, such as `MINTER_ROLE`, before allowing these operations.

## Client Response

client response for Saaj: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ER

20/SEndToken.sol#L78-L80
client response for Saaj: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L88-L90
client response for grep-er: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L78-L80
client response for Cara: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L78-L80
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L88-L90
client response for danielt: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L78-L80
client response for danielt: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L88-L90
client response for biakia: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L78-L80
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L88-L90
client response for shealtielanz: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L78-L80
client response for shealtielanz: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L88-L90
client response for BradMoonUESTC: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L78-L80
client response for Bauer: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L88-L90
client response for n16h7m4r3: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L88-L90
client response for Yaodao: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L78-L80
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L88-L90
client response for 0xac: Fixed.
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L78-L80
https://github.com/enderprotocol/ender-v1/blob/80141790418418b37834bef9b74760565352e243/contracts/ERC20/SEndToken.sol#L88-L90

# ED1-3:When depositing ETH into a bond, the principal is incorrect

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Medium | Acknowledged | biakia |

## Code Reference

- code/contracts/EnderBond.sol#L335-L368

```
335: function deposit(
336:         address user,
337:         uint256 principal,
338:         uint256 maturity,
339:         uint256 bondFee,
340:         address token
341:     ) public payable nonReentrant depositEnabled bondPaused returns (uint256 tokenId) {
342:         if (principal < minDepositAmount) revert InvalidAmount();
343:         if (maturity < 7 || maturity > 365) revert InvalidMaturity();
344:         if (token != address(0) && !bondableTokens[token]) revert NotBondableToken();
345:         if (bondFee > 10000) revert InvalidBondFee();
346:
347:         // calculate the fee index
348:         IEndToken(endToken).distributeRefractionFees();
349:         endStaking.epochStakingReward(stEth);
350:         uint256 stEthFromDeposit = principal;
351:         // token transfer
352:         if (token == address(0)) {
353:             if (msg.value != principal) revert InvalidAmount();
354:             (bool suc, ) = payable(lido).call{value: msg.value}(
355:                 abi.encodeWithSignature("submit(address)", address(this))
356:             );
357:             require(suc, "lido eth deposit failed");
358:             stEthFromDeposit = IERC20(stEth).balanceOf(address(this));
359:             IERC20(stEth).safeTransfer(address(endTreasury), stEthFromDeposit);
360:             tokenId = _deposit(user, stEthFromDeposit, maturity, stEth, bondFee);
361:         } else {
362:             // send directly to the ender treasury
363:             IERC20(token).safeTransferFrom(msg.sender, address(endTreasury), principal);
364:             tokenId = _deposit(user, stEthFromDeposit, maturity, token, bondFee);
365:         }
366:
367:         emit Deposit(user, tokenId, stEthFromDeposit, maturity, token, bondFee);
368:     }
```

## Description

**biakia:** In `EnderBond`, the user can call `deposit` to deposit ETH into a bond. It will submit the ETH into lido and use the submitted balance `stEthFromDeposit` as the principal:

```
            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);
            tokenId = _deposit(user, stEthFromDeposit, maturity, stEth, bondFee);
```

The issue there is that **`stEthFromDeposit`** is the balance of the stETH and it will slowly get bigger over time(See this comment:https://github.com/lidofinance/core/blob/master/contracts/0.4.24/StETH.sol#L12-L49). However, the **`stEthFromDeposit`** will be recorded in the **`bonds[tokenId]`** and never be changed:

```
bonds[tokenId] = Bond(
            false,
            principal,
            currentTimestamp,
            maturity,
            token,
            bondFee,
            depositPrincipal,
            totalBondRewardAmount,
            currentTimestamp,
            refractionPrincipal,
            tradingFeeShareIndex,
            rebasingFeeShareIndex
        );
```

When the user withdraw the bond, it will call **`withdraw`** function:

```
function withdraw(uint256 tokenId) external nonReentrant withdrawEnabled bondPaused {
        address nftOwner = bondNFT.ownerOf(tokenId);
        if (msg.sender != nftOwner) revert NotBondNFTOwner();
        _withdraw(tokenId);
        emit Withdrawal(msg.sender, tokenId);
    }
function _withdraw(uint256 _tokenId) internal {
        Bond memory bond = bonds[_tokenId];
        if (bond.withdrawn) revert BondAlreadyWithdrawn();
        if (block.timestamp < (bond.startTime + (bond.maturity * SECONDS_IN_DAY))) revert BondNotM
atured();

        uint256 feeEndEthAmount = (bond.principal * bond.bondFee) / 10000;
        uint256 requireEndEth = bond.principal - feeEndEthAmount;
        address nftOwner = bondNFT.ownerOf(_tokenId);
        uint256 endEthBalance = IEnderStakeEth(enderStakeEth).balanceOf(nftOwner);
        if (endEthBalance < requireEndEth) revert InsufficientEndETH();
        IEnderStakeEth(enderStakeEth).burn(nftOwner, requireEndEth);

        claimRewards(_tokenId);
        bonds[_tokenId].withdrawn = true;

        endTreasury.withdraw(IEnderBase.EndRequest(msg.sender, bond.token, requireEndEth));

        totalDeposit -= bond.principal;

        totalRefractionPrincipal -= bond.refractionPrincipal;
        totalEpochBondPrincipal -= bond.depositPrincipal;
        totalDepositReturn -= requireEndEth;
    }
```

It will only withdraw **`bond.principal`** stETH and ignore the yield generated by lido. As a result, users will lost all their lido yields.

## Recommendation

**biakia:** Consider using shares of lido instead as the **`principal`** in **`deposit`** function:

```
        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 = StETH(stEth).sharesOf(address(this));
        StETH(stEth).transferShares(address(endTreasury), stEthFromDeposit);
        tokenId = _deposit(user, stEthFromDeposit, maturity, stEth, bondFee);
    }
```

In `EnderTreasury`, when it tries to withdraw stETH, the `StETH(stEth).transferShares` function should be called.

## Client Response

client response for biakia: Acknowledged.

# ED1-4:Users will receive more sEndAmount when they stake

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Medium | Fixed | Bauer |

## Code Reference

- code/contracts/EnderStaking.sol#L145-L147

```
145: ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);
146:         uint256 sEndAmount = calculateSEndTokens(amount);
147:         ISEndToken(sEndToken).mint(msg.sender, sEndAmount);
```

## Description

**Bauer:**
The **`EnderStaking.stake()`** function allows users to stake endToken into the protocol to receive rewards. The protocol uses **`endToken.safeTransferFrom()`** to transfer endToken from the user, and then it calculates **`sEndAmount`** based on the transferred amount.

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

        ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);
        uint256 sEndAmount = calculateSEndTokens(amount);
        ISEndToken(sEndToken).mint(msg.sendxer, sEndAmount);

        calculateRebaseIndexPerSecond();
```

However, endToken is a fee-on-transfer token, meaning each transaction incurs a fee, resulting in the protocol receiving fewer tokens than the actual amount transferred.

```
function _transfer(address from, address to, uint256 amount) internal override {
        if (excludeWallets[from] || excludeWallets[to]) {
            super._transfer(from, to, amount);
        } else {
            uint256 fee = (amount * refractionFeePercentage) / 10000;

            if (fee != 0) {
                unchecked {
                    refractionFeeTotal += fee;
                    totalRefractionFee += fee;
                }

                super._transfer(from, address(this), fee);
            }

            super._transfer(from, to, amount - fee);
        }
    }
```

Despite this, the protocol still calculates `sEndAmount` based on the full transferred amount, leading to an overestimation of `sEndAmount`.

## Recommendation

**Bauer:** The recommended fix is to use the actual received balance for calculations and to store the fee separately in a designated address.

## Client Response

client response for Bauer: Fixed.
https://github.com/enderprotocol/ender-v1/pull/78

# ED1-5:Transferring funds before calculating shares will result in user losses

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Medium | Fixed | Bauer |

## Code Reference

- code/contracts/EnderStaking.sol#L141-L152

```
141: function stake(uint256 amount) external stakingEnabled stakingContractPaused {
142:         if (amount < 1e16) revert InvalidAmount();
143:         _epochStakingReward(stEth);
144:
145:         ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);
146:         uint256 sEndAmount = calculateSEndTokens(amount);
147:         ISEndToken(sEndToken).mint(msg.sender, sEndAmount);
148:
149:         calculateRebaseIndexPerSecond();
150:
151:         emit Stake(msg.sender, amount);
152:     }
```

## Description

**Bauer:** The **`EnderStaking.stake()`** function allows users to stake **`endToken`** to receive **`sEndToken`**. The protocol first transfers the **`endToken`** in, then calculates the number of **`sEndTokens`**.

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

        ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);
        uint256 sEndAmount = calculateSEndTokens(amount);
        ISEndToken(sEndToken).mint(msg.sender, sEndAmount);

        calculateRebaseIndexPerSecond();

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

This approach is incorrect and leads to users receiving fewer tokens than expected.
For example, if **`endBalStaking = 1000`** and **`sEndTotalSupply = 1000`**, and a user wants to stake 100 tokens, the rebasingIndex is calculated as:

```
rebasingIndex = (1000 + 100) / 1000 = 1.1
```

Thus, the sEndTokens received would be:

```
sEndTokens = 100 / 1.1 = 90
```

This results in the user receiving fewer tokens than expected. According to the implementation in ERC4626, the protocol should first calculate the number of shares the user will receive, then transfer the user's tokens.

```
    function deposit(uint256 assets, address receiver) public virtual returns (uint256) {
        uint256 maxAssets = maxDeposit(receiver);
        if (assets > maxAssets) {
            revert ERC4626ExceededMaxDeposit(receiver, assets, maxAssets);
        }

        uint256 shares = previewDeposit(assets);
        _deposit(_msgSender(), receiver, assets, shares);

        return shares;
    }
```

## Recommendation

**Bauer:** The recommended fix is to first calculate the shares the user should receive, and then transfer the user's funds.

## Client Response

client response for Bauer: Fixed.
https://github.com/enderprotocol/ender-v1/blob/35706499eed7df308c001b3841b7a413ebbe9a53/contracts/EnderStaking.sol#L154-L165

# ED1-6:Should initialize the reentrancy module.

| Category | Severity | Client Response | Contributor |
|---|---|---|---|
| Language Specific | Medium | Fixed | Saaj, 0xffchain |

## Code Reference

- code/contracts/EnderBond.sol#L162
- code/contracts/EnderBond.sol#L162C4-L179C6

```
162: function initialize(address endToken_, address enderStakeEth_, address _lido, address _signer)
public initializer {
```

```
NaN: function initialize(address endToken_, address enderStakeEth_, address _lido, address _signer)
public initializer {
NaN:         __AccessControl_init();
NaN:         _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
NaN:         __EIP712_init(SIGNING_DOMAIN, SIGNATURE_VERSION);
NaN:         rateOfChange = 100;
NaN:         lido = _lido;
NaN:         setAddress(endToken_, 2);
NaN:         setAddress(enderStakeEth_, 4);
NaN:         // todo set the value according to doc
NaN:         minDepositAmount = 1000000000000000;
NaN:         contractSigner = _signer;
NaN:         bondYieldBaseRate = 300;
NaN:         SECONDS_IN_DAY = 3600; // note for testing purpose we have set it to 10 mint
NaN:         latestRebaseUpdateTime = block.timestamp;
NaN:         depositEnable = true; // for testing purpose
NaN:         isWithdrawPause = true; // for testing purpose
NaN:         bondPause = true; // for testing purpose
NaN:     }
```

## Description

**Saaj:**

## Vulnerability Details

`EnderBond` contract inherits `ReentrancyGuardUpgradeable` but does not initialize in the `initialize` function.

```
import "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol";
```

OpenZeppelin clearly provide guide in context of initializing OZ upgradeable contracts in the initialize function. The guide clearly mention using the ` __{ContractName}_init` method for all upgradeable contracts to initialize them.

## Impact

The `ReentrancyGuardUpgradeable` contract is left uninitialized, leaving the possibility of reentrancy attack as the `nonReentrant` modifier will be of no use.
`deposit` and `withdraw` functions in `EnderBond` contract are directly impact due to this uninitialized contract as they are meant to be protected from reentrancy attack.
This allows an attacker to continuously reenter `deposit` function at time of depositing some amount and use same `msg.value` multiple time to have staked amount more than deposited.
The same applies for `withdraw` can continuously reenter the function and withdraw more amount than deposited and draining the whole protocol.

## POC

Here is a test that demonstrate the impact of uninitialized `ReentrancyGuardUpgradeable` contract.
If change is made to `_status` variable, visibility from private to public we can access it in our test. `_status` value change from `0` to `1`, when we initialize the contract.

```
-    uint256 private _status;
+    uint256 public _status;
```

The test calls the `EnderBond` contract, `initialize` function and checks on the value of `_status`.

```
    // forge t --mt test_initialize -vv
    function test_initialize() external {
        vm.startPrank(Authrized_caller); // starting call
        EnderBond.initialize();
        uint status_reentrancy = EnderBond._status(); // _status variable from ReentrancyGuardUpgr
adeable
        console.log("status_set:", status_reentrancy); // logging status value
        vm.assertEq(status_reentrancy, 0);
    }
```

The test passes clearly shows no change in value of `_status` as `__ReentrancyGuard_init()` is not called in the `initialize` function.

```
[PASS] test_initialize() (gas: 40972)
Logs:
  status_set: 0

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.95ms (222.25µs CPU time)

Ran 1 test suite in 72.01ms (1.95ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

**0xffchain:** When initializing the EnderBond contract, it initializers all concerned modules except the reentrancy module.

```
 function initialize(address endToken_, address enderStakeEth_, address _lido, address _signer) pu
blic initializer {
        __AccessControl_init();
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        __EIP712_init(SIGNING_DOMAIN, SIGNATURE_VERSION);
       //---------------------- more code ->
    }
```

The internal function `__ReentrancyGuard_init()` is provided by Openzepline to initialize the status of the reentrancy state to appropriate status at initialization.

```
 function __ReentrancyGuard_init() internal onlyInitializing {
     __ReentrancyGuard_init_unchained();
 }

function __ReentrancyGuard_init_unchained() internal onlyInitializing {
    ReentrancyGuardStorage storage $ = _getReentrancyGuardStorage();
    $._status = NOT_ENTERED;
}
```

## Recommendation

**Saaj:** The recommendation is made for initializing the `ReentrancyGuardUpgradeable` contract in `EnderBond` implementation contract to have effective protection mechanism.

```
__ReentrancyGuard_init();
```

**0xffchain:** The initialization of EnderBond should include `__ReentrancyGuard_init` function in it.

## Client Response

client response for Saaj: Fixed.
https://github.com/enderprotocol/ender-v1/pull/84
client response for 0xffchain: Fixed.
https://github.com/enderprotocol/ender-v1/pull/84

# ED1-7:First depositor can inflate share price and steal funds from other users

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Medium | Fixed | Bauer |

## Code Reference

- code/contracts/EnderStaking.sol#L141-L152

```
141: function stake(uint256 amount) external stakingEnabled stakingContractPaused {
142:         if (amount < 1e16) revert InvalidAmount();
143:         _epochStakingReward(stEth);
144:
145:         ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);
146:         uint256 sEndAmount = calculateSEndTokens(amount);
147:         ISEndToken(sEndToken).mint(msg.sender, sEndAmount);
148:
149:         calculateRebaseIndexPerSecond();
150:
151:         emit Stake(msg.sender, amount);
152:     }
```

## Description

**Bauer:** In the **`EnderStaking.stake()`** function, the protocol calculates the number of **`sEndAmount`** tokens a user receives using the simplified formula:

```
sEndAmount = _endAmount * totalSupply / endBalStaking
```

where **`endBalStaking`** is the current balance of endToken in the pool. There is a known issue with ERC4626 related to this approach: First depositor can inflate share price and steal funds from other users.
Here's how the attack works:
1.The attacker deposits 1,000,001 tokens, receiving 1,000,001 shares.
2.The attacker then calls unstake() to burn 1,000,000 shares, leaving the pool with only 1 share and 1 token.
3.Suppose another user tries to deposit 2 * 10^18 tokens into the pool. The attacker notices this.Due to the flawed staking logic, the protocol first transfers the user's funds and then calculates shares.
This results in:

```
sEndAmount = 2 * 10^18 * 1 / (2 * 10^18 + 1)
```

Due to precision loss, the calculated sEndAmount is 0(Because the calculation uses **`endToken.balanceOf(address(this))`**, a malicious user can also transfer some funds in advance to make the calculation result zero or cause significant precision loss), causing the user's deposited funds to be entirely lost. The malicious user then calls unstake() to withdraw 1 share:

```
endAmount = 1 * (2 * 10^18 + 1) / 1 = 2 * 10^18 + 1
```

The attacker ends up with all the funds.

## Recommendation

**Bauer:** It is recommended to use a global variable to record the amount of endToken deposited by users.

## Client Response

client response for Bauer: Fixed.
https://github.com/enderprotocol/ender-v1/blob/fc67eae6b4c02ac9f6156967a640a179088b6803/contracts/EnderStaking.sol#L153
https://github.com/enderprotocol/ender-v1/blob/fc67eae6b4c02ac9f6156967a640a179088b6803/contracts/EnderStaking.sol#L174

# ED1-8:Donation attack

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Medium | Fixed | danielt |

## Code Reference

- code/contracts/EnderStaking.sol#L221-L229

```
221: function rebasingIndex() public view returns(uint256) {
222:         uint256 endBalStaking = ISEndToken(endToken).balanceOf(address(this));
223:         uint256 sEndTotalSupply = ISEndToken(sEndToken).totalSupply();
224:         if (endBalStaking == 0 || sEndTotalSupply == 0) {
225:             return 1e18;
226:         } else {
227:             return (endBalStaking * 1e18) / sEndTotalSupply;
228:         }
229:     }
```

## Description

**danielt:** The `rebasingIndex()` function rebases index based on the `endToken` balance of the contract and the totalsupply of `sEndToken` token:

```
    function rebasingIndex() public view returns(uint256) {
        uint256 endBalStaking = ISEndToken(endToken).balanceOf(address(this));
        uint256 sEndTotalSupply = ISEndToken(sEndToken).totalSupply();
        if (endBalStaking == 0 || sEndTotalSupply == 0) {
            return 1e18;
        } else {
            return (endBalStaking * 1e18) / sEndTotalSupply;
        }
    }
```

Note that the `rebasingIndex` function is used to calculate reward on the `claimRebaseValue` function, and the `unstake` function:

```
    function claimRebaseValue(uint256 _sendAmount) public view returns (uint256 reward) {
        reward = (_sendAmount * rebasingIndex()) / 1e18;
    }

    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);

        // transfer token
ISEndToken(endToken).safeTransfer(msg.sender, reward);
        ...
    }
```

So, malicious user, epecially the first user, can magnify the `ISEndToken(endToken).balanceOf(address(this))` by donating the `endToken` to the contract, to get over-distributed reward.
For example

- the first user stake 10 wei `endToken` when the `ISEndToken(sEndToken).totalSupply()` is 1 wei,
- Donate 1 ether `endToken` to the contract
- The reward is 10 wei * 1 ether, which is greater than the cost, 10 wei + 1 ether

## Recommendation

**danielt:** Staking an amount of token into the contract once the contract deployed.

## Client Response

client response for danielt: Fixed.
https://github.com/enderprotocol/ender-v1/blob/eadcf78a3f5d4014e9dbe6880f838a2a2dfa696d/contracts/EnderStaking.sol#L153
https://github.com/enderprotocol/ender-v1/blob/eadcf78a3f5d4014e9dbe6880f838a2a2dfa696d/contracts/EnderStaking.sol#L176

# ED1-9:Wrong initialization for `SECONDS_IN_DAY`

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Acknowledged | biakia, 0xac, 0xWeb3boy, shealtielanz, danielt, Saaj, Cara |

## Code Reference

- code/contracts/EnderBond.sol#L162-L179
- code/contracts/EnderBond.sol#L174
- code/contracts/EnderBond.sol#L385-L386

```
162: function initialize(address endToken_, address enderStakeEth_, address _lido, address _signer)
public initializer {
163:         __AccessControl_init();
164:         _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
165:         __EIP712_init(SIGNING_DOMAIN, SIGNATURE_VERSION);
166:         rateOfChange = 100;
167:         lido = _lido;
168:         setAddress(endToken_, 2);
169:         setAddress(enderStakeEth_, 4);
170:         // todo set the value according to doc
171:         minDepositAmount = 1000000000000000;
172:         contractSigner = _signer;
173:         bondYieldBaseRate = 300;
174:         SECONDS_IN_DAY = 3600; // note for testing purpose we have set it to 10 mint
175:         latestRebaseUpdateTime = block.timestamp;
176:         depositEnable = true; // for testing purpose
177:         isWithdrawPause = true; // for testing purpose
178:         bondPause = true; // for testing purpose
179:     }
```

```
174: SECONDS_IN_DAY = 3600; // note for testing purpose we have set it to 10 mint
```

```
385: (365 * SECONDS_IN_DAY * 10000000000);
386:         uint256 totalBondRewardAmount = depositPrincipal * maturity * SECONDS_IN_DAY * 1000;
```

- code/contracts/EnderStaking.sol#L64-L77
- code/contracts/EnderStaking.sol#L72

```
64: function initialize(address _end, address _sEnd, address _stEth, address _signer) external initi
alizer {
65:         __AccessControl_init();
66:         _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
67:         __EIP712_init(SIGNING_DOMAIN, SIGNATURE_VERSION);
68:         contractSigner = _signer;
69:         stakingEnable = true; // for testing purpose
70:         unstakeEnable = true; // for testing purpose
71:         stakingContractPause = true; // for testing purpose
72:         SECONDS_IN_DAY = 3600; // 3600 - testing value; 86400 - production value
73:         setAddress(_end, 3);
74:         setAddress(_sEnd, 4);
75:         setAddress(_stEth, 5);
76:         bondRewardPercentage = 0;
77:     }
```

```
72: SECONDS_IN_DAY = 3600; // 3600 - testing value; 86400 - production value
```

- code/contracts/EnderTreasury.sol#L110
- code/contracts/EnderTreasury.sol#L474-L475

```
110: SECONDS_IN_DAY = 3600;
```

```
474: uint256 currentQuarter = ((currentTime - lastMintTime) / SECONDS_IN_DAY) / 90;
475:         uint256 spentDate = ((currentTime - lastUpdateTime) / SECONDS_IN_DAY);
```

## Description

**biakia:** In contract `EnderTreasury`, according to the definition of the variable, `SECONDS_IN_DAY` represents the number of seconds in a day, and it should be 86400 instead of 3600. This may cause the calculation in `treasuryMint` function to be wrong.
The same issue exists in `EnderBond` and `EnderStaking`.
**0xac:** In the initialize function of the `EnderStaking` contract, the `SECONDS_IN_DAY` variable is incorrectly set to 3600, which actually corresponds to the number of seconds in an hour, not a day. The correct value, representing the number of seconds in a day, should be 86400. This incorrect setting can lead to significant miscalculations in functions that rely on this constant, affecting token economics, such as rebasing calculations and rewards distribution.
Relevant Code:

```
SECONDS_IN_DAY = 3600; // 3600 - testing value; 86400 - production value
```

**0xWeb3boy:**

```
SECONDS_IN_DAY = 3600;
```

the seconds in day are represented as 3600 , although the natspec says it is for testing purpose but it is better practise to change the code to actual values during the audits.
The contract EnderTreasury.sol doesn't have natspec comment in the `initializeTreasury` function which can be easily forgotten while deploying and so needs to be checked.
It is recommended to change this value to the actual seconds in a day.

**shealtielanz:** The value for the amount of seconds in a day is set as 3600 which is wrong as the actual seconds in a day is 86400
there whenever this value is used in any of the two contracts, wrong values will be generated leading to an accounting error in the protocol.

```
        uint256 currentQuarter = ((currentTime - lastMintTime) / SECONDS_IN_DAY) / 90;
        uint256 spentDate = ((currentTime - lastUpdateTime) / SECONDS_IN_DAY);
```

```
        uint256 depositPrincipal = (getInterest(maturity) * (10000 + (bondFee)) * principal) /
            (365 * SECONDS_IN_DAY * 10000000000);
        uint256 totalBondRewardAmount = depositPrincipal * maturity * SECONDS_IN_DAY * 1000;
```

**danielt:** The `SECONDS_IN_DAY` is initialized with a wrong value `3600`, it should be assigned with `86400`.
As a result, the wrong `SECONDS_IN_DAY` makes huge side effects on functions, `_deposit`, and `_withdraw`.
For example, in the `_deposit` function, the depositPrincipal is magnified 24 times:

```
    function _deposit(
        address user,
        uint256 principal,
        uint256 maturity,
        address token,
        uint256 bondFee
    ) internal returns (uint256 tokenId) {
        endTreasury.depositTreasury(IEnderBase.EndRequest(user, token, principal));
        // mint bond nft
        tokenId = bondNFT.mint(user);
        IEnderStakeEth(enderStakeEth).mint(user, principal, bondFee);
        uint256 currentTimestamp = block.timestamp;
        uint256 refractionPrincipal = calculateRefractionData(user, principal, maturity, tokenId);

        uint256 depositPrincipal = (getInterest(maturity) * (10000 + (bondFee)) * principal) /
            (365 * SECONDS_IN_DAY * 10000000000);
        uint256 totalBondRewardAmount = depositPrincipal * maturity * SECONDS_IN_DAY * 1000;
```

In the `_withdraw` function, the if condtion deponds on the `SECONDS_IN_DAY` also incurs side effect:

```
    function _withdraw(uint256 _tokenId) internal {
        Bond memory bond = bonds[_tokenId];
        if (bond.withdrawn) revert BondAlreadyWithdrawn();
        if (block.timestamp < (bond.startTime + (bond.maturity * SECONDS_IN_DAY))) revert BondNotM
atured();
```

Similar to the `SECONDS_IN_DAY` in the `initialize` function of the `EnderStaking` contract.
**Saaj:**

## Vulnerability Details

`SECONDS_IN_DAY` variable is used in ` EnderStaking`, ` EnderBond` and ` EnderTreasury` contract with value set to `3600`` which is equal to `1 hour```.
The use of `SECONDS_IN_DAY` with value of 1 hour is not recommended as testnet are design to simulate real onchain txns with all factors similar to avoid any issue on deployed contract functionality.

## Impact

The use of less value for `SECONDS_IN_DAY` will impact testing certain functionalities by generating less value if multiplication is carried out and even less which may be rounding down even to `0` if division is carried out.
The can lead to ignoring certain bugs that may arise during normal functionality of smart contract on real onchain.
The issue also lies issue in context of if the value are left accidentally it will result in opening exploit potential for overall protocol. There will be no option left beside redeploying the contract.

## POC

The below test is use to demonstrate the impact of result when `SECONDS_IN_DAY` have value of `1 hour` against `1 day` for calculating reward.

```
    function test_secondsDays() external {
        // uint256 totalBondRewardAmount = depositPrincipal * maturity * SECONDS_IN_DAY * 1000;

        uint256 totalBondRewardAmount1 = 20e18 * 1234567890 * 3600 * 1000; // amount with testing
value
        uint256 totalBondRewardAmount = 20e18 * 1234567890 * 86400 * 1000; // amount with real val
ue

        console.log("totalBondRewardAmount1: %e", totalBondRewardAmount1);
        console.log("totalBondRewardAmount: %e", totalBondRewardAmount);
    }
```

The result clearly shows the difference in result for calculating `totalBondRewardAmount` in function ` _deposit` of ` EnderBond` contract.

```
    [PASS] test_secondsDays() (gas: 4146)
Logs:
  totalBondRewardAmount1: 8.888888808e34
  totalBondRewardAmount: 2.13333331392e36

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.20ms (239.22µs CPU time)

Ran 1 test suite in 96.02ms (2.20ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

**Cara:** `SECONDS_IN_DAY` should be 86400 instead of 3600. An incorrect `SECONDS_IN_DAY` value will have a significant impact on subsequent financial calculations.

```
SECONDS_IN_DAY = 3600;
```

## Recommendation

**biakia:** Consider following fix:

```
SECONDS_IN_DAY = 86400;
```

**0xac:** Update the initialization of `SECONDS_IN_DAY` to use the correct number of seconds per day:

```
SECONDS_IN_DAY = 86400; // Correct number of seconds in a day
```

**0xWeb3boy:** ```
SECONDS_IN_DAY = 86400;

```

**shealtielanz:** The state variable holding the value of SECONDS IN A DAY should be corrected to `86400` which is the actual value for the number of seconds in a day.
**danielt:** Initializing the `SECONDS_IN_DAY` with `86400` instead of `3600`.
**Saaj:** The recommendation is made to use correct value i.e., `86400` which equals to seconds in number of days to avoid miscalculation even during testing.

```
-    SECONDS_IN_DAY = 3600; // equals to 1 hour
+    SECONDS_IN_DAY = 86400; // equal to 1 days
```

**Cara:** It is recommended to change `SECONDS_IN_DAY` to 86400 throughout.

## Client Response

client response for biakia: Acknowledged.
We know this, but we are using the 3600 for testing.
If we decide the deployment to the mainnet, we should update this value as 86400 in all smart contracts.
client response for 0xac: Acknowledged.
We know this, but we are using the 3600 for testing.
If we decide the deployment to the mainnet, we should update this value as 86400 in all smart contracts.
client response for 0xWeb3boy: Acknowledged.
We know this, but we are using the 3600 for testing.
If we decide the deployment to the mainnet, we should update this value as 86400 in all smart contracts.
client response for shealtielanz: Acknowledged.
We know this, but we are using the 3600 for testing.
If we decide the deployment to the mainnet, we should update this value as 86400 in all smart contracts.
client response for danielt: Acknowledged.
We know this, but we are using the 3600 for testing.
If we decide the deployment to the mainnet, we should update this value as 86400 in all smart contracts.
client response for Saaj: Acknowledged.
We know this, but we are using the 3600 for testing.
If we decide the deployment to the mainnet, we should update this value as 86400 in all smart contracts.
client response for Cara: Acknowledged.
We know this, but we are using the 3600 for testing.
```

If we decide the deployment to the mainnet, we should update this value as 86400 in all smart contracts.

# ED1-10:Use upgradeable instead of contract variants of OpenZeppelin

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Language Specific | Low | Fixed | Saaj |

## Code Reference

- code/contracts/EnderBond.sol#L1

```
1: // SPDX-License-Identifier: BSL
```

- code/contracts/EnderStaking.sol#L1

```
1: // SPDX-License-Identifier: BSL
```

- code/contracts/EnderTreasury.sol#L1

```
1: // SPDX-License-Identifier: BSL
```

## Description

**Saaj:**

## Vulnerability Details

When contracts are deployed with upgradeability they must consider the Upgradeable variant of OpenZeppelin rather than the simple contract variants as defined in OZ's official docs.

## Impact

OpenZeppelin's contracts variants when used with upgradeability can result in negative impact on the overall contract functionality.
Check this OpenZeppelin warning about mixing contract variants with upgradeable-contracts ones.

## POC

**`EnderTreasury`**, **`EnderBond`** and **`EnderStaking`** have used contract variant of OZ's with upgradeable variant which is not recommended by OpenZeppelin's:
"you should not be using these contracts in your OpenZeppelin Upgrades project. Instead, make sure to use @openzeppelin/contracts-upgradeable.

Whether using OpenZeppelin contracts or another smart contract library, always make sure that the package is set up to handle upgradeable contracts."

```
File: EnderTreasury.sol

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";


File: EnderBond.sol

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/utils/cryptography/EIP712Upgradeable.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";


File: EnderStaking.sol
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/utils/cryptography/EIP712Upgradeable.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

## Recommendation

**Saaj:** Recommendation is made to use OZ's Upgradeable Variants for `EnderTreasury`, `EnderBond` and `EnderStaking` contract.

## Client Response

client response for Saaj: Fixed. commit-85c76f09520eb9a3403bf1fe653a7c95689e2a72

# ED1-11:Use `disableInitializers` to prevent front-running on the initialize function

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Language Specific | Low | Fixed | Saaj, biakia, danielt |

## Code Reference

- code/contracts/EnderBond.sol#L25
- code/contracts/EnderBond.sol#L162

```
25: contract EnderBond is Initializable, AccessControlUpgradeable, ReentrancyGuardUpgradeable, EIP71
2Upgradeable {
```

```
162: function initialize(address endToken_, address enderStakeEth_, address _lido, address _signer)
public initializer {
```

- code/contracts/EnderStaking.sol#L14
- code/contracts/EnderStaking.sol#L64

```
14: contract EnderStaking is Initializable, EIP712Upgradeable, AccessControlUpgradeable {
```

```
64: function initialize(address _end, address _sEnd, address _stEth, address _signer) external initi
alizer {
```

- code/contracts/EnderTreasury.sol#L17
- code/contracts/EnderTreasury.sol#L78

```
17: contract EnderTreasury is Initializable, AccessControlUpgradeable, EnderELStrategy {
```

```
78: function initializeTreasury(
```

- code/contracts/ERC20/EndToken.sol#L21
- code/contracts/ERC20/EndToken.sol#L45

```
21: contract EndToken is IEndToken, ERC20Upgradeable, AccessControlUpgradeable {
```

```
45: function initialize() external initializer {
```

- code/contracts/ERC20/SEndToken.sol#L8
- code/contracts/ERC20/SEndToken.sol#L25

```
8: contract SEndToken is ERC20Upgradeable, AccessControlUpgradeable {
```

```
25: function initialize() external initializer {
```

## Description

**Saaj:**

## Vulnerability Details

Uninitialized implementation in contract can be taken over by an attacker with initialize function.
All contracts in scope are are **`Upgradeable`** but does not have constructors which makes call to the **`_disableInitializers`** to have protection from implementation initialized to any version.

## Impact

According to OZ'S guideline for protection of initialize function with **`_disableInitializers()`** method, implementation contracts should not remain uninitalized.Uninitalization can lead to attack where a malicious attacker can take over control of contract.
Ensure prevention of initialization by an attacker which will have a direct impact on the contract as the implementation contract's constructor should have **`_disableInitializers()`** method .

## POC

All contracts in scope are upgradeable contracts that utilises **`initialize`** function but does not provide any protection to the method from front running.
The initialize function of **`SEndToken`** does not provide protection to initialize function from front running by implementing **`_disableInitializers()`** method in constructor.

```
function initialize() external initializer {
     __ERC20_init("sEndToken", "sEnd");
     _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
     setStatus(1);
     status = 1;
 }
```

The same issue is found in all other contracts in scope.
**biakia:** The contracts **`EndToken`**, **`SEndToken`**, **`EnderBond`**, **`EnderTreasury`** and **`EnderStaking`** are upgradeable contracts.
The implementation contract behind a proxy can be initialized by any address. This is not a security problem in the sense that it impacts the system directly, as the attacker will not be able to cause any contract to self-destruct or modify any value in the proxy contract. However, taking ownership of implementation contracts can open other attack vectors, like social engineer or phishing attack.
See docs: https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers--
**danielt:** The contracts **`EndToken`**, **`SEndToken`**, **`EnderBond`**, **`EnderStaking`**, and **`EnderTreasury`** are using proxy patterns. The implementation contract behind a proxy can be initialized by any address. It is recommended not

to leave an implementation contract uninitialized, to prevent other attack vectors.
It is recommended to use this to lock implementation contracts that are designed to be called through proxies.
See docs: _disableInitializers function:

## Recommendation

**Saaj:** Add constructor to all contracts in scope that calls `_disableInitializers()` method .
**biakia:** Consider using disableInitializers to prevent front-running on the initialize function:

```
constructor() {
        _disableInitializers();
}
```

**danielt:** Consider calling the `disableInitializers` from the constructor to prevent front-running on the initialize functions.

## Client Response

client response for Saaj: Fixed. commit-faabd3c113c5448cd3ab58be3c11a7849af0d963
client response for biakia: Fixed. commit-faabd3c113c5448cd3ab58be3c11a7849af0d963
client response for danielt: Fixed. commit-faabd3c113c5448cd3ab58be3c11a7849af0d963

# ED1-12:Role not revoked.

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Privilege Related | Low | Fixed | n16h7m4r3 |

## Code Reference

- code/contracts/ERC20/EndToken.sol#L64-L68
- code/contracts/ERC20/EndToken.sol#L87-L93

```
64: function setBond(address addrs) external onlyRole(DEFAULT_ADMIN_ROLE) {
65:         if (addrs == address(0)) revert ZeroAddress();
66:         enderBond = addrs;
67:         _grantRole(ENDERBOND_ROLE, enderBond);
68:     }
```

```
87: function setTreasury(address treasury_) external onlyRole(DEFAULT_ADMIN_ROLE) {
88:         if (treasury_ == address(0)) revert ZeroAddress();
89:
90:         treasury = treasury_;
91:         _grantRole(MINTER_ROLE, treasury);
92:         emit TreasuryContractChanged(treasury_);
93:     }
```

## Description

**n16h7m4r3:** Priveleged role is not revoked from the previous contract. This could lead to skewed calculations and lock of user funds if there are any user interactions in other contracts that have yet to be updated.

## Recommendation

**n16h7m4r3:** Consider revoking the role from the previous contract before granting the role.

## Client Response

client response for n16h7m4r3: Fixed. commit-c07af83df501c78dd0be8377a07e48986a465bc4

# ED1-13:Pragma non-specification can lead to non-functional / corrupted contract when deployed on Arbitrum

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Language Specific | Low | Acknowledged | Saaj |

## Code Reference

- code/contracts/EnderBond.sol#L1

```
1: // SPDX-License-Identifier: BSL
```

- code/contracts/EnderStaking.sol#L1

```
1: // SPDX-License-Identifier: BSL
```

- code/contracts/EnderTreasury.sol#L1

```
1: // SPDX-License-Identifier: BSL
```

- code/contracts/ERC20/EndToken.sol#L1

```
1: // SPDX-License-Identifier: BSL
```

- code/contracts/ERC20/SEndToken.sol#L1

```
1: // SPDX-License-Identifier: BSL
```

## Description

**Saaj:**

## Vulnerability Details

Contracts compiled with non specified versions will result in a non-functional or potentially damaged version that won't behave as expected.
The default behaviour of compiler would be to use the newest version which would mean by default it will be compiled with the 0.8.20 version which will produce broken code.

## Impact

All contracts in scope will be corrupted or non-functional contracts when deployed on Arbitrum chain due to lack of support for `**PUSH0**` opcode.

## POC

Pragma has been set to ^0.8.18 allowing all contracts in scope to be compiled with a compiler equal or greater than 0.8.19. The problem with compiling is that Arbitrum is NOT compatible with 0.8.20 and later.

## Recommendation

**Saaj:** The first recommendation is made to Lock or Constrain pragma as follows: pragma solidity 0.8.18 or pragma solidity >=0.8.0 <=0.8.19.
The second rexommendation is made to change the evm version to `pragma` is version 0.8.20 or later is used.

## Client Response

client response for Saaj: Acknowledged.
We only deposit to the Ethereum mainnet.

# ED1-14:Incorrect interest calculation

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Fixed | Bryce, biakia |

## Code Reference

- code/contracts/EnderBond.sol#L279-L310
- code/contracts/EnderBond.sol#L286

```
279: function getInterest(uint256 maturity) public view returns (uint256 rate) {
280:         uint256 maturityModifier;
281:         if (maturity < 15) {
282:             maturityModifier = linearInterpolate(7, 15, 70, 80, maturity);
283:         } else if (maturity < 30) {
284:             maturityModifier = linearInterpolate(15, 30, 80, 85, maturity);
285:         } else if (maturity < 60) {
286:             maturityModifier = linearInterpolate(30, 60, 85, 85, maturity);
287:         } else if (maturity < 90) {
288:             maturityModifier = linearInterpolate(60, 90, 90, 100, maturity);
289:         } else if (maturity < 120) {
290:             maturityModifier = linearInterpolate(90, 120, 100, 105, maturity);
291:         } else if (maturity < 150) {
292:             maturityModifier = linearInterpolate(120, 150, 105, 110, maturity);
293:         } else if (maturity < 180) {
294:             maturityModifier = linearInterpolate(150, 180, 110, 115, maturity);
295:         } else if (maturity < 220) {
296:             maturityModifier = linearInterpolate(180, 220, 115, 120, maturity);
297:         } else if (maturity < 260) {
298:             maturityModifier = linearInterpolate(220, 260, 120, 125, maturity);
299:         } else if (maturity < 280) {
300:             maturityModifier = linearInterpolate(260, 280, 125, 130, maturity);
301:         } else if (maturity < 320) {
302:             maturityModifier = linearInterpolate(280, 320, 130, 140, maturity);
303:         } else if (maturity < 360) {
304:             maturityModifier = linearInterpolate(320, 360, 140, 150, maturity);
305:         } else {
306:             maturityModifier = 150;
307:         }
308:
309:         rate = bondYieldBaseRate * maturityModifier;
310:     }
```

```
286: maturityModifier = linearInterpolate(30, 60, 85, 85, maturity);
```

## Description

**Bryce:** According to the design, when users have different bond maturities, they will receive different yields. When a user's debt maturity is between 30-60 days, the formula for calculating the maturityModifier has y0 mistakenly set equal to y1: **`linearInterpolate(30, 60, 85, 85, maturity)`**. This causes the numerator of the formula to become zero, resulting in the interest rate in this interval not actually changing in line with the real bond maturity, but instead being fixed at the minimum level. This leads to the user losing a portion of the yield they should have rightfully earned.

**biakia:** In contract **`EnderBond`**, the function **`getInterest`** is used to calculate interest based on maturity. When the maturity is between 30 and 60, the calculation is incorrect:

```
else if (maturity < 60) {
            maturityModifier = linearInterpolate(30, 60, 85, 85, maturity);
        }
```

```
function linearInterpolate(uint256 x0, uint256 x1, uint256 y0, uint256 y1, uint256 x) internal pur
e returns (uint256) {
        return y0 + (x - x0) * (y1 - y0) / (x1 - x0);
    }
```

Both `y0` and `y1` are 85, the formula `y0 + (x - x0) * (y1 - y0) / (x1 - x0)` will be `y0 + (x - x0) * 0 / (x1 - x0) = y0`. This means that when the maturity is between 30 and 60, the interest will always be 85.

## Recommendation

**Bryce:** It is recommended for the project team to modify the values to the correct y0 and y1, in accordance with the business logic design.
**biakia:** Consider following fix:

```
else if (maturity < 60) {
            maturityModifier = linearInterpolate(30, 60, 85, 90, maturity);
        }
```

## Client Response

client response for Bryce: Fixed.
https://github.com/enderprotocol/ender-v1/blob/f6bbddeefe051dd0b91b1b6c601f8cbb082a9a64/contracts/EnderBond.sol#L286
client response for biakia: Fixed.
https://github.com/enderprotocol/ender-v1/blob/f6bbddeefe051dd0b91b1b6c601f8cbb082a9a64/contracts/EnderBond.sol#L286

# ED1-15:If the `_asset` is `USDT`, the approve may fail

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Fixed | biakia, Saaj |

## Code Reference

- code/contracts/EnderTreasury.sol#L254-L301
- code/contracts/EnderTreasury.sol#L258
- code/contracts/EnderTreasury.sol#L291

```
254: function _depositInStrategy(address _asset, address _strategy, uint256 _depositAmt) internal va
lidStrategy(_strategy) {
255:         if (_depositAmt == 0) revert ZeroAmount();
256:         if (_asset == address(0) || _strategy == address(0)) revert ZeroAddress();
257:         if (_strategy == instadapp) {
258:             IERC20(_asset).approve(_strategy, _depositAmt);
259:             IInstadappLiteV2(instadapp).deposit(_depositAmt, address(this)); // note for testin
g we changed the function sig.
260:             instaDappDepositValuations += _depositAmt;
261:         }
262:         // else if (_strategy == lybraFinance) {
263:         //     IERC20(_asset).approve(lybraFinance, _depositAmt);
264:         //     ILybraFinance(lybraFinance).depositAssetToMint(_depositAmt, 0);
265:         // } else if (_strategy == eigenLayer) {
266:         //     //Todo will add the instance while going on mainnet.
267:         // }
268:         emit StrategyDeposit(_asset, _strategy, _depositAmt);
269:     }
270:
271:     /**
272:      * @notice function to deposit available funds to strategies.
273:      * @param _asset address of underlying staked asset e.g: stETH
274:      * @param _strategy address of the strategy from which admin wish to withdraw,
275:      * @param _withdrawAmt amount of stETH admin wants to withdraw from the strategy.
276:      */
277:     function withdrawFromStrategy(
278:         address _to,
279:         address _asset,
280:         address _strategy,
281:         uint256 _withdrawAmt
282:     ) internal validStrategy(_strategy) returns (uint256 _returnAmount) {
283:         if (_asset == address(0) || _strategy == address(0)) revert ZeroAddress();
284:         if (_withdrawAmt == 0) revert ZeroAmount();
285:         address receiptToken = instadapp;
286:         uint256 receiptTokenAmount = IInstadappLiteV2(receiptToken).balanceOf(address(this));
287:         if (IInstadappLiteV2(receiptToken).convertToAssets(receiptTokenAmount) < _withdrawAmt)
revert InsufficientAmount();
288:
289:         if (_strategy == instadapp) {
290:             //Todo set the asset as recipt tokens and need to check the assets ratio while depo
lying on mainnet
291:             IERC20(_asset).approve(instadapp, _withdrawAmt);
292:             IInstadappLiteV2(instadapp).withdraw(_withdrawAmt, _to, address(this));
293:             instaDappWithdrawlValuations += _withdrawAmt;
294:             _returnAmount = _withdrawAmt;
295:         }
296:         // else if (_strategy == lybraFinance) {
297:         //     IERC20(_asset).approve(lybraFinance, _withdrawAmt);
298:         //     _returnAmount = ILybraFinance(lybraFinance).withdraw(address(this), _withdrawAm
t);
299:         // }
300:         emit StrategyWithdraw(_asset, _strategy, _withdrawAmt);
301:     }
```

```
258: IERC20(_asset).approve(_strategy, _depositAmt);
```

```
291: IERC20(_asset).approve(instadapp, _withdrawAmt);
```

## Description

**biakia:** In `EnderTreasury`, the function `_depositInStrategy` and `withdrawFromStrategy` will call `approve` directly:

```
if (_strategy == instadapp) {
            IERC20(_asset).approve(_strategy, _depositAmt);
            IInstadappLiteV2(instadapp).deposit(_depositAmt, address(this)); // note for testing w
e changed the function sig.
            instaDappDepositValuations += _depositAmt;
        }
```

```
if (_strategy == instadapp) {
            //Todo set the asset as recipt tokens and need to check the assets ratio while depolyi
ng on mainnet
            IERC20(_asset).approve(instadapp, _withdrawAmt);
            IInstadappLiteV2(instadapp).withdraw(_withdrawAmt, _to, address(this));
            instaDappWithdrawlValuations += _withdrawAmt;
            _returnAmount = _withdrawAmt;
        }
```

The `approve` function may fail if the asset being used is USDT. This is because USDT uses a different implementation of the ERC20 standard(https://etherscan.io/token/0xdac17f958d2ee523a2206206994597c13d831ec7#code):

```
function approve(address _spender, uint _value) public onlyPayloadSize(2 * 32) {

        // To change the approve amount you first have to reduce the addresses`
        //  allowance to zero by calling `approve(_spender, 0)` if it is not
        //  already 0 to mitigate the race condition described here:
        //  https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
        require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));

        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
    }
```

To change the approve amount you first have to reduce the addresses allowance to zero by calling `approve(_spender, 0)`.
**Saaj:**

## Vulnerability Details

There are 2 instances in `EnderTreasury` where `IERC20.approve()` function is called only once without setting the allowance to zero. USDT require first reducing the address' allowance to zero by calling `approve(_spender, 0)` before.
The first is in `_depositInStrategy` function.

```
            IERC20(_asset).approve(_strategy, _depositAmt);
```

Same issue exist in `withdrawFromStrategy` function.

```
            IERC20(_asset).approve(instadapp, _withdrawAmt);
```

The usage of `approve()` in `_depositInStrategy` and `withdrawFromStrategy` function of `EnderTreasury` contract does not implement the recommended approach for approving USDT tokens as provided by the USDT firm contract.

## Impact

The `_depositInStrategy` and `withdrawFromStrategy` function respectively are impacted as they will revert when call are made to for approving USDT assert.
Reverting of above mentioned functions, making it impossible to deposit `USDT` and similar tokens into the contract for reward purpose.

## POC

The `USDT` contract clearly mentions `approve` function to be used for changing allowance only after resetting the approved amount to zero.

```
// To change the approve amount you first have to reduce the addresses`
//  allowance to zero by calling `approve(_spender, 0)` if it is not
//  already 0
```require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));```
```

Here is a foundry test that demonstrate if an owner `random_Caller` approves `200` tokens from his balance to the spender i.e., `address(1)` it get approved.
However when we made another call to the approve function it reverted as expected based on the context of condition `require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));` that does not allow approving additional amount on already approved value.
```

```
// forge t --mt test_usdtApprove -vv
    function test_usdtApprove() external {
        USDT._mint(random_Caller, 1000); // minting 1000 usdt to a random caller aka owner
        uint256 balance = USDT.balanceOf(random_Caller); // checking balance of user
        console.log("balance Owner", balance); // logging balanceof user

        vm.startPrank(random_Caller); // starting call with the owner

        uint256 allowance = USDT.allowance(random_Caller, address(1)); // caching allowance to add
ress(1)
        console.log("allowance before approval", allowance); // logging  allowance of address(1)
        assertEq(allowance, 0); // asserting no amount approved yest

        USDT.approve(address(1), 200); // calling approve function

        uint256 allowance_1 = USDT.allowance(random_Caller, address(1)); // // caching allowance t
o address(1)
        console.log("Allowance After approval", allowance_1); // caching allowance to address(1)
        assertEq(allowance_1, 200); // asserting no amount approved yest

        vm.expectRevert(); // expecting revert as allpwance is already > 0
        USDT.approve(address(1), 200); // again calling approve function
    }
```

The result clearly shows passing of test on first call to `approve` method while reverted as expected.

```
[PASS] test_usdtApprove() (gas: 91519)
Logs:
  balance Owner 1000
  allowance before approval 0
  Allowance After approval 200

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.37ms (249.88µs CPU time)

Ran 1 test suite in 67.06ms (1.37ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

## Recommendation

**biakia:** Consider using openzeppelin's `SafeERC20.forceApprove()` instead(https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol#L71-L83).
**Saaj:** Use `approve(_spender, 0)` to set the allowance to zero before the line each of the existing approve() calls made.

```
    function withdrawFromStrategy(
        address _to,
        address _asset,
        address _strategy,
        uint256 _withdrawAmt
    ) internal validStrategy(_strategy) returns (uint256 _returnAmount) {
        if (_asset == address(0) || _strategy == address(0)) revert ZeroAddress();
        if (_withdrawAmt == 0) revert ZeroAmount();
        address receiptToken = instadapp;
        uint256 receiptTokenAmount = IInstadappLiteV2(receiptToken).balanceOf(address(this));
        if (IInstadappLiteV2(receiptToken).convertToAssets(receiptTokenAmount) < _withdrawAmt) rev
ert InsufficientAmount();

        if (_strategy == instadapp) {
            //Todo set the asset as recipt tokens and need to check the assets ratio while depolyi
ng on mainnet
+           IERC20(_asset).approve(instadapp, 0); // appove to 0 first for USDT
            IERC20(_asset).approve(instadapp, _withdrawAmt);
            IInstadappLiteV2(instadapp).withdraw(_withdrawAmt, _to, address(this));
            instaDappWithdrawlValuations += _withdrawAmt;
            _returnAmount = _withdrawAmt;
        }
        // else if (_strategy == lybraFinance) {
        //     IERC20(_asset).approve(lybraFinance, _withdrawAmt);
        //     _returnAmount = ILybraFinance(lybraFinance).withdraw(address(this), _withdrawAmt);
        // }
        emit StrategyWithdraw(_asset, _strategy, _withdrawAmt);
    }
```

Same recommendation is made for `**_depositInStrategy**` function.

```
    function _depositInStrategy(address _asset, address _strategy, uint256 _depositAmt) internal v
alidStrategy(_strategy) {
        if (_depositAmt == 0) revert ZeroAmount();
        if (_asset == address(0) || _strategy == address(0)) revert ZeroAddress();
        if (_strategy == instadapp) {
+           IERC20(_asset).approve(_strategy, 0); // appove to 0 first for USDT
            IERC20(_asset).approve(_strategy, _depositAmt);
            IInstadappLiteV2(instadapp).deposit(_depositAmt, address(this)); // note for testing w
e changed the function sig.
            instaDappDepositValuations += _depositAmt;
        }
        // else if (_strategy == lybraFinance) {
        //     IERC20(_asset).approve(lybraFinance, _depositAmt);
        //     ILybraFinance(lybraFinance).depositAssetToMint(_depositAmt, 0);
        // } else if (_strategy == eigenLayer) {
        //     //Todo will add the instance while going on mainnet.
        // }
        emit StrategyDeposit(_asset, _strategy, _depositAmt);
    }
```

## Client Response

client response for biakia: Fixed. commit-1e320a8bff7d9bfd83f1ae5857549559c19c96c9
client response for Saaj: Fixed. commit-1e320a8bff7d9bfd83f1ae5857549559c19c96c9

# ED1-16:Fee sanity check is missing

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Fixed | Saaj |

## Code Reference

- code/contracts/ERC20/EndToken.sol#L70

```
70: function setFee(uint256 fee) public onlyRole(DEFAULT_ADMIN_ROLE) {
```

## Description

**Saaj:**

## Vulnerability Details

`EndToken` have function `setFee` which is used for setting the value for `refractionFeePercentage`.
`refractionFeePercentage` is the main element in `_transfer` function for charging fee by defining the percentage to be charged for every transaction.
However, the `setFee` function does not define any limit for value to be set for the fees to be charge through `refractionFeePercentage`.

## Impact

The lack for check for setting value of `refractionFeePercentage` in `setFee` function could result in setting to high percentage to be set that can impact user by charging high fee for carrying out any transaction.

## POC

The below test in foundry demonstrate any value to be passed in `setFee` function for `refractionFeePercentage`.

```
    // forge t --mt test_setFee -vv
    function test_setFee() external {
        vm.startPrank(Authrized_caller); // starting call
        EndToken.setFee(1000); // calling setfee function

        uint256 fee_set = EndToken.refractionFeePercentage(); // caching fee set
        // console.log("fee set: %e", fee_set); // logging fee
        vm.assertEq(fee_set, 1000); // asserting fee percentage set to 10%

        // fee charge in _transfer function of EndToken
        // uint256 fee = (amount * refractionFeePercentage) / 10000;

        uint256 amount_Transferred = 100e18; // token of amount to be transferred
        uint256 fee = (amount_Transferred * EndToken.refractionFeePercentage()) / 10000; // calcul
ation of fee charge in _transfer function of EndToken
        // console.log("fee charged: %e", fee); // logging fee amount charges
        vm.assertEq(fee, 10e18); // asserting fee charge is equal to 10% of total amount transferr
ed
    }
```

The `**refractionFeePercentage**` is set with value `**1000**` which will impact the `**_transfer**` function to charge `**10%**` percent fees.
The test clearly passed with clearly indicating the `**fee**` charged in `**_transfer**` function to have taken 10% of the whole txn amount.

```
[PASS] test_setFee() (gas: 35065)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.80ms (348.57µs CPU time)

Ran 1 test suite in 96.68ms (2.80ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

## Recommendation

**Saaj:**
The recommendation is made for implementing a check in function to prevent setting too much fees to be charged that can lead to exploiting.

```
function setFee(uint256 fee) public onlyRole(DEFAULT_ADMIN_ROLE) {
+       require(fee <= 500, 'Fee too high'); // value for charging 5% fee or modify according to n
eed
        refractionFeePercentage = fee;
        emit FeeUpdated(fee);
    }
```

## Client Response

client response for Saaj: Fixed.
https://github.com/enderprotocol/ender-v1/blob/514ed36a27c39a6256fa5cd00aac8a11229a967f/contracts/ERC

0/EndToken.sol#L78

# ED1-17:Contract does not account fee on transfer.

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Fixed | n16h7m4r3 |

## Code Reference

- code/contracts/EnderStaking.sol#L145
- code/contracts/EnderStaking.sol#L167

```
145: ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);
```

```
167: ISEndToken(endToken).safeTransfer(msg.sender, reward);
```

## Description

**n16h7m4r3:** Function **`stake()`** and **`unstake()`** in the **`EnderStaking`** contract does not account for the transfer fee implemented in the **`EndToken`** contract, this can be exploited by the wallets to drain fee (EndToken tokens) from the staking contract.

## Recommendation

**n16h7m4r3:** Function needs to be modified to account for the transfer fee.

## Client Response

client response for n16h7m4r3: Fixed.
https://github.com/enderprotocol/ender-v1/blob/f6bbddeefe051dd0b91b1b6c601f8cbb082a9a64/contracts/EnderStaking.sol#L157-L159

# ED1-18:Unsafe Casting can lead to over/underflow

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Informational | Fixed | Saaj |

## Code Reference

- code/contracts/EnderTreasury.sol#L200

```
200: function stakeRebasingReward(address _tokenAddress) external onlyStaking returns (int256 rebase
Reward) {
```

## Description

**Saaj:** ## Vulnerability Details
Unsafe downcasting from uint256 to int256 could lead to overflow/underflow without reverting check OZ's guide.
This is mainly due maximum value of uint256 is `2 ** 256 - 1` while int256 value for its maximum limit is `2 ** 256 / 2 - 1` while also allowing for storing negative value.
In `_ stakeRebasingReward` function multiple `uint256` values are casted to `int256` with carrying out arithmetic operation that is further used for multiple casting without any safe practice.

## Impact

The reward amount can get significantly less or even round down to zero due to multiple unsafe casting that is carried out and stored within different variables which potentially reduce the actual reward value.

## POC

If we conduct a simple test for maximum value of `uint256` and `int256` the test clearly shows the difference in values that each can store.

```
  // forge t --mt test_intValue -vv
    function test_intValue() external {
        uint256 MAX_UINT_TYPE = type(uint256).max;
        int256 MAX_INT_TYPE = type(int256).max;

        console.log("Uint Value:", MAX_UINT_TYPE); // logging status value

        console.log("Int  Value:", uint256(MAX_INT_TYPE)); // logging status value

    }


   [PASS] test_intValue() (gas: 4105)
Logs:
  Uint Value: 115792089237316195423570985008687907853269984665640564039457584007913129639935
  Int  Value: 57896044618658097711785492504343953926634992332820282019728792003956564819967

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.21ms (136.59µs CPU time)

Ran 1 test suite in 47.88ms (1.21ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

## Recommendation

**Saaj:** The recommendation is made for using OpenZeppelin's SafeCast library when casting from one type to another one to avoid the risk of rounding down of reward prize.

## Client Response

client response for Saaj: Fixed. https://github.com/enderprotocol/ender-v1/pull/87

# ED1-19:Test data leave in the contract

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Informational | Fixed | danielt |

## Code Reference

- code/contracts/EnderBond.sol#L176-L178

```
176: depositEnable = true; // for testing purpose
177:         isWithdrawPause = true; // for testing purpose
178:         bondPause = true; // for testing purpose
```

## Description

**danielt:** The `initialize` function sets `depositEnable`, `isWithdrawPause`, and `bondPause` as true. The comments for them also mentioned that their initial values are test data.

```
    function initialize(address endToken_, address enderStakeEth_, address _lido, address _signer)
public initializer {
        ...
        depositEnable = true; // for testing purpose @audit test data?
        isWithdrawPause = true; // for testing purpose
        bondPause = true; // for testing purpose
    }
```

## Recommendation

**danielt:** Removing the test data and using production configurations.

## Client Response

client response for danielt: Fixed. https://github.com/enderprotocol/ender-v1/pull/81

# ED1-20:Risky approve

| Category | Severity | Client Response | Contributor |
|---|---|---|---|
| Logical | Informational | Fixed | danielt |

## Code Reference

- code/contracts/EnderTreasury.sol#L277-L295

```
277: function withdrawFromStrategy(
278:         address _to,
279:         address _asset,
280:         address _strategy,
281:         uint256 _withdrawAmt
282:     ) internal validStrategy(_strategy) returns (uint256 _returnAmount) {
283:         if (_asset == address(0) || _strategy == address(0)) revert ZeroAddress();
284:         if (_withdrawAmt == 0) revert ZeroAmount();
285:         address receiptToken = instadapp;
286:         uint256 receiptTokenAmount = IInstadappLiteV2(receiptToken).balanceOf(address(this));
287:         if (IInstadappLiteV2(receiptToken).convertToAssets(receiptTokenAmount) < _withdrawAmt)
revert InsufficientAmount();
288:
289:         if (_strategy == instadapp) {
290:             //Todo set the asset as recipt tokens and need to check the assets ratio while depo
lying on mainnet
291:             IERC20(_asset).approve(instadapp, _withdrawAmt);
292:             IInstadappLiteV2(instadapp).withdraw(_withdrawAmt, _to, address(this));
293:             instaDappWithdrawlValuations += _withdrawAmt;
294:             _returnAmount = _withdrawAmt;
295:         }
```

## Description

**danielt:** The **`withdrawFromStrategy`** withdraw asset from the **`instadapp`** contract:

```
    function withdrawFromStrategy(
        address _to,
        address _asset,
        address _strategy,
        uint256 _withdrawAmt
    ) internal validStrategy(_strategy) returns (uint256 _returnAmount) {
        ...

        if (_strategy == instadapp) {
            //Todo set the asset as recipt tokens and need to check the assets ratio while depolyi
ng on mainnet
            IERC20(_asset).approve(instadapp, _withdrawAmt);
            IInstadappLiteV2(instadapp).withdraw(_withdrawAmt, _to, address(this));
            instaDappWithdrawlValuations += _withdrawAmt;
            _returnAmount = _withdrawAmt;
        }
```

But, before the withdraw, `IInstadappLiteV2(instadapp).withdraw(_withdrawAmt, _to, address(this));`, an approve action is done before, `IERC20(_asset).approve(instadapp, _withdrawAmt);`.
The approve action is unnecessary because it is only a withdraw operation and no need to approve any tokens previously, and it is also dangeours because the extra approvals can result in token loss for approvers if the `instadapp` contract has any `transferFrom` function.

## Recommendation

**danielt:** Removing redundant and dangeours approve operation.

## Client Response

client response for danielt: Fixed. https://github.com/enderprotocol/ender-v1/pull/82

# ED1-21:Restricted user can transfer

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Informational | Acknowledged | csanuragjain |

## Code Reference

- code/contracts/ERC20/SEndToken.sol#L44

```
44: if (!isWhitelisted[msg.sender]) revert TransactionDisabled();
```

## Description

**csanuragjain:** If status==2 then only whitelisted users are allowed to make transfer.
It is assumed that caller is always the person who is making the transfer. But this assumption is wrong in case of transferFrom where msg.sender is not transferrer
POC:

1. User A is a whitelisted user and User B is non whitelisted
2. Status is currently set to 2 which means transfer is only allowed for whitelisted user which in this case is User A
3. User A should only be able to make transfer of his own balance
4. Still User B can transfer its balance using help from User A
5. User B can give allowance to User A for its fund
6. User A can now call transferFrom which will transfer User B (non whitelisted) to any destination even though User B is non whitelisted
7. This happens since whitelisting check only happens on msg.sender and not on from

```
function verifyStatus() internal view {
        if (status == 1) {
            revert TransactionDisabled();
        } else if (status == 2) {
            if (!isWhitelisted[msg.sender]) revert TransactionDisabled();
        } else if (status == 3) {
            //-----
        }
    }
```

## Recommendation

**csanuragjain:** Check both from and msg.sender in whitelisting

## Client Response

client response for csanuragjain: Acknowledged.

# ED1-22:Redundant state update

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Code Style | Informational | Fixed | Bryce, 0xac, biakia, n1 6h7m4r3, Cara |

## Code Reference

- code/contracts/ERC20/SEndToken.sol#L25-L30
- code/contracts/ERC20/SEndToken.sol#L28-L29
- code/contracts/ERC20/SEndToken.sol#L29

```
25: function initialize() external initializer {
26:         __ERC20_init("sEndToken", "sEnd");
27:         _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
28:         setStatus(1);
29:         status = 1;
30:     }
```

```
28: setStatus(1);
29:         status = 1;
```

```
29: status = 1;
```

## Description

**Bryce:** In the **`initialize`** function, the **`setStatus(1)`** function has already set the global variable status to 1. The subsequent manual setting of the status value to 1 is a redundant operation, resulting in redundant code.
**0xac:** The status state variable is set twice during initialization which is redundant. The **`setStatus(1);`** function call already sets the status to 1, and directly after, status is set again to 1.
**biakia:** The **`initialize()`** function will set the status variable twice, once through the setStatus() function:

```
setStatus(1);
```

and then directly assigning a value to it:

```
status = 1;
```

This redundancy can lead to confusion and potential inconsistencies in the code.
**n16h7m4r3:** State variable **`status`** is already updated when the function **`setStatus()`** is executed.
**Cara:** In the **`initialize`** function of the **`SEndToken`** contract, the **`setStatus`** function is called to set the **`status`** to 1. However, immediately after, the **`status`** is once again assigned the value of 1. This is a repetitive and redundant operation that also consumes additional gas.

```
function initialize() external initializer {
    __ERC20_init("sEndToken", "sEnd");
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    setStatus(1);
    status = 1;
}
```

## Recommendation

**Bryce:** Remove the redundant line of code setting `status = 1`.
**0xac:** Remove the direct assignment `status = 1;` after the `setStatus(1);` function call within the initialize method to clean up the code and avoid redundancy.
**biakia:** Consider following fix:

```
function initialize() external initializer {
        __ERC20_init("sEndToken", "sEnd");
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
        setStatus(1);
    }
```

**n16h7m4r3:** The highlighted code can be safely removed.
**Cara:** It is suggested to remove the duplicate operation `status = 1`.

## Client Response

client response for Bryce: Fixed. https://github.com/enderprotocol/ender-v1/blob/e43aa0b54e383b95b9e5ddb71d96797e8290faf9/contracts/ERC20/SEndToken.sol#L28-L33
client response for 0xac: Fixed. https://github.com/enderprotocol/ender-v1/blob/e43aa0b54e383b95b9e5ddb71d96797e8290faf9/contracts/ERC20/SEndToken.sol#L28-L33
client response for biakia: Fixed. https://github.com/enderprotocol/ender-v1/blob/e43aa0b54e383b95b9e5ddb71d96797e8290faf9/contracts/ERC20/SEndToken.sol#L28-L33
client response for n16h7m4r3: Fixed. https://github.com/enderprotocol/ender-v1/blob/d2391e79a5bd38de32b9ea6ef194222be4f80c5a/contracts/ERC20/SEndToken.sol#L33-L38
client response for Cara: Fixed. https://github.com/enderprotocol/ender-v1/blob/e43aa0b54e383b95b9e5ddb71d96797e8290faf9/contracts/ERC20/SEndToken.sol#L28-L33

# ED1-23:No prevention for setting old value to again set as new one

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Informational | Fixed | Saaj |

## Code Reference

- code/contracts/EnderStaking.sol#L114

```
114: function setsigner(address _signer) external onlyRole(DEFAULT_ADMIN_ROLE) {
```

- code/contracts/EnderTreasury.sol#L151
- code/contracts/EnderTreasury.sol#L173

```
151: function setBondYieldBaseRate(uint256 _newBaseRate) public onlyRole(DEFAULT_ADMIN_ROLE) {
```

```
173: function setPriorityStrategy(address _priorityStrategy) public onlyRole(DEFAULT_ADMIN_ROLE) {
```

- code/contracts/ERC20/EndToken.sol#L64
- code/contracts/ERC20/EndToken.sol#L87

```
64: function setBond(address addrs) external onlyRole(DEFAULT_ADMIN_ROLE) {
```

```
87: function setTreasury(address treasury_) external onlyRole(DEFAULT_ADMIN_ROLE) {
```

## Description

**Saaj:** ## Vulnerability Details
`setBond` function in `EndToken` contract sets address for ` enderBond` without any check for passing old address and setting it again.
The same checks is also not implemented in ` setTreasury` function of the same in `EndToken` contract.
` setBondYieldBaseRate` and `setPriorityStrategy` function of `EnderStaking` contract also lacks this implementation to prevent old address is being as a new one.

## POC

The test below shows if we passed the same address twice it passes.

```
    // forge t --mt test_setBond -vv
    function test_setBond() external {
        vm.startPrank(Authrized_caller); // starting call
        EndToken.setBond(random_One); // calling function and setting address
        console.log("random_One:", random_One); // logging maker_t address
        assertEq(EndToken.enderBond(), random_One); // assering address set
        vm.stopPrank();

        vm.startPrank(Authrized_caller); // starting another new call
        EndToken.setBond(random_One); // calling again
        console.log("random_One:", random_One); // logging maker_t address
        assertEq(EndToken.enderBond(), random_One); // assering again same address setting
    }
```

The result clearly shows same address is passed and assigned to **`enderBond`** again.

```
[PASS] test_setBond() (gas: 43671)
Logs:
  random_One: 0xf4B4bA33c1983c842FBCece05d09C3a233CE1593
  random_One: 0xf4B4bA33c1983c842FBCece05d09C3a233CE1593

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.55ms (209.62µs CPU time)

Ran 1 test suite in 41.12ms (1.55ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

## Recommendation

**Saaj:**
The recommendation is made to have sanity check that prevent setting of same address to be set for all the functions highlighted in above contracts.

```
    function setBond(address addrs) external onlyRole(DEFAULT_ADMIN_ROLE) {
+       require(addrs != enderBond, 'already Set');
        if (addrs == address(0)) revert ZeroAddress();
        enderBond = addrs;
        _grantRole(ENDERBOND_ROLE, enderBond);
    }
```

## Client Response

client response for Saaj: Fixed.
https://github.com/enderprotocol/ender-v1/blob/eadcf78a3f5d4014e9dbe6880f838a2a2dfa696d/contracts/ERC20/EndToken.sol#L74
https://github.com/enderprotocol/ender-v1/blob/eadcf78a3f5d4014e9dbe6880f838a2a2dfa696d/contracts/ERC20/EndToken.sol#L101
https://github.com/enderprotocol/ender-v1/blob/eadcf78a3f5d4014e9dbe6880f838a2a2dfa696d/contracts/EnderStaking.sol#L106

https://github.com/enderprotocol/ender-v1/blob/eadcf78a3f5d4014e9dbe6880f838a2a2dfa696d/contracts/EnderTreasury.sol#L133

# ED1-24:Missing event trigger

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Informational | Fixed | Cara, Bryce |

## Code Reference

- code/contracts/ERC20/EndToken.sol#L64-L68
- code/contracts/ERC20/EndToken.sol#L95-L98

```
64: function setBond(address addrs) external onlyRole(DEFAULT_ADMIN_ROLE) {
65:         if (addrs == address(0)) revert ZeroAddress();
66:         enderBond = addrs;
67:         _grantRole(ENDERBOND_ROLE, enderBond);
68:     }
```

```
95: function setAdmin(address _admin) external onlyRole(DEFAULT_ADMIN_ROLE) {
96:         if (_admin == address(0)) revert ZeroAddress();
97:         admin = _admin;
98:     }
```

## Description

**Cara:** When some critical variables in the contract, such as **`_admin`** changes, an event should be emitted so that the changes of the variable can be tracked off-chain.

```
function setAdmin(address _admin) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (_admin == address(0)) revert ZeroAddress();
    admin = _admin;
}
```

**Bryce:** 1. In the **`setBond`** function, modifications are made to the critical data enderBond in the contract, but does not trigger the corresponding event, which is not conducive to obtaining on-chain data.
2. The **`setAdmin`** function also has the same issue.

## Recommendation

**Cara:** It is recommended to emit an event when critical variables change.
**Bryce:** It is recommended to declare the corresponding event and trigger it in the function.

## Client Response

client response for Cara: Fixed. https://github.com/enderprotocol/ender-v1/pull/74
client response for Bryce: Fixed. https://github.com/enderprotocol/ender-v1/pull/74

# ED1-25:Missing Current Stake Limit check will revert the whole `deposit()` function

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Informational | Acknowledged | Bauer |

## Code Reference

- code/contracts/EnderBond.sol#L352-L361

```
352: if (token == address(0)) {
353:             if (msg.value != principal) revert InvalidAmount();
354:             (bool suc, ) = payable(lido).call{value: msg.value}(
355:                 abi.encodeWithSignature("submit(address)", address(this))
356:             );
357:             require(suc, "lido eth deposit failed");
358:             stEthFromDeposit = IERC20(stEth).balanceOf(address(this));
359:             IERC20(stEth).safeTransfer(address(endTreasury), stEthFromDeposit);
360:             tokenId = _deposit(user, stEthFromDeposit, maturity, stEth, bondFee);
361:         } else {
```

## Description

**Bauer:** In the **`EnderBond.deposit()`** function, if **`token == address(0)`**, the protocol will call **`lido.submit()`** to receive stETH.

```
    // 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);
            tokenId = _deposit(user, stEthFromDeposit, maturity, stEth, bondFee);
```

According to Lido's documentation, there is a daily staking limit of 150,000 ETH. The **`deposit()`** function will revert if this limit is reached.From the docs:

**`Staking rate limits In order to handle the staking surge in case of some unforeseen market conditions, the Lido protocol implemented staking rate limits aimed at reducing the surge's impact on the staking queue & Lido’s socialized rewards distribution model. There is a sliding window limit that is parametrized with _maxStakingLimit and _stakeLimitIncreasePerBlock. This means it is only possible to submit this much ether to the Lido staking contracts within a 24 hours timeframe. Currently, the daily staking limit is set at 150,000 ether. You can picture this as a health globe from Diablo 2 with a maximum of _maxStakingLimit and regenerating with a constant speed per block. When you deposit ether to the protocol, the level of health is reduced by its amount and the current limit becomes smaller and smalle`**

**akeLimit() >= amountToStake, and if it's not you can go with an alternative route. The staking rate limits are denominated in ether, thus, it makes no difference if the stake is being deposited for stETH or using the wstETH shortcut, the limits apply in both cases.`**
However, there are no checks in place to validate whether the current staking limit has been reached before staking funds onto Lido.

## Recommendation

**Bauer:** Check the staking limit.

## Client Response

client response for Bauer: Acknowledged.

# ED1-26:Lack of reasonable upper boundary

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Informational | Fixed | Yaodao, grep-er, biakia, csanuragjain, 0xac, danielt |

## Code Reference

- code/contracts/EnderBond.sol#L186-L188

```
186: function setRateOfChange(uint _rateOfChange) external onlyRole(DEFAULT_ADMIN_ROLE) {
187:         rateOfChange = _rateOfChange;
188:     }
```

- code/contracts/EnderStaking.sol#L132-135
- code/contracts/EnderStaking.sol#L133

```
132: function setBondRewardPercentage(uint256 percent) external onlyRole(DEFAULT_ADMIN_ROLE) {
133:         bondRewardPercentage = percent;
134:         emit PercentUpdated(bondRewardPercentage);
135:     }
```

```
133: bondRewardPercentage = percent;
```

- code/contracts/ERC20/EndToken.sol#L70
- code/contracts/ERC20/EndToken.sol#L70-L73
- code/contracts/ERC20/EndToken.sol#L71
- code/contracts/ERC20/EndToken.sol#L111-L128

```
70: function setFee(uint256 fee) public onlyRole(DEFAULT_ADMIN_ROLE) {
```

```
70: function setFee(uint256 fee) public onlyRole(DEFAULT_ADMIN_ROLE) {
71:         refractionFeePercentage = fee;
72:         emit FeeUpdated(fee);
73:     }
```

```
71: refractionFeePercentage = fee;
```

```
111: function _transfer(address from, address to, uint256 amount) internal override {
112:         if (excludeWallets[from] || excludeWallets[to]) {
113:             super._transfer(from, to, amount);
114:         } else {
115:             uint256 fee = (amount * refractionFeePercentage) / 10000;
116:
117:             if (fee != 0) {
118:                 unchecked {
119:                     refractionFeeTotal += fee;
120:                     totalRefractionFee += fee;
121:                 }
122:
123:                 super._transfer(from, address(this), fee);
124:             }
125:
126:             super._transfer(from, to, amount - fee);
127:         }
128:     }
```

## Description

**Yaodao:** The function `setBondRewardPercentage()` is used to update the value of `bondRewardPercentage`. However, the value of `bondRewardPercentage` is not limited and can be updated to be a high value.

**grep-er:**

In `EndToken::setFee()` and `EnderStaking::setBondRewardPercentage()`

EndToken.sol

```
    function setFee(uint256 fee) public onlyRole(DEFAULT_ADMIN_ROLE) {
        refractionFeePercentage = fee;
        emit FeeUpdated(fee);
    }
```

EnderStaking.sol

```
    function setBondRewardPercentage(uint256 percent) external onlyRole(DEFAULT_ADMIN_ROLE) {
        bondRewardPercentage = percent;
        emit PercentUpdated(bondRewardPercentage);
    }
```

**biakia:** In contract `EndToken`, the function `setFee` is used to set a refraction fee, which will be used in `_transfer` function:

```
            uint256 fee = (amount * refractionFeePercentage) / 10000;

            if (fee != 0) {
                unchecked {
                    refractionFeeTotal += fee;
                    totalRefractionFee += fee;
                }

                super._transfer(from, address(this), fee);
            }
```

It is possible that the `refractionFeePercentage` is set to 100% because there is no reasonable upper boundary in function `setFee`.
**csanuragjain:** It is observed that `refractionFeePercentage` can be set to more than 100%
Since this restriction does not exist, such fees could result in loss of funds for user
**0xac:** The `setBondRewardPercentage` function does not validate that the input percentage is within a reasonable or expected range. This could lead to unexpected behavior if the percentage exceeds 100% (or 10000 in basis points), potentially disrupting the reward distribution logic.
Relevant Code:

```
function setBondRewardPercentage(uint256 percent) external onlyRole(DEFAULT_ADMIN_ROLE) {
    bondRewardPercentage = percent;
    emit PercentUpdated(bondRewardPercentage);
}
```

**danielt:** There is no upper/bottom boundary for the update of the `rateOfChange`:

```
    function setRateOfChange(uint _rateOfChange) external onlyRole(DEFAULT_ADMIN_ROLE) {
        rateOfChange = _rateOfChange;
    }
```

It is used to calculate the fee in the `calculateRefractionData` function:

```
    function calculateRefractionData(
        address user,
        uint256 _principal,
        uint256 _maturity,
        uint256 _tokenId
    ) internal view returns (uint256 refractionPrincipal) {
        if (bondNFT.ownerOf(_tokenId) != user) revert NotBondUser();
        uint avgRefractionIndex = 100 + ((rateOfChange * _maturity) / 100);
        refractionPrincipal = (_principal * avgRefractionIndex) / 100;
    }
```

So, if the `rateOfChange` is set as a extremely big value, the `avgRefractionIndex` will also be magnified as a big value, which is harmful to the protocol
**danielt:** There is no upper/bottom boundary for the update of the `refractionFeePercentage`:

```
    function setFee(uint256 fee) public onlyRole(DEFAULT_ADMIN_ROLE) {
        refractionFeePercentage = fee;
        emit FeeUpdated(fee);
    }
```

It is used to calculate the fee in the `_tranfer` function:

```
    function _transfer(address from, address to, uint256 amount) internal override {
        if (excludeWallets[from] || excludeWallets[to]) {
            super._transfer(from, to, amount);
        } else {
            uint256 fee = (amount * refractionFeePercentage) / 10000;
    ...
    super._transfer(from, to, amount - fee);
    ...
```

So, if the `refractionFeePercentage` is above 10000, the fee will be greater than the transfer amount and revert when do the math of `amount - fee`.

## Recommendation

**Yaodao:** Recommend adding a reasonable upper boundary.
**grep-er:**
recommendation
EndToken.sol

```
    function setFee(uint256 fee) public onlyRole(DEFAULT_ADMIN_ROLE) {
++          require(fee<=10000);
        refractionFeePercentage = fee;
        emit FeeUpdated(fee);
    }
```

EnderStaking.sol

```
   function setBondRewardPercentage(uint256 percent) external onlyRole(DEFAULT_ADMIN_ROLE) {
++      require(percent<=10000);
        bondRewardPercentage = percent;
        emit PercentUpdated(bondRewardPercentage);
    }
```

**biakia:** Consider adding a reasonable upper boundary in function `setFee`.
**csanuragjain:** Ensure that `refractionFeePercentage` cannot be set to more than 100%
**0xac:** Implement a check to ensure that the percent does not exceed the maximum allowed value, typically 100% (or 10000 in basis points), before setting it:

```
function setBondRewardPercentage(uint256 percent) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(percent <= 10000, "Percentage cannot exceed 10000 basis points.");
    bondRewardPercentage = percent;
    emit PercentUpdated(bondRewardPercentage);
}
```

Each of these recommendations aims to correct or improve the contract's robustness, correctness, and clarity in operation, enhancing overall security and functionality.
**danielt:** Adding upper/bottom boundary for the update of the `**rateOfChange**`.
**danielt:** Adding upper/bottom boundary for the update of the `**refractionFeePercentage**`

## Client Response

client response for Yaodao: Fixed. https://github.com/enderprotocol/ender-v1/pull/75
client response for grep-er: Fixed. https://github.com/enderprotocol/ender-v1/pull/75
client response for biakia: Fixed. https://github.com/enderprotocol/ender-v1/pull/75
client response for csanuragjain: Fixed. https://github.com/enderprotocol/ender-v1/pull/75
client response for 0xac: Fixed. https://github.com/enderprotocol/ender-v1/pull/75
client response for danielt: Fixed. https://github.com/enderprotocol/ender-v1/pull/75
client response for danielt: Fixed. https://github.com/enderprotocol/ender-v1/pull/75

# ED1-27:Lack of logic for status 3

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Informational | Fixed | Yaodao, Saaj |

## Code Reference

- code/contracts/ERC20/SEndToken.sol#L40-48
- code/contracts/ERC20/SEndToken.sol#L45

```
40: function verifyStatus() internal view {
41:         if (status == 1) {
42:             revert TransactionDisabled();
43:         } else if (status == 2) {
44:             if (!isWhitelisted[msg.sender]) revert TransactionDisabled();
45:         } else if (status == 3) {
46:             //-----
47:         }
48:     }
```

```
45: } else if (status == 3) {
```

## Description

**Yaodao:** The function `verifyStatus()` is used to verify the status and for `_transfer()` and revert in some conditions. However, if branch `status == 3` exists, there is no logic to deal with the branch.
**Saaj:**

## Vulnerability Details

`SEndToken` contract have `verifyStatus()` function that is left incomplete while being called in the ` _transfer` function.
The issue lies in `_transfer` can executed without any requirement that is needed to be fulfilled by `verifyStatus()` function, if `status` has value of 3.

## Impact

Empty code block for 3rd condition allows silent bypassing without any check that may impact calling of `_transfer` function on specific condition.

## POC

The below test is used to demonstrate if we set the `status` value to `3` and then call the `transfer` function.
The function executed silently and passing condition in `verifyStatus()` as it has no code.

```
// forge t --mt test_verifyStatus -vv
    function test_verifyStatus() external {
        vm.startPrank(Authrized_caller); // starting call
        SEndToken.mint(random_One, 10e18);

        SEndToken.setStatus(3); // setting status to 3
        vm.assertEq(SEndToken.status(), 3); // asserting value is set to 3

        SEndToken.transfer(random_One, random_Two, 1e18); // transferring balance
        vm.assertEq(SEndToken.balanceOf(random_Two), 1e18); // asserting balance transfer
    }
```

The result shows passing of test by transferring balance from one user to another.

```
[PASS] test_verifyStatus() (gas: 113025)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.70ms (213.17µs CPU time)

Ran 1 test suite in 51.64ms (1.70ms CPU time): 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

## Recommendation

**Yaodao:** Recommend adding the correspond logic or removing this if branch.
**Saaj:**
The recommendation is made to add a code in condition **`3`** or remove the **`verifyStatus()`** function from **`_transfer`** function.

## Client Response

client response for Yaodao: Fixed. https://github.com/enderprotocol/ender-v1/pull/77
client response for Saaj: Fixed. https://github.com/enderprotocol/ender-v1/pull/77

# ED1-28:Incorrectly Named Modifier for Staking Contract Pause Check

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Code Style | Informational | Fixed | 0xac |

## Code Reference

- code/contracts/EnderStaking.sol#L89

```
89: modifier stakingContractPaused() {
```

## Description

**0xac:**
The modifier **`stakingContractPaused`** is misleadingly named as it checks if the staking contract is not paused. The current name suggests that it checks for the contract being paused, potentially leading to confusion and misuse in the context where it's applied.
Relevant Code:

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

## Recommendation

**0xac:** Rename the modifier to accurately reflect its functionality. A suggested name is **`stakingContractNotPaused`**:

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

And update all uses of this modifier in the contract to the new name.

## Client Response

client response for 0xac: Fixed. https://github.com/enderprotocol/ender-v1/blob/f6bbddeefe051dd0b91b1b6c601f8cbb082a9a64/contracts/EnderStaking.sol#L96

# ED1-29:Incorrect event notification

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Informational | Fixed | Cara, 0xac, danielt, gr ep-er |

## Code Reference

- code/contracts/ERC20/SEndToken.sol#L32-L37
- code/contracts/ERC20/SEndToken.sol#L32
- code/contracts/ERC20/SEndToken.sol#L37

```
32: function setAddress(address _addr, uint256 _type) public onlyRole(DEFAULT_ADMIN_ROLE) {
33:         if (_addr == address(0)) revert ZeroAddress();
34:
35:         if (_type == 1) staking = _addr;
36:
37:         emit AddressUpdated(_addr, _type);
```

```
32: function setAddress(address _addr, uint256 _type) public onlyRole(DEFAULT_ADMIN_ROLE) {
```

```
37: emit AddressUpdated(_addr, _type);
```

## Description

**Cara:** In the `setAddress` function of the `SEndToken` contract, when `_type` is not 1, the `staking` is not updated, therefore the `AddressUpdated` event should not be emitted. Incorrect event notifications can lead to misleading information for off-chain monitoring tools.

```
function setAddress(address _addr, uint256 _type) public onlyRole(DEFAULT_ADMIN_ROLE) {
    if (_addr == address(0)) revert ZeroAddress();

    if (_type == 1) staking = _addr;

    emit AddressUpdated(_addr, _type);
}
```

**0xac:** The function `setAddress` emits an `AddressUpdated` event regardless of whether `_type` equals 1 or not. This could lead to confusion and unnecessary event logs when `_type` is not equal to 1, as no actual address setting happens for other `_type` values.
**danielt:** The `setAddress` function always emits `AddressUpdated` event:

```
function setAddress(address _addr, uint256 _type) public onlyRole(DEFAULT_ADMIN_ROLE) {
    if (_addr == address(0)) revert ZeroAddress();

    if (_type == 1) staking = _addr;

    emit AddressUpdated(_addr, _type);
}
```

But, in fact, the `staking` is only updated if the `_type` is 1. So, if the `_type` is not 1, the `staking` will not be updated and it is no need to emit the `AddressUpdated` event with wrong parameters.
**grep-er:** ## Summary
The `SEndToken::setAddress()` function allows setting an address for a specific type. However, it emits an event regardless of the type, potentially misleading.

## Vulnerability

The function emits `AddressUpdated` event unconditionally, leading to confusion regarding address updates.

```
function setAddress(address _addr, uint256 _type) public onlyRole(DEFAULT_ADMIN_ROLE) {
    if (_addr == address(0)) revert ZeroAddress();

    if (_type == 1) staking = _addr;

    emit AddressUpdated(_addr, _type);
}
```

## Mitigation

Update the function to emit the event only when the type is valid.

## Recommendation

**Cara:** It is advised to emit event notifications only when the `staking` is actually updated.
**0xac:** Modify the `setAddress` method to only emit the `AddressUpdated` event if `_type` equals 1, ensuring that events are only logged when meaningful changes occur.
**danielt:** Only emitting event if the `_type` is 1.
**grep-er:** ## Recommendation
Add a condition to emit the event only when the type matches the expected value.

```
    function setAddress(address _addr, uint256 _type) public onlyRole(DEFAULT_ADMIN_ROLE) {
        if (_addr == address(0)) revert ZeroAddress();

        if (_type == 1) staking = _addr;
++          else revert();

        emit AddressUpdated(_addr, _type);
    }
```

## Client Response

client response for Cara: Fixed. https://github.com/enderprotocol/ender-v1/pull/83
client response for 0xac: Fixed. https://github.com/enderprotocol/ender-v1/pull/83
client response for danielt: Fixed. https://github.com/enderprotocol/ender-v1/pull/83
client response for grep-er: Fixed. https://github.com/enderprotocol/ender-v1/pull/83

# ED1-30:Incompatibility With Deflationary Tokens

| Category | Severity | Client Response | Contributor |
|---|---|---|---|
| Language Specific | Informational | Fixed | Saaj, biakia, danielt |

## Code Reference

- code/contracts/EnderBond.sol#L361-L365
- code/contracts/EnderBond.sol#L444

```
361: } else {
362:             // send directly to the ender treasury
363:             IERC20(token).safeTransferFrom(msg.sender, address(endTreasury), principal);
364:             tokenId = _deposit(user, stEthFromDeposit, maturity, token, bondFee);
365:         }
```

```
444: endTreasury.withdraw(IEnderBase.EndRequest(msg.sender, bond.token, requireEndEth));
```

- code/contracts/EnderStaking.sol#L141-L152

```
141: function stake(uint256 amount) external stakingEnabled stakingContractPaused {
142:         if (amount < 1e16) revert InvalidAmount();
143:         _epochStakingReward(stEth);
144:
145:         ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);
146:         uint256 sEndAmount = calculateSEndTokens(amount);
147:         ISEndToken(sEndToken).mint(msg.sender, sEndAmount);
148:
149:         calculateRebaseIndexPerSecond();
150:
151:         emit Stake(msg.sender, amount);
152:     }
```

- code/contracts/EnderTreasury.sol#L259
- code/contracts/EnderTreasury.sol#L292

```
259: IInstadappLiteV2(instadapp).deposit(_depositAmt, address(this)); // note for testing we changed
the function sig.
```

```
292: IInstadappLiteV2(instadapp).withdraw(_withdrawAmt, _to, address(this));
```

## Description

**Saaj:**

## Vulnerability Details

The problem arises when every transfer of underlying tokens blocking all functions in for the token, since the contract wrongly assumes balances values.

This becomes particularly problematic in the following scenario: a market for USDT is running without problems, then they activate the fee.

## Impact

Fee on transfer can result in always receiving and transferring less funds than intended causing loss to both project and user.

## POC

Some tokens take a transfer fee(e.g. STA, PAXG), some do not currently charge a fee but may do so in the future (e.g. USDT, USDC)see WeirdERC20for reference.
**biakia:** In **`EnderBond`**, the function **`deposit`** will transfer token to the treasury:

```
IERC20(token).safeTransferFrom(msg.sender, address(endTreasury), principal);
```

If the **`token`** is a deflationary token, the input param **`principal`** may not be equal to the received amount due to the charged transaction fee. When user withdraws tokens from treasury, it may fail due to insufficient tokens.
**danielt:** The **`stake`** in the contract contains a critical flaw that
fails to correctly handle fee-on-transfer tokens. The contract assumes that the full amount of tokens will be transferred to the contract. However, if the token deducts a fee on the transfer, the contract will not receive the full amount. As a result, when the function attempts to withdraw the same amount from the **`endToken`**, the contract may not have enough tokens to perform all the withdrawals.

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

        ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);
        uint256 sEndAmount = calculateSEndTokens(amount);
        ISEndToken(sEndToken).mint(msg.sender, sEndAmount);

        calculateRebaseIndexPerSecond();

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

## Recommendation

**Saaj:** If the protocol wants to use all possible tokens, a way to handle these tokens must be implemented.
A possible way to do it is to check the balance of the contract before and after every time a token is transferred to see the effective quantity.
**biakia:** Consider following fix:

```
            uint256 before = IERC20(token).balanceOf(address(endTreasury));
            // send directly to the ender treasury
            IERC20(token).safeTransferFrom(msg.sender, address(endTreasury), principal);
            stEthFromDeposit = IERC20(token).balanceOf(address(endTreasury)) - before;
            tokenId = _deposit(user, stEthFromDeposit, maturity, token, bondFee);
```

**danielt:** Calculating the actual balance received by subtracting the balance of the token after the transfer to that of before the transfer, and applying the actual balance for the following business.

```
uint256 initialBalance = ISEndToken(endToken).balanceOf(address(this));
ISEndToken(endToken).safeTransferFrom(msg.sender, address(this), amount);
uint256 finalBalance = ISEndToken(endToken).balanceOf(address(this));
uint256 actualReceived = finalBalance - initialBalance;
```

## Client Response

client response for Saaj: Acknowledged. We only use the stETH token in the current version
client response for biakia: Fixed. https://github.com/enderprotocol/ender-v1/blob/d67d208b319954e7b4f588c2c21323af78c26a27/contracts/EnderBond.sol#L334-L376
client response for danielt: Acknowledged. Currently we don't deduct the fee from the staking token ( in specifically end token). Therefore I think we don't need to consider this problem

# Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.