Competitive Security Assessment

Ender Deposit

Jan 11th, 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

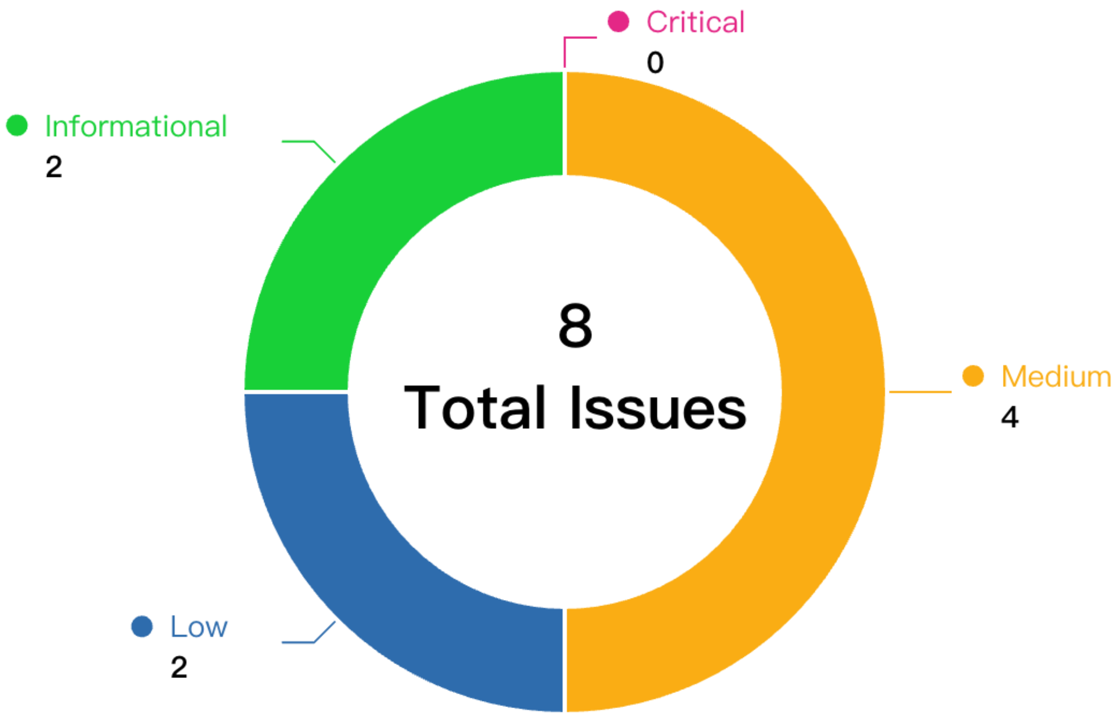
Project Detail

Project Name	Ender Deposit
Platform & Language	Solidity
Codebase	• https://github.com/enderprotocol/depositContract • audit commit - ed4136ed9091dfc565af5d3666e0b53fec28cca8 • final commit - b378b379dbeb7c3fc5a3035e69ea454a86d375ab
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
contracts/EnderBondLiquidityDeposit.sol	**2b210e8ac8b46df3219a1dbc8737a57e9f0b39133b19369bf1cd6c08e450d8b5**

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
END-1	Use `safeTransferFrom` instead of `transferFrom`	Logical	Medium	Acknowledged	zigzag, toffee
END-2	`deposit()` issue with transfer-on-fee/deflationary tokens	Logical	Medium	Acknowledged	toffee
END-3	`deposit()` reward accounting issue with `stETH` rebase and `token`	Logical	Medium	Acknowledged	toffee
END-4	`totalStaked` will be reset to stEth's balance when user deposit	Logical	Medium	Fixed	ethprinter

END-5	**Use disableInitializers to prevent any future reinitialization**	**Logical**	**Low**	**Acknowledged**	**zigzag**
END-6	**Redundant logic in `calculatingSForReward()`**	**Logical**	**Low**	**Fixed**	**ethprinter**
END-7	**Missing __ReentrancyGuard_init()**	**Logical**	**Informational**	**Fixed**	**zigzag**
END-8	**redundant code with hardhat console in production deployment**	**Logical**	**Informational**	**Fixed**	**toffee, zigzag**

END-1:Use `safeTransferFrom` instead of `transferFrom`

Category	Severity	Client Response	Contributor
Logical	**Medium**	**Acknowledged**	**zigzag, toffee**

Code Reference

- code/contracts/EnderBondLiquidityDeposit.sol#L184

```
184:IERC20(token).transferFrom(msg.sender, address(this), principal);

184:IERC20(token).transferFrom(msg.sender, address(this), principal);
```

Description

zigzag : the return value of transfer and transferFrom function is checked, and it can be failure.
toffee : the return value of ERC20 transferFrom is not checked, and it could be failure. While the `token` is controlled by the owner and reduces the risk of fake tokens, it is still a good idea to use `safeTransferFrom` to make sure the transfer is success before making contract accounting state changes.

Recommendation

zigzag : check the return value of the `transfer` and `transferFrom` to make sure the token transfer is successful, or simply use the `SafeERC20` - https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol lib
toffee : use `safeTransferFrom` of `SafeERC20` library https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol

Client Response

Acknowledged. This library is mainly used for catch the failure and get the transferred return value. which we don't need in our case. So it should not impact anything.

END-2: `deposit()` issue with transfer-on-fee/deflationary tokens

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	toffee

Code Reference

- code/contracts/EnderBondLiquidityDeposit.sol#L184

```
184:IERC20(token).transferFrom(msg.sender, address(this), principal);
```

Description

toffee : In the `deposit()` function, it there is transfer for the `token` of `principal` and uses `principal` directly for accounting.

If the `token` is a transfer-on-fee/deflationary token, the actually received amount could be less than `principal`, and as a result, it will introduce accounting error

Recommendation

toffee : Consider getting the actual received amount by calculating the difference of token balance before and after the `transferFrom`.

```
else {
            // send directly to the deposit contract
+           uint balanceBefore = IERC20(token).balanceOf(address(this));
            IERC20(token).transferFrom(msg.sender, address(this), principal);
+           uint balanceafter = IERC20(token).balanceOf(address(this));
+           principal = balanceafter - balanceBefore; 

        }
```

Client Response

Acknowledged.we are depositing the stETH token and ETH which are neither taxed token nor deflationary. So it's fine to use it.

END-3: `deposit()` reward accounting issue with `stETH` rebase and `token`

Category	Severity	Client Response	Contributor
Logical	**Medium**	**Acknowledged**	**toffee**

Code Reference

- code/contracts/EnderBondLiquidityDeposit.sol#L157

```
157:function deposit(
```

Description

toffee : There are few potential issues with `deposit()`

First, `totalStaked += principal` is updated for both Lido and other token. However, in `calculatingSForReward` only accounts for the balance of stEth from Lido. Consider when there are two deposit calls for lido and other token seperatly , the `uint256 reward = IERC20(stEth).balanceOf(address(this)) - totalStaked;` would never be positive, while in reality, the `stETH` is rebased and has positive reward value since `lido.submit()`

Second, for `token` it does not account for different decimals and assumes 1e18, and it can mess up the `bonds` as it uses fixed `expandTo6Decimal`

Recommendation

toffee : consider use different states to record `totalStaked` for lido and other tokens and checks the `IERC(token).decimals`

Client Response

Acknowledged. same, now we are calculating it on the bases of stETH's share.

END-4: `totalStaked` will be reset to stEth's balance when user deposit

Category	Severity	Client Response	Contributor
Logical	**Medium**	**Fixed**	**ethprinter**

Code Reference

- code/contracts/EnderBondLiquidityDeposit.sol#L172-L176

```
172:uint256 reward = IERC20(stEth).balanceOf(address(this)) - totalStaked;
173:        if (reward > 0){
174:            calculatingSForReward();
175:            totalStaked += reward;
176:        }
```

Description

ethprinter : In `depost()` function, it make `totalStaked += reward` when `IERC20(stEth).balanceOf(address(this)) > totalStaked;`, however the `reward` is calculated from `IERC20(stEth).balanceOf(address(this)) - totalStaked`, which means the final formula will be like `totalStaked = IERC20(stEth).balanceOf(address(this)) - totalStaked + totalStaked`, and the `totalStaked` will be set to `IERC20(stEth).balanceOf(address(this))`, which means if a user deposit another bondableTokens apart from `stEth`, the `totalStaked` will be overrided and cause some unexpected results.

Recommendation

ethprinter : Consider remove the assignment statement or process each token individually.

Client Response

Fixed. we have removed the reward calculation logic from this function because we are now using the stETH share based functions.

END-5:Use disableInitializers to prevent any future reinitialization

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	zigzag

Code Reference

- code/contracts/EnderBondLiquidityDeposit.sol#L67

```
67:function initialize(address _stEth, address _lido, address _signer, address _admin) public initializer {
```

Description

zigzag : The EnderPreLounchDeposit contract serves as an implementation contract for the TakerUpgradeableProxy proxy. It 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--

Recommendation

zigzag : Consider using `disableInitializers` :

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

Client Response

Acknowledged. we were using the hardhat upgrades for proxy. but at the time of mainnet deployment, it reverted and partially deployed the implementation contract. So then we used the simple proxy contract where we can call implementation once a time. and we called at the same time. So now the initialize function is disabled.

END-6:Redundant logic in `calculatingSForReward()`

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	ethprinter

Code Reference

- code/contracts/EnderBondLiquidityDeposit.sol#L173-L176
- code/contracts/EnderBondLiquidityDeposit.sol#L214-L220

```
173:if (reward > 0){
174:            calculatingSForReward();
175:            totalStaked += reward;
176:        }

214:function calculatingSForReward() internal {
215:        uint256 reward = IERC20(stEth).balanceOf(address(this)) - totalStaked;
216:        if (reward > 0){
217:            // multipling the rewardShareIndex with 1e6 to avoid underflow
218:            rewardShareIndex = rewardShareIndex + ((reward * expandTo6Decimal())/totalStaked);
219:        }
220:    }
```

Description

ethprinter : In `deposit()` function, it already checked `reward > 0` in line 172-173, however,It does the same check again in `calculatingSForReward()` which is duplicated because the state of `IERC20(stEth).balanceOf(address(this))` and `totalStaked` doesn't change between the two processes.

Recommendation

ethprinter : Remove the redundant code to make the logic clear and easy to understand.

Client Response

Fixed. We're not using calculatingSForReward() anymore because the stEth mainnet contract is giving the round of value so we use the direct stEth functions to calculate reward of users

END-7:Missing __ReentrancyGuard_init()

Category	Severity	Client Response	Contributor
Logical	**Informational**	**Fixed**	**zigzag**

Code Reference

- code/contracts/EnderBondLiquidityDeposit.sol#L67

```
67:function initialize(address _stEth, address _lido, address _signer, address _admin) public initializer {
```

Description

zigzag : https://www.npmjs.com/package/@openzeppelin/contracts-upgradeable/v/4.9.2?activeTab=code

Most contracts use the delegateCall proxy pattern and hence their implementations require the use of initialize() functions instead of constructors. This requires derived contracts to call the corresponding init functions of their inherited base contracts. This is done in most places except a few. The inherited base classes do not get initialized which may lead to undefined behavior.

For the upgradeable variants of OpenZipplin contracts, they should be initialized by calling the __***_init() function in the initializer function.

Therefore, initialize() should call __ReentrancyGuard_init() .

Recommendation

zigzag :

```
    function initialize(address _stEth, address _lido, address _signer, address _admin) public initializer {
        __Ownable_init();
        __ReentrancyGuard_init();
        __EIP712_init(SIGNING_DOMAIN, SIGNATURE_VERSION);
        stEth = _stEth;
        lido = _lido;
        signer = _signer;
        admin = _admin;
        _transferOwnership(admin);
        bondableTokens[_stEth] = true;
        minDepositAmount = 100000000000000000;
    }
```

Client Response

Fixed. we have implemented the reentrancy guard

END-8:redundant code with hardhat console in production deployment

Category	Severity	Client Response	Contributor
Logical	**Informational**	**Fixed**	**toffee, zigzag**

Code Reference

- code/contracts/EnderBondLiquidityDeposit.sol#L9
- code/contracts/EnderBondLiquidityDeposit.sol#L168-L170
- code/contracts/EnderBondLiquidityDeposit.sol#L170
- code/contracts/EnderBondLiquidityDeposit.sol#L198

```
9:import "hardhat/console.sol";

9:import "hardhat/console.sol";

168:console.log("sssssss");
169:        address signAddress = _verify(userSign);
170:        console.log("sssssss--------", signAddress, signer);

170:console.log("sssssss--------", signAddress, signer);

198:console.log("0000000000000000");

198:console.log("0000000000000000");
```

Description

toffee : `console.log` should be removed from the production deployment
zigzag : The code is not ues.

Recommendation

toffee : `console.log` should be removed from the production deployment
zigzag : Consider removing the redundant code.

Client Response

Fixed.we have removed the consoles.

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.