Security Assessment

ClearDAO Staking

July 27th, 2022

Secure3

secure3.io

Summary

ClearDAO's staking protocol is a novel financial protocol that allows users to receive rewards on deposited token and getting NFT as proof of staking. The protocol also has its platform token Clear Token (CLH) for users.

This report has been prepared for ClearDAO to identify issues and vulnerabilities in the smart contract source code of the ClearDAO project. A comprehensive examination with Static Analysis and Manual Review techniques has been performed.

The 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 scanner to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and 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: Informational, Low, Medium, Critical. For each of the findings we have provided recommendation of a fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	ClearDAO
Platform & Language	Ethereum, Solidity
Codebase	https://github.com/DerivStudio/staking-contract audit commit - cc2ea4c1d2eb2ca7878b344cb12db1524874df12 final commit - 7de4b9c3c710e28c318b2b431f527ca5fc15bd4c
Audit Methodology	• Business Logic Understanding and Review • Privileged Roles Review • Static Analysis • Code Review

Business Logic Review Summary

Total Number of Features	Caution	Information	Verified
5	0	1	4

Privileged Role Review Summary

Total Number of Privileged Roles	Caution	Information	Verified
2	0	0	2

Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowleged	Fixed	Mitigated
Critical	2	0	0	2	0
Medium	0	0	0	0	0
Low	3	0	0	3	0
Informational	2	0	1	1	0

Audit Scope

File	Commit Hash
contracts/CProxy.sol	cc2ea4c1d2eb2ca7878b344cb12db1524874df12
contracts/CProxyAdmin.sol	cc2ea4c1d2eb2ca7878b344cb12db1524874df12
contracts/Note.sol	cc2ea4c1d2eb2ca7878b344cb12db1524874df12
contracts/Staking.sol	cc2ea4c1d2eb2ca7878b344cb12db1524874df12
contracts/interface/INote.sol	cc2ea4c1d2eb2ca7878b344cb12db1524874df12

Business Logic Review

In this section, we asked project team to provide a list of business features of their contracts, our team verified each feature one by one and provided the verification results below.

How to read the table

1. **Left column is from project team**, describing their business intent
2. **Right column is from auditing team**, verifying if the code implementation meets the claimed business intent

Business Feature Claimed	Business Feature Audit Result
NFT ERC721 - Note is a ERC721 Token	◉ **Auditor Evaluation:** **Verified** ◉ **Code Reference:** contracts/Note.sol:7 ◉ **Detail:** The Note token is ERC721 token which can only be minted by the `miner` address. It's used as staking certificate.
Staking - User can stake tokens	◉ **Auditor Evaluation:** **Verified,** ◉ **Code Reference:** contracts/Staking.sol:88,119 ◉ **Detail:** The `withdraw()` and `withdrawBatch()` function can unstake all or part of tokens.
Staking - User can restake or split collateral	◉ **Auditor Evaluation:** **Verified,** ◉ **Code Reference:** contracts/Staking.sol:98,130 ◉ **Detail:** The `reStake()` and `reStakeBatch()` function can restake or split collateral.
Staking - User can choose different lock time and earn rewards from staking.	◉ **Auditor Evaluation:** **Informational** ◉ **Code Reference:** contracts/Staking.sol ◉ **Detail:** The lock time and reward calculation logic is off-chain, hence we cannot verify these use cases from the smart contracts along.

Privilege Role Review

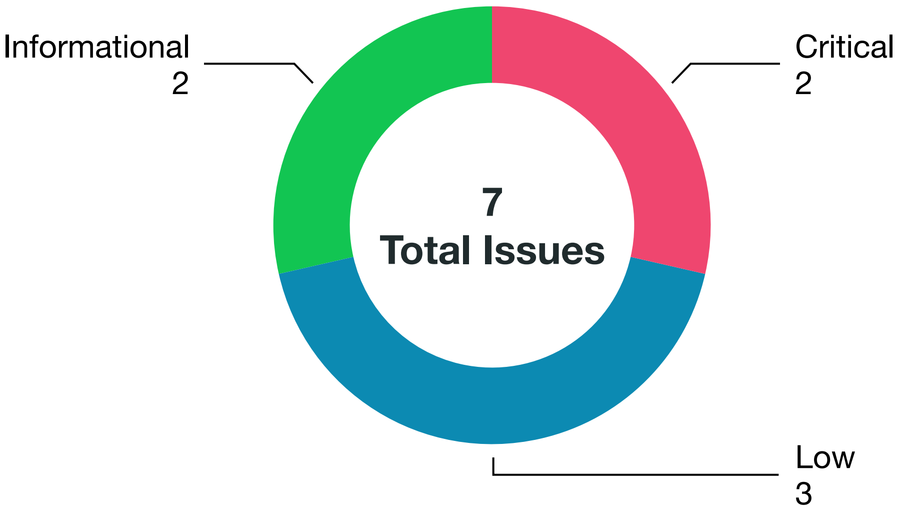
In this section, we reviewed all the privileged roles in the contracts. We listed all the findings in the following table.

How to read the table

1. **Left column:** privileged role name
2. **Middle column:** privileged permission of the role
3. **Right column:** verified code implementation and roles permission by auditing team

Contract Role	Privileged Functionalities	Audit Review
Miner Address	◉ mint ◉ burn	◉ **Auditor Evaluation:** **Verified,** ◉ **Code Reference:** contracts/**Note.sol** ◉ **Detail:** critical functionalities can only be called by `miner` address
Staking owner address	◉ setSigner	◉ **Auditor Evaluation:** **Verified** ◉ **Code Reference:** contracts/**Staking.sol** ◉ **Detail:** critical functionalities can only be called by contract owner

Code Assessment Findings



ID	Name	Category	Severity	Status
CLD-1	**Solidity compiler version is not consistent across the project**	Language Specific	Low	Fixed
CLD-2	`Staking` **should use upgradeable contract libraries**	Logical	Critical	Fixed
CLD-3	`Staking` **events' parameter is not indexed**	Code Style	Informational	Acknowledged
CLD-4	`Staking::stake()` **always reverts**	Logical	Critical	Fixed
CLD-5	`Staking::setSigner()` **does not validate** `_signer`	Logical	Low	Fixed
CLD-6	`Staking::_subBalanceBatch()` **does not validate** `ids` **and** `profits`	Logical	Low	Fixed
CLD-7	`Staking::_subBalanceBatch()` **variable typo**	Code Style	Informational	Fixed

CLD-1: Solidity compiler version is not consistent across the project

Category	Severity	Code Reference	Status
Language Specific	Low	All contracts	Fixed

Code

```
pragma solidity 0.8.9;

pragma solidity ^0.8.0;
```

Description

There are `0.8.9` and `^0.8.0` solidity versions used in the contracts and the compiler version is floating. Having non fixed compiler version is not the best practice.

Recommendation

Fix the compiler version to `0.8.9` or a version preferred.

Client Response

Compiler version fixed.

CLD-2: `Staking` should use upgradeable contract libraries

Category	Severity	Code Reference	Status
Logical	Critical	contracts/Staking.sol:12	Fixed

Code

```
12: contract Staking is Ownable, ReentrancyGuard, IERC777Recipient, Initializable {
```

Description

Staking contract is using proxy pattern, so it uses `initialize` function to set the initial states instead of constructor. `Ownable` and `ReentrancyGuard` parent contracts do not work here because all the internal states are set in the constructor, causing the `onlyOwner` and `nonReentrant` checks to fail.

Recommendation

Use `ReentrancyGuardUpgradeable` and `OwnableUpgradeable` in the `@openzeppelin/contracts-upgradeable` library and call `__Ownable_init()` in `__ReentrancyGuard_init()` the `initialize()` function. More details please refer to below code links:

- https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/OwnableUpgradeable.sol
- https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/security/ReentrancyGuardUpgradeable.sol

Client Response

Fixed. Client modified the contracts to use the `contracts-upgradeable` library.

CLD-3: `Staking` events' parameter is not indexed

Category	Severity	Code Reference	Status
Code Style	**Informational**	contracts/Staking.sol:22-47	Acknowledged

Code

```
22:     event Stake(
23:         address user,
24:         uint256 amount,
25:         uint256 id,
26:         uint256 productId,
27:         bool autoReinvestment
28:     );
29:     event Withdraw(address user, uint256 id, uint256 amount);
30:     event ReStake(
31:         address user,
32:         uint256 id,
33:         uint256 newId,
34:         uint256 amount,
35:         uint256 productId,
36:         bool autoReinvestment
37:     );
38:     event WithdrawBatch(address user, uint256[] id, uint256 amount);
39:     event ReStakeBatch(
40:         address user,
41:         uint256[] id,
42:         uint256 newId,
43:         uint256 amount,
44:         uint256 productId,
45:         bool autoReinvestment
46:     );
47:     event SetSigner(address signer);
```

Description

The indexed parameters for logged events will allow searching events using the indexed parameters as filters. If not indexed, it would be difficult to search certain events.

Recommendation

Add `indexed` keyword to some key fields such as `user` and `id` so that the events can be easily filtered.

Client Response

Acknowledged. All events are monitored and logged by the backend to save gas cost.

CLD-4: `Staking::stake()` always reverts

Category	Severity	Code Reference	Status
Logical	Critical	contracts/Staking.sol:152-158	Fixed

Code

```
152:     function stake(
153:         uint256 amount,
154:         uint256 productId,
155:         bool autoReinvestment
156:     ) external {
157:         IERC20(CLH).safeTransferFrom(msg.sender, address(this), amount);
158:         _stake(msg.sender, amount, productId, autoReinvestment);
159:     }

175:         require(msg.sender == CLH, "Staking:token error");
176:         (uint256 _productId, bool autoReinvestment) = abi.decode(
177:             userData,
178:             (uint256, bool)
179:         );
```

Description

`CLH` is ERC777 token, and `safeTransferFrom` function will call `Staking::tokensReceived()` hook function with an empty userData. While `tokensReceived` function trying to decode empty `userData` as `(uint256, bool)` format data, it will fail and revert the transaction. Below shows more details about the call stack.

```
Staking::stake()
IERC20(CLH).safeTransferFrom(msg.sender, address(this), amount);
------------------------
SafeERC20::safeTransferFrom() SafeERC20::transferFrom()
        abi.encodeWithSelector(token.transferFrom.selector, from, to, value)
            _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from,
to, value));
------------------------
ERC777::transferFrom()
_send(holder, recipient, amount, "", "", false);
        _callTokensReceived()
            IERC777Recipient(implementer).tokensReceived(operator, from, to, amount, userData,
operatorData);
------------------------
Staking::tokensReceived()
    abi.decode(userData,(uint256, bool));
               ^ is ""
```

Recommendation

Users can stake by directly calling `ClearToken(CLH)` contract's inherited `ERC777::send(address recipient, uint256 amount, bytes memory data)` function and pass `_productId` and `autoReinvestment` as the abi encoded data to Staking contract, hence this `stake` function may be unnecessary.

Client Response

Fixed by adding `ignoreTokensReceived` modifier and adding `_ignoreReceive` state check in the `tokensReceived` function.

CLD-5: `Staking::setSigner()` does not validate `_signer`

Category	Severity	Code Reference	Status
Logical Issue	Informational	contracts/Staking.sol:161-165	Fixed

Code

```
161:     function setSigner(address _signer) external onlyOwner {
162:         require(_signer != signer, "Staking:repeat");
163:         signer = _signer;
164:         emit SetSigner(_signer);
165:     }
```

Description

The `_signer` can be zero address, causing to lose control of the contract.

Recommendation

Add a require statement to make sure `_signer != address(0)`

Client Response

Fixed. Added require statement to ensure that the `_signer` is not `address(0)`.

CLD-6: `Staking::_subBalanceBatch()` does not validate `ids` and `profits`

Category	Severity	Code Reference	Status
Logical Issue	Low	Contracts/Staking.sol:196-201	Fixed

Code

```
196:    function _subBalanceBatch(
197:        uint256[] memory ids,
198:        uint256[] memory profits,
199:        uint256 amount,
200:        bytes calldata signature
201:    ) internal nonReentrant {
202:        uint256[] memory operatioIds = new uint256[](ids.length);
```

Description

The `ids` and `profits` parameter can have different length, in that case the loop would have error.

Recommendation

Add a require statement to make sure `ids.length == profits.length`

Client Response

Fixed. Added check to make sure `ids.length == profits.length`.

CLD-7: `Staking::_subBalanceBatch()` variable typo

Category	Severity	Code Reference	Status
Code Style	Informational	Contracts/Staking.sol:202	Fixed

Code

```
202:        uint256[] memory operatioIds = new uint256[](ids.length);
```

Description

The `operatioIds` variable contains a typo, should be `operationIds`.

Recommendation

Fix the typo.

Client Response

Fixed.

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.