Security Audit Report for DAO4ART Contracts

Date: December 28, 2022

Version: 1.0

Contact: contact@blocksec.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.12.28 16:46:38 +08'00'

Contents

Report Manifest

Item	Description
Client	DAO4ART
Target	DAO4ART Contracts

Version History

Version	Date	Description
1.0	December 28, 2022	First Release

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 5 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repo of DAO4ART Contracts [1] of the DAO4ART project. DAO4ART is designed as a DAO to produce more DAOs that respectively or collectively create art works. Driven by an incentive-compatible model with automatically generated ERC20 tokens, i.e. DAO Token, both DAO Creators and Canvas Creators are rewarded immediately and long-term to create a better art collection with aligned interests. Compared to traditional NFT collection production, D4A DAOs enable a collaborative workflow that involves and incentivises all players from the get-go. The goal is to build bootstrapping DAO for all kinds of art DAOs to grow.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
DAO4ART Contracts	`Version 1`	`862604b7bfa73d7368bcd8e87026570b6407166a`
	`Version 2`	`3927a48487937ea7b89ec3df64449bc0afe6a6f0`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

[1] https://github.com/simple4016/d4a-audit

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we find **three** potential issues. Besides, we have **four** recommendations and **one** note.

- Medium Risk: 3
- Recommendation: 4
- Note: 1

ID	Severity	Description	Category	Status
1	Medium	Potential inconsistency due to the mixed usage of local and global configurations	Software Security	Fixed
2	Medium	Incorrect reward redeeming procedure	DeFi Security	Fixed
3	Medium	Potential DoS due to the lack of validity check for the NFT token URI	NFT Security	Acknowledged
4	-	Unify the implementation of contract creation	Recommendation	Acknowledged
5	-	Avoid permanently losing `DEFAULT_ADMIN_ROLE`	Recommendation	Fixed
6	-	Add sanity checks for the global configurations	Recommendation	Acknowledged
7	-	Remove unnecessary checks and calculations	Recommendation	Acknowledged
8	-	PRB's `start_block` and `period_block` should not be changed	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 Potential inconsistency due to the mixed usage of local and global configurations

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description There exists a mixed use of local and global configurations in the DAO4ART project, which may lead to unexpected behaviors. Specifically, in DAO4ART, all canvases are registered in the projects, which may have different price gradients. The D4A protocol would create an ERC20 token to reward the project owner and the canvas creators. When creating a project, the total supply of the reward token needs to be set, as shown in the code snippet of the `createProject` function.

```
37    function createProject(mapping(bytes32=>project_info) storage all_projects,
38                           ID4ASetting _settings,
39                          uint256 _start_prb,
40                           uint256 _mintable_rounds,
41                           uint256 _floor_price_rank,
42                           uint256 _max_nft_rank,
43                           uint96 _royalty_fee,
44                           uint256 _project_index,
45                           string memory _project_uri) public returns(bytes32 project_id){
46
          ...
108       pi.erc20_total_supply = _settings.erc20_total_supply();
```

```
109         for(uint i = 0; i < _settings.floor_prices_length(); i++){
110             pi.floor_prices.push(_settings.floor_prices(i));
111         }
112         require(pi.floor_price_rank < pi.floor_prices.length, "invalid floor price rank");
113
114         pi.exist = true;
115         emit NewProject(project_id, _project_uri, pool, pi.erc20_token, pi.erc721_token,
                _royalty_fee);
116         }
```

Listing 2.1: D4AProject.sol

Notice that the second argument named `_settings` points to a global configuration contract, which suggests that the local configuration of the project is set to the current global configuration. In other words, the total reward token supply should be fixed, despite the change of the global configuration. However, when claiming the reward, the protocol still fetches the total supply parameter from the current global configuration, as shown in the following code snippet (see line 195 and line 197).

```
166     function claimProjectReward(mapping(bytes32=>reward_info) storage all_rewards,
167                             ID4ASetting _settings,
168                             bytes32 _project_id,
169                             address erc20_token,
170                             uint256 _start_round,
171                             uint256 _total_rounds) internal
172     returns(uint256){
        ...
193
194     uint256 d4a_amount =
195       _settings.erc20_total_supply() * _settings.d4a_erc20_ratio() * n /(_settings.ratio_base() *
              _total_rounds);
196     uint256 project_amount =
197       _settings.erc20_total_supply() * _settings.project_erc20_ratio() * n /(_settings.ratio_base
              () *_total_rounds);
        ...
206     return project_amount;
207   }
```

Listing 2.2: D4AReward.sol

Impact Mixed usage of local and global configurations may bring unexpected behaviors.

Suggestion Use project's local configuration after the creation to keep consistency.

2.2 DeFi Security

2.2.1 Incorrect reward redeeming procedure

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The project rewards are distributed by issuing and claiming reward tokens, and the project owners and canvas creators can redeem their reward tokens for ETH. However, the incorrect redeeming procedure can make the reward distribution unbalanced.

Specifically, the platform divides a project's lifetime into rounds. Every time the users mint NFT tokens, fees must be paid to the canvas creator. The platform would charge fee from that payment to a reward pool and record the amount for the current round. The project has a fixed total round amount and a fixed reward token total supply. The reward tokens are distributed proportionally to each round. The distribution is done by the `issueTokenToCurrentRound` function, which would issue reward tokens to rounds before the current round.

```
28  function issueTokenToCurrentRound(mapping(bytes32=>reward_info) storage all_rewards,
29                                ID4ASetting _settings,
30                                bytes32 _project_id,
31                                address erc20_token,
32                                uint256 _start_round,
33                                uint256 total_rounds) internal returns(uint256){
34  ID4APRB prb = _settings.PRB();
35  uint256 cur_round = prb.currentRound();
36  if(cur_round <= _start_round){
37    return 0;
38  }
39  reward_info storage ri = all_rewards[_project_id];
40
41  uint256 n = ri.issued_rounds;
42  if(n >= total_rounds){
43    return 0;
44  }
45  {
46    uint i = 0;
47    for(i = ri.to_issue_round_index ; i < ri.active_rounds.length; i++){
48      if(ri.active_rounds[i] == cur_round){
49        break;
50      }
51      if(all_rewards[_project_id].round_2_total_amount[ri.active_rounds[i]] != 0){
52        n = n + 1;
53        all_rewards[_project_id].final_issued_round_index = i;
54        all_rewards[_project_id].to_issue_round_index = i + 1;
55        if(n == total_rounds){
56          break;
57        }
58      }
59    }
60  }
61
62
63  uint256 amount = (n - all_rewards[_project_id].issued_rounds)*_settings.erc20_total_supply()/
        total_rounds;
64
65  ID4AMintableERC20(erc20_token).mint(address(this), amount);
66  all_rewards[_project_id].issued_rounds = n;
67  return amount;
68 }
```

Listing 2.3: D4AReward.sol

After that, the canvas creators can claim their reward tokens according to the canvas's contributions to each round. Then they can redeem the reward tokens into ETH.

```
153   function claimCanvasRewardWithETH(
154                           ID4ASetting _settings,
155                           bytes32 _project_id,
156                           bytes32 _canvas_id,
157                           address erc20_token,
158                           uint256 erc20_amount,
159                           address _fee_pool,
160                           mapping (bytes32 => mapping (uint256 => uint256)) storage
                                  round_2_total_eth) internal returns(uint256){
161     if (erc20_amount == 0) return 0;
162     address _owner = _settings.owner_proxy().ownerOf(_canvas_id);
163     uint256 to_send = sendETH(_settings, erc20_token, _fee_pool, _project_id, _owner, _owner,
            erc20_amount, round_2_total_eth);
164     return to_send;
165   }
```

Listing 2.4: D4AReward.sol

```
224   function sendETH(ID4ASetting _settings,
225                 address erc20_token, address fee_pool, bytes32 project_id, address _owner,
                        address _to, uint256 erc20_amount,
226                 mapping (bytes32 => mapping (uint256 => uint256)) storage round_2_total_eth)
227   internal returns(uint256){
228     ID4AMintableERC20(erc20_token).burn(_owner, erc20_amount);
229     ID4AMintableERC20(erc20_token).mint(fee_pool, erc20_amount);
230
231     ID4APRB prb = _settings.PRB();
232     uint256 cur_round = prb.currentRound();
233
234     uint256 circulate_erc20 = IERC20Upgradeable(erc20_token).totalSupply() + erc20_amount -
            IERC20Upgradeable(erc20_token).balanceOf(fee_pool);
235     uint256 avaliable_eth = fee_pool.balance - round_2_total_eth[project_id][cur_round];
236     uint256 to_send = erc20_amount * avaliable_eth / circulate_erc20;
237     if(to_send != 0){
238       D4AFeePool(payable(fee_pool)).transfer(address(0x0), payable(_to), to_send);
239     }
240     emit D4AExchangeERC20ToETH(_owner, _to, erc20_amount, to_send);
241     return to_send;
242   }
```

Listing 2.5: D4AReward.sol

The `sendETH` function is used for converting the reward tokens to ETH. It first calculates the ratio of redeeming amount and the reward token's total circulation. Then, it deducts the ETH amount of the current round from the total fee pool. Finally, it distributes ETH based on the ratio and the adjusted fee pool.

However, the adjusted fee pool contains not only the reward for issued rounds but also for the ended rounds that have not been issued. As a result, the reward distribution can be unbalanced if there is a way

to redeem reward tokens without triggering the `issueTokenToCurrentRound` function. Unfortunately, there exists such a code logic in the `exchangeERC20ToETH` function of the `D4AProtocol` contract.

```
198 function exchangeERC20ToETH(bytes32 _project_id, uint256 amount, address _to) public nonReentrant
        returns(uint256){
199     require(!settings.d4a_pause(), "D4A Paused");
200     require(!settings.pause_status(_project_id), "Project Paused");
201
202     D4AProject.project_info storage pi = all_projects[_project_id];
203     return D4AReward.ToETH(settings, pi.erc20_token, pi.fee_pool, _project_id, msg.sender, _to,
            amount, round_2_total_eth);
204 }
```

Listing 2.6: D4AProtocol.sol

```
244 function ToETH(ID4ASetting _settings, address erc20_token, address fee_pool,
245               bytes32 project_id, address _owner, address _to, uint256 amount,
246               mapping (bytes32 => mapping (uint256 => uint256)) storage round_2_total_eth)
247   internal returns(uint256){
248   return sendETH(_settings, erc20_token, fee_pool, project_id, _owner, _to, amount,
          round_2_total_eth);
249 }
```

Listing 2.7: D4AReward.sol

Impact Unbalanced reward distribution will bring losses to canvas creators.

Suggestion Invoke the `issueTokenToCurrentRound` function in the `exchangeERC20ToETH` function.

2.3 NFT Security

2.3.1 Potential DoS due to the lack of validity check for the NFT token URI

Severity Medium

Status Acknowledged

Introduced by `Version 1`

Description In the normal case, a user can choose the work and mint the NFT in the corresponding canvas. Specifically, the `mintNFT` function allows the caller to specify the token URI. However, a malicious user could also insert evil contents into a canvas' collection. By doing so, it is possible to launch the DoS attack to break the whole canvas.

```
 89   function mintNFT(bytes32 _canvas_id, string memory _token_uri) public payable nonReentrant
          returns(uint256 token_id){
      ...
137   token_id = ID4AERC721(all_projects[proj_id].erc721_token).mintItem(msg.sender, _token_uri);
138   all_projects[proj_id].nft_supply++;
139   all_canvases[_canvas_id].nft_tokens.push(token_id);
140   all_canvases[_canvas_id].nft_token_number++;
141   tokenid_2_canvas[keccak256(abi.encodePacked(proj_id, token_id))] = _canvas_id;
142   emit D4AMintNFT(proj_id, _canvas_id, token_id, _token_uri, price);
143 }
```

Listing 2.8: D4AProtocol.sol

Impact Potential DoS if the URI is illegal for presenting.

Suggestion Check the validity of the token URI.

2.4 Additional Recommendation

2.4.1 Unify the implementation of contract creation

Status Acknowledged

Introduced by `Version 1`

Description There are several factory contracts for creating contracts. The `D4AFeePoolFactory` contract uses `TransparentUpgradeableProxy`, while the others uses the `clone` method of OpenZeppelin's `Clones` library.

```
65    function createD4AFeePool(string memory _name) public returns(address pool){
66        bytes memory data = abi.encodeWithSignature("initialize(string)", _name);
67        TransparentUpgradeableProxy proxy = new TransparentUpgradeableProxy(address(impl),
              proxy_admin, data);
68        D4AFeePool(payable(address(proxy))).changeAdmin(msg.sender);
69        emit NewD4AFeePool(address(proxy), proxy_admin);
70        return address(proxy);
71      }
```

Listing 2.9: D4AFeePool.sol

```
49    function createD4AERC20(string memory _name, string memory _symbol, address _minter) public
          returns(address){
50        address t = address(impl).clone();
51        D4AERC20(t).initialize(_name, _symbol, _minter);
52        D4AERC20(t).changeAdmin(msg.sender);
53        emit NewD4AERC20(t);
54        return t;
55      }
```

Listing 2.10: D4AERC20.sol

Impact N/A

Suggestion Unify the implementation of contract creation.

2.4.2 Avoid permanently losing `DEFAULT_ADMIN_ROLE`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In this project, all `changeAdmin` functions will first grant `DEFAULT_ADMIN_ROLE` to `new_admin`, and then revoke the same role from `msg.sender`. However, if `new_admin` is the same as `msg.sender`, the function would grant and revoke `DEFAULT_ADMIN_ROLE` from the same address. In other words, `DEFAULT_ADMIN_ROLE` would be permanently lost.

```
33  function changeAdmin(address new_admin) public onlyRole(DEFAULT_ADMIN_ROLE){
34    _grantRole(DEFAULT_ADMIN_ROLE, new_admin);
35    _revokeRole(DEFAULT_ADMIN_ROLE, msg.sender);
36  }
```

Listing 2.11: D4AERC20.sol

Impact N/A

Suggestion Add the sanity check accordingly.

2.4.3 Add sanity checks for the global configurations

Status Acknowledged

Introduced by Version 1

Description The validity of some global configuration fields are not checked when they are set. Specifically,

- Address parameters cannot be zero.
- The distribution ratio related parameters shall be within the valid range.
- `floor_prices` needs to be sorted in the correct order.

```
67  function changeAddress(address _prb, address _erc20_factory,
68                         address _erc721_factory,
69                         address _feepool_factory,
70                         address _owner_proxy) public onlyOwner{
71    PRB = ID4APRB(_prb);
72    erc20_factory = ID4AERC20Factory(_erc20_factory);
73    erc721_factory = ID4AERC721Factory(_erc721_factory);
74    feepool_factory = ID4AFeePoolFactory(_feepool_factory);
75    owner_proxy = ID4AOwnerProxy(_owner_proxy);
76    emit ChangeAddress(_prb, _erc20_factory, _erc721_factory, _feepool_factory, _owner_proxy);
77  }
```

Listing 2.12: D4ASetting.sol

```
49  function changeERC20Ratio(uint256 _d4a_ratio, uint256 _project_ratio, uint256 _canvas_ratio)
        public onlyOwner{
50    d4a_erc20_ratio = _d4a_ratio;
51    project_erc20_ratio = _project_ratio;
52    canvas_erc20_ratio = _canvas_ratio;
53    require(_d4a_ratio + _project_ratio + _canvas_ratio == ratio_base, "invalid ratio");
54
55    emit ChangeERC20Ratio(d4a_erc20_ratio, project_erc20_ratio, canvas_erc20_ratio);
56  }
```

Listing 2.13: D4ASetting.sol

```
86  function changeFloorPrices(uint256[] memory _prices) public onlyOwner{
87    delete floor_prices;
88    floor_prices = _prices;
89    emit ChangeFloorPrices(_prices);
```

```
90  }
```

Listing 2.14: D4ASetting.sol

Impact N/A

Suggestion Add the sanity checks accordingly.

2.4.4 Remove unnecessary checks and calculations

Status Acknowledged

Introduced by `Version 1`

Description There are some unnecessary checks and calculations that may cause extra gas consumption, as follows:

- Line 36 and 37 in the `createCanvas` function of the `D4ACanvas` contract.

```
20   function createCanvas(mapping(bytes32=>canvas_info) storage all_canvases,
21                         ID4ASetting _settings,
22                         address fee_pool,
23                         bytes32 _project_id,
24                         uint256 _project_start_prb,
25                         uint256 canvas_num,
26                         string memory _canvas_uri) public returns(bytes32){
27
28   {
29     ID4APRB prb = _settings.PRB();
30     uint256 cur_round = prb.currentRound();
31     require(cur_round >= _project_start_prb, "project not start yet");
32   }
33
34   {
35     uint256 minimal = _settings.create_canvas_fee();
36     require(minimal <= msg.value, "not enough ether to create canvas");
37     if(msg.value < minimal) revert D4AInsufficientEther(minimal);
```

Listing 2.15: D4ACanvas.sol

- Line 28 and 29 in `transferOwnership`.

```
27     function transferOwnership (bytes32 hash, address newOwner) external{
28       require(ownerMap[hash]!=address(0),"NaiveOwner: This hash doesn't exist");
29       require(ownerMap[hash]==msg.sender,"NaiveOwner: The caller is not the owner");
30       ownerMap[hash]=newOwner;
31     }
```

Listing 2.16: NaiveOwner.sol

- Line 52 and 53 in the `createProject` function, and line 72 and 73 in the `createCanvas` function of the `D4AProtocol` contract.

```
45   function createProject(uint256 _start_prb,
46                          uint256 _mintable_rounds,
47                          uint256 _floor_price_rank,
48                          uint256 _max_nft_rank,
```

```
49                      uint96 _royalty_fee,
50                      string memory _project_uri) public override payable nonReentrant
                            returns(bytes32 project_id){
51    require(!settings.d4a_pause(), "D4A Paused");
52    require(!uri_exists[keccak256(abi.encodePacked(_project_uri))], "project_uri already
          exist");
53    uri_exists[keccak256(abi.encodePacked(_project_uri))] = true;
54    project_num ++;
55    return all_projects.createProject(settings, _start_prb, _mintable_rounds,
          _floor_price_rank, _max_nft_rank, _royalty_fee, project_num, _project_uri);
56  }
```

Listing 2.17: D4AProtocol.sol

```
66    function createCanvas(bytes32 _project_id, string memory _canvas_uri) public payable
          nonReentrant
67    returns(bytes32 canvas_id){
68    require(!settings.d4a_pause(), "D4A Paused");
69    if(!all_projects[_project_id].exist) revert D4AProjectNotExist(_project_id);
70    require(!settings.pause_status(_project_id), "Project Paused");
71
72    require(!uri_exists[keccak256(abi.encodePacked(_canvas_uri))], "canvas_uri already exist
          ");
73    uri_exists[keccak256(abi.encodePacked(_canvas_uri))] = true;
74
75    canvas_id = all_canvases.createCanvas(settings,
76                                          all_projects[_project_id].fee_pool,
77                                          _project_id,
78                                          all_projects[_project_id].start_prb,
79                                          all_projects.getProjectCanvasCount(_project_id),
80                                          _canvas_uri);
81
82    all_projects[_project_id].canvases.push(canvas_id);
83    //creating canvas does not affect price
84    //all_prices.updateCanvasPrice(settings, _project_id, canvas_id, 0);
85  }
```

Listing 2.18: D4AProtocol.sol

Impact N/A

Suggestion Revise the code.

2.5 Note

2.5.1 PRB's `start_block` and `period_block` should not be changed

Description The `D4APRB` contract is used for counting rounds. The `currentRound` function calculates current round by dividing the difference between `block.number` and `start_block` by `period_block`.

```
30    function currentRound() public view returns(uint256){
31       return (block.number - start_block)/period_block;
32     }
```

Listing 2.19: D4APRB.sol

The contract provides interfaces for the owner to update `start_block` and `period_block`. However, for the active projects, these two variables should not be changed. Otherwise, the round calculation and reward distribution process might have unexpected behaviors.