Competitive Security Assessment

Decider NFT

Feb 22nd, 2023

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	Decider NFT
Platform & Language	Solidity
Codebase	• https://github.com/teamdesider/contract • audit commit - e66e8464062a9f91d138047482c675a24865e061 • final commit - 31ad8bd39b880d34425c348aa67ac232327fc757
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

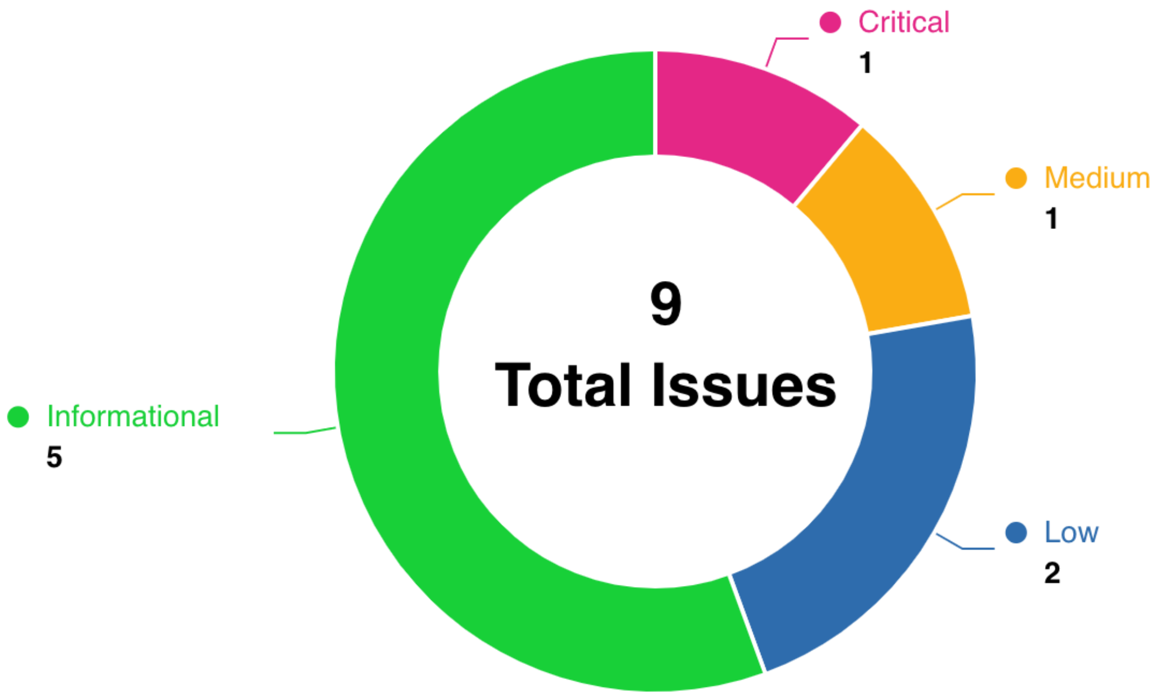
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	1	0	0	1	0	0
Medium	1	0	0	1	0	0
Low	2	0	0	2	0	0
Informational	5	0	1	4	0	0

Audit Scope

File	Commit Hash
./SpaceRender.sol	e66e8464062a9f91d138047482c675a24865e061
./DesiderNft.sol	e66e8464062a9f91d138047482c675a24865e061

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
DCD-1	Redundant `totalSupply()` calling in `_mintNft` function	Gas Optimization	Informational	Fixed	0xac
DCD-2	Redundant judgment code	Gas Optimization	Informational	Fixed	0xac
DCD-3	Reentrancy Risk in `SpaceRander` Contract `ogMint` Function	Reentrancy	Critical	Fixed	Hellobloc, Secure3
DCD-4	Use OpenZeppelin 4.x version contracts	Code Style	Informational	Acknowledged	Secure3

DCD-5	Use library functions to access `_tokenIdCounter`	Code Style	Informational	Fixed	Secure3
DCD-6	`DesiderNft::receiveFromL1ReMint` Redundant ERC721 hook function calls and event emission	Logical	Low	Fixed	Secure3
DCD-7	`SpaceRander.endTime` is defined but not used	Logical	Low	Fixed	Hellobloc, Secure3
DCD-8	`SpaceRander.wlMint` Mint amount not limited	Logical	Medium	Fixed	Secure3
DCD-9	`SpaceRender` gas optionmization by using `constant` values	Gas Optimization	Informational	Fixed	porotta, Secure3

DCD-1:Redundant `totalSupply()` calling in `_mintNft` function

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/SpaceRender.sol#L125-L135	Fixed	0xac

Code

```
125:    function _mintNft(uint256 tokenQuantity, address to) internal {
126:        for (uint256 i = 0; i < tokenQuantity; i++) {
127:            //
128:            uint256 mintIndex = totalSupply();
129:            if (totalSupply() < MAX_SUPPLY) {
130:                _tokenIdCounter._value += 1;
131:                initTokenId(mintIndex);
132:                _safeMint(to, mintIndex);
133:            }
134:        }
135:    }
```

Description

0xac : In `uint256 mintIndex = totalSupply();`, `mintIndex` is equal to the value of `totalSupply()`. There is no need to call the `totalSupply()` function again in `if (totalSupply() < MAX_SUPPLY)`.

```
function _mintNft(uint256 tokenQuantity, address to) internal {
    for (uint256 i = 0; i < tokenQuantity; i++) {
        //
        uint256 mintIndex = totalSupply();
        if (totalSupply() < MAX_SUPPLY) {
            _tokenIdCounter._value += 1;
            initTokenId(mintIndex);
            _safeMint(to, mintIndex);
        }
    }
}
```

Recommendation

0xac : Consider below fix in the `SpaceRender._mintNft()` function for saving gas.

```
function _mintNft(uint256 tokenQuantity, address to) internal {
    for (uint256 i = 0; i < tokenQuantity; i++) {
        //
        uint256 mintIndex = totalSupply();
        if (mintIndex < MAX_SUPPLY) {
            _tokenIdCounter._value += 1;
            initTokenId(mintIndex);
            _safeMint(to, mintIndex);
        }
    }
}
```

Client Response

We change the second `totalsupply()` to `mintIndex`

DCD-2:Redundant judgment code

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/SpaceRender.sol#L91 • code/SpaceRender.sol#L116	Fixed	0xac

Code

```
91:         require(tokenQuantity <= maxBalance, "Can only mint {maxBalance} tokens at a time");

116:         require(tokenQuantity <= maxBalance, "Can only mint {maxBalance} tokens at a time");
```

Description

0xac : The value of `balanceOf(msg.sender)` is not less than 0, so we can ensure that `tokenQuantity <= maxBalance` by the fist `require` code. The second `require` code is redundant.

```
require(
    balanceOf(msg.sender) + tokenQuantity <= maxBalance,
    "Sale would exceed max balance"
);
require(tokenQuantity <= maxBalance, "Can only mint {maxBalance} tokens at a time");
```

Recommendation

0xac : Removing the redundant code to saving gas.

```
require(
    balanceOf(msg.sender) + tokenQuantity <= maxBalance,
    "Sale would exceed max balance"
);
// require(tokenQuantity <= maxBalance, "Can only mint {maxBalance} tokens at a time");
```

Client Response

We remove the redundant judgment code.

DCD-3:Reentrancy Risk in `SpaceRander` Contract `ogMint` Function

Category	Severity	Code Reference	Status	Contributor
Reentrancy	Critical	• code/SpaceRender.sol#L57-L79	Fixed	Hellobloc, Secure3

Code

```
57:    function ogMint() external {
58:        require(block.timestamp > ogStart, "mint not start");
59:        uint256 num = _ogmint[msg.sender];
60:        require(
61:            num == 0,
62:            "one og can only use once"
63:        );
64:        require(
65:            totalSupply() + 1 <= MAX_SUPPLY,
66:            "Sale would exceed max supply"
67:        );
68:        require(
69:            balanceOf(msg.sender) + 1 <= maxBalance,
70:            "Sale would exceed max balance"
71:        );
72:        uint256 ogcheck =
DesiderOG(0xAfa3CA7A79091CEbb035f490a51C6bfD45Cb4FC8).balanceOf(msg.sender);
73:        require(ogcheck >= 1,
74:            "only og can use this mint"
75:        );
76:
77:        _mintNft(1, msg.sender);
78:        _ogmint[msg.sender] = 1;
79:    }
```

Description

Hellobloc : `ogmint` does not follow the `CEI` principle that `_ogmint` is marked after an external call, and the presence of `_checkOnERC721Received` in the `_safemint` function in the `ERC721` will lead to potential `reentrancy` problems.

```
  function ogMint() external {
          ...
          uint256 num = _ogmint[msg.sender];
          require(
              num == 0,
              "one og can only use once"
          );
          ...
          _mintNft(1, msg.sender);
          _ogmint[msg.sender] = 1;
      }
```

Eventually the problem may lead to `og` users being able to mint tokens without `only use once` limit.
Secure3 : The `SpaceRander:ogmint()` function has reentrancy risk because the internal state `_ogmint` is only updated after the external call `_mintNft()` the call stack is `_mintNft() -> ERC721._safeMint() -> _checkOnERC721Received() -> IERC721Receiver(to).onERC721Received(_msgSender(), ...)`.

Recommendation

Hellobloc : Use the Checks-Effects-Interactions best practice and make all state changes before calling external contracts. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.
Secure3 : make `_ogmint[msg.sender] = 1;` update before `_mintNft(1, msg.sender);` and follow the Checks-Effects-Interactions best practice. Also, consider using function modifiers such as `nonReentrant`.

Client Response

We make `_ogmint[msg.sender] = 1;` update before `_mintNft(1, msg.sender);`

DCD-4:Use OpenZeppelin 4.x version contracts

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/DesiderNft.sol#L1-L3 • code/SpaceRender.sol#L1-L3	Acknowledged	Secure3

Code

```
1:// Contract based on https://docs.openzeppelin.com/contracts/3.x/erc721
2:// SPDX-License-Identifier: MIT
3:pragma solidity ^0.8.13;

1:// Contract based on https://docs.openzeppelin.com/contracts/3.x/erc721
2:// SPDX-License-Identifier: MIT
3:pragma solidity ^0.8.13;
```

Description

Secure3 : Based on the comment `Contract based on https://docs.openzeppelin.com/contracts/3.x/erc721` the version is `3.x` and solidity version is `pragma solidity ^0.8.13;`. However OpenZeppelin 3.x does not support solidity 0.8.

Recommendation

Secure3 : Use OpenZeppelin 4.x version contracts and correct code comments.

Client Response

We use OpenZeppelin 4.x version contracts and delete the wrong code comments

DCD-5:Use library functions to access `_tokenIdCounter`

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	• **code/SpaceRender.sol#L54** • **code/SpaceRender.sol#L130**	**Fixed**	**Secure3**

Code

```
54:        return _tokenIdCounter._value;

130:                _tokenIdCounter._value += 1;
```

Description

Secure3 : According to Counters.sol, `Counter._value` should never be directly accessed by users: interactions must be restricted to the library's internal functions.

Recommendation

Secure3 : Change

```
return _tokenIdCounter._value;
```

to

```
return _tokenIdCounter.current();
```

Change

```
_tokenIdCounter._value += 1;
```

to

```
_tokenIdCounter.increment();
```

Client Response

We change `_tokenIdCounter._value` to `_tokenIdCounter.current();` and change `_tokenIdCounter._value += 1;` to `_tokenIdCounter.increment();`

DCD-6: `DesiderNft::receiveFromL1ReMint` Redundant ERC721 hook function calls and event emission

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/DesiderNft.sol#L105** • **code/DesiderNft.sol#L115** • **code/DesiderNft.sol#L117**	**Fixed**	**Secure3**

Code

```
105:        _beforeTokenTransfer(address(0), to, tokenId);

115:        emit Transfer(address(0), to, tokenId);

117:        _afterTokenTransfer(address(0), to, tokenId);
```

Description

Secure3 : `_beforeTokenTransfer` and `_afterTokenTransfer` are already called in the `_safeMint` function(reference: OpenZeppelin ERC721), so it is incorrect to call them again in the `receiveFromL1ReMint` function.

Similarly, `Transfer event` is already emitted in the _mint function.

The redudant call of `_beforeTokenTransfer()` and `_afterTokenTransfer()` can lead to problem if the `to` has a state change hook before and after functons

Recommendation

Secure3 : Delete duplicate hook function calls and event emission in the `receiveFromL1ReMint` function.

Client Response

We delete duplicate hook function calls and event emission in the receiveFromL1ReMint function

DCD-7: `SpaceRander.endTime` is defined but not used

Category	Severity	Code Reference	Status	Contributor
Logical	Low	code/SpaceRender.sol#L31 • code/SpaceRender.sol#L31	Fixed	Hellobloc, Secure3

Code

```
31:    uint256 private endTime = 1775145329;

31:    uint256 private endTime = 1775145329;
```

Description

Hellobloc : `endTime` is not used in the code and may cause unnecessary GAS consumption.

```
uint256 private endTime = 1775145329;
```

Secure3 : The variable `endTime` is defined in the `SpaceRander` contract but never used.

Recommendation

Hellobloc : We recommend removing the useless code

Secure3 : Add `endTime` limit in `ogMint`, `wlMint` and `pubMint` function.

```
require("block.timestamp <= endTime", "mint is over");
```

Client Response

We delete the variable `endTime`

DCD-8: `SpaceRander.wlMint` Mint amount not limited

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	• **code/SpaceRender.sol#L108-L111**	**Fixed**	**Secure3**

Code

```
        require(
            balanceOf(msg.sender) + tokenQuantity <= maxBalance,
            "Sale would exceed max balance"
        );
```

Description

Secure3 : Whitelisted users can mint any amount of NFTs. It only restricts users balance of NFTs to be less than `maxBalance`. Whitelisted users can transfer NFTs to other accounts and then mint again.

Recommendation

Secure3 : Add a storage variable storing mint amount of whitelisted users.

```
mapping(address => uint256) private _wlmint;
//...
require(
    _wlmint[msg.sender] + tokenQuantity <= maxBalance,
    "Sale would exceed max balance"
);
_wlmint[msg.sender] = _wlmint[msg.sender] +tokenQuantity ;
```

Client Response

We add variable _wlmint and add check limit in function wlMint.

DCD-9: `SpaceRender` gas optionmization by using `constant` values

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/SpaceRender.sol#L19-L23 • code/SpaceRender.sol#L20-L22 • code/SpaceRender.sol#L28-L31	Fixed	porotta, Secure3

Code

```
19:    uint256 public constant MAX_SUPPLY = 5500;
20:    uint256 public wlPrice = 0.001 ether;
21:    uint256 public pubPrice = 0.002 ether;
22:    uint256 public maxBalance = 3;
23:

20:    uint256 public wlPrice = 0.001 ether;
21:    uint256 public pubPrice = 0.002 ether;
22:    uint256 public maxBalance = 3;

28:    uint256 private ogStart = 1675145329;
29:    uint256 private wlStart = 1675145329;
30:    uint256 private pubStart = 1675145329;
31:    uint256 private endTime = 1775145329;
```

Description

porotta : Assuming the wlPrice, pubPrice and maxBalance will stay the same it would be a good practice to add the constant keyword for the above mentioned vars

Secure3 : Using the `constant` keyword for variables that do not change helps to save on gas used. In the SpaceRender contract, `wlPrice`, `pubPrice`, `maxBalance`, `ogStart`, `wlStart`, `pubStart`, `endTime` can be `constact`.

Recommendation

porotta : Add constant key word.

```
    uint256 public constant wlPrice = 0.001 ether;
    uint256 public constant pubPrice = 0.002 ether;
    uint256 public constant maxBalance = 3;
```

Secure3 : Use `constant` keyword for `wlPrice`, `pubPrice`, `maxBalance`, `ogStart`, `wlStart`, `pubStart` and `endTime`.

Client Response

We add the constant keyword to the varibles(MAX_SUPPLY, wlPrice , pubPrice, maxBalance, pubStart);

Disclaimer