Competitive Security Assessment

KreationJP

Apr 26th, 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.

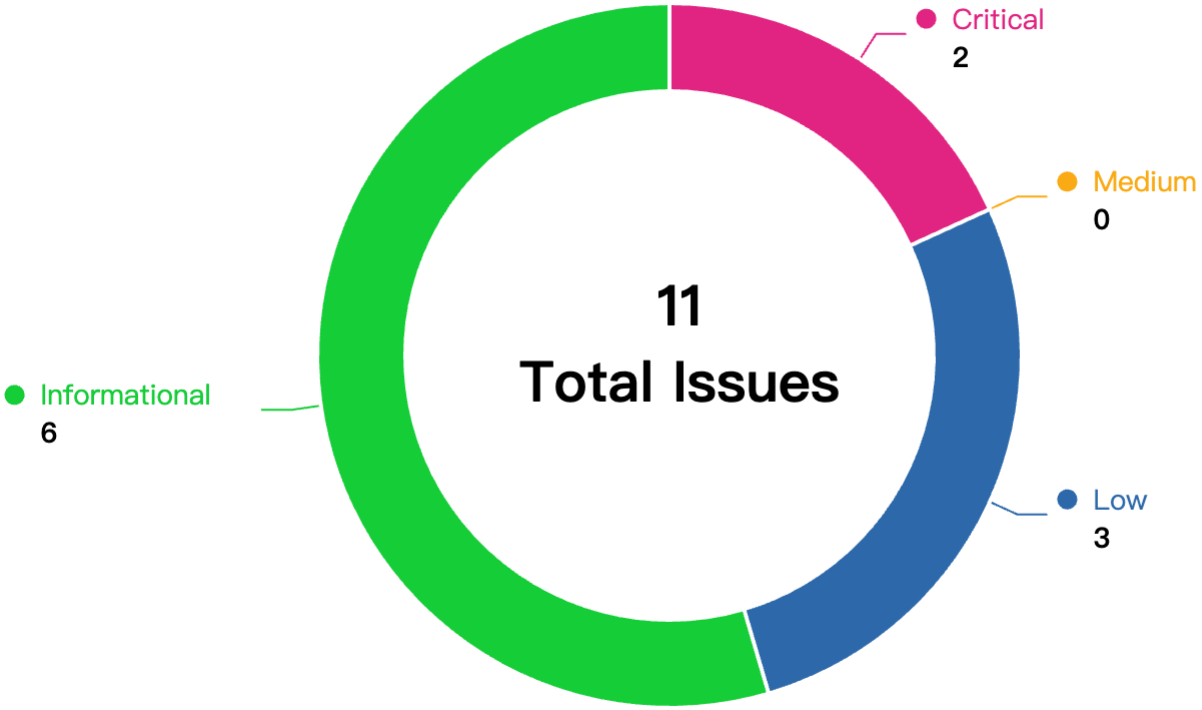
Overview

Project Name	KreationJP
Language	Solidity
Codebase	• https://github.com/kreation-team/tat-bcg-contracts.git • audit version - 79d34daa650fa3f7c1b2c9238ece07cf64eb6b51 • final version - 08f79a0af3f03188ad26e0e8de6744825d2a8773
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
contracts/ITATRumble.sol	79d34daa650fa3f7c1b2c9238ece07cf64eb6b51
contracts/TATRumble.sol	79d34daa650fa3f7c1b2c9238ece07cf64eb6b51

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
KJP-1	ERC721 Safemint Reentrancy	Reentrancy	Critical	Fixed	0xCO2
KJP-2	Amount of Minted Token Over the MaxSupply	Logical	Critical	Fixed	0xCO2
KJP-3	Privilege Account Set Configs Function Without Limit	Logical	Low	Fixed	xyzqwe123
KJP-4	Centralized Risk with Token Supply	Privilege Related	Low	Fixed	0xCO2
KJP-5	Use Deprecated Function	Logical	Low	Fixed	0xCO2
KJP-6	Use The Latest Solidity Version	Language Specific	Informational	Fixed	0xCO2
KJP-7	Unused State Variable/Library	Code Style	Informational	Fixed	0xCO2
KJP-8	Unnecessary Checked Arithmetic In Loop	Gas Optimization	Informational	Fixed	0xCO2
KJP-9	Storage Variable Caching In Memory	Gas Optimization	Informational	Fixed	0xCO2
KJP-10	Missing event record	Code Style	Informational	Fixed	0xCO2

KJP-11	Cache Array Length Outside For Loop	Gas Optimization	Informational	Fixed	0xCO2

KJP-1:ERC721 Safemint Reentrancy

Category	Severity	Client Response	Contributor
Reentrancy	Critical	Fixed	0xCO2

Code Reference

- code/contracts/TATRumble.sol#L70

```
70: _safeMint(to, quantity);
```

Description

0xCO2: It's a reentrancy attack caused by the **`_safeMint`** function of ERC721. This function checks whether the receiver can receive ERC721 tokens. If address to refers to a smart contract, it must implement **`onERC721Received`**, which is called upon a safe transfer. The implementation of **`_safeMint`** function is in https://github.com/chiru-labs/ERC721A/blob/6f8a82a7b2833ad8b2fc7b54349281143a731fdd/contracts/ERC721A.sol#L933C5-L954C6

Recommendation

0xCO2: Introducing a reentrancy (modifier) to protect mint function for reentrancy. It is recommended to use the Reentrancy Guard provided by OpenZeppelin.

Client Response

client response for 0xCO2: Fixed.

KJP-2:Amount of Minted Token Over the MaxSupply

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	0xCO2

Code Reference

- code/contracts/TATRumble.sol#L68-L71
- code/contracts/TATRumble.sol#L102-L106

```
68: function mint(address to, uint256 quantity) external onlyRole(MINTER_ROLE) {
69:         require(maxSupply == 0 || totalSupply() + quantity <= maxSupply, "Max supply reached");
70:         _safeMint(to, quantity);
71:     }
```

```
102: function setMaxSupply(uint256 amount) external onlyRole(EDITOR_ROLE) {
103:         require(maxSupply == 0 , "Max supply has already been set");
104:         require(amount > 0 && amount >= totalSupply(), "The maximum supply must be set higher t
han the already supplied amount");
105:         maxSupply = amount;
106:     }
```

Description

0xCO2: If function **`mint()`** is called before function **`setMaxSupply()`**, then **`_safeMint(to, quantity)`** will be executed since **`maxSupply==0`** at this point. After that, if function **`setMaxSupply()`** is called with the argument **`amount`**, and **`amount<quantity`**, the tokens minted will exceed **`maxSupply`**, and no more tokens can be minted(because of the statement **`totalSupply() + quantity <= maxSupply`**).

Recommendation

0xCO2: It is recommended to set **`maxSupply`** in the **`constructor`**.

Client Response

client response for 0xCO2: Fixed.

KJP-3:Privilege Account Set Configs Function Without Limit

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	xyzqwe123

Code Reference

- code/contracts/TATRumble.sol#L87-L106

```
87: function setBaseURI(string calldata baseURI) external onlyRole(EDITOR_ROLE) {
88:         _baseTokenURI = baseURI;
89:     }
90:
91:     function setPlaceholderURI(string memory placeholderTokenUri) external onlyRole(EDITOR_ROLE)
{
92:         _placeholderTokenURI = placeholderTokenUri;
93:     }
94:
95:     function setCustomURI(uint256[] memory tokens, string[] memory uriList) external onlyRole(ED
ITOR_ROLE) {
96:         require(tokens.length == uriList.length, "urilist length does not match tokens length");
97:         for (uint256 i = 0; i < tokens.length; i++) {
98:             _setCustomURI(tokens[i], uriList[i]);
99:         }
100:     }
101:
102:     function setMaxSupply(uint256 amount) external onlyRole(EDITOR_ROLE) {
103:         require(maxSupply == 0 , "Max supply has already been set");
104:         require(amount > 0 && amount >= totalSupply(), "The maximum supply must be set higher t
han the already supplied amount");
105:         maxSupply = amount;
106:     }
```

Description

xyzqwe123: Privileged accounts can set configs without limits, which could enable serious manipulation. This could result in major losses if exploited.

Recommendation

xyzqwe123: Avoid using centralized risk contracts.

Client Response

client response for xyzqwe123: Fixed.

KJP-4:Centralized Risk with Token Supply

Category	Severity	Client Response	Contributor
Privilege Related	Low	Fixed	0xCO2

Code Reference

- code/contracts/TATRumble.sol#L42

```
42: MinterCreatorSharedRoyalties(royaltyFeeNumerator_, minterShares_, creatorShares_, creator_, payme
ntSplitterReference_) {
```

Description

0xCO2: The constructor function **`MinterCreatorSharedRoyalties`** and the function is setting the arguments without any checks or validations. This means that these values can be set by the contract deployer or owner without any restrictions, potentially leading to a centralized control over the token supply and royalties distribution.

Recommendation

0xCO2: Avoid using centralized risk logic. Consider setting up some functions to mitigate the centralized risk with token supply vulnerability.

Client Response

client response for 0xCO2: Fixed.

KJP-5: Use Deprecated Function

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	0xCO2

Code Reference

- code/contracts/TATRumble.sol#L43

```
43: _setupRole(DEFAULT_ADMIN_ROLE, administrator);
```

Description

0xCO2: Function `**_setupRole()**` is deprecated in "@openzeppelin/contracts/access/AccessControl.sol". It may lead to some unexpected results.
Check here.

Recommendation

0xCO2: It is recommended to use the `**_grantRole()**` function instead.

Client Response

client response for 0xCO2: Fixed.

KJP-6:Use The Latest Solidity Version

Category	Severity	Client Response	Contributor
Language Specific	Informational	Fixed	0xCO2

Code Reference

- code/contracts/ITATRumble.sol#L3
- code/contracts/ITATRumble.sol#L3

```
3: pragma solidity ^0.8.9;
```


```
3: pragma solidity ^0.8.9;
```


- code/contracts/TATRumble.sol#L3
- code/contracts/TATRumble.sol#L3

```
3: pragma solidity ^0.8.9;
```


```
3: pragma solidity ^0.8.9;
```


Description

0xCO2: Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.
0xCO2: Developers should stay away from using floating pragma. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation

0xCO2: It is recommended to use a recent version of the Solidity compiler.

```
pragma solidity 0.8.25;
```

0xCO2: The best practice is to lock a pragma version.

```
pragma solidity 0.8.25;
```

Client Response

client response for 0xCO2: Fixed.
client response for 0xCO2: Fixed.

KJP-7:Unused State Variable/Library

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	0xCO2

Code Reference

- code/contracts/TATRumble.sol#L5
- code/contracts/TATRumble.sol#L22

```
5: import "@openzeppelin/contracts/utils/Strings.sol";
```

```
22: mapping(uint256 => bool) public utilized;
```

Description

0xCO2: The contract has declared a state variable `**utilized**` but it is not used anywhere in the code. This represents dead code or missing logic.
Unused state variable `**utilized**` leads to higher gas costs.
Having unused code or import statements incurs extra gas usage when deploying the contract. The contract imports the file "`**@openzeppelin/contracts/utils/Strings.sol**`" which is not used anywhere in the code.

Recommendation

0xCO2: It is recommended to remove the unused state variable `**utilized**`.
It is recommended to remove the import statement
`**import "@openzeppelin/contracts/utils/Strings.sol"**`.

Client Response

client response for 0xCO2: Fixed.

KJP-8:Unnecessary Checked Arithmetic In Loop

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	0xCO2

Code Reference

- code/contracts/TATRumble.sol#L97

```
97: for (uint256 i = 0; i < tokens.length; i++) {
```

Description

0xCO2: Increments inside a loop could never overflow due to the fact that the transaction will run out of gas before the variable reaches its limits. Therefore, it makes no sense to have checked arithmetic in such a place.

Recommendation

0xCO2: It is recommended to have the increment value inside the unchecked block and use `**++i**` instead of `**i++**` to save gas.

```
function setCustomURI(uint256[] memory tokens, string[] memory uriList) external onlyRole(EDITOR_R
OLE) {
    ...
    for (uint256 i = 0; i < tokens.length) {
        ...
        unchecked {
             ++i;
        }
    }
}
```

Client Response

client response for 0xCO2: Fixed.

KJP-9:Storage Variable Caching In Memory

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	0xCO2

Code Reference

- code/contracts/TATRumble.sol#L69

```
69: require(maxSupply == 0 || totalSupply() + quantity <= maxSupply, "Max supply reached");
```

Description

0xCO2: The state variable maxSupply is used multiple times in the function mint(). Opcode `SLOAD` is expensive (100 gas after the accessed) compared to `MLOAD/MSTORE` (3 gas each).

Recommendation

0xCO2: Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 SLOAD) and then read from this cache to avoid multiple SLOADs.

```
function mint(address to, uint256 quantity) external onlyRole(MINTER_ROLE) {
    uint256 maxSupply_ = maxSupply;
    require(maxSupply_ == 0 || totalSupply() + quantity <= maxSupply_, "Max supply reached");
    ...
}
```

Client Response

client response for 0xCO2: Fixed.

KJP-10:Missing event record

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	0xCO2

Code Reference

- code/contracts/TATRumble.sol#L68-L71
- code/contracts/TATRumble.sol#L73-L77
- code/contracts/TATRumble.sol#L79-L81
- code/contracts/TATRumble.sol#L83-L85
- code/contracts/TATRumble.sol#L87-L89
- code/contracts/TATRumble.sol#L91-L93
- code/contracts/TATRumble.sol#L102-L106

```
68: function mint(address to, uint256 quantity) external onlyRole(MINTER_ROLE) {
69:         require(maxSupply == 0 || totalSupply() + quantity <= maxSupply, "Max supply reached");
70:         _safeMint(to, quantity);
71:     }
```

```
73: function burn(uint256 tokenId) external onlyRole(BURNER_ROLE) {
74:         require(getApproved(tokenId) == msg.sender || isApprovedForAll(ownerOf(tokenId), msg.sen
der), "");
75:
76:         _burn(tokenId);
77:     }
```

```
79: function pause() external onlyRole(DEFAULT_ADMIN_ROLE) {
80:         _pause();
81:     }
```

```
83: function unpause() external onlyRole(DEFAULT_ADMIN_ROLE) {
84:         _unpause();
85:     }
```

```
87: function setBaseURI(string calldata baseURI) external onlyRole(EDITOR_ROLE) {
88:         _baseTokenURI = baseURI;
89:     }
```

```
91: function setPlaceholderURI(string memory placeholderTokenUri) external onlyRole(EDITOR_ROLE) {
92:         _placeholderTokenURI = placeholderTokenUri;
93:     }
```

```
102: function setMaxSupply(uint256 amount) external onlyRole(EDITOR_ROLE) {
103:         require(maxSupply == 0 , "Max supply has already been set");
104:         require(amount > 0 && amount >= totalSupply(), "The maximum supply must be set higher t
han the already supplied amount");
105:         maxSupply = amount;
106:     }
```

Description

0xCO2: Set-related functions, mint/burn-related functions, etc., should emit events. By emitting events, the smart contract can provide transparency and allow external systems to react to changes in the contract state.

Recommendation

0xCO2: Events should be emitted after state changes in functions that modify the contract state.
For instance:

```
function mint(address to, uint256 quantity) external onlyRole(MINTER_ROLE) {
        require(maxSupply == 0 || totalSupply() + quantity <= maxSupply, "Max supply reached");
        _safeMint(to, quantity);
        emit Minted(to, quantity);
}
```

Client Response

client response for 0xCO2: Fixed.

KJP-11:Cache Array Length Outside For Loop

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	0xCO2

Code Reference

- code/contracts/TATRumble.sol#L51
- code/contracts/TATRumble.sol#L60
- code/contracts/TATRumble.sol#L96-L97

```
51: for (uint256 i = 0; i < tokenList.length;) {
```

```
60: for (uint256 i = 0; i < tokenList.length;) {
```

```
96: require(tokens.length == uriList.length, "urilist length does not match tokens length");
97:         for (uint256 i = 0; i < tokens.length; i++) {
```

Description

0xCO2: Reading array length at each iteration of the loop takes 6 gas (3 gas for `**mload**` and 3 gas to place `**memory_offset**`) in the stack.
Caching the array length in the stack saves around 3 gas per iteration.

Recommendation

0xCO2: It is recommended to store the array's length in a variable before the for-loop.
For instance:

```
function deposit(uint256[] memory tokenList) external {
    uint256 len = tokenList.length;
    for (uint256 i = 0; i < len;) {
        ...
    }
}
```

Client Response

client response for 0xCO2: 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.