Audit report of RatKing Society

Prepared By: - Kishan Patel
Prepared On: - 13/12/2022

Prepared for: Nifty Splits Ltd

Table of contents

THIS AUDIT REPORT WILL CONTAIN CONFIDENTIAL INFORMATION ABOUT THE SMART CONTRACT AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES OF THEIR EXPLOITATION.

THE INFORMATION FROM THIS AUDIT REPORT CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.

1. Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). Because the total numbers of test cases are unlimited, the audit makes no statements or warranties on the security of the code.

It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

2. Introduction

Kishan Patel (Consultant) was contacted by Nifty Splits Ltd (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contracts and its code review conducted between 13/12/2022 – 14/12/2022.

The project has 2 file. It contains approx 600 lines of Solidity code. All the functions and state variables are well commented on using the natspec documentation, but that does not create any vulnerability.

3. Project information

Token Name	RatKing Society
Token Symbol	RatKing
Platform	Ethereum
Order Started Date	08/12/2022
Order Completed Date	10/12/2022

4. List of attacks checked

- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing Eth to a contract
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference

5. Severity Definitions

Risk	Level Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
Medium	Medium-level vulnerabilities are important to fix; however, they can’t lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can’t have significant impact on execution

6. Good things in code

- **Good required condition in functions:-**

  File Name: - RatKing Society.sol

  - Here you are checking that msg.sender has not already minted, and current mint counter does not reach MAX_PUBLIC_SUPPLY.

```
89      */
90      function mintRatKing() public {
91          //require(minterList[msg.sender] == false, Errors.RatKingAl
92          if (minterList[msg.sender] == true) revert Errors.RatKingAl
93          //require(_publicMintCounter.current() < MAX_PUBLIC_SUPPLY,
94          if (_publicMintCounter.current() >= MAX_PUBLIC_SUPPLY) reve
```

  - Here you are checking that length of to addresses + current mint counter is not bigger than MAX_GIFT_SUPPLY.
  - Question: - If owner is giving token as gift then also do you like to put address in minterList? Because then address will not able to mint in future.

```
106     */
107     function giftRatKing(address[] memory to) public onlyOwner {
108
109         //require(to.length + _giftCounter.current() <= MAX_GIFT_SU
110         if (to.length + _giftCounter.current() >= MAX_GIFT_SUPPLY)
```

  - Here you are checking that transfer to owner address is successfully done or not.

```
203     */
204     function withdraw() public onlyOwner nonReentrant {
205         (bool success, ) = payable(owner()).call{value: address(thi
206         require(success,"Transfer failed!");
207         emit WithdrawBalance(address(this).balance);
```

- Here you are checking that transfer to owner address is successfully done or not.

```
215     function withdrawERC20(address erc20TokenContract) public onlyO
216         IERC20 tokenContract = IERC20(erc20TokenContract);
217         //tokenContract.transfer(msg.sender, tokenContract.balanceO
218
219         (bool success, ) = payable(owner()).call{value: tokenContra
220         require(success,"Transfer failed!");
```

File Name: -FabNovel.sol

- Here you are checking that current tokenIdCounter is not bigger than or equal to MAX_FAB_NOVEL_SUPPLY, ratKing address is not already minted, and ratKing address is not owner of ratking contract.

```
94      function mintFabNovel(uint256 ratKing) public {
95          if(_tokenIdCounter.current() >= MAX_FAB_NOVEL_SUPPLY) rever
96
97          if(ratKingMinterList[ratKing] == true) revert Errors.RatKin
98
99          if(RK.ownerOf(ratKing) != msg.sender) revert Errors.NotYour
```

- Here you are checking that transfer to owner address is successfully done or not.

```
197     function withdraw() public onlyOwner nonReentrant {
198
199         (bool success, ) = payable(owner()).call{value: address(thi
200         require(success,"Transfer failed!");
```

- Here you are checking that transfer to owner address is successfully done or not.

```
209     function withdrawERC20(address erc20TokenContract) public onlyO
210         IERC20 tokenContract = IERC20(erc20TokenContract);
211         //tokenContract.transfer(msg.sender, tokenContract.balanceO
212
213         (bool success, ) = payable(owner()).call{value: tokenContra
214         require(success,"Transfer failed!");
```

7. Critical vulnerabilities in code

- **No Critical vulnerabilities found**

8. Medium vulnerabilities in code

- **No Medium vulnerabilities found**

9. Low vulnerabilities in code

9.1. Suggestions to add code validations:-

=> You have implemented required validation in contract.
=> There are some place where you can improve validation and security of your code.
=> These are all just suggestion it is not bug.

File Name: - RatKing Society.sol

- **No Low vulnerabilities found**

File Name: -FabNovel.sol

- **No Low vulnerabilities found**

10. Summary

- **Number of problems in the smart contract as per severity level**

Critical	Medium	Low
0	**0**	**0**

According to the assessment, the smart contract code is well secured. The code is written with all validation and all security is implemented. Code is performing well and there is no way to steal funds from this contract.

- **Good Point:** Code performance and quality are good. All kind of necessary validation added into smart contract and all validations are working as excepted.