Audit report of ChefGizmoToken

Prepared By: - Kishan Patel

Prepared On: - 23/04/2024.

Prepared for: Chef Gizmo Token Team

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 Chef Gizmo Token Team. (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 23/04/2024 – 25/04/2024.

The project has 1 file. It contains approx 300 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	Chef Gizmo Token
Token Symbol	Gizmo
Platform	Etherscan
Order Started Date	23/04/2024
Order Completed Date	24/04/2024

4. List of attacks checked

- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing BNB 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:-**
  - Here smart contract is checking that newOwner address is valid and proper.

```
39    function transferOwnership(address newOwner) public virtual onlyOwne
40        require(newOwner != address(0), "Ownable: new owner is the zero
41        _transferOwnership(newOwner);
42    }
```

  - Here smart contract is checking that sender and recipient addresses are valid and proper, and sender has sufficient balance to transfer amount.

```
146   function _transfer(    infinite gas
147       address sender,
148       address recipient,
149       uint256 amount
150   ) internal virtual {
151       require(sender != address(0), "ERC20: transfer from the zero ad
152       require(recipient != address(0), "ERC20: transfer to the zero a
153
154       uint256 senderBalance = _balances[sender];
155       require(senderBalance >= amount, "ERC20: transfer amount exceed
```

  - Here smart contract is checking that account address is valid and proper.

```
162   function _mint(address account, uint256 amount) internal virtual {
163       require(account != address(0), "ERC20: mint to the zero address"
164
```

  - Here smart contract is checking that owner and spender addresses are valid and proper.

```
169
170   function _approve(    infinite gas
171       address owner,
172       address spender,
173       uint256 amount
174   ) internal virtual {
175       require(owner != address(0), "ERC20: approve from the zero addre
176       require(spender != address(0), "ERC20: approve to the zero addre
177
```

- Here smart contract is checking that _uniswapV2Pair address is valid and proper.

```
244
245     function initializeAMMPair(address _uniswapV2Pair) public onlyOwner
246         require(_uniswapV2Pair != address(0), "ChefGizmoToken: AMM pair
247         uniswapV2Pair = _uniswapV2Pair;
248         automatedMarketMakerPairs[_uniswapV2Pair] = true;
249     }
```

- Here smart contract is checking that pair address is valid and proper, and pair address is not uniswapV2Pair address with value false.

```
251     function setAutomatedMarketMakerPair(address pair, bool value) publi
252         require(pair != address(0), "ChefGizmoToken: AMM pair address ca
253         require(pair != uniswapV2Pair || value == true, "ChefGizmoToken
```

- Here smart contract is checking that from and to addresses are valid and proper.

```
285     function _transfer(address from, address to, uint256 amount) interna
286         require(from != address(0), "ERC20: transfer from the zero addre
287         require(to != address(0), "ERC20: transfer to the zero address")
288
```

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.

- **Function: - _approve**

- Here in _approve function smart contract can check that owner address has sufficient balance to give allowance to spender address.

10. Summary

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

Critical	Medium	Low
0	0	1

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.

- **Suggestions:** Please try to implement suggested code validations.