Audit report of MSK

Prepared By:- Kishan Patel
Prepared On:- 24/01/2021

Prepared for: MSK

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 MSK (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 21/01/2021 – 24/01/2021.

The project has 1 file. It contains approx 2000 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	MSK
Token Symbol	MSK
Platform	Ethereum
Order Started Date	21/01/2021
Order Completed Date	24/01/2021

4. List of attacks checked

- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing Ethereum 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

- **SafeMath library is used:-**
  - You are using SafeMath library it is a good thing. This protects you from underflow and overflow attacks.

```
307  library SafeMath {
308      /**
309       * @dev Returns the addition of two unsigned integers, with an
310       *
```

- **Good required condition in functions:-**
  - Here you are checking that balance of contract should be bigger than or equal to amount, and transfer is successfully done or not.

```
554      function sendValue(address payable recipient, uint256 amount) i
555          require(address(this).balance >= amount, "Address: insuffic
556
557          // solhint-disable-next-line avoid-low-level-calls, avoid-c
558          (bool success, ) = recipient.call{ value: amount }("");
559          require(success, "Address: unable to send value, recipient
```

  - Here you are checking that balance of contract should be bigger than or equal to amount, and target address is contract address or not.

```
615      function functionCallWithValue(address target, bytes memory dat
616          require(address(this).balance >= value, "Address: insuffici
617          require(isContract(target), "Address: call to non-contract"
618
```

  - Here you are checking that target address is contract address or not.

```
639       */
640      function functionStaticCall(address target, bytes memory data,
641          require(isContract(target), "Address: static call to non-co
642
```

```
664      function functionDelegateCall(address target, bytes memory data
665          require(isContract(target), "Address: delegate call to non-
666
```

- Here you are checking that owner address is valid and proper.

```
1372    function balanceOf(address owner) public view virtual override
1373        require(owner != address(0), "ERC721: balance query for the
1374        return _holderTokens[owner].length();
```

- Here you are checking that tokenId is exists or not.

```
1401    function tokenURI(uint256 tokenId) public view virtual override
1402        require(_exists(tokenId), "ERC721Metadata: URI query for no
```

```
1467     */
1468    function getApproved(uint256 tokenId) public view virtual overr
1469        require(_exists(tokenId), "ERC721: approved query for nonex
```

```
1558    function _isApprovedOrOwner(address spender, uint256 tokenId) i
1559        require(_exists(tokenId), "ERC721: operator query for nonex
1560        address owner = ERC721.ownerOf(tokenId);
```

```
1676     */
1677    function _setTokenURI(uint256 tokenId, string memory _tokenURI)
1678        require(_exists(tokenId), "ERC721Metadata: URI set of nonex
```

- Here you are checking that to address is not same as owner, msg.sender is owner or approved or not.

```
1454    function approve(address to, uint256 tokenId) public virtual ov
1455        address owner = ERC721.ownerOf(tokenId);
1456        require(to != owner, "ERC721: approval to current owner");
```

- Here you are checking that operator address is msg.sender or not.

```
1476     */
1477    function setApprovalForAll(address operator, bool approved) pub
1478        require(operator != _msgSender(), "ERC721: approve to calle
```

- Here you are checking that to address is valid and proper, tokenId is exists or not.

```
1599    function _mint(address to, uint256 tokenId) internal virtual {
1600        require(to != address(0), "ERC721: mint to the zero address
1601        require(!_exists(tokenId), "ERC721: token already minted");
```

- Here you are checking that owner of tokenId is same as from address, to address is valid and proper.

```
1653    function _transfer(address from, address to, uint256 tokenId) i
1654        require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer
1655        require(to != address(0), "ERC721: transfer to the zero add
```

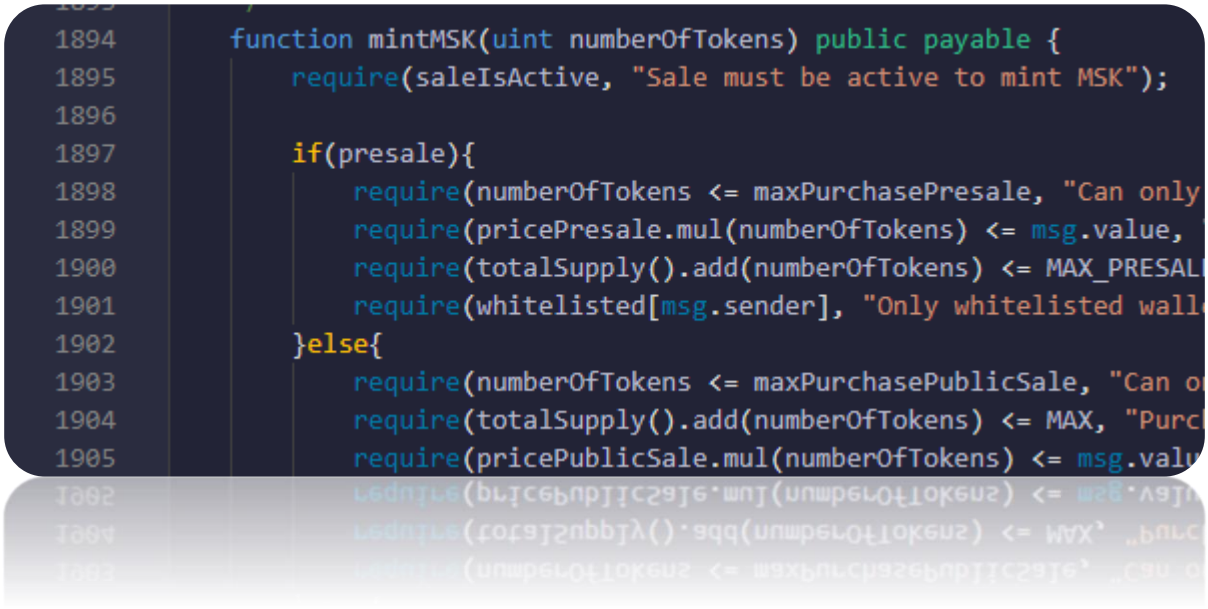
- Here you are checking that newOwner address is valid and proper.

```
1803    function transferOwnership(address newOwner) public virtual onl
1804        require(newOwner != address(0), "Ownable: new owner is the
1805        emit OwnershipTransferred(_owner, newOwner);
```

- Here you are checking that price value should be bigger than 0.14 eth.

```
1869
1870    function changePublicSalePrice(uint256 price) public onlyOwner
1871        require(price >= 140000000000000000, "Price cant go below 0
1872        pricePublicSale = price;
```

- Here you are checking that sale is active or not, user can mint 5 token at a time, msg.value is sufficient given or not, total supply + numberOfTokens should be smaller or equal to maxSupply, and msg.sender is whitelisted or not.



```
1894    function mintMSK(uint numberOfTokens) public payable {
1895        require(saleIsActive, "Sale must be active to mint MSK");
1896
1897        if(presale){
1898            require(numberOfTokens <= maxPurchasePresale, "Can only
1899            require(pricePresale.mul(numberOfTokens) <= msg.value,
1900            require(totalSupply().add(numberOfTokens) <= MAX_PRESAL
1901            require(whitelisted[msg.sender], "Only whitelisted wall
1902        }else{
1903            require(numberOfTokens <= maxPurchasePublicSale, "Can o
1904            require(totalSupply().add(numberOfTokens) <= MAX, "Purc
1905            require(pricePublicSale.mul(numberOfTokens) <= msg.valu
```

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. Compiler version is not fixed:-

=> In this file you have put “pragma solidity >=0.6.0;” which is not a good way to define compiler version.

=> Solidity source files indicate the versions of the compiler they can be compiled with. Pragma solidity >=0.6.0; // bad: compiles 0.6.0 and above pragma solidity 0.6.0; //good: compiles 0.6.0 only

=> If you put(>=) symbol then you are able to get compiler version 0.6.0 and above. But if you don’t use(^/>=) symbol then you are able to use only 0.6.0 version. And if there are some changes come in the compiler and you use the old version then some issues may come at deploy time.

9.2. 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: - changePresalePrice, changePublicSaleMaxMintPerTx, changePreSaleMaxMintPerTx**

```
1866    function changePresalePrice(uint256 price) public onlyOwner {
1867        pricePresale = price;
1868    }
```

```
1875    function changePublicSaleMaxMintPerTx(uint256 maxMint) public o
1876        maxPurchasePublicSale = maxMint;
1877    }
1878
1879    function changePreSaleMaxMintPerTx(uint256 maxMint) public only
1880        maxPurchasePresale = maxMint;
1881    }
```

  - Here in changePresalePrice function you can check that price value should be bigger than 0.
  - Here in changePublicSaleMaxMintPerTx function you can check that maxMint value should be bigger than 0.
  - Here in changePreSaleMaxMintPerTx function you can check that maxMint value should be bigger than 0.

9.3. Suggestions to add code validations:-

=> I have found that you are transferring fund to address using a transfer methods.

=> It is always good to check the return value or response from a function call.

=> Here are some functions where you forgot to check a response.

=> I suggest, if there is a possibility then please check the response.

- **Function: - withdraw**

```
1847    function withdraw() public onlyOwner {
1848        uint balance = address(this).balance;
1849
1850        address payable one = payable(owner());
1851        one.transfer(balance);
```

  - Here you are calling transfer method 1 time. It is good to check that the transfer is successfully done or not.

10. Summary

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

Critical	Medium	Low
0	**0**	**3**

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. Address validation and value validation is done properly

- **Suggestions:** Please use the static version of solidity, try to add suggested code validation, and try to check return response of transfer method.