Smart Contract
Security Audit Report

The SlowMist Security Team received the team's application for smart contract security audit of the MSQUARE on 2022.07.18. The following are the details and results of this smart contract security audit:

Token Name :

MSQUARE

The contract address :

https://github.com/webloom-io/token/blob/main/token_create.sol

commit: 85b12244adc0757efda2a99ae680bf7d632074a4

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

NO.	Audit Items	Result
1	Replay Vulnerability	Passed
2	Denial of Service Vulnerability	Passed
3	Race Conditions Vulnerability	Passed
4	Authority Control Vulnerability	Passed
5	Integer Overflow and Underflow Vulnerability	Passed
6	Gas Optimization Audit	Passed
7	Design Logic Audit	Passed
8	Uninitialized Storage Pointers Vulnerability	Passed
9	Arithmetic Accuracy Deviation Vulnerability	Passed
10	"False top-up" Vulnerability	Passed
11	Malicious Event Log Audit	Passed

NO.	Audit Items	Result
12	Scoping and Declarations Audit	Passed
13	Safety Design Audit	Passed
14	Non-privacy/Non-dark Coin Audit	Passed

Audit Result : Passed

Audit Number : 0X002207200002

Audit Date : 2022.07.18 - 2022.07.20

Audit Team : SlowMist Security Team

Summary conclusion : This is a token contract that does not contain the tokenVault section and does not contain dark coin functions. The total amount of contract tokens remains unchangeable. SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
pragma solidity ^0.5.2;
//SlowMist// SafeMath security module is used, which is a recommended approach
library SafeMath {

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {

        require(b > 0);
```

```
        uint256 c = a / b;

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}

interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns
(bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns
(uint256);

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

```
contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }
}
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowed;

    uint256 private _totalSupply;

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    function allowance(address owner, address spender) public view returns (uint256)
{
        return _allowed[owner][spender];
```

```
    }

    function transfer(address to, uint256 value) public returns (bool) {
        _transfer(msg.sender, to, value);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    function transferFrom(address from, address to, uint256 value) public returns
(bool) {
        _transfer(from, to, value);
        _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
        //SlowMist// The return value conforms to the EIP20 specification
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public returns
(bool) {
        _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public
returns (bool) {
        _approve(msg.sender, spender, _allowed[msg.sender]
[spender].sub(subtractedValue));
        return true;
    }

    function _transfer(address from, address to, uint256 value) internal {
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to the loss of token during transfer
        require(to != address(0));

        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(value);
        emit Transfer(from, to, value);
    }
```

```
    function _mint(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.add(value);
        _balances[account] = _balances[account].add(value);
        emit Transfer(address(0), account, value);
    }

    function _burn(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    function _approve(address owner, address spender, uint256 value) internal {
        //SlowMist// This kind of check is very good, avoiding user mistake leading
to approve errors
        require(spender != address(0));
        require(owner != address(0));

        _allowed[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _burnFrom(address account, uint256 value) internal {
        _burn(account, value);
        _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
    }
}

contract MSQUARE is ERC20, ERC20Detailed {
    uint8 public constant DECIMALS = 18;
    uint256 public constant INITIAL_SUPPLY = 25916431 * (10 ** uint256(DECIMALS));

    constructor () public ERC20Detailed("MSQUARE", "MSQ", DECIMALS) {
        _mint(msg.sender, INITIAL_SUPPLY);
    }
}
```

Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist