Smart Contract Security Audit Report

The SlowMist Security Team received the Orbit Chain team's application for smart contract security audit of the ORC on Oct. 29, 2020. The following are the details and results of this smart contract security audit:

Token name :

ORC

The Contract address :

0x662b67d00A13FAf93254714DD601F5Ed49Ef2F51

Link address :

https://etherscan.io/address/0x662b67d00A13FAf93254714DD601F5Ed49Ef2F51

The audit items and results :

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

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False Deposit” vulnerability Audit	-	Passed

9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : Passed

Audit Number : 0X002010300001

Audit Date : Oct. 30, 2020

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed, users can burn their tokens through the burn function. The owner can freeze or unfreeze the account of any user through the freezeAccount function. The contract does not have the Overflow and the Race Conditions issue.

The source code:

```
/**
 *Submitted for verification at Etherscan.io on 2019-06-24
*/

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

pragma solidity ^0.5.0;

contract owned {
    address public owner;
```

```
    constructor() public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    //SlowMist// The owner role does not use the event log to record after the ownership transfer

    function transferOwnership(address newOwner) onlyOwner public {
        owner = newOwner;
    }
}

contract ORCToken is owned{

    string public name;
    string public symbol;
    uint8 public decimals = 18;
    uint256 public totalSupply;

    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;
    mapping (address => bool) public frozenAccount;

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

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

    event Burn(address indexed from, uint256 value);

    event FrozenFunds(address target, bool frozen);

    constructor(
        uint256 initialSupply,
        string memory tokenName,
        string memory tokenSymbol
    ) public {
        totalSupply = initialSupply * 10 ** uint256(decimals);
        balanceOf[msg.sender] = totalSupply;
        name = tokenName;
```

```
        symbol = tokenSymbol;
    }

    function _transfer(address _from, address _to, uint _value) internal {
        require(_to != address(0x0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
        require(balanceOf[_from] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]);

        require(!frozenAccount[_from]);
        require(!frozenAccount[_to]);

        uint previousBalances = balanceOf[_from] + balanceOf[_to];
        balanceOf[_from] -= _value;
        balanceOf[_to] += _value;

        emit Transfer(_from, _to, _value);

        assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
    }

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

    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success) {
        require(_value <= allowance[_from][msg.sender]);     // Check allowance
        allowance[_from][msg.sender] -= _value;
        _transfer(_from, _to, _value);
        return true; //SlowMist// The return value conforms to the EIP20 specification
    }

    function approve(address _spender, uint256 _value) public returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true; //SlowMist// The return value conforms to the EIP20 specification
```

```
    }

    function burn(uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);
        balanceOf[msg.sender] -= _value;
        totalSupply -= _value;
        emit Burn(msg.sender, _value);
        return true;
    }

    //SlowMist// Because burnFrom() and transferFrom() share the allowance amount of approve(),
if the agent be evil, there is the possibility of malicious burn

    function burnFrom(address _from, uint256 _value) public returns (bool success) {
        require(balanceOf[_from] >= _value);
        require(_value <= allowance[_from][msg.sender]);
        balanceOf[_from] -= _value;
        allowance[_from][msg.sender] -= _value;
        totalSupply -= _value;
        emit Burn(_from, _value);
        return true;
    }

    //SlowMist// The owner can freeze or unfreeze the account of any user through the
freezeAccount function

    function freezeAccount(address target, bool freeze) onlyOwner public {
        frozenAccount[target] = freeze;
        emit FrozenFunds(target, freeze);
    }
}
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist