Smart Contract Security Audit Report

The SlowMist Security Team received the RUSH team's application for smart contract security audit of the RUSH on Jan. 08, 2021. The following are the details and results of this smart contract security audit:

Token name :

RUSH

The Contract address :

main.sol: 0x382A1667C9062F0621362F49076Ef6e4fE4C9eC7

LUSH_DOC_v02: 0x71803034Bc6Eb34363F46f1DE33b5805766c1967

Link address :

main.sol:

https://etherscan.io/address/0x382A1667C9062F0621362F49076Ef6e4fE4C9eC7

LUSH_DOC_v02:

https://etherscan.io/address/0x71803034Bc6Eb34363F46f1DE33b5805766c1967

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 authority audit	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 top-up” 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 : 0X002101120001

Audit Date : Jan. 12, 2021

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, admin role can burn his own tokens through the burn function. SafeMath security module is used, which is a recommend approach.

During the audit, we found the following issues:

1. The admin can arbitrarily change the address of the targetAddress contract through the setTargetAddress function.

2. The admin role can lock any user balance through the Lock_wallet function

The source code:

main.sol:

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

pragma solidity 0.5.0;

contract Proxy {
    address private targetAddress;

    address private admin;
    constructor() public {
        targetAddress = 0xba25E0c5D352655189bF9e3503189e4557B6d3c9;
        admin = msg.sender;
    }

    //SlowMist// The admin can change the address of the targetAddress contract at will.

    function setTargetAddress(address _address) public {
        require(msg.sender==admin , "Admin only function");
        require(_address != address(0));
        targetAddress = _address;
    }

    function getContAdr() public view returns (address) {
        require(msg.sender==admin , "Admin only function");
        return targetAddress;

    }
    function () external payable {
        address contractAddr = targetAddress;
        assembly {
            let ptr := mload(0x40)
            calldatacopy(ptr, 0, calldatasize)
            let result := delegatecall(gas, contractAddr, ptr, calldatasize, 0, 0)
            let size := returndatasize
            returndatacopy(ptr, 0, size)

            switch result
            case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
    }
```

```
}
```

LUSH_DOC_v02.sol:

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

pragma solidity 0.5.0;
//import "./SafeMath.sol"; 0x9205C049C231DdA51bAce0ba569f047E3E1e9979

//SlowMist// SafeMath security module is used, which is a recommend approach

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;
        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {return sub(a, b, "SafeMath: subtraction overflow");}
}

contract LUSH_DOC_v02 {
    using SafeMath for uint256;
    mapping (address => uint256) private _balances;
    mapping (address => mapping (address => uint256)) private _allowances;

    address private admin;
    string private _name;
    string private _symbol;
    uint8 private _decimals;
    uint256 private _totalSupply;
    string private last_useVersion;
    uint8 private isInit;

    struct LockDetails{
        uint256 lockedTokencnt;
        uint256 releaseTime;
    }
    struct managerDetail{
```

```
    string managername;
    uint8 managerlevel;
}
mapping(address => LockDetails) private Locked_list;
address[] private managerList;
mapping(address => managerDetail) private Managers;
mapping(address => mapping(bytes32 => string)) user_dataList;

event Transfer(address indexed from, address indexed to, uint256 value);
event Approval(address indexed owner, address indexed spender, uint256 value);
event lockwallet(address account, uint256 amount,uint256 releaseTime);

////////////////////////////////////////// Mint handle //////////////////////////////////////////
function Contadmin() public view returns (address) {return admin;}
function totalSupply() public view returns (uint256) {return _totalSupply;}
function name() public view returns (string memory) {return _name;}
function symbol() public view returns (string memory) {return _symbol;}
function getlast_useVersion() public view returns (string memory) {return last_useVersion;}
function decimals() public view returns (uint8) {return _decimals;}
////////////////////////////////////////// manager handle //////////////////////////////////////////



////////////////////////////////////////// Lock token handle //////////////////////////////////////////
```

//SlowMist// Admin can lock any user balance through the Lock_wallet function

```
function Lock_wallet(address _adr, uint256 lockamount,uint256 releaseTime ) public returns (bool) {
    require(_adr != address(0), "ERC20: transfer to the zero address");
    require(msg.sender== admin, "admin only function");
    require(releaseTime > block.timestamp , "Lock time must larger than now");
    require(lockamount >= _balances[_adr] , "Lock amount must bigger than balance");
    _Lock_wallet(_adr,lockamount,releaseTime);
    return true;
}
function _Lock_wallet(address account, uint256 amount,uint256 releaseTime) internal {
    LockDetails memory eaLock = Locked_list[account];
    if( eaLock.releaseTime > 0 ){
        eaLock.lockedTokencnt = amount;
        eaLock.releaseTime = releaseTime;
    }else{
        eaLock = LockDetails(amount, releaseTime);
    }
```

```
        Locked_list[account] = eaLock;
        emit lockwallet(account, amount, releaseTime);
    }

    function getwithdrawablemax(address account) public view returns (uint256) {
        return Locked_list[account].lockedTokencnt;
    }

    function getLocked_list(address account) public view returns (uint256) {
        return Locked_list[account].releaseTime;
    }
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);

        return true; //SlowMist// The return value conforms to the EIP20 specification

    }
    function _transfer(address sender, address recipient, uint256 amount) internal {
        last_useVersion = "Ver 1";
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");
        uint256 LockhasTime = Locked_list[sender].releaseTime;
        uint256 LockhasMax = Locked_list[sender].lockedTokencnt;
        if( block.timestamp < LockhasTime){
            uint256 OK1 = _balances[sender].sub(LockhasMax, "ERC20: transfer amount exceeds allowance");
            require( OK1 >= amount , "Your Wallet has time lock");
        }

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(msg.sender, spender, amount);

        return true;   //SlowMist// The return value conforms to the EIP20 specification

    }
```

```
    function transferFrom(address sender, address recipient, uint256 amount) public returns (bool) {
        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount, "ERC20: transfer amount exceeds allowance"));
        _transfer(sender, recipient, amount);
        return true;  //SlowMist// The return value conforms to the EIP20 specification
    }

    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }
    function burn(uint256 amount) public returns (bool) {
        require(msg.sender== admin, "admin only function");
        require(amount >= _balances[msg.sender] , "burn amount must bigger than balance");
        _burn(msg.sender, amount);
    }


    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");
        require(msg.sender == admin, "Admin only can burn   8547");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");

        require(spender != address(0), "ERC20: approve to the zero address"); //SlowMist// This kind of check is very good, avoiding user mistake leading to approve errors


        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

```
    ////////////////////////////////////// Lock token handle //////////////////////////////////////

    function getStringData(bytes32 key) public view returns (string memory) {
        return user_dataList[msg.sender][key];
    }
    function setStringData(bytes32 key, string memory value) public {
        user_dataList[msg.sender][key] = value;
    }
}
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist