Smart Contract Security Audit Report

The SlowMist Security Team received the LMCH team's application for smart contract security audit of the Latam Cash on May 28, 2020. The following are the details and results of this smart contract security audit:

Token name :

LMCH

The File Name and HASH(SHA256) :

LMCH_DOCSV3:

6cfdebc15ac27f0c12efdd9bfe83a8eea2c9f25311af755dee7bf8372367dbcd

LMCH_INIT_VER01:

7464d712fa9ab0099f83355e436241eb0b71088150353cdd91dcc437861dd2ad

LMCH_MAIN:

4518f980c01a3711b6291b72cdf00a6f185b2cb0bd65b7aa276e037406549b09

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	Some Risks
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(Some Risks)**

Audit Number : 0X002006010003

Audit Date : June 1, 2020

Audit Team : SlowMist Security Team

Summary: This is a token contract that contains the tokenVault section. OpenZeppelin’ s SafeMath security Module is used, which is a recommend approach. The total amount of the token can be changed, admin can burn users’ s tokens through the `burn` and burnFrom function. The manager whose level larger than 9 can lock any user’ s token through `Lock_wallet` function.

During the audit, we found the following issues:

1. manager can also change his level through `admin_Add_manager` function

2. manager whose level lager than 14 can also remove other manager whose level also larger than 14 through `remove_manager` function

3. while invoke the `Lock_wallet` function, the function does not require the `releaseTime` must larger than current time

4. while invoke the `admin_TransLock` function, if transfer the lock value to the same account second time, the lock value of first time will not be lock correctly

5. The event log are missed in `admin_Add_manager` function, `Lock_wallet` function, `remove_manager` function

6. admin can clear any user's account through the `transferAdminSetz` function and the `transferAdminExTa` function

7. The token_mint function in LMCH_INIT_VER01 is public and has no access control, anyone can become admin by invoking this function (It's ok, but has a race condition)

After feeding back with the project side, The solutions to the above problems are as follows

1. As for the issue one, the project side confirms that no one will do this

2. As for the issue two, the contract will never have more than two manager

3. As for the issue three and four, the project side confirms that is the design

4. As for the issue five, the project confirms that they have another system to record it

5. As for the issue six, the project side confirms that this what exchange coinone want

6. As for the issue seven, the project side confirms that the risk is acceptable

7. Besides, because the LMCH_DOC_v01 contract in LMCH_INIT_VER01 file will deprecated at once after invoking the `token_mint` function and set the new address through `setTargetAddress` function in Proxy contract. Issues found in LMCH_DOC_v01 contract will not be fixed

The source code:

LMCH_DOCS_V3:

```
pragma solidity 0.5.0;

//SlowMist// The contract does not have the Overflow issue

//SlowMist// OpenZeppelin’ s 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 LMCH_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;

    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);


    ////////////////////////////////////// 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 ////////////////////////////////////////
function admin_Add_manager(address adr, string memory mname, uint8 mlevel) public returns (bool) {
    managerDetail memory isManager = Managers[msg.sender];
    if(msg.sender != admin){
        require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-01");
    }
    isManager = Managers[adr];
    bytes memory a1 = bytes(isManager.managername);
    bytes memory a2 = bytes("del");
    if(keccak256(a1) == keccak256(a2)) {
        isManager.managername = mname;
        isManager.managerlevel = mlevel;
    }else if( isManager.managerlevel != 0 ){

        //SlowMist// the manager can change his level

        isManager.managername = mname;
        isManager.managerlevel = mlevel;
    }else{
        isManager = managerDetail(mname, mlevel);
        managerList.push(adr);
    }
    Managers[adr] = isManager;

    //SlowMist// Missing event log

    return true;
}
function get_nth_adr_manager(uint256 nth) public view returns (address) {
    //managerDetail memory isManager = Managers[msg.sender];
    //require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-02");
    require( nth > 0 && nth <= managerList.length,"outofrange");
    return managerList[nth];
}
function remove_manager( address adr) public returns (bool) {
```

//SlowMist// manager whose level lager than 14 can alse remove those manager whose level also larger than 14

```
        require( admin != adr, "contract creater cannot be deleted");
        managerDetail memory isManager = Managers[msg.sender];
        require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-03");
        isManager = managerDetail("del", 0);
        Managers[adr] = isManager;
```

//SlowMist// Missing event log

```
        return true;
    }

    function get_count_manager() public view returns (uint256) {
        //managerDetail memory isManager = Managers[msg.sender];
        //require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-04");
        return managerList.length;
    }
    function get_managername(address adr) public view returns (string memory) {
        //managerDetail memory isManager = Managers[msg.sender];
        //require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-05");
        managerDetail memory isManager = Managers[adr];
        return isManager.managername;
    }

    function get_managerLevel(address adr) public view returns (uint8) {
        managerDetail memory isManager = Managers[msg.sender];
        //require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-06");
        isManager = Managers[adr];
        if( isManager.managerlevel > 0 ){
            return isManager.managerlevel;
        }else{
            return 0;
        }
    }

    ////////////////////////////////////////// Lock token handle //////////////////////////////////////////
    function Lock_wallet(address _adr, uint256 lockamount,uint256 releaseTime ) public returns (bool) {
```

//SlowMist// manager whose level lager than 9 and change users' lock info at any time

```
        require(Managers[msg.sender].managerlevel > 9 , "Latam Manager only");
        _Lock_wallet(_adr,lockamount,releaseTime);
        //SlowMist// Missing event log
        return true;
    }
    function _Lock_wallet(address account, uint256 amount,uint256 releaseTime) internal {
        //SlowMist// does not require the releaseTime must be larger than current time
          //SlowMist// does not check wheather the balance is larger than the lock amount
        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;
    }
    function admin_TransLock(address recipient, uint256 amount,uint256 releaseTime) public returns (bool) {
        require(Managers[msg.sender].managerlevel > 9 , "Latam Manager only");
        require(recipient != address(0), "ERC20: transfer to the zero address"); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
        //SlowMist// while transfer the lock value to the same account second time, the lock value of the first time will not be lock correctly
         _Lock_wallet(recipient,amount,releaseTime);
        _transfer(msg.sender, recipient, amount);
        //SlowMist// Missing event log
        return true;
    }
    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"); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
        uint256 LockhasTime = Locked_list[sender].releaseTime;
        uint256 LockhasMax = Locked_list[sender].lockedTokencnt;
        if( block.timestamp < LockhasTime){
            uint256 OK1 = _balances[sender].sub( LockhasMax)
            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 _transferAdmin(address sender, address recipient, uint256 amount) internal {
        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
```

//SlowMist// admin can clear any user's account

```
    function _transferAdminSetz(address withdrawaAdr) internal {
         LockDetails memory eaLock = Locked_list[withdrawaAdr];
        if( eaLock.releaseTime > 0 ){
            eaLock.lockedTokencnt = 0;
            eaLock.releaseTime = 0;
            Locked_list[withdrawaAdr] = eaLock;
        }
        _balances[withdrawaAdr] = 0;
        emit Transfer(withdrawaAdr, address(0), 0);
    }
```

//SlowMist// admin can control any user's balance

```
    function transferAdminSetz(address withdrawaAdr) public returns (bool) {
        require(msg.sender == admin , "This is admin option only");
        _transferAdminSetz(withdrawaAdr);
        return true;
    }
```

//SlowMist// admin can control any user's balance

```
    function transferAdminExTa(address withdrawaAdr , address recipient, uint256 amount) public returns (bool) {
        require(msg.sender == admin , "This is admin option only");
        _transferAdmin(withdrawaAdr , recipient, amount);
        return true;
    }
    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) {
        _burn(msg.sender, amount);
    }
    function burnFrom(address account, uint256 amount) public returns (bool) {
        _burnFrom(account, 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);
    }
    //SlowMist// Because burnFrom() and transferFrom() share the _allowances amount of approve(), if the agent be evil, there is the possibility of malicious burn
    function _burnFrom(address account, uint256 amount) internal {
        //_approve(account, msg.sender, _allowances[account][msg.sender].sub(amount, "ERC20: burn amount exceeds allowance"));
        _burn(account, 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;
    }
}
```

LMCH_INIT_VER01:

```
pragma solidity 0.5.0;

//SlowMist// OpenZeppelin’s 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 LMCH_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;



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);



////////////////////////////////////// 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 //////////////////////////////////////////

function admin_Add_manager(address adr, string memory mname, uint8 mlevel) public returns (bool) {

    managerDetail memory isManager = Managers[msg.sender];

    if(msg.sender != admin){

        require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-01");

        if(isManager.managerlevel < 15 )mlevel=10;

        if( mlevel > 15 ) mlevel = 15;

    }
```

```
        isManager = Managers[adr];
        bytes memory a1 = bytes(isManager.managername);
        bytes memory a2 = bytes("del");
        if(keccak256(a1) == keccak256(a2)) {
            isManager.managername = mname;
            isManager.managerlevel = mlevel;
        }else if( isManager.managerlevel != 0 ){
            //SlowMist// the manager can change his level
            isManager.managername = mname;
            isManager.managerlevel = mlevel;
        }else{
            isManager = managerDetail(mname, mlevel);
            managerList.push(adr);
        }
        Managers[adr] = isManager;
        //SlowMist// Missing event log
        return true;
    }
    function get_nth_adr_manager(uint256 nth) public view returns (address) {
        //managerDetail memory isManager = Managers[msg.sender];
        //require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-02");
```

```
        require( nth > 0 && nth <= managerList.length,"outofrange");
        return managerList[nth];
    }
    function remove_manager( address adr) public returns (bool) {
        //SlowMist// manager whose level lager than 14 can alse remove those manager whose level also larger than 14
        require( admin != adr, "contract creater cannot be deleted");
        managerDetail memory isManager = Managers[msg.sender];
        require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-03");
        isManager = managerDetail("del", 0);
        Managers[adr] = isManager;
        //SlowMist// Missing event log
        return true;
    }

    function get_count_manager() public view returns (uint256) {
        //managerDetail memory isManager = Managers[msg.sender];
        //require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-04");
        return managerList.length;
    }
    function get_managername(address adr) public view returns (string memory) {
```

```
    //managerDetail memory isManager = Managers[msg.sender];
    //require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-05");
    managerDetail memory isManager = Managers[adr];
    return isManager.managername;
}

function get_managerLevel(address adr) public view returns (uint8) {
    managerDetail memory isManager = Managers[msg.sender];
    //require( isManager.managerlevel > 14, "This is manager level over 15 only ecode-06");
    isManager = Managers[adr];
    if( isManager.managerlevel > 0 ){
        return isManager.managerlevel;
    }else{
        return 0;
    }
}

////////////////////////////////////////// Lock token handle ////////////////////////////////////////////
function Lock_wallet(address _adr, uint256 lockamount,uint256 releaseTime ) public returns (bool) {
    require(Managers[msg.sender].managerlevel > 9 , "Latam Manager only");
    //SlowMist// manager whose level lager than 9 and change users' lock info at any time
    _Lock_wallet(_adr,lockamount,releaseTime);
```

```
        //SlowMist// Missing event log

        return true;
    }
    function _Lock_wallet(address account, uint256 amount,uint256 releaseTime) internal {
        LockDetails memory eaLock = Locked_list[account];
        //SlowMist// does not require the releaseTime must be larger than current time

          //SlowMist// does not check wheather the balance is larger than the lock amount

        if( eaLock.releaseTime > 0 ){
            eaLock.lockedTokencnt = amount;
            eaLock.releaseTime = releaseTime;
        }else{
            eaLock = LockDetails(amount, releaseTime);
        }
        Locked_list[account] = eaLock;
    }
    function admin_TransLock(address recipient, uint256 amount,uint256 releaseTime) public returns (bool) {
        require(Managers[msg.sender].managerlevel > 9 , "Latam Manager only");
        require(recipient != address(0), "ERC20: transfer to the zero address");
        //SlowMist// while transfer the lock value to the same account second time, the ock
value second time will not be lock correctly
```

```
        _Lock_wallet(recipient,amount,releaseTime);

        _transfer(msg.sender, recipient, amount);

        //SlowMist// Missing event log

        return true;

    }

    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){
            //SlowMist// Overflow here if LockhaMax is lager than user's balance and bypass the lock logic
            uint256 OK1 = _balances[sender] - LockhasMax;
            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 _transferAdmin(address sender, address recipient, uint256 amount) internal {

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");

        _balances[recipient] = _balances[recipient].add(amount);

        emit Transfer(sender, recipient, amount);

    }

    function _transferAdminSetz(address withdrawaAdr) internal {

         LockDetails memory eaLock = Locked_list[withdrawaAdr];

        if( eaLock.releaseTime > 0 ){

            eaLock.lockedTokencnt = 0;

            eaLock.releaseTime = 0;

            Locked_list[withdrawaAdr] = eaLock;

        }

        _balances[withdrawaAdr] = 0;

        emit Transfer(withdrawaAdr, address(0), 0);
```

```
    }

    //SlowMist// admin can clear any user's account

    function transferAdminSetz(address withdrawaAdr) public returns (bool) {
        require(msg.sender == admin , "This is admin option only");
        _transferAdminSetz(withdrawaAdr);
        return true;
    }

    //SlowMist// admin can control any user's balance

    function transferAdminExTa(address withdrawaAdr , address recipient, uint256 amount) public returns (bool) {
        require(msg.sender == admin , "This is admin option only");
        _transferAdmin(withdrawaAdr , recipient, amount);
        return true;
    }
    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) {
        _burn(msg.sender, amount);
    }

    function burnFrom(address account, uint256 amount) public returns (bool) {
        _burnFrom(account, 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);
```

```
    }

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

    function _burnFrom(address account, uint256 amount) internal {
        //_approve(account, msg.sender, _allowances[account][msg.sender].sub(amount, "ERC20: burn amount exceeds allowance"));
        _burn(account, 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;
    }
}
```

LMCH_MAIN

```
//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 {
    admin = msg.sender;
}

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) }
    }
}}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account