智能合约安全审计报告

慢雾安全团队于 2019-12-05 日，收到 HBTC 团队对 Huobi BTC 项目智能合约安全审计申请。如下为本次智能合约安全审计细节及结果：

Token 名称：

HBTC

文件名及 HASH(SHA256)：

hbtc-eth-contract1206.zip:

df6f4c0e02b6b5e0b51c1ab12f46cb0becc7cfcbc6220c2a21437add997c0907

本次审计项及结果：

（其他未知安全漏洞不包含在本次审计责任范围）

序号	审计大类	审计子类	审计结果
1	溢出审计	-	通过
2	条件竞争审计	-	通过
3	权限控制审计	权限漏洞审计	通过
		权限过大审计	通过
4	安全设计审计	Zeppelin 模块使用安全	通过
		编译器版本安全	通过
		硬编码地址安全	通过
		Fallback 函数使用安全	通过
		显现编码安全	通过
		函数返回值安全	通过
		call 调用安全	通过
5	拒绝服务审计	-	通过
6	Gas 优化审计	-	通过
7	设计逻辑审计	-	通过
8	“假充值”漏洞审计	-	通过
9	恶意 Event 事件日志审计	-	通过

10	变量声明及作用域审计	-	通过
11	重放攻击审计	ECDSA 签名重放审计	通过
12	未初始化的存储指针	-	通过
13	算术精度误差	-	通过

备注：审计意见及建议见代码注释 **//SlowMist//……**

审计结果：**通过**

审计编号：0X001912110001

审计日期：2019 年 12 月 11 日

审计团队：慢雾安全团队

总结：此为代币(token)合约，包含多签(Multisignature)部分。使用了 SafeMath 安全模块，值得称赞的做法。合约不存在溢出、条件竞争问题。综合评估合约无风险。

合约源代码如下：

Container.sol:

//SlowMist// 合约不存在溢出、条件竞争问题

```
pragma solidity ^0.5.11;

contract Container{
    struct Item{
        uint256 itemType;
        uint256 status;
        address[] addresses;
    }
    uint256 MaxItemAdressNum = 255;

    mapping (bytes32 => Item) private container;
```

```
function itemAddressExists(bytes32 id, address oneAddress) internal view returns(bool){
    for(uint256 i = 0; i < container[id].addresses.length; i++){
        if(container[id].addresses[i] == oneAddress)
            return true;
    }
    return false;
}
function getItemAddresses(bytes32 id) internal view returns(address[] memory){
    return container[id].addresses;
}

function getItemInfo(bytes32 id) internal view returns(uint256, uint256, uint256){
    return (container[id].itemType, container[id].status, container[id].addresses.length);
}

function getItemAddressCount(bytes32 id) internal view returns(uint256){
    return container[id].addresses.length;
}

function setItemInfo(bytes32 id, uint256 itemType, uint256 status) internal{
    container[id].itemType = itemType;
    container[id].status = status;
}

function addItemAddress(bytes32 id, address oneAddress) internal{
    require(!itemAddressExists(id, oneAddress), "dup address added");
    require(container[id].addresses.length < MaxItemAdressNum, "too many addresses");
    container[id].addresses.push(oneAddress);
}
function removeItemAddresses(bytes32 id) internal{
    container[id].addresses.length = 0;
}

function removeOneItemAddress(bytes32 id, address oneAddress) internal{
    for(uint256 i = 0; i < container[id].addresses.length; i++){
        if(container[id].addresses[i] == oneAddress){
            container[id].addresses[i] = container[id].addresses[container[id].addresses.length - 1];
            container[id].addresses.length--;
            return;
        }
    }
    revert("not exist address");
```

```
    }

    function removeItem(bytes32 id) internal{
        delete container[id];
    }

    function replaceItemAddress(bytes32 id, address oneAddress, address anotherAddress) internal{
        require(!itemAddressExists(id,anotherAddress),"dup address added");
        for(uint256 i = 0; i < container[id].addresses.length; i++){
            if(container[id].addresses[i] == oneAddress){
                container[id].addresses[i] = anotherAddress;
                return;
            }
        }
        revert("not exist address");
    }
}
```

HBTCAdmin.sol:

```
//SlowMist// 多签入口合约

pragma solidity ^0.5.11;

import "./Container.sol";

contract HBTCAdmin is Container{

    bytes32 internal constant OWNERHASH =
0x02016836a56b71f0d02689e69e326f4f4c1b9057164ef592671cf0d37c8040c0;
    bytes32 internal constant OPERATORHASH =
0x46a52cf33029de9f84853745a87af28464c80bf0346df1b32e205fc73319f622;
    bytes32 internal constant PAUSERHASH =
0x0cc58340b26c619cd4edc70f833d3f4d9d26f3ae7d5ef2965f81fe5495049a4f;
    bytes32 internal constant STOREHASH =
0xe41d88711b08bdcd7556c5d2d24e0da6fa1f614cf2055f4d7e10206017cd1680;
    bytes32 internal constant LOGICHASH =
0x397bc5b97f629151e68146caedba62f10b47e426b38db589771a288c0861f182;
    uint256 internal constant MAXUSERNUM = 255;
    bytes32[] private classHashArray;

    uint256 internal ownerRequireNum;
    uint256 internal operatorRequireNum;
```

```
event AdminChanged(string TaskType, string class, address oldAddress, address newAddress);
event AdminRequiredNumChanged(string TaskType, string class, uint256 previousNum, uint256 requiredNum);
event AdminTaskDropped(bytes32 taskHash);

function initAdmin(address owner0, address owner1, address owner2) internal{
    addItemAddress(OWNERHASH, owner0);
    addItemAddress(OWNERHASH, owner1);
    addItemAddress(OWNERHASH, owner2);
    addItemAddress(LOGICHASH, address(0x0));
    addItemAddress(STOREHASH, address(0x1));

    classHashArray.push(OWNERHASH);
    classHashArray.push(OPERATORHASH);
    classHashArray.push(PAUSERHASH);
    classHashArray.push(STOREHASH);
    classHashArray.push(LOGICHASH);
    ownerRequireNum = 2;
    operatorRequireNum = 2;
}

function classHashExist(bytes32 aHash) private view returns(bool){
    for(uint256 i = 0; i < classHashArray.length; i++)
        if(classHashArray[i] == aHash) return true;
    return false;
}
function getAdminAddresses(string memory class) public view returns(address[] memory) {
    bytes32 classHash = getClassHash(class);
    return getItemAddresses(classHash);
}
function getOwnerRequiredNum() public view returns(uint256){
    return ownerRequireNum;
}
function getOperatorRequiredNum() public view returns(uint256){
    return operatorRequireNum;
}

function resetRequiredNum(string memory class, uint256 requiredNum)
    public onlyOwner returns(bool){
    bytes32 classHash = getClassHash(class);
    require((classHash == OPERATORHASH) || (classHash == OWNERHASH),"wrong class");
    if (classHash == OWNERHASH)
```

```
        require(requiredNum <= getItemAddressCount(OWNERHASH),"num larger than existed owners");

    bytes32 taskHash = keccak256(abi.encodePacked("resetRequiredNum", class, requiredNum));
    addItemAddress(taskHash, msg.sender);

    if(getItemAddressCount(taskHash) >= ownerRequireNum){
        removeItem(taskHash);
        uint256 previousNum = 0;
        if (classHash == OWNERHASH){
            previousNum = ownerRequireNum;
            ownerRequireNum = requiredNum;
        }
        else if(classHash == OPERATORHASH){
            previousNum = operatorRequireNum;
            operatorRequireNum = requiredNum;
        }else{
            revert("wrong class");
        }
        emit AdminRequiredNumChanged("resetRequiredNum", class, previousNum, requiredNum);
    }
    return true;
}


function modifyAddress(string memory class, address oldAddress, address newAddress)
    internal onlyOwner returns(bool){
    bytes32 classHash = getClassHash(class);
    require(!itemAddressExists(classHash,newAddress),"address existed already");
    require(itemAddressExists(classHash,oldAddress),"address not existed");
    bytes32 taskHash = keccak256(abi.encodePacked("modifyAddress", class, oldAddress, newAddress));
    addItemAddress(taskHash, msg.sender);
    if(getItemAddressCount(taskHash) >= ownerRequireNum){
        replaceItemAddress(classHash, oldAddress, newAddress);
        emit AdminChanged("modifyAddress", class, oldAddress, newAddress);
        removeItem(taskHash);
        return true;
    }
    return false;
}

function getClassHash(string memory class) private view returns (bytes32){
    bytes32 classHash = keccak256(abi.encodePacked(class));
```

```
        require(classHashExist(classHash), "invalid class");
        return classHash;
    }

    function dropAddress(string memory class, address oneAddress)
        public onlyOwner returns(bool){
        bytes32 classHash = getClassHash(class);
        require(classHash != STOREHASH && classHash != LOGICHASH, "wrong class");
        require(itemAddressExists(classHash, oneAddress), "no such address exist");

        if(classHash == OWNERHASH)
            require(getItemAddressCount(classHash) > ownerRequireNum, "insuffience addresses");

        bytes32 taskHash = keccak256(abi.encodePacked("dropAddress", class, oneAddress));
        addItemAddress(taskHash, msg.sender);
        if(getItemAddressCount(taskHash) >= ownerRequireNum){
            removeOneItemAddress(classHash, oneAddress);
            emit AdminChanged("dropAddress", class, oneAddress, oneAddress);
            removeItem(taskHash);
            return true;
        }
        return false;

    }

    function addAddress(string memory class, address oneAddress)
        public onlyOwner returns(bool){
        bytes32 classHash = getClassHash(class);
        require(classHash != STOREHASH && classHash != LOGICHASH, "wrong class");
        require(!itemAddressExists(classHash,oneAddress),"address existed already");

        bytes32 taskHash = keccak256(abi.encodePacked("addAddress", class, oneAddress));
        addItemAddress(taskHash, msg.sender);
        if(getItemAddressCount(taskHash) >= ownerRequireNum){
            addItemAddress(classHash, oneAddress);
            emit AdminChanged("addAddress", class, oneAddress, oneAddress);
            removeItem(taskHash);
            return true;
        }
        return false;
    }
```

```
    function dropTask(bytes32 taskHash)
    public onlyOwner returns (bool){
        removeItem(taskHash);
        emit AdminTaskDropped(taskHash);
        return true;
    }

    modifier onlyOwner() {
        require(itemAddressExists(OWNERHASH, msg.sender), "only use owner to call");
        _;
    }

}
```

HBTCLogic.sol:

```
pragma solidity ^0.5.11;

import "./SafeMath.sol";
import "./HBTCStorage.sol";

contract HBTCLogic {

    using SafeMath for uint256;

    string public constant name = "HBTCLogic";

    uint256 public constant TASKINIT = 0;
    uint256 public constant TASKPROCESSING = 1;
    uint256 public constant TASKCANCELLED = 2;
    uint256 public constant TASKDONE = 3;
    uint256 public constant MINTTASK = 1;
    uint256 public constant BURNTASK = 2;

    address private caller;
    HBTCStorage private store;

    constructor(address aCaller) public{
        caller = aCaller;
    }

    modifier onlyCaller(){
```

```
        require(msg.sender == caller, "only main contract can call");
        _;
    }

    function mintLogic(uint256 value,address to,string calldata proof,
        bytes32 taskHash, address supportAddress, uint256 requireNum)
        external onlyCaller returns(uint256){

        require(to != address(0), "cannot be burned from zero address"); //SlowMist// 这类检查很好，避免操作失
```

误导致铸币时 Token 转丢

```
        require(value > 0, "value need > 0");
        require(taskHash == keccak256((abi.encodePacked(to,value,proof))),"taskHash is wrong");
        uint256 status = supportTask(MINTTASK, taskHash, supportAddress, requireNum);

        if( status == TASKDONE){
            uint256 totalSupply = store.getTotalSupply();
            uint256 balanceTo = store.balanceOf(to);
            balanceTo = balanceTo.safeAdd(value);
            totalSupply = totalSupply.safeAdd(value);
            store.setBalance(to,balanceTo);
            store.setTotalSupply(totalSupply);
        }
        return status;
    }

    function burnLogic(address from, uint256 value,string calldata btcAddress,
        string calldata proof,bytes32 taskHash, address supportAddress, uint256 requireNum)
        external onlyCaller returns(uint256){

        uint256 balance = store.balanceOf(from);
        require(balance >= value,"sender address not have enough HBTC");
        require(value > 0, "value need > 0");
        require(taskHash == keccak256((abi.encodePacked(from,value,btcAddress,proof))),"taskHash is wrong");
        uint256 status = supportTask(BURNTASK, taskHash, supportAddress, requireNum);

        if ( status == TASKDONE ){
            uint256 totalSupply = store.getTotalSupply();
            totalSupply = totalSupply.safeSub(value);
            balance = balance.safeSub(value);
            store.setBalance(from,balance);
            store.setTotalSupply(totalSupply);
```

```
        }
        return status;
    }

    function transferLogic(address sender,address to,uint256 value) external onlyCaller returns(bool) {

        require(to != address(0), "cannot transfer to address zero"); //SlowMist// 这类检查很好，避免用户失误导致 Token 转丢

        require(sender != to, "sender need != to");
        require(value > 0, "value need > 0");
        require(address(store) != address(0), "dataStore address error");

        uint256 balanceFrom = store.balanceOf(sender);
        uint256 balanceTo = store.balanceOf(to);
        require(value <= balanceFrom, "insufficient funds");
        balanceFrom = balanceFrom.safeSub(value);
        balanceTo = balanceTo.safeAdd(value);
        store.setBalance(sender,balanceFrom);
        store.setBalance(to,balanceTo);

        return true; //SlowMist// 返回值符合 EIP20 规范
    }

    function transferFromLogic(address sender,address from,address to,uint256 value) external onlyCaller returns(bool) {
        require(from != address(0), "cannot transfer from address zero");

        require(to != address(0), "cannot transfer to address zero"); //SlowMist// 这类检查很好，避免用户失误导致 Token 转丢

        require(value > 0, "can not tranfer zero Token");
        require(from!=to,"from and to can not be be the same ");
        require(address(store) != address(0), "dataStore address error");

        uint256 balanceFrom = store.balanceOf(from);
        uint256 balanceTo = store.balanceOf(to);
        uint256 allowedvalue = store.getAllowed(from,sender);

        require(value <= allowedvalue, "insufficient allowance");
        require(value <= balanceFrom, "insufficient funds");
```

```
        balanceFrom = balanceFrom.safeSub(value);
        balanceTo = balanceTo.safeAdd(value);
        allowedvalue = allowedvalue.safeSub(value);

        store.setBalance(from,balanceFrom);
        store.setBalance(to,balanceTo);
        store.setAllowed(from,sender,allowedvalue);

        return true; //SlowMist// 返回值符合 EIP20 规范

    }

    function approveLogic(address sender,address spender,uint256 value)   external onlyCaller returns(bool success){

        require(spender != address(0), "spender address zero"); //SlowMist// 这类检查很好，避免用户失误导致授
权错误

        require(value > 0, "value need > 0");
        require(address(store) != address(0), "dataStore address error");

        store.setAllowed(sender,spender,value);

        return true; //SlowMist// 返回值符合 EIP20 规范

    }

    function resetStoreLogic(address storeAddress) external onlyCaller {
        store = HBTCStorage(storeAddress);
    }

    function getTotalSupply() public view returns (uint256 supply) {
        return store.getTotalSupply();
    }

    function balanceOf(address owner) public view returns (uint256 balance) {
        return store.balanceOf(owner);
    }

    function getAllowed(address owner, address spender) public view returns (uint256 remaining){
        return store.getAllowed(owner,spender);
    }

    function getStoreAddress() public view returns(address){
        return address(store);
```

```
    }

    function supportTask(uint256 taskType, bytes32 taskHash, address oneAddress, uint256 requireNum) private
returns(uint256){
        require(!store.supporterExists(taskHash, oneAddress), "supporter already exists");
        (uint256 theTaskType,uint256 theTaskStatus,uint256 theSupporterNum) = store.getTaskInfo(taskHash);
        require(theTaskStatus < TASKDONE, "wrong status");

        if (theTaskStatus != TASKINIT)
            require(theTaskType == taskType, "task type not match");
        store.addSupporter(taskHash, oneAddress);
        theSupporterNum++;
        if(theSupporterNum >= requireNum)
            theTaskStatus = TASKDONE;
        else
            theTaskStatus = TASKPROCESSING;
        store.setTaskInfo(taskHash, taskType, theTaskStatus);
        return theTaskStatus;
    }

    function cancelTask(bytes32 taskHash)   external onlyCaller returns(uint256){
        (uint256 theTaskType,uint256 theTaskStatus,uint256 theSupporterNum) = store.getTaskInfo(taskHash);
        require(theTaskStatus == TASKPROCESSING, "wrong status");
        if(theSupporterNum > 0) store.removeAllSupporter(taskHash);
        theTaskStatus = TASKCANCELLED;
        store.setTaskInfo(taskHash, theTaskType, theTaskStatus);
        return theTaskStatus;
    }
}
```

HBTCStorage.sol:

```
pragma solidity ^0.5.11;

import "./Container.sol";

contract HBTCStorage is Container{

    string public constant name = "HBTCStorage";

    address private caller;

    constructor(address aCaller) public{
```

```
        totalSupply = 0;
        caller = aCaller;
    }
    uint256 public totalSupply;

    mapping (address => uint256) private balances;

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

    function supporterExists(bytes32 taskHash, address user) public view returns(bool){
        return itemAddressExists(taskHash, user);
    }

    function setTaskInfo(bytes32 taskHash, uint256 taskType, uint256 status) external onlyCaller{
        setItemInfo(taskHash, taskType, status);
    }

    function getTaskInfo(bytes32 taskHash) public view returns(uint256, uint256, uint256){
        return getItemInfo(taskHash);
    }

    function addSupporter(bytes32 taskHash, address oneAddress) external onlyCaller{
        addItemAddress(taskHash, oneAddress);
    }

    function removeAllSupporter(bytes32 taskHash) external onlyCaller{
        removeItemAddresses(taskHash);
    }

    modifier onlyCaller() {
        require(msg.sender == caller, "only use main main contract to call");
        _;
    }

    function getTotalSupply() external view returns(uint256) {
        return totalSupply;
    }

    function setTotalSupply(uint256 amount) external onlyCaller {
        totalSupply = amount;
    }
```

```
    function balanceOf(address account) external view returns(uint256) {
        return balances[account];
    }

    function setBalance(address account,uint256 amount) external onlyCaller {
        require(account != address(0),"account address error");
        balances[account] = amount;
    }

    function getAllowed(address owner,address spender) external view returns(uint256) {
        return allowed[owner][spender];
    }

    function setAllowed(address owner,address spender,uint256 amount) external onlyCaller {
        require(owner != address(0),"owner address error");
        require(spender != address(0),"spender address error");
        require(amount <= balances[owner], "owner balance need >= amount");
        allowed[owner][spender] = amount;
    }
}
```

HBTCToken.sol:

```
pragma solidity ^0.5.11;

import "./IERC20Token.sol";
import "./HBTCAdmin.sol";
import "./HBTCLogic.sol";
import "./HBTCStorage.sol";
import "./Pausable.sol";

contract HBTCToken is IERC20Token,Pausable, HBTCAdmin{
    string public constant name = "Huobi BTC";

    string public constant symbol = "HBTC";

    uint8 public constant decimals = 18;

    HBTCLogic private logic;

    event Burning(address indexed from, uint256 value, string proof, string  btcAddress, address burner);
    event Burned(address indexed from, uint256 value, string proof, string  btcAddress);
    event Minting(address indexed to, uint256 value, string proof, address  minter);
```

```
event Minted(address indexed to, uint256 value, string proof);

constructor(address owner0, address owner1, address owner2) public{
    initAdmin(owner0, owner1, owner2);
}


function totalSupply() public view returns (uint256 supply) {
    return logic.getTotalSupply();
}

function balanceOf(address owner) public view returns (uint256 balance) {
    return logic.balanceOf(owner);
}

function mint(address to, uint256 value, string memory proof,bytes32 taskHash) public whenNotPaused returns(bool){
    require(itemAddressExists(OPERATORHASH, msg.sender), "wrong operator");
    uint256 status = logic.mintLogic(value,to,proof,taskHash, msg.sender, operatorRequireNum);
    if (status == 1){
        emit Minting(to, value, proof, msg.sender);
    }else if (status == 3) {
        emit Minting(to, value, proof, msg.sender);
        emit Minted(to, value, proof);
        emit Transfer(address(0x0),to,value);
    }
    return true;
}


function burn(address from,uint256 value,string memory btcAddress,string memory proof, bytes32 taskHash)
public whenNotPaused returns(bool){
    require(itemAddressExists(OPERATORHASH, msg.sender), "wrong operator");
    uint256 status = logic.burnLogic(from,value,btcAddress,proof,taskHash, msg.sender, operatorRequireNum);
    if (status == 1){
        emit Burning(from, value, proof,btcAddress, msg.sender);
    }else if (status == 3) {
        emit Burning(from, value, proof,btcAddress,   msg.sender);
        emit Burned(from, value, proof,btcAddress);
        emit Transfer(from, address(0x0),value);
    }
    return true;
}
```

```
function cancelTask(bytes32 taskHash)    public returns(uint256){
    require(itemAddressExists(OPERATORHASH, msg.sender), "wrong operator");
    return logic.cancelTask(taskHash);
}

function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
    bool flag = logic.transferLogic(msg.sender,to,value);
    require(flag, "transfer failed");
    emit Transfer(msg.sender,to,value);
    return true;
}

function transferFrom(address from, address to, uint256 value) public whenNotPaused    returns (bool){
    bool flag = logic.transferFromLogic(msg.sender,from,to,value);
    require(flag,"transferFrom failed");
    emit Transfer(from, to, value);
    return true;
}


function approve(address spender, uint256 value) public whenNotPaused returns (bool){
    bool flag = logic.approveLogic(msg.sender,spender,value);
    require(flag, "approve failed");
    emit Approval(msg.sender, spender, value);
    return true;
}

function allowance(address owner, address spender) public view returns (uint256 remaining){
    return logic.getAllowed(owner,spender);
}

function modifyAdminAddress(string memory class, address oldAddress, address newAddress) public whenPaused{
    require(newAddress != address(0x0), "wrong address");
    bool flag = modifyAddress(class, oldAddress, newAddress);
    if(flag){
        bytes32 classHash = keccak256(abi.encodePacked(class));
        if(classHash == LOGICHASH){
            logic = HBTCLogic(newAddress);
        }else if(classHash == STOREHASH){
            logic.resetStoreLogic(newAddress);
        }
```

```
        }
    }

    function getLogicAddress() public view returns(address){
        return address(logic);
    }

    function getStoreAddress() public view returns(address){
        return logic.getStoreAddress();
    }

    //SlowMist// 在出现重大交易异常时可以暂停所有交易，值得称赞的做法
    function pause() public{
        require(itemAddressExists(PAUSERHASH, msg.sender), "wrong user to pauser");
        doPause();
    }

}
```

IERC20Token.sol:

```
pragma solidity ^0.5.11;

contract IERC20Token {
    function totalSupply() public view returns (uint256 supply);
    /// @param owner The address from which the balance will be retrieved
    /// @return The balance
    //solium-disable security/enforce-explicit-visibility
    function balanceOf(address owner) public view returns (uint256 balance);

    /// @notice send `value` token to `to` from `msg.sender`
    /// @param to The address of the recipient
    /// @param value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transfer(address to, uint256 value) public returns (bool success);

    /// @notice send `value` token to `to` from `from` on the condition it is approved by `from`
    /// @param from The address of the sender
    /// @param to The address of the recipient
    /// @param value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transferFrom(address from, address to, uint256 value) public returns (bool success);
```

```
    /// @notice `msg.sender` approves `spender` to spend `value` tokens
    /// @param spender The address of the account able to transfer the tokens
    /// @param value The amount of tokens to be approved for transfer
    /// @return Whether the approval was successful or not
    function approve(address spender, uint256 value) public returns (bool success);

    /// @param owner The address of the account owning tokens
    /// @param spender The address of the account able to transfer the tokens
    /// @return Amount of remaining tokens allowed to spent
    function allowance(address owner, address spender) public view returns (uint256 remaining);

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

}
```

Pausable.sol:

```
pragma solidity ^0.5.11;

contract Pausable {

    bool private pauseState = true;

    event PauseChangedTo(bool pauseState);

    function doPause() internal {
        pauseState = !pauseState;
        emit PauseChangedTo(pauseState);
    }

    function isPaused() public view returns (bool) {
        return pauseState;
    }

    modifier whenPaused() {
        require(pauseState, "it is not paused now");
        _;
    }

    modifier whenNotPaused() {
        require(!pauseState, "it is paused now");
```

```
        _;
    }

}
```

SafeMath.sol:

```
// solium-disable linebreak-style
pragma solidity ^0.5.11;

//SlowMist// 使用了 SafeMath 安全模块，值得称赞的做法

library SafeMath {
    function safeAdd(uint a, uint b) public pure returns (uint c) {
        c = a + b;
        require(c >= a,"");
    }
    function safeSub(uint a, uint b) public pure returns (uint c) {
        require(b <= a,"");
        c = a - b;
    }
    function safeMul(uint a, uint b) public pure returns (uint c) {
        c = a * b;
        require(a == 0 || c / a == b,"");
    }
    function safeDiv(uint a, uint b) public pure returns (uint c) {
        require(b > 0,"");
        c = a / b;
    }
}
```

慢雾科技
slow mist

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号