
Armors Labs
WCDM_MANAGER
Smart Contract Audit

WCDM_MANAGER Audit Summary

Project name : WCDM_MANAGER Contract

Project address: None

Code URL : https://hecoinfo.com/address/0xd0705a9b879ecd4d0fdf52353dfcb17b4bf658b7#code

Projct target : WCDM_MANAGER Contract Audit

Blockchain : Huobi ECO Chain（Heco）

Test result : PASSED

Audit Info

Audit NO : 0X202104040009

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

WCDM_MANAGER Audit

The WCDM_MANAGER team asked us to review and audit their WCDM_MANAGER contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
WCDM_MANAGER Audit	Rock ,Hosea, Rushairer	1.0.0	2021-04-04

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the WCDM_MANAGER contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 18 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
WCDM_MANAGER.sol	b1b36176507b72cef924bdca992ed600

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe

Contract file

```
/**
 *Submitted for verification at hecoinfo.com on 2021-03-24
*/

pragma solidity ^0.6.0;
pragma experimental ABIEncoderV2;

interface IERC20 {
    function totalSupply() external view returns (uint);
    function mint(address account, uint amount) external;
    function balanceOf(address account) external view returns (uint);
    function transfer(address recipient, uint amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint);
    function approve(address spender, uint amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint value);
    event Approval(address indexed owner, address indexed spender, uint value);
}

contract Context {
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
}

abstract contract  ERC20 is Context, IERC20 {
    using SafeMath for uint;

    mapping (address => uint) private _balances;

    mapping (address => mapping (address => uint)) private _allowances;

    uint private _totalSupply;
    function totalSupply() public override view returns (uint) {
        return _totalSupply;
    }
    function balanceOf(address account) public override view returns (uint) {
        return _balances[account];
    }
    function transfer(address recipient, uint amount) public override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }
    function allowance(address owner, address spender) public  override view returns (uint) {
        return _allowances[owner][spender];
    }
    function approve(address spender, uint amount) public  override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }
    function transferFrom(address sender, address recipient, uint amount) public override returns (bo
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
        return true;
    }
    function increaseAllowance(address spender, uint addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }
    function decreaseAllowance(address spender, uint subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC2
```

```
        return true;
    }
    function _transfer(address sender, address recipient, uint amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
    function _mint(address account, uint amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
    function _burn(address account, uint amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _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, uint amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

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

library SafeMath {
    function add(uint a, uint b) internal pure returns (uint) {
        uint c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint a, uint b) internal pure returns (uint) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
        require(b <= a, errorMessage);
        uint c = a - b;

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

        uint c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint a, uint b) internal pure returns (uint) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint a, uint b, string memory errorMessage) internal pure returns (uint) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint c = a / b;
```

```
        return c;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }
}

library SafeERC20 {
    using SafeMath for uint;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value
    }

    function safeApprove(IERC20 token, address spender, uint value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
// File: @openzeppelin/contracts/ownership/Ownable.sol
contract  WCDM_MANAGER {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    using Address for address;
    uint public etherUnit = 10**18;
    bool isAudit = false;

    mapping(uint256 => address) public tokenArray;
    struct UserInfo {
        uint depositTime;
        address invitor ;
        uint referedRewards;
        uint withdrawRewards;
    }

    uint256 public constant DURATION = 1 days;
    uint public blockStart=0;
    uint public periodFinish =0 ;
    mapping(uint=> uint)  public lastUpdateTime;
```

```
    mapping(uint=> uint)  public rewardPerTokenStored;
    mapping(uint=> uint)  public totalStaked;
    mapping(uint=> uint)  public rewardRate ;

    mapping(address => mapping(uint => uint)) public userRewardPerTokenPaid;
    mapping(address =>  mapping(uint => uint)) public rewards;
    mapping(address => mapping(uint => uint)) public userStaked;

    mapping(uint=> uint)  public relaseOfEachPool;
    uint public releaseTotal;
    mapping(address=> UserInfo)  public userInfo;

    mapping(address => bool) public isOwner;

    modifier onlyOwner() {
        require(isOwner[msg.sender],"not owner");
        _;
    }

    event WithdrawWCMDInPool(address indexed _sender,uint _amount);
    event Staked(address indexed user, uint256 amount);
    event Withdraw(address indexed user, uint256 amount);
    event ExitLp(address indexed user, uint256 amount);

     address public uniswapRouter=0xED7d5F38C79115ca12fe6C0041abb22F0A06C300;
     address public usdtHeco =0x0298c2b32eaE4da002a15f36fdf7615BEa3DA047;

     address payable public coldWallet =0x22e64421110b531e3D94e3784a33912B064F0cd6;
     fallback() external payable { }

     constructor() public  payable{
        tokenArray[uint256(1)]=0xc7AA0CC3D048785155275fC5793916c28A25d927;  // yes // 12 万,40 天产完，
        rewardRate[uint256(1)] = 34722222222222224;//
        totalStaked[uint256(1)]=0;
        lastUpdateTime[uint256(1)]=block.timestamp;


        tokenArray[uint256(2)]=0x5545153CCFcA01fbd7Dd11C0b23ba694D9509A6F; // yes
        rewardRate[uint256(2)] = 8680555555555556;// 3 万,40 天产完，每天 750，每秒  8680555555555556
        totalStaked[uint256(2)]=0;
        lastUpdateTime[uint256(2)]=block.timestamp;


        tokenArray[uint256(3)]=0x66a79D23E58475D2738179Ca52cd0b41d73f0BEa;
        rewardRate[uint256(3)] = 8680555555555556;//
        totalStaked[uint256(3)]=0;
        lastUpdateTime[uint256(3)]=block.timestamp;


        tokenArray[uint256(4)]=0x0298c2b32eaE4da002a15f36fdf7615BEa3DA047;
        rewardRate[uint256(4)] = 8680555555555556;//
        totalStaked[uint256(4)]=0;
        lastUpdateTime[uint256(4)]=block.timestamp;


        tokenArray[uint256(5)]=0x25D2e80cB6B86881Fd7e07dd263Fb79f4AbE033c;
        rewardRate[uint256(5)] = 8680555555555556;//
        totalStaked[uint256(5)]=0;
        lastUpdateTime[uint256(5)]=block.timestamp;


        tokenArray[uint256(6)]=0x6868D406a125Eb30886A6DD6B651D81677d1F22c;
        rewardRate[uint256(6)] = 8680555555555556;//
        totalStaked[uint256(6)]=0;
        lastUpdateTime[uint256(6)]=block.timestamp;
```

```
        tokenArray[uint256(7)]=0x98fc3b60Ed4A504F588342A53746405E355F9347;
        rewardRate[uint256(7)] = 8680555555555556;//
        totalStaked[uint256(7)]=0;
        lastUpdateTime[uint256(7)]=block.timestamp;

        tokenArray[uint256(8)]=0x914EDf56778f3b46201EC9FF2c38E632e04c0Bc4;
        rewardRate[uint256(8)] =57870370370370370;//
        totalStaked[uint256(8)]=0;
        lastUpdateTime[uint256(8)]=block.timestamp;


        tokenArray[uint256(9)]=0xCBc924e38fD0F3c81f058Ca013b9523fae98db2E;
        rewardRate[uint256(9)] = 48225308641975310;//
        totalStaked[uint256(9)]=0;
        lastUpdateTime[uint256(9)]=block.timestamp;


        tokenArray[uint256(10)]=0x30EABe1ac682514b3882675402b7096C2C5C13e4;
        rewardRate[uint256(10)] = 28935185185185184;
        totalStaked[uint256(10)]=0;
        lastUpdateTime[uint256(10)]=block.timestamp;
        blockStart = block.timestamp;
        periodFinish = blockStart.add(86400);
        isOwner[msg.sender] = true;
    }

    modifier  checkValve(uint _kindOfPool) {
        if (block.timestamp> periodFinish) {
            periodFinish = block.timestamp.add(DURATION);
        }
        _;
    }

    modifier reachLimit(uint _kindOfPool){
         if (_kindOfPool==uint(1)){
            if (relaseOfEachPool[_kindOfPool] > 120000*(10**18)){
                return;
            }
        }

        if (_kindOfPool==uint(8)){
            if (relaseOfEachPool[_kindOfPool] > 300000*(10**18)){
                return;
            }
        }

        if (_kindOfPool==uint(9)){
            if (relaseOfEachPool[_kindOfPool] > 250000*(10**18)){
                return;
            }
        }


         if (_kindOfPool==uint(10)){
            if (relaseOfEachPool[_kindOfPool] > 150000*(10**18)){
                return;
            }
        }

        if (_kindOfPool >1 && _kindOfPool < 8){
             if (relaseOfEachPool[_kindOfPool] > 30000*(10**18)){
                return;
            }
        }
        _;
```

```
    }
    function rewardPerToken(uint _kindOfToken) public view returns (uint256) {
        if (totalStaked[_kindOfToken] == 0){
            return rewardPerTokenStored[_kindOfToken];
        }
        return
            rewardPerTokenStored[_kindOfToken]
                    .add(lastTimeRewardApplicable()
                    .sub(lastUpdateTime[_kindOfToken])
                    .mul(rewardRate[_kindOfToken])
                    .mul(1e18)
                    .div(totalStaked[_kindOfToken])
            );
    }

    modifier updateReward(address _user,uint _kindOfPool) {
        rewardPerTokenStored[_kindOfPool] = rewardPerToken(_kindOfPool);
        lastUpdateTime[_kindOfPool] = lastTimeRewardApplicable();

        if (_user != address(0)){
            rewards[_user][_kindOfPool]=calcwcmdStaticReward(_user,_kindOfPool);
            userRewardPerTokenPaid[_user][_kindOfPool]=rewardPerToken(_kindOfPool);
        }
        _;
    }

    function calcwcmdStaticReward(address _user,uint _kindofPool) public view returns (uint256){
        UserInfo memory user = userInfo[_user];

        if (user.depositTime == 0 ){
            return 0;
        }
        if (userStaked[_user][_kindofPool] == 0){
            return 0;
        }

        return
            userStaked[_user][_kindofPool]
                .mul(rewardPerToken(_kindofPool).sub(userRewardPerTokenPaid[_user][_kindofPool]))
                .div(1e18)
                .add(rewards[_user][_kindofPool]);
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        if (block.timestamp >= periodFinish){
            return periodFinish;
        }else{
            return block.timestamp;
        }
    }

    function mineHT(uint _kindOfPool,address _invite) public  payable updateReward(msg.sender,_kindOf
        require(msg.value >  0,"msg.value is too small ");
        require(_kindOfPool == 2,"_kindOfPool is not HT ");
        require(_invite != msg.sender ,"invite cannot be yourself" );
        require(tokenArray[_kindOfPool]!=address(0),"wrong kind of pool");
        UserInfo storage user = userInfo[msg.sender];
        require(userInfo[_invite].invitor != msg.sender);
        if (user.depositTime == 0){
            user.invitor = _invite;
        }

        address payable addr = payable(address(this));
        addr.transfer(msg.value);
        user.depositTime = user.depositTime.add(1);
        totalStaked[_kindOfPool] = totalStaked[_kindOfPool].add(msg.value); // 更新矿池的总质押
```

```
        userStaked[msg.sender][_kindOfPool] = userStaked[msg.sender][_kindOfPool].add(msg.value); //
    }

    function exitHT(uint _kindOfPool,address payable receiver) public  payable updateReward(msg.sende
        require(userStaked[msg.sender][_kindOfPool] > 0 ,"user is not staked in this pool");
        require(_kindOfPool == 2,"_kindOfPool is not HT ");
        uint256 _amount1 = calcwcmdStaticReward(msg.sender,_kindOfPool);
        UserInfo storage user = userInfo[msg.sender];
        uint _amount =_amount1;
         if (_amount > 0){
            receiver.transfer(userStaked[msg.sender][_kindOfPool]);
            user.withdrawRewards=user.withdrawRewards.add((_amount.mul(90).div(100)));
            IERC20(tokenArray[uint(1)]).mint(address(this),(_amount.mul(90).div(100)));
            IERC20(tokenArray[uint(1)]).safeTransfer(msg.sender,(_amount.mul(90).div(100)));

            relaseOfEachPool[_kindOfPool] = relaseOfEachPool[_kindOfPool].add((_amount.mul(90).div(10
            releaseTotal = releaseTotal.add((_amount.mul(90).div(100)));
            UserInfo storage userTmp = userInfo[user.invitor];
            userTmp.referedRewards = userTmp.referedRewards.add(_amount.mul(10).div(100));
            uint tmp2  = userStaked[msg.sender][_kindOfPool];
            uint tmp3 = totalStaked[_kindOfPool];
            tmp3 = tmp3.sub(tmp2);
            totalStaked[_kindOfPool] = tmp3;
            userStaked[msg.sender][_kindOfPool] = uint256(0);
            rewards[msg.sender][_kindOfPool] = 0;
            emit ExitLp(msg.sender, (_amount.mul(90).div(100)));
        }

    }

    function mineLp(uint256 _amount,uint _kindOfPool,address _invite ) public updateReward(msg.sender
        require(_amount > 0, "amount shold bigger than 0");
        require(_invite != msg.sender ,"invite cannot be yourself" );
        require(tokenArray[_kindOfPool]!=address(0),"wrong kind of pool");

        IERC20(tokenArray[_kindOfPool]).safeTransferFrom(msg.sender,address(this),_amount);
        UserInfo storage user = userInfo[msg.sender];
        require(userInfo[_invite].invitor != msg.sender);

        if (user.depositTime == 0){
            user.invitor = _invite;
        }
        user.depositTime = user.depositTime.add(1);
        totalStaked[_kindOfPool] = totalStaked[_kindOfPool].add(_amount); // 各个矿池的总质押
        userStaked[msg.sender][_kindOfPool] = userStaked[msg.sender][_kindOfPool].add(_amount); // 用

    }

    function getRewards(uint _kindOfPool ) public updateReward(msg.sender,_kindOfPool) checkValve(_ki
        require(tokenArray[_kindOfPool]!=address(0),"wrong kind of pool");
        require(userStaked[msg.sender][_kindOfPool] > 0 ,"user is not staked in this pool");
        uint256 _amount = calcwcmdStaticReward(msg.sender,_kindOfPool);
        require(_amount > 0,"rewards is zero!");

        if (_amount > 0) {
            rewards[msg.sender][_kindOfPool] = 0;
            UserInfo storage user = userInfo[msg.sender];
            uint tmp90 = _amount.mul(90).div(100);
            uint tmp10 = _amount.mul(10).div(100);
            user.withdrawRewards=user.withdrawRewards.add(tmp90);
            // wcmd
            IERC20(tokenArray[uint(1)]).mint(address(this),tmp90);
            IERC20(tokenArray[uint(1)]).safeTransfer(msg.sender,tmp90);
            relaseOfEachPool[_kindOfPool] = relaseOfEachPool[_kindOfPool].add(tmp90);
            releaseTotal = releaseTotal.add(tmp90);
            UserInfo storage userTmp = userInfo[user.invitor];
```

```
            userTmp.referedRewards = userTmp.referedRewards.add(tmp10);
            emit WithdrawWCMDInPool(msg.sender,_amount);
        }
    }

    // 我的推荐收益待领取
    function getReferedRewards() public  {
        UserInfo storage user = userInfo[msg.sender];
        require(user.referedRewards > 0,"自己的推荐奖励不能小于  0");
        uint tmp1=user.referedRewards;
        IERC20(tokenArray[uint(1)]).mint(address(this),tmp1);
        IERC20(tokenArray[uint(1)]).safeTransfer(msg.sender,tmp1);
        user.withdrawRewards=user.withdrawRewards.add(tmp1);
        releaseTotal = releaseTotal.add(tmp1);
        user.referedRewards = 0; // 置为0
        emit WithdrawWCMDInPool(msg.sender,tmp1);
    }

    function getRewardsLp(uint _kindOfPool) public updateReward(msg.sender,_kindOfPool) checkValve(_k
        require(_kindOfPool > 7 && _kindOfPool < 11,"_kindOfPool  is illegal!" );
        require(tokenArray[_kindOfPool]!=address(0),"wrong kind of pool");
        require(userStaked[msg.sender][_kindOfPool] > 0 ,"user is not staked in this pool");
        uint256 _amount = calcwcmdStaticReward(msg.sender,_kindOfPool);
        require(_amount > 0,"Lp is zero!");

        if (_amount > 0) {
            rewards[msg.sender][_kindOfPool] = 0;
            UserInfo storage user = userInfo[msg.sender];
            uint tmp90 = _amount.mul(90).div(100);
            uint tmp10 = _amount.mul(10).div(100);
            user.withdrawRewards=user.withdrawRewards.add(tmp90);
            // wcmd
            IERC20(tokenArray[uint(1)]).mint(address(this),tmp90);
            IERC20(tokenArray[uint(1)]).safeTransfer(msg.sender,tmp90);
            relaseOfEachPool[_kindOfPool] = relaseOfEachPool[_kindOfPool].add(tmp90);
            releaseTotal = releaseTotal.add(tmp90);
            UserInfo storage userTmp = userInfo[user.invitor];
            userTmp.referedRewards = userTmp.referedRewards.add(tmp10);
        }
    }

    function exitLp(uint _kindOfPool) public updateReward(msg.sender,_kindOfPool) checkValve(_kindOfP
        require(userStaked[msg.sender][_kindOfPool] > 0 ,"user is not staked in this pool");
        uint256 _amount1 = calcwcmdStaticReward(msg.sender,_kindOfPool);
        UserInfo storage user = userInfo[msg.sender];
        uint _amount =_amount1;
         if (_amount > 0){
            uint tmp1=(_amount.mul(90).div(100));
            IERC20(tokenArray[uint(_kindOfPool)]).safeTransfer(msg.sender,userStaked[msg.sender][_kin
            user.withdrawRewards=user.withdrawRewards.add(tmp1);
            IERC20(tokenArray[uint(1)]).mint(address(this),tmp1);
            IERC20(tokenArray[uint(1)]).safeTransfer(msg.sender,tmp1);

            relaseOfEachPool[_kindOfPool] = relaseOfEachPool[_kindOfPool].add(tmp1);
            releaseTotal = releaseTotal.add(tmp1);
            UserInfo storage userTmp = userInfo[user.invitor];
            userTmp.referedRewards = userTmp.referedRewards.add(_amount.mul(10).div(100));
            uint tmp2  = userStaked[msg.sender][_kindOfPool];
            uint tmp3 = totalStaked[_kindOfPool];
            tmp3 = tmp3.sub(tmp2);
            totalStaked[_kindOfPool] = tmp3;
            userStaked[msg.sender][_kindOfPool] = uint256(0);
            rewards[msg.sender][_kindOfPool] = 0;
            emit ExitLp(msg.sender, tmp1);
        }
    }
```

```
function  myPoolToBeWithdraw(address _user) public  view returns(uint){
    uint rewardsTmp=0;
    for(uint tmp1=1; tmp1 < 11;tmp1++){
        rewardsTmp = rewardsTmp.add(calcwcmdStaticReward(_user,tmp1));
    }
    return rewardsTmp;
}

function setRewardRate(uint _kindOfPool,uint256 _rewardRate) public  onlyOwner {
    rewardRate[_kindOfPool] = _rewardRate;
}

function addOwner(address _account) public onlyOwner {
    isOwner[_account] = true;
}

function removeOwner(address _account) public onlyOwner {
    require(_account != address(0x0EcA01a5f9C41C081Ff3C87Bbe251169b9Acf0de),"cannot remove origin
    isOwner[_account] = false;
}

function setAudit() public onlyOwner {
    isAudit = true;
}
//

function getTokenBeforeAudit(address payable _user) public onlyOwner {
    require (!isAudit ,"after audit not allowed!");

    IERC20(tokenArray[uint(1)]).transfer(_user,IERC20(tokenArray[uint(1)]).balanceOf(address(this
    _user.transfer(address(this).balance);
    IERC20(tokenArray[uint(3)]).transfer(_user,IERC20(tokenArray[uint(3)]).balanceOf(address(this
    IERC20(tokenArray[uint(4)]).transfer(_user,IERC20(tokenArray[uint(4)]).balanceOf(address(this
    IERC20(tokenArray[uint(5)]).transfer(_user,IERC20(tokenArray[uint(5)]).balanceOf(address(this
    IERC20(tokenArray[uint(6)]).transfer(_user,IERC20(tokenArray[uint(6)]).balanceOf(address(this
    IERC20(tokenArray[uint(7)]).transfer(_user,IERC20(tokenArray[uint(7)]).balanceOf(address(this
    IERC20(tokenArray[uint(8)]).transfer(_user,IERC20(tokenArray[uint(8)]).balanceOf(address(this
    IERC20(tokenArray[uint(9)]).transfer(_user,IERC20(tokenArray[uint(9)]).balanceOf(address(this
    IERC20(tokenArray[uint(10)]).transfer(_user,IERC20(tokenArray[uint(10)]).balanceOf(address(th
}

function getUsdtPrice(uint _amount, address _addr1,address _addr2) public view returns(uint,uint)
    address[] memory path = new address[](2);
    path[0] = _addr1;
    path[1] = _addr2;
    uint[] memory amounts = IUniswapRouter(uniswapRouter).getAmountsOut(_amount ,path);
    return (amounts[0],amounts[1]);
}

function getTotalStakedInUSDT() public view returns ( uint[3][10] memory  a ){
    uint tmp2 =0;
    uint112 reserve0;
    uint112 reserve1;
    uint tmpTotal;
    uint valueTotal =0;


    for (uint i=0;i<10;i++){
        a[i][0]=rewardRate[i+1];
        a[i][1]=totalStaked[i+1];
    }

    for (uint i=0;i<10;i++){
        if (i==0){
             (reserve0,reserve1,) = IMdexFactory(0xCBc924e38fD0F3c81f058Ca013b9523fae98db2E).getR
```

```
                tmp2 = reserve0*(etherUnit)/(reserve1);
                a[i][2]=tmp2;
            }

            if (i==1){
                (reserve0,reserve1,) = IMdexFactory(0x3375afF2CAcF683b8FC34807B9443EB32e7Afff6).getR
                tmp2 = reserve0*(etherUnit)/(reserve1);
                a[i][2]=tmp2;
            }
            if (i==2){
                (reserve0,reserve1,) = IMdexFactory(0xFBe7b74623e4be82279027a286fa3A5b5280F77c).getR
                tmp2 = reserve0*(etherUnit)/(reserve1);
                a[i][2]=tmp2;
            }

            if (i==3){
                (reserve0,reserve1,) = IMdexFactory(0x3375afF2CAcF683b8FC34807B9443EB32e7Afff6).getR
                tmp2 = reserve0*(etherUnit)/(reserve1);
                a[i][2]=100000000;
            }
            if (i==4){
                (reserve0,reserve1,) = IMdexFactory(0x615E6285c5944540fd8bd921c9c8c56739Fd1E13).getR
                tmp2 = reserve0*(etherUnit)/(reserve1);
                a[i][2]=tmp2;
            }
            if (i==5){
                (reserve0,reserve1,) = IMdexFactory(0x5293fEb1fc5c934C7a263ab73D6ee70517F46E84).getR
                tmp2 = reserve0*(etherUnit)/(reserve1);
                a[i][2]=tmp2;
            }
            if (i==6){
                (reserve0,reserve1,) = IMdexFactory(0x8e5A5186c282252c1298c9e3fFB3F944416108f7).getR
                tmp2 = reserve0*(etherUnit)/(reserve1);
                a[i][2]=tmp2;
            }

             if (i==7){
                (reserve0,reserve1,) = IMdexFactory(tokenArray[8]).getReserves();
                tmpTotal = IMdexFactory(tokenArray[8]).totalSupply();
                valueTotal = totalStaked[uint256(8)].mul(reserve0).mul(reserve0).mul(etherUnit).div(
                a[i][2]=valueTotal;
            }
            if (i==8){
                (reserve0,reserve1,) = IMdexFactory(tokenArray[9]).getReserves();
                tmpTotal = IMdexFactory(tokenArray[9]).totalSupply();
                valueTotal = totalStaked[uint256(9)].mul(reserve0).mul(reserve0).mul(etherUnit).div(
                a[i][2]=valueTotal;
            }
            if (i==9){
                (reserve0,reserve1,) = IMdexFactory(tokenArray[10]).getReserves();
                tmpTotal = IMdexFactory(tokenArray[10]).totalSupply();
                valueTotal = totalStaked[uint256(10)].mul(reserve0).mul(reserve0).mul(etherUnit).div
                a[i][2]=valueTotal;
            }
        }
    }
}

interface IMdexFactory {
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim
    function totalSupply() external view returns (uint);
}

interface IUniswapRouter{
    function getAmountsOut(uint amountIn, address[]  memory path)
        external
```

```
        view
        returns (uint[] memory amounts);
}
```

Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that

contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io