KNOWNSEC

Blockchain Lab

Smart Contract Audit Report

Version description

The revision	Date	Revised	Version
Write documentation	**20220104**	**KNOWNSEC Blockchain Lab**	**V1.0**

Document information

Title	Version	Document Number	Type
HPC **Smart Contract Audit Report**	V1.0	7775ca70353749e4959b1ff09126d58f	**Open to project team**

Statement

KNOWNSEC Blockchain Lab only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. KNOWNSEC Blockchain Lab is unable to determine the security status of its smart contracts and is not responsible for the facts that will occur or exist in the future. The security audit analysis and other content made in this report are only based on the documents and information provided to us by the information provider as of the time this report is issued. KNOWNSEC Blockchain Lab 's assumption: There is no missing, tampered, deleted or concealed information. If the information provided is missing, tampered with, deleted, concealed or reflected in the actual situation, KNOWNSEC Blockchain Lab shall not be liable for any losses and adverse effects caused thereby.

Directory

1. Summarize

The effective test time of this report is **from December 29, 2021 to January 4, 2022.** During this period, the security and standardization of **the token code of HPC smart contracts** will be audited and used as the statistical basis for the report.

The scope of this smart contract security audit does not include external contract calls, new attack methods that may appear in the future, and code after contract upgrades or tampering. (With the development of the project, the smart contract may add a new pool , New functional modules, new external contract calls, etc.), does not include front-end security and server security.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 6). **The smart contract code of the HPC** is comprehensively assessed as **PASS**.

Since the testing is under non-production environment, all codes are the latest version. In addition, the testing process is communicated with the relevant engineer, and testing operations are carried out under the controllable operational risk to avoid production during the testing process, such as: Operational risk, code security risk.

KNOWNSEC Attest information:

classification	information
report number	7775ca70353749e4959b1ff09126d58f
report query link	https://attest.im/attestation/searchResult?qurey=7775ca70353749e4959b1ff09126d58f

2. Item information

2.1. Item description

HashPark is a decentralized platform that highly integrates DEFI, GameFi, NFT triple play and blockchain technology, and a meta-universe project with Defi income farm and GameFi as the core.

2.2. The project's website

Not yet.

2.3. The project's website

https://defihashpark.gitbook.io/defihashpark/

2.4. Review version code

DataPublic:

0x4b8070Ae0aB06b2ebb4837fCDcC9A9E836CDF0d7

MinerPool：

0x7C3497f9419e6a505f60025907b2adf599AAb693

Game:

0xB2f356F6CD3b6339C18ABd8c6e8B2b143205c546

LP:

0x997C77dD200205C6EE9F53169E1FC6715A56867c

2.5. Contract file and Hash/contract deployment address

The contract documents	MD5
MinerPool.sol	E5BD5FF6507E22D56D2C7705FB88FE88

DataPublic.sol	8ab93f3cd6fb3ac67c0546203a779462
Game.sol	4c72fb532b7d6889b66bc0df8fc0ca51
GameWallet.sol	e98bbf86855611c0fa19927340da09f1
LPWallet.sol	1623abd19670f0a6b88c7cacc4eea016
LP.sol	8f8a1766863aac459ad5b06e4537c4d2

3. External visibility analysis

3.1. DataPublic contracts

DataPublic					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
init	public	True	---	---	---
getChildrenCount	public	False	---	---	---
getChildren	public	False	---	---	---
getBindLogCount	public	False	---	---	---
getBindRecord	public	False	---	---	---
setMgr	public	True	---	---	---
getMgr	public	False	---	---	---
bind	private	True	---	---	---
checkParent	public	False	onlyMgr	---	---
bindParent	public	True	onlyMgr	---	---
getParent	public	False	---	---	---
onChildWithdrawLp	public	False	onlyMgr	---	---
onWithdrawGameRebate	public	False	onlyMgr	---	---
addGameRebate	public	True	onlyMgr	---	---
getRebateFromChild	public	False	---	---	---
getRevenuesCount	public	False	---	---	---
getRevenuesRecord1	public	False	---	---	---

3.2. MinerPool contracts

MinerPool					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
init	Public	True	---	---	---
getAddr	Public	False	---	---	---
setAddr	Public	True	onlyMgr	---	---
setUint256	Public	True	onlyMgr	---	---
getUint256	Public	False	---	---	---
getUint256s	Public	False	---	---	---
sendOut	Public	True	onlyMgr	---	---
getToday	Public	False	---	---	---
onMint	private	True	---	---	---
mineOut	Public	True	---	---	---
developerMint	Public	True	onlyMgr	---	---
setMgr	Public	True	onlyMgr	---	---
getMgr	Public	False	---	---	---
getTodayMint	Public	False	---	---	---
getDayMint	Public	False	---	---	---

3.3. LP contracts

LP					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
init	public	True	---	---	There are inspection restrictions
setAddr	public	True	onlyMgr	---	---
setUint256	public	True	onlyMgr	---	---
setEnable	public	True	onlyMgr	---	---
setMgr	public	False	onlyMgr	---	---
addPool	public	True	onlyMgr	---	---
fixPool	public	True	onlyMgr	---	---
beforeChangeLp	private	True	---	---	---
deposite	public	True	onlyMgr	payable	---
takeBack	Public	True	---	---	_mainContract Call restriction
reedeemFromVenus	Public	True	---	---	_mainContract|_owner Call restriction
depositeToVenus	Public	True	---	---	_mainContract|_owner Call restriction

3.4. LPWallet contracts

LPWallet					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
setMainContract	Public	True	---	---	_owner Call restriction
setVToken	Public	True	---	---	_mainContract Call restriction
approveVToken	Public	True	---	---	_mainContract Call restriction
deposite	Public	True	---	payable	_mainContract Call restriction
takeBack	Public	True	---	---	_mainContract Call restriction
reedeemFromVenus	Public	True	---	---	_mainContract|_owner Call restriction
depositeToVenus	Public	True	---	---	_mainContract|_owner Call restriction

3.5. Game contracts

Game					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
init	public	True	---	---	There are inspection restrictions
setMutiple	public	True	---	---	mgr Call restriction

setMgr	public	True	---	---	mgr Call restriction
setEnable	public	True	---	---	mgr Call restriction
caculateBlock	public	False	---	---	---
recharge	public	True	---	---	---
withdrawl	public	True	---	---	---
withdrawlByAmt	public	True	---	---	---
withdrawlRebate	public	True	---	---	---
incRebat	private	True	---	---	---
callIssue	private	True	---	---	---
bet	public	True	---	---	---
settle	private	True	---	---	---
doSettle	public	True	---	---	mgr Call restriction
doASettle	public	True	---	---	mgr Call restriction
sendFeeAndBurn	public	True	---	---	mgr Call restriction

3.6. GameWallet contracts

GameWallet					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
addBalance	Public	True	---	---	_game Call restriction
decBalance	Public	True	---	---	_game Call restriction
withdrawlByAmt	Public	True	---	---	_game Call restriction

withdrawl	Public	True	---	---	_game Call restriction
burn	Public	True	---	---	_game Call restriction
sendRebate	Public	True	---	---	_game Call restriction
sendFee	Public	True	---	---	_game Call restriction

4. Code vulnerability analysis

4.1. Summary description of the audit results

Audit results			
audit project	**audit content**	**condition**	**description**
Business security detection	DataPublic contract sub-address binding and sub-address information acquisition function	Pass	After testing, there is no security issue.
	DataPublic contract LpRebate information acquisition function	Pass	After testing, there is no security issue.
	DataPublic contract GameRebate information acquisition function	Pass	After testing, there is no security issue.
	Minerpool contract minting and pledge reward collection function	Pass	After testing, there is no security issue.
	Minerpool contract authority management function	Pass	After testing, there is no security issue.
	GameWallet.sol reduces user balance function	Pass	After testing, there is no security issue.
	LP contract takes away the reward function	Pass	After testing, there is no security issue.
	Game contract settlement function	Pass	After testing, there is no security issue.

Code basic vulnerability detection	Compiler version security	Pass	After testing, there is no security issue.
	Redundant code	Pass	After testing, there is no security issue.
	Use of safe arithmetic library	Pass	After testing, there is no security issue.
	Not recommended encoding	Pass	After testing, there is no security issue.
	Reasonable use of require/assert	Pass	After testing, there is no security issue.
	fallback function safety	Pass	After testing, there is no security issue.
	tx.origin authentication	Pass	After testing, there is no security issue.
	Owner permission control	Pass	After testing, there is no security issue.
	Gas consumption detection	Pass	After testing, there is no security issue.
	call injection attack	Pass	After testing, there is no security issue.
	Low-level function safety	Pass	After testing, there is no security issue.
	Vulnerability of additional token issuance	Pass	After testing, there is no security issue.
	Access control defect detection	Pass	After testing, there is no security issue.
	Numerical overflow detection	Pass	After testing, there is no security issue.
	Arithmetic accuracy error	Pass	After testing, there is no security issue.
	Wrong use of random number detection	Pass	After testing, there is no security issue.
	Unsafe interface use	Pass	After testing, there is no security issue.
	Variable coverage	Pass	After testing, there is no security issue.

	Uninitialized storage pointer	Pass	After testing, there is no security issue.
	Return value call verification	Pass	After testing, there is no security issue.
	Transaction order dependency detection	Pass	After testing, there is no security issue.
	Timestamp dependent attack	Pass	After testing, there is no security issue.
	Denial of service attack detection	Pass	After testing, there is no security issue.
	Fake recharge vulnerability detection	Pass	After testing, there is no security issue.
	Reentry attack detection	Pass	After testing, there is no security issue.
	Replay attack detection	Pass	After testing, there is no security issue.
	Rearrangement attack detection	Pass	After testing, there is no security issue.

5. Business security detection

5.1. DataPublic contract sub-address binding and sub-address information acquisition function【Pass】

Audit analysis: Perform a security audit on the sub-address detection and binding function in the contract. This function consists of getChildrenCount, getChildren, getBindRecord, bind, checkParent, bindParent, getParent functions. After auditing, the logical design is reasonable and no security issues are found.

```
function getChildrenCount(address addr) public view returns (uint256) {
//knownsec // Get the total number of subaddresses
        return _children[addr].length;
    }

    function getChildren(address addr, uint256 from) public view returns(address[20] memory) {
//knownsec    // Get sub address
        address[20] memory rs;
        uint256 index = 0;
        while (from < _children[addr].length && index < 20) {
            rs[index] = _children[addr][from];
            index++;
            from++;
        }
        return rs;
    }

    function getBindLogCount(address addr) public view returns (uint256) {
//knownsec // Get the total number of binding lists
        return _bindRecord[addr].length;
    }
```

```
function getBindRecord(address addr, uint256 from) public view returns (uint256[20] memory, address[20] memory, uint256[20] memory, uint256[20] memory) {
        //knownsec    // Get binding list details
        uint256[20] memory times;
        address[20] memory addrs;
        uint256[20] memory gRebate;
        uint256[20] memory lpRebate;

        uint256 index = 0;
        while (index < 20 && from < _bindRecord[addr].length) {
            times[index] = _bindRecord[addr][from].blockTime;
            addrs[index] = _bindRecord[addr][from].child;
            gRebate[index] = _gameRebates[addr][addrs[index]];
            lpRebate[index] = _lpRebates[addr][addrs[index]];
            index++;
            from++;
        }

        return (times, addrs, gRebate, lpRebate);
    }
 function bind(address addr, address parent) private {
// knownsec // Bind parent-child address pair
        _parent[addr] = parent;
        _isBindParent[addr] = true;
        if (parent == address(0)) {
            return;
        }
        _children[parent].push(addr);

        _bindRecord[parent].push(BindRecord({
            blockTime: block.timestamp,
            child: addr
```

```
        }));
    }

    function checkParent(address addr, address parent) public onlyMgr {
// knownsec // Check and bind the parent address
        if (addr == parent) {
            return;
        }
        if (_isBindParent[addr]) {
            return;
        }
        bind(addr, parent);
    }

    function bindParent(address addr, address parent) public onlyMgr {
// knownsec // Directly bind the parent address
        if (addr == parent) {
            return;
        }
        bind(addr, parent);
    }

    function getParent(address addr) public view returns (address, bool) {
// knownsec // Get the parent address and return whether it is bound
        return (_parent[addr],_isBindParent[addr]);
    }
```

Security advice: None.

5.2. DataPublic contract LpRebate information acquisition function【Pass】

Audit analysis: Perform a security audit on the LpRebate information acquisition function in the contract. This function is composed of onWithdrawGameRebate and getRevenuesRecord functions. After auditing, the logical design is reasonable and no security issues are found.

```
    function onChildWithdrawLp(address parent, address child, uint256 amt, uint256 pRebate, bool isReward) public onlyMgr {

            // knownsec // Get sub address lp rebate
            _lpRebates[parent][child] +=   pRebate;
            _revenuesRecord[child].push(RevenuesRecord({
                blockTime: block.timestamp,
                amt: amt,
                tp: isReward ? 1 : 2
            }));
        }

        }
    function getRevenuesRecord(address addr, uint256 from) public view returns(uint256[20] memory, uint256[20] memory, uint8[20] memory) {
            //knownsec // Get a list of income records
             uint256[20] memory times;
            uint256[20] memory amts;
            uint8[20] memory tps;
            uint256 index = 0;
            while (from < _revenuesRecord[addr].length && index < 20) {
                times[index] = _revenuesRecord[addr][from].blockTime;
                amts[index] = _revenuesRecord[addr][from].amt;
                tps[index] = _revenuesRecord[addr][from].tp;
                index++;
                from++;
            }
            return (times, amts, tps);
```

```
}
```

Security advice: None.

5.3. DataPublic contract GameRebate information acquisition function【Pass】

Audit analysis: Perform a security audit on the GameRebate information acquisition function in the contract. This function is composed of onWithdrawGameRebate, addGameRebate, and getRebateFromChild functions. After auditing, the logical design is reasonable and no security issues are found.

```
function addGameRebate(address parent, address child, uint256 amt) public onlyMgr{
    // knownsec // Increase gamerebate data
    _gameRebates[parent][child] += amt;
}

function getRebateFromChild(address parent, address child) public view returns(uint256, uint256) {
        //knownsec // Get sub address rebate record

    return (_gameRebates[parent][child], _lpRebates[parent][child]);
}


function getRevenuesCount(address addr) public view returns(uint256) {
    //knownsec // Get the total number of revenue records
    return _revenuesRecord[addr].length;
}
```

Security advice: None.

5.4. Minerpool contract minting and pledge reward collection function 【Pass】

Audit analysis: Perform security audits on the onMint/ mineOut/ developerMint functions in the contract. onMint implements the update of the number of coins and the balance of tokens, mineOut implements the function of pledge rewards, and developerMint implements the function of developer coin minting. The function permission is onlyMgr which is a normal business requirement. After auditing, the logical design is reasonable and no security issues are found.

```
    function onMint(uint256 mintAmt, uint256 devMintAmt, bool isInc) private    {
        if (isInc) {-
            _dataUint256[eUint256.lpMint]                                          =
_dataUint256[eUint256.lpMint].add(mintAmt);
            _dataUint256[eUint256.developerMint]                                   =
_dataUint256[eUint256.developerMint].add(devMintAmt);
            _dataUint256[eUint256.developerBalance]                                =
_dataUint256[eUint256.developerBalance].add(devMintAmt);
            uint256 today = getToday();
            _dayMintLog[today] = _dayMintLog[today].add(mintAmt).add(devMintAmt);
        } else {
            _dataUint256[eUint256.developerBalance]                                =
_dataUint256[eUint256.developerBalance].subBe0(devMintAmt);
        }
    }//knownsec // Token update

    function mineOut(address to,uint256 amount) public    {
        require(msg.sender == _dataAddr[eAddr.lpContract], "L");

        uint256 rate = _dataUint256[eUint256.developerRate];
```

```
        uint256 mintAmt = amount.mul(rate).div(1e10);

        onMint(amount, mintAmt, true);
        _dataAddr[eAddr.hpc].safeTransfer(to, amount);
    }// knownsec//Pledge reward

    function developerMint(address to) public onlyMgr {
        uint256 amount = _dataUint256[eUint256.developerBalance];
        if (amount == 0) {
            return;
        }
        onMint(0, amount, false);

        _dataAddr[eAddr.hpc].safeTransfer(to, amount);
    }// knownsec//Developer Minting
```

Security advice: None.

5.5. Minerpool contract authority management function 【Pass】

Audit analysis: conduct a security audit on the authority management function in the contract. The function setMgr sets the management authority by setting the Boolean value of the address mapping, and the function authority is onlyMgr. After auditing, the logical design is reasonable and no security issues are found.

```
    function setMgr(address addr, bool v) public onlyMgr {
        if (addr == _dataAddr[eAddr.owner]) { // knownsec//Determine if you are an administrator
            return;
        }
```

```
        if (_mgr[addr] != v) {
            _mgr[addr] = v;
        }
    }// knownsec // Set management permissions
```

Security advice: None.

5.6. GameWallet.sol reduces user balance function【Pass】

Audit analysis: Perform a security audit on GameWallet.sol's function of adding user balance. This function is implemented by the function decBalance. The function logic of decBalance is to verify whether the caller is _game, compare the user's balance with the passed parameters, and subtract the user's balance from the pass. Parameter value. After auditing, the permission to use this method is: _game, this function is controlled by the user account by game/is a normal business requirement.

```
function decBalance(address addr, uint256 amt) public {
        require(msg.sender == _game, "g");
        require(_balances[addr] >= amt);
        _balances[addr] = _balances[addr].sub(amt);
    }// knownsec // Reduce user balance
```

Security advice: None.

5.7. LP contract takes away the reward function【Pass】

Audit analysis: Perform a security audit on the LP contract take away reward function. This function is mainly implemented by the functions withdrawReward and getPendingReawrd. The function logic is that the function getPendingReawrd first obtains user reward information, judges the reward and updates the lp information, and obtains the referrer of the caller. Information, add rebates to the recommender, and finally mint coins to the caller. The method call permission is: public. After audit, this method is a function for ordinary users to obtain rewards/is a normal business requirement.

```
    function withdrawReward(address token) public /*lockAddr(msg.sender)*/ {
        uint256[3] memory reward = getPendingReawrd(msg.sender, token);
        if (reward[0] == 0) {
            return;
        }

        _lp[msg.sender][token].lastCheckBlock = uint256(block.number);
        _lp[msg.sender][token].pendingReward = 0;

        (address parent,) = IDataPublic(_dataAddr[eAddr.dataPublic]).getParent(msg.sender);
        if (parent != address(0)) {
            uint256 prebate = reward[0].mul(_dataUint256[eUint256.rebateRate]).div(1e10);
            if (prebate > 0) {
                incRebate(parent, msg.sender, prebate, reward[0], true);
            }
        }

        IMinerPool(_dataAddr[eAddr.minerPool]).mineOut(msg.sender, reward[0]);
        emit WithdrawReward(msg.sender, token, reward[0]);
    }// knownsec // Take away rewards

    function getPendingReawrd(address addr, address token) public view returns (uint256[3] memory) {
        uint256[3] memory rs;
```

```
        rs[1] = _lp[addr][token].lastCheckBlock;
        rs[2] = uint256(block.number);
        if (_lp[addr][token].amountInUsdt == 0) {
            rs[0] = _lp[addr][token].pendingReward;
            return rs;
        }

        uint256 b = block.number;
        uint256 bn = b.sub(_lp[addr][token].lastCheckBlock);

        // scale is 1e10
        uint256 rate = _pool[token].interestRatePerBlock;
        if (_userInfo[addr][eUser.gameAccelerate] != 0) {
            rate +=  rate.mul(_dataUint256[eUint256.gac]).div(1e10);
        }

        // price scale 1e18
        // revenuePerBlock = total deposite * rate / hpcprice
        uint256 revenuePerBlock = _lp[addr][token].amountInUsdt.mul(rate).mul(1e18).div(_lp[addr][token].hpcPrice);
        // rate scale is 1e10
        uint256 interest  = bn.mul(revenuePerBlock).div(1e10);

        rs[0] = interest.add(_lp[addr][token].pendingReward);
        return rs;
    }// knownsec // Get reward information
```

Security advice: None.

5.8. Game contract settlement function【Pass】

Audit analysis: Perform security audit on the game contract settlement function. This function is mainly implemented by the function settle. Its main functional logic is to intercept the betting information and compare it with the winning logic, select the winning address and update the actual reward income, and update the actual fee. , And finally update the bet and record the lottery block. After auditing, the permission to use this function is: mgr. This method is a function of the normal operation of the game/is a normal business requirement.

```
    function settle(uint256 startBlock, address addr,  uint8[3] memory result, uint256 d)
private /*lockAddr(addr)*/ {
        if (addr == address(0)) {
            return;
        }
        uint256 totalBet = _issueBets[startBlock].betInfo[addr].totalBet;
        if (totalBet == 0) {
            return;
        }
        uint256 win = 0;
        uint256 notWinBet = 0;

        for (uint8 i = eBet.odd; i <= eBet.num9; i++) {
            uint256 amt = _issueBets[startBlock].betInfo[addr].bets[i];
            if (amt == 0) {
                continue;
            }
            bool isWin = false;
            if (i == eBet.odd) {
                isWin = result[2] % 2 == 1;
```

```
        } else if (i == eBet.even) {
            isWin = result[2] % 2 == 0;
        } else if (i == eBet.small) {
            isWin = result[2] <= 4;
        } else if (i == eBet.big) {
            isWin = result[2] >= 5;
        } else if (i == eBet.oddBig) {
            isWin = result[2] == 5 || result[2] == 7 || result[2] == 9;
        } else if (i == eBet.oddSmall) {
            isWin = result[2] == 1 || result[2] == 3;
        } else if (i == eBet.evenBig) {
            isWin = result[2] == 6 || result[2] == 8;
        } else if (i == eBet.evenSmall) {
            isWin = result[2] == 0 || result[2] == 2 || result[2] == 4;
        } else if (i == eBet.r2) {
            isWin =   result[2] == result[1];
        } else if (i == eBet.r3) {
            isWin = result[2] == result[1] && result[1] == result[0];
        } else if (i >= eBet.num0 && i <= eBet.num9) {
            isWin =   result[2] == (i - eBet.num0);
        }

        if (isWin) {
            win = win.add(amt.mul(_mutiple[i]).div(10000));
        } else {
            notWinBet = notWinBet.add(amt);
        }
    }

    // win = win.mul(10000 - _dataUint256[eUint256.fee]).div(10000);
    if (win > 0) {
        uint256 actualWin = win.mul(10000 - _dataUint256[eUint256.fee]).div(10000);
```

```
                    _issueBets[startBlock].betInfo[addr].totalWin =
_issueBets[startBlock].betInfo[addr].totalWin.add(actualWin);
                    _wallet.addBalance(addr, actualWin);
                    _dataUint256[eUint256.totalWin] =
_dataUint256[eUint256.totalWin].add(actualWin);
                }


            uint256 fee = notWinBet.add(win);
            if (fee > 0) {
                uint256 feeToOwner = fee.mul(_dataUint256[eUint256.feeToOwner]).div(10000);
                if (feeToOwner > 0) {
                    // _wallet.sendFee(_dataAddr[eAddr.feeAddr], feeToOwner);
                    _dataUint256[eUint256.curFee] =
_dataUint256[eUint256.curFee].add(feeToOwner);
                }

                uint256 feeToParent = fee.mul(_dataUint256[eUint256.feeToParent]).div(10000);
                if (feeToParent > 0){
                    (address parent,) =
IDataPublic(_dataAddr[eAddr.dataPublic]).getParent(addr);
                    if (parent != address(0)) {
                        incRebate(parent, feeToParent, addr);
                    }
                }

                uint256 feeBurn = fee.mul(_dataUint256[eUint256.feeBurn]).div(10000);
                if (feeBurn > 0) {
                    // _wallet.burn(burn);
                    _dataUint256[eUint256.curBurn] =
_dataUint256[eUint256.curBurn].add(feeBurn);
                    _dataUint256[eUint256.totalBurn] =
_dataUint256[eUint256.totalBurn].add(feeBurn);
```

```
                _dayBurnLog[d] = _dayBurnLog[d].add(feeBurn);
            }
        }

        // _issueBets[startBlock].betInfo[addr].timestamp = block.timestamp;
        _dataUint256[eUint256.totalBet] = _dataUint256[eUint256.totalBet].add(totalBet);

        _betBlocks[addr].push(startBlock);
    }// knownsec // Settlement

    function doSettle(uint256 startBlock, uint256 from, uint256 to, uint8[3] memory result,
bool isFinish) public {
        require(_mgrs[msg.sender], "mgr");
        uint256 today = getToday();
        for (uint256 i = from; i <= to && i < _issueBets[startBlock].betAddrs.length; i++) {
            address addr = _issueBets[startBlock].betAddrs[i];
            settle(startBlock, addr,   result, today);
        }
        _issueBets[startBlock].lastSettleIndex = to;
        if (isFinish) {
            _dataUint256[eUint256.startBlock] = 0;
        }
        if (_issueBets[startBlock].result.length == 0) {
            _issueBets[startBlock].result.push(result[0]);
            _issueBets[startBlock].result.push(result[1]);
            _issueBets[startBlock].result.push(result[2]);
        }
    }// knownsec // Final settlement

    function doASettle(uint256 startBlock, address addr, uint8[3] memory result) public {
        require(_mgrs[msg.sender], "mgr");
        uint256 today = getToday();
        settle(startBlock, addr,   result, today);
```

```
}// knownsec // Settlement
```

Security advice: None.

6. Code basic vulnerability detection

6.1. Compiler version security【Pass】

Check to see if a secure compiler version is used in the contract code implementation.

Detection results: After detection, the smart contract code has developed a compiler version of 0.8.0 or more, there is no security issue.

Security advice: None.

6.2. Redundant code【Pass】

Check that the contract code implementation contains redundant code.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.3. Use of safe arithmetic library【Pass】

Check to see if the SafeMath security abacus library is used in the contract code implementation.

Detection results: The SafeMath security abacus library has been detected in the smart contract code and there is no such security issue.

Security advice: None.

6.4. Not recommended encoding【Pass】

Check the contract code implementation for officially uns recommended or deprecated coding methods.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.5. Reasonable use of require/assert【Pass】

Check the reasonableness of the use of require and assert statements in contract code implementations.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.6. Fallback function safety【Pass】

Check that the fallback function is used correctly in the contract code implementation.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.7. tx.origin authentication【Pass】

tx.origin is a global variable of Solidity that 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 makes contracts vulnerable to phishing-like attacks.z

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.8. Owner permission control【Pass】

Check that theowner in the contract code implementation has excessive permissions. For example, modify other account balances at will, and so on.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.9. Gas consumption detection【Pass】

Check that the consumption of gas exceeds the maximum block limit.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.10. call injection attack【Pass】

When a call function is called, strict permission control should be exercised, or the function called by call calls should be written directly to call calls.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.11. Low-level function safety【Pass】

Check the contract code implementation for security vulnerabilities in the use of call/delegatecall

The execution context of the call function is in the contract being called, while the execution context of the delegatecall function is in the contract in which the function is currently called.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.12. Vulnerability of additional token issuance【Pass】

Check to see if there are functions in the token contract that might increase the total token volume after the token total is initialized.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.13. Access control defect detection 【Pass】

Different functions in the contract should set reasonable permissions, check whether the functions in the contract correctly use pubic, private and other keywords for visibility modification, check whether the contract is properly defined and use modifier access restrictions on key functions, to avoid problems caused by overstepping the authority.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.14. Numerical overflow detection 【Pass】

The arithmetic problem in smart contracts is the integer overflow and integer overflow, with Solidity able to handle up to 256 digits (2^256-1), and a maximum number increase of 1 will overflow to get 0. Similarly, when the number is an unsigned type, 0 minus 1 overflows to get the maximum numeric value.

Integer overflows and underflows are not a new type of vulnerability, but they are particularly dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the likelihood is not anticipated, which can affect the reliability and safety of the program.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.15. Arithmetic accuracy error【Pass】

Solidity has a data structure design similar to that of a normal programming language, such as variables, constants, arrays, functions, structures, and so on, and there is a big difference between Solidity and a normal programming language - Solidity does not have floating-point patterns, and all of Solidity's numerical operations result in integers, without the occurrence of decimals, and without allowing the definition of decimal type data. Numerical operations in contracts are essential, and numerical operations are designed to cause relative errors, such as sibling operations: 5/2 x 10 x 20, and 5 x 10/2 x 25, resulting in errors, which can be greater and more obvious when the data is larger.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.16. Incorrect use of random numbers【Pass】

Random numbers may be required in smart contracts, and while the functions and variables provided by Solidity can access significantly unpredictable values, such as block.number and block.timestamp, they are usually either more public than they seem, or are influenced by miners, i.e. these random numbers are somewhat predictable, so malicious users can often copy it and rely on its unpredictability to attack the feature.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.17. Unsafe interface usage【Pass】

Check the contract code implementation for unsafe external interfaces, which can be controlled, which can cause the execution environment to be switched and control contract execution arbitrary code.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.18. Variable coverage【Pass】

Check the contract code implementation for security issues caused by variable overrides.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.19. Uninitialized storage pointer【Pass】

A special data structure is allowed in solidity as a strut structure, while local variables within the function are stored by default using stage or memory.

The existence of store (memory) and memory (memory) is two different concepts, solidity allows pointers to point to an uninitialized reference, while uninitialized local stage causes variables to point to other stored variables, resulting in variable overrides, and even more serious consequences, and should avoid initializing the task variable in the function during development.

Detection results: After detection, the smart contract code does not have the problem.

Security advice: None.

6.20. Return value call verification【Pass】

This issue occurs mostly in smart contracts related to currency transfers, so it is also known as silent failed sending or unchecked sending.

In Solidity, there are transfer methods such as transfer(), send(), call.value(), which can be used to send tokens to an address, the difference being: transfer send failure will be throw, and state rollback; Call.value returns false when it fails to send, and passing all available gas calls (which can be restricted by incoming gas_value parameters) does not effectively prevent reentration attacks.

If the return values of the send and call.value transfer functions above are not checked in the code, the contract continues to execute the subsequent code, possibly with unexpected results due to token delivery failures.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.21. Transaction order dependency【Pass】

Because miners always get gas fees through code that represents an externally owned address (EOA), users can specify higher fees to trade faster. Since blockchain is public, everyone can see the contents of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transactions at a higher cost to preempt the original solution.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.22. Timestamp dependency attack【Pass】

Block timestamps typically use miners' local time, which can fluctuate over a range of about 900 seconds, and when other nodes accept a new chunk, they only need to verify that the timestamp is later than the previous chunk and has a local time error of less than 900 seconds. A miner can profit from setting the timestamp of a block to meet as much of his condition as possible.

Check the contract code implementation for key timestamp-dependent features.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.23. Denial of service attack【Pass】

Smart contracts that are subject to this type of attack may never return to normal operation. There can be many reasons for smart contract denial of service, including malicious behavior as a transaction receiver, the exhaustion of gas caused by the artificial addition of the gas required for computing functionality, the misuse of access control to access the private component of smart contracts, the exploitation of confusion and negligence, and so on.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.24. Fake recharge vulnerability【Pass】

The transfer function of the token contract checks the balance of the transfer initiator (msg.sender) in the if way, when the balances < value enters the else logic part and return false, and ultimately does not throw an exception, we think that only if/else is a gentle way of judging in a sensitive function scenario such as transfer is a less rigorous way of coding.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.25. Reentry attack detection【Pass】

The call.value() function in Solidity consumes all the gas it receives when it is used to send tokens, and there is a risk of re-entry attacks when the call to the call tokens occurs before the balance of the sender's account is actually reduced.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.26. Replay attack detection【Pass】

If the requirements of delegate management are involved in the contract, attention should be paid to the non-reusability of validation to avoid replay attacks

In the asset management system, there are often cases of entrustment management, the principal will be the assets to the trustee management, the principal to pay a certain fee to the trustee. This business scenario is also common in smart contracts.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.27. Rearrangement attack detection【Pass】

A reflow attack is an attempt by a miner or other party to "compete" with a smart contract participant by inserting their information into a list or mapping, giving an attacker the opportunity to store their information in a contract.

Detection results: After detection, there are no related vulnerabilities in the smart contract code.

Security advice: None.

7. Appendix A: Security Assessment of Contract Fund Management

Contract fund management		
The type of asset in the contract	**The function is involved**	**Security risks**
User token assets	Deposite、takeBack、withdrawl、withdrawlRebate、withdrawlByAmt	**SAFE**
Official transfer of contract assets	Deposite、reedeemFromVenus、depositeToVenus、burn	**SAFE**

Check the security of the management of **digital currency assets** transferred by users in the business logic of the contract. Observe whether there are security risks that may cause the loss of customer funds, such as **incorrect recording, incorrect transfer, and backdoor** withdrawal of the **digital currency assets** transferred into the contract.

KNOWNSEC
Blockchain Lab

Official Website
www.knownseclab.com

E-mail
blockchain@knownsec.com

WeChat Official Account