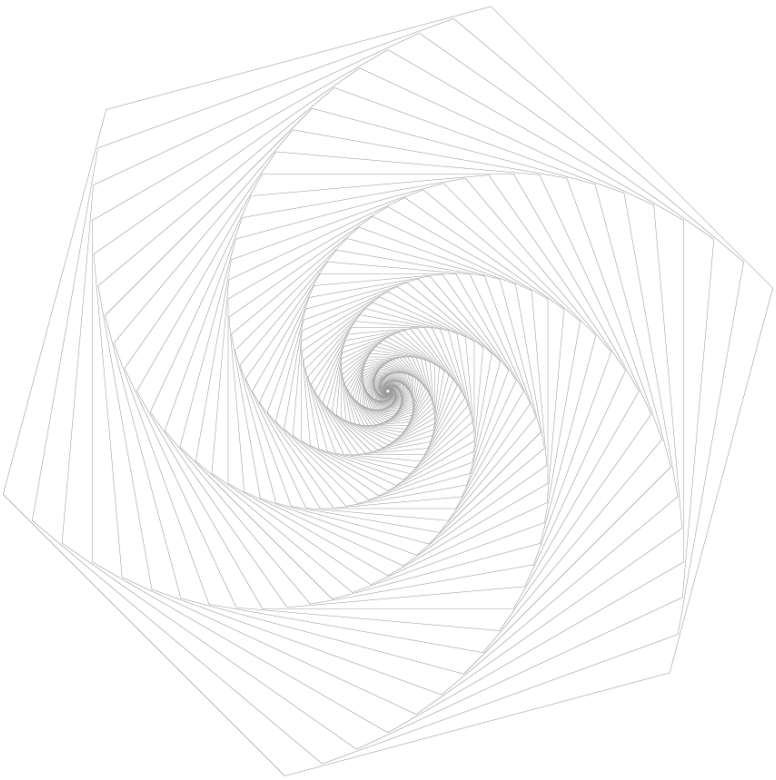
知道创宇

区块链安全实验室

智能合约审计报告



版本说明

修订内容	时间	修订者	版本号
编写文档	20220104	知道创宇区块链安全实验室	V1.0

文档信息

文档名称	文档版本	报告编号	保密级别
HPC 智能合约审计报告	V1.0	7775ca70353749e4959b1ff09126d58f	项目组公开

声明

创宇区块链安全实验室仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇区块链安全实验室无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向创宇区块链安全实验室提供的文件和资料。创宇区块链安全实验室假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇区块链安全实验室对由此而导致的损失和不利影响不承担任何责任。

目录

1. 综述 - 6 -

2. 项目信息 - 7 -

2.1. 项目描述 - 7 -

2.2. 项目官网 - 7 -

2.3. 白皮书 - 7 -

2.4. 审核版本代码 - 7 -

2.5. 合约文件及哈希/合约部署地址 - 7 -

3. 外部可见性分析 - 9 -

3.1. DataPublic 合约 - 9 -

3.2. MinerPool 合约 - 10 -

3.3. LP 合约 - 10 -

3.4. LPWallet 合约 - 11 -

3.5. Game 合约 - 12 -

3.6. GameWallet 合约 - 13 -

4. 代码漏洞分析 - 14 -

4.1. 审计结果汇总说明 - 14 -

5. 业务安全性检测 - 16 -

5.1. DataPublic 合约子地址绑定与子地址信息获取功能【通过】 - 16 -

5.2. DataPublic 合约 LpRebate 信息获取功能【通过】 - 18 -

5.3. DataPublic 合约 GameRebate 信息获取功能【通过】 - 20 -

5.4. Minerpool 合约铸币及质押奖励领取功能【通过】 - 20 -
5.5. Minerpool 合约权限管理功能【通过】 - 22 -
5.6. GameWallet.sol 减少用户余额功能【通过】 - 22 -
5.7. LP 合约取走奖励功能【通过】 - 23 -
5.8. game 合约结算功能【通过】 - 25 -
6. 代码基本漏洞检测 - 30 -
6.1. 编译器版本安全【通过】 - 30 -
6.2. 冗余代码【通过】 - 30 -
6.3. 安全算数库的使用【通过】 - 30 -
6.4. 不推荐的编码方式【通过】 - 30 -
6.5. require/assert 的合理使用【通过】 - 31 -
6.6. fallback 函数安全【通过】 - 31 -
6.7. tx.origin 身份验证【通过】 - 31 -
6.8. owner 权限控制【通过】 - 31 -
6.9. gas 消耗检测【通过】 - 32 -
6.10. call 注入攻击【通过】 - 32 -
6.11. 低级函数安全【通过】 - 32 -
6.12. 增发代币漏洞【通过】 - 32 -
6.13. 访问控制缺陷检测【通过】 - 33 -
6.14. 数值溢出检测【通过】 - 33 -
6.15. 算术精度误差【通过】 - 33 -
6.16. 错误使用随机数【通过】 - 34 -

6.17. 不安全的外部调用【通过】 - 34 -
6.18. 变量覆盖【通过】 - 34 -
6.19. 未初始化的储存指针【通过】 - 35 -
6.20. 返回值调用验证【通过】 - 35 -
6.21. 交易顺序依赖【通过】 - 36 -
6.22. 时间戳依赖攻击【通过】 - 36 -
6.23. 拒绝服务攻击【通过】 - 37 -
6.24. 假充值漏洞【通过】 - 37 -
6.25. 重入攻击检测【通过】 - 37 -
6.26. 重放攻击检测【通过】 - 38 -
6.27. 重排攻击检测【通过】 - 38 -
7. 附录 A：合约资金管理安全评估 - 39 -

1. 综述

本次报告有效测试时间是从 2021 年 12 月 29 日开始到 2022 年 1 月 4 日结束，在此期间针对 **HPC 智能合约的代币代码**安全性和规范性进行审计并以此作为报告统计依据。

本次智能合约安全审计的范围，不包含外部合约调用，不包含未来可能出现的新型攻击方式，不包含合约升级或篡改后的代码(随着项目方的发展，智能合约可能会增加新的 pool、新的功能模块，新的外部合约调用等)，不包含前端安全与服务器安全。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第六章节）以及合约具体业务安全项进行了全面的分析，**综合评定为通过**。

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

创宇存证信息:

类别	信息
报告编号	7775ca70353749e4959b1ff09126d58f
报告查询链接	https://attest.im/attestation/searchResult?qurey=7775ca70353749e4959b1ff09126d58f

2. 项目信息

2.1. 项目描述

HashPark 是在高度融合 DEFI、GameFi、NFT 三重玩法和区块链技术的去中心平台，以 Defi 收益农场和 GameFi 为核心的元宇宙项目。

2.2. 项目官网

暂无

2.3. 白皮书

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

2.4. 审核版本代码

DataPublic:

0x4b8070Ae0aB06b2ebb4837fCDcC9A9E836CDF0d7

MinerPool：

0x7C3497f9419e6a505f60025907b2adf599AAb693

Game:

0xB2f356F6CD3b6339C18ABd8c6e8B2b143205c546

LP:

0x997C77dD200205C6EE9F53169E1FC6715A56867c

2.5. 合约文件及哈希/合约部署地址

合约文件	MD5
MinerPool.sol	E5BD5FF6507E22D56D2C7705FB88FE88

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

3. 外部可见性分析

3.1. DataPublic 合约

DataPublic					
函数名	**可见性**	**状态修改**	**修饰器**	**可支付接收**	**说明**
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 合约

MinerPool					
函数名	**可见性**	**状态修改**	**修饰器**	**可支付接收**	**说明**
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 合约

LP					
函数名	**可见性**	**状态修改**	**修饰器**	**可支付接收**	**说明**
init	public	True	---	---	有检查限制
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 调用限制
reedeemFromVenus	Public	True	---	---	_mainContract|_owner 调用限制
depositeToVenus	Public	True	---	---	_mainContract|_owner 调用限制

3.4. LPWallet 合约

LPWallet					
函数名	**可见性**	**状态修改**	**修饰器**	**可支付接收**	**说明**
setMainContract	Public	True	---	---	_owner 调用限制
setVToken	Public	True	---	---	_mainContract 调用限制
approveVToken	Public	True	---	---	_mainContract 调用限制

deposite	Public	True	---	payable	_mainContract 调用限制
takeBack	Public	True	---	---	_mainContract 调用限制
reedeemFromVenus	Public	True	---	---	_mainContract|_owner 调用限制
depositeToVenus	Public	True	---	---	_mainContract|_owner 调用限制

3.5. Game 合约

Game					
函数名	**可见性**	**状态修改**	**修饰器**	**可支付接收**	**说明**
init	public	True	---	---	有检查限制
setMutiple	public	True	---	---	mgr 调用限制
setMgr	public	True	---	---	mgr 调用限制
setEnable	public	True	---	---	mgr 调用限制
caculateBlock	public	False	---	---	---
recharge	public	True	---	---	---
withdrawl	public	True	---	---	---
withdrawlByAmt	public	True	---	---	---
withdrawlRebate	public	True	---	---	---
incRebat	private	True	---	---	---

calIssue	private	True	---	---	---
bet	public	True	---	---	---
settle	private	True	---	---	---
doSettle	public	True	---	---	mgr 调用限制
doASettle	public	True	---	---	mgr 调用限制
sendFeeAndBurn	public	True	---	---	mgr 调用限制

3.6. GameWallet 合约

GameWallet					
函数名	**可见性**	**状态修改**	**修饰器**	**可支付 接收**	**说明**
addBalance	Public	True	---	---	_game 调用限制
decBalance	Public	True	---	---	_game 调用限制
withdrawlByAmt	Public	True	---	---	_game 调用限制
withdrawl	Public	True	---	---	_game 调用限制
burn	Public	True	---	---	_game 调用限制
sendRebate	Public	True	---	---	_game 调用限制
sendFee	Public	True	---	---	_game 调用限制

4. 代码漏洞分析

4.1. 审计结果汇总说明

审计结果			
审计项目	**审计内容**	**状态**	**描述**
业务安全性检测	**DataPublic 合约子地址绑定与子地址信息获取功能**	通过	经检测，不存在安全问题。
	DataPublic 合约 LpRebate 信息获取功能	通过	经检测，不存在安全问题。
	DataPublic 合约 GameRebate 信息获取功能	通过	经检测，不存在安全问题。
	Minerpool 合约铸币及质押奖励领取功能	通过	经检测，不存在安全问题。
	Minerpool 合约权限管理功能	通过	经检测，不存在安全问题。
	GameWallet.sol 减少用户余额功能	通过	经检测，不存在安全问题。
	LP 合约取走奖励功能	通过	经检测，不存在安全问题。
	game 合约结算功能	提示	经检测，不存在安全问题。
代码基本漏洞检测	**编译器版本安全**	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.origin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。

	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	不安全的外部调用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

5. 业务安全性检测

5.1. DataPublic 合约子地址绑定与子地址信息获取功能

【通过】

审计分析：对合约中的子地址检测与绑定功能进行安全审计。该功能由 getChildrenCount, getChildren, getBindRecord, bind, checkParent, bindParent, getParent 函数构成。经审计，逻辑设计合理，未发现安全问题。

```
        function getChildrenCount(address addr) public view returns (uint256) {
    //knownsec 获取子地址总数
            return _children[addr].length;
        }

        function getChildren(address addr, uint256 from) public view returns(address[20] memory) {
    //knownsec 获取子地址
            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 获取绑定清单总数
            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  获取绑定清单详情
        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  绑定父子地址对
        _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 检查并绑定父地址
        if (addr == parent) {
            return;
        }
        if (_isBindParent[addr]) {
            return;
        }
        bind(addr, parent);
    }

    function bindParent(address addr, address parent) public onlyMgr {
// knownsec 直接绑定父地址
        if (addr == parent) {
            return;
        }
        bind(addr, parent);
    }

    function getParent(address addr) public view returns (address, bool) {
// knownsec 获取父地址并返回是否绑定
        return (_parent[addr],_isBindParent[addr]);
    }
```

安全建议：无。

5.2. DataPublic 合约 LpRebate 信息获取功能【通过】

审计分析：对合约中的 LpRebate 信息获取功能进行安全审计。该功能由 onWithdrawGameRebate，getRevenuesRecord 函数构成。经审计，逻辑设计合

理，未发现安全问题。

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


            // knownsec  获取子地址 lp 回扣
            _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  获取收入记录清单
             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);
        }
```

安全建议：无。

5.3. DataPublic 合约 GameRebate 信息获取功能【通过】

审计分析：对合约中的 GameRebate 信息获取功能进行安全审计。该功能由 onWithdrawGameRebate，addGameRebate，getRebateFromChild 函数构成。经审计，逻辑设计合理，未发现安全问题。

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

        function getRebateFromChild(address parent, address child) public view returns(uint256, uint256) {
                //knownsec  获取子地址回扣记录

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


        function getRevenuesCount(address addr) public view returns(uint256) {
            //knownsec  获取收入记录总数
            return _revenuesRecord[addr].length;
        }
```

安全建议：无。

5.4. Minerpool 合约铸币及质押奖励领取功能【通过】

审计分析：对合约中的 onMint/mineOut/ developerMint 功能进行安全审计。onMint 实现了对铸币数量以及代币余额的更新，mineOut 实现了质押奖励的功能，developerMint 实现开发者铸币功能,该函数权限为 onlyMgr 属业务正常需求。

经审计，逻辑设计合理，未发现安全问题。

```
        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 代币更新

        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 质押奖励

        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 开发者铸币
```

安全建议：无。

5.5. Minerpool 合约权限管理功能【通过】

审计分析：对合约中的权限管理功能进行安全审计。函数 setMgr 通过设置地址映射的布尔值来进行管理权限的设置，该函数权限为 onlyMgr。经审计，逻辑设计合理，未发现安全问题。

```
    function setMgr(address addr, bool v) public onlyMgr {
        if (addr == _dataAddr[eAddr.owner]) { // knownsec 判断是否为管理员
            return;
        }
        if (_mgr[addr] != v) {
            _mgr[addr] = v;
        }
    }// knownsec  设置管理权限
```

安全建议：无。

5.6. GameWallet.sol 减少用户余额功能【通过】

审计分析：对 GameWallet.sol 增加用户余额功能进行安全审计，该功能由函数 decBalance 实现，decBalance 函数功能逻辑为验证调用者是否为_game，对用户余额与传参进行比较，将用户余额减去传参值 。经审计，该方法使用权限为：_game ，该功能是**由 game 对用户账户进行控制/属于业务正常需求**。

```
    function decBalance(address addr, uint256 amt) public {
```

```
        require(msg.sender == _game, "g");
        require(_balances[addr] >= amt);
        _balances[addr] = _balances[addr].sub(amt);
    }// knownsec 减少用户余额
```

安全建议：无。

5.7. LP 合约取走奖励功能【通过】

审计分析：对 LP 合约取走奖励功能进行安全审计，该功能主要由函数 withdrawReward 和 getPendingReawrd 实现，其功能逻辑为函数 getPendingReawrd 首先获取用户奖励信息，对奖励进行判断并更新 lp 信息，获取调用者推荐人信息，向推荐人添加返利，最后向调用者铸币。方法调用权限为：public，经审计，该方法为**普通用户获取奖励的功能/属于业务正常需求**。

```
    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  取走奖励

    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  获取奖励信息
```

安全建议：无。

5.8. game 合约结算功能【通过】

审计分析：对 **game** 合约结算功能进行安全审计，该功能主要由函数 settle 实现，其主要功能逻辑为截取下注信息与获奖逻辑进行对比，挑选出获奖地址并进行实际奖励收益更新，进行实际 fee 更新，最后将赌注更新，记录开奖区块。经审计，该函数使用权限为：mgr，该方法为游戏正常运行功能/**属于业务正常需求**。

```
        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  结算

        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  最后结算

        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  结算
```

安全建议：无。

6. 代码基本漏洞检测

6.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本。

检测结果：经检测，智能合约代码中制定了编译器版本 0.8.0，不存在该安全问题。

安全建议：无。

6.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库。

检测结果：经检测，智能合约代码中已使用 SafeMath 安全算数库，不存在该安全问题。

安全建议：无。

6.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.7. tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.8. owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他账户余额等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.10. call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.11. 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞

call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.12. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.13. 访问控制缺陷检测【通过】

合约中不同函数应设置合理的权限，检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.14. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢，Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.15. 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不

可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20,而5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.16. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.17. 不安全的外部调用【通过】

检查合约代码实现中是否使用了不安全的外部接口，接口可控可能导致执行环境被切换，控制合约执行任意代码。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.18. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.19. 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码不存在该问题。

安全建议：无。

6.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送代币，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继

续执行后面的代码，可能由于代币发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.21. 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.22. 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.23. 拒绝服务攻击【通过】

遭受该类型攻击的智能合约可能永远无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.24. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.25. 重入攻击检测【通过】

Solidity 中的 call.value()函数在被用来发送代币的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送代币的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.27. 重排攻击检测【通过】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表(list)或映射(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议：无。

7. 附录 A：合约资金管理安全评估

合约资金管理		
合约内资产类型	**涉及函数**	**安全风险**
用户代币资产	Deposite、takeBack、withdrawl、withdrawlRebate、withdrawlByAmt	安全
官方转移合约资产	Deposite、reedeemFromVenus、depositeToVenus、burn	安全

检查合约业务逻辑中用户转入**数字货币资产**的**管理安全**性。观察是否存在转入合约的**数字货币资产**被**错误记录、错误转出、后门提现**等易引起客户资金损失的安全风险。

知道创宇

区块链安全实验室

官方网站

www.knownseclab.com

邮箱

blockchain@knownsec.com

微信公众号