知道创宇

密级 ★ | 项目组公开

智能合约审计报告

HyperBC Farm

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20201119	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档版	文档编号	保密级别
HyperBC Farm 智能合约审计报告	V1.0	HyperBC-ZNNY-20201118	项目组公开

声明

创宇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提供的文件和资料。创宇假设: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. 代码漏洞分析 - 8 -

2.1 漏洞等级分布 - 8 -

2.2 审计结果汇总说明 - 9 -

3. 业务安全性检测 - 11 -

3.1. Governance 更新逻辑【通过】 - 11 -

3.2. nameFilter 逻辑设计【通过】 - 11 -

3.3. 添加交易池逻辑【通过】 - 14 -

3.4. 交易池点数分配【通过】 - 16 -

3.5. LP Token 迁移逻辑设计【通过】 - 16 -

3.6. LP Token 抵押逻辑设计【通过】 - 17 -

3.7. LP Token 提取逻辑设计【通过】 - 18 -

3.8. HBT 抵押逻辑设计【通过】 - 19 -

3.9. HBT 收益提取设计【通过】 - 21 -

4. 代码基本漏洞检测 - 24 -

4.1. 编译器版本安全【通过】 - 24 -

4.2. 冗余代码【通过】 - 24 -

4.3. 安全算数库的使用【通过】 - 24 -

4.4. 不推荐的编码方式【通过】 - 24 -

4.5. require/assert 的合理使用【通过】 - 25 -

4.6. fallback 函数安全【通过】 - 25 -
4.7. tx.origin 身份验证【通过】 - 25 -
4.8. owner 权限控制【通过】 - 25 -
4.9. gas 消耗检测【通过】 - 26 -
4.10. call 注入攻击【通过】 - 26 -
4.11. 低级函数安全【通过】 - 26 -
4.12. 增发代币漏洞【通过】 - 26 -
4.13. 访问控制缺陷检测【通过】 - 27 -
4.14. 数值溢出检测【通过】 - 27 -
4.15. 算术精度误差【通过】 - 28 -
4.16. 错误使用随机数【通过】 - 28 -
4.17. 不安全的接口使用【通过】 - 28 -
4.18. 变量覆盖【通过】 - 29 -
4.19. 未初始化的储存指针【通过】 - 29 -
4.20. 返回值调用验证【通过】 - 29 -
4.21. 交易顺序依赖【通过】 - 30 -
4.22. 时间戳依赖攻击【通过】 - 30 -
4.23. 拒绝服务攻击【通过】 - 31 -
4.24. 假充值漏洞【通过】 - 31 -
4.25. 重入攻击检测【通过】 - 31 -
4.26. 重放攻击检测【通过】 - 32 -
4.27. 重排攻击检测【通过】 - 32 -

5. 附录 A：合约代码 - 33 -

6. 附录 B：安全风险评级标准 - 64 -

7. 附录 C：智能合约安全审计工具简介 - 65 -

6.1 Manticore - 65 -

6.2 Oyente - 65 -

6.3 securify.sh - 65 -

6.4 Echidna - 65 -

6.5 MAIAN - 65 -

6.6 ethersplay - 66 -

6.7 ida-evm - 66 -

6.8 Remix-ide - 66 -

6.9 知道创宇区块链安全审计人员专用工具包 - 66 -

1. 综述

本次报告有效测试时间是从 2020 年 11 月 18 日开始到 2020 年 11 月 19 日结束，在此期间针对 **HyperBC Farm 智能合约代码**的安全性和规范性进行审计并以此作为报告统计依据。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，综合评定为**通过**。

本次智能合约安全审计结果：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。

本次测试的目标信息：

目标信息	
代币名称	HyperBC Farm
代码类型	以太坊智能合约
代码语言	solidity

合约文件	MD5
HBTLock.sol	83F446514B375C1EAF0527E59A248071
HBTToken.sol	A6FA92BF988D177D3B86B5D59A434D05
MasterChef.sol	77FA2CCEA4D039DE2831C7C17913D0B1
Migrations.sol	CA8D6CA8A6EDF34F149A5095A8B074C9
MockERC20.sol	5801424F9432ACD5B9CAEC7EA3A15F2E
Governance.sol	0895ADD1485AE6F106000F2A8E2C0235

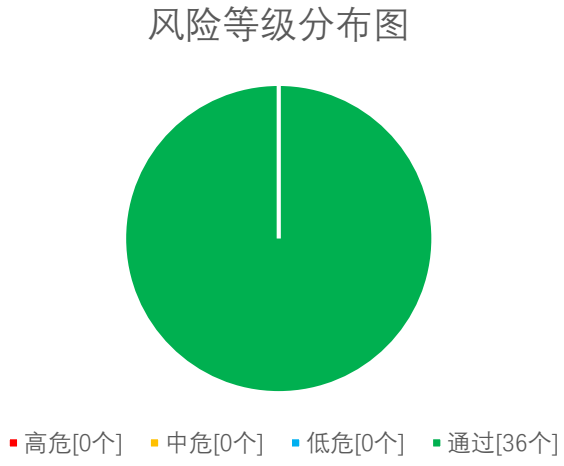
NameFilter.sol	203DC67CEA334E9975B9845A9FB6AD99
IPlayerBook.sol	72F3325B3DD7FD94D5BB08492C7F886A

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计:

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	36



2.2 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	Governance 更新逻辑	通过	经检测，不存在安全问题。
	NameFilter 逻辑设计	通过	经检测，不存在安全问题。
	添加交易池逻辑	通过	经检测，不存在安全问题。
	交易池点数分配	通过	经检测，不存在安全问题。
	LP Token 迁移逻辑	通过	经检测，不存在安全问题。
	LP Token 抵押逻辑	通过	经检测，不存在安全问题。
	LP Token 提取逻辑	通过	经检测，不存在安全问题。
	HBT 抵押逻辑设计	通过	经检测，不存在安全问题。
	HBT 收益提取逻辑	通过	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.orgin 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。
	不安全的接口使用	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。

3. 业务安全性检测

3.1. Governance 更新逻辑【通过】

对 Governance 的更新逻辑进行审计，检查在更新时是否有对权限进行校验以及对更新账户的的安全检查，例如地址的合法性等。

审计结果：Governance 更新时使用修饰器对调用者身份进行权限检查，只能由当前 Governance 设置新的 Governance，同时对新的 Governance 地址进行非空检查，之后重置 Governance。

```
modifier onlyGovernance {//knownsec//权限检查修饰器
    require(msg.sender == _governance, "not governance");
    _;
}
function setGovernance(address governance)  public  onlyGovernance
{
    require(governance != address(0), "new governance the zero address");//knownsec//地址非空检查
    emit GovernanceTransferred(_governance, governance);
    _governance = governance;//knownsec//更新 Governance
}
```

安全建议：无。

3.2. nameFilter 逻辑设计【通过】

NameFilter 用于对 name 字符串进行过滤检查，首先需要将字符串进行转小写处理，同时字符串需要满足一下条件：

1、不能以“0x”或”0X”开头

2、字符必须为 A~Z、a~z、0~9、空格

3、不能全是数字

4、不能以空格开头或结尾

5、一行中不能含有多个空格

审计结果： nameFilter 逻辑设计无误。

```
function nameFilter(string memory _input)
        internal
        pure
        returns(bytes32)
{
        bytes memory _temp = bytes(_input);
        uint256 _length = _temp.length;

        //knownsec//限制在 32 个字符之间
        require (_length <= 32 && _length > 0, "string must be between 1 and 32 characters");
        //knownsec//不能以 0x 开头
        if (_temp[0] == 0x30)
        {
            require(_temp[1] != 0x78, "string cannot start with 0x");
            require(_temp[1] != 0x58, "string cannot start with 0X");
        }

        //knwonsec//声明一个布尔值用于后期检查是否不含数字
        bool _hasNonNumber;

        //knownsec//大写转小写
        for (uint256 i = 0; i < _length; i++)
        {
            // if its uppercase A-Z
            if (_temp[i] > 0x40 && _temp[i] < 0x5b)
```

```
            {
                // convert to lower case a-z
                _temp[i] = byte(uint8(_temp[i]) + 32);

                // we have a non number
                if (_hasNonNumber == false)//knownsec//检查是否不含数字
                    _hasNonNumber = true;
            } else {
                require
                (
                    // OR lowercase a-z
                    (_temp[i] > 0x60 && _temp[i] < 0x7b) ||
                    // or 0-9
                    (_temp[i] > 0x2f && _temp[i] < 0x3a),
                    "string contains invalid characters"
                );
                // see if we have a character other than a number
                if (_hasNonNumber == false && (_temp[i] < 0x30 || _temp[i] > 0x39))
                    _hasNonNumber = true;
            }
        }
        require(_hasNonNumber == true, "string cannot be only numbers");//knownsec//检查不能不含数字
        bytes32 _ret;
        assembly {
            _ret := mload(add(_temp, 32))
        }
        return (_ret);
    }
```

安全建议：无。

3.3. 添加交易池逻辑【通过】

对添加交易池逻辑进行安全审计，检查是否有对函数调用者进行权限校验，以及添加交易池逻辑是否设计合理。

审计结果：添加交易池函数只允许合约的 owner 调用，在添加交易池时会批量更新一次当前所有交易池信息，并重新确定 startBlock，之后更新交易池信息。

```
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();//knownsec//批量更新交易池信息
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number :
startBlock;//knownsec//重设 lastRewardBlock
        totalAllocPoint = totalAllocPoint.add(_allocPoint);//knownsec//更新交易池总点数
        poolInfo.push(PoolInfo({//knownse//添加交易池信息
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accHbtPerShare: 0
        }));
    }
    function massUpdatePools() public {//knownsec//批量更新逻辑
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);//knownsec//通过 for 循环调用 updatPool 更新
        }
    }
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {//knownsec//检查氮气区块高度是否小于
```

```
等于分配发生的最后一个块号，如果大于直接 return
            return;
        }
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));//knownsec//获取当前交易池的 LP 总数
        if (lpSupply == 0) {//knownsec//如果当前 LP 总数为 0，重置 lastRewardBlock 并 return
            pool.lastRewardBlock = block.number;
            return;
        }
        uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);//kownsec//调用接口查询_from 到_to 经过了多少区块，并计算乘数
        uint256 hbtReward = multiplier.mul(hbtPerBlock).mul(pool.allocPoint).div(totalAllocPoint);//knownsec// 计 算 hbtReward
        // hbt.allowMint(devaddr, hbtReward.div(10));
        hbt.allowMint(address(this), hbtReward);
        pool.accHbtPerShare = pool.accHbtPerShare.add(hbtReward.mul(1e12).div(lpSupply));
        pool.lastRewardBlock = block.number;
    }
```

安全建议：无

3.4. 交易池点数分配【通过】

对交易池分配点数进行设置，检查是否有对权限检查交易，以及分配点数设置逻辑是否合理。

审计结果：交易池分配函数只能由合约的 owner 调用，逻辑设计合理。

```
function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();//konwnsec//批量更新池中信息
    }
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);//konwnsec//计算交易池总点数
    poolInfo[_pid].allocPoint = _allocPoint;//knownsec//设置分配点数
}
```

安全建议：无

3.5. LP Token 迁移逻辑设计【通过】

对 LP Token 迁移逻辑进行审计，检查逻辑的合理性。

审计结果：LP Token 逻辑设计合理。

```
function migrate(uint256 _pid) public {
    require(address(migrator) != address(0), "migrate: no migrator");//knownsec//地址非空检查
    PoolInfo storage pool = poolInfo[_pid];
    IERC20 lpToken = pool.lpToken;
    uint256 bal = lpToken.balanceOf(address(this));//knownsec//获取资产数量
    lpToken.safeApprove(address(migrator), bal);
    IERC20 newLpToken = migrator.migrate(lpToken);
    require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");//knownsec//检查迁移是否成功
    pool.lpToken = newLpToken;
}
```

安全建议：无

3.6. LP Token 抵押逻辑设计【通过】

对 LP Token 抵押逻辑进行审计，检查抵押逻辑设计是否合理。

审计结果：抵押逻辑设计合理。

```
    function deposit(uint256 _pid, uint256 _amount) public {
        PoolInfo storage pool = poolInfo[_pid];//knownsec//根据 pid 查找对应池地址
        UserInfo storage user = userInfo[_pid][msg.sender];//knownsec//获取用户抵押信息
        updatePool(_pid);//knownsec//更新指定池奖励信息
        if (user.amount > 0) {//knownsec//检查用户抵押数量是否大于 0
            uint256 pending = user.amount.mul(pool.accHbtPerShare).div(1e12).sub(user.rewardDebt);//knownsec//计算用户之前抵押可获得的资产数量
            // safeHbtTransfer(msg.sender, pending);
            address refer = playerBook.getPlayerLaffAddress(msg.sender);
            uint256 referRewardRate = playerBook._referRewardRate();//knownsec//计算报酬率
            uint256 baseRate = playerBook._baseRate();
            uint256 toRefer = pending.mul(referRewardRate).div(baseRate);
            // safeHbtTransfer(msg.sender, pending.sub(toRefer));
            userRewardInfo[_pid][msg.sender] = userRewardInfo[_pid][msg.sender].add(pending.sub(toRefer));//knownsec//更新用户奖励信息
            safeHbtTransfer(refer, toRefer);
            emit PlayerBookEvent(refer, msg.sender, toRefer);
        }
        pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
        user.amount = user.amount.add(_amount);//knownsec//更新抵押总量
        user.rewardDebt = user.amount.mul(pool.accHbtPerShare).div(1e12);//knownsec//更新债务
        emit Deposit(msg.sender, _pid, _amount);
    }
```

安全建议：无。

3.7. LP Token 提取逻辑设计【通过】

对 LP Token 提取逻辑进行安全审计，检查是否有对提取数量进行检查以及提取后的更新逻辑是否设计合理等。

审计结果：LP Token 提取逻辑设计无误。

```
function withdraw(uint256 _pid, uint256 _amount) public {
        PoolInfo storage pool = poolInfo[_pid];//knownsec//根据 pid 查找 pool
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");//knownsec//检查提取数量是否小于用户当前数量
        updatePool(_pid);//knownsec//更新池信息
        uint256 pending = user.amount.mul(pool.accHbtPerShare).div(1e12).sub(user.rewardDebt);
        address refer = playerBook.getPlayerLaffAddress(msg.sender);
        uint256 referRewardRate = playerBook._referRewardRate();
        uint256 baseRate = playerBook._baseRate();
        uint256 toRefer = pending.mul(referRewardRate).div(baseRate);
        // safeHbtTransfer(msg.sender, pending.sub(toRefer));
        userRewardInfo[_pid][msg.sender] = userRewardInfo[_pid][msg.sender].add(pending.sub(toRefer));
        safeHbtTransfer(refer, toRefer);
```

```
        emit PlayerBookEvent(refer, msg.sender, toRefer);
        user.amount = user.amount.sub(_amount);//knownsec/更新用户资产数量
        user.rewardDebt = user.amount.mul(pool.accHbtPerShare).div(1e12);
        if(_amount > 0){
            pool.lpToken.safeTransfer(address(msg.sender), _amount);//knownsec//提取操作
            emit Withdraw(msg.sender, _pid, _amount);
        }
    }
```

安全建议：无

3.8. HBT 抵押逻辑设计【通过】

对 HBT 抵押逻辑进行安全审计，检查抵押逻辑的设计是否合理。

审计结果：HBT 抵押逻辑设计无误。

```
    function disposit(address _address,uint256 _number, uint256 _times) public returns (bool) {
        require(_number > 0, "HBTLock:disposit _number Less than zero");//knownsec//对抵押数量进行检查
        require(times[_times] > 0, "HBTLock:disposit _times Less than zero");//knownsec//抵押次数
        require(msg.sender == masterChef, "HBTLock:msg.sender Not equal to masterChef");//knownsec//调用者身份检查
        require(depositCountTotal > userInfo[_address].depositCount, "HBTLock: The maximum mortgage times have been exceeded");
        require(close == false, "HBTLock: The contract has been closed ");//knownsec//合约状态检查

        uint256 _endBlockTime = times[_times];
```

```
            timesAwardTotal = timesAwardTotal.add(_number.mul(_times).div(10)).sub(_number);
            depositTotal = depositTotal.add(_number);//knownsec//更新抵押数量

            userInfo[_address].timesAward =
userInfo[_address].timesAward.add(_number.mul(_times).div(10).sub(_number));
            userInfo[_address].deposit = userInfo[_address].deposit.add(_number);
            userInfo[_address].depositCount = userInfo[_address].depositCount.add(1);//knownsec//
更新抵押次数
            uint256 _endBlock =
_endBlockTime.mul(1e12).div(blockTime).div(1e12).add(block.number); //knownsec//结束时间
            uint256 index;
            bool isNew;
            (index,isNew) =  newDepositInfoMode(_address);//knownsec//查询是否新增锁定记
录
            if(isNew == true){
                depositInfo[_address].push(DepositInfo({//knownsec//新增抵押记录
                    endBlock: _endBlock,
                    number: _number,
                    times: _times
                }));
            }else{//knownsec//更新抵押信息
                depositInfo[_address][index].endBlock = _endBlock;
                depositInfo[_address][index].number = _number;
                depositInfo[_address][index].times = _times;
            }
            return true;
        }
```

安全建议：无

3.9. HBT 收益提取设计【通过】

对提取收益逻辑进行审计，检查在提取收益时是否有对提取数量以及提取是是否处于锁定状态进行安全检查。

审计结果：未发现设计缺陷。

```
    function    withdraw() public {

        uint256 unlockNumber;
        uint256 unlockDispositNumber;
        address _address = address(msg.sender);

        ( unlockNumber, unlockDispositNumber) = unlockInfoOpt(_address);//knownsec//获取可解锁的数量
        require(unlockNumber > 0 , "HBTLock: unlock number Less than zero");//knownsec//对解锁的数量做检查

        hbtSafe.safeTransfer(_address,unlockNumber);//knownsec//提取收益
        // hbtSafe.safeTransferFrom(address(this),_address,unlockNumber);

        pickDepositTotal = pickDepositTotal.add(unlockDispositNumber);//knownsec//计算解锁抵押总量
        pickTimesAwardTotal = pickTimesAwardTotal.add(unlockNumber.sub(unlockDispositNumber));//knownsec//计算解锁倍数奖励

        userInfo[_address].pickDeposit = userInfo[_address].pickDeposit.add(unlockDispositNumber);//knownsec//更新解锁抵押总量
        userInfo[_address].pickTimesAward = userInfo[_address].pickTimesAward.add(unlockNumber.sub(unlockDispositNumber));//knownsec//更新解锁倍数奖励
        emit Withdraw(msg.sender, unlockNumber);
```

```
    }
        function unlockInfoOpt(address _address) private returns (uint256, uint256) {//knownsec//获取可解锁数量
            uint256 _blcokNumber = block.number;
            uint256 length = depositInfo[_address].length;

            uint256 unlockNumber = 0;
            uint256 unlockDispositNumber = 0;
            for (uint256 id = 0; id < length; ++id) {
                if(depositInfo[_address][id].endBlock < _blcokNumber && depositInfo[_address][id].endBlock != 0) {
                    unlockNumber = unlockNumber.add(depositInfo[_address][id].number.mul(depositInfo[_address][id].times).div(10));
                    unlockDispositNumber = unlockDispositNumber.add(depositInfo[_address][id].number);

                    depositInfo[_address][id].endBlock = 0;
                    depositInfo[_address][id].number = 0;
                    depositInfo[_address][id].times = 0;

                    userInfo[_address].depositCount = userInfo[_address].depositCount.sub(1);
                }
            }

            return (unlockNumber,unlockDispositNumber);
        }

        //提取收益&延时提取
        function extractReward(uint256 _pid, uint256 _times, bool _profitLock) public {

            withdraw(_pid,0);
```

```
        // PoolInfo storage pool = poolInfo[_pid];
        // UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 pending = userRewardInfo[_pid][msg.sender];

        if (_profitLock == false) {
            safeHbtTransfer(msg.sender, pending);
            emit ExtractReward(msg.sender, _pid, pending);
        } else {
            uint256 _pendingTimes = pending.mul(_times).div(10);
            hbt.allowMint(address(this), _pendingTimes.sub(pending));

            safeHbtTransfer(address(hbtLock), _pendingTimes);
            hbtLock.disposit(msg.sender,pending,_times);
            emit ProfitLock(msg.sender, _pid, pending, _times);
        }
        userRewardInfo[_pid][msg.sender] = 0;
    }
```

安全建议：无

4. 代码基本漏洞检测

4.1. 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 0.5.15 以上，不存在该安全问题。

安全建议：无。

4.2. 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。

4.3. 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。

4.4. 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。

4.5. 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。

4.6. 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。

4.7. 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。

4.8. 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。

4.9. 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。

4.10. 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，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.11. 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。

4.12. 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，智能合约代码中存在增发代币的功能，但由于流动性挖矿需要增发代币，故通过。

安全建议：无。

4.13. 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。

4.14. 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。

4.15. 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、函数、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。

4.16. 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。

4.17. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

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。

4.18. 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。

4.19. 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，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

4.20. 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 Ether，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 Ether 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。

4.21. 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。由于以太坊区块链是公开的，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。

4.22. 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。

4.23. 拒绝服务攻击【通过】

在以太坊的世界中，拒绝服务是致命的，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。

4.24. 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。

4.25. 重入攻击检测【通过】

重入漏洞是最著名的以太坊智能合约漏洞，曾导致了以太坊的分叉（The DAO hack）。

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 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。

4.26. 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，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.27. 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。

5. 附录 A：合约代码

本次测试代码来源:

IPlayerBook.sol

```
pragma solidity    0.6.12;
interface IPlayerBook {
    function settleReward( address from,uint256 amount ) external returns (uint256);
    function bindRefer( address from,string calldata    affCode ) external    returns (bool);
    function hasRefer(address from)    external returns(bool);
    function getPlayerLaffAddress(address from) external returns(address);

}
```

Governance.sol

```
pragma solidity    0.6.12;
contract Governance {
    address public _governance;
    constructor() public {
        _governance = tx.origin;
    }
    event GovernanceTransferred(address indexed previousOwner, address indexed newOwner);
    modifier onlyGovernance {
        require(msg.sender == _governance, "not governance");
        _;
    }
    function setGovernance(address governance)    public    onlyGovernance
    {
        require(governance != address(0), "new governance the zero address");
        emit GovernanceTransferred(_governance, governance);
        _governance = governance;
    }
}
```

NameFilter.sol

```
pragma solidity    0.6.12;
library NameFilter {
    /**
     * @dev filters name strings
     * -converts uppercase to lower case.
     * -makes sure it does not start/end with a space
     * -makes sure it does not contain multiple spaces in a row
     * -cannot be only numbers
     * -cannot start with 0x
     * -restricts characters to A-Z, a-z, 0-9, and space.
     * @return reprocessed string in bytes32 format
     */
    function nameFilter(string memory _input)
        internal
        pure
        returns(bytes32)
    {
        bytes memory _temp = bytes(_input);
        uint256 _length = _temp.length;

        //sorry limited to 32 characters
        require (_length <= 32 && _length > 0, "string must be between 1 and 32 characters");
        // make sure first two characters are not 0x
        if (_temp[0] == 0x30)
        {
            require(_temp[1] != 0x78, "string cannot start with 0x");
            require(_temp[1] != 0x58, "string cannot start with 0X");
        }

        // create a bool to track if we have a non number character
        bool _hasNonNumber;
```

```
// convert & check
for (uint256 i = 0; i < _length; i++)
{
    // if its uppercase A-Z
    if (_temp[i] > 0x40 && _temp[i] < 0x5b)
    {
        // convert to lower case a-z
        _temp[i] = byte(uint8(_temp[i]) + 32);

        // we have a non number
        if (_hasNonNumber == false)
            _hasNonNumber = true;
    } else {
        require
        (
            // OR lowercase a-z
            (_temp[i] > 0x60 && _temp[i] < 0x7b) ||
            // or 0-9
            (_temp[i] > 0x2f && _temp[i] < 0x3a),
            "string contains invalid characters"
        );

        // see if we have a character other than a number
        if (_hasNonNumber == false && (_temp[i] < 0x30 || _temp[i] > 0x39))
            _hasNonNumber = true;
    }
}

require(_hasNonNumber == true, "string cannot be only numbers");

bytes32 _ret;
assembly {
    _ret := mload(add(_temp, 32))
```

```
            }
            return (_ret);
        }
    }
```

HBTLock.sol

```
pragma solidity 0.6.12;
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/access/Ownable.sol";


contract HBTLock is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    uint256 public blockTime = 15;          //出块时长
    uint256 public timesAwardTotal;         //倍数总奖励
    uint256 public depositTotal;            //抵押总数量
    uint256 public pickDepositTotal;        //已解锁数量
    uint256 public pickTimesAwardTotal;     //已解锁倍数奖励

    address public masterChef;
    IERC20 public hbtSafe;
    uint256 public depositCountTotal = 20;     //用户最大抵押次数

    //锁定记录 struct
    struct DepositInfo {
        uint256 endBlock;       //抵押结束区块号
        uint256 number;         //抵押数量
        uint256 times;          //倍数
    }
    mapping (address => DepositInfo[]) public depositInfo;      //锁定记录
```

```
//用户信息
struct UserInfo {
    uint256 timesAward;          //倍数奖励
    uint256 deposit;             //抵押数量
    uint256 pickDeposit;         //已解锁数量
    uint256 pickTimesAward;      //已解锁倍数奖励
    uint256 depositCount;        //抵押次数
}
mapping (address => UserInfo) public userInfo;      //用户记录

//times
mapping (uint256 => uint256) times;

constructor(
    IERC20 _hbt              //HBT Token 合约地址
) public {
    hbtSafe = _hbt;

    times[12] = 300;
    times[15] = 600;
    times[25] = 1200;
}

bool public close = false;

event Withdraw(address indexed user,uint256 unlockNumber);

//masterChef
function setMasterChef(address _address) public    onlyOwner {
    masterChef = _address;
}
```

```
    //close hbtlock
    function setClose(bool _bool) public    onlyOwner {
        close = _bool;
    }

    //查询新增锁定记录方式
    function newDepositInfoMode(address _address) public view returns(uint256,bool) {
        uint256 length = depositInfo[_address].length;
        if (length == 0 ){
            return (0,true);
        }
        uint256 index = 0;
        bool isNew = true;

        for (uint256 id = 0; id < length; id++) {
            if(depositInfo[_address][id].number == 0){
                index = id;
                isNew = false;
                break;
            }
        }
        return (index,isNew);
    }

    //抵押
    function disposit(address _address,uint256 _number, uint256 _times) public returns (bool) {
        require(_number > 0, "HBTLock:disposit _number Less than zero");
        require(times[_times] > 0, "HBTLock:disposit _times Less than zero");
        require(msg.sender == masterChef, "HBTLock:msg.sender Not equal to masterChef");
        require(depositCountTotal > userInfo[_address].depositCount, "HBTLock: The
maximum mortgage times have been exceeded");
        require(close == false, "HBTLock: The contract has been closed ");
```

```
        uint256 _endBlockTime = times[_times];
        timesAwardTotal = timesAwardTotal.add(_number.mul(_times).div(10)).sub(_number);
        depositTotal = depositTotal.add(_number);

        userInfo[_address].timesAward =
userInfo[_address].timesAward.add(_number.mul(_times).div(10).sub(_number));
        userInfo[_address].deposit = userInfo[_address].deposit.add(_number);

        userInfo[_address].depositCount = userInfo[_address].depositCount.add(1);

        uint256 _endBlock =
_endBlockTime.mul(1e12).div(blockTime).div(1e12).add(block.number); //结束时间

        uint256 index;
        bool isNew;

        (index,isNew) =  newDepositInfoMode(_address);

        if(isNew == true){
            depositInfo[_address].push(DepositInfo({
                endBlock: _endBlock,
                number: _number,
                times: _times
            }));
        }else{
            depositInfo[_address][index].endBlock = _endBlock;
            depositInfo[_address][index].number = _number;
            depositInfo[_address][index].times = _times;
        }


        return true;
```

```
    }

    //可解锁数量
    function unlockInfo(address _address) public view returns (uint256, uint256) {
        uint256 _blcokNumber = block.number;
        uint256 length = depositInfo[_address].length;
        if(length == 0){
            return (0,0);
        }

        uint256 unlockNumber = 0;
        uint256 unlockDispositNumber = 0;
        for (uint256 id = 0; id < length; ++id) {
            if(depositInfo[_address][id].endBlock < _blcokNumber) {
                unlockNumber =
unlockNumber.add(depositInfo[_address][id].number.mul(depositInfo[_address][id].times).div(10
));
                unlockDispositNumber =
unlockDispositNumber.add(depositInfo[_address][id].number);
            }
        }
        return (unlockNumber,unlockDispositNumber);
    }

    //获取可解锁数量,将符合的记录重置成
    function unlockInfoOpt(address _address) private  returns (uint256, uint256) {
        uint256 _blcokNumber = block.number;
        uint256 length = depositInfo[_address].length;

        uint256 unlockNumber = 0;
        uint256 unlockDispositNumber = 0;
        for (uint256 id = 0; id < length; ++id) {
            if(depositInfo[_address][id].endBlock < _blcokNumber &&
```

```
depositInfo[_address][id].endBlock != 0) {
                    unlockNumber                                    =
unlockNumber.add(depositInfo[_address][id].number.mul(depositInfo[_address][id].times).div(10
));
                    unlockDispositNumber                            =
unlockDispositNumber.add(depositInfo[_address][id].number);

                    depositInfo[_address][id].endBlock = 0;
                    depositInfo[_address][id].number = 0;
                    depositInfo[_address][id].times = 0;

                    userInfo[_address].depositCount = userInfo[_address].depositCount.sub(1);
                }
            }

            return (unlockNumber,unlockDispositNumber);
        }
        //提取收益
        function    withdraw() public {

            uint256 unlockNumber;
            uint256 unlockDispositNumber;
            address _address = address(msg.sender);

            ( unlockNumber, unlockDispositNumber) = unlockInfoOpt(_address);
            require(unlockNumber > 0 , "HBTLock: unlock number Less than zero");


            hbtSafe.safeTransfer(_address,unlockNumber);
            // hbtSafe.safeTransferFrom(address(this),_address,unlockNumber);

            pickDepositTotal = pickDepositTotal.add(unlockDispositNumber);
            pickTimesAwardTotal                                     =
```

```
pickTimesAwardTotal.add(unlockNumber.sub(unlockDispositNumber));

            userInfo[_address].pickDeposit =
userInfo[_address].pickDeposit.add(unlockDispositNumber);
            userInfo[_address].pickTimesAward =
userInfo[_address].pickTimesAward.add(unlockNumber.sub(unlockDispositNumber));
            emit Withdraw(msg.sender, unlockNumber);
        }

}
```

HBTToken.sol

```
pragma solidity 0.6.12;


import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";


// SushiToken with Governance.
contract HBTToken is ERC20("HBTToken", "HBT"), Ownable {


    mapping (address => bool) public allowMintAddr; //铸币白名单
    uint256 private _capacity;

    constructor () public {
        _capacity = 100000000000000000000000000;
    }

    //最大发行量
    function capacity() public view    returns (uint256) {
```

```
        return _capacity;
    }

    //设置白名单
    function setAllowMintAddr(address _address,bool _bool) public onlyOwner {
        allowMintAddr[_address] = _bool;
    }
    //获取白名单
    function allowMintAddrInfo(address _address) external view    returns(bool) {
        return allowMintAddr[_address];
    }

    /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
    function mint(address _to, uint256 _amount) public onlyOwner {
        require(  totalSupply().add(_amount)  <=  _capacity,  "ERC20:  Maximum  capacity
exceeded");

        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _amount);
    }

    //白名单铸币
    function allowMint(address _to, uint256 _amount) public {
        require(allowMintAddr[msg.sender],"HBT:The address is not in the allowed range");
        require(  totalSupply().add(_amount)  <=  _capacity,  "ERC20:  Maximum  capacity
exceeded");

        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _amount);
    }

    // Copied and modified from YAM code:
    //                                                    https://github.com/yam-finance/yam-
```

```
protocol/blob/master/contracts/token/YAMGovernanceStorage.sol
    //                                                  https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernance.sol
    // Which is copied and modified from COMPOUND:
    //                                                  https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/Comp.sol

    /// @notice A record of each accounts delegate
    mapping (address => address) internal _delegates;

    /// @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    /// @notice A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

    /// @notice The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints;

    /// @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

    /// @notice The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    /// @notice A record of states for signing / validating signatures
    mapping (address => uint) public nonces;
```

```
    /// @notice An event thats emitted when an account changes its delegate
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

    /// @notice An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
     * @param delegator The address to get delegatee for
     */
    function delegates(address delegator)
        external
        view
        returns (address)
    {
        return _delegates[delegator];
    }

   /**
    * @notice Delegate votes from `msg.sender` to `delegatee`
    * @param delegatee The address to delegate votes to
    */
    function delegate(address delegatee) external {
        return _delegate(msg.sender, delegatee);
    }

    /**
     * @notice Delegates votes from signatory to `delegatee`
     * @param delegatee The address to delegate votes to
     * @param nonce The contract state required to match the signature
     * @param expiry The time at which to expire the signature
```

```
    * @param v The recovery byte of the signature
    * @param r Half of the ECDSA signature pair
    * @param s Half of the ECDSA signature pair
    */
function delegateBySig(
        address delegatee,
        uint nonce,
        uint expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
)
        external
{
        bytes32 domainSeparator = keccak256(
                abi.encode(
                        DOMAIN_TYPEHASH,
                        keccak256(bytes(name())),
                        getChainId(),
                        address(this)
                )
        );

        bytes32 structHash = keccak256(
                abi.encode(
                        DELEGATION_TYPEHASH,
                        delegatee,
                        nonce,
                        expiry
                )
        );

        bytes32 digest = keccak256(
```

```
            abi.encodePacked(
                "\x19\x01",
                domainSeparator,
                structHash
            )
        );

        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "HBT::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "HBT::delegateBySig: invalid nonce");
        require(now <= expiry, "HBT::delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
    }

    /**
     * @notice Gets the current votes balance for `account`
     * @param account The address to get votes balance
     * @return The number of current votes for `account`
     */
    function getCurrentVotes(address account)
        external
        view
        returns (uint256)
    {
        uint32 nCheckpoints = numCheckpoints[account];
        return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
    }

    /**
     * @notice Determine the prior number of votes for an account as of a block number
     * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
     * @param account The address of the account to check
```

```
    * @param blockNumber The block number to get the vote balance at
    * @return The number of votes the account had as of the given block
    */
    function getPriorVotes(address account, uint blockNumber)
        external
        view
        returns (uint256)
    {
        require(blockNumber < block.number, "HBT::getPriorVotes: not yet determined");

        uint32 nCheckpoints = numCheckpoints[account];
        if (nCheckpoints == 0) {
            return 0;
        }

        // First check most recent balance
        if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
            return checkpoints[account][nCheckpoints - 1].votes;
        }

        // Next check implicit zero balance
        if (checkpoints[account][0].fromBlock > blockNumber) {
            return 0;
        }

        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
            uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
            Checkpoint memory cp = checkpoints[account][center];
            if (cp.fromBlock == blockNumber) {
                return cp.votes;
            } else if (cp.fromBlock < blockNumber) {
```

```
                lower = center;
            } else {
                upper = center - 1;
            }
        }
        return checkpoints[account][lower].votes;
    }

    function _delegate(address delegator, address delegatee)
        internal
    {
        address currentDelegate = _delegates[delegator];
        uint256 delegatorBalance = balanceOf(delegator); // balance of underlying HBTs (not scaled);
        _delegates[delegator] = delegatee;

        emit DelegateChanged(delegator, currentDelegate, delegatee);

        _moveDelegates(currentDelegate, delegatee, delegatorBalance);
    }

    function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
        if (srcRep != dstRep && amount > 0) {
            if (srcRep != address(0)) {
                // decrease old representative
                uint32 srcRepNum = numCheckpoints[srcRep];
                uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
                uint256 srcRepNew = srcRepOld.sub(amount);
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
            }

            if (dstRep != address(0)) {
```

```
                // increase new representative
                uint32 dstRepNum = numCheckpoints[dstRep];
                uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
                uint256 dstRepNew = dstRepOld.add(amount);
                _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
            }
        }
    }

    function _writeCheckpoint(
        address delegatee,
        uint32 nCheckpoints,
        uint256 oldVotes,
        uint256 newVotes
    )
        internal
    {
        uint32 blockNumber = safe32(block.number, "HBT::_writeCheckpoint: block number exceeds 32 bits");

        if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
            checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
        } else {
            checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
            numCheckpoints[delegatee] = nCheckpoints + 1;
        }

        emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
    }

    function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
```

```
        require(n < 2**32, errorMessage);
        return uint32(n);
    }

    function getChainId() internal pure returns (uint) {
        uint256 chainId;
        assembly { chainId := chainid() }
        return chainId;
    }


}
```

MasterChef.sol

```
pragma solidity 0.6.12;


import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "./PlayerBook.sol";
import "./HBTToken.sol";
import "./HBTLock.sol";


interface IMigratorChef {
    // Perform LP token migration from legacy UniswapV2 to SushiSwap.
    // Take the current LP token address and return the new LP token address.
    // Migrator should have full access to the caller's LP token.
    // Return the new LP token address.
```

```
    //
    // XXX Migrator must have allowance access to UniswapV2 LP tokens.
    // SushiSwap must mint EXACTLY the same amount of SushiSwap LP tokens or
    // else something bad will happen. Traditional UniswapV2 does not
    // do that so be careful!
    function migrate(IERC20 token) external returns (IERC20);
}

// MasterChef is the master of Hbt. He can make Hbt and he is a fair guy.
//
// Note that it's ownable and the owner wields tremendous power. The ownership
// will be transferred to a governance smart contract once HBT is sufficiently
// distributed and the community can show to govern itself.
//
// Have fun reading it. Hopefully it's bug-free. God bless.
contract MasterChef is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    // Info of each user.
    struct UserInfo {
        uint256 amount;         // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt. See explanation below.          奖励的债务
        //
        // We do some fancy math here. Basically, any point in time, the amount of HBTs
        // entitled to a user but is pending to be distributed is:
        //
        //     pending reward = (user.amount * pool.accHbtPerShare) - user.rewardDebt
        //
        // Whenever a user deposits or withdraws LP tokens to a pool. Here's what happens:
        //     1. The pool's `accHbtPerShare` (and `lastRewardBlock`) gets updated.
        //     2. User receives the pending reward sent to his/her address.
        //     3. User's `amount` gets updated.
```

```
        //      4. User's `rewardDebt` gets updated.
    }

    // Info of each pool.
    struct PoolInfo {
        IERC20 lpToken;             // Address of LP token contract.   LP token 合约地址.
        uint256 allocPoint;         // How many allocation points assigned to this pool. HBTs to distribute per block.   分配给该池的分配点数
        uint256 lastRewardBlock;    // Last block number that HBTs distribution occurs.     YMI 分配发生的最后一个块号。
        uint256 accHbtPerShare; // Accumulated HBTs per share, times 1e12. See below.   每股累计的 YMI
    }

    // The HBT TOKEN!
    HBTToken public hbt;
    // The HBTLock Contract.
    HBTLock public hbtLock;
    // Dev address.
    // address public devaddr;
    // Block number when bonus HBT period ends.
    uint256 public bonusEndBlock;
    // HBT tokens created per block.
    uint256 public hbtPerBlock;
    // Bonus muliplier for early hbt makers.
    uint256 public constant BONUS_MULTIPLIER = 1;
    // The migrator contract. It has a lot of power. Can only be set through governance (owner).
    IMigratorChef public migrator;

    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping (uint256 => mapping (address => UserInfo)) public userInfo;
```

```
// Total allocation poitns. Must be the sum of all allocation points in all pools.
uint256 public totalAllocPoint = 0;
// The block number when HBT mining starts.
uint256 public startBlock;

mapping (uint256 => mapping (address => uint256)) public userRewardInfo;

PlayerBook public playerBook;
event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);
event ProfitLock(address indexed user, uint256 indexed pid, uint256 pt, uint256 times);
event ExtractReward(address indexed user, uint256 indexed pid, uint256 amount);
event PlayerBookEvent(address indexed user, address indexed fromUser, uint256 amount);


constructor(
    HBTToken _hbt, //HBT Token 合约地址
    HBTLock _hbtLock, //HBTLock  合约地址
    uint256 _hbtPerBlock, //每个块产生的 HBT Token 的数量
    uint256 _startBlock,   //开挖 HBT 的区块高度
    uint256 _bonusEndBlock, //HBT 倍数结束块
    address payable _playerBook
) public {
    hbt = _hbt;
    hbtLock = _hbtLock;
    hbtPerBlock = _hbtPerBlock;
    bonusEndBlock = _bonusEndBlock;
    startBlock = _startBlock;

    playerBook = PlayerBook(_playerBook);
}
```

```
    function poolLength() external view returns (uint256) {
        return poolInfo.length;
    }

    function getUserRewardInfo(uint256 _pid,address _address) external view returns (uint256) {
        return userRewardInfo[_pid][_address];
    }

    // Add a new lp to the pool. Can only be called by the owner.
    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
    //添加新的LP 交易池， 仅合约拥有者可以调用，注意，不能添加相同地址的LP 交易池
    //param: _allocPoint, 分配的点数(即每个池的占比为：当前分配点数 / 总点数)
    //param: _lpToken, LP Token 合约的地址
    //param: _withUpdate, 是否更新交易池（备注：查询 sushi 的交易，一般都是传 true）
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accHbtPerShare: 0
        }));
    }

    // Update the given pool's HBT allocation point. Can only be called by the owner.
    //设置交易池的分配点数, 仅合约拥有者可以调用
    //param： _pid， pool id (即通过 pool id 可以找到对应池的的地址)
    //param：_allocPoint， 新的分配点数
```

```
    //param: _withUpdate, 是否更新交易池（备注：查询 sushi 的交易，一般都是传 true）
    function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // Set the migrator contract. Can only be called by the owner.
    //设置迁移合约,  仅合约拥有者可以调用
    //param：_migrator，迁移合约的地址
    function setMigrator(IMigratorChef _migrator) public onlyOwner {
        migrator = _migrator;
    }

    // Migrate lp token to another lp contract. Can be called by anyone. We trust that migrator contract is good.
    //将 lp token 迁移到另一个 lp token，需要谨慎操作
    //param：_pid，  pool id (即通过 pool id  可以找到对应池的的地址)
    function migrate(uint256 _pid) public {
        require(address(migrator) != address(0), "migrate: no migrator");
        PoolInfo storage pool = poolInfo[_pid];
        IERC20 lpToken = pool.lpToken;
        uint256 bal = lpToken.balanceOf(address(this));
        lpToken.safeApprove(address(migrator), bal);
        IERC20 newLpToken = migrator.migrate(lpToken);
        require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");
        pool.lpToken = newLpToken;
    }

    // Return reward multiplier over the given _from to _to block.
    //查询接口，  获取_from 到_to 区块之间过了多少区块，并计算乘数
```

```
//param：_from from  区块高度
//param：_to to  区块高度
// function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
//         if (_to <= bonusEndBlock) {
//             return _to.sub(_from).mul(BONUS_MULTIPLIER);
//         } else if (_from <= bonusEndBlock) {
//             return bonusEndBlock.sub(_from);
//         } else {
//             return 0;
//         }
// }
    // Return reward multiplier over the given _from to _to block.
function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
    if (_to <= bonusEndBlock) {
        return _to.sub(_from).mul(BONUS_MULTIPLIER);
    } else if (_from >= bonusEndBlock) {
        return _to.sub(_from);
    } else {
        return bonusEndBlock.sub(_from).mul(BONUS_MULTIPLIER).add(
            _to.sub(bonusEndBlock)
        );
    }
}

// View function to see pending HBTs on frontend.
//查询接口，查询当前阶段指定地址_user 在_pid 池中赚取的 YMI
//param：_pid，   pool id (即通过 pool id  可以找到对应池的的地址)
//param：_user，  用户地址
function pendingHbt(uint256 _pid, address _user) public view returns (uint256) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];
    uint256 accHbtPerShare = pool.accHbtPerShare;
    uint256 lpSupply = pool.lpToken.balanceOf(address(this));
```

```
        if (block.number > pool.lastRewardBlock && lpSupply != 0) {
            uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
            uint256 hbtReward =
multiplier.mul(hbtPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
            accHbtPerShare = accHbtPerShare.add(hbtReward.mul(1e12).div(lpSupply));
        }
        return user.amount.mul(accHbtPerShare).div(1e12).sub(user.rewardDebt);
    }

    //前端页面查询接口，扣除返佣
    function pendingHbtShow(uint256 _pid, address _user) external view returns (uint256) {

        uint256 pending = pendingHbt(_pid,_user);
        uint256 baseRate = playerBook._baseRate();
        uint256 referRewardRate = playerBook._referRewardRate();
        uint256 toRefer = pending.mul(referRewardRate).div(baseRate);
        return pending.sub(toRefer).add(userRewardInfo[_pid][_user]);
    }

    // Update reward vairables for all pools. Be careful of gas spending!
    //更新所有池的奖励等信息
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    //更新将指定池奖励等信息
    //param：_pid，  pool id (即通过 pool id  可以找到对应池的的地址)
    function updatePool(uint256 _pid) public {
```

```
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (lpSupply == 0) {
            pool.lastRewardBlock = block.number;
            return;
        }
        uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
        uint256 hbtReward = multiplier.mul(hbtPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
        // hbt.allowMint(devaddr, hbtReward.div(10));
        hbt.allowMint(address(this), hbtReward);
        pool.accHbtPerShare = pool.accHbtPerShare.add(hbtReward.mul(1e12).div(lpSupply));
        pool.lastRewardBlock = block.number;
    }

    // Deposit LP tokens to MasterChef for HBT allocation.
    //抵押 LP toekn 进行挖矿获取 YMI（抵押前，当前操作地址需要先在对应的 LP toekn 合约进行授权给 MasterChef 合约）
    //param：_pid， pool id (即通过 pool id 可以找到对应池的的地址)
    //param：_amount, 抵押的金额
    function deposit(uint256 _pid, uint256 _amount) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pending = user.amount.mul(pool.accHbtPerShare).div(1e12).sub(user.rewardDebt);
            // safeHbtTransfer(msg.sender, pending);
            address refer = playerBook.getPlayerLaffAddress(msg.sender);
            uint256 referRewardRate = playerBook._referRewardRate();
```

```
            uint256 baseRate = playerBook._baseRate();
            uint256 toRefer = pending.mul(referRewardRate).div(baseRate);
            // safeHbtTransfer(msg.sender, pending.sub(toRefer));
            userRewardInfo[_pid][msg.sender] =
userRewardInfo[_pid][msg.sender].add(pending.sub(toRefer));
            safeHbtTransfer(refer, toRefer);
            emit PlayerBookEvent(refer, msg.sender, toRefer);
        }
        pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
        user.amount = user.amount.add(_amount);
        user.rewardDebt = user.amount.mul(pool.accHbtPerShare).div(1e12);
        emit Deposit(msg.sender, _pid, _amount);
    }

    // Withdraw LP tokens from MasterChef.
    //当前地址提取 LP token
    //param：_pid，   pool id (即通过 pool id 可以找到对应池的的地址)
    //param：_amount, 提取的金额
    function withdraw(uint256 _pid, uint256 _amount) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");
        updatePool(_pid);

        //user.amount 减少 会影响收益
        uint256 pending =
user.amount.mul(pool.accHbtPerShare).div(1e12).sub(user.rewardDebt);
        address refer = playerBook.getPlayerLaffAddress(msg.sender);
        uint256 referRewardRate = playerBook._referRewardRate();
        uint256 baseRate = playerBook._baseRate();
        uint256 toRefer = pending.mul(referRewardRate).div(baseRate);
        // safeHbtTransfer(msg.sender, pending.sub(toRefer));
        userRewardInfo[_pid][msg.sender] =
```

```
userRewardInfo[_pid][msg.sender].add(pending.sub(toRefer));
        safeHbtTransfer(refer, toRefer);
        emit PlayerBookEvent(refer, msg.sender, toRefer);

        user.amount = user.amount.sub(_amount);
        user.rewardDebt = user.amount.mul(pool.accHbtPerShare).div(1e12);
        if(_amount > 0){
            pool.lpToken.safeTransfer(address(msg.sender), _amount);
            emit Withdraw(msg.sender, _pid, _amount);
        }
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    //当前地址紧急提取指定池的 LP Token，但得不到任何 YMI,谨慎使用
    //param：_pid，    pool id (即通过 pool id  可以找到对应池的的地址)
    function emergencyWithdraw(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        pool.lpToken.safeTransfer(address(msg.sender), user.amount);
        emit EmergencyWithdraw(msg.sender, _pid, user.amount);
        user.amount = 0;
        user.rewardDebt = 0;
    }

    // Safe hbt transfer function, just in case if rounding error causes pool to not have enough
HBTs.
    function safeHbtTransfer(address _to, uint256 _amount) internal {
        uint256 hbtBal = hbt.balanceOf(address(this));
        if (_amount > hbtBal) {
            hbt.transfer(_to, hbtBal);
        } else {
            hbt.transfer(_to, _amount);
        }
```

```
        // hbt.transfer(_to, _amount);
    }


    //提取收益&延时提取
    function extractReward(uint256 _pid, uint256 _times, bool _profitLock) public {

        withdraw(_pid,0);

        // PoolInfo storage pool = poolInfo[_pid];
        // UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 pending = userRewardInfo[_pid][msg.sender];

        if (_profitLock == false) {
            safeHbtTransfer(msg.sender, pending);
            emit ExtractReward(msg.sender, _pid, pending);
        } else {
            uint256 _pendingTimes = pending.mul(_times).div(10);
            hbt.allowMint(address(this), _pendingTimes.sub(pending));

            safeHbtTransfer(address(hbtLock), _pendingTimes);
            hbtLock.disposit(msg.sender,pending,_times);
            emit ProfitLock(msg.sender, _pid, pending, _times);
        }

        userRewardInfo[_pid][msg.sender] = 0;
    }
}
```

Migrations.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.8.0;
```

```
contract Migrations {
    address public owner = msg.sender;
    uint public last_completed_migration;

    modifier restricted() {
        require(
            msg.sender == owner,
            "This function is restricted to the contract's owner"
        );
        _;
    }

    function setCompleted(uint completed) public restricted {
        last_completed_migration = completed;
    }
}
```

MockERC20.sol

```
pragma solidity 0.6.12;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MockERC20 is ERC20 {
        constructor(
                string memory name,
                string memory symbol,
                uint256 supply
        ) public ERC20(name, symbol) {
                _mint(msg.sender, supply);
        }
}
```

6. 附录 B：安全风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 ETH 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 ETH 导致的拒绝服务漏洞、因 gas 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 ETH 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 gas 触发的事务顺序依赖风险等。

7. 附录 C：智能合约安全审计工具简介

6.1 Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号以太坊虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。 与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

6.2 Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

6.3 securify.sh

Securify 可以验证以太坊智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

6.4 Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

6.5 MAIAN

MAIAN 是一个用于查找以太坊智能合约漏洞的自动化工具，Maian 处理合

约的字节码，并尝试建立一系列交易以找出并确认错误。

6.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7 ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

6.8 Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建以太坊合约并调试交易。

6.9 知道创宇区块链安全审计人员专用工具包

知道创宇渗透测试人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509