KNOWNSEC

Degree of secrets ★ | Project team open

Smart contract audit report

HyperBC Farm

Security status

Security

★ ★ ★ ★ ★

Chief test Officer： *Knownsec blockchain security team*

Version Summary

Content	Date	Version
Editing Document	20201119	V1.0

Report Information

Title	Version	Document Number	Type
HyperBC Farm contract audit report	V1.0	【HyperBC-ZNNY-20201118】	Open to project team

Copyright Notice

Knownsec only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. Knownsec 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'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 shall not be liable for any losses and adverse effects caused thereby.

Table of Contents

1. Introduction .... - 6 -

2. Code vulnerability analysis .... - 8 -

2.1 Vulnerability Level Distribution .... - 8 -

2.2 Audit Result .... - 9 -

3. Analysis of code audit results .... - 12 -

3.1. Governance update logic【PASS】 .... - 12 -

3.2. nameFilter logic design【PASS】 .... - 12 -

3.3. Add transaction pool logic【PASS】 .... - 15 -

3.4. Distribution of trading pool points【PASS】 .... - 17 -

3.5. LP Token migration logic design【PASS】 .... - 17 -

3.6. LP Token mortgage logic design【PASS】 .... - 18 -

3.7. LP Token extraction logic design【PASS】 .... - 19 -

3.8. HBT mortgage logic design【PASS】 .... - 20 -

3.9. HBT income extraction logic【PASS】 .... - 22 -

4. Basic code vulnerability detection .... - 25 -

4.1. Compiler version security【PASS】 .... - 25 -

4.2. Redundant code【PASS】 .... - 25 -

4.3. Use of safe arithmetic library【PASS】 .... - 25 -

4.4. Not recommended encoding【PASS】 .... - 25 -

4.5. Reasonable use of require/assert【PASS】 .... - 26 -

4.6. Fallback function safety【PASS】 .... - 26 -

4.7. tx.origin authentication【PASS】 - 26 -

4.8. Owner permission control【PASS】 - 27 -

4.9. Gas consumption detection【PASS】 - 27 -

4.10. call injection attack【PASS】 - 27 -

4.11. Low-level function safety【PASS】 - 28 -

4.12. Vulnerability of additional token issuance【PASS】 - 28 -

4.13. Access control defect detection【PASS】 - 28 -

4.14. Numerical overflow detection【PASS】 - 29 -

4.15. Arithmetic accuracy error【PASS】 - 30 -

4.16. Incorrect use of random numbers【PASS】 - 30 -

4.17. Unsafe interface usage【PASS】 - 31 -

4.18. Variable coverage【PASS】 - 31 -

4.19. Uninitialized storage pointer【PASS】 - 31 -

4.20. Return value call verification【PASS】 - 32 -

4.21. Transaction order dependency【PASS】 - 33 -

4.22. Timestamp dependency attack【PASS】 - 33 -

4.23. Denial of service attack【PASS】 - 34 -

4.24. Fake recharge vulnerability【PASS】 - 34 -

4.25. Reentry attack detection【PASS】 - 35 -

4.26. Replay attack detection【PASS】 - 35 -

4.27. Rearrangement attack detection【PASS】 - 36 -

5. Appendix A：Contract code - 37 -

6. Appendix B：Vulnerability rating standard .... - 68 -

7. Appendix C：Introduction to auditing tools .... - 69 -

7.1 Manticore .... - 69 -

7.2 Oyente .... - 69 -

7.3 securify.sh .... - 69 -

7.4 Echidna .... - 70 -

7.5 MAIAN .... - 70 -

7.6 ethersplay .... - 70 -

7.7 ida-evm .... - 70 -

7.8 Remix-ide .... - 70 -

7.9 Knownsec Penetration Tester Special Toolkit.. - 70 -

1. Introduction

The effective test time of this report is from November 18, 2020 to November 19, 2020. During this period, the security and standardization of **the smart contract code of the HyperBC Farm** will be audited and used as the statistical basis for the report.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 3).**the smart contract code of the HyperBC Farm** is comprehensively assessed as **SAFE**.

Results of this smart contract security audit： SAFE

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.

Target information of the HyperBC Farm audit:

Target information	
Token name	HyperBC Farm
Code type	Eth smart contract code
Code language	solidity

Contract documents and hash:

Contract documents	MD5
HBTLock.sol	83F446514B375C1EAF0527E59A248071
HBTToken.sol	A6FA92BF988D177D3B86B5D59A434D05
MasterChef.sol	77FA2CCEA4D039DE2831C7C17913D0B1
Migrations.sol	CA8D6CA8A6EDF34F149A5095A8B074C9
MockERC20.sol	5801424F9432ACD5B9CAEC7EA3A15F2E

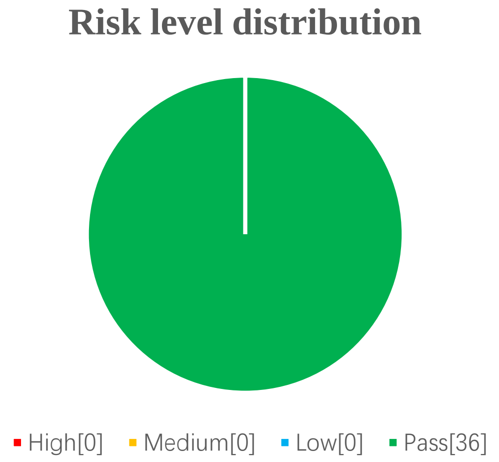
Governance.sol	0895ADD1485AE6F106000F2A8E2C0235
NameFilter.sol	203DC67CEA334E9975B9845A9FB6AD99
IPlayerBook.sol	72F3325B3DD7FD94D5BB08492C7F886A

2. Code vulnerability analysis

2.1 Vulnerability Level Distribution

Vulnerability risk statistics by level：

Vulnerability risk level statistics table			
High	Medium	Low	Pass
0	0	0	36



2.2 Audit Result

Result of audit			
Audit Target	**Audit**	**Status**	**Audit Description**
Business security testing	**Governance update logic**	Pass	After testing, there is no such safety vulnerability.
	NameFilter logic design	Pass	After testing, there is no such safety vulnerability.
	Add transaction pool logic	Pass	After testing, there is no such safety vulnerability.
	Distribution of trading pool points	Pass	After testing, there is no such safety vulnerability.
	LP Token migration logic	Pass	After testing, there is no such safety vulnerability.
	LP Token mortgage logic	Pass	After testing, there is no such safety vulnerability.
	LP Token extraction logic	Pass	After testing, there is no such safety vulnerability.
	HBT mortgage logic design	Pass	After testing, there is no such safety vulnerability.
	HBT income extraction logic	Pass	After testing, there is no such safety vulnerability.
Basic code vulnerability detection	**Compiler version security**	Pass	After testing, there is no such safety vulnerability.
	Redundant code	Pass	After testing, there is no such safety vulnerability.

	Use of safe arithmetic library	Pass	After testing, there is no such safety vulnerability.
	Not recommended encoding	Pass	After testing, there is no such safety vulnerability.
	Reasonable use of require/assert	Pass	After testing, there is no such safety vulnerability.
	fallback function safety	Pass	After testing, there is no such safety vulnerability.
	tx.orgin authentication	Pass	After testing, there is no such safety vulnerability.
	Owner permission control	Pass	After testing, there is no such safety vulnerability.
	Gas consumption detection	Pass	After testing, there is no such safety vulnerability.
	call injection attack	Pass	After testing, there is no such safety vulnerability.
	Low-level function safety	Pass	After testing, there is no such safety vulnerability.
	Vulnerability of additional token issuance	Pass	After testing, there is no such safety vulnerability.
	Access control defect detection	Pass	After testing, there is no such safety vulnerability.
	Numerical overflow detection	Pass	After testing, there is no such safety vulnerability.
	Arithmetic accuracy error	Pass	After testing, there is no such safety vulnerability.

	Wrong use of random number detection	Pass	After testing, there is no such safety vulnerability.
	Unsafe interface use	Pass	After testing, there is no such safety vulnerability.
	Variable coverage	Pass	After testing, there is no such safety vulnerability.
	Uninitialized storage pointer	Pass	After testing, there is no such safety vulnerability.
	Return value call verification	Pass	After testing, there is no such safety vulnerability.
	Transaction order dependency detection	Pass	After testing, there is no such safety vulnerability.
	Timestamp dependent attack	Pass	After testing, there is no such safety vulnerability.
	Denial of service attack detection	Pass	After testing, there is no such safety vulnerability.
	Fake recharge vulnerability detection	Pass	After testing, there is no such safety vulnerability.
	Reentry attack detection	Pass	After testing, there is no such safety vulnerability.
	Replay attack detection	Pass	After testing, there is no such safety vulnerability.
	Rearrangement attack detection	Pass	After testing, there is no such safety vulnerability.

3. Analysis of code audit results

3.1. Governance update logic【PASS】

Audit the update logic of Governance, check whether there is permission verification and security check of the update account during the update, such as the legality of the address.

Audit analysis: When Governance is updated, the modifier is used to check the authority of the caller. Only the current Governance can set a new Governance, and at the same time, the new Governance address is checked for non-empty, and then the Governance is reset.

```
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;
    }
```

Recommendation： nothing.

3.2. nameFilter logic design【PASS】

NameFilter is used to filter and check the name string. First, the string needs to be converted to lowercase, and the string needs to meet the following conditions:

1. Cannot start with "0x" or "0X"

2. The characters must be A~Z, a~z, 0~9, space

3. Not all numbers

4. Cannot start or end with spaces

5. There cannot be multiple spaces in a line

Audit analysis: the logical design of nameFilter is correct.

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

        require (_length <= 32 && _length > 0, "string must be between 1 and 32 characters");
        if (_temp[0] == 0x30)
        {
            require(_temp[1] != 0x78, "string cannot start with 0x");
            require(_temp[1] != 0x58, "string cannot start with 0X");
        }

        bool _hasNonNumber;

        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);
    }
```

Recommendation： nothing.

3.3. Add transaction pool logic【PASS】

Perform security audits on the added transaction pool logic, check whether there is permission verification for the function caller, and whether the added transaction pool logic is properly designed.

Audit analysis: Adding the transaction pool function only allows the owner of the contract to call. When adding a transaction pool, all current transaction pool information will be updated in batches, startBlock will be determined again, and then the transaction pool information will be updated.

```
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
        }));
    }
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }
    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;
    }
```

Recommendation： nothing.

3.4. Distribution of trading pool points【PASS】

Set the number of points allocated to the trading pool, check whether there is a permission check transaction, and whether the logic for setting the number of allocated points is reasonable.

Audit analysis: The transaction pool allocation function can only be called by the owner of the contract, and the logic design is reasonable.

```
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;
    }
```

Recommendation： nothing.

3.5. LP Token migration logic design【PASS】

Audit the LP Token migration logic and check the rationality of the logic.

Audit analysis: The logical design of LP Token is reasonable.

```
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;
```

```
    }
```

Recommendation： nothing.

3.6. LP Token mortgage logic design【PASS】

NameFilter is used to filter and check the name string. First, the string needs to be converted to lowercase, and the string needs to meet the following conditions:

Audit analysis: The mortgage logic design is reasonable.

```
  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);
    }
```

Recommendation： nothing.

3.7. LP Token extraction logic design【PASS】

Perform security audits on the LP Token extraction logic, check whether the extraction quantity is checked, and whether the updated logic after extraction is reasonably designed.

Audit analysis: LP Token extraction logic design is correct.

```
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);
        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);

        }

    }
```

Recommendation： nothing.

3.8. HBT mortgage logic design 【PASS】

Perform a security audit on the HBT mortgage logic to check whether the design of the mortgage logic is reasonable.

Audit analysis: HBT mortgage logic design is correct.

```
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);
        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;
    }
```

Recommendation： nothing.

3.9. HBT income extraction logic【PASS】

Audit the profit withdrawal logic, and check whether there is a security check on the withdrawal amount and whether the withdrawal is locked.

Audit analysis: No design defects were found.

```
    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);
    }
        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);
        }
        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;
    }
```

Recommendation： nothing.

4. Basic code vulnerability detection

4.1. Compiler version security【PASS】

Check whether a safe compiler version is used in the contract code implementation.

Audit result: After testing, the smart contract code has a compiler version 0.5.15 or higher, and there is no such security problem.

Recommendation： nothing.

4.2. Redundant code【PASS】

Check whether the contract code implementation contains redundant code.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.3. Use of safe arithmetic library【PASS】

Check whether the SafeMath safe arithmetic library is used in the contract code implementation.

Audit result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.4. Not recommended encoding【PASS】

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.5. Reasonable use of require/assert【PASS】

Check the rationality of the use of require and assert statements in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.6. Fallback function safety【PASS】

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

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.7. 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 a smart contract makes the contract vulnerable to attacks like phishing.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.8. Owner permission control【PASS】

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.9. Gas consumption detection【PASS】

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

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.10. call injection attack【PASS】

When the call function is called, strict permission control should be done, or the function called by the call should be written dead.

Audit result: After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.11. Low-level function safety【PASS】

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.12. Vulnerability of additional token issuance【PASS】

Check whether there is a function that may increase the total amount of tokens in the token contract after initializing the total amount of tokens.

Audit result: After testing, the smart contract code has the function of issuing additional tokens, but because liquid mining requires additional tokens, it is approved.

Recommendation： nothing.

4.13. Access control defect detection【PASS】

Different functions in the contract should set reasonable permissions.

Check whether each function in the contract correctly uses keywords such as public and private for visibility modification, check whether the contract is correctly defined and use modifier to restrict access to key functions to avoid problems caused by unauthorized access.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.14. Numerical overflow detection【PASS】

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.15. Arithmetic accuracy error【PASS】

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages-Solidity does not float Point type, and all the numerical calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same level of calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.16. Incorrect use of random numbers【PASS】

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they appear or are affected by miners. These random numbers are predictable to a certain extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.17. Unsafe interface usage【PASS】

Check whether unsafe interfaces are used in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.18. Variable coverage【PASS】

Check whether there are security issues caused by variable coverage in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.19. Uninitialized storage pointer【PASS】

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.20. Return value call verification【PASS】

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

In Solidity, there are transfer(), send(), call.value() and other currency transfer methods, which can all be used to send Ether to an address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when call.value fails to be sent; all available gas will be passed for calling (can be Limit by passing in gas_value parameters), which cannot effectively prevent reentry attacks.

If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may lead to unexpected results due to Ether sending failure.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.21. Transaction order dependency【PASS】

Since miners always get gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees for faster transactions. Since the Ethereum blockchain is public, everyone can see the content 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 transaction at a higher fee to preempt the original solution.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.22. Timestamp dependency attack【PASS】

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it only needs to verify whether the timestamp is later than the previous block and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.23. Denial of service attack【PASS】

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior as the transaction recipient, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.24. Fake recharge vulnerability【PASS】

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, enter the else logic part and return false, and finally no exception is thrown. We believe that only if/else this kind of gentle judgment method is an imprecise coding method in sensitive function scenarios such as transfer.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.25. Reentry attack detection【PASS】

Re-entry vulnerability is the most famous Ethereum smart contract vulnerability, which caused the fork of Ethereum(The DAO hack).

The **call.value()** function in Solidity consumes all the gas it receives when it is used to send Ether. When the **call.value()** function to send Ether occurs before the actual reduction of the sender’s account balance, There is a risk of reentry attacks.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

4.26. Replay attack detection【PASS】

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

In the asset management system, there are often cases of entrusted management. The principal assigns assets to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario is also common in smart contracts.

Audit results： After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.27. Rearrangement attack detection【PASS】

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping, so that the attacker has the opportunity to store their own information in the contract. in.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

5. Appendix A：Contract code

Source code:

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 = 1000000000000000000000000000;
      }

      //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. Appendix B：Vulnerability rating standard

Smart contract vulnerability rating standards	
Level	**Level Description**
High	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: value overflow loopholes that can cause the value of tokens to zero, fake recharge loopholes that can cause exchanges to lose tokens, and can cause contract accounts to lose ETH or tokens. Access loopholes, etc.; Vulnerabilities that can cause loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to bypassing of access control of key functions, etc.; Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending ETH to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.
Medium	High-risk vulnerabilities that require specific addresses to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; access control defects for non-critical functions, and logical design defects that cannot cause direct capital losses, etc.
Low	Vulnerabilities that are difficult to be triggered, vulnerabilities with limited damage after triggering, such as value overflow vulnerabilities that require a large amount of ETH or tokens to trigger, vulnerabilities where attackers cannot directly profit after triggering value overflow, and the transaction sequence triggered by specifying high gas depends on the risk Wait.

7. Appendix C：Introduction to auditing tools

7.1 Manticore

Manticore is a symbolic execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum Virtual Machine (EVM), an EVM disassembler/assembler and a convenient interface for automatic compilation and analysis of Solidity. It also integrates Ethersplay, Bit of Traits of Bits visual disassembler for EVM bytecode, used for visual analysis. Like binary files, Manticore provides a simple command line interface and a Python for analyzing EVM bytecode API.

7.2 Oyente

Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction sequencing dependencies, etc. More convenient, Oyente's design is modular, so this allows advanced users to implement and Insert their own detection logic to check the custom attributes in their contract.

7.3 securify.sh

Securify can verify common security issues of Ethereum smart contracts, such as disordered transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify also has a specific language for specifying vulnerabilities, which makes Securify can keep an eye on current security and other reliability issues at any time.

7.4 Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

7.5 MAIAN

MAIAN is an automated tool for finding vulnerabilities in Ethereum smart contracts. Maian processes the bytecode of the contract and tries to establish a series of transactions to find and confirm the error.

7.6 ethersplay

ethersplay is an EVM disassembler, which contains relevant analysis tools.

7.7 ida-evm

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

7.8 Remix-ide

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

7.9 Knownsec Penetration Tester Special Toolkit

Pen-Tester tools collection is created by KnownSec team. It contains plenty of

Pen-Testing tools such as automatic testing tool, scripting tool, Self-developed tools etc.

KNOWNSEC

Beijing KnownSec Information Technology Co., Ltd.

Advisory telephone	+86(10)400 060 9587
E-mail	sec@knownsec.com
Website	www.knownsec.com
Address	wangjing soho T2-B2509,Chaoyang District, Beijing