KNOWNSEC

Degree of secrecy ★ | Project team open

# Smart Contract Audit Report

Security status

## Safe

★ ★ ★ ★ ★

Principal tester: Knownsec blockchain security team

# Version Summary

| Content | Date | Version |
|---|---|---|
| Editing Document | 20210526 | V1.0 |

# Report Information

| Title | Version | Document Number | Type |
|---|---|---|---|
| **OOE Smart Contract Audit Report** | V1.0 | 19dad4e2061248ee8cf0406e21024eef | 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

The effective test time of this report is from From **May 20, 2021** to **May 26, 2021** . During this period, the security and standardization of **the smart contract code of the OOE** will be audited and used as the statistical basis for the report.

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

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 3). **The smart contract code of the OOE** 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.

**Report information of this audit:**

**Report Number:** 19dad4e2061248ee8cf0406e21024eef

**Report query address link:**

https://attest.im/attestation/searchResult?qurey=19dad4e2061248ee8cf0406e21024eef

**Target information of the OOE audit:**

| Target information | |
|---|---|
| **Project name** | OOE |

| | |
|---|---|
| **Token address** | factory:<br>0xd76d8C2A7CA0a1609Aea0b9b5017B3F7782891bf<br>router:<br>0xBeB43fbb2f7AEA8AC904975816BB1b4cA9f4D9c5<br>Farming:<br>0xB3ccece7f26acd558a2Eb7Eab53ae4D840b3401D |
| **Code type** | Token code, Ethereum&BSC smart contract code |
| **Code language** | Solidity |

**Contract documents and hash:**

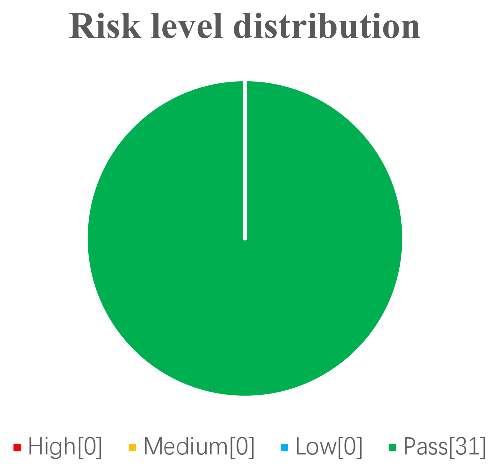
| Contract documents | MD5 |
|---|---|
| **Factory.sol** | 174BF188F54A8829A845133D6914D1C6 |
| **Router.sol** | 55CD44C3B9A8BF0003BE60EDA7A5B678 |
| **SimpleStaking.sol** | 030BF10989C266DC153E31BDD62FACEE |

# 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 | 31 |



## 2.2 Audit Result

| Result of audit | | | |
|---|---|---|---|
| **Audit Target** | **Audit** | **Status** | **Audit Description** |
| Business security testing | Factory.sol adds trading pair function | Pass | After testing, there is no such safety vulnerability. |
| | Router.sol contract liquidity related functions | Pass | After testing, there is no such safety vulnerability. |
| | SimpleStaking contract deposit and withdrawal related functions | Pass | After testing, there is no such safety vulnerability. |
| | SimpleStaking contract reward related functions | 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.oriigin 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. Factory.sol adds trading pair function 【PASS】

**Audit analysis:** The createPair function of the contract will determine whether the transaction pair exists, and if it does not exist, it will add a liquid mining transaction pair. The function code is standardized, and no obvious security problems have been found.

```
function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'OpenOcean: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'OpenOcean: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'OpenOcean: PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(PancakePair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IPancakePair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }
```

**Recommendation:** nothing.

## 3.2. Router.sol contract liquidity related functions 【PASS】

**Audit analysis:** The addLiquidity function of the contract adds liquidity to the mining pool, and the removeLiquidity function removes the liquidity of the mining pool. Function code specification, will not cause overflow or underflow vulnerabilities.

```
function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal virtual returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        if (IUniswapV2Factory(factory).getPair(tokenA, tokenB) == address(0)) {
            IUniswapV2Factory(factory).createPair(tokenA, tokenB);
        }
        (uint reserveA, uint reserveB) = PancakeLibrary.getReserves(factory, tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = PancakeLibrary.quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'OpenOceanRouter: INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = PancakeLibrary.quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'OpenOceanRouter: INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin, amountBMin);
        address pair = PancakeLibrary.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IUniswapV2Pair(pair).mint(to);
    }

function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
        address pair = PancakeLibrary.pairFor(factory, tokenA, tokenB);
        IUniswapV2Pair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
        (uint amount0, uint amount1) = IUniswapV2Pair(pair).burn(to);
        (address token0,) = PancakeLibrary.sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'OpenOceanRouter: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'OpenOceanRouter: INSUFFICIENT_B_AMOUNT');
    }
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountToken, uint amountETH) {
        (amountToken, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
```

**Recommendation：** nothing.

## 3.3. SimpleStaking contract deposit and withdrawal related functions【PASS】

**Audit analysis:** The deposit function of the contract deposits pledged tokens, and the withdraw function withdraws the pledged tokens. Function code specification, will not cause overflow or underflow vulnerabilities.

```
    function deposit(uint amount) public {
        updateIndex();
        distributeReward(msg.sender);
        require(stakeToken.transferFrom(msg.sender, address(this), amount), "transferFrom failed");
        userCollateral[msg.sender] = userCollateral[msg.sender].add(amount);
        totalCollateral = totalCollateral.add(amount);
        emit Deposit(msg.sender, amount);
    }

    function withdraw(uint amount) public {
        updateIndex();
        distributeReward(msg.sender);
        require(stakeToken.transfer(msg.sender, amount), "transfer failed");
        userCollateral[msg.sender] = userCollateral[msg.sender].sub(amount);
        totalCollateral = totalCollateral.sub(amount);
        emit Withdraw(msg.sender, amount);
    }
```

**Recommendation：** nothing.

## 3.4. SimpleStaking contract reward related functions【PASS】

**Audit analysis:** The distributeReward function of the contract is used to distribute rewards, the claimReward function is used to declare rewards, and the withdrawRemainReward function is used to withdraw rewards from the balance.

```
function distributeReward(address user) private {
        if (userIndex[user] == 0 && index > 0) {
            userIndex[user] = doubleScale;
        }
        uint indexDelta = index - userIndex[user];
        userIndex[user] = index;
        uint rewardDelta = indexDelta.mul(userCollateral[user]).div(doubleScale);
        userAccrued[user] = userAccrued[user].add(rewardDelta);
        if (rewardToken.balanceOf(address(this)) >= userAccrued[user] && userAccrued[user] > 0) {
            if (rewardToken.transfer(user, userAccrued[user])) {
                userAccrued[user] = 0;
            }
        }
        emit RewardDistributed(user, rewardDelta, index);
    }

function claimReward(address[] memory user) public {
        updateIndex();
        for (uint i = 0; i < user.length; i++) {
            distributeReward(user[i]);
        }
    }

    function withdrawRemainReward() public onlyOwner {
        uint balance = rewardToken.balanceOf(address(this));
        rewardToken.transfer(owner(), balance);
    }

    function pendingReward(address user) public view returns (uint){
        uint blockDelta = block.number.sub(lastDistributedBlock);
        uint rewardAccrued = blockDelta.mul(rewardSpeed);
        if (totalCollateral == 0) {
            return userAccrued[user];
        }
        uint ratio = rewardAccrued.mul(doubleScale).div(totalCollateral);
        uint currentIndex = index.add(ratio);
        uint uIndex = userIndex[user] == 0 && index > 0 ? doubleScale : userIndex[user];
        uint indexDelta = currentIndex - uIndex;
        uint rewardDelta = indexDelta.mul(userCollateral[user]).div(doubleScale);
        return rewardDelta + userAccrued[user];
    }
```

**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 formulated the compiler version 0.5.15 within the major version, 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 does not have the function of issuing additional tokens, and the upper limit is set, so it is passed.

**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 smart contract code does not use structure, and there is no such problem.

**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 once led to 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 (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 of code for this test:

```
Factory.sol
pragma solidity =0.5.16;

import './interfaces/IPancakeFactory.sol';
import './PancakePair.sol';

contract PancakeFactory is IPancakeFactory {
    bytes32          public          constant          INIT_CODE_PAIR_HASH          =
keccak256(abi.encodePacked(type(PancakePair).creationCode));

    address public feeTo;
    address public feeToSetter;

    mapping(address => mapping(address => address)) public getPair;
    address[] public allPairs;

    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
    }

    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'OpenOcean: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'OpenOcean: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'OpenOcean: PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(PancakePair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IPancakePair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'OpenOcean: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'OpenOcean: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
}

Router.sol
pragma solidity =0.6.6;

import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol';
import '@uniswap/lib/contracts/libraries/TransferHelper.sol';

import './interfaces/IPancakeRouter02.sol';
import './libraries/PancakeLibrary.sol';
import './libraries/SafeMath.sol';
import './interfaces/IERC20.sol';
import './interfaces/IWETH.sol';

contract PancakeRouter is IPancakeRouter02 {
    using SafeMath for uint;

    address public immutable override factory;
    address public immutable override WETH;

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'OpenOceanRouter: EXPIRED');
        _;
    }

    constructor(address _factory, address _WETH) public {
        factory = _factory;
        WETH = _WETH;
    }

    receive() external payable {
        assert(msg.sender == WETH); // only accept ETH via fallback from the WETH contract
    }

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal virtual returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        if (IUniswapV2Factory(factory).getPair(tokenA, tokenB) == address(0)) {
            IUniswapV2Factory(factory).createPair(tokenA, tokenB);
        }
        (uint reserveA, uint reserveB) = PancakeLibrary.getReserves(factory, tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = PancakeLibrary.quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'OpenOceanRouter:
INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = PancakeLibrary.quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'OpenOceanRouter:
INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin,
amountBMin);
        address pair = PancakeLibrary.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IUniswapV2Pair(pair).mint(to);
    }
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint amountToken, uint amountETH, uint liquidity)
{
        (amountToken, amountETH) = _addLiquidity(
            token,
            WETH,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = PancakeLibrary.pairFor(factory, token, WETH);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWETH(WETH).deposit{value: amountETH}();
        assert(IWETH(WETH).transfer(pair, amountETH));
        liquidity = IUniswapV2Pair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender, msg.value - amountETH);
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
        address pair = PancakeLibrary.pairFor(factory, tokenA, tokenB);
        IUniswapV2Pair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
        (uint amount0, uint amount1) = IUniswapV2Pair(pair).burn(to);
        (address token0,) = PancakeLibrary.sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'OpenOceanRouter: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'OpenOceanRouter: INSUFFICIENT_B_AMOUNT');
    }
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountToken, uint amountETH) {
        (amountToken, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountA, uint amountB) {
        address pair = PancakeLibrary.pairFor(factory, tokenA, tokenB);
        uint value = approveMax ? uint(-1) : liquidity;
        IUniswapV2Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin, amountBMin, to,
deadline);
    }
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountToken, uint amountETH) {
        address pair = PancakeLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IUniswapV2Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountETH) = removeLiquidityETH(token, liquidity, amountTokenMin, amountETHMin,
to, deadline);
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountETH) {
        (, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountETH) {
        address pair = PancakeLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IUniswapV2Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountETHMin, to, deadline
        );
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = PancakeLibrary.sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amountOut,
uint(0));
            address to = i < path.length - 2 ? PancakeLibrary.pairFor(factory, output, path[i + 2]) : _to;
            IUniswapV2Pair(PancakeLibrary.pairFor(factory, input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = PancakeLibrary.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'OpenOceanRouter:
INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, PancakeLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = PancakeLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'OpenOceanRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, PancakeLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'OpenOceanRouter: INVALID_PATH');
        amounts = PancakeLibrary.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'OpenOceanRouter:
INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(PancakeLibrary.pairFor(factory, path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
    }
    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to,
uint deadline)
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WETH, 'OpenOceanRouter: INVALID_PATH');
        amounts = PancakeLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'OpenOceanRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, PancakeLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }
    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WETH, 'OpenOceanRouter: INVALID_PATH');
        amounts = PancakeLibrary.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'OpenOceanRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, PancakeLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }
    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'OpenOceanRouter: INVALID_PATH');
        amounts = PancakeLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, 'OpenOceanRouter: EXCESSIVE_INPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(PancakeLibrary.pairFor(factory, path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender, msg.value - amounts[0]);
    }

    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
    function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = PancakeLibrary.sortTokens(input, output);
            IUniswapV2Pair pair = IUniswapV2Pair(PancakeLibrary.pairFor(factory, input, output));
            uint amountInput;
            uint amountOutput;
            { // scope to avoid stack too deep errors
            (uint reserve0, uint reserve1,) = pair.getReserves();
            (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
            amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput);
            amountOutput = PancakeLibrary.getAmountOut(amountInput, reserveInput, reserveOutput);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) : (amountOutput, uint(0));
            address to = i < path.length - 2 ? PancakeLibrary.pairFor(factory, output, path[i + 2]) : _to;
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }
    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) {
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, PancakeLibrary.pairFor(factory, path[0], path[1]), amountIn
        );
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'OpenOceanRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        payable
        ensure(deadline)
    {
        require(path[0] == WETH, 'OpenOceanRouter: INVALID_PATH');
        uint amountIn = msg.value;
        IWETH(WETH).deposit{value: amountIn}();
        assert(IWETH(WETH).transfer(PancakeLibrary.pairFor(factory, path[0], path[1]), amountIn));
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'OpenOceanRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        ensure(deadline)
    {
        require(path[path.length - 1] == WETH, 'OpenOceanRouter: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, PancakeLibrary.pairFor(factory, path[0], path[1]), amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20(WETH).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'OpenOceanRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).withdraw(amountOut);
        TransferHelper.safeTransferETH(to, amountOut);
    }

    // **** LIBRARY FUNCTIONS ****
    function quote(uint amountA, uint reserveA, uint reserveB) public pure virtual override returns (uint amountB)
{
        return PancakeLibrary.quote(amountA, reserveA, reserveB);
    }

    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
        public
        pure
        virtual
        override
        returns (uint amountOut)
    {
        return PancakeLibrary.getAmountOut(amountIn, reserveIn, reserveOut);
    }

    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
        public
        pure
        virtual
        override
        returns (uint amountIn)
    {
        return PancakeLibrary.getAmountIn(amountOut, reserveIn, reserveOut);
    }

    function getAmountsOut(uint amountIn, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return PancakeLibrary.getAmountsOut(factory, amountIn, path);
    }

    function getAmountsIn(uint amountOut, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return PancakeLibrary.getAmountsIn(factory, amountOut, path);
    }
}
```

**SimpleStake.sol**

```
// SPDX-License-Identifier: LGPL-3.0-or-later
pragma solidity ^0.7.0;

import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

/* users could create staking-reward model with this contract at single mode */

contract SimpleStaking is Ownable {
    using SafeMath for uint;

    uint constant doubleScale = 10 ** 36;

    // stake token
    IERC20 public stakeToken;

    // reward token
    IERC20 public rewardToken;

    // the number of reward token distribution for each block
    uint public rewardSpeed;

    // user deposit
    mapping(address => uint) public userCollateral;
    uint public totalCollateral;

    // use index to distribute reward token
    // index is compound exponential
    mapping(address => uint) public userIndex;
    uint public index;

    mapping(address => uint) public userAccrued;

    // record latest block height of reward token distributed
    uint public lastDistributedBlock;

    /* event */
    event Deposit(address user, uint amount);
    event Withdraw(address user, uint amount);
    event RewardSpeedUpdated(uint oldSpeed, uint newSpeed);
    event RewardDistributed(address indexed user, uint delta, uint index);

    constructor(IERC20 _stakeToken, IERC20 _rewardToken) Ownable(){
        stakeToken = _stakeToken;
        rewardToken = _rewardToken;
        index = doubleScale;
    }

    function deposit(uint amount) public {
        updateIndex();
        distributeReward(msg.sender);
        require(stakeToken.transferFrom(msg.sender, address(this), amount), "transferFrom failed");
        userCollateral[msg.sender] = userCollateral[msg.sender].add(amount);
        totalCollateral = totalCollateral.add(amount);
        emit Deposit(msg.sender, amount);
    }

    function withdraw(uint amount) public {
        updateIndex();
        distributeReward(msg.sender);
        require(stakeToken.transfer(msg.sender, amount), "transfer failed");
        userCollateral[msg.sender] = userCollateral[msg.sender].sub(amount);
        totalCollateral = totalCollateral.sub(amount);
        emit Withdraw(msg.sender, amount);
    }

    function setRewardSpeed(uint speed) public onlyOwner {
        updateIndex();
        uint oldSpeed = rewardSpeed;
        rewardSpeed = speed;
        emit RewardSpeedUpdated(oldSpeed, speed);
    }

    function updateIndex() private {
        uint blockDelta = block.number.sub(lastDistributedBlock);
        if (blockDelta == 0) {
            return;
        }
        uint rewardAccrued = blockDelta.mul(rewardSpeed);
        if (totalCollateral > 0) {
            uint indexDelta = rewardAccrued.mul(doubleScale).div(totalCollateral);
            index = index.add(indexDelta);
        }
        lastDistributedBlock = block.number;
    }

    function distributeReward(address user) private {
        if (userIndex[user] == 0 && index > 0) {
            userIndex[user] = doubleScale;
        }
        uint indexDelta = index - userIndex[user];
        userIndex[user] = index;
        uint rewardDelta = indexDelta.mul(userCollateral[user]).div(doubleScale);
        userAccrued[user] = userAccrued[user].add(rewardDelta);
        if (rewardToken.balanceOf(address(this)) >= userAccrued[user] && userAccrued[user] > 0) {
            if (rewardToken.transfer(user, userAccrued[user])) {
                userAccrued[user] = 0;
            }
        }
        emit RewardDistributed(user, rewardDelta, index);
    }

    function claimReward(address[] memory user) public {
        updateIndex();
        for (uint i = 0; i < user.length; i++) {
            distributeReward(user[i]);
        }
    }

    function withdrawRemainReward() public onlyOwner {
        uint balance = rewardToken.balanceOf(address(this));
        rewardToken.transfer(owner(), balance);
    }

    function pendingReward(address user) public view returns (uint){
        uint blockDelta = block.number.sub(lastDistributedBlock);
        uint rewardAccrued = blockDelta.mul(rewardSpeed);
        if (totalCollateral == 0) {
            return userAccrued[user];
        }
        uint ratio = rewardAccrued.mul(doubleScale).div(totalCollateral);
        uint currentIndex = index.add(ratio);
        uint uIndex = userIndex[user] == 0 && index > 0 ? doubleScale : userIndex[user];
        uint indexDelta = currentIndex - uIndex;
        uint rewardDelta = indexDelta.mul(userCollateral[user]).div(doubleScale);
        return rewardDelta + userAccrued[user];
    }
}
```

# 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 BNB or tokens. Access loopholes, etc.;<br>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.;<br>Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending BNB 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 BNB 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 |