Smart contract audit report

DPMC

CHAINLION

NO.0C002207040001

JULY 04, 2022

CATALOGUE

CHAINLION

1. PROJECT SUMMARY

Entry type	Specific description
Entry name	DPMC
Project type	DEFI
Application platform	BSC

2. AUDIT SUMMARY

Entry type	Specific description
Project cycle	JUNE/29/2022-JULY/04/2022
Audit method	Black box test、White box test、Grey box test
Auditors	Two

3. VULNERABILITY SUMMARY

Audit results are as follows:

Entry type	Specific description
Serious vulnerability	0
High risk vulnerability	0
Moderate risk	2
Low risk vulnerability	3

Security vulnerability rating description：

1) **Serious vulnerability：** Security vulnerabilities that can directly cause token contracts or user capital losses， For example: shaping overflow vulnerability、Fake recharge vulnerability、Reentry attacks, vulnerabilities, etc.

2) **High risk vulnerability：** Security vulnerabilities that can directly cause the contract to fail to work normally, such as reconstructed smart contract caused by constructor design error, denial of service vulnerability caused by unreasonable design of require / assert detection conditions, etc.

3) **Moderate risk：** Security problems caused by unreasonable business logic design, such as accuracy problems caused by unreasonable numerical operation sequence design, variable ambiguous naming, variable coverage, call injection, conditional competition, etc.

4) **Low risk vulnerability：** Security vulnerabilities that can only be triggered by users with special permissions, such as contract backdoor vulnerability, duplicate name pool addition vulnerability, non-standard contract coding, contract detection bypass, lack of necessary events for key state variable change, and security vulnerabilities that are harmful in theory but have harsh utilization conditions.

4. EXECUTIVE SUMMARY

This report is prepared for **DPMC** smart contract， The purpose is to find the security vulnerabilities and non-standard coding problems in the smart contract through the security audit of the source code of the smart contract. This audit mainly involves the following test methods:

White box test

Conduct security audit on the source code of smart contract and check the security issues such as coding specification, DASP top 10 and business logic design

Grey box test

Deploy smart contracts locally and conduct fuzzy testing to check function robustness, function call permission and business logic security

Black box test

Conduct security test attacks on smart contracts from the perspective of attackers, combined with black-and-white and testing techniques, to check whether there are exploitable vulnerabilities.

This audit report is subject to the latest contract code provided by the current project party, does not include the newly added business logic function module after the contract upgrade, does not include new attack methods in the future, and does not include web front-end security and server-side security.

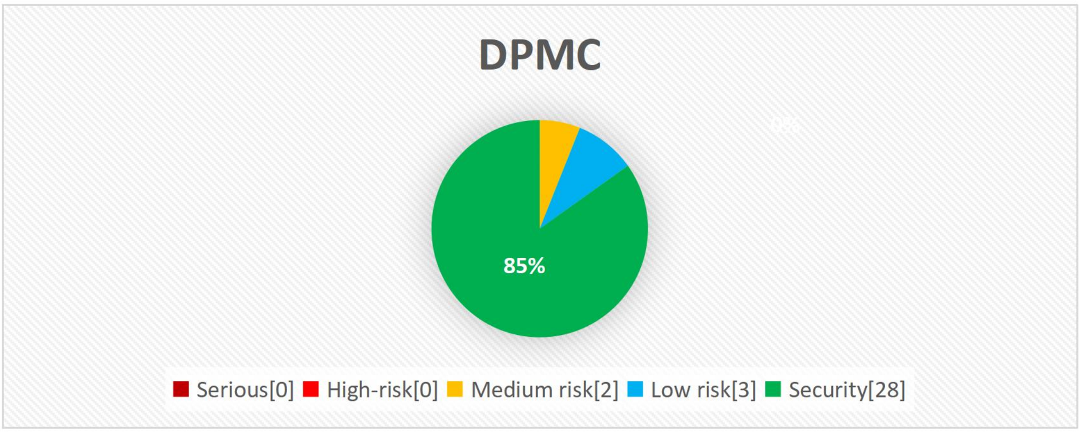
5. Directory structure

```
├─DPMCPools.sol
```

6. File hashes

Contract	SHA1 Checksum
DPMCPools.sol	2C92592F6CA574A39155E3C4E6B5F98759BE535D

7. Vulnerability distribution



8. Audit content

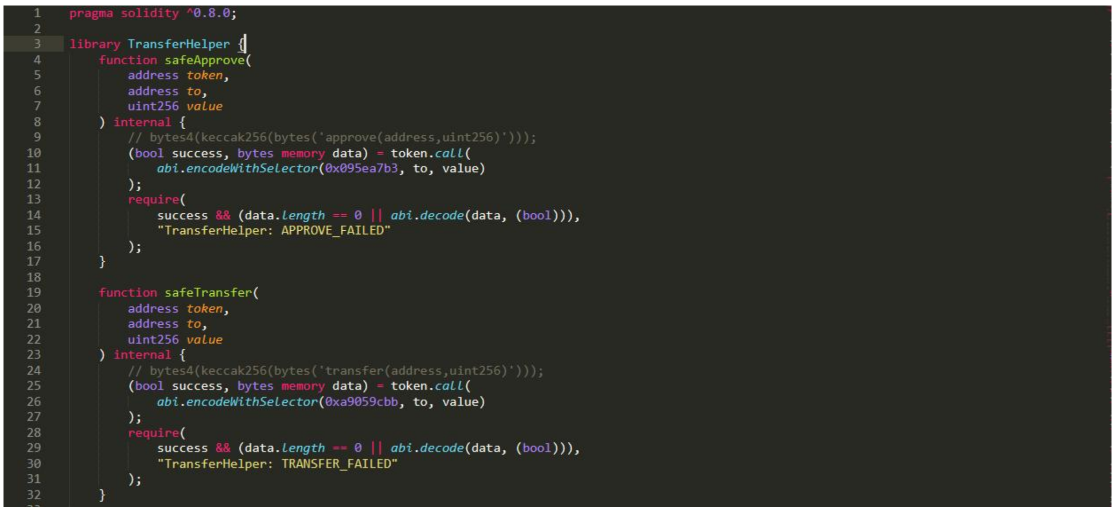
8.1. Coding specification

Smart contract supports contract development in programming languages such as solid, Vyper, C + +, Python and rust. Each programming language has its own coding specification. In the development process, the coding specification of the development language should be strictly followed to avoid security problems such as business function design defects.

8.1.1. Compiler Version 【security】

Audit description： The compiler version should be specified in the smart contract code. At the same time, it is recommended to use the latest compiler version. The old version of the compiler may cause various known security problems. At present, the latest version is v 0.8 x. And this version has been protected against shaping overflow.

Audit results： According to the audit, the compiler version used in the smart contract code is 0.8.6, so there is no such security problem.



```
pragma solidity ^0.8.0;

library TransferHelper {
    function safeApprove(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0x095ea7b3, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: APPROVE_FAILED"
        );
    }

    function safeTransfer(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0xa9059cbb, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: TRANSFER_FAILED"
        );
    }
```

Safety advice： NONE.

8.1.2. Return value verification 【security】

Audit description： Smart contract requires contract developers to strictly follow EIP / tip and other standards and specifications during contract development. For transfer, transferfrom and approve functions, Boolean values should be returned to feed back the final execution results. In the smart contract, the relevant business logic code often calls the transfer or transferfrom function to transfer. In this case, the return value involved in the transfer operation should be strictly checked to determine whether the transfer is successful or not, so as to avoid security vulnerabilities such as false recharge caused by the lack of return value verification.

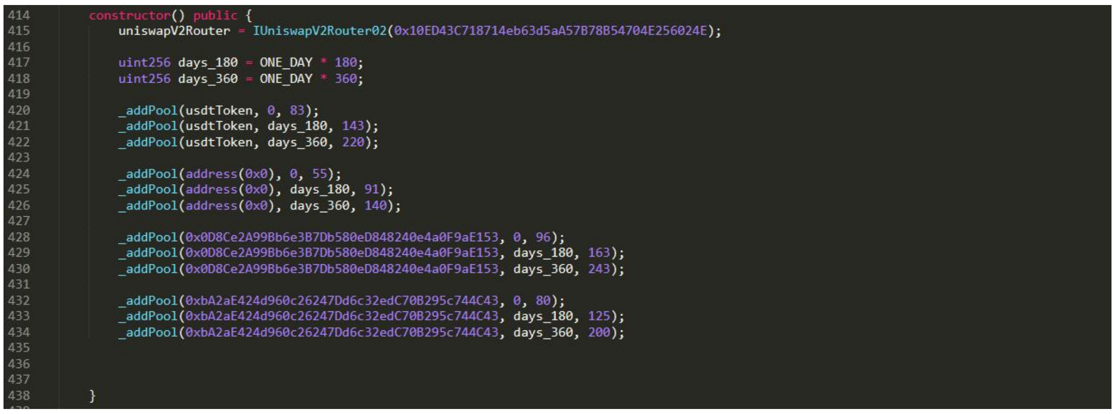
Audit results： According to the audit, there is no embedded function calling the official standards transfer and transferfrom in the smart contract, so there is no such security problem.

Safety advice： NONE.

8.1.3. Constructor writing 【security】

Audit description： In solid v0 The smart contract written by solidity before version 4.22 requires that the constructor must be consistent with the contract name. When the constructor name is inconsistent with the contract name, the constructor will become an ordinary public function. Any user can call the constructor to initialize the contract. After version V 0.4.22, The constructor name can be replaced by constructor, so as to avoid the coding problems caused by constructor writing.

Audit results： After audit, the constructor in the smart contract is written correctly, and there is no such security problem.



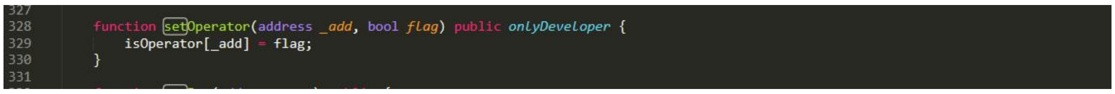
```
    constructor() public {
        uniswapV2Router = IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);

        uint256 days_180 = ONE_DAY * 180;
        uint256 days_360 = ONE_DAY * 360;

        _addPool(usdtToken, 0, 83);
        _addPool(usdtToken, days_180, 143);
        _addPool(usdtToken, days_360, 220);

        _addPool(address(0x0), 0, 55);
        _addPool(address(0x0), days_180, 91);
        _addPool(address(0x0), days_360, 140);

        _addPool(0x0D8Ce2A99Bb6e3B7Db580eD848240e4a0F9aE153, 0, 96);
        _addPool(0x0D8Ce2A99Bb6e3B7Db580eD848240e4a0F9aE153, days_180, 163);
        _addPool(0x0D8Ce2A99Bb6e3B7Db580eD848240e4a0F9aE153, days_360, 243);

        _addPool(0xbA2aE424d960c26247Dd6c32edC70B295c744C43, 0, 80);
        _addPool(0xbA2aE424d960c26247Dd6c32edC70B295c744C43, days_180, 125);
        _addPool(0xbA2aE424d960c26247Dd6c32edC70B295c744C43, days_360, 200);



    }
```

Safety advice： NONE.

8.1.4. Key event trigger 【Low risk】

Audit description： Most of the key global variable initialization or update operations similar to setXXX exist in the smart contract. It is recommended to trigger the corresponding event through emit when operating on similar key events.

Audit results： According to the audit, the initialization or update of key global variables in the smart contract lacks necessary event records and emit trigger events.



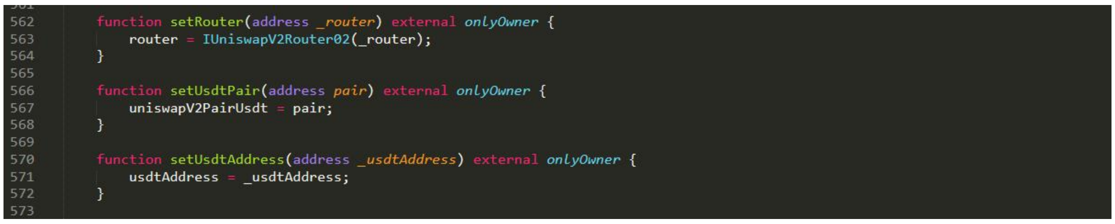
```
    function setOperator(address _add, bool flag) public onlyDeveloper {
        isOperator[_add] = flag;
    }
```

Safety advice： Add event event records and use emit to trigger, and check the address validity.

8.1.5. Address non-zero check 【Low risk】

Audit description： The smart contract initializes the key information of the contract through the constructor. When it comes to address initialization, the address should be non-zero checked to avoid irreparable economic losses.

Audit results： According to the audit, the legality of the address was not strictly checked in the contract.



```
    function setRouter(address _router) external onlyOwner {
        router = IUniswapV2Router02(_router);
    }

    function setUsdtPair(address pair) external onlyOwner {
        uniswapV2PairUsdt = pair;
    }

    function setUsdtAddress(address _usdtAddress) external onlyOwner {
        usdtAddress = _usdtAddress;
    }
```

Safety advice： Strictly check the legitimacy of the address.

8.1.6. Code redundancy check 【security】

Audit description： The deployment and execution of smart contracts need to consume certain gas costs. The business logic design should be optimized as much as possible, while avoiding unnecessary redundant code to improve efficiency and save costs.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

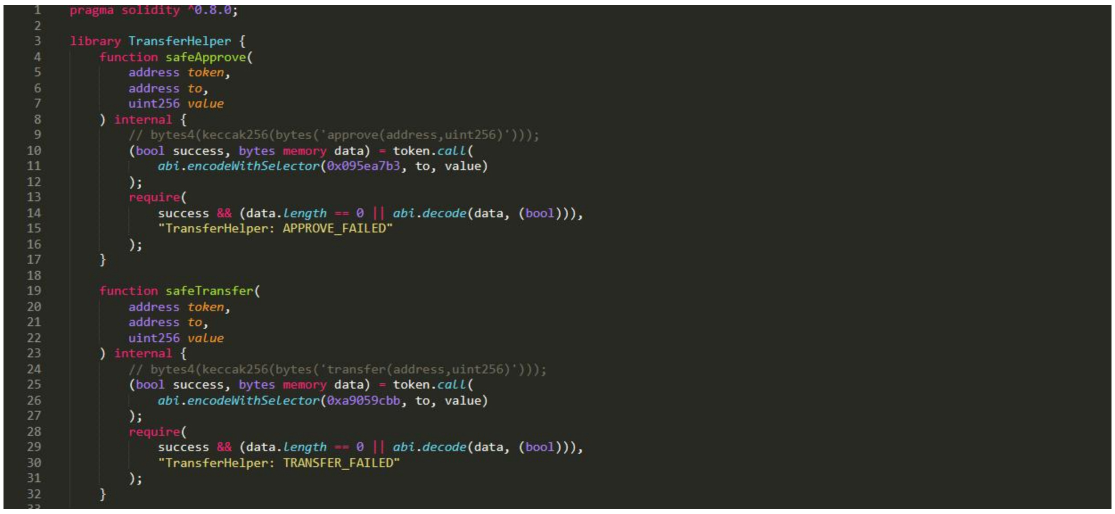
8.2. Coding design

DASP top 10 summarizes the common security vulnerabilities of smart contracts. Smart contract developers can study smart contract security vulnerabilities before developing contracts to avoid security vulnerabilities during contract development. Contract auditors can quickly audit and check the existing security vulnerabilities of smart contracts according to DASP top 10.

8.2.1. Shaping overflow detection 【security】

Audit description： Solid can handle 256 digits at most. When the number is unsigned, the maximum value will overflow by 1 to get 0, and 0 minus 1 will overflow to get the maximum value. The problem of shaping overflow often appears in the relevant logic code design function modules such as transaction transfer, reward calculation and expense calculation. The security problems caused by shaping overflow are also very serious, such as excessive coinage, high sales and low income, excessive distribution, etc. the problem of shaping overflow can be solved by using solid V 0.8 X version or by using the safemath library officially provided by openzenppelin.

Audit results： According to the audit, the smart contract is applicable to the compiler of version 0.8.0, and the safemath library is used for numerical operation, which better prevents the problem of shaping overflow.



```
pragma solidity ^0.8.0;

library TransferHelper {
    function safeApprove(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0x095ea7b3, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: APPROVE_FAILED"
        );
    }

    function safeTransfer(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0xa9059cbb, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: TRANSFER_FAILED"
        );
    }
```

Safety advice： NONE.

8.2.2. Reentry detection 【security】

Audit description： The in solidity provides call Value(), send(), transfer() and other functions are used for transfer operation. When call When value() sends ether, it will send all gas for transfer operation by default. If the transfer function can be called recursively again through call transfer, it can cause reentry attack.

Audit results： After audit, there is no such security problem.

Safety advice： **NONE.**

8.2.3. Rearrangement attack detection 【security】

Audit description： Rearrangement attack means that miners or other parties try to compete with smart contract participants by inserting their information into the list or mapping, so that attackers have the opportunity to store their information in the contract.

Audit results： After audit, there is no such security problem.

Safety advice： **NONE.**

8.2.4. Replay Attack Detection 【security】

Audit description： When the contract involves the business logic of delegated management, attention should be paid to the non reusability of verification to avoid replay attacks. In common asset management systems, there are often delegated management businesses. The principal gives the assets to the trustee for management, and the principal pays a certain fee to the trustee. In similar delegated management scenarios, it is necessary to ensure that the verification information will become invalid once used.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.2.5. False recharge detection 【security】

Audit description：When a smart contract uses the transfer function for transfer, it should use require / assert to strictly check the transfer conditions. It is not recommended to use if Use mild judgment methods such as else to check, otherwise it will misjudge the success of the transaction, resulting in the security problem of false recharge.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.2.6. Access control detection 【security】

Audit description： Solid provides four function access domain Keywords: public, private, external and internal to limit the scope of function. In the smart contract, the scope of function should be reasonably designed to avoid the security risk of improper access control. The main differences of the above four keywords are as follows:

1 . public： The marked function or variable can be called or obtained by any account, which can be a function in the contract, an external user or inherit the function in the contract

2 . external： The marked functions can only be accessed from the outside and cannot be called directly by the functions in the contract, but this can be used Func() calls this function as an external call

3 . private： Marked functions or variables can only be used in this contract (Note: the limitation here is only at the code level. Ethereum is a public chain, and anyone can directly obtain the contract status information from the chain)

4 . internal: It is generally used in contract inheritance. The parent contract is marked as an internal state variable or function, which can be directly accessed and called by the child contract (it cannot be directly obtained and called externally)

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.2.7. Denial of service detection 【security】

Audit description： Denial of service attack is a DoS attack on Ethereum contract, which makes ether or gas consume a lot. In more serious cases, it can make the contract code logic unable to operate normally. The common causes of DoS attack are: unreasonable design of require check condition, uncontrollable number of for cycles, defects in business logic design, etc.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.2.8. Conditional competition detection 【security】

Audit description： The Ethereum node gathers transactions and forms them into blocks. Once the miners solve the consensus problem, these transactions are considered effective. The miners who solve the block will also choose which transactions from the mine pool will be included in the block. This is usually determined by gasprice transactions. Attackers can observe whether there are transactions in the transaction pool that may contain problem solutions, After that, the attacker can obtain data from this transaction, create a higher-level transaction gasprice, and include its transaction in a block before the original, so as to seize the original solution.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.2.9. Consistency detection 【security】

Audit description： The update logic in smart contract (such as token quantity update, authorized transfer quota update, etc.) is often accompanied by the check logic of the operation object (such as anti overflow check, authorized transfer quota check, etc.), and when the update object is inconsistent with the check object, the check operation may be invalid, Thus, the conditional check logic is ignored and unexpected logic is executed. For example, the authorized transfer function function function transfer from (address _from, address _to, uint256 _value) returns (bool success) is used to authorize others to transfer on behalf of others. During transfer, the permission [_from] [MSG. Sender] authorized transfer limit will be checked, After passing the check, the authorized transfer limit will be updated at the same time of transfer. When the update object in the update logic is inconsistent with the check object in the check logic, the authorized transfer limit of the authorized transfer user will not change, resulting in that the authorized transfer user can transfer all the assets of the authorized account.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.2.10. Variable coverage detection 【security】

Audit description： Smart contracts allow inheritance relationships, in which the child contract inherits all the methods and variables of the parent contract. If a global variable with the same name as the parent contract is defined in the child contract, it may lead to variable coverage and corresponding asset losses.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.2.11. Random number detection 【security】

Audit description： Random numbers are often used in smart contracts. When designing the random number generation function, the generation and selection of random seeds should avoid the data information that can be queried on the blockchain, such as block Number and block Timestamp et al. These data are vulnerable to the influence of miners, resulting in the predictability of random numbers to a certain extent.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.2.12. Numerical operation detection 【security】

Audit description： Solidity supports addition, subtraction, multiplication, division and other conventional numerical operations, but solidty does not support floating-point types. When multiplication and division operations exist at the same time, the numerical operation order should be adjusted reasonably to reduce the error as much as possible.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.2.13. Call injection detection 【security】

Audit description： In the solid language, you can call a contract or a method of a local contract through the call method. There are roughly two ways to call: < address > Call (method selector, arg1, arg2,...) or < address > Call (bytes). When using call call, we can pass method selectors and parameters by passing parameters, or directly pass in a byte array. Based on this function, it is recommended that strict permission check or hard code the function called by call when using call function call.

Audit results： After audit, there is no such security problem.

Safety advice： NONE.

8.3. Business logic

Business logic design is the core of smart contract. When using programming language to develop contract business logic functions, developers should fully consider all aspects of the corresponding business, such as parameter legitimacy check, business permission design, business execution conditions, interaction design between businesses, etc.

8.3.1. Constructor initialization logic 【security】

Audit description： Conduct security audit on the business logic design of constructor initialization in the contract, and check whether the initialization value is consistent with the description of the requirements document.

Audit results： The constructor initialization business logic in the contract is designed correctly, and the token initialization value is consistent with the requirements document.

Code file： DPMCPools.sol L306~309 L393~438

Code information：

```
mapping(address => address) private _superior;   //Superior relationship
    mapping(address => address[]) private _childrens; //Subordinate relationship
    address _deployer; //Deploy users
    address public leadAddress; //Lead address
    mapping(address => bool) internal isOperator; //Whether to operate users
    mapping(address => bool) _isValied;    //Is the address valid

    event BindAddress(address indexed _add, address indexed _par);    //Relationship binding

    constructor() public {
        _deployer = msg.sender; //Initialize deployment user address
        leadAddress = msg.sender;    //Initialize lead address
    }

    struct OrderInfo {
        uint256 poolId;              //Pool ID
        uint256 stakedTime;          //Pledge time
        uint256 stakedAmount;        //Pledge quantity
        uint256 rewardTime;          //Reward time
        bool preUnlocked;            //Pre unlock
    }

    IUniswapV2Router02 public uniswapV2Router;
    address  usdtToken  =  0x55d398326f99059fF775485246999027B3197955;                    //usdt
address
    mapping(address  =>  OrderInfo[])  public  stakedOrders;          //Order  information  of  a  certain
address
    mapping(address => mapping(address => uint256)) public gottenAmount;      //Amount obtained
    mapping(address => mapping(address => uint256)) public inviteReward;      //Invitation reward
    mapping(address => uint256) public gottenTeamAmount;    //Team get quantity
    mapping(uint256 => bool) isBaned;
    mapping(uint256 => bool) isExisted;
    mapping(uint256 => uint256) public dayRate; // / 10000
    uint256[] public allPools;

    uint256 ONE_DAY = 86400;    //Minutes of the day

    constructor() public {
        uniswapV2Router                                                                          =
IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);      //uniswapV2Router
address

        uint256 days_180 = ONE_DAY * 180;    //Minutes in 180 days
        uint256 days_360 = ONE_DAY * 360;    //Minutes in 360 days

        _addPool(usdtToken, 0, 83);
        _addPool(usdtToken, days_180, 143);
        _addPool(usdtToken, days_360, 220);

        _addPool(address(0x0), 0, 55);
        _addPool(address(0x0), days_180, 91);
        _addPool(address(0x0), days_360, 140);

        _addPool(0x0D8Ce2A99Bb6e3B7Db580eD848240e4a0F9aE153, 0, 96);
        _addPool(0x0D8Ce2A99Bb6e3B7Db580eD848240e4a0F9aE153, days_180, 163);
        _addPool(0x0D8Ce2A99Bb6e3B7Db580eD848240e4a0F9aE153, days_360, 243);

        _addPool(0xbA2aE424d960c26247Dd6c32edC70B295c744C43, 0, 80);
        _addPool(0xbA2aE424d960c26247Dd6c32edC70B295c744C43, days_180, 125);
        _addPool(0xbA2aE424d960c26247Dd6c32edC70B295c744C43, days_360, 200);
    }
    function _addPool(
        address _token,
        uint256 _lockTime,
        uint256 _dayRate
    ) internal {
        uint256 poolId = (_lockTime << 192) | uint160(_token); //Calculate pool ID
        require(!isExisted[poolId], "Already Added"); //Check whether the pool already exists. It is
forbidden to create a pool with the same ID
        isExisted[poolId] = true; //Update pool status
        allPools.push(poolId); //Add pool
        dayRate[poolId] = _dayRate; //Update rate
        emit AddPool(_token, _lockTime, poolId);
    }
```

Safety advice： NONE.

8.3.2. Transferadmin logic design 【security】

Audit description： Audit the business logic design of admin replacement in the contract, check whether the relevant business logic is reasonable, whether the parameters are checked, and check the authority of the caller.

Audit results： The transferadmin function in the contract only allows the initial admin user of the contract (i.e. the deployment user of the contract) to call. At the same time, the legitimacy of the parameters is strictly checked, and the relevant business logic design is correct.

Code file： DPMCPools.sol 320~326

Code information：

```
        function transferAdmin(address newOwner) public onlyDeveloper { //Only the deployment user of the contract is allowed to call
            require(
                newOwner != address(0),
                "Ownable: new owner is the zero address"
            ); //Address non-zero check
            _deployer = newOwner; //Update deployment user address
        }
        modifier onlyDeveloper() {
            require(msg.sender == _deployer, "not owner of tree"); //Only the deployment user of the contract is allowed to call
            _;
        }
```

Safety advice： NONE.

8.3.3. Setoperator set operation white list 【security】

Audit results： Conduct security audit on the business logic design of setoperator setting operation user white list address in the contract, check whether the relevant business logic is reasonable, whether the parameters are checked, and check the caller's permissions.

Audit results： The setoperator function in the contract only allows the deployment user of the contract to call, and the relevant business logic design is correct.

Code file： DPMCPools.sol 328~330

Code information：

```
    function setOperator(address _add, bool flag) public onlyDeveloper {//Only contract deployment user calls are allowed
        isOperator[_add] = flag;
    }
    modifier onlyDeveloper() {
        require(msg.sender == _deployer, "not owner of tree"); //Only the deployment user of the contract is allowed to call
        _;
    }
```

Safety advice： NONE.

8.3.4. Setpar relation binding business logic 【security】

Audit description： Conduct security audit on the business logic design of setpar relationship binding in the contract, check whether the parameters are checked, and whether the relevant business logic is reasonable, etc.

Audit results： The setpar business logic design in the contract is reasonable and correct.

Code file： DPMCPools.sol 332~340

Code information：

```
    function setPar(address _par) public {
            require(isValied(_par), "invalid parent"); //Address validity check
            // require(!isValied[msg.sender], 'already played');
            require(_par != msg.sender, "can not bind self"); //You cannot be your own parent
            require(_superior[msg.sender] == address(0), "already has parent"); //Current address has no
parent
            _superior[msg.sender] = _par; //Bind parent relationship
            _childrens[_par].push(msg.sender); //Bind subordinate relationship
            emit BindAddress(msg.sender, _par);
    }
     function isValied(address _address) public view returns (bool) {
            return _isValied[_address] || _address == leadAddress; //The address is valid or the address is
a lead address
     }
```

Safety advice： NONE.

8.3.5. Hierarchical relationship retrieval business 【Moderate risk】

Audit description： Audit the business logic design related to the hierarchical relationship retrieval in the contract, and check whether the logic design is reasonable and whether there is a risk of denial of service attack.

Audit results： The design of the business logic getchildrencount and getchildat of the parent-child relationship retrieval in the contract is correct, but there is a design defect in the business logic getchildren of the relationship retrieval. This function is used to obtain an array set of consecutive n child addresses starting from from the child list of a specified address, but it is not verified here_ From address. At the same time, the maximum number of obtained data is not checked. If the address account has more than 100 subordinates, multiple searches at a single time will lead to self DOS. At the same time, if it is searched in a small range, for example: five subordinates, search from the third, and retrieve 10 data in turn, there will be 7 applications for free memory space, which has design defects.

Code file: DPMCPools.sol 354~390

Code information:

```
    //Retrieve the number of subordinates
    function getChildrenCount(address account)
        public
        view
        override
        returns (uint256)
    {
        return _childrens[account].length;
    }
    //Get the subscript of a subordinate
    function getChildAt(address account, uint256 index)
        public
        view
        override
        returns (address)
    {
        return _childrens[account][index];
    }
    //Get a specified number of subordinate from
    function getChilds(
        address account,
        uint256 _from,
        uint256 _length
    ) public view returns (address[] memory) {
        address[] memory _childs = new address[](_length);
        for (uint256 i = 0; i < _length; i++) {//Without checking the maximum number of accesses, if
the address account has more than 100 subordinates, multiple searches at a time will lead to self DOS
            if (_childrens[account].length > i + _from) { //There is a design defect, for example: five
subordinate levels, search from the third, and search 10 data in turn, then 7 free memory will be applied
for, and there is a design defect
                    _childs[i] = _childrens[account][i + _from];
               }
          }
          return _childs;
          }
```

Safety advice： Check_ From and length parameters to prevent self dos and abnormal data values.

8.3.6. Addpool new pool business 【security】

Audit description： Conduct security audit on the business logic design of the new pool of addpool in the contract, and check whether there are design defects in the business logic design.

Audit results： In the contract, addpool checks whether the pool already exists when creating a new pool, which can ensure the uniqueness of the pool and the correct design of relevant business logic.

Code file： DPMCPools.sol 482~488

Code information：

```
        function addPool(
         address token,
         uint256 lockTime,
         uint256 _dayRate
    ) public onlyOperator {
         _addPool(token, lockTime, _dayRate); //Call_ Addpool add pool
    }

    function _addPool(
         address _token,
         uint256 _lockTime,
         uint256 _dayRate
    ) internal {
         uint256 poolId = (_lockTime << 192) | uint160(_token); //Calculate pool ID
         require(!isExisted[poolId], "Already Added"); //Check whether the pool already exists. It is
forbidden to create a pool with the same ID
         isExisted[poolId] = true; //Update pool status
         allPools.push(poolId); //Add pool
         dayRate[poolId] = _dayRate; //Update rate
         emit AddPool(_token, _lockTime, poolId);
    }
```

Safety advice： NONE.

8.3.7. Stakebnb pledge BNB logic 【Moderate risk】

Audit description： Conduct security audit on the business logic design of pledge BNB in the contract, and check whether there are design defects in the business logic design.

Audit results： There are some design defects in the business logic design of pledge BNB in the contract. According to the function name stakebnb, the main function of changing the function is to pledge BNB, and the change project runs on the BSC chain, and the gas fee is used as the pledge quantity, but here uniswapv2router is used as the router (according to the address query, it is actually pancakeswap), and at the same time, it is required when pledging BNB_ The token address is a zero address, which results in the getusdtvaluefrom function_ The token address is given with the wet address, which is inconsistent with the chain information deployed by the project, and there is a certain error with the logical design of the project. Please check by the project party.

Code file： DPMCPools.sol 503~525

Code information：

```
 function stakeBNB(uint256 poolId) public payable {
        require(isExisted[poolId], "not existed");   //Check whether the pool exists
        require(!isBaned[poolId], "not allowed");    //Check whether the pool is allowed to be pledged
        require(superior(msg.sender) != address(0), "Bind first");   //Check whether there is a superior relationship (invitee). If not, pledge is not allowed

        (address token, ) = infosFromPoolId(poolId);   //Get pool information
        require(token == address(0)); //Require token as 0 address

        uint256 amount = msg.value; //Gas with pledge quantity of BNB

        OrderInfo memory _info = OrderInfo(
            poolId,
            block.timestamp,
            amount,
            block.timestamp,
            false
        ); //Create pledge order
        OrderInfo[] storage _orders = stakedOrders[msg.sender];
        _orders.push(_info); //Add to order list
        _setValid(msg.sender); //Update the pledge user address to a valid trusted address
        uint usdtValue = getUsdtValueFrom(token, amount); //Calculate usdt quantity
        emit Staked(msg.sender, token, amount, poolId, usdtValue);
    }
    function infosFromPoolId(uint256 _poolId)
        public
        pure
        returns (address token, uint256 lockTime)
    {
        assembly {
            mstore(0, _poolId)
            token := mload(0)
        }
        lockTime = _poolId >> 192;
    }
    function getUsdtValueFrom(address _token, uint amount) public view returns(uint) {
        if (amount == 0){ //If the pledge quantity is 0, 0 will be returned directly
            return 0;
        }
        if (_token == address(0)){ //Address check, assign weth contract address
            _token = uniswapV2Router.WETH();
        }
        if (_token == usdtToken) {//Address check
            return amount;
        }
        IUniswapV2Factory _factory = IUniswapV2Factory(uniswapV2Router.factory());

        address _pairAddress = _factory.getPair(usdtToken, _token); //Obtain transaction pair
information
        if (_pairAddress == address(0)) { //Confirm that there are transaction pairs
            return 0;
        }
        IUniswapV2Pair _pair = IUniswapV2Pair(_pairAddress);
        (uint reserve0, uint reserve1, ) =   _pair.getReserves();
        if (_pair.token0() == usdtToken) {
            return uniswapV2Router.quote(amount, reserve1, reserve0); //Calculate the number of
available usdts
        }else if (_pair.token0() == _token) {
            return uniswapV2Router.quote(amount, reserve0, reserve1); //Calculate the number of
available usdts
        }
    }
```

Safety advice： Please check the router used by the project party and the contract address information used in the project.

8.3.8. Logical design of stack token pledge 【security】

Audit description： Conduct security audit on the token pledge business logic design in the contract, and check whether there are design defects in the business logic design.

Audit results： The logic design of token pledge business in the contract is correct.

Code file： DPMCPools.sol 527~553

Code information：

```
function stake(uint256 amount, uint256 poolId) public {
        require(isExisted[poolId], "not existed");   //Check whether the pool already exists
        require(!isBaned[poolId], "not allowed");   //Check whether the pool is allowed to be pledged
        require(superior(msg.sender) != address(0), "Bind first"); //Check whether there is a superior relationship (invitee)， If not, pledge is not allowed

        (address token, ) = infosFromPoolId(poolId); //Get pool information
        require(token != address(0));//The token is not required to be 0 address
        TransferHelper.safeTransferFrom(
            token,
            msg.sender,
            address(this),
            amount
        ); //Pledge operation

        OrderInfo memory _info = OrderInfo(
            poolId,
            block.timestamp,
            amount,
            block.timestamp,
            false
        );//Create pledge order
        OrderInfo[] storage _orders = stakedOrders[msg.sender];
        _orders.push(_info);//Add to order list
        _setValid(msg.sender);//Update the pledge user address to a valid trusted address
        uint usdtValue = getUsdtValueFrom(token, amount);//Calculate usdt quantity
        emit Staked(msg.sender, token, amount, poolId, usdtValue);
    }
    function infosFromPoolId(uint256 _poolId)
        public
        pure
        returns (address token, uint256 lockTime)
    {
        assembly {
            mstore(0, _poolId)
            token := mload(0)
        }
        lockTime = _poolId >> 192;
    }
    function safeTransferFrom(
        address token,
        address from,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0x23b872dd, from, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: TRANSFER_FROM_FAILED"
        );
    }
    function getUsdtValueFrom(address _token, uint amount) public view returns(uint) {
        if (amount == 0){ //If the pledge quantity is 0, 0 will be returned directly
            return 0;
        }
        if (_token == address(0)){ //Address check
            _token = uniswapV2Router.WETH();
        }
        if (_token == usdtToken) {//Address check
            return amount;
        }
        IUniswapV2Factory _factory = IUniswapV2Factory(uniswapV2Router.factory());

        address _pairAddress = _factory.getPair(usdtToken, _token); //Obtain transaction pair
information
        if (_pairAddress == address(0)) { //Confirm that there are transaction pairs
            return 0;
        }
        IUniswapV2Pair _pair = IUniswapV2Pair(_pairAddress);
        (uint reserve0, uint reserve1, ) =  _pair.getReserves();
        if (_pair.token0() == usdtToken) {
            return uniswapV2Router.quote(amount, reserve1, reserve0); //Calculate the number of
available usdts
        }else if (_pair.token0() == _token) {
            return uniswapV2Router.quote(amount, reserve0, reserve1); //Calculate the number of
available usdts
        }
```

Safety advice：：NONE.

8.3.9. Preunstake cancel pre pledge logic 【security】

Audit description： Conduct security audit on the logic design of canceling pre pledge business in the contract, and check whether there are design defects in the logic design of business, etc.

Audit results： The logic design of canceling pledge in the contract is correct.

Code file： DPMCPools.sol 555~572

Code information:

```
function preUnStake(uint256 _index) public {
        OrderInfo[] storage _orders = stakedOrders[msg.sender]; //Get the pledge order information of the function caller user
        require(_orders.length > _index, "not exist"); //Order is required to exist
        (
            address _token,
            ,
            uint256 unlockTime,
            uint256 _stakedAmount,
            uint256 rate,
            uint256 rewardAmount
        ) = userPoolsInfo(msg.sender, _index); //Obtain pledged order information
        require(unlockTime < block.timestamp, "not unlocked"); //Check whether the pledge release time meets the requirements
        claim(_index); //De pledge
        OrderInfo storage _info = _orders[_index];
        _info.preUnlocked = true; //update preUnlocked

        emit PreUnStake(msg.sender, _token, _stakedAmount, _info.poolId);
    }
    function userPoolsInfo(address _add, uint256 _index)
        public
        view
        returns (
            address _token,
            uint256 stakeTime,
            uint256 unlockTime,
            uint256 _stakedAmount,
            uint256 rate,
            uint256 interest
        )
    {
        OrderInfo[] memory _orders = stakedOrders[_add];    //Pledge order
        if (_index >= _orders.length) { //Whether the index is valid
```

```
                return (address(0), 0, 0, 0, 0, 0); //Invalid directly put back address (0), 0, 0, 0, 0, 0
            } else {
                OrderInfo memory _info = _orders[_index]; //Retrieve information
                if (_info.stakedAmount == 0) { //If the pledge quantity is 0, update the interest to 0
                    interest = 0;
                }
                uint256 pastDays = block.timestamp - _info.rewardTime; //Calculate the past time
                interest =
                    (pastDays * dayRate[_info.poolId] * _info.stakedAmount) /
                    ONE_DAY /
                    100000; //Calculate interest
                (address token, uint256 lockTimes) = infosFromPoolId(_info.poolId); //Get the token
address and locking time
                _token = token;
                unlockTime = _info.stakedTime + lockTimes;
                _stakedAmount = _info.stakedAmount;
                rate = dayRate[_info.poolId];
                stakeTime = _info.stakedTime;
                if (_info.preUnlocked) {
                    interest = 0;
                }
            }
        }
        function claim(uint256 _index) public {
            (address token, , , , , uint256 rewardAmount) = userPoolsInfo(
                msg.sender,
                _index
            ); //Get the information of the pool pledged by the user
            if (rewardAmount > 0) { //Check whether the number of rewards is greater than 0
                OrderInfo[] storage _orders = stakedOrders[msg.sender]; //Get list information
                OrderInfo storage _info = _orders[_index];
                require(!_info.preUnlocked, "Already Unlocked");    //Check the value of preunlocked
(initialized to false)
                _info.rewardTime = block.timestamp; //Get reward time
                if (token == address(0)) { //Check whether the token is a zero address
                    payable(msg.sender).transfer(rewardAmount);
                } else {
                    TransferHelper.safeTransfer(token, msg.sender, rewardAmount);
                }
                gottenAmount[msg.sender][token] =
                    gottenAmount[msg.sender][token] +
                    rewardAmount; //Update quantity
                emit Claimed(msg.sender, token, rewardAmount, _info.poolId);

                address _par = superior(msg.sender); //Father generation
                address _par2 = superior(_par);         //The second generation of father
                address _par3 = superior(_par2);        //Three generations of father
                if (_par != address(0)) {
                    uint256 needPay = (rewardAmount * 10) / 100;      //10%
                    if (token == address(0)) {
                        payable(msg.sender).transfer(needPay);
                    } else {
                        TransferHelper.safeTransfer(token, _par, needPay);
                    }
                    emit SendPar(msg.sender, token, _par, needPay, 1);
                    inviteReward[_par][token] = inviteReward[_par][token] + needPay;
                }
                if (_par2 != address(0)) {
                    uint256 needPay = (rewardAmount * 5) / 100;      //5%
                    if (token == address(0)) {
                        payable(msg.sender).transfer(needPay);
                    } else {
                        TransferHelper.safeTransfer(token, _par2, needPay);
                    }
                    emit SendPar(msg.sender, token, _par2, needPay, 2);
                    inviteReward[_par2][token] =
                        inviteReward[_par2][token] +
                        needPay;
                }
                if (_par3 != address(0)) {
                    uint256 needPay = (rewardAmount * 5) / 100;    //5%
                    if (token == address(0)) {
                        payable(msg.sender).transfer(needPay);
                    } else {
                        TransferHelper.safeTransfer(token, _par3, needPay);
                    }
                    emit SendPar(msg.sender, token, _par3, needPay, 3);
                    inviteReward[_par3][token] =
                        inviteReward[_par3][token] +
                        needPay;
                }
            }
        }
```

Safety advice：：NONE.

8.3.10. Unstake cancel pledge logic design 【security】

Audit description： Conduct security audit on the logic design of the pledge cancellation business in the contract, and check whether there are design defects in the logic design of the business.

Audit results： The logic design of canceling pledge in the contract is correct.

Code file： DPMCPools. salt 574~601

Code information：

```
function unStake(uint256 _index) public {
        OrderInfo[] storage _orders = stakedOrders[msg.sender];//Get the pledge order information
of the function caller user
        require(_orders.length > _index, "not exist");//Order is required to exist
        (
            address _token,
            ,
            uint256 unlockTime,
            uint256 _stakedAmount,
            uint256 rate,
            uint256 rewardAmount
        ) = userPoolsInfo(msg.sender, _index);//Obtain pledged order information

        OrderInfo memory _info = _orders[_index];
        require(_info.preUnlocked, "Pre Unlock first"); //Check whether the pledge release time
meets the requirements
        require(
            _info.rewardTime + ONE_DAY < block.timestamp,
            "Not unlocked one day"
        ); //Check whether the reward withdrawal time meets the time requirements
        _orders[_index] = _orders[_orders.length - 1];
        _orders.pop();    //更新_orders
        if (_token == address(0)) {
            payable(msg.sender).transfer(_stakedAmount);
        } else {
            TransferHelper.safeTransfer(_token, msg.sender, _stakedAmount);
        }

        emit UnStaked(msg.sender, _token, _stakedAmount, _info.poolId);
  }
```

Safety advice： NONE.

8.3.11. Userpoolsdetaileinfo pool details 【security】

Audit description： Conduct security audit on the business logic design that obtains the detailed information of the pool in the contract, and check whether there are design defects in the business logic design.

Audit results：The business logic design for obtaining the detailed information of the pool in the contract is correct.

Code file： DPMCPools.sol 640~677

Code information：

```
    function userPoolsDetailInfo(address _add, uint256 _index)
        public
        view
        returns (
            address _token,
            uint256 stakeTime,
            uint256 unlockTime,
            uint256 _stakedAmount,
            uint256 rate,
            uint256 interest,
            uint256 preUnstakeTime
        )
    {
        OrderInfo[] memory _orders = stakedOrders[_add];    //Get orders list
        if (_index >= _orders.length) {    //Check whether it is effective
            return (address(0), 0, 0, 0, 0, 0, 0);    //If there is no list, put it back directly to 0
        } else {
            OrderInfo memory _info = _orders[_index];    //Query the sub details of the list according to the index
            if (_info.stakedAmount == 0) {
                interest = 0;
            }
            uint256 pastDays = block.timestamp - _info.rewardTime; //Calculate the past time
            interest =
                (pastDays * dayRate[_info.poolId] * _info.stakedAmount) /
                ONE_DAY /
                100000; //Calculate interest
            (address token, uint256 lockTimes) = infosFromPoolId(_info.poolId);
            _token = token;
            unlockTime = _info.stakedTime + lockTimes;
            _stakedAmount = _info.stakedAmount;
            rate = dayRate[_info.poolId];
            stakeTime = _info.stakedTime;
            if (_info.preUnlocked) {
                interest = 0;
                preUnstakeTime = _info.rewardTime;
            }
        }
    }
```

Safety advice：：NONE.

8.3.12. Batchuserpoolsinfo batch get information【security】

Audit description： Conduct security audit on the business logic design of batch obtaining pool details in the contract, and check whether there are design defects in the business logic design.

Audit results： The business logic design of batch obtaining pool details in the contract is correct.

Code file： DPMCPools.sol 745~784

Code information：

```
function batchUserPoolsInfo(
        address _add,
        uint256 _from,
        uint256 _length
    )
        public
        view
        returns (
            address[] memory tokens,
            uint256[] memory unlockTimes,
            uint256[] memory stakedAmounts,
            uint256[] memory rates,
            uint256[] memory interests
        )
    {
        require(_from + _length <= orderLengthOf(_add), "Out of bound");    //Out of range

        unlockTimes = new uint256[](_length);
        stakedAmounts = new uint256[](_length);
        rates = new uint256[](_length);
        interests = new uint256[](_length);
        tokens = new address[](_length);
        for (uint256 i = _from; i < _length + _from; i++) {
            (
                address a1,
                uint256 r0,
                uint256 r1,
                uint256 r2,
                uint256 r3,
                uint256 r4
            ) = userPoolsInfo(_add, i);

            tokens[i - _from] = a1;
            unlockTimes[i - _from] = r1;
            stakedAmounts[i - _from] = r2;
            rates[i - _from] = r3;
            interests[i - _from] = r4;
        }
    }
```

Safety advice： NONE.

8.3.13. Poolslist pool sublist retrieval 【security】

Audit description： Conduct security audit on the business logic design of retrieving pool list information in the contract, and check whether there are design defects in the business logic design.

Audit results： The business logic design of retrieving pool list information in the contract is correct.

Code file： DPMCPools.sol 805~827

Code information：

CHAINLION

```
function poolsList(uint256 _from, uint256 _length)
        public
        view
        returns (
            address[] memory tokens,
            uint256[] memory lockTimes,
            uint256[] memory rates,
            uint256[] memory poolIds
        )
    {
        require(_from + _length <= poolsCount(), "Out of bound");    //Check whether it is out of range
        tokens = new address[](_length);
        lockTimes = new uint256[](_length);
        rates = new uint256[](_length);
        poolIds = new uint256[](_length);
        for (uint256 i = _from; i < _length + _from; i++) {
            (address a1, uint256 r1, uint256 r2, uint256 r3) = poolAt(i);
            tokens[i - _from] = a1;
            lockTimes[i - _from] = r1;
            rates[i - _from] = r2;
            poolIds[i - _from] = r3;
        }
    }
    function poolAt(uint256 _index)
        public
        view
        returns (
            address _token,
            uint256 _lockTime,
            uint256 _rate,
            uint256 _poolId
        )
    {
        if (_index >= poolsCount()) {
            return (address(0), 0, 0, 0);
        } else {
            _poolId = allPools[_index];
            (_token, _lockTime) = infosFromPoolId(_poolId);
            _rate = dayRate[_poolId];
        }
    }
    function poolsCount() public view returns (uint256) {
        return allPools.length;
    }
    function infosFromPoolId(uint256 _poolId)
        public
        pure
        returns (address token, uint256 lockTime)
    {
        assembly {
            mstore(0, _poolId)
            token := mload(0)
        }
        lockTime = _poolId >> 192;
    }
```

Safety advice： NONE.

8.3.14. Contract authority concentration detection 【Low risk】

Audit description： Detect the concentration of authority in the contract and check whether the relevant business logic is reasonable.

Audit results： In the contract, the operator has high permissions, which can be used to add pools and set rates. The permissions are relatively centralized. It is recommended that the project party properly manage the operator account.

Code file： DPMCPools.sol 845~847

Code information：

```
modifier onlyDeveloper() {
        require(msg.sender == _deployer, "not owner of tree"); //Only the deployment user of the contract is allowed to call
        _;
    }
    function setOperator(address _add, bool flag) public onlyDeveloper {//Only the deployment user of the contract is allowed to call
        isOperator[_add] = flag;
    }
    modifier onlyOperator() {
        require(isOperator[msg.sender], "not Operator of tree"); //Only operation user whitelist address calls are allowed
        _;
    }
    function addPool(
        address token,
        uint256 lockTime,
        uint256 _dayRate
    ) public onlyOperator {
        _addPool(token, lockTime, _dayRate); //Call_ Addpool add pool
    }

    function _addPool(
        address _token,
        uint256 _lockTime,
        uint256 _dayRate
    ) internal {
        uint256 poolId = (_lockTime << 192) | uint160(_token); //Calculate pool ID
        require(!isExisted[poolId], "Already Added"); //Check whether the pool already exists. It is forbidden to create a pool with the same ID
        isExisted[poolId] = true; //Update pool status
        allPools.push(poolId); //Add pool
        dayRate[poolId] = _dayRate; //Update rate
        emit AddPool(_token, _lockTime, poolId);
    }

    function setRate(uint256 _rate, uint256 _poolId) public onlyOperator {
        dayRate[_poolId] = _rate;
    }
```

Safety advice： It is recommended that the project party properly keep the operator authority.

9. Contract source code

```
pragma solidity ^0.8.0;

library TransferHelper {
    function safeApprove(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0x095ea7b3, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: APPROVE_FAILED"
        );
    }

    function safeTransfer(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0xa9059cbb, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: TRANSFER_FAILED"
        );
    }

    function safeTransferFrom(
        address token,
        address from,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0x23b872dd, from, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: TRANSFER_FROM_FAILED"
        );
    }

    function safeMint(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('mint(address,uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0x40c10f19, to, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: MINT_FAILED"
        );
    }
    function safeBurn(address token, uint256 value) internal {
        // bytes4(keccak256(bytes('burn(uint256)')));
        (bool success, bytes memory data) = token.call(
            abi.encodeWithSelector(0x42966c68, value)
        );
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "TransferHelper: BURN_FAILED"
        );
    }
}



interface IUniswapV2Factory {
    // event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    // function feeTo() external view returns (address);
    // function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    // function allPairs(uint) external view returns (address pair);
    // function allPairsLength() external view returns (uint);

    // function createPair(address tokenA, address tokenB) external returns (address pair);

    // function setFeeTo(address) external;
    // function setFeeToSetter(address) external;
}

interface IUniswapV2Pair {
//      event Approval(address indexed owner, address indexed spender, uint value);
//      event Transfer(address indexed from, address indexed to, uint value);

//      function name() external pure returns (string memory);
//      function symbol() external pure returns (string memory);
//      function decimals() external pure returns (uint8);
//      function totalSupply() external view returns (uint);
//      function balanceOf(address owner) external view returns (uint);
//      function allowance(address owner, address spender) external view returns (uint);

//      function approve(address spender, uint value) external returns (bool);
//      function transfer(address to, uint value) external returns (bool);
//      function transferFrom(address from, address to, uint value) external returns (bool);

//      function DOMAIN_SEPARATOR() external view returns (bytes32);
//      function PERMIT_TYPEHASH() external pure returns (bytes32);
//      function nonces(address owner) external view returns (uint);

//      function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r,
bytes32 s) external;

//      event Mint(address indexed sender, uint amount0, uint amount1);
//      event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
//      event Swap(
//          address indexed sender,
//          uint amount0In,
//          uint amount1In,
//          uint amount0Out,
//          uint amount1Out,
//          address indexed to
//      );
//      event Sync(uint112 reserve0, uint112 reserve1);

//      function MINIMUM_LIQUIDITY() external pure returns (uint);
//      function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32
blockTimestampLast);
//      function price0CumulativeLast() external view returns (uint);
//      function price1CumulativeLast() external view returns (uint);
//      function kLast() external view returns (uint);

//      function mint(address to) external returns (uint liquidity);
//      function burn(address to) external returns (uint amount0, uint amount1);
//      function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
//      function skim(address to) external;
//      function sync() external;

//      function initialize(address, address) external;
}

interface IUniswapV2Router01 {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    // function addLiquidityETH(
    //      address token,
    //      uint amountTokenDesired,
    //      uint amountTokenMin,
    //      uint amountETHMin,
    //      address to,
    //      uint deadline
    // ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    // function removeLiquidity(
    //      address tokenA,
//          address tokenB,
//          uint liquidity,
//          uint amountAMin,
//          uint amountBMin,
//          address to,
//          uint deadline
// ) external returns (uint amountA, uint amountB);
// function removeLiquidityETH(
//          address token,
//          uint liquidity,
//          uint amountTokenMin,
//          uint amountETHMin,
//          address to,
//          uint deadline
// ) external returns (uint amountToken, uint 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 returns (uint amountA, uint amountB);
// function removeLiquidityETHWithPermit(
//          address token,
//          uint liquidity,
//          uint amountTokenMin,
//          uint amountETHMin,
//          address to,
//          uint deadline,
//          bool approveMax, uint8 v, bytes32 r, bytes32 s
// ) external returns (uint amountToken, uint amountETH);
// function swapExactTokensForTokens(
//          uint amountIn,
    //        uint amountOutMin,
    //        address[] calldata path,
    //        address to,
    //        uint deadline
    // ) external returns (uint[] memory amounts);
    // function swapTokensForExactTokens(
    //        uint amountOut,
    //        uint amountInMax,
    //        address[] calldata path,
    //        address to,
    //        uint deadline
    // ) external returns (uint[] memory amounts);
    // function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint
deadline)
    //        external
    //        payable
    //        returns (uint[] memory amounts);
    // function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path,
address to, uint deadline)
    //        external
    //        returns (uint[] memory amounts);
    // function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path,
address to, uint deadline)
    //        external
    //        returns (uint[] memory amounts);
    // function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint
deadline)
    //        external
    //        payable
    //        returns (uint[] memory amounts);

    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
    // function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns
(uint amountOut);
    // function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns
(uint amountIn);
```

```
    // function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[]
memory amounts);
    // function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[]
memory amounts);
}

interface IUniswapV2Router02 is IUniswapV2Router01 {
    // function removeLiquidityETHSupportingFeeOnTransferTokens(
    //      address token,
    //      uint liquidity,
    //      uint amountTokenMin,
    //      uint amountETHMin,
    //      address to,
    //      uint deadline
    // ) external returns (uint amountETH);
    // function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
    //      address token,
    //      uint liquidity,
    //      uint amountTokenMin,
    //      uint amountETHMin,
    //      address to,
    //      uint deadline,
    //      bool approveMax, uint8 v, bytes32 r, bytes32 s
    // ) external returns (uint amountETH);

    // function swapExactTokensForTokensSupportingFeeOnTransferTokens(
    //      uint amountIn,
    //      uint amountOutMin,
    //      address[] calldata path,
    //      address to,
    //      uint deadline
    // ) external;
    // function swapExactETHForTokensSupportingFeeOnTransferTokens(
    //      uint amountOutMin,
    //      address[] calldata path,
    //      address to,
    //          uint deadline
    // ) external payable;
    // function swapExactTokensForETHSupportingFeeOnTransferTokens(
    //          uint amountIn,
    //          uint amountOutMin,
    //          address[] calldata path,
    //          address to,
    //          uint deadline
    // ) external;
}

interface ITree {
    function superior(address account) external view returns (address);

    function getChildrenCount(address account) external view returns (uint256);

    function getChildAt(address account, uint256 index)
        external
        view
        returns (address);

    function isValied(address _address) external view returns (bool);
    // function setValid(address _add) external    returns (bool);
}

contract DPMCtree is ITree {
    //上级关系
    mapping(address => address) private _superior;
    mapping(address => address[]) private _childrens;
    address _deployer;
    address public leadAddress;
    mapping(address => bool) internal isOperator;
    mapping(address => bool) _isValied;

    event BindAddress(address indexed _add, address indexed _par);
```

```
constructor() public {
    _deployer = msg.sender;
    leadAddress = msg.sender;
}

modifier onlyDeveloper() {
    require(msg.sender == _deployer, "not owner of tree");
    _;
}
modifier onlyOperator() {
    require(isOperator[msg.sender], "not Operator of tree");
    _;
}

function transferAdmin(address newOwner) public onlyDeveloper {
    require(
        newOwner != address(0),
        "Ownable: new owner is the zero address"
    );
    _deployer = newOwner;
}

function setOperator(address _add, bool flag) public onlyDeveloper {
    isOperator[_add] = flag;
}

function setPar(address _par) public {
    require(isValied(_par), "invalid parent");
    // require(!isValied[msg.sender], 'already played');
    require(_par != msg.sender, "can not bind self");
    require(_superior[msg.sender] == address(0), "already has parent");
    _superior[msg.sender] = _par;
    _childrens[_par].push(msg.sender);
    emit BindAddress(msg.sender, _par);
}
```

```
function _setValid(address _add) internal returns (bool) {
        if (!isValied(_add)) {
                _isValied[_add] = true;
                return true;
        }
        return false;
}

function isValied(address _address) public view returns (bool) {
        return _isValied[_address] || _address == leadAddress;
}

function superior(address account) public view override returns (address) {
        return _superior[account];
}

function getChildrenCount(address account)
        public
        view
        override
        returns (uint256)
{
        return _childrens[account].length;
}

function getChildAt(address account, uint256 index)
        public
        view
        override
        returns (address)
{
        return _childrens[account][index];
}

function getChilds(
        address account,
        uint256 _from,
        uint256 _length
    ) public view returns (address[] memory) {
        address[] memory _childs = new address[](_length);
        for (uint256 i = 0; i < _length; i++) {
            if (_childrens[account].length > i + _from) {
                _childs[i] = _childrens[account][i + _from];
            }
        }
        return _childs;
    }

}

contract DPMCStakePoolStorage is DPMCtree {
    struct OrderInfo {
        uint256 poolId;
        uint256 stakedTime;
        uint256 stakedAmount;
        uint256 rewardTime;
        bool preUnlocked;
    }

    IUniswapV2Router02 public uniswapV2Router;
    address usdtToken = 0x55d398326f99059fF775485246999027B3197955;
    mapping(address => OrderInfo[]) public stakedOrders;
    mapping(address => mapping(address => uint256)) public gottenAmount;
    mapping(address => mapping(address => uint256)) public inviteReward;
    mapping(address => uint256) public gottenTeamAmount;
    mapping(uint256 => bool) isBaned;
    mapping(uint256 => bool) isExisted;
    mapping(uint256 => uint256) public dayRate; // / 10000
    uint256[] public allPools;

    uint256 ONE_DAY = 86400;
    constructor() public {
        uniswapV2Router                                                                    =
IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);

        uint256 days_180 = ONE_DAY * 180;
        uint256 days_360 = ONE_DAY * 360;

        _addPool(usdtToken, 0, 83);
        _addPool(usdtToken, days_180, 143);
        _addPool(usdtToken, days_360, 220);

        _addPool(address(0x0), 0, 55);
        _addPool(address(0x0), days_180, 91);
        _addPool(address(0x0), days_360, 140);

        _addPool(0x0D8Ce2A99Bb6e3B7Db580eD848240e4a0F9aE153, 0, 96);
        _addPool(0x0D8Ce2A99Bb6e3B7Db580eD848240e4a0F9aE153, days_180, 163);
        _addPool(0x0D8Ce2A99Bb6e3B7Db580eD848240e4a0F9aE153, days_360, 243);

        _addPool(0xbA2aE424d960c26247Dd6c32edC70B295c744C43, 0, 80);
        _addPool(0xbA2aE424d960c26247Dd6c32edC70B295c744C43, days_180, 125);
        _addPool(0xbA2aE424d960c26247Dd6c32edC70B295c744C43, days_360, 200);



    }

    event Staked(
        address indexed from,
        address indexed token,
        uint256 amount,
        uint256 poolId,
        uint256 usdtValue
    );
    event UnStaked(
        address indexed from,
        address indexed token,
        uint256 amount,
        uint256 poolId
    );
    event PreUnStake(
        address indexed from,
        address indexed token,
        uint256 amount,
        uint256 poolId
    );
    event Claimed(
        address indexed from,
        address indexed token,
        uint256 amount,
        uint256 poolId
    );
    event SendPar(
        address indexed from,
        address indexed token,
        address indexed par,
        uint256 amount,
        uint256 distance
    );
    event AddPool(
        address indexed token,
        uint256 lockTime,
        uint256 poolId
    );




    function addPool(
        address token,
        uint256 lockTime,
        uint256 _dayRate
    ) public onlyOperator {
        _addPool(token, lockTime, _dayRate);
    }

    function _addPool(
        address _token,
        uint256 _lockTime,
        uint256 _dayRate
    ) internal {
        uint256 poolId = (_lockTime << 192) | uint160(_token);
        require(!isExisted[poolId], "Already Added");
        isExisted[poolId] = true;
        allPools.push(poolId);
        dayRate[poolId] = _dayRate;
        emit AddPool(_token, _lockTime, poolId);
    }

    function stakeBNB(uint256 poolId) public payable {
        require(isExisted[poolId], "not existed");
        require(!isBaned[poolId], "not allowed");
        require(superior(msg.sender) != address(0), "Bind first");

        (address token, ) = infosFromPoolId(poolId);
        require(token == address(0));

        uint256 amount = msg.value;

        OrderInfo memory _info = OrderInfo(
            poolId,
            block.timestamp,
            amount,
            block.timestamp,
            false
        );
        OrderInfo[] storage _orders = stakedOrders[msg.sender];
        _orders.push(_info);
        _setValid(msg.sender);
        uint usdtValue = getUsdtValueFrom(token, amount);
        emit Staked(msg.sender, token, amount, poolId, usdtValue);
    }

    function stake(uint256 amount, uint256 poolId) public {
        require(isExisted[poolId], "not existed");
        require(!isBaned[poolId], "not allowed");
        require(superior(msg.sender) != address(0), "Bind first");

        (address token, ) = infosFromPoolId(poolId);
        require(token != address(0));
        TransferHelper.safeTransferFrom(
            token,
            msg.sender,
            address(this),
            amount
        );

        OrderInfo memory _info = OrderInfo(
            poolId,
            block.timestamp,
            amount,
            block.timestamp,
            false
        );
        OrderInfo[] storage _orders = stakedOrders[msg.sender];
        _orders.push(_info);
        _setValid(msg.sender);
        uint usdtValue = getUsdtValueFrom(token, amount);
        emit Staked(msg.sender, token, amount, poolId, usdtValue);
    }

    function preUnStake(uint256 _index) public {
        OrderInfo[] storage _orders = stakedOrders[msg.sender];
        require(_orders.length > _index, "not exist");
        (
            address _token,
            ,
            uint256 unlockTime,
            uint256 _stakedAmount,
            uint256 rate,
            uint256 rewardAmount
        ) = userPoolsInfo(msg.sender, _index);
        require(unlockTime < block.timestamp, "not unlocked");
        claim(_index);
        OrderInfo storage _info = _orders[_index];
        _info.preUnlocked = true;

        emit PreUnStake(msg.sender, _token, _stakedAmount, _info.poolId);
    }

    function unStake(uint256 _index) public {
        OrderInfo[] storage _orders = stakedOrders[msg.sender];
        require(_orders.length > _index, "not exist");
        (
            address _token,
            ,
            uint256 unlockTime,
            uint256 _stakedAmount,
            uint256 rate,
            uint256 rewardAmount
        ) = userPoolsInfo(msg.sender, _index);

        OrderInfo memory _info = _orders[_index];
        require(_info.preUnlocked, "Pre Unlock first");
        require(
            _info.rewardTime + ONE_DAY < block.timestamp,
            "Not unlocked one day"
        );
        _orders[_index] = _orders[_orders.length - 1];
        _orders.pop();
        if (_token == address(0)) {
            payable(msg.sender).transfer(_stakedAmount);
        } else {
            TransferHelper.safeTransfer(_token, msg.sender, _stakedAmount);
        }

        emit UnStaked(msg.sender, _token, _stakedAmount, _info.poolId);
    }

    function userPoolsInfo(address _add, uint256 _index)
        public
        view
        returns (
            address _token,
            uint256 stakeTime,
            uint256 unlockTime,
            uint256 _stakedAmount,
            uint256 rate,
            uint256 interest
        )
    {
        OrderInfo[] memory _orders = stakedOrders[_add];
        if (_index >= _orders.length) {
            return (address(0), 0, 0, 0, 0, 0);
        } else {
            OrderInfo memory _info = _orders[_index];
            if (_info.stakedAmount == 0) {
                interest = 0;
            }
            uint256 pastDays = block.timestamp - _info.rewardTime;
            interest =
                (pastDays * dayRate[_info.poolId] * _info.stakedAmount) /
                ONE_DAY /
                100000;
            (address token, uint256 lockTimes) = infosFromPoolId(_info.poolId);
            _token = token;
            unlockTime = _info.stakedTime + lockTimes;
            _stakedAmount = _info.stakedAmount;
            rate = dayRate[_info.poolId];
            stakeTime = _info.stakedTime;
            if (_info.preUnlocked) {
                interest = 0;
            }
        }
    }

    function userPoolsDetailInfo(address _add, uint256 _index)
        public
        view
        returns (
            address _token,
            uint256 stakeTime,
            uint256 unlockTime,
            uint256 _stakedAmount,
            uint256 rate,
            uint256 interest,
            uint256 preUnstakeTime
        )
    {
        OrderInfo[] memory _orders = stakedOrders[_add];
        if (_index >= _orders.length) {
            return (address(0), 0, 0, 0, 0, 0, 0);
        } else {
            OrderInfo memory _info = _orders[_index];
            if (_info.stakedAmount == 0) {
                interest = 0;
            }
            uint256 pastDays = block.timestamp - _info.rewardTime;
            interest =
                (pastDays * dayRate[_info.poolId] * _info.stakedAmount) /
                ONE_DAY /
                100000;
            (address token, uint256 lockTimes) = infosFromPoolId(_info.poolId);
            _token = token;
            unlockTime = _info.stakedTime + lockTimes;
            _stakedAmount = _info.stakedAmount;
            rate = dayRate[_info.poolId];
            stakeTime = _info.stakedTime;
            if (_info.preUnlocked) {
                interest = 0;
                preUnstakeTime = _info.rewardTime;
            }
        }
    }

    function claim(uint256 _index) public {
        (address token, , , , , uint256 rewardAmount) = userPoolsInfo(
            msg.sender,
            _index
        );
        if (rewardAmount > 0) {
            OrderInfo[] storage _orders = stakedOrders[msg.sender];
            OrderInfo storage _info = _orders[_index];
            require(!_info.preUnlocked, "Already Unlocked");
            _info.rewardTime = block.timestamp;
            if (token == address(0)) {
                payable(msg.sender).transfer(rewardAmount);
            } else {
                TransferHelper.safeTransfer(token, msg.sender, rewardAmount);
            }

            gottenAmount[msg.sender][token] =
                gottenAmount[msg.sender][token] +
                rewardAmount;
            emit Claimed(msg.sender, token, rewardAmount, _info.poolId);

            address _par = superior(msg.sender);
```

```
address _par2 = superior(_par);
address _par3 = superior(_par2);
if (_par != address(0)) {
    uint256 needPay = (rewardAmount * 10) / 100;
    if (token == address(0)) {
        payable(msg.sender).transfer(needPay);
    } else {
        TransferHelper.safeTransfer(token, _par, needPay);
    }
    emit SendPar(msg.sender, token, _par, needPay, 1);
    inviteReward[_par][token] = inviteReward[_par][token] + needPay;
}
if (_par2 != address(0)) {
    uint256 needPay = (rewardAmount * 5) / 100;
    if (token == address(0)) {
        payable(msg.sender).transfer(needPay);
    } else {
        TransferHelper.safeTransfer(token, _par2, needPay);
    }
    emit SendPar(msg.sender, token, _par2, needPay, 2);
    inviteReward[_par2][token] =
        inviteReward[_par2][token] +
        needPay;
}
if (_par3 != address(0)) {
    uint256 needPay = (rewardAmount * 5) / 100;
    if (token == address(0)) {
        payable(msg.sender).transfer(needPay);
    } else {
        TransferHelper.safeTransfer(token, _par3, needPay);
    }
    emit SendPar(msg.sender, token, _par3, needPay, 3);
    inviteReward[_par3][token] =
        inviteReward[_par3][token] +
        needPay;
}
```

```
        }
    }

    function orderLengthOf(address _add) public view returns (uint256) {
        OrderInfo[] memory _orders = stakedOrders[_add];
        return _orders.length;
    }

    function batchUserPoolsInfo(
        address _add,
        uint256 _from,
        uint256 _length
    )
        public
        view
        returns (
            address[] memory tokens,
            uint256[] memory unlockTimes,
            uint256[] memory stakedAmounts,
            uint256[] memory rates,
            uint256[] memory interests
        )
    {
        require(_from + _length <= orderLengthOf(_add), "Out of bound");

        unlockTimes = new uint256[](_length);
        stakedAmounts = new uint256[](_length);
        rates = new uint256[](_length);
        interests = new uint256[](_length);
        tokens = new address[](_length);
        for (uint256 i = _from; i < _length + _from; i++) {
            (
                address a1,
                uint256 r0,
                uint256 r1,
                uint256 r2,
                uint256 r3,
                uint256 r4
            ) = userPoolsInfo(_add, i);

            tokens[i - _from] = a1;
            unlockTimes[i - _from] = r1;
            stakedAmounts[i - _from] = r2;
            rates[i - _from] = r3;
            interests[i - _from] = r4;
        }

    }

    function poolAt(uint256 _index)
        public
        view
        returns (
            address _token,
            uint256 _lockTime,
            uint256 _rate,
            uint256 _poolId
        )
    {
        if (_index >= poolsCount()) {
            return (address(0), 0, 0, 0);
        } else {
            _poolId = allPools[_index];
            (_token, _lockTime) = infosFromPoolId(_poolId);
            _rate = dayRate[_poolId];
        }
    }

    function poolsList(uint256 _from, uint256 _length)
        public
        view
        returns (
            address[] memory tokens,
            uint256[] memory lockTimes,
            uint256[] memory rates,
            uint256[] memory poolIds
        )
    {
        require(_from + _length <= poolsCount(), "Out of bound");
        tokens = new address[](_length);
        lockTimes = new uint256[](_length);
        rates = new uint256[](_length);
        poolIds = new uint256[](_length);
        for (uint256 i = _from; i < _length + _from; i++) {
            (address a1, uint256 r1, uint256 r2, uint256 r3) = poolAt(i);
            tokens[i - _from] = a1;
            lockTimes[i - _from] = r1;
            rates[i - _from] = r2;
            poolIds[i - _from] = r3;
        }
    }

    function poolsCount() public view returns (uint256) {
        return allPools.length;
    }

    function infosFromPoolId(uint256 _poolId)
        public
        pure
        returns (address token, uint256 lockTime)
    {
        assembly {
            mstore(0, _poolId)
            token := mload(0)
        }
        lockTime = _poolId >> 192;
    }
    function setRate(uint256 _rate, uint256 _poolId) public onlyOperator {
        dayRate[_poolId] = _rate;
    }

    function getUsdtValueFrom(address _token, uint amount) public view returns(uint) {
        if (amount == 0){
            return 0;
        }
        if (_token == address(0)){
            _token = uniswapV2Router.WETH();
        }
        if (_token == usdtToken) {
            return amount;
        }
        IUniswapV2Factory _factory = IUniswapV2Factory(uniswapV2Router.factory());

        address _pairAddress = _factory.getPair(usdtToken, _token);
        if (_pairAddress == address(0)) {
            return 0;
        }
        IUniswapV2Pair _pair = IUniswapV2Pair(_pairAddress);
        (uint reserve0, uint reserve1, ) =    _pair.getReserves();
        if (_pair.token0() == usdtToken) {
            return uniswapV2Router.quote(amount, reserve1, reserve0);
        }else if (_pair.token0() == _token) {
            return uniswapV2Router.quote(amount, reserve0, reserve1);
        }
    }
}
```

10. Appendix:Analysis tools

10.1.Solgraph

Solgraph is used to generate a graph of the call relationship between smart contract functions, which is convenient for quickly understanding the call relationship between smart contract functions.

Project address： https://github.com/raineorshine/solgraph

10.2.Sol2uml

Sol2uml is used to generate the calling relationship between smart contract functions in the form of UML diagram.

Project address： https://github.com/naddison36/sol2uml

10.3.Remix-ide

Remix is a browser based compiler and IDE that allows users to build contracts and debug transactions using the solid language.

Project address： http://remix.ethereum.org

10.4.Ethersplay

Etherplay is a plug-in for binary ninja. It can be used to analyze EVM bytecode and graphically present the function call process.

Project address： https://github.com/crytic/ethersplay

10.5.Mythril

Mythril is a security audit tool for EVM bytecode, and supports online contract audit.

Project address： https://github.com/ConsenSys/mythril

10.6.Echidna

Echidna is a security audit tool for EVM bytecode. It uses fuzzy testing technology and supports integrated use with truss.

Project address： https://github.com/crytic/echidna

11. DISCLAIMERS

Chainlion only issues this report on the facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities. For the facts occurring or existing after the issuance, chainlion cannot judge the security status of its smart contract, and is not responsible for it. The security audit analysis and other contents in this report are only based on the documents and materials provided by the information provider to chainlion as of the issuance of this report. Chainlion assumes that the information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed or reflected inconsistent with the actual situation, chainlion shall not be liable for the losses and adverse effects caused thereby. Chainlion only conducted the agreed safety audit on the safety of the project and issued this report. Chainlion is not responsible for the background and other conditions of the project.

Blockchain world patron saint building blockchain ecological security