Smart contract audit report

BUMB

CHAINLION

NO. OC002208110001

August 11, 2022

CHAINLION

CATALOGUE

1. PROJECT SUMMARY

Entry type	Specific description
Entry name	BUMB
Project type	DEFI
Application platform	BSC
DawnToken	

2. AUDIT SUMMARY

Entry type	Specific description
Project cycle	August/08/2022-August/11/2022
Audit method	Black box test、White box test、Grey box test
Auditors	THREE

3. VULNERABILITY SUMMARY

Audit results are as follows:

Entry type	Specific description
Serious vulnerability	0
High risk vulnerability	0
Moderate risk	0

Low risk vulnerability	2

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 **BUMB** 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.

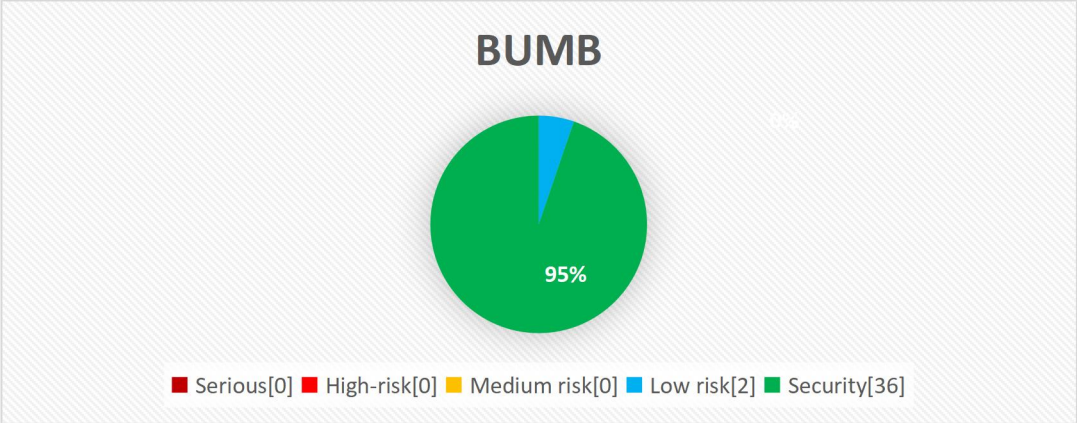
5. Directory structure

```
BUMB.sol
```

6. File hashes

Contract	SHA1 Checksum
BUMB.sol	CE4739AEDAA5D2A44EF08AF2BC1D7515FAC72BCE

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.

```
26  pragma solidity ^0.8.6;
27
28  // SPDX-License-Identifier: Unlicensed
29  interface IERC20 {
30      function totalSupply() external view returns (uint256);
31
32      /**
33       * @dev Returns the amount of tokens owned by `account`.
34       */
35      function balanceOf(address account) external view returns (uint256);
36
37      /**
38       * @dev Moves `amount` tokens from the caller's account to `recipient`.
39       *
40       * Returns a boolean value indicating whether the operation succeeded.
41       *
42       * Emits a {Transfer} event.
43       */
44      function transfer(address recipient, uint256 amount)
45          external
46          returns (bool);
47
```

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.

```
110  abstract contract Ownable {
111      address private _owner;
112      address private _previousOwner;
113      uint256 private _lockTime;
114
115      event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
116
117      constructor () {
118          address msgSender = msg.sender;
119          _owner = msgSender;
120          emit OwnershipTransferred(address(0), msgSender);
121      }
122
```

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.

```
995     function setspeed(uint256 _speed) external onlyvote() {
996         payspeed = _speed;
997     }
998
```

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.

```
980     function setprojectad(address projectAd) external onlyOwner(){
981         projectad = projectAd;
982         _isExcludedFromFee[projectAd] = true;
983     }
984
```

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.

```
26  pragma solidity ^0.8.6;
27
28  // SPDX-License-Identifier: Unlicensed
29  interface IERC20 {
30      function totalSupply() external view returns (uint256);
31
32      /**
33       * @dev Returns the amount of tokens owned by `account`.
34       */
35      function balanceOf(address account) external view returns (uint256);
36
37      /**
38       * @dev Moves `amount` tokens from the caller's account to `recipient`.
39       *
40       * Returns a boolean value indicating whether the operation succeeded.
41       *
42       * Emits a {Transfer} event.
43       */
44      function transfer(address recipient, uint256 amount)
45          external
46          returns (bool);
```

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 constructor initialization and business logic design in the contract, and check whether the initialization value is consistent with the requirements document.

Audit results： The constructor initialization business logic design in the contract is correct, and no relevant security risks are found.

Code file： BUMP.sol L617~620 L629~781

Code information：

```
using SafeMath for uint256;
address _token;

IERC20 usdt = IERC20(0x55d398326f99059fF775485246999027B3197955);
constructor () {
```

```
        _token = msg.sender;      //_token address is initialized to the deployer user address of the contract
    }
    using SafeMath for uint256;

    string private _name = "Bumb";         //Token name
    string private _symbol =    "Bumb";    //Token symbol
    uint8 private _decimals = 18;          //Token accuracy
    mapping(address => uint256) private _tOwned;
    uint256 private _tTotal = 2100*10**4 * 10**18;    //Total amount of tokens issued
    uint256 public _burnAward = 640;          //Destroyer dividends, fomo and liquidity
    uint256 public _inviterFee = 160;         //Level reward
    uint256 public _devoteFee = 300;          //Used to repay the creditor's rights of the destroyer
    uint256 public _DevoteFee = 1100;         //Used to repay the creditor's rights of the destroyer during transfer
    uint256 public _burninviter = 3200;     //Level allocation during destruction
    //uint256 public _price = 7*10**14;
    //uint256 public multiple = 1;

    address public beforeaction ;             //The manager who last called the voting and triggered the voting result
    address public projectad = 0x421C1Ac3d4492649E8d9646E978e4A996da7AEc1;        //Address of the project party
    address public projectDAO = 0xeBa2DeFb11134667362830cB2D065aFE6Ca70EaD; //Foundation address
    uint256 distributorGas = 200000;       //The gas fee for dividends goes online

    mapping(address => mapping(address => uint256)) private _allowances;     //Authorized limit mapping

    mapping(address => bool) private _isExcludedFromFee;     //Whitelist address

    address DEAD = 0x000000000000000000000000000000000000dEaD;    //Destruction address


    mapping(address => bool) public _iscommunity;       //Judge whether it is a manager
```

```
    address[] public communiters;     //Manager array
    mapping (address => uint256) public comdexes;       //Serial number of the array where the manager is located

    uint256 public votetime;         //Timestamp of voting
    uint256 public vote;             //Number of votes
    uint256 public votegap = 3600 seconds;   //Effective duration of voting

    mapping(uint256 => mapping(address => bool)) public _istimepoll;      //Has the corresponding manager voted during the current voting time

     modifier onlycommunity {           //Only managers can trigger
         require(_iscommunity[msg.sender]);
         _;
    }

    mapping(address => bool) public groupEquity;       //Allow nodes to buy in advance
    address[] public groupers;                         //Node array
    mapping(address => bool) public isgroup;           //Determine whether it is a node
    mapping (address => uint256) public groupdexes; //The sequence number of the array where the node is located
    mapping (address => uint256) public groupLock;   //Dynamic lock up amount
    mapping (address => uint256) public GroupLock;   //Fixed lock up amount
    mapping (address => uint16) public GroupGrade;    //The level of the node, the large node is 2, and the small node is 1

    mapping (address => uint256) public Burnbusiness;     //Total destruction amount of the network of the node
    mapping (address => uint256) public Swapbusiness;     //Total purchase amount of the network of the node

    address public Totalprojectad;          //Address of total point
    uint256 public Totalnode;                //Total destruction and purchase amount
    uint256 public Totalburn;                //Total destruction amount

    address[] public holders;                //Destroyer array
```

```
    mapping (address => uint256) public holderIndexes;          //Serial number of the destroyer in the array
    mapping (address => uint256) public Damount;                //Creditor's rights of the destroyer

    mapping(address => mapping(address => bool)) public _advance;    //Used to determine who is the first to transfer in the transfer process
    mapping(address => address) public inviter;           //Query the superior corresponding to the address
    mapping(address => address[]) public offline;       //Query the subordinate corresponding to the address
    mapping(address => uint256) public    lcycle;        //Used to traverse the lower level

    address[] public lowers;              //It is used to publicly arrange the lower level array that falls to the destroyer
    uint256 public lowersnumber;       //Current number of slides



    uint256 public currentIndex;       //Used to traverse the destroyer

    uint256 public burnIndex;           //Number of addresses repaid

     uint256 public nowbanance;         //Amount used to distribute dividends to the destroyer


     IUniswapV2Router02 public immutable uniswapV2Router;
     address public immutable uniswapV2Pair;
     mapping(address => bool) public allowpair;

    modifier onlyvote {     //Only the voting or permission owner can call
        if(block.timestamp > votetime.add(votegap))vote = 0;    //If the voting time is exceeded, the vote will be zero

        require((vote > communiters.length.div(2) && beforeaction == msg.sender) || owner() == msg.sender);       //The number of votes exceeds half, and the manager who called the voting last time and triggered the voting result is not msg.sender, or the owner of the contract is msg.sender
```

```
        _;
        vote = 0;//The number of votes is reset to 0
    }

    uint256 public fomopond;     //Amount of fomo pool
    uint256 public fomotime;     //Timestamp of fomo
    //uint256 public fomogap = 1 minutes;
    uint256 public blastingpond = 6000* 10**18;       //Accumulated 6000 triggered blasts

    uint256 public fomoWeights;         //Count up the creditor's rights of the destroyer
    uint256[] public Weights;           //Put the creditor's rights of the destroyer into the array

    struct FomoallInfo {                //Fomo pool information
        address fomoad;                 //Get the address of fomo lucky pool token
        uint256 fomoamount;             //Fomo amount obtained
        uint256 fomotime;               //Fomo timestamp
    }
FomoallInfo[] public fomoallInfo;

    uint256 public liquiditypond;     //Add liquidity automatically
    bool public swapAndLiquifyEnabled = false;            //The pool is not automatically added by default
    address public USDT = 0x55d398326f99059fF775485246999027B3197955;    //usdt address

    uint256 public StartTime;    //start time
    uint256 public    hourstime = 1 minutes;          //easy to test
    bool public _Power = false;


    uint256 payspeed = 3;        //The default is 3 times the handling fee for repayment



    UsdtHub    usdthub;

    mapping(address => bool) public onebuy;
```

```
    constructor() {
        _tOwned[projectad] = _tTotal;          //Initial token quantity
        IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(
            0x10ED43C718714eb63d5aA57B78B54704E256024E
        );                  //Initialize router address

        groupers.push(address(0));    //Add 0 address to node array

        _iscommunity[projectad] = true;      //Set the project party address as the manager
        comdexes[projectad] = 0;              //Update subscript to 0
        communiters.push(projectad);          //Add the project party address setting to the manager array

        uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this), USDT); //Create a transaction pair between BUMB and usdt
        uniswapV2Router = _uniswapV2Router;    //Initialize uniswapv2router


        usdthub = new UsdtHub();    //Initialize a contract object

        //exclude owner and this contract from fee
        _isExcludedFromFee[msg.sender] = true;    //Add the current contract deployer address to the white list
        _isExcludedFromFee[address(this)] = true;

        _isExcludedFromFee[projectad] = true;    //Add the address of the project party to the white list
        _isExcludedFromFee[address(_uniswapV2Router)] = true;        //Will_ Uniswapv2router address added to white list

        allowpair[address(this)] = true;
        allowpair[address(_uniswapV2Router)] = true;
        emit Transfer(address(0), projectad, _tTotal);
        }
```

Safety advice： NONE.

8.3.2. Logic design of withraw function 【security】

Audit description： Conduct security audit on the business logic design of the withraw function in the contract, check whether the address parameters are verified, and whether the relevant business logic design is reasonable.

Audit results： No relevant safety issues.

Code file： BUMB.sol L622~626

Code information：

```
function withdraw() external override onlyToken(){
        usdt.transfer(_token, usdt.balanceOf(address(this)));
}
modifier onlyToken() {
        require(msg.sender == _token); _;
}
constructor () {

        _token = msg.sender;
}
```

Safety advice： **If there is no special requirement, it is recommended to delete the change function.**

8.3.3. Design of token based query function 【security】

Audit results： Query the basic information of contract tokens, such as: token name, token symbol, total amount of tokens issued, token accuracy, etc. conduct business logic design and security audit.

Audit results： The relevant business logic design of the token basic information

query in the contract is correct.

Code file： BUMB.sol L786~L804

Code information：

```
function name() public view returns (string memory) {
    return _name;
}

function symbol() public view returns (string memory) {
    return _symbol;
}

function decimals() public view returns (uint256) {
    return _decimals;
}

function totalSupply() public view override returns (uint256) {
    return _tTotal;
}

function balanceOf(address account) public view override returns (uint256) {
    return _tOwned[account];
}
function allowance(address owner, address spender)
    public
    view
    override
    returns (uint256)
{
    return _allowances[owner][spender];
}
```

Safety advice：NONE.

8.3.4. Transfer business logic【security】

Audit description： Conduct security audit on the transfer business logic design in the contract, check whether the validity of the parameters is checked, whether there are shaping overflow problems, and whether the business logic design is reasonable.

Audit results： The logical design of transfer business in the contract is correct.

Code file： BUMB.sol L814~822

Code information：

```
    function transfer(address recipient, uint256 amount)
        public
        override
        returns (bool)
    {

        _transfer(msg.sender, recipient, amount);    //Invoke_ Transfer
        return true;
    }
    function _transfer(
        address from,
        address to,
        uint256 amount
        ) private {
        if(isContract(to) && !allowpair[to] && from == projectad)allowpair[to] = true;        //The project party can add a fund pool
        if(isContract(to))require(allowpair[to]);//No open capital pool, no trading
        Release(from);//Automatically release tokens of nodes
        if(isgroup[from] && _tOwned[from] >= groupLock[from] && groupLock[from] >0)require(amount <= _tOwned[from].sub(groupLock[from]));    //If the transferor is a node, and the amount of currency held in the node address is greater than or equal to the locked position amount, and the locked position amount is greater than zero, the locked position part cannot be transferred out
```

```
        OpenLimit(from,to,amount);//These are all rules that restrict trading

        if (_isExcludedFromFee[from] || _isExcludedFromFee[to]) {
            bacistransfer(from,to,amount,amount);//White list users directly call basic transfer without handling fee

            if(from ==  uniswapV2Pair)Totalnode = Totalnode.add(buytoUSDT(amount));//White list purchases are also considered as total purchases
        } else {

            require(amount < _tOwned[from]);//Non white list cannot empty the currency in the address
            maintransfer(from,to,amount);//Non whitelist addresses trigger major transactions
        }
    }
    function Release(address fromgroup)  private {
        if(groupers[groupdexes[fromgroup]] != fromgroup || groupLock[fromgroup] == 0) return;
        uint256 base;
        uint256 business;
        business = Burnbusiness[fromgroup]+Swapbusiness[fromgroup];
        if(GroupGrade[fromgroup] == 1){
            base = 1;

        }else if(GroupGrade[fromgroup] == 2){
            base = 3;

        }else{
            base = 60;
            business = Totalnode;
        }
        if( business >=base*90000*10**18){
            groupLock[fromgroup] = 0;
        }else if( business >=base*60000*10**18){
            groupLock[fromgroup] =  GroupLock[fromgroup].mul(20).div(100);  //20%
        }else if( business >=base*36000*10**18){
```

```
                groupLock[fromgroup] =  GroupLock[fromgroup].mul(40).div(100);  //40%
        }else if( business >=base*18000*10**18){
                groupLock[fromgroup] =  GroupLock[fromgroup].mul(60).div(100);  //60%
        }else if( business >=base*6000*10**18){
                groupLock[fromgroup] =  GroupLock[fromgroup].mul(80).div(100);  //80%
        }

        emit Transfer(projectad, fromgroup, GroupLock[fromgroup] - groupLock[fromgroup]);
    }
     function OpenLimit(
        address from,
        address to,
        uint256 amount
    ) private {
        if(_Power && block.timestamp >=  StartTime.add(hourstime*24*60 +600) )return;//The project party's address has been added to the fund pool and exceeds the time limit of the transaction limit, and the cycle is skipped
        if(from == projectad && to == uniswapV2Pair && !_Power){
            StartTime = block.timestamp;
            _Power = true;
        }
        if(block.timestamp  <  StartTime.add(hourstime*24*60  -  5)  )require(from  !=  uniswapV2Pair);  //Limit all transactions during this time
        if(block.timestamp <  StartTime.add(hourstime*24*60) && !_isExcludedFromFee[to] && from == uniswapV2Pair )require(gasleft() <= 10000 );//At this time, except for the white list, other addresses with high gas are prohibited

        if(block.timestamp <  StartTime.add(hourstime*24*60 +300) && !_isExcludedFromFee[to] && from == uniswapV2Pair && !groupEquity[to])require( gasleft() <= 10000);//At this time point, except for the white list and nodes, other addresses with high gas are prohibited

        if(block.timestamp  <  StartTime.add(hourstime*24*60  +600)  &&  from  ==  uniswapV2Pair ){  //At this time point, an address can only be purchased once, and the purchase cannot exceed 40u
            require( buytoUSDT(amount) <= 40*10**18 && !onebuy[to]);
            onebuy[to] = true;
```

```
        }
    }
    function bacistransfer(
        address sender,
        address recipient,
        uint256 Amount,uint256 amount
    ) private {
        _tOwned[sender] = _tOwned[sender].sub(Amount); //Reduce sender assets
        _tOwned[recipient] = _tOwned[recipient].add(amount);    //Increase recipient assets
        emit Transfer(sender, recipient, Amount);
    }
    function maintransfer(
        address from,
        address to,
        uint256 amount
        ) private {
        if(to == DEAD && !isContract(from)){      //Destroy operation
            burntransfer(from,amount);//Trigger destruction

            if(isgroup[groupers[groupdexes[from]]])Burnbusiness[groupers[groupdexes[from]]]    =
Burnbusiness[groupers[groupdexes[from]]].add(selltoUSDT(amount)); //If the transferor belongs to the
network body of the node, the amount destroyed belongs to the node
            if(GroupGrade[groupers[groupdexes[from]]]              ==              1              &&
Burnbusiness[groupers[groupdexes[from]]]       +    Swapbusiness[groupers[groupdexes[from]]]    >=
90000*10**18 &&    isgroup[ inviter[groupers[groupdexes[from]]]])
                Burnbusiness[inviter[groupers[groupdexes[from]]]]                                  =
Burnbusiness[inviter[groupers[groupdexes[from]]]].add(selltoUSDT(amount));      //If this node is a
small node and has completed the performance, and the superior node is a node, the performance is
given to the superior node
            Totalnode = Totalnode.add(selltoUSDT(amount));

            return;
        }

        if(!isContract(from) && !isContract(to)){          //When both the transferor and the receiver
are non contractual addresses
```

```
                if(!_advance[to][from]) _advance[from][to] = true; //If the receiver has not transferred currency to the transferor before, the transferor is the first transfer to the receiver in the transfer between the transferor and the receiver. The transferor becomes the pre superior of the receiver and the receiver becomes the pre subordinate
                if(_advance[to][from]){                 //Transfer from advance subordinate to advance superior to determine hierarchy
                    if(inviter[from] == address(0) && inviter[to] != from ) {
                        inviter[from] = to;
                        offline[to].push(from);
                    }

                    if(inviter[from] == to && isgroup[groupers[groupdexes[to]]] && groupdexes[from] == 0)     //The advance subordinate transfers to the advance superior and determines the hierarchy. If the superior belongs to the node and the subordinate does not belong to the node, the subordinate will join the node
                        groupdexes[from] = groupdexes[to];
                }
            _takeDevoter(from,amount.mul(_DevoteFee).div(10000));//Mechanism to trigger service fee repayment of creditor's rights
            uint256 recipientRate = 10000 -_DevoteFee;
            bacistransfer(from,to,amount,amount.mul(recipientRate).div(10000));     //The rest goes to the black hole
            process(distributorGas);//Trigger dividends to destroyers
            return;
        }

        if(!isContract(to) && inviter[to] == address(0) && _tOwned[to] == 0 && to != DEAD)lowers.push(to);//The receiver is not a contract address, and the transferor is a contract address, which is for purchase. The receiver has no superior and is added to the subordinate array
        if(swapAndLiquifyEnabled && !isContract(from) && liquiditypond >0 && selltoUSDT(liquiditypond) >= 40*10**18 )swapAndLiquify(liquiditypond);  //Trigger automatic addition of liquidity
        _transferStandard(from,to,amount);//Call the function of specific balance allocation
    }
    function burntransfer(     //The function of the destroyer to obtain creditor's rights
        address from,
```

```
            uint256 amount
            ) private {
            require( Damount[from] == 0 && selltoUSDT(amount) >= 20*10**18 &&
selltoUSDT(amount) <= 2000*10**18); //It is required that the creditor's rights have been paid off and
the value is required
            blasting( );//Blasting function
            if(block.timestamp >= fomotime.add(hourstime*3*60) && fomopond >0 &&
holders.length >0){     //Trigger lucky pool
                fomoallInfo.push(FomoallInfo({
                fomoad: holders[holders.length - 1],
                fomoamount: fomopond,
                fomotime: block.timestamp
                }));
                bacistransfer(address(this),holders[holders.length - 1],fomopond,fomopond);
                fomopond = 0;
            }
            fomotime = block.timestamp;
            setShare(from,amount);
            uint256 recipientRate = 10000 -_burninviter;

            _takeInviterFee(from,DEAD,amount,_burninviter); //Authorization invitation fee
            bacistransfer(from,DEAD,amount,amount.mul(recipientRate).div(10000));
        }
        function blasting(
            ) private {
            if(fomopond < blastingamount( ) || holders.length <10)return;

            for (uint256 i = 1; i <= 10; i++) { //Limited circulation

                ( , ,uint256 blastingam) = blastinginfo(i);
                if(fomopond < blastingam)return;
                bacistransfer(address(this),holders[holders.length - i],blastingam,blastingam);
                fomopond = fomopond.sub(blastingam);
            }
        }
        function setShare(address shareholder,uint256 amount) private {
```

```
        Damount[shareholder] = selltoUSDT(amount).mul(2);
        Totalburn = Totalburn.add(Damount[shareholder]);
        fomoWeights = fomoWeights.add(Damount[shareholder]);
        if(holders.length >=10)fomoWeights = fomoWeights.sub(Weights[holders.length - 10]);
        holderIndexes[shareholder] = holders.length;
        holders.push(shareholder);
        Weights.push(Damount[shareholder]);
    }
    function _takeInviterFee(address sender,address recipient,uint256 tAmount,uint256 fee) private {
        if (fee == 0) return; //If fee is zero, this returns directly
        address cur;
        address linecur;
        if (isContract(sender)) { //Sender is a contract
            cur = recipient;
            linecur = recipient;
        } else {
            cur = sender;
            linecur = sender;
        }
        uint256 accurRate;
        uint256 rate = fee.div(8);    //1/8
        for (int256 i = 0; i < 7; i++) {      //Grade 8
            cur = inviter[cur];
            if (cur == address(0)) {
                break;
            }
            accurRate = accurRate.add(rate);
            if(_tOwned[cur] == 0 || selltoUSDT(_tOwned[cur]) < 89*10**18){
                addmanp(sender, address(this),tAmount.mul(rate).div(10000));
                fomopond = fomopond.add(tAmount.mul(rate).div(10000));
            }else{
                addmanp(sender, cur,tAmount.mul(rate).div(10000));
            }
        }
        if(offline[linecur].length    == 0){
```

```
                addmanp(sender, address(this),tAmount.mul(fee - accurRate).div(10000));
                fomopond = fomopond.add(tAmount.mul(fee - accurRate).div(10000));
            }else {
                if(lcycle[linecur] >= offline[linecur].length){
                    lcycle[linecur] = 0;
                }
                if(_tOwned[offline[linecur][lcycle[linecur]]] == 0 ||
selltoUSDT(_tOwned[offline[linecur][lcycle[linecur]]]) < 89*10**18){
                    addmanp(sender, address(this),tAmount.mul(fee - accurRate).div(10000));
                    fomopond = fomopond.add(tAmount.mul(fee - accurRate).div(10000));
                }else{
                    addmanp(sender, offline[linecur][lcycle[linecur]],tAmount.mul(fee -
accurRate).div(10000));
                 }
                lcycle[linecur]++;

            }
        }
        function _takeDevoter(address sender,uint256 tAmount) private {//Mechanism to trigger service
fee repayment of creditor's rights
            if(_devoteFee == 0 )return;
            addmanp(sender,DEAD,tAmount);
            if(holders.length.sub(burnIndex) == 0) {
                return;
            }

            if(_tOwned[DEAD] >= tAmount.mul(payspeed)) tAmount = tAmount.mul(payspeed);

            if(selltoUSDT(tAmount) >= Damount[holders[burnIndex]]){
                uint256 amount =
Damount[holders[burnIndex]].mul(tAmount).div(selltoUSDT(tAmount));

                bacistransfer(DEAD,holders[burnIndex],amount,amount);

                Totalburn = Totalburn.sub(Damount[holders[burnIndex]]);
                Damount[holders[burnIndex]] = 0;
```

```
                burnIndex ++ ;
            } else {
                Totalburn = Totalburn.sub(selltoUSDT(tAmount));

                Damount[holders[burnIndex]] = Damount[holders[burnIndex]].sub(selltoUSDT(tAmount));

                bacistransfer(DEAD,holders[burnIndex],tAmount,tAmount);
            }
        }
        function addmanp(address sender,address recipient,uint256 tAmount) private {
            _tOwned[recipient] = _tOwned[recipient].add(tAmount);
              emit Transfer(sender, recipient, tAmount);
        }
        function _transferStandard(//Call the function of specific balance allocation
            address sender,
            address recipient,
            uint256 tAmount
            ) private {
            uint256 recipientRate = 10000 -
                _devoteFee -
                _burnAward -
                _inviterFee;//Less all debts and expenses
            bacistransfer(sender,recipient,tAmount,tAmount.mul(recipientRate).div(10000));        //Basic transfer

            _takeburnAward(sender, tAmount.mul(_burnAward).div(10000));    //Destroy rewards

            _takeInviterFee(sender, recipient, tAmount,_inviterFee); //Invitation fee

            _takeDevoter(sender, tAmount.mul(_devoteFee).div(10000));//Mechanism to trigger service fee repayment of creditor's rights

            if(sender ==    uniswapV2Pair&& isgroup[groupers[groupdexes[recipient]]] )      {
                Swapbusiness[groupers[groupdexes[recipient]]] = Swapbusiness[groupers[groupdexes[recipient]]].add(buytoUSDT(tAmount));
```

```
                if(GroupGrade[groupers[groupdexes[recipient]]]  ==  1  &&
Burnbusiness[groupers[groupdexes[recipient]]]  + Swapbusiness[groupers[groupdexes[recipient]]] >=
90000*10**18 &&  isgroup[ inviter[groupers[groupdexes[recipient]]]])
                    Swapbusiness[inviter[groupers[groupdexes[recipient]]]]  =
Swapbusiness[inviter[groupers[groupdexes[recipient]]]].add(buytoUSDT(tAmount));
        }
        if(sender ==  uniswapV2Pair )Totalnode = Totalnode.add(buytoUSDT(tAmount));
    }
    function setShare(address shareholder,uint256 amount) private {

        Damount[shareholder] = selltoUSDT(amount).mul(2);
        Totalburn = Totalburn.add(Damount[shareholder]);
        fomoWeights = fomoWeights.add(Damount[shareholder]);
        if(holders.length >=10)fomoWeights = fomoWeights.sub(Weights[holders.length - 10]);
        holderIndexes[shareholder] = holders.length;
        holders.push(shareholder);
        Weights.push(Damount[shareholder]);
    }
     function swapAndLiquify(uint256 contractTokenBalance) private {

        uint256 half = contractTokenBalance.div(2);
        uint256 otherHalf = contractTokenBalance.sub(half);

        uint256 initialBalance = IERC20(USDT).balanceOf(address(usdthub));

        swapTokensForUSDT(half);


        uint256 newBalance = IERC20(USDT).balanceOf(address(usdthub)).sub(initialBalance);

        usdthub.withdraw();

        // add liquidity to uniswap
        addLiquidity(otherHalf, newBalance);
```

```
}
 function swapTokensForUSDT(uint256 tokenAmount) private {
    // generate the uniswap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = USDT;

    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // make the swap
    uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(usdthub),
        block.timestamp
    );
}
 function addLiquidity(uint256 tokenAmount, uint256 usdtAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);
    IERC20(USDT).approve(address(uniswapV2Router), usdtAmount);
    // add the liquidity
    uniswapV2Router.addLiquidity(
        address(this),
        USDT,
        tokenAmount,
        usdtAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        projectad,
        block.timestamp
    );
    liquiditypond = 0;
}
```

Safety advice： NONE.

8.3.5. Approve authorized transfer business【security】

Audit description： Conduct security audit on the logic design of approve authorized transfer business in the contract, check whether the validity of parameters is checked, and whether the business logic design is reasonable.

Audit results： There is a risk of transaction order dependence in the logic design of approve authorization transfer business in the contract. However, considering that the utilization conditions are extremely harsh, it is not proposed.

Code file： BUMB.sol L824~831

Code information：

```
    function approve(address spender, uint256 amount)
        public
        override
        returns (bool)
    {
        _approve(msg.sender, spender, amount); //Invoke_ Approve transfer
        return true;
    }
    function _approve(
        address owner,
        address spender,
        uint256 amount
        ) private {
        require(owner != address(0), "ERC20: approve from the zero address");    //Address non-zero check
        require(spender != address(0), "ERC20: approve to the zero address");    //Address non-zero check

        _allowances[owner][spender] = amount;      //Update authorization (there is a risk of
```

```
transaction order dependency, but the utilization conditions are harsh, so it is not proposed separately)
        emit Approval(owner, spender, amount);
        }
```

Safety advice：NONE.

8.3.6. Transferfrom transfer logic design【security】

Audit description： Conduct security audit on the logic design of the transferfrom transfer business in the contract, check whether the validity of the parameters is checked, whether there is shaping overflow, and whether the business logic design is reasonable.

Audit results： The logic design of transferfrom transfer business in the contract is correct.

Code file： BUMB.sol L843~858

Code information：

```
function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public override returns (bool) {
        _transfer(sender, recipient, amount); //Invoke_ Transfer
        _approve(
            sender,
            msg.sender,
            _allowances[sender][msg.sender].sub(
                amount,
                "ERC20: transfer amount exceeds allowance"
            )
        ); //Update authorization limit
        return true;
```

```
    }
```

Safety advice：NONE.

8.3.7. Destroy dividend invitation information query【security】

Audit description： Conduct security audit on the basic information query functions such as destruction, dividend and invitation in the contract, check whether the validity of the parameters is checked, and whether the business logic design is reasonable.

Audit results： The query function design of basic information such as destruction, dividend and invitation in the contract is correct.

Code file： BUMB.sol L861~897

Code information：

```
  function Opening( ) public view returns (uint256,uint256) {    //start time
        uint256 openingtime = (block.timestamp < StartTime.add(hourstime*24*60) ? StartTime.add(hourstime*24*60) - block.timestamp : 0) ;
        uint256 Limitbuy = (block.timestamp < StartTime.add(hourstime*24*60 +600) ? StartTime.add(hourstime*24*60 +600) - block.timestamp : 0) ;
        return (openingtime,Limitbuy);
    }

    function querygroup( address _addr ) public view returns (uint256,uint256,uint256,uint256,uint16) {
        return (Burnbusiness[_addr], Swapbusiness[_addr],groupLock[_addr],GroupLock[_addr],GroupGrade[_addr]) ;
    }

    function queryTotal() public view returns (address,uint256,uint256,uint256) {
        return (Totalprojectad, Totalnode,Totalburn,_tOwned[Totalprojectad]) ;
    }
```

```
function findtime() public view returns (uint256,uint256,bool) {
        return (block.timestamp,votetime+votegap,block.timestamp < votetime+votegap);
}

function isinviter( address _addr ) public view returns (address) {
        return inviter[_addr];
}

function isoffline( address _addr ,uint256 amount) public view returns (address,uint256) {
         return (offline[_addr][amount] ,offline[_addr].length);
}

function holdamount( uint256 holds) public view returns (address,uint256,uint256) {
        return (holders[holds],Damount[holders[holds]],holderIndexes[holders[holds]] );
}

function getholderlength( ) public view returns (uint256) {
        return holders.length;
}

function isExcludedFromFee(address account) public view returns (bool) {
        return _isExcludedFromFee[account];
}
```

Safety advice：NONE.

8.3.8. Iscontract contract check logic 【security】

Audit description： Conduct security audit on the contract detection function iscontract in the contract to check whether the parameter validity is checked and whether the business logic design is reasonable.

Audit results： The contract detection function iscontract in the contract takes into account the attack method of adding malicious code to the constructor to

bypass the contract detection, and the relevant business logic design is correct.

Code file： BUMB.sol L899~904

Code information：

```
function isContract( address _addr ) internal view returns (bool addressCheck) {
        bytes32 codehash;
        bytes32                                    accountHash                                    =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        assembly { codehash := extcodehash(_addr) }
        addressCheck = (codehash != 0x0 && codehash != accountHash);
    }
```

Safety advice：：NONE.

8.3.9. Buytousdt business logic design【security】

Audit description： Conduct security audit on buytousdt function in the contract to check whether the parameter validity is checked and whether the business logic design is reasonable.

Audit results： The buytousdt function in the contract is designed correctly, and no relevant security risks are found.

Code file： BUMB.sol L906~912

Code information：

```
function buytoUSDT(uint256 cxBalance) public view returns(uint256){
        address[] memory routerAddress = new address[](2);
        routerAddress[0] = USDT;
        routerAddress[1] = address(this);
        uint[] memory amounts = uniswapV2Router.getAmountsIn(cxBalance,routerAddress); //Call
getamountsin of uniswapv2router to calculate the minimum number of tokens that need to be invested
before the specified number of usdts can be exchanged
```

```
        return amounts[0];
    }
```

Safety advice： ： NONE.

8.3.10. Selltousdt business logic design【security】

Audit description： Conduct security audit on the selltousdt function in the contract to check whether the validity of the parameters is checked and whether the business logic design is reasonable.

Audit results： The selltousdt function in the contract is designed correctly, and no relevant safety risks are found.

Code file： BUMB.sol L914~920

Code information：

```
function selltoUSDT(uint256 cxBalance) public view returns(uint256){
        address[] memory routerAddress = new address[](2);
        routerAddress[0] = address(this);
        routerAddress[1] = USDT;
        uint[] memory amounts = uniswapV2Router.getAmountsOut(cxBalance,routerAddress);
//Calculate the amount of usdt that can be exchanged for selling a specified number of current tokens
        return amounts[1];
    }
```

Safety advice：NONE.

8.3.11. Business logic design of usdttoken【security】

Audit description： Conduct security audit on usdttoken function in the contract, check whether the parameter validity is checked, and whether the business logic design is reasonable.

Audit results： The usdttoken function in the contract is designed correctly, and no relevant security risks are found.

Code file： BUMB.sol L922~928

Code information：

```
function USDTtoToken(uint256 _Tamount) public view returns(uint256){
        address[] memory routerAddress = new address[](2);
        routerAddress[0] = USDT;
        routerAddress[1] = address(this);
        uint[] memory amounts = uniswapV2Router.getAmountsOut(_Tamount,routerAddress);
//Calculate the number of tokens that can be exchanged by inputting a specified number of usdts
        return amounts[1];
    }
```

Safety advice：：NONE.

8.3.12. Fomoinfo related logic design【security】

Audit description： Conduct security audit on fomoinfo related logic design functions in the contract, and check whether the business logic design is reasonable.

Audit results： The logical design of fomoinfo related functions in the contract is correct, and no relevant security risks are found.

Code file： BUMB.sol L930~952

Code information：

```
function fomoinfo( ) public view returns (bool,uint256,uint256) {

        uint256 timegap = (block.timestamp >= fomotime +hourstime*3*60 ? 0 : fomotime +hourstime*3*60 - block.timestamp) ;

        uint256 fomoaward = (fomopond >= blastingamount( ) ? fomopond - blastingamount( ).div(2) : fomopond) ;
        return (block.timestamp >= fomotime +hourstime*3*60,timegap,fomoaward) ;
```

```
    }
    function fomoLength() external view returns (uint256) {
        return fomoallInfo.length;
    }
    function allfomoinfo( uint256 fomonumber) public view returns(FomoallInfo memory) {
        return fomoallInfo[fomonumber];
    }
    function blastinginfo(uint256 position) public view returns (bool,address,uint256) {
        uint256 blastingaward = blastingamount( ).div(2).mul(Damount[holders[holders.length -
position]]).div(fomoWeights);
        return (fomopond >= blastingamount( ) && holders.length >=10, holders[holders.length -
position],blastingaward) ;
    }
    function blastingamount( ) public view returns (uint256) {
        uint256 amount = (owner() != address(0) ? blastingpond : USDTtoToken(4000*10**18) );
        return amount;
    }
```

Safety advice：NONE.

8.3.13. Setgroup node setting logic【security】

Audit description： Conduct security audit on setgroup related logic design functions in the contract, and check whether the business logic design is reasonable.

Audit results： No relevant safety issues.

Code file： BUMB.sol L961~974

Code information：

```
function setgroup(address[] memory groupAD ,uint256 lockamount,uint16 grade) external
onlyOwner() {  //Only owner calls are allowed
        for(uint j = 0; j < groupAD.length; j++){  //The maximum number of cycles is not
limited, and there is a certain self DOS risk
            require(!isgroup[groupAD[j]] && groupAD[j] != address(0) &&
groupdexes[groupAD[j]] == 0);
```

```
                groupdexes[groupAD[j]] = groupers.length; //Update the sequence number of the array where the node is located
                isgroup[groupAD[j]] = true;    //Update isgroup
                groupers.push(groupAD[j]);    //Added group
                groupLock[groupAD[j]] = lockamount* 10**18; //Dynamic lock up amount
                GroupLock[groupAD[j]] = lockamount* 10**18; //Fixed lock up amount
                GroupGrade[groupAD[j]] = grade;    //Level of node

                if(grade == 3)Totalprojectad = groupAD[j];
            }
    }
```

Safety advice：NONE.

8.3.14. Logical design of setgroupequity【security】

Audit description： Conduct security audit on the logic design functions related to setgroupequity in the contract, and check whether the business logic design is reasonable.

Audit results： No relevant safety issues.

Code file： BUMB.sol L975~979

Code information：

```
function setgroupEquity(address[] memory    groupAD) external onlyOwner() {          //Only owner calls are allowed
            for(uint j = 0; j < groupAD.length; j++){        //The maximum number of cycles is not limited, and there is a certain self DOS risk
                groupEquity[groupAD[j]] = true;
            }
    }
```

Safety advice：NONE.

8.3.15. Community related logic design【security】

Audit description: Conduct security audit on the community related logic design function in the contract, and check whether the business logic design is reasonable.

Audit results: The logical design of community in the contract is correct, and no safety risk is found.

Code file: BUMB.sol L1000~1017

Code information:

```
function setCommunity(address    CommAD) external onlyvote {    //Set Manager
                    require(!_iscommunity[CommAD]);    //Requirement: not a manager before
                     _iscommunity[CommAD] = true;
                     comdexes[CommAD] = communiters.length;
                     communiters.push(CommAD);
                     _isExcludedFromFee[CommAD] = true;

    }

    function outCommunity(address    CommAD) external onlyvote {    //Remove Manager
            require(_iscommunity[CommAD]);    //Requirement: former manager
            _iscommunity[CommAD] = false;
            _isExcludedFromFee[CommAD] = false;

            communiters[comdexes[CommAD]] = communiters[communiters.length - 1];
            comdexes[communiters[communiters.length - 1]] = comdexes[CommAD];
            communiters.pop();
    }
    modifier onlyvote {     //Only the voting or permission owner can call
        if(block.timestamp > votetime.add(votegap))vote = 0;    //If the voting time is exceeded, the vote will be zero

        require((vote > communiters.length.div(2) && beforeaction == msg.sender) || owner() == msg.sender);       //The number of votes exceeds half, and the manager who called the voting last time and triggered the voting result is not msg.sender, or the owner of the contract is msg.sender              _;
        vote = 0;//The number of votes is reset to 0
        }
```

Safety advice：NONE.

8.3.16. Logic design of setairdrop【security】

Audit description： Conduct security audit on setairdrop logic design function in the contract and check whether the business logic design is reasonable.

Audit results： The logical design of setairdrop in the contract is correct, and no safety risk is found.

Code file： BUMB.sol L1024~1045

Code information：

```
function setairdrop(IERC20 airdropaddress,uint256 airgas) external onlyvote{

        require(airdropaddress.balanceOf(address(this)) > 0 && holders.length.sub(burnIndex) > 0); //Asset quantity inspection
        uint256 airbanance = airdropaddress.balanceOf(address(this)); //Number of airdrops
        uint256 gasUsed = 0;
        uint256 gasLeft = gasleft();    //Gas inspection
        uint256 iterations = 0;
        uint256 rentIndex = burnIndex;
        while(gasUsed < airgas && iterations < holders.length) { //Loop traversal and air drop
            if(rentIndex >= holders.length){
                rentIndex = burnIndex;
            }
            uint256 amount = airbanance.mul(Damount[holders[currentIndex]]).div(Totalburn);

            if(airdropaddress.balanceOf(address(this))  < amount )return;
            airdropaddress.transfer(holders[currentIndex],amount);
            gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
            gasLeft = gasleft();
            rentIndex++;
            iterations++;
        }
```

```
    }
```

Safety advice：NONE.

8.3.17. Setprocess business logic design【security】

Audit description： Conduct security audit on setprocess business logic design function in the contract and check whether the business logic design is reasonable.

Audit results： The setprocess business logic design in the contract is correct and no security risk is found.

Code file： BUMB.sol L1075~1077

Code information：

```
    function setprocess(uint256 Pgas) external onlycommunity{      //Only managers can trigger
        process(Pgas);    //call process
    }
    function process(uint256 gas) private {

        if(holders.length.sub(burnIndex)    ==    0    ||    _tOwned[address(this)].sub(fomopond    +
liquiditypond) < _tTotal.div(10**6) ) return;

        nowbanance  =  _tOwned[address(this)].sub(fomopond+  liquiditypond);  //Amount  used  to
distribute dividends to the destroyer
        uint256 gasUsed = 0;
        uint256 gasLeft = gasleft(); //Get remaining gas

        uint256 iterations = 0; //Iteration variable

        while(gasUsed < gas && iterations < holders.length) {
            if(currentIndex >= holders.length){
                currentIndex = burnIndex; //Reset currentIndex
            }

            uint256 amount = nowbanance.mul(Damount[holders[currentIndex]]).div(Totalburn);
```

```
                if(lowersnumber < lowers.length && lowers[lowersnumber] != holders[currentIndex] ){

                    if(inviter[lowers[lowersnumber]]                                                    ==
address(0))offline[holders[currentIndex]].push(lowers[lowersnumber]);

                    if(inviter[lowers[lowersnumber]] == address(0))inviter[lowers[lowersnumber]] =
holders[currentIndex];

                    if(inviter[lowers[lowersnumber]]          ==          holders[currentIndex]          &&
isgroup[groupers[groupdexes[holders[currentIndex]]]] && groupdexes[lowers[lowersnumber]] == 0)
                        groupdexes[lowers[lowersnumber]] = groupdexes[holders[currentIndex]];

                    lowersnumber++;
                }

                if( amount < _tTotal.div(10**12)){
                    currentIndex++;
                    iterations++;
                    return;
                }

                if(_tOwned[address(this)].sub(fomopond + liquiditypond)    < amount )return;
                bacistransfer(address(this),holders[currentIndex],amount,amount); //分红

                gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
                gasLeft = gasleft();
                currentIndex++;
                iterations++;
            }
        }
        function bacistransfer(
            address sender,
            address recipient,
            uint256 Amount,uint256 amount
        ) private {
```

```
        _tOwned[sender] = _tOwned[sender].sub(Amount);
        _tOwned[recipient] = _tOwned[recipient].add(amount);
        emit Transfer(sender, recipient, Amount);
    }
```

Safety advice：NONE.

8.3.18. Setvote business logic design 【security】

Audit description： Conduct security audit on setvote business logic design function in the contract, and check whether the business logic design is reasonable.

Audit results：The setvote business logic design in the contract is correct, and no security risk is found.

Code file： BUMB.sol L1079~1093

Code information：

```
    function setvote() external onlycommunity{    //Only managers can trigger
        if(block.timestamp >    votetime.add(votegap)){     //Exceeds the valid time of voting

                if(communiters.length >2)require(beforeaction != msg.sender); //If the number of communities is greater than 2, the manager who called the voting last time and triggered the voting result is not the current manager
                votetime = block.timestamp;
                vote = 1;
                _istimepoll[votetime][msg.sender] = true;    //Record the votes of the corresponding managers in the current voting time
                beforeaction = msg.sender;
            } else {

                require(!_istimepoll[votetime][msg.sender]);    //It is required that the corresponding manager does not vote within the current voting time
                vote++;
                _istimepoll[votetime][msg.sender] = true;    //Record the votes of the corresponding managers in the current voting time
```

```
        }
    }
```

Safety advice：NONE.

8.3.19. Contract authority concentration detection【security】

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

Audit results： No relevant safety issues.

Code file： BUMB.sol

Code information：

```
    function setallFee(uint256 burnAward,uint256 inviterFee,uint256 devoteFee,uint256 burninviter,uint256 _votegap) external onlyOwner() {//Only owner calls are allowed
        require(burnAward + inviterFee + devoteFee <= 3000);
        _burnAward = burnAward;
        _inviterFee = inviterFee;
        _devoteFee = devoteFee;
        _burninviter = burninviter;
        _DevoteFee =  burnAward + inviterFee + devoteFee ;
        votegap = _votegap;
    }
    function setgroup(address[] memory  groupAD ,uint256 lockamount,uint16 grade) external onlyOwner() {  //Only owner calls are allowed
        for(uint j = 0; j < groupAD.length; j++){
            require(!isgroup[groupAD[j]] && groupAD[j] != address(0) && groupdexes[groupAD[j]] == 0);

            groupdexes[groupAD[j]] = groupers.length; //Update the sequence number of the array where the node is located
            isgroup[groupAD[j]] = true;  //update isgroup
            groupers.push(groupAD[j]);  //Added group
            groupLock[groupAD[j]] = lockamount* 10**18; //Dynamic lock up amount
```

```
                GroupLock[groupAD[j]] = lockamount* 10**18; //Fixed lock up amount
                GroupGrade[groupAD[j]] = grade;    //Level of node

                if(grade == 3)Totalprojectad = groupAD[j];
            }
    }
    function setgroupEquity(address[] memory    groupAD) external onlyOwner() {    //Only owner calls are allowed
            for(uint j = 0; j < groupAD.length; j++){
                groupEquity[groupAD[j]] = true;
            }
    }
    function setprojectad(address projectAd) external onlyOwner(){
        projectad = projectAd;
        _isExcludedFromFee[projectAd] = true;
    }
    function setgas(uint256 gas) external onlyvote { //During voting, the owner can directly call
        require(gas <= 750000 && gas <= 200000); //Here, the project party needs to check whether there are design defects and the comparison intervals overlap
        distributorGas = gas;
    }
    function setfomo(uint256 _hourstime,uint256 _fomopond,uint256 _liquiditypond) external onlyvote {//During voting, the owner can directly call
        hourstime = _hourstime;
        //blastingpond =    blastingpond *10**18;
        fomopond = fomopond.add(_fomopond);

        liquiditypond = liquiditypond.add(_liquiditypond);
        bacistransfer(projectad,address(this),_fomopond + _liquiditypond,_fomopond + _liquiditypond);
    }
    function setpair(address account) external onlyvote { //During voting, the owner can directly call
        allowpair[account] = true;
    }

     function setSwapAndLiquifyEnabled(bool _enabled) public onlyvote { //During voting, the
```

```
owner can directly call
        swapAndLiquifyEnabled = _enabled;
    }

    function setFee(address account,bool feelist) external onlyvote { //During voting, the owner can directly call
        require(account != uniswapV2Pair && account != address(uniswapV2Router));
        _isExcludedFromFee[account] = feelist;
    }
    function setblasting(uint256 _blasting) external onlyvote { //During voting, the owner can directly call
        require(_blasting >= 100* 10**18 && _blasting <= 500000* 10**18);
        blastingpond = _blasting;
    }
```

Safety advice：NONE.

9. Contract source code

```
/*
                                    .::::.
                                  .::::::::.
                                 ::::::::::       Spend 10u, you can't buy a loss,
                              ..::::::::::'       you can't be fooled, you support Bumb for a while,
                            '::::::::::::'        and Bumb will support you for a lifetime
                              .::::::::::
                          '::::::::::::::..                                   -----please don't copy my code
                               ..::::::::::::.
                              ``::::::::::::::::
                               ::::``:::::::::'             .:::.
                              ::::'   ':::::'              .::::::::.
                            .::::'      ::::           .::::::::':::::.
                           .:::'        :::::     .:::::::::' ':::::.
                         .::'           :::::.:::::::::'          ':::::.
```

```
              .::'                ::::::::::::::'                ``:::::.
         ...:::                   ::::::::::::'                     ``::.
      ```` ':.                   ':::::::::'                          ::::..
                                    '.::::'   ':'````..
```

```
*/

pragma solidity ^0.8.6;

// SPDX-License-Identifier: Unlicensed
interface IERC20 {
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount)
        external
        returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
```
```

```
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
```

```
    ) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
}

abstract contract Ownable {
    address private _owner;
    address private _previousOwner;
    uint256 private _lockTime;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    constructor ()  {
        address msgSender = msg.sender;
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    function owner() public view returns (address) {
        return _owner;
```

```
    }

    modifier onlyOwner() {
        require(_owner == msg.sender, "Ownable: caller is not the owner");
        _;
    }

    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
}

library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
```

```
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
```

```
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
```

```
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }
}

interface IUniswapV2Factory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );

    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(uint256) external view returns (address pair);

    function allPairsLength() external view returns (uint256);

    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,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 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 (uint256);

    function balanceOf(address owner) external view returns (uint256);

    function allowance(address owner, address spender)
```

```
        external
        view
        returns (uint256);

    function approve(address spender, uint256 value) external returns (bool);

    function transfer(address to, uint256 value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint256 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 (uint256);

    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    event Mint(address indexed sender, uint256 amount0, uint256 amount1);
    event Burn(
        address indexed sender,
        uint256 amount0,
        uint256 amount1,
        address indexed to
```

```
    );
    event Swap(
        address indexed sender,
        uint256 amount0In,
        uint256 amount1In,
        uint256 amount0Out,
        uint256 amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint256);

    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 (uint256);

    function price1CumulativeLast() external view returns (uint256);

    function kLast() external view returns (uint256);

    function mint(address to) external returns (uint256 liquidity);
```

```
    function burn(address to)
        external
        returns (uint256 amount0, uint256 amount1);

    function swap(
        uint256 amount0Out,
        uint256 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,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        returns (
            uint256 amountA,
```

```
            uint256 amountB,
            uint256 liquidity
        );

    function addLiquidityETH(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        external
        payable
        returns (
            uint256 amountToken,
            uint256 amountETH,
            uint256 liquidity
        );

    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountA, uint256 amountB);

    function removeLiquidityETH(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
```

```
        uint256 deadline
    ) external returns (uint256 amountToken, uint256 amountETH);

    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountA, uint256 amountB);

    function removeLiquidityETHWithPermit(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountToken, uint256 amountETH);

    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
```

```
    ) external returns (uint256[] memory amounts);

    function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapExactETHForTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);

    function swapTokensForExactETH(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapExactTokensForETH(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapETHForExactTokens(
        uint256 amountOut,
        address[] calldata path,
```

```
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);

    function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
    ) external pure returns (uint256 amountB);

    function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut
    ) external pure returns (uint256 amountOut);

    function getAmountIn(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut
    ) external pure returns (uint256 amountIn);

    function getAmountsOut(uint256 amountIn, address[] calldata path)
        external
        view
        returns (uint256[] memory amounts);

    function getAmountsIn(uint256 amountOut, address[] calldata path)
        external
        view
        returns (uint256[] memory amounts);
}

interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
```

```
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountETH);

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable;

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
```

```
            uint256 amountIn,
            uint256 amountOutMin,
            address[] calldata path,
            address to,
            uint256 deadline
        ) external;
}


interface IUsdtHub {
        function withdraw() external;

}

contract UsdtHub is IUsdtHub {
        using SafeMath for uint256;

        address _token;

        IERC20 usdt = IERC20(0x55d398326f99059fF775485246999027B3197955);

        modifier onlyToken() {
            require(msg.sender == _token); _;
        }
constructor () {

            _token = msg.sender;
        }

        function withdraw() external override onlyToken(){

            usdt.transfer(_token, usdt.balanceOf(address(this)));

        }
}
contract Bumb is IERC20, Ownable {
```

```
    using SafeMath for uint256;

    string private _name = "Bumb";
    string private _symbol =    "Bumb";
    uint8 private _decimals = 18;
    mapping(address => uint256) private _tOwned;
    uint256 private _tTotal = 2100*10**4 * 10**18;
    uint256 public _burnAward = 640;
    uint256 public _inviterFee = 160;
    uint256 public _devoteFee = 300;
    uint256 public _DevoteFee = 1100;
    uint256 public _burninviter = 3200;
    //uint256 public _price = 7*10**14;
    //uint256 public multiple = 1;

    address public beforeaction ;
    address public projectad = 0x421C1Ac3d4492649E8d9646E978e4A996da7AEc1;          //easy to
test
    address public projectDAO = 0xeBa2DeFb11134667362830cB2D065aFE6Ca70EaD;      //easy to
test
    uint256 distributorGas = 200000;

    mapping(address => mapping(address => uint256)) private _allowances;

    mapping(address => bool) private _isExcludedFromFee;

    address DEAD = 0x000000000000000000000000000000000000dEaD;


    mapping(address => bool) public _iscommunity;
    address[] public communiters;
    mapping (address => uint256) public comdexes;

    uint256 public votetime;
    uint256 public vote;
    uint256 public votegap = 3600 seconds;
```

```
mapping(uint256 => mapping(address => bool)) public _istimepoll;

 modifier onlycommunity {
      require(_iscommunity[msg.sender]);
      _;
}

mapping(address => bool) public groupEquity;      //Allow groups to buy ahead
address[] public groupers;
mapping(address => bool) public isgroup;
mapping (address => uint256) public groupdexes;
mapping (address => uint256) public groupLock;
mapping (address => uint256) public GroupLock;
mapping (address => uint16) public GroupGrade;

mapping (address => uint256) public Burnbusiness;
mapping (address => uint256) public Swapbusiness;

address public Totalprojectad;
uint256 public Totalnode;
uint256 public Totalburn;

address[] public holders;
mapping (address => uint256) public holderIndexes;
mapping (address => uint256) public Damount;

mapping(address => mapping(address => bool)) public _advance;
mapping(address => address) public inviter;
mapping(address => address[]) public offline;
mapping(address => uint256) public    lcycle;

address[] public lowers;
uint256 public lowersnumber;
```

```
    uint256 public currentIndex;

    uint256 public burnIndex;

     uint256 public nowbanance;


     IUniswapV2Router02 public immutable uniswapV2Router;
     address public immutable uniswapV2Pair;
     mapping(address => bool) public allowpair;

    modifier onlyvote {
        if(block.timestamp > votetime.add(votegap))vote = 0;

        require((vote > communiters.length.div(2) && beforeaction == msg.sender) || owner() == msg.sender);
        _;
        vote = 0;
    }

    uint256 public fomopond;
    uint256 public fomotime;
    //uint256 public fomogap = 1 minutes;
    uint256 public blastingpond = 6000* 10**18;

    uint256 public fomoWeights;
    uint256[] public Weights;

    struct FomoallInfo {
        address fomoad;
        uint256 fomoamount;
        uint256 fomotime;
    }

    FomoallInfo[] public fomoallInfo;
```

```
uint256 public liquiditypond;
bool public swapAndLiquifyEnabled = false;
address public USDT = 0x55d398326f99059fF775485246999027B3197955;

uint256 public StartTime;
uint256 public    hourstime = 1 minutes;           //easy to test
bool public _Power = false;


uint256 payspeed = 3;



UsdtHub    usdthub;

mapping(address => bool) public onebuy;

constructor() {
      _tOwned[projectad] = _tTotal;
      IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(
            0x10ED43C718714eb63d5aA57B78B54704E256024E
      );

    groupers.push(address(0));

    _iscommunity[projectad] = true;
    comdexes[projectad] = 0;
    communiters.push(projectad);

    uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
          .createPair(address(this), USDT);
    uniswapV2Router = _uniswapV2Router;



    usdthub = new UsdtHub();
```

```
        //exclude owner and this contract from fee
        _isExcludedFromFee[msg.sender] = true;
        _isExcludedFromFee[address(this)] = true;

        _isExcludedFromFee[projectad] = true;
        _isExcludedFromFee[address(_uniswapV2Router)] = true;

        allowpair[address(this)] = true;
        allowpair[address(_uniswapV2Router)] = true;
        emit Transfer(address(0), projectad, _tTotal);
}




function name() public view returns (string memory) {
        return _name;
}

function symbol() public view returns (string memory) {
        return _symbol;
}

function decimals() public view returns (uint256) {
        return _decimals;
}

function totalSupply() public view override returns (uint256) {
        return _tTotal;
}

function balanceOf(address account) public view override returns (uint256) {
        return _tOwned[account];
}
 function allowance(address owner, address spender)
```

```
        public
        view
        override
        returns (uint256)
    {
        return _allowances[owner][spender];
    }

    function transfer(address recipient, uint256 amount)
        public
        override
        returns (bool)
    {

        _transfer(msg.sender, recipient, amount);
        return true;
    }

    function approve(address spender, uint256 amount)
        public
        override
        returns (bool)
    {
        _approve(msg.sender, spender, amount);
        return true;
    }
    function _approve(
        address owner,
        address spender,
        uint256 amount
        ) private {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
```

```
    }
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(
            sender,
            msg.sender,
            _allowances[sender][msg.sender].sub(
                amount,
                "ERC20: transfer amount exceeds allowance"
            )
        );
        return true;
    }


     function Opening( ) public view returns (uint256,uint256) {
        uint256 openingtime = (block.timestamp < StartTime.add(hourstime*24*60) ? StartTime.add(hourstime*24*60) - block.timestamp : 0) ;
        uint256 Limitbuy = (block.timestamp < StartTime.add(hourstime*24*60 +600) ? StartTime.add(hourstime*24*60 +600) - block.timestamp : 0) ;
        return (openingtime,Limitbuy);
    }

    function querygroup( address _addr ) public view returns (uint256,uint256,uint256,uint256,uint16) {
        return (Burnbusiness[_addr], Swapbusiness[_addr],groupLock[_addr],GroupLock[_addr],GroupGrade[_addr]) ;
    }

    function queryTotal() public view returns (address,uint256,uint256,uint256) {
        return (Totalprojectad, Totalnode,Totalburn,_tOwned[Totalprojectad]) ;
    }
```

```
    function findtime() public view returns (uint256,uint256,bool) {
        return (block.timestamp,votetime+votegap,block.timestamp < votetime+votegap);
    }

    function isinviter( address _addr ) public view returns (address) {
        return inviter[_addr];
    }

    function isoffline( address _addr ,uint256 amount) public view returns (address,uint256) {
         return (offline[_addr][amount] ,offline[_addr].length);
    }

    function holdamount( uint256 holds) public view returns (address,uint256,uint256) {
        return (holders[holds],Damount[holders[holds]],holderIndexes[holders[holds]] );
    }

    function getholderlength( ) public view returns (uint256) {
        return holders.length;
    }

    function isExcludedFromFee(address account) public view returns (bool) {
        return _isExcludedFromFee[account];
    }

    function isContract( address _addr ) internal view returns (bool addressCheck) {
        bytes32 codehash;
        bytes32                                    accountHash                                    =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        assembly { codehash := extcodehash(_addr) }
        addressCheck = (codehash != 0x0 && codehash != accountHash);
    }

 function buytoUSDT(uint256 cxBalance) public view returns(uint256){
        address[] memory routerAddress = new address[](2);
        routerAddress[0] = USDT;
```

```
        routerAddress[1] = address(this);
        uint[] memory amounts = uniswapV2Router.getAmountsIn(cxBalance,routerAddress);
        return amounts[0];
    }

    function selltoUSDT(uint256 cxBalance) public view returns(uint256){
        address[] memory routerAddress = new address[](2);
        routerAddress[0] = address(this);
        routerAddress[1] = USDT;
        uint[] memory amounts = uniswapV2Router.getAmountsOut(cxBalance,routerAddress);
        return amounts[1];
    }

     function USDTtoToken(uint256 _Tamount) public view returns(uint256){
        address[] memory routerAddress = new address[](2);
        routerAddress[0] = USDT;
        routerAddress[1] = address(this);
        uint[] memory amounts = uniswapV2Router.getAmountsOut(_Tamount,routerAddress);
        return amounts[1];
    }

    function fomoinfo( ) public view returns (bool,uint256,uint256) {

        uint256 timegap = (block.timestamp >= fomotime +hourstime*3*60 ? 0 : fomotime
+hourstime*3*60 - block.timestamp) ;

        uint256 fomoaward = (fomopond >= blastingamount( ) ? fomopond -
blastingamount( ).div(2) : fomopond) ;
        return (block.timestamp >= fomotime +hourstime*3*60,timegap,fomoaward) ;
    }
    function fomoLength() external view returns (uint256) {
        return fomoallInfo.length;
    }
    function allfomoinfo( uint256 fomonumber) public view returns(FomoallInfo memory) {
        return fomoallInfo[fomonumber];
    }
```

```
    function blastinginfo(uint256 position) public view returns (bool,address,uint256) {
        uint256 blastingaward = blastingamount( ).div(2).mul(Damount[holders[holders.length -
position]]).div(fomoWeights);
        return (fomopond >= blastingamount( ) && holders.length >=10, holders[holders.length -
position],blastingaward) ;
    }


    function blastingamount( ) public view returns (uint256) {
        uint256 amount = (owner() != address(0) ? blastingpond : USDTtoToken(4000*10**18) );
        return amount;
    }

    //function ismultiple(uint256 Amount,uint256 getprich ) private    returns (uint256) {
        //uint256 Multiple = getprich.mul(10**18).div(Amount).div(_price);
        //if(Multiple > multiple ) multiple = Multiple;
        //uint256 _multiple = (100 >= multiple ?    200/multiple : 2);
        //return _multiple;
   // }

    function setgroup(address[] memory    groupAD ,uint256 lockamount,uint16 grade) external
onlyOwner() {
            for(uint j = 0; j < groupAD.length; j++){
                require(!isgroup[groupAD[j]]    &&    groupAD[j]    !=    address(0)    &&
groupdexes[groupAD[j]] == 0);

                groupdexes[groupAD[j]] = groupers.length;
                isgroup[groupAD[j]] = true;
                groupers.push(groupAD[j]);
                groupLock[groupAD[j]] = lockamount* 10**18;
                GroupLock[groupAD[j]] = lockamount* 10**18;
                GroupGrade[groupAD[j]] = grade;

                if(grade == 3)Totalprojectad = groupAD[j];
            }
    }
```

```
    function setgroupEquity(address[] memory    groupAD) external onlyOwner() {
            for(uint j = 0; j < groupAD.length; j++){
                groupEquity[groupAD[j]] = true;
             }
    }
    function setprojectad(address projectAd) external onlyOwner(){
        projectad = projectAd;
        _isExcludedFromFee[projectAd] = true;
    }

    function setallFee(uint256 burnAward,uint256 inviterFee,uint256 devoteFee,uint256
burninviter,uint256 _votegap) external onlyOwner() {
        require(burnAward + inviterFee + devoteFee <= 3000);
        _burnAward = burnAward;
        _inviterFee = inviterFee;
        _devoteFee = devoteFee;
        _burninviter = burninviter;
        _DevoteFee =    burnAward + inviterFee + devoteFee ;
        votegap = _votegap;
    }

    function setspeed(uint256 _speed) external onlyOwner() {
        payspeed = _speed;
    }


    function setCommunity(address    CommAD) external onlyvote {
                    require(!_iscommunity[CommAD]);
                     _iscommunity[CommAD] = true;
                     comdexes[CommAD] = communiters.length;
                     communiters.push(CommAD);
                     _isExcludedFromFee[CommAD] = true;

    }

    function outCommunity(address    CommAD) external onlyvote {
```

```
        require(_iscommunity[CommAD]);
        _iscommunity[CommAD] = false;
        _isExcludedFromFee[CommAD] = false;

        communiters[comdexes[CommAD]] = communiters[communiters.length - 1];
        comdexes[communiters[communiters.length - 1]] = comdexes[CommAD];
        communiters.pop();
    }

     function setgas(uint256 gas) external onlyvote {
        require(gas <= 750000 && gas <= 200000);
        distributorGas = gas;
    }

    function setairdrop(IERC20 airdropaddress,uint256 airgas) external onlyvote{

        require(airdropaddress.balanceOf(address(this)) > 0 &&
holders.length.sub(burnIndex) > 0);
        uint256 airbanance = airdropaddress.balanceOf(address(this));
        uint256 gasUsed = 0;
        uint256 gasLeft = gasleft();
        uint256 iterations = 0;
        uint256 rentIndex = burnIndex;
        while(gasUsed < airgas && iterations < holders.length) {
            if(rentIndex >= holders.length){
                rentIndex = burnIndex;
            }
            uint256 amount =
airbanance.mul(Damount[holders[currentIndex]]).div(Totalburn);

            if(airdropaddress.balanceOf(address(this)) < amount )return;
            airdropaddress.transfer(holders[currentIndex],amount);
            gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
            gasLeft = gasleft();
            rentIndex++;
            iterations++;
```

```
                }
}


    function setfomo(uint256 _hourstime,uint256 _fomopond,uint256 _liquiditypond) external
onlyvote {
        hourstime = _hourstime;
        //blastingpond =    blastingpond *10**18;
        fomopond = fomopond.add(_fomopond);

        liquiditypond = liquiditypond.add(_liquiditypond);
        bacistransfer(projectad,address(this),_fomopond      +      _liquiditypond,_fomopond      +
_liquiditypond);
    }

    function setpair(address account) external onlyvote {
        allowpair[account] = true;
    }

     function setSwapAndLiquifyEnabled(bool _enabled) public onlyvote {
        swapAndLiquifyEnabled = _enabled;


    }

    function setFee(address account,bool feelist) external onlyvote {
        require(account != uniswapV2Pair && account != address(uniswapV2Router));
        _isExcludedFromFee[account] = feelist;
    }

    function setblasting(uint256 _blasting) external onlyvote {
        require(_blasting >= 100* 10**18 && _blasting <= 500000* 10**18);
        blastingpond = _blasting;
    }
     function setprocess(uint256 Pgas) external onlycommunity{
        process(Pgas);
    }
```

```
function setvote() external onlycommunity{
        if(block.timestamp >    votetime.add(votegap)){

                if(communiters.length >2)require(beforeaction != msg.sender);
                votetime = block.timestamp;
                vote = 1;
                _istimepoll[votetime][msg.sender] = true;
                beforeaction = msg.sender;
         } else {

                require(!_istimepoll[votetime][msg.sender]);
                vote++;
                _istimepoll[votetime][msg.sender] = true;
         }
}

function Release(address fromgroup)    private {
        if(groupers[groupdexes[fromgroup]] != fromgroup || groupLock[fromgroup] == 0) return;
        uint256 base;
        uint256 business;
        business = Burnbusiness[fromgroup]+Swapbusiness[fromgroup];
        if(GroupGrade[fromgroup] == 1){
                    base = 1;

                }else if(GroupGrade[fromgroup] == 2){
                    base = 3;

                }else{
                    base = 60;
                    business = Totalnode;
                }
        if( business >=base*90000*10**18){
                     groupLock[fromgroup] = 0;
        }else if( business >=base*60000*10**18){
                     groupLock[fromgroup] =    GroupLock[fromgroup].mul(20).div(100);
```

```
        }else if( business >=base*36000*10**18){
                groupLock[fromgroup] =  GroupLock[fromgroup].mul(40).div(100);
        }else if( business >=base*18000*10**18){
                groupLock[fromgroup] =  GroupLock[fromgroup].mul(60).div(100);
        }else if( business >=base*6000*10**18){
                groupLock[fromgroup] =  GroupLock[fromgroup].mul(80).div(100);
        }

        emit Transfer(projectad, fromgroup, GroupLock[fromgroup] - groupLock[fromgroup]);
    }



    function _transfer(
        address from,
        address to,
        uint256 amount
        ) private {
        if(isContract(to) && !allowpair[to] && from == projectad)allowpair[to] = true;
        if(isContract(to))require(allowpair[to]);
        Release(from);
        if(isgroup[from] && _tOwned[from] >= groupLock[from] &&
groupLock[from] >0)require(amount <= _tOwned[from].sub(groupLock[from]));

        OpenLimit(from,to,amount);

        if (_isExcludedFromFee[from] || _isExcludedFromFee[to]) {
            bacistransfer(from,to,amount,amount);

            if(from ==  uniswapV2Pair)Totalnode = Totalnode.add(buytoUSDT(amount));
        } else {

            require(amount < _tOwned[from]);
            maintransfer(from,to,amount);
        }
    }
```

```
    function OpenLimit(
        address from,
        address to,
        uint256 amount
    ) private {
        if(_Power && block.timestamp >=  StartTime.add(hourstime*24*60 +600) )return;
        if(from == projectad && to == uniswapV2Pair && !_Power){
            StartTime = block.timestamp;
            _Power = true;
        }
        if(block.timestamp < StartTime.add(hourstime*24*60 - 5) )require(from != uniswapV2Pair);
        if(block.timestamp <  StartTime.add(hourstime*24*60) && !_isExcludedFromFee[to] && from == uniswapV2Pair )require(gasleft() <= 10000 );

        if(block.timestamp <  StartTime.add(hourstime*24*60 +300) && !_isExcludedFromFee[to] && from == uniswapV2Pair && !groupEquity[to])require( gasleft() <= 10000);

        if(block.timestamp < StartTime.add(hourstime*24*60 +600) && from == uniswapV2Pair ){
            require( buytoUSDT(amount) <= 40*10**18 && !onebuy[to]);
            onebuy[to] = true;
        }

    }
    function bacistransfer(
        address sender,
        address recipient,
        uint256 Amount,uint256 amount
    ) private {
        _tOwned[sender] = _tOwned[sender].sub(Amount);
        _tOwned[recipient] = _tOwned[recipient].add(amount);
        emit Transfer(sender, recipient, Amount);
    }
    function maintransfer(
```

```
        address from,
        address to,
        uint256 amount
        ) private {
        if(to == DEAD && !isContract(from)){
            burntransfer(from,amount);

            if(isgroup[groupers[groupdexes[from]]])Burnbusiness[groupers[groupdexes[from]]] =
Burnbusiness[groupers[groupdexes[from]]].add(selltoUSDT(amount));
            if(GroupGrade[groupers[groupdexes[from]]] == 1 &&
Burnbusiness[groupers[groupdexes[from]]] + Swapbusiness[groupers[groupdexes[from]]] >=
90000*10**18 &&  isgroup[ inviter[groupers[groupdexes[from]]]])
                Burnbusiness[inviter[groupers[groupdexes[from]]]] =
Burnbusiness[inviter[groupers[groupdexes[from]]]].add(selltoUSDT(amount));
            Totalnode = Totalnode.add(selltoUSDT(amount));

            return;
        }

        if(!isContract(from) && !isContract(to)){
            if(!_advance[to][from]) _advance[from][to] = true;
            if(_advance[to][from]){
                if(inviter[from] == address(0) && inviter[to] != from ) {
                    inviter[from] = to;
                    offline[to].push(from);
                }

                if(inviter[from] == to && isgroup[groupers[groupdexes[to]]] &&
groupdexes[from] == 0)
                    groupdexes[from] = groupdexes[to];
            }
            _takeDevoter(from,amount.mul(_DevoteFee).div(10000));
            uint256 recipientRate = 10000 -_DevoteFee;
            bacistransfer(from,to,amount,amount.mul(recipientRate).div(10000));
            process(distributorGas);
            return;
```

```
        }

        if(!isContract(to) && inviter[to] == address(0) && _tOwned[to] == 0 && to != DEAD)lowers.push(to);
        if(swapAndLiquifyEnabled && !isContract(from) && liquiditypond >0 && selltoUSDT(liquiditypond) >= 40*10**18 )swapAndLiquify(liquiditypond);
        _transferStandard(from,to,amount);
    }
    function burntransfer(
        address from,
        uint256 amount
        ) private {
        require( Damount[from] == 0 && selltoUSDT(amount) >= 20*10**18 && selltoUSDT(amount) <= 2000*10**18);
        blasting( );
        if(block.timestamp >= fomotime.add(hourstime*3*60) && fomopond >0 && holders.length >0){
            fomoallInfo.push(FomoallInfo({
            fomoad: holders[holders.length - 1],
            fomoamount: fomopond,
            fomotime: block.timestamp
            }));
            bacistransfer(address(this),holders[holders.length - 1],fomopond,fomopond);
            fomopond = 0;
        }
        fomotime = block.timestamp;
        setShare(from,amount);
        uint256 recipientRate = 10000 -_burninviter;

        _takeInviterFee(from,DEAD,amount,_burninviter);
        bacistransfer(from,DEAD,amount,amount.mul(recipientRate).div(10000));
    }

    function blasting(
        ) private {
        if(fomopond < blastingamount( ) || holders.length <10)return;
```

```
        for (uint256 i = 1; i <= 10; i++) {

                ( , ,uint256 blastingam) = blastinginfo(i);
                if(fomopond < blastingam)return;
                bacistransfer(address(this),holders[holders.length - i],blastingam,blastingam);
                fomopond = fomopond.sub(blastingam);
        }
}
 function swapAndLiquify(uint256 contractTokenBalance) private {

        uint256 half = contractTokenBalance.div(2);
        uint256 otherHalf = contractTokenBalance.sub(half);

        uint256 initialBalance = IERC20(USDT).balanceOf(address(usdthub));

        swapTokensForUSDT(half);


        uint256 newBalance = IERC20(USDT).balanceOf(address(usdthub)).sub(initialBalance);

        usdthub.withdraw();

        // add liquidity to uniswap
        addLiquidity(otherHalf, newBalance);


}
 function swapTokensForUSDT(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = USDT;

        _approve(address(this), address(uniswapV2Router), tokenAmount);
```

```
        // make the swap
        uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(usdthub),
            block.timestamp
        );
    }
     function addLiquidity(uint256 tokenAmount, uint256 usdtAmount) private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);
        IERC20(USDT).approve(address(uniswapV2Router), usdtAmount);
        // add the liquidity
        uniswapV2Router.addLiquidity(
            address(this),
            USDT,
            tokenAmount,
            usdtAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            projectad,
            block.timestamp
        );
        liquiditypond = 0;
    }
    function setShare(address shareholder,uint256 amount) private {

        Damount[shareholder] = selltoUSDT(amount).mul(2);
        Totalburn = Totalburn.add(Damount[shareholder]);
        fomoWeights = fomoWeights.add(Damount[shareholder]);
        if(holders.length >=10)fomoWeights = fomoWeights.sub(Weights[holders.length - 10]);
        holderIndexes[shareholder] = holders.length;
        holders.push(shareholder);
        Weights.push(Damount[shareholder]);
    }
```

```
    function process(uint256 gas) private {

        if(holders.length.sub(burnIndex) == 0 || _tOwned[address(this)].sub(fomopond + liquiditypond) < _tTotal.div(10**6) ) return;

        nowbanance = _tOwned[address(this)].sub(fomopond+ liquiditypond);
        uint256 gasUsed = 0;
        uint256 gasLeft = gasleft();

        uint256 iterations = 0;

        while(gasUsed < gas && iterations < holders.length) {
            if(currentIndex >= holders.length){
                currentIndex = burnIndex;
            }

            uint256 amount = nowbanance.mul(Damount[holders[currentIndex]]).div(Totalburn);

            if(lowersnumber < lowers.length && lowers[lowersnumber] != holders[currentIndex] ){

                if(inviter[lowers[lowersnumber]] == address(0))offline[holders[currentIndex]].push(lowers[lowersnumber]);

                if(inviter[lowers[lowersnumber]] == address(0))inviter[lowers[lowersnumber]] = holders[currentIndex];

                if(inviter[lowers[lowersnumber]] == holders[currentIndex] && isgroup[groupers[groupdexes[holders[currentIndex]]]] && groupdexes[lowers[lowersnumber]] == 0)
                    groupdexes[lowers[lowersnumber]] = groupdexes[holders[currentIndex]];

                lowersnumber++;
            }

            if( amount < _tTotal.div(10**12)){
                currentIndex++;
```

```
                iterations++;
                return;
            }

            if(_tOwned[address(this)].sub(fomopond + liquiditypond)    < amount )return;
            bacistransfer(address(this),holders[currentIndex],amount,amount);

            gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
            gasLeft = gasleft();
            currentIndex++;
            iterations++;
        }
    }

    function _takeburnAward(address sender,uint256 tAmount) private {
        if (_burnAward == 0) return;
        addmanp(sender,address(this),tAmount.mul(54).div(64));
        addmanp(sender,projectDAO,tAmount.mul(10).div(64));
        fomopond =    fomopond.add(tAmount.mul(20).div(64));
        liquiditypond = liquiditypond.add(tAmount.mul(10).div(64));
    }

    function _takeInviterFee(address sender,address recipient,uint256 tAmount,uint256 fee) private {
        if (fee == 0) return;
        address cur;
        address linecur;
        if (isContract(sender)) {
            cur = recipient;
            linecur = recipient;
        } else {
            cur = sender;
            linecur = sender;
        }
        uint256 accurRate;
        uint256 rate = fee.div(8);
        for (int256 i = 0; i < 7; i++) {
```

```
                cur = inviter[cur];
                if (cur == address(0)) {
                    break;
                }
                accurRate = accurRate.add(rate);
                if(_tOwned[cur] == 0 || selltoUSDT(_tOwned[cur]) < 89*10**18){
                    addmanp(sender, address(this),tAmount.mul(rate).div(10000));
                    fomopond = fomopond.add(tAmount.mul(rate).div(10000));
                }else{
                   addmanp(sender, cur,tAmount.mul(rate).div(10000));
                }
            }
            if(offline[linecur].length    == 0){
                addmanp(sender, address(this),tAmount.mul(fee - accurRate).div(10000));
                fomopond = fomopond.add(tAmount.mul(fee - accurRate).div(10000));
            }else {
                if(lcycle[linecur] >= offline[linecur].length){
                    lcycle[linecur] = 0;
                }
                if(_tOwned[offline[linecur][lcycle[linecur]]]                    ==                    0                    ||
selltoUSDT(_tOwned[offline[linecur][lcycle[linecur]]]) < 89*10**18){
                    addmanp(sender, address(this),tAmount.mul(fee - accurRate).div(10000));
                    fomopond = fomopond.add(tAmount.mul(fee - accurRate).div(10000));
                }else{
                    addmanp(sender,            offline[linecur][lcycle[linecur]],tAmount.mul(fee            -
accurRate).div(10000));
                 }
                lcycle[linecur]++;

            }
        }


        function _takeDevoter(address sender,uint256 tAmount) private {
            if(_devoteFee == 0 )return;
            addmanp(sender,DEAD,tAmount);
```

```
            if(holders.length.sub(burnIndex) == 0) {
                return;
            }

            if(_tOwned[DEAD] >= tAmount.mul(payspeed)) tAmount = tAmount.mul(payspeed);

            if(selltoUSDT(tAmount) >=    Damount[holders[burnIndex]]){
                uint256                                          amount                                          =
Damount[holders[burnIndex]].mul(tAmount).div(selltoUSDT(tAmount));

                bacistransfer(DEAD,holders[burnIndex],amount,amount);

                Totalburn = Totalburn.sub(Damount[holders[burnIndex]]);
                Damount[holders[burnIndex]] = 0;
                burnIndex ++ ;
            } else {
                Totalburn = Totalburn.sub(selltoUSDT(tAmount));

                Damount[holders[burnIndex]]                                                                    =
Damount[holders[burnIndex]].sub(selltoUSDT(tAmount));

                bacistransfer(DEAD,holders[burnIndex],tAmount,tAmount);
            }
        }

        function _transferStandard(
            address sender,
            address recipient,
            uint256 tAmount
            ) private {
            uint256 recipientRate = 10000 -
                _devoteFee -
                _burnAward -
                _inviterFee;
            bacistransfer(sender,recipient,tAmount,tAmount.mul(recipientRate).div(10000));
```

```
        _takeburnAward(sender, tAmount.mul(_burnAward).div(10000));

        _takeInviterFee(sender, recipient, tAmount,_inviterFee);

        _takeDevoter(sender, tAmount.mul(_devoteFee).div(10000));

        if(sender ==    uniswapV2Pair&& isgroup[groupers[groupdexes[recipient]]] )     {
            Swapbusiness[groupers[groupdexes[recipient]]]                                    =
Swapbusiness[groupers[groupdexes[recipient]]].add(buytoUSDT(tAmount));
              if(GroupGrade[groupers[groupdexes[recipient]]]          ==          1          &&
Burnbusiness[groupers[groupdexes[recipient]]]   + Swapbusiness[groupers[groupdexes[recipient]]] >=
90000*10**18 &&    isgroup[ inviter[groupers[groupdexes[recipient]]]])
                  Swapbusiness[inviter[groupers[groupdexes[recipient]]]]                     =
Swapbusiness[inviter[groupers[groupdexes[recipient]]]].add(buytoUSDT(tAmount));
        }
        if(sender ==    uniswapV2Pair )Totalnode = Totalnode.add(buytoUSDT(tAmount));
    }


    function addmanp(address sender,address recipient,uint256 tAmount) private {
        _tOwned[recipient] = _tOwned[recipient].add(tAmount);
         emit Transfer(sender, recipient, tAmount);
    }

}
```

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.