# Smart contract audit report

# BFAV

CHAINLION

NO.0C002206290001

JUNE 29, 2022

## CATALOGUE

## 1. PROJECT SUMMARY

| Entry type | Specific description |
|---|---|
| Entry name | BFAV |
| Project type | DEFI |
| Application platform | BSC |

## 2. AUDIT SUMMARY

| Entry type | Specific description |
|---|---|
| Project cycle | JUNE/26/2022-JUNE/29/2022 |
| Audit method | Black box test、White box test、Grey box test |
| Auditors | Two |

## 3. VULNERABILITY SUMMARY

Audit results are as follows:

| Entry type | Specific description |
|---|---|
| **Serious vulnerability** | 0 |
| **High risk vulnerability** | 0 |
| **Moderate risk** | 3 |
| **Low risk vulnerability** | 3 |

Security vulnerability rating description：

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

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

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

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

## 4. EXECUTIVE SUMMARY

This report is prepared for **BFAV** 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.

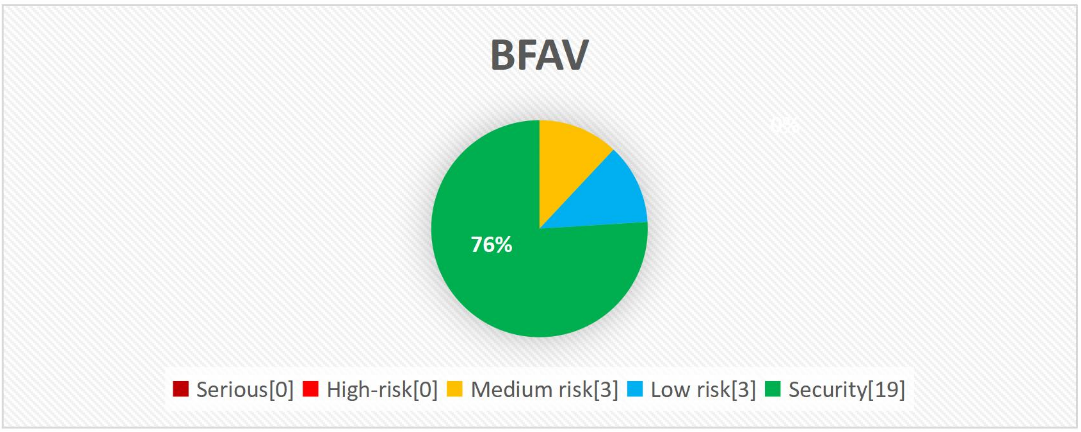
## 5. Directory structure

```
BFAV.sol
```

## 6. File hashes

| Contract | SHA1 Checksum |
|---|---|
| BFAV.sol | B6459B9C2CDB71FEE9756E92AA6FBA296B519EA5 |

## 7. Vulnerability distribution



# 8. Audit content

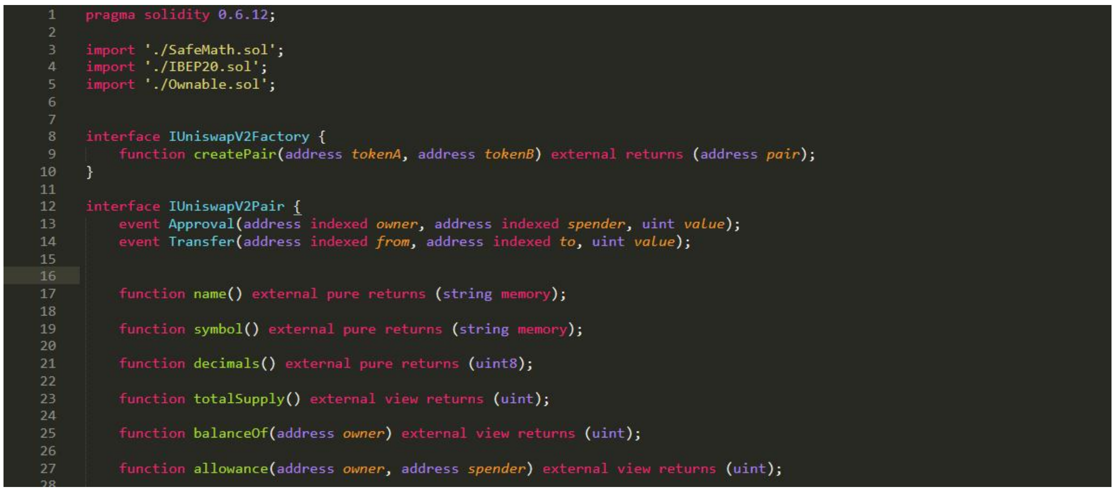
## 8.1. Coding specification

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

### 8.1.1. Compiler Version 【security】

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

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



```
pragma solidity 0.6.12;

import './SafeMath.sol';
import './IBEP20.sol';
import './Ownable.sol';


interface IUniswapV2Factory {
    function createPair(address tokenA, address tokenB) external returns (address pair);
}

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


    function name() external pure returns (string memory);

    function symbol() external pure returns (string memory);

    function decimals() external pure returns (uint8);

    function totalSupply() external view returns (uint);

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

    function allowance(address owner, address spender) external view returns (uint);
```

**Safety advice： NONE.**

## 8.1.2. Return value verification 【security】

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

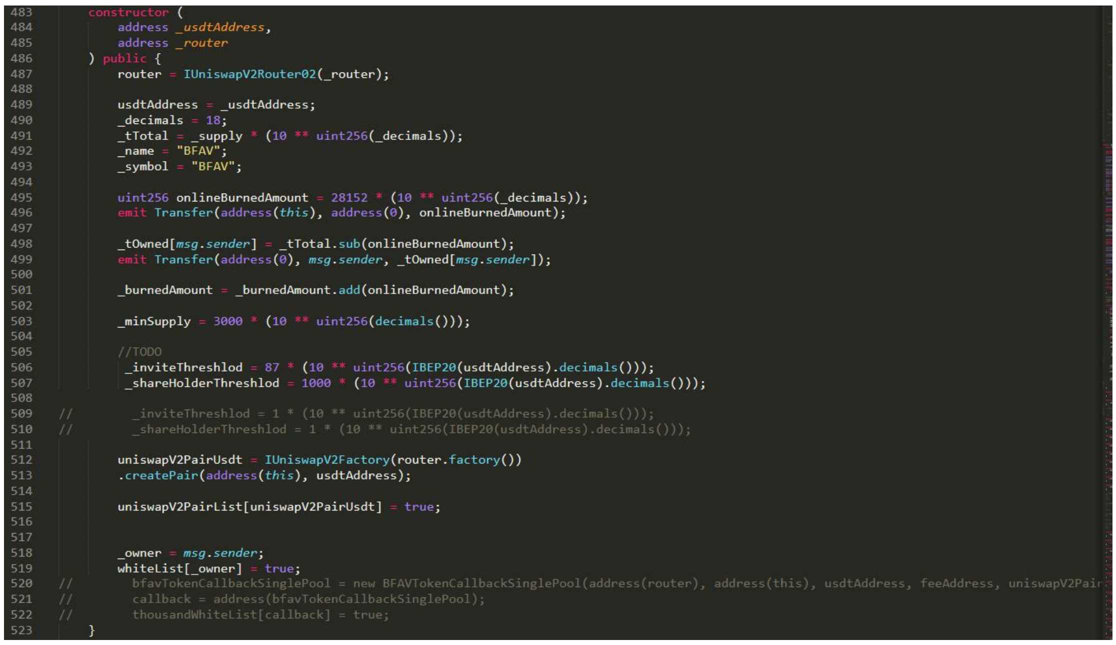
**Audit results：** According to the audit, there is no embedded function calling the official standards transfer and transferfrom in the smart contract, so there is no such security problem.

**Safety advice： NONE.**

### 8.1.3. Constructor writing 【security】

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

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



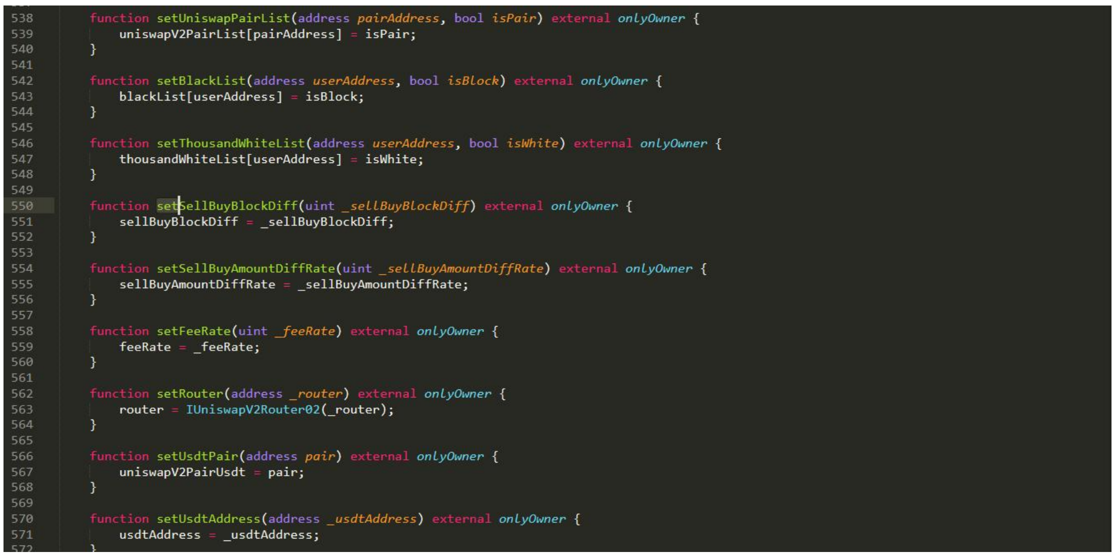
```
    constructor (
        address _usdtAddress,
        address _router
    ) public {
        router = IUniswapV2Router02(_router);

        usdtAddress = _usdtAddress;
        _decimals = 18;
        _tTotal = _supply * (10 ** uint256(_decimals));
        _name = "BFAV";
        _symbol = "BFAV";

        uint256 onlineBurnedAmount = 28152 * (10 ** uint256(_decimals));
        emit Transfer(address(this), address(0), onlineBurnedAmount);

        _tOwned[msg.sender] = _tTotal.sub(onlineBurnedAmount);
        emit Transfer(address(0), msg.sender, _tOwned[msg.sender]);

        _burnedAmount = _burnedAmount.add(onlineBurnedAmount);

        _minSupply = 3000 * (10 ** uint256(decimals()));

        //TODO
         _inviteThreshlod = 87 * (10 ** uint256(IBEP20(usdtAddress).decimals()));
         _shareHolderThreshlod = 1000 * (10 ** uint256(IBEP20(usdtAddress).decimals()));

//         _inviteThreshlod = 1 * (10 ** uint256(IBEP20(usdtAddress).decimals()));
//         _shareHolderThreshlod = 1 * (10 ** uint256(IBEP20(usdtAddress).decimals()));

        uniswapV2PairUsdt = IUniswapV2Factory(router.factory())
        .createPair(address(this), usdtAddress);

        uniswapV2PairList[uniswapV2PairUsdt] = true;


        _owner = msg.sender;
        whiteList[_owner] = true;
//         bfavTokenCallbackSinglePool = new BFAVTokenCallbackSinglePool(address(router), address(this), usdtAddress, feeAddress, uniswapV2Pair
//         callback = address(bfavTokenCallbackSinglePool);
//         thousandWhiteList[callback] = true;
    }
```

**Safety advice： NONE.**

### 8.1.4. Key event trigger 【Low risk】

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

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



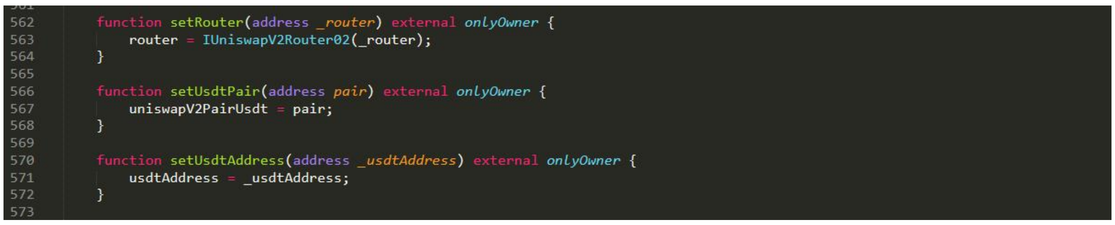
```
function setUniswapPairList(address pairAddress, bool isPair) external onlyOwner {
    uniswapV2PairList[pairAddress] = isPair;
}

function setBlackList(address userAddress, bool isBlock) external onlyOwner {
    blackList[userAddress] = isBlock;
}

function setThousandWhiteList(address userAddress, bool isWhite) external onlyOwner {
    thousandWhiteList[userAddress] = isWhite;
}

function setSellBuyBlockDiff(uint _sellBuyBlockDiff) external onlyOwner {
    sellBuyBlockDiff = _sellBuyBlockDiff;
}

function setSellBuyAmountDiffRate(uint _sellBuyAmountDiffRate) external onlyOwner {
    sellBuyAmountDiffRate = _sellBuyAmountDiffRate;
}

function setFeeRate(uint _feeRate) external onlyOwner {
    feeRate = _feeRate;
}

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

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

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

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

### 8.1.5. Address non-zero check 【Low risk】

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

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



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

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

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

**Safety advice： Strictly check the legitimacy of the address.**

### 8.1.6. Code redundancy check 【security】

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

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

**Safety advice：** **NONE.**

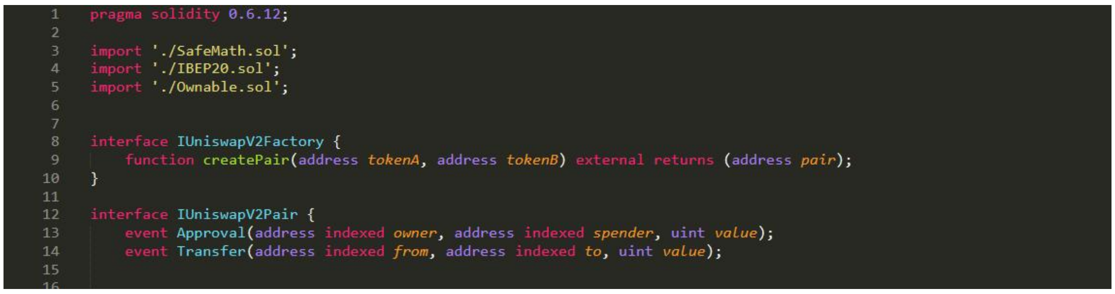
## 8.2. Coding design

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

### 8.2.1. Shaping overflow detection 【security】

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

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



```
pragma solidity 0.6.12;

import './SafeMath.sol';
import './IBEP20.sol';
import './Ownable.sol';


interface IUniswapV2Factory {
    function createPair(address tokenA, address tokenB) external returns (address pair);
}

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

```

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

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

**Code file：** BFAV.sol L483~523

**Code information：**

```
address public _owner;
    address public foundationAddress = 0xa9056272Ca777a63ae3A275d7aab078fd90A1691;
//Foundation address
    address public feeAddress = 0xF8f21e8CE19099399C7A15Bd205e87C8B571bd6E;    //Fee
address
    uint public feeRate = 13;  //Service charge transfer 13%

    uint public foundationRate = 4;  //dao community construction cost
    uint public inviteRelationRate = 1;  // 1%
    uint public parentOneRelationRate = 2;  // 2%
    uint public parentTwoRelationRate = 2;  // 2%
    uint public parentThreeRelationRate = 2;    // 2%
    uint public blackHoleRate = 2;  // black hole  2%
//    uint public liquidRate = 5;
//    uint public liquidRewardRate = 5;
```

```
    address public uniswapV2PairUsdt;

    uint256 public _supply = 90000;    //Total quantity of 9W pieces initially provided

    address burnAddress = address(0);    //Destruction address
    mapping(address => bool) public blackList;    //Blacklist address
    mapping(address => bool) public whiteList;    //White list address

    mapping(address => bool) public thousandWhiteList;

    mapping(address => bool) public uniswapV2PairList;
    bool public useWhiteListSwith = true;    //White list switch

    address public    callback;    //Fallback address
//      BFAVTokenCallbackSinglePool bfavTokenCallbackSinglePool;
    IUniswapV2Router02 public router;    //router address
    address public usdtAddress;      //usdt address

    uint256 inviterLength = 200;    //Maximum invitation length

    mapping(address => address[]) public memberInviter;    //Invite people
    mapping(address => address) public inviter;    //Invitee
    mapping(address => uint256 ) userInviteNumber;    //Number of people invited

    uint256 sellBuyBlockDiff = 3; //Robots within 3 blocks
    uint256 sellBuyAmountDiffRate = 2; //Quantity difference within 2%

    uint256 internal _minSupply;    //Minimum supply
    uint256 _burnedAmount;      //Quantity destroyed

    uint256    _inviteThreshlod;      // Invitation threshold
    uint256    _shareHolderThreshlod;    //Shareholder threshold

    constructor (
        address _usdtAddress,
        address _router
    ) public {
        router = IUniswapV2Router02(_router);   //Initial uniswapv2router address

        usdtAddress = _usdtAddress;   //usdt address
        _decimals = 18;     //Token precision
        _tTotal = _supply * (10 ** uint256(_decimals));   //Total issued 9W pieces
        _name = "BFAV";      //Token name
        _symbol = "BFAV";   //Token ID

        uint256 onlineBurnedAmount = 28152 * (10 ** uint256(_decimals)); //Total online
destruction
        emit Transfer(address(this), address(0), onlineBurnedAmount);
        _tOwned[msg.sender] = _tTotal.sub(onlineBurnedAmount);    //Update total tokens(Initial
total quantity - destroyed quantity)
        emit Transfer(address(0), msg.sender, _tOwned[msg.sender]);
        _burnedAmount = _burnedAmount.add(onlineBurnedAmount); //Update destroy count
        _minSupply = 3000 * (10 ** uint256(decimals())); //Minimum quantity 3000 pieces

        //TODO
         _inviteThreshlod = 87 * (10 ** uint256(IBEP20(usdtAddress).decimals()));
         _shareHolderThreshlod = 1000 * (10 ** uint256(IBEP20(usdtAddress).decimals()));

//           _inviteThreshlod = 1 * (10 ** uint256(IBEP20(usdtAddress).decimals()));
//           _shareHolderThreshlod = 1 * (10 ** uint256(IBEP20(usdtAddress).decimals()));

        uniswapV2PairUsdt = IUniswapV2Factory(router.factory())
        .createPair(address(this), usdtAddress); //Create transaction pair

        uniswapV2PairList[uniswapV2PairUsdt] = true;

        _owner = msg.sender;   //Update the owner of the contract
        whiteList[_owner] = true; //Add owner to the white list
//              bfavTokenCallbackSinglePool = new BFAVTokenCallbackSinglePool(address(router),
address(this), usdtAddress, feeAddress, uniswapV2PairUsdt);
//           callback = address(bfavTokenCallbackSinglePool);
//              thousandWhiteList[callback] = true;
        }
```

**Safety advice： NONE.**

### 8.3.2. Transferowner logic design 【Low risk】

**Audit description：** Conduct security audit on the transferowner business logic design in the contract.

**Audit results：** The transferowner function in the contract is used to update the owner address of the contract. This function can only be called by the owner of the contract. The permission design is reasonable, but the non-zero check of the address is not carried out. There is a design defect.

**Code file：** BFAV.sol 529~531

**Code information：**

```
        function transferOwner(address newOwner) external onlyOwner { //Only owner calls are allowed
            _owner = newOwner;    //Missing non-zero check for address
        }
```

**Safety advice： Non zero check for address.**

### 8.3.3. Contract authority concentration detection 【Moderate risk】

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

**Audit results：** The owner in the contract has high permissions, and can set the contract owner address, blacklist address, whitelist address, transaction rate, router address, transaction pair information, etc., with highly centralized permissions.

**Code file:** BFAV.sol 538~596

**Code information:**

```
function setUniswapPairList(address pairAddress, bool isPair) external onlyOwner {
    uniswapV2PairList[pairAddress] = isPair;
}

function setBlackList(address userAddress, bool isBlock) external onlyOwner {
    blackList[userAddress] = isBlock;
}

function setThousandWhiteList(address userAddress, bool isWhite) external onlyOwner {
    thousandWhiteList[userAddress] = isWhite;
}

function setSellBuyBlockDiff(uint _sellBuyBlockDiff) external onlyOwner {
    sellBuyBlockDiff = _sellBuyBlockDiff;
}

function setSellBuyAmountDiffRate(uint _sellBuyAmountDiffRate) external onlyOwner {
    sellBuyAmountDiffRate = _sellBuyAmountDiffRate;
}

function setFeeRate(uint _feeRate) external onlyOwner {
    feeRate = _feeRate;
}

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

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

```
function setUsdtAddress(address _usdtAddress) external onlyOwner {
    usdtAddress = _usdtAddress;
}
function setFoundationAddress(address _foundationAddress) external onlyOwner {
    foundationAddress = _foundationAddress;
}
modifier onlyOwner() {
    require(msg.sender == _owner, "admin: wut?"); //Only owner calls of contracts are allowed
    _;
}
```

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

### 8.3.4. Transfer transfer business logic 【Moderate risk】

**Audit description：** Conduct a security audit on the transfer business logic in the contract, check whether there is a risk of plastic overflow, and whether the transfer logic design is reasonable, etc.

**Audit results：** The transfer business logic in the contract checks whether the number of invitees exceeds the maximum limit. Currently, the maximum limit in the contract is 200. There is a certain self DOS risk when using the for loop traversal. At the same time, there is the nature of pyramid selling (only from the maximum invitable level, there are relevant restrictions on rewards, but there is no such feature). It is recommended to reduce it to about 10, and the parameter block on the blockchain is used Difficulty, as a random number generation parameter, has the risk of being predicted.

**Code file：** BFAV.sol 630~633

**Code information：**

```
function transfer(address recipient, uint256 amount) public override returns (bool) {
        _transfer(msg.sender, recipient, amount);    //Call_ Transfer to transfer
        return true;
    }

    function _transfer(
        address from,
        address to,
        uint256 amount
    ) private {
        require(from != address(0), "ERC20: transfer from the zero address"); //Address non-zero
check
        require(amount > 0, "Transfer amount must be greater than zero"); //Transfer quantity check
        uint256 leftAmount = _tTotal.sub(_burnedAmount); //Calculate the current issued token
quantity (initial total - destroyed total)
        // if (uniswapV2PairList[to] || uniswapV2PairList[from]) {
        if (uniswapV2PairList[to] && thousandWhiteList[from] != true) { //Check if the address is
restricted
            require(amount < leftAmount.div(1000), "Transfer amount must be less than
thousandth");    //Transaction quantity must be less than 1/1000 of the remaining total
        }
        require(!blackList[from] && !blackList[to], "black transfer not allowed");    //Check whether
the transaction address is in the blacklist

        uint256 priceDownAmount = 0;
        uint256 fee = 0;
        if (whiteList[from] != true) {    //From address is not in the white list

            if (!uniswapV2PairList[to] && !uniswapV2PairList[from]
            && from != address(this) && to != address(this)
            && from != address(router) && to != address(router)
            && from != address(callback) && to != address(callback)
            ) { //Non transaction pair address, non contract address, non router address, non callback
address
                uint256 toBalance = balanceOf(to);    //Number of tokens held by the receiving
token address
```

```
                if (toBalance == 0) { //If the token quantity is 0, call addmemberinviter to set the invitation relationship (from->to)
                    addMemberInviter(from, to);
                }
            }
            address sellBuyUser = address(0);
            if (uniswapV2PairList[from]) {
                sellBuyUser = to;
            }
            if (uniswapV2PairList[to]) {
                sellBuyUser = from;
            }

            if (sellBuyUser != address(this) && sellBuyUser != address(callback) &&
            sellBuyUser != address(0) && sellBuyUser != address(router)
            && !uniswapV2PairList[sellBuyUser]) {    //The trading address is not the current contract address, not the callback address, not the zero address, not the router address, not the address in the uniswapv2pairlist
                //Anti robot
                SellBuyBlock memory sellBuyBlock = lastSellBuyBlockMap[sellBuyUser];
                uint256 amountDiff = 0;
                if (amount > sellBuyBlock.amount) {
                    amountDiff = amount.sub(sellBuyBlock.amount);
                } else {
                    amountDiff = sellBuyBlock.amount.sub(amount);
                }
                if (sellBuyBlock.amount > 0) {
                    uint256 amountDiffRate = amountDiff.mul(100).div(sellBuyBlock.amount);

                    if (block.number.sub(sellBuyBlock.blockNumber) <= sellBuyBlockDiff && amountDiffRate <= sellBuyAmountDiffRate) {
                        blackList[sellBuyUser] = true;
                        emit blackUser(from, to, amount); //Join the blacklist
                        return;
                    }
                    }
                    lastSellBuyBlockMap[sellBuyUser]      =      SellBuyBlock({blockNumber      :
block.number, amount : amount});
                }

                amount = amount.mul(999).div(1000); //A single sale cannot exceed 0.1% of the total
circulation

                if (leftAmount > _minSupply && from != callback && to != callback) {

                    fee = calculateFee(amount); //Calculate rate
                    if (fee > 0) {
                        // address    uniswapV2Pair = from;

                        //                  uint public foundationRate = 4;    //dao
                        //                  uint public inviteRelationRate = 1;
                        //                  uint public parentOneRelationRate = 2;
                        //                  uint public parentTwoRelationRate = 2;
                        //                  uint public parentThreeRelationRate = 2;
                        //                  uint public blackHoleRate = 2;



                        //                  uint public liquidRate = 5;
                        //                  uint public liquidRewardRate = 5;
                        uint256 leftAmountSubFee = leftAmount.sub(fee);
                        if (leftAmountSubFee < _minSupply) {
                            fee = leftAmount.sub(_minSupply);
                        }


                        uint256 foundationAmount = fee.mul(foundationRate).div(13); //Dao address
4% (community construction cost)

                        uint256 blackHoleAmount = fee.mul(blackHoleRate).div(13); // black hole
2%
```

```
                        uint256  inviteRewardunAssigned  =  _assignInviteReward(from,  to,  fee);
//Invitation reward

                        _tOwned[burnAddress]                                                    =
_tOwned[burnAddress].add(inviteRewardunAssigned);
                        _burnedAmount = _burnedAmount.add(inviteRewardunAssigned);
                        emit Transfer(from, burnAddress, inviteRewardunAssigned);

                        _tOwned[burnAddress] = _tOwned[burnAddress].add(blackHoleAmount);
                        _burnedAmount = _burnedAmount.add(blackHoleAmount);
                        emit Transfer(from, burnAddress, blackHoleAmount);

//                      uint256 newPrice = getNewPrice();
//                      if (uniswapV2PairList[to]) {//sell
//                          uint256 liquidRewardAmount = fee.mul(liquidRewardRate).div(13);
//
//                                                                  _tOwned[address(this)]    =
_tOwned[address(this)].add(liquidRewardAmount);
//                          emit Transfer(from, address(this), liquidRewardAmount);
//
//                          uint256 tokenProfitValue = newPrice.mul(_tOwned[address(this)]);
//                          if (tokenProfitValue > _shareHolderThreshlod) {
//
//                                                          _tOwned[uniswapV2PairUsdt]    =
_tOwned[uniswapV2PairUsdt].add(_tOwned[address(this)]);
//                              _tOwned[address(this)] = 0;
//                                              emit  Transfer(address(this),  uniswapV2PairUsdt,
liquidRewardAmount);
//                          }
//                      }
//                      else {
//                          uint256 liquidAmount = fee.mul(liquidRate).div(13);
//                          if (address(callback) != address(0)) {
//                                                              _tOwned[address(callback)]    =
_tOwned[address(callback)].add(liquidAmount);
//                                      emit Transfer(from, address(callback), liquidAmount);
//
//                                 } else {
//                                                                          _tOwned[foundationAddress]   =
_tOwned[foundationAddress].add(liquidAmount);
//                                      emit Transfer(from, foundationAddress, liquidAmount);
//                                 }
//                            }


                           _tOwned[foundationAddress]                                                 =
_tOwned[foundationAddress].add(foundationAmount);
                           emit Transfer(from, foundationAddress, foundationAmount);


                      } else {
                           fee = 0;
                      }



                      // enough    liquid
                      if          (IBEP20(uniswapV2PairUsdt).totalSupply()          >          0          &&
balanceOf(uniswapV2PairUsdt) > 10 * 10 ** 18) {
//                                        if  (!uniswapV2PairList[from]  &&  !uniswapV2PairList[to]  &&
balanceOf(address(callback)) > 0 && address(callback) != address(0)) {
//                                 BFAVTokenCallbackSinglePool(address(callback)).swapAndLiquify();
//                            }

                           uint256 newPrice = getNewPrice();

                           uint256 priceDownRate = getPriceDownRate(newPrice);
                           priceDownAmount = amount.mul(uint256(priceDownRate)).div(
                                100
                           );
                           if (priceDownAmount > 0) {
//                                                                        _tOwned[uniswapV2PairUsdt]   =
_tOwned[uniswapV2PairUsdt].add(priceDownAmount);
//                                          emit Transfer(from, uniswapV2PairUsdt, priceDownAmount);
                                        _tOwned[foundationAddress]                                       =
_tOwned[foundationAddress].add(priceDownAmount);
                                        emit Transfer(from, foundationAddress, priceDownAmount);
                                }
//                                  addPriceToCurrentDay(block.timestamp, newPrice);
                            }

                    }

            }

            uint acceptAmount = amount - fee - priceDownAmount;

            _tOwned[from] = _tOwned[from].sub(amount);
            _tOwned[to] = _tOwned[to].add(acceptAmount);

            uint256 newPrice = getNewPrice();
            if(newPrice>0){
                addPriceToCurrentDay(block.timestamp, newPrice);
            }

            emit Transfer(from, to, acceptAmount);
    }
    function addMemberInviter(address _inviter, address child) private {    //From as the invitee and
to as the invitee

            //require(child != _inviter, 'cannot invite yourself');
            address parent = inviter[child]; //Get the invitee of child
            if (parent == address(0) && _inviter != child) {    //If the address is not zero and_ If the
inviter is not its invited subordinate, it will enter the if statement
                inviter[child] = _inviter; //Update the invitee relationship of child
                userInviteNumber[_inviter] = userInviteNumber[_inviter].add(1);    //Update the number
of people invited
```

```
            if (memberInviter[_inviter].length >= inviterLength) { //Check whether the number of invitees exceeds the maximum limit. The maximum limit here is 200. There is a self DOS risk and a pyramid scheme. It is recommended to reduce it to about 10
                delete memberInviter[_inviter][0]; //If yes, delete the first address
                for (uint256 i = 0; i < memberInviter[_inviter].length - 1; i++) { //Then move in sequence
                    memberInviter[_inviter][i] = memberInviter[_inviter][i + 1];
                }

                memberInviter[_inviter].pop(); //Out of stack
            }
            memberInviter[_inviter].push(child); //Stack in

        }
    }
    function calculateFee(uint256 _amount) public view returns (uint256) {
        return _amount.mul(uint256(feeRate)).div(
            10 ** 2
        );
    }
    function _assignInviteReward(address from, address to, uint256 fee) private returns (uint256){
        uint256 inviteRelationAmount = fee.mul(inviteRelationRate).div(13);   //Previous generation 1%
        uint256 parentOneRelationAmount = fee.mul(parentOneRelationRate).div(13); //Next generation 2%
        uint256 parentTwoRelationAmount = fee.mul(parentTwoRelationRate).div(13); //The second generation 2%
        uint256 parentThreeRelationAmount = fee.mul(parentThreeRelationRate).div(13);//The next three generations 2%

        uint256 totalInviteReward = inviteRelationAmount.add(parentOneRelationAmount)
        .add(parentTwoRelationAmount).add(parentThreeRelationAmount); //Calculate the total amount of invitation rewards

        uint256 inviteRewardAssigned = 0;
        address parentOne = address(0); //Next generation initialization to zero address
        if (uniswapV2PairList[from]) {
            parentOne = inviter[to];
        } else {
            parentOne = inviter[from];
        }
        if (parentOne != address(0) && getLpPriceByAddress(parentOne) >= _inviteThreshlod &&
userInviteNumber[parentOne]>=1) { //calculation LP

            _tOwned[parentOne] = _tOwned[parentOne].add(parentOneRelationAmount);
            inviteRewardAssigned = inviteRewardAssigned.add(parentOneRelationAmount);
            emit Transfer(from, parentOne, parentOneRelationAmount);
        }
        address parentTwo = inviter[parentOne];
        if (parentTwo != address(0) && getLpPriceByAddress(parentTwo) >= _inviteThreshlod &&
userInviteNumber[parentTwo]>=2) {
            _tOwned[parentTwo] = _tOwned[parentTwo].add(parentTwoRelationAmount);
            inviteRewardAssigned = inviteRewardAssigned.add(parentTwoRelationAmount);
            emit Transfer(from, parentTwo, parentTwoRelationAmount);
        }

        address parentThree = inviter[parentTwo];
        if (parentThree != address(0) && getLpPriceByAddress(parentThree) >= _inviteThreshlod
&&    userInviteNumber[parentThree]>=3) {
            _tOwned[parentThree] = _tOwned[parentThree].add(parentThreeRelationAmount);
            inviteRewardAssigned = inviteRewardAssigned.add(parentThreeRelationAmount);
            emit Transfer(from, parentThree, parentThreeRelationAmount);
        }

        address memberAddress = from;
        if (uniswapV2PairList[from]) {
            memberAddress = to;
            //buy
        }

        address memberTmp = address(0);
        if (memberInviter[memberAddress].length > 0) {
            uint   random   =   rand(memberInviter[memberAddress].length);   //Random   number
predictability
            memberTmp = memberInviter[memberAddress][random];

        }
        if (memberTmp != address(0) && getLpPriceByAddress(memberTmp) >= _inviteThreshlod)
{

            _tOwned[memberTmp] = _tOwned[memberTmp].add(inviteRelationAmount);
            inviteRewardAssigned = inviteRewardAssigned.add(inviteRelationAmount);
            emit Transfer(from, memberTmp, inviteRelationAmount);
        }

        return totalInviteReward.sub(inviteRewardAssigned);
    }

}
    function getLpPriceByAddress(address user) public view returns (uint256 newPrice){
        if (user == address(0)) { //Address check
            return 0;
        }
        uint256 lpAmount = IBEP20(uniswapV2PairUsdt).balanceOf(user);

        uint256 lpPrice = getLpPrice(); //Calculate current price
        newPrice                =                lpPrice.mul(lpAmount).div(10                **
uint256(IBEP20(uniswapV2PairUsdt).decimals())); //Get the latest price
    }
    function getLpPrice() public view returns (uint256 newPrice){
        if (IBEP20(uniswapV2PairUsdt).totalSupply() == 0)
        {
            return 0;
        }
        uint256 usdtValue = IBEP20(usdtAddress).balanceOf(uniswapV2PairUsdt);
        if (IBEP20(uniswapV2PairUsdt).totalSupply() > 0) {
            newPrice                =                usdtValue.mul(2).mul(10                **
uint256(IBEP20(uniswapV2PairUsdt).decimals())).div(IBEP20(uniswapV2PairUsdt).totalSupply());
```

CHAINLION

```
        }
    }
    function getNewPrice() public view returns (uint256 newPrice){
        if (IBEP20(uniswapV2PairUsdt).totalSupply() > 0 && balanceOf(uniswapV2PairUsdt) > 10
* 10 ** 18) {
            address[] memory t = new address[](2);

            t[0] = address(this);
            t[1] = usdtAddress;

            uint256[] memory amounts = router.getAmountsOut(1 * (10 ** uint256(_decimals)), t);
            newPrice = amounts[1];
        }
    }
    function getPriceDownRate(uint256 newPrice) public view returns (uint256 downRate) {

        if (newPrice < lastBlockPrice) {
            uint256 priceDiff = lastBlockPrice.sub(newPrice
            );
            if (priceDiff > 0 && lastBlockPrice > 0) {
                uint256 diffRate = priceDiff.mul(100).div(lastBlockPrice);
                //For every 10% drop in price, the selling slide increases by 5% Up to 25%
                if (diffRate > 10) {
                    downRate = 5;
                }
                if (diffRate > 20) {
                    downRate = 10;
                }
                if (diffRate > 30) {
                    downRate = 15;
                }
                if (diffRate > 40) {
                    downRate = 20;
                }
                if (diffRate > 50) {
                    downRate = 25;
                }
            }
        }
    }
    uint256    public lastBlockDay = 0;
    uint256 public lastBlockPrice = 0;
    uint256    onlineBurnedAmount = 0;
    // let time = BigInt.fromI32(portfolio.timstamp.toI32()/86400);
    function addPriceToCurrentDay(uint256 blockTime, uint256 newPrice) private {
        uint256 blockDay = blockTime.div(86400);
        if (blockDay > lastBlockDay) {
            lastBlockPrice = newPrice;
            lastBlockDay = blockDay;
        } else {
            //equals
            if (newPrice > lastBlockPrice) {
                lastBlockPrice = newPrice;
            }
        }
    }
    function rand(uint256 _length) public view returns (uint256) {
        uint256 random = uint256(keccak256(abi.encodePacked(block.difficulty, now)));    //Random
number predictability
        return random % _length;
    }
```

**Safety advice： The parameters on the blockchain are not used as the parameter values generated by random numbers, and the maximum number of invitation relationships is limited. On the one hand, the DOS risk caused by the for cycle is avoided, on the other hand, the pyramid selling nature is avoided.**

### 8.3.5. Approve authorization transfer logic 【security】

**Audit description：** Conduct a security audit on the transfer business logic in the contract, check whether there is a risk of plastic overflow, and whether the transfer logic design is reasonable, etc.

**Audit results：** The transfer business logic in the contract checks whether the number of invitees exceeds the maximum limit. Currently, the maximum limit in the contract is 200. There is a certain self DOS risk when using the for loop traversal. At the same time, there is a pyramid selling nature. It is recommended to reduce it to about 10. At the same time, the parameter block on the block chain is used Difficulty, as a random number generation parameter, has the risk of being predicted.

**Code file：** BFAV.sol 640~660

**Code information：**

```
    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(msg.sender, spender, amount); //Authorized transfer
        return true;
    }
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool)
{ //Additional authorization limit
```

```
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(msg.sender,    spender,    _allowances[msg.sender][spender].sub(subtractedValue,
"ERC20: decreased allowance below zero"));//Reduce authorization limit
        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 limit
        emit Approval(owner, spender, amount);
        }
```

**Safety advice： NONE.**

### 8.3.6. Transferfrom business 【Moderate risk】

**Audit description：** Conduct a security audit on the transferfrom transfer business logic in the contract, check whether there is a risk of plastic overflow, and whether the transfer logic design is reasonable, etc.

**Audit results：** The transfer business logic in the contract checks whether the number of invitees exceeds the maximum limit. Currently, the maximum limit in the contract is 200. There is a certain self DOS risk when using the for loop traversal. At the same time, there is the nature of pyramid selling (only from the maximum invitable level, there are relevant restrictions on rewards, but there is no such feature). It is recommended to reduce it to about 10, and the parameter block on the blockchain is used Difficulty, as a random number generation parameter, has the risk of being predicted.

**Code file:** BFAV.sol 645~650

**Code information:**

```
    function transferFrom(address sender, address recipient, uint256 amount) public override returns
(bool) {
            _transfer(sender, recipient, amount); //Transfer (see 8.3.4 for specific question code)
            address msgSender = msg.sender;
            _approve(sender,    msgSender,    _allowances[sender][msgSender].sub(amount,    "ERC20:
transfer amount exceeds allowance")); //Update authorization limit
            return true;
    }
```

**Safety advice: The parameters on the blockchain are not used as the parameter values generated by random numbers, and the maximum number of invitation relationships is limited. On the one hand, the DOS risk caused by the for cycle is avoided, on the other hand, the pyramid selling nature is avoided.**

## 9. Contract source code

```
pragma solidity 0.6.12;

import './SafeMath.sol';
import './IBEP20.sol';
import './Ownable.sol';

interface IUniswapV2Factory {
        function createPair(address tokenA, address tokenB) external returns (address pair);
}

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

        function name() external pure returns (string memory);

        function symbol() external pure returns (string memory);

        function decimals() external pure returns (uint8);

        function totalSupply() external view returns (uint);

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

        function allowance(address owner, address spender) external view returns (uint);

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

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

        function transferFrom(address from, address to, uint value) external returns (bool);

        function DOMAIN_SEPARATOR() external view returns (bytes32);

        function PERMIT_TYPEHASH() external pure returns (bytes32);

        function nonces(address owner) external view returns (uint);

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

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

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

    function MINIMUM_LIQUIDITY() external pure returns (uint);

    function factory() external view returns (address);

    function token0() external view returns (address);

    function token1() external view returns (address);

    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32
blockTimestampLast);

    function price0CumulativeLast() external view returns (uint);

    function price1CumulativeLast() external view returns (uint);

    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);

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

    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;

    function skim(address to) external;

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

// pragma solidity >=0.6.2;

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

    function WDCC() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);

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

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

    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);

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

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

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

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

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

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

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

    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint
deadline)
    external
    payable
    returns (uint[] memory amounts);
    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);

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

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

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

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



// pragma solidity >=0.6.2;

interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountETH);

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

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

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

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
}
//
//contract BFAVTokenCallbackSinglePool is Ownable {
//      using SafeMath for uint256;
//
//      IUniswapV2Router02 public router;
//      address public cfTokenAddress;
//      address public toAddress;
//      address public usdtAddress;
//      address public uniswapV2PairUsdt;
```

```
//
//      uint256    _backLiquidThreshlod; //100u
//
//      bool inSwapAndLiquify;
//      int public locki = 0;
//      modifier lockTheSwap() {
//          inSwapAndLiquify = true;
//          locki = locki + 1;
//          _;
//          inSwapAndLiquify = false;
//      }
//
//      constructor (
//          address _router,
//          address _cfToken,
//          address _usdtAddress,
//          address _toAddress,
//          address _uniswapV2PairUsdt
//      ) public {
//          router = IUniswapV2Router02(_router);
//          cfTokenAddress = _cfToken;
//          usdtAddress = _usdtAddress;
//          toAddress = _toAddress;
//          uniswapV2PairUsdt = _uniswapV2PairUsdt;
//           _backLiquidThreshlod = 87 * (10 ** uint256(IBEP20(usdtAddress).decimals()));
////          _backLiquidThreshlod = 1 * (10 ** uint256(IBEP20(usdtAddress).decimals()));
//      }
//
//
//      function getNewPrice() public view returns (uint256 newPrice){
//                      if    (IBEP20(uniswapV2PairUsdt).totalSupply()    >    0    &&
IBEP20(usdtAddress).balanceOf(uniswapV2PairUsdt) > 10 * 10 ** 18) {
//              address[] memory t = new address[](2);
//
//              t[0] = cfTokenAddress;
//              t[1] = usdtAddress;
```

CHAINLION

```
//
//                          uint256[]   memory   amounts   =   router.getAmountsOut(1   *   (10   **
uint256(IBEP20(cfTokenAddress).decimals())), t);
//              newPrice = amounts[1];
//          }
//      }
//
//      function setOnAddLiquid(bool switchLiquid) public onlyOwner {
//          liquidSwitch = switchLiquid;
//
//      }
//
//      function setOnSwap(bool _swapSwitch) public onlyOwner {
//          swapSwitch = _swapSwitch;
//
//      }
//
//      bool public    liquidSwitch = true;
//      bool public    swapSwitch = true;
//      //      uint256 public halfAmountTmp    ;
//      //      uint256 public otherHalfAmountTmp    ;
//
//      function swapAndLiquify() public {
//          if (!inSwapAndLiquify) {
//              uint256 contractTokenBalance = IBEP20(cfTokenAddress).balanceOf(address(this));
//
//              uint256 newPrice = getNewPrice();
//
//              uint256 tokenUsdtValue = contractTokenBalance.mul(newPrice);
//
//              // split the contract balance into halves
//              if (tokenUsdtValue > _backLiquidThreshlod) {
//
//
//                  // split the contract balance into halves
//                  uint256 half = contractTokenBalance.div(2);
//                     uint256 otherHalf = contractTokenBalance.sub(half);
//                     if (swapSwitch) {
//                                                                  uint256    initialBalanceToken0    =
IBEP20(usdtAddress).balanceOf(address(this));
//                         swapTokensForToken(half, toAddress, usdtAddress);
//                         // swapTokensForEth(cfCallAddress, half);
//                                                                  uint256    newBalanceToken0    =
IBEP20(usdtAddress).balanceOf(address(this));
//                         half = newBalanceToken0.sub(initialBalanceToken0);
//
//                         //                    halfAmountTmp = half;
//                         //                    otherHalfAmountTmp = otherHalf;
//                     } else {
//                         half = IBEP20(usdtAddress).balanceOf(address(this));
//                         otherHalf = IBEP20(cfTokenAddress).balanceOf(address(this));
//                     }
//
//                     if (otherHalf > 0 && half > 0 && liquidSwitch) {
//                         // add liquidity to uniswap
//                               addLiquidity(address(cfTokenAddress), otherHalf, usdtAddress, half,
toAddress);
//                     }
//                 }
//             }
//         }
//
//
//         function swapTokensForToken(
//             uint256 tokenAmount,
//             address to,
//             address usdtAddress
//         ) private lockTheSwap {
//             // generate the uniswap pair path of token -> weth
//             address[] memory path = new address[](2);
//             path[0] = address(cfTokenAddress);
//             path[1] = usdtAddress;
//            IBEP20(address(cfTokenAddress)).approve(address(router), tokenAmount);
//            // make the swap
//            router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
//                tokenAmount,
//                0, // accept any amount of ETH
//                path,
//                address(this),
//                block.timestamp
//            );
//        }
//
//        function addLiquidity(
//            address token0,
//            uint256 token0Amount,
//            address token1,
//            uint256 token1Amount,
//            address to
//        ) private lockTheSwap {
//            // approve token transfer to cover all possible scenarios
//            IBEP20(token0).approve(address(router), token0Amount);
//            IBEP20(token1).approve(address(router), token1Amount);
//
//            // add the liquidity
//            router.addLiquidity(
//                token0,
//                token1,
//                token0Amount,
//                token1Amount,
//                0, // slippage is unavoidable
//                0, // slippage is unavoidable
//                address(1),
//                block.timestamp
//            );
//        }
//
//        function setToAddress(address _toAddress) external onlyOwner {
```

```
//              toAddress = _toAddress;
//        }
//
//        function transferToken(address token, address to) public onlyOwner {
//              require(token != address(0), 'CallBack::transferToken::TOKEN_ZERO_ADDRESS');
//              require(to != address(0), 'CallBack::transferToken::TO_ZERO_ADDRESS');
//              uint256 newBalanceToken0 = IBEP20(token).balanceOf(address(this));
//
//              IBEP20(token).transfer(to, newBalanceToken0);
//        }
//
//
//}

contract BFAVToken is IBEP20 {
      using SafeMath for uint256;

      mapping(address => uint256) internal _tOwned;
      mapping(address => mapping(address => uint256)) internal _allowances;

      string internal _name;
      string internal _symbol;
      uint8 internal _decimals;

      uint256 internal _tTotal;

      address public _owner;
      address public foundationAddress = 0xa9056272Ca777a63ae3A275d7aab078fd90A1691;
      address public feeAddress = 0xF8f21e8CE19099399C7A15Bd205e87C8B571bd6E;
      uint public feeRate = 13;

      uint public foundationRate = 4;    //dao    总共 13%
      uint public inviteRelationRate = 1;
      uint public parentOneRelationRate = 2;
      uint public parentTwoRelationRate = 2;
      uint public parentThreeRelationRate = 2;
```

```
    uint public blackHoleRate = 2;
//      uint public liquidRate = 5;
//      uint public liquidRewardRate = 5;

    address public uniswapV2PairUsdt;

    uint256 public _supply = 90000;

    address burnAddress = address(0);
    mapping(address => bool) public blackList;
    mapping(address => bool) public whiteList;

    mapping(address => bool) public thousandWhiteList;

    mapping(address => bool) public uniswapV2PairList;
    bool public useWhiteListSwith = true;

    address public    callback;
//      BFAVTokenCallbackSinglePool bfavTokenCallbackSinglePool;
    IUniswapV2Router02 public router;
    address public usdtAddress;

    uint256 inviterLength = 200;

    mapping(address => address[]) public memberInviter;
    mapping(address => address) public inviter;
    mapping(address => uint256 ) userInviteNumber;

    uint256 sellBuyBlockDiff = 3; //5 个区块以内机器人
    uint256 sellBuyAmountDiffRate = 2; //数量差别再 2%内

    uint256 internal _minSupply;
    uint256 _burnedAmount;

    uint256    _inviteThreshlod;
    uint256    _shareHolderThreshlod;
```

```
struct SellBuyBlock {
        uint256 blockNumber;
        uint256 amount;
}

mapping(address => SellBuyBlock) public    lastSellBuyBlockMap;

event blackUser(address indexed from, address indexed to, uint value);

modifier onlyOwner() {
        require(msg.sender == _owner, "admin: wut?");
        _;
}

//上线销毁 20742+7410=28152 枚
//最后销毁至 3000 枚。
//第一次产出 35000 枚

constructor (
        address _usdtAddress,
        address _router
) public {
        router = IUniswapV2Router02(_router);

        usdtAddress = _usdtAddress;
        _decimals = 18;
        _tTotal = _supply * (10 ** uint256(_decimals));
        _name = "BFAV";
        _symbol = "BFAV";

        uint256 onlineBurnedAmount = 28152 * (10 ** uint256(_decimals));
        emit Transfer(address(this), address(0), onlineBurnedAmount);

        _tOwned[msg.sender] = _tTotal.sub(onlineBurnedAmount);
        emit Transfer(address(0), msg.sender, _tOwned[msg.sender]);

        _burnedAmount = _burnedAmount.add(onlineBurnedAmount);

        _minSupply = 3000 * (10 ** uint256(decimals()));

        //TODO
         _inviteThreshlod = 87 * (10 ** uint256(IBEP20(usdtAddress).decimals()));
         _shareHolderThreshlod = 1000 * (10 ** uint256(IBEP20(usdtAddress).decimals()));

//          _inviteThreshlod = 1 * (10 ** uint256(IBEP20(usdtAddress).decimals()));
//          _shareHolderThreshlod = 1 * (10 ** uint256(IBEP20(usdtAddress).decimals()));

        uniswapV2PairUsdt = IUniswapV2Factory(router.factory())
        .createPair(address(this), usdtAddress);

        uniswapV2PairList[uniswapV2PairUsdt] = true;



        _owner = msg.sender;
        whiteList[_owner] = true;
//              bfavTokenCallbackSinglePool = new BFAVTokenCallbackSinglePool(address(router),
address(this), usdtAddress, feeAddress, uniswapV2PairUsdt);
//          callback = address(bfavTokenCallbackSinglePool);
//          thousandWhiteList[callback] = true;
    }

    function minSupply() public view returns (uint256) {
        return _minSupply;
    }

    function transferOwner(address newOwner) external onlyOwner {
        _owner = newOwner;
    }

//      function setBFAVTokenCallback(address _bfavTokenCallback) external onlyOwner {
```

```
//              callback = _bfavTokenCallback;
//              thousandWhiteList[_bfavTokenCallback] = true;
//        }

    function setUniswapPairList(address pairAddress, bool isPair) external onlyOwner {
        uniswapV2PairList[pairAddress] = isPair;
    }

    function setBlackList(address userAddress, bool isBlock) external onlyOwner {
        blackList[userAddress] = isBlock;
    }

    function setThousandWhiteList(address userAddress, bool isWhite) external onlyOwner {
        thousandWhiteList[userAddress] = isWhite;
    }

    function setSellBuyBlockDiff(uint _sellBuyBlockDiff) external onlyOwner {
        sellBuyBlockDiff = _sellBuyBlockDiff;
    }

    function setSellBuyAmountDiffRate(uint _sellBuyAmountDiffRate) external onlyOwner {
        sellBuyAmountDiffRate = _sellBuyAmountDiffRate;
    }

    function setFeeRate(uint _feeRate) external onlyOwner {
        feeRate = _feeRate;
    }

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

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

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

// function setFeeAddress(address _feeAddress) external onlyOwner {
//         feeAddress = _feeAddress;
//         bfavTokenCallbackSinglePool.setToAddress(feeAddress);
// }

// function setOnAddLiquid(bool switchLiquid) public onlyOwner {
//         bfavTokenCallbackSinglePool.setOnAddLiquid(switchLiquid);

// }

// function setOnSwap(bool _swapSwitch) public onlyOwner {
//         bfavTokenCallbackSinglePool.setOnSwap(_swapSwitch);

// }

// function transferToken(address token, address to) external onlyOwner {
//         bfavTokenCallbackSinglePool.transferToken(token, to);
// }

function setFoundationAddress(address _foundationAddress) external onlyOwner {
        foundationAddress = _foundationAddress;
}

function burnedAmount() public view returns (uint256) {
        return _burnedAmount;
}

function setFoundationRate(uint _foundationRate) external onlyOwner {
        foundationRate = _foundationRate;
}

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

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

function decimals() public override view returns (uint8) {
        return _decimals;
}

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



function getOwner() public view override returns (address){
        return _owner;
}

function balanceOf(address account) public view override returns (uint256) {
        return _tOwned[account];
}

function transfer(address recipient, uint256 amount) public override returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
}

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

function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(msg.sender, spender, amount);
        return true;
}

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

function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue));
        return true;
}

function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(subtractedValue,
"ERC20: decreased allowance below zero"));
        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 calculateFee(uint256 _amount) public view returns (uint256) {
        return _amount.mul(uint256(feeRate)).div(
            10 ** 2
        );
}
```

```
function addMemberInviter(address _inviter, address child) private {

    //            require(child != _inviter, 'cannot invite yourself');
    address parent = inviter[child];
    if (parent == address(0) && _inviter != child) {
        inviter[child] = _inviter;
        userInviteNumber[_inviter] = userInviteNumber[_inviter].add(1);

        if (memberInviter[_inviter].length >= inviterLength) {
            delete memberInviter[_inviter][0];
            for (uint256 i = 0; i < memberInviter[_inviter].length - 1; i++) {
                memberInviter[_inviter][i] = memberInviter[_inviter][i + 1];
            }

            memberInviter[_inviter].pop();
        }
        memberInviter[_inviter].push(child);

    }
}

uint256   public lastBlockDay = 0;
uint256 public lastBlockPrice = 0;
uint256   onlineBurnedAmount = 0;
// let time = BigInt.fromI32(portfolio.timstamp.toI32()/86400);
function addPriceToCurrentDay(uint256 blockTime, uint256 newPrice) private {
    uint256 blockDay = blockTime.div(86400);
    if (blockDay > lastBlockDay) {
        lastBlockPrice = newPrice;
        lastBlockDay = blockDay;
    } else {
        //equals
        if (newPrice > lastBlockPrice) {
            lastBlockPrice = newPrice;
        }
```

```
            }
        }

        function getLpPrice() public view returns (uint256 newPrice){
            if (IBEP20(uniswapV2PairUsdt).totalSupply() == 0)
            {
                return 0;
            }
            uint256 usdtValue = IBEP20(usdtAddress).balanceOf(uniswapV2PairUsdt);
            if (IBEP20(uniswapV2PairUsdt).totalSupply() > 0) {
                newPrice          =          usdtValue.mul(2).mul(10          **
uint256(IBEP20(uniswapV2PairUsdt).decimals())).div(IBEP20(uniswapV2PairUsdt).totalSupply());
            }
        }

        function getLpPriceByAddress(address user) public view returns (uint256 newPrice){
            if (user == address(0)) {
                return 0;
            }
            uint256 lpAmount = IBEP20(uniswapV2PairUsdt).balanceOf(user);

            uint256 lpPrice = getLpPrice();
            newPrice          =          lpPrice.mul(lpAmount).div(10          **
uint256(IBEP20(uniswapV2PairUsdt).decimals()));
        }

        function getNewPrice() public view returns (uint256 newPrice){
            if (IBEP20(uniswapV2PairUsdt).totalSupply() > 0 && balanceOf(uniswapV2PairUsdt) > 10 *
10 ** 18) {
                address[] memory t = new address[](2);

                t[0] = address(this);
                t[1] = usdtAddress;

                uint256[] memory amounts = router.getAmountsOut(1 * (10 ** uint256(_decimals)), t);
                newPrice = amounts[1];
        }
    }

    function getPriceDownRate(uint256 newPrice) public view returns (uint256 downRate) {

        if (newPrice < lastBlockPrice) {
            uint256 priceDiff = lastBlockPrice.sub(newPrice
            );
            if (priceDiff > 0 && lastBlockPrice > 0) {
                uint256 diffRate = priceDiff.mul(100).div(lastBlockPrice);
                //价格每下跌 10%，卖出的滑点增加 5%.最高到 25%
                if (diffRate > 10) {
                    downRate = 5;
                }
                if (diffRate > 20) {
                    downRate = 10;
                }
                if (diffRate > 30) {
                    downRate = 15;
                }
                if (diffRate > 40) {
                    downRate = 20;
                }
                if (diffRate > 50) {
                    downRate = 25;
                }
            }
        }

    }


    function rand(uint256 _length) public view returns (uint256) {
        uint256 random = uint256(keccak256(abi.encodePacked(block.difficulty, now)));
        return random % _length;
    }
    function _transfer(
        address from,
        address to,
        uint256 amount
    ) private {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(amount > 0, "Transfer amount must be greater than zero");
        uint256 leftAmount = _tTotal.sub(_burnedAmount);
        // if (uniswapV2PairList[to] || uniswapV2PairList[from]) {
        if (uniswapV2PairList[to] && thousandWhiteList[from] != true) {
            require(amount < leftAmount.div(1000), "Transfer amount must be less than
thousandth");
        }
        require(!blackList[from] && !blackList[to], "black transfer not allowed");

        uint256 priceDownAmount = 0;
        uint256 fee = 0;
        if (whiteList[from] != true) {

            if (!uniswapV2PairList[to] && !uniswapV2PairList[from]
            && from != address(this) && to != address(this)
            && from != address(router) && to != address(router)
            && from != address(callback) && to != address(callback)
            ) {
                uint256 toBalance = balanceOf(to);

                if (toBalance == 0) {
                    addMemberInviter(from, to);
                }
            }
            address sellBuyUser = address(0);
            if (uniswapV2PairList[from]) {
                sellBuyUser = to;
            }
            if (uniswapV2PairList[to]) {
                sellBuyUser = from;
            }

            if (sellBuyUser != address(this) && sellBuyUser != address(callback) &&
            sellBuyUser != address(0) && sellBuyUser != address(router)
            && !uniswapV2PairList[sellBuyUser]) {

                SellBuyBlock memory sellBuyBlock = lastSellBuyBlockMap[sellBuyUser];
                uint256 amountDiff = 0;
                if (amount > sellBuyBlock.amount) {
                    amountDiff = amount.sub(sellBuyBlock.amount);
                } else {
                    amountDiff = sellBuyBlock.amount.sub(amount);
                }
                if (sellBuyBlock.amount > 0) {
                    uint256 amountDiffRate = amountDiff.mul(100).div(sellBuyBlock.amount);

                    if (block.number.sub(sellBuyBlock.blockNumber) <= sellBuyBlockDiff &&
amountDiffRate <= sellBuyAmountDiffRate) {
                    blackList[sellBuyUser] = true;
                    emit blackUser(from, to, amount);
                    return;
                    }
                }
                lastSellBuyBlockMap[sellBuyUser] = SellBuyBlock({blockNumber : block.number,
amount : amount});
            }

            amount = amount.mul(999).div(1000);

            if (leftAmount > _minSupply && from != callback && to != callback) {

                fee = calculateFee(amount);
                if (fee > 0) {
```

CHAINLION

```
                    // address    uniswapV2Pair = from;

                    //                   uint public foundationRate = 4;    //dao
                    //                   uint public inviteRelationRate = 1;
                    //                   uint public parentOneRelationRate = 2;
                    //                   uint public parentTwoRelationRate = 2;
                    //                   uint public parentThreeRelationRate = 2;
                    //                   uint public blackHoleRate = 2;



                    //                   uint public liquidRate = 5;
                    //                   uint public liquidRewardRate = 5;
                    uint256 leftAmountSubFee = leftAmount.sub(fee);
                    if (leftAmountSubFee < _minSupply) {
                        fee = leftAmount.sub(_minSupply);
                    }


                    uint256 foundationAmount = fee.mul(foundationRate).div(13);

                    uint256 blackHoleAmount = fee.mul(blackHoleRate).div(13);

                    uint256 inviteRewardunAssigned = _assignInviteReward(from, to, fee);

                    _tOwned[burnAddress]                                            =
_tOwned[burnAddress].add(inviteRewardunAssigned);
                    _burnedAmount = _burnedAmount.add(inviteRewardunAssigned);
                    emit Transfer(from, burnAddress, inviteRewardunAssigned);

                    _tOwned[burnAddress] = _tOwned[burnAddress].add(blackHoleAmount);
                    _burnedAmount = _burnedAmount.add(blackHoleAmount);
                    emit Transfer(from, burnAddress, blackHoleAmount);

//                   uint256 newPrice = getNewPrice();
//                   if (uniswapV2PairList[to]) {//sell
//                              uint256 liquidRewardAmount = fee.mul(liquidRewardRate).div(13);
//
//                                                                              _tOwned[address(this)]   =
_tOwned[address(this)].add(liquidRewardAmount);
//                              emit Transfer(from, address(this), liquidRewardAmount);
//
//                              uint256 tokenProfitValue = newPrice.mul(_tOwned[address(this)]);
//                              if (tokenProfitValue > _shareHolderThreshlod) {
//
//                                                                         _tOwned[uniswapV2PairUsdt]   =
_tOwned[uniswapV2PairUsdt].add(_tOwned[address(this)]);
//                                   _tOwned[address(this)] = 0;
//                                                  emit  Transfer(address(this),  uniswapV2PairUsdt,
liquidRewardAmount);
//                              }
//                         }
//                         else {
//                              uint256 liquidAmount = fee.mul(liquidRate).div(13);
//                              if (address(callback) != address(0)) {
//                                                                           _tOwned[address(callback)]   =
_tOwned[address(callback)].add(liquidAmount);
//                                   emit Transfer(from, address(callback), liquidAmount);
//
//                              } else {
//                                                                         _tOwned[foundationAddress]   =
_tOwned[foundationAddress].add(liquidAmount);
//                                   emit Transfer(from, foundationAddress, liquidAmount);
//                              }
//                         }


                        _tOwned[foundationAddress]                                                    =
_tOwned[foundationAddress].add(foundationAmount);
                        emit Transfer(from, foundationAddress, foundationAmount);
```

```
                    } else {
                        fee = 0;
                    }


                    // enough    liquid
                    if           (IBEP20(uniswapV2PairUsdt).totalSupply()           >           0           &&
balanceOf(uniswapV2PairUsdt) > 10 * 10 ** 18) {
//                                        if  (!uniswapV2PairList[from]  &&  !uniswapV2PairList[to]  &&
balanceOf(address(callback)) > 0 && address(callback) != address(0)) {
//                                BFAVTokenCallbackSinglePool(address(callback)).swapAndLiquify();
//                           }

                        uint256 newPrice = getNewPrice();

                        uint256 priceDownRate = getPriceDownRate(newPrice);
                        priceDownAmount = amount.mul(uint256(priceDownRate)).div(
                            100
                        );
                        if (priceDownAmount > 0) {
//                                                                     _tOwned[uniswapV2PairUsdt]    =
_tOwned[uniswapV2PairUsdt].add(priceDownAmount);
//                                emit Transfer(from, uniswapV2PairUsdt, priceDownAmount);
                              _tOwned[foundationAddress]                                             =
_tOwned[foundationAddress].add(priceDownAmount);
                              emit Transfer(from, foundationAddress, priceDownAmount);
                        }
//                           addPriceToCurrentDay(block.timestamp, newPrice);
                    }


               }


          }

          uint acceptAmount = amount - fee - priceDownAmount;
```

```
        _tOwned[from] = _tOwned[from].sub(amount);
        _tOwned[to] = _tOwned[to].add(acceptAmount);

        uint256 newPrice = getNewPrice();
        if(newPrice>0){
            addPriceToCurrentDay(block.timestamp, newPrice);
        }

        emit Transfer(from, to, acceptAmount);
    }

    function _assignInviteReward(address from, address to, uint256 fee) private returns (uint256){
        uint256 inviteRelationAmount = fee.mul(inviteRelationRate).div(13);
        uint256 parentOneRelationAmount = fee.mul(parentOneRelationRate).div(13);
        uint256 parentTwoRelationAmount = fee.mul(parentTwoRelationRate).div(13);
        uint256 parentThreeRelationAmount = fee.mul(parentThreeRelationRate).div(13);

        uint256 totalInviteReward = inviteRelationAmount.add(parentOneRelationAmount)
        .add(parentTwoRelationAmount).add(parentThreeRelationAmount);

        uint256 inviteRewardAssigned = 0;
        address parentOne = address(0);
        if (uniswapV2PairList[from]) {
            parentOne = inviter[to];
        } else {
            parentOne = inviter[from];
        }
        if (parentOne != address(0) && getLpPriceByAddress(parentOne) >= _inviteThreshlod &&
userInviteNumber[parentOne]>=1) {

            _tOwned[parentOne] = _tOwned[parentOne].add(parentOneRelationAmount);
            inviteRewardAssigned = inviteRewardAssigned.add(parentOneRelationAmount);
            emit Transfer(from, parentOne, parentOneRelationAmount);
        }
        address parentTwo = inviter[parentOne];
        if (parentTwo != address(0) && getLpPriceByAddress(parentTwo) >= _inviteThreshlod &&
userInviteNumber[parentTwo]>=2) {
            _tOwned[parentTwo] = _tOwned[parentTwo].add(parentTwoRelationAmount);
            inviteRewardAssigned = inviteRewardAssigned.add(parentTwoRelationAmount);
            emit Transfer(from, parentTwo, parentTwoRelationAmount);
        }

        address parentThree = inviter[parentTwo];
        if (parentThree != address(0) && getLpPriceByAddress(parentThree) >= _inviteThreshlod &&
userInviteNumber[parentThree]>=3) {
            _tOwned[parentThree] = _tOwned[parentThree].add(parentThreeRelationAmount);
            inviteRewardAssigned = inviteRewardAssigned.add(parentThreeRelationAmount);
            emit Transfer(from, parentThree, parentThreeRelationAmount);
        }

        address memberAddress = from;
        if (uniswapV2PairList[from]) {
            memberAddress = to;
            //buy
        }

        address memberTmp = address(0);
        if (memberInviter[memberAddress].length > 0) {
            uint random = rand(memberInviter[memberAddress].length);
            memberTmp = memberInviter[memberAddress][random];



        }
        if (memberTmp != address(0) && getLpPriceByAddress(memberTmp) >= _inviteThreshlod) {

            _tOwned[memberTmp] = _tOwned[memberTmp].add(inviteRelationAmount);
            inviteRewardAssigned = inviteRewardAssigned.add(inviteRelationAmount);
            emit Transfer(from, memberTmp, inviteRelationAmount);
        }

        return totalInviteReward.sub(inviteRewardAssigned);
    }
```

```
}
```

## 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.

CHAINLION

Blockchain world patron saint building blockchain ecological security