Smart contract audit report

CHAINLION

NO.OC002206120001

JUNE 12, 2022

CHAINLION

CATALOGUE

CHAINLION

1. PROJECT SUMMARY

Entry type	Specific description
Entry name	Dawn
Project type	DEFI
Application platform	BSC
DawnToken	0x79684C659150A18BC7E8b69E206e80F48787bf13

2. AUDIT SUMMARY

Entry type	Specific description
Project cycle	JUNE/09/2022-JUNE/12/2022
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	1

Low risk vulnerability	4

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 **Dawn** 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.

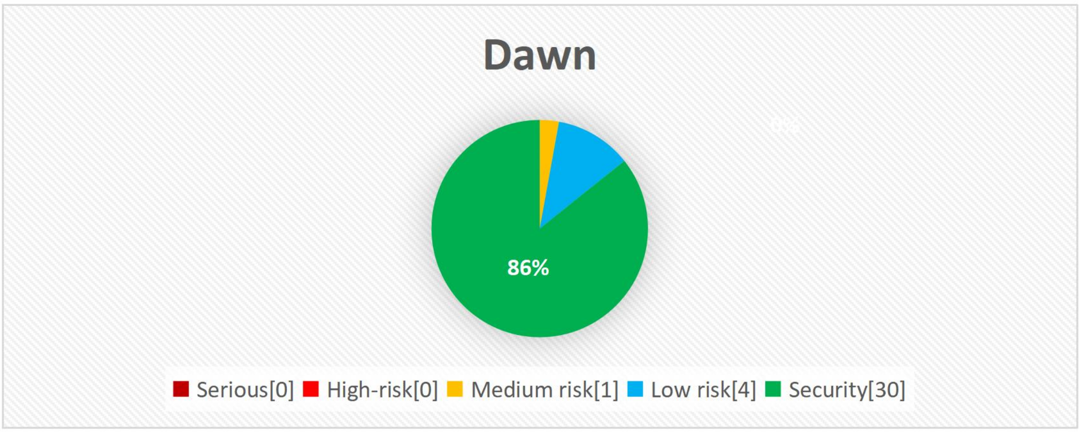
5. Directory structure

```
├─Dawn.sol
```

6. File hashes

Contract	SHA1 Checksum
Dawn.sol	F9F455149147C14D20AB6E39338A2702D3B3D47B

7. Vulnerability distribution



8. Audit content

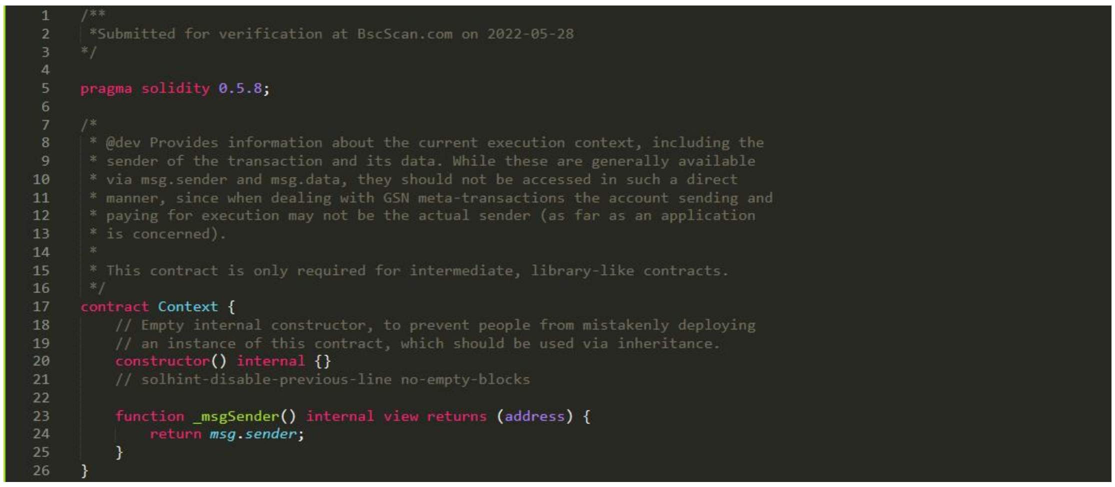
8.1. Coding specification

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.



```
/**
 *Submitted for verification at BscScan.com on 2022-05-28
*/

pragma solidity 0.5.8;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor() internal {}
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address) {
        return msg.sender;
    }
}
```

Safety advice：NONE.

8.1.2. Return value verification 【security】

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

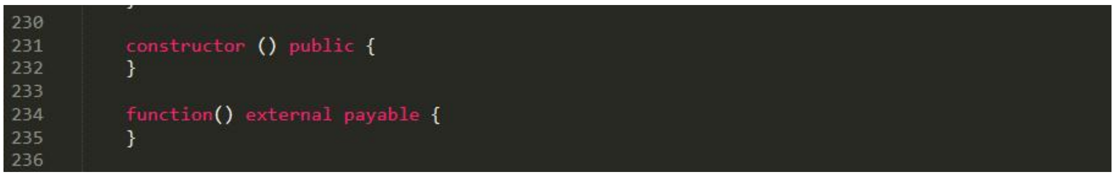
Audit results： According to the audit, there is no embedded function calling the official standards transfer and transferfrom in the smart contract, so there is no such security problem.

Safety advice： NONE.

8.1.3. Constructor writing 【security】

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

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



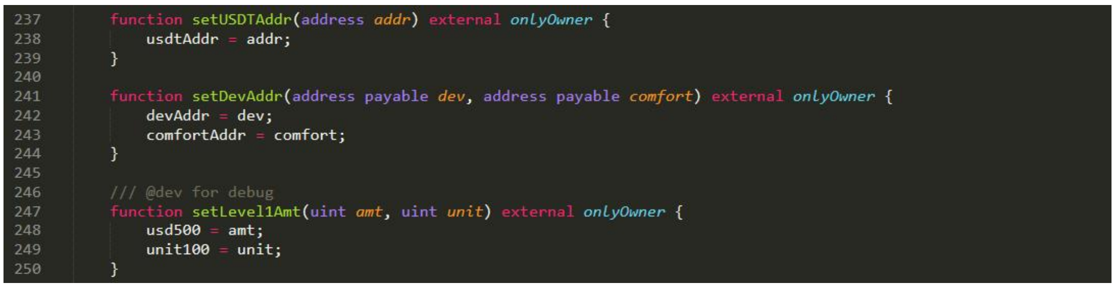
```
    constructor () public {
    }

    function() external payable {
    }
```

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.



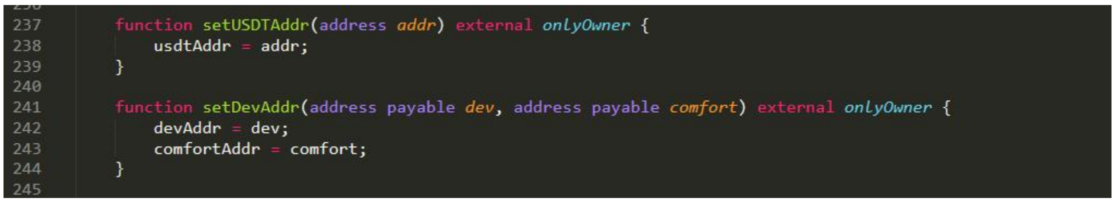
```
    function setUSDTAddr(address addr) external onlyOwner {
        usdtAddr = addr;
    }

    function setDevAddr(address payable dev, address payable comfort) external onlyOwner {
        devAddr = dev;
        comfortAddr = comfort;
    }

    /// @dev for debug
    function setLevel1Amt(uint amt, uint unit) external onlyOwner {
        usd500 = amt;
        unit100 = unit;
    }
```

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 setUSDTAddr(address addr) external onlyOwner {
        usdtAddr = addr;
    }

    function setDevAddr(address payable dev, address payable comfort) external onlyOwner {
        devAddr = dev;
        comfortAddr = comfort;
    }
```

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.

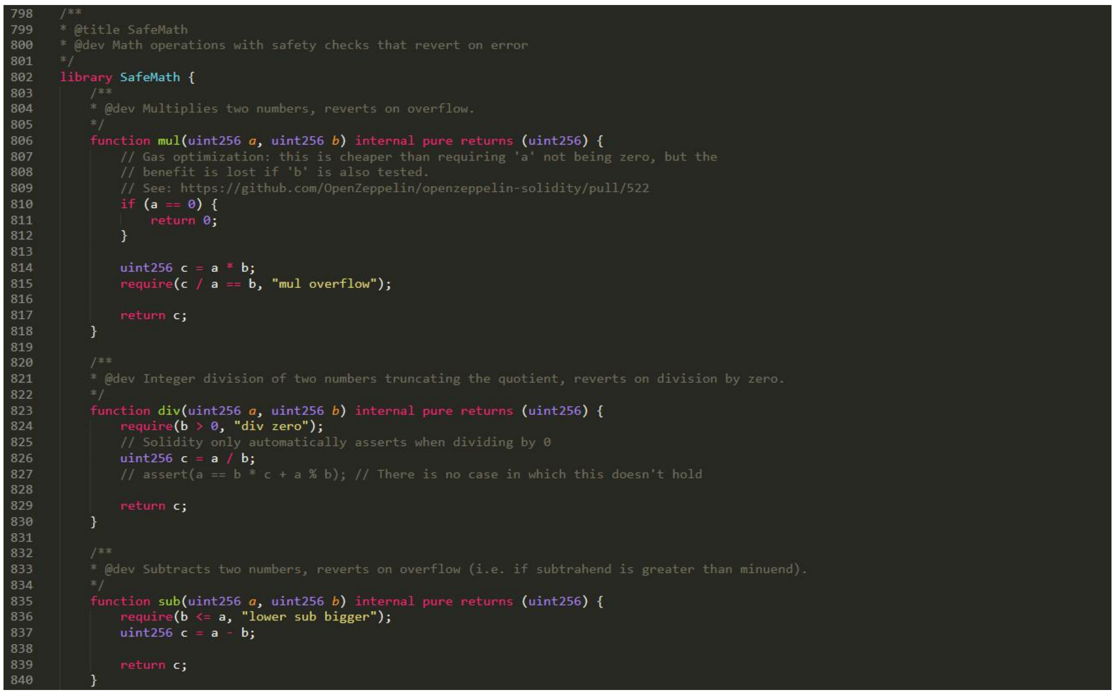
8.2. Coding design

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.



```
/**
* @title SafeMath
* @dev Math operations with safety checks that revert on error
*/
library SafeMath {
    /**
    * @dev Multiplies two numbers, reverts on 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-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "mul overflow");

        return c;
    }

    /**
    * @dev Integer division of two numbers truncating the quotient, reverts on division by zero.
    */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "div zero");
        // Solidity only automatically asserts when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
    * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "lower sub bigger");
        uint256 c = a - b;

        return c;
    }
```

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. Transferownership logic 【security】

Audit description： Conduct security audit on the business logic design of the transferownership contract owner change in the contract, and check whether the relevant business logic design is reasonable.

Audit results： The transferownership contract owner change function in the contract only allows the owner of the contract to call. At the same time, it checks the non-zero address of the new contract owner. The business logic design is correct.

Code file： Dawn.sol L66~69

Code information：

```
/**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
      require(newOwner != address(0), "Ownable: new owner is the zero address"); //Address
non-zero check
     _owner = newOwner; //Update owner address
    }
    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner"); //The caller address must be the
owner address
        _;
    }
```

Safety advice： NONE.

8.3.2. Role management related logic design 【security】

Audit description： Conduct security audit on the business logic design related to role management in the contract, check whether the relevant business logic design is reasonable, and whether the parameters are checked for legitimacy, etc.

Audit results： The business logic design related to role management in the contract is correct.

Code file： Dawn.sol L77~104

Code information：

```
    struct Role {
        mapping(address => bool) bearer;
    }
    /**
     * @dev Give an account access to this role.
     */
    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role"); //Check whether the account has been set with a role
        role.bearer[account] = true; //Set up roles
    }

    /**
     * @dev Remove an account's access to this role.
     */
    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role"); //Check whether the account is set with a role
        role.bearer[account] = false; //Remove role
    }

    /**
     * @dev Check if an account has this role.
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) { //Check whether the account is set with a role
        require(account != address(0), "Roles: account is the zero address"); //Address non-zero check
        return role.bearer[account]; //Retrieve whether to set roles
    }
```

Safety advice： NONE.

8.3.3. Business logic design of white list management 【security】

Audit results： Conduct security audit on the business logic design related to the white list management in the contract, check whether the relevant business logic design is reasonable, and whether the parameters are checked for legitimacy, etc.

Audit results: The business logic design related to the white list management in the contract is correct.

Code file: Dawn.sol L111~135

Code information:

```
    using Roles for Roles.Role;

    Roles.Role private _whitelistAdmins;

    constructor () internal {
    }

    modifier onlyWhitelistAdmin() {
        require(isWhitelistAdmin(_msgSender()) || isOwner(), "WhitelistAdminRole: caller does not have the WhitelistAdmin role"); //Caller permission check
        _;
    }

    function isWhitelistAdmin(address account) public view returns (bool) { //Check whether white list address
        return _whitelistAdmins.has(account) || isOwner();
    }

    function addWhitelistAdmin(address account) public onlyOwner { //Set the white list address, which can only be called by the contract owner
        _whitelistAdmins.add(account);
    }

    function removeWhitelistAdmin(address account) public onlyOwner { //Remove the whitelist address, which can only be called by the contract owner
        _whitelistAdmins.remove(account);
    }
```

Safety advice： NONE.

8.3.4. Global variable initialization design 【Low risk】

Audit description： Conduct security audit on the business logic design of global variable initialization in the contract, and check whether the relevant business logic is reasonable.

Audit results： There are differences in variable initialization in the contract. According to the context format and notes, the USD500 here should be 500*(10**18). Please check whether the relevant design meets the expectations.

Code file： Dawn.sol L152

Code information：

```
  // uint ethWei = 1 ether;
    uint unit100        = 100 * (10 ** 18);
    uint usd500          = 100 * (10 ** 18);    //The combination context seems to be inconsistent with
the variable name, so self inspection is required
    uint usd1000        = 1000 * (10 ** 18);
    uint usd1100        = 1100 * (10 ** 18);
    uint usd2000        = 2000 * (10 ** 18);
    uint usd2100        = 2100 * (10 ** 18);
    uint usd3000        = 3000 * (10 ** 18);
    uint usd20000       = 20000 * (10 ** 18);

    address    payable    private    devAddr    =
address(0x0Ee152aB24453F32A5A3585AF68002FA87dC38eF);
    address    payable    private    comfortAddr    =
address(0x8170FC3FCb38468a28b04Cf497ffe1c12812f562);
    address public usdtAddr = address(0x55d398326f99059fF775485246999027B3197955); // bsc
usdt

    //Inconsistent with remarks
    uint bonuslimit = 3000 * (10 ** 18); // 15 ether;
    uint sendLimit = 20000 * (10 ** 18); // 100 ether;
    uint withdrawLimit = 3000 * (10 ** 18); // 15 ether;
```

Safety advice： The project party shall check whether the relevant design meets the expectation.

8.3.5. Contract authority concentration detection 【Moderate risk】

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

Audit results： In the contract, the owner has high authority and can set usdt address, dev address, levelamt value, project start and stop and other sensitive operations. It is recommended that the project party properly keep the owner authority.

Code file： Dawn.sol

Code information：

```
    function setUSDTAddr(address addr) external onlyOwner {
        usdtAddr = addr;
    }

    function setDevAddr(address payable dev, address payable comfort) external onlyOwner {
        devAddr = dev;
        comfortAddr = comfort;
    }

    /// @dev for debug
    function setLevel1Amt(uint amt, uint unit) external onlyOwner {
        usd500 = amt;
        unit100 = unit;
    }
    function verydangerous(uint time) external onlyOwner {
        require(canSetStartTime == 1, "verydangerous, limited!"); //Check if cansetstarttime is 1
        require(time > now, "no, verydangerous"); //Check whether the time is greater than now
        startTime = time; //Set starttime
        canSetStartTime = 0; //Update cansetstarttime
    }
    function setBonusInterval(uint interval) external onlyOwner {
        bonusInterval = interval;
    }
    function endRound() external onlyOwner { //Only owner calls are allowed
        require(usdtbalance() < usd500, "contract balance must be lower than 500 usdt"); //Condition
check
        rid++;
        startTime = now.add(period).div(1 days).mul(1 days);
        canSetStartTime = 1;
        }
```

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

8.3.6. Classification related business logic【security】

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

Audit results： There is no error in the design of business logic related to grading in the contract.

Code file： Dawn.sol L252~335

Code information：

```
function getLevel(uint value) public view returns (uint) {
        if (value >= usd500 && value <= usd1000) { //500~1000 Grade 1
            return 1;
        }
        if (value >= usd1100 && value <= usd2000) {//1000~2000 Grade 2
            return 2;
        }
        if (value >= usd2100 && value <= usd3000) { //2100~3000 Grade 3
            return 3;
        }
        return 0; //Level 0 for other conditions
    }

    function getNodeLevel(uint value) public view returns (uint) {
        if (value >= usd500 && value <= usd1000) {    //500~1000 Grade 1
            return 1;
        }
        if (value >= usd1100 && value <= usd2000) {//1000~2000 Grade 2
            return 2;
        }
        if (value >= usd2100) { //greater than2100 Grade 13
            return 3;
        }
        return 0;
    }

    function getScByLevel(uint level) public pure returns (uint) {
        if (level == 1) {
            return 4;
        }
        if (level == 2) {
            return 6;
        }
        if (level == 3) {
            return 10;
        }
        return 0;
    }

    function getFireScByLevel(uint level) public pure returns (uint) {
        if (level == 1) {
            return 3;
        }
        if (level == 2) {
            return 6;
        }
        if (level == 3) {
            return 10;
        }
        return 0;
    }

    function getRecommendScaleByLevelAndTim(uint level, uint times) public pure returns (uint){
        if (level == 1 && times == 1) {
            return 40;
        }
        if (level == 2 && times == 1) {
            return 60;
        }
        if (level == 2 && times == 2) {
            return 40;
        }
        if (level == 3) {
            if (times == 1) {
                return 90;
            }
            if (times == 2) {
                return 60;
            }
            if (times == 3) {
                return 40;
            }
```

```
                if (times >= 4 && times <= 10) {
                    return 8;
                }
                if (times >= 11 && times <= 20) {
                    return 4;
                }
                if (times >= 21) {
                    return 1;
                }
            }
            return 0;
        }
```

Safety advice： NONE.

8.3.7. Verydangeous business logic 【security】

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

Audit results： The verydangeous function in the contract can only be called by the owner of the contract, and the parameters and trigger conditions are strictly checked. The relevant business logic design is correct.

Code file： Dawn.sol L344~349

Code information：

```
    function verydangerous(uint time) external onlyOwner {
            require(canSetStartTime == 1, "verydangerous, limited!"); //Check if cansetstarttime is 1
            require(time > now, "no, verydangerous"); //Check whether the time is greater than now
            startTime = time; //Set starttime
            canSetStartTime = 0; //update canSetStartTime
        }
```

Safety advice： NONE.

8.3.8. Actalllimit business logic design 【security】

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

Audit results: The actalllimit function in the contract can only be called by the owner of the contract, and the parameters and trigger conditions are strictly checked. The relevant business logic design is correct.

Code file: Dawn.sol L363~369

Code information:

```
function actAllLimit(uint bonusLi, uint sendLi, uint withdrawLi) external onlyOwner {
            require(bonusLi >= usd3000 && sendLi >= usd20000 && withdrawLi >= usd3000, "invalid
amount");
            bonuslimit = bonusLi;
            sendLimit = sendLi;
            withdrawLimit = withdrawLi;
    }
```

Safety advice: **: NONE.**

8.3.9. Actuserstatus business logic 【security】

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

Audit results: The actuserstatus function in the contract can only be called by the admin in the white list, and the parameters and trigger conditions are strictly checked. The relevant business logic design is correct.

Code file: Dawn.sol L363~369

Code information：

```
function actUserStatus(address addr, uint status) external onlyWhitelistAdmin {
        require(status == 0 || status == 1 || status == 2, "bad parameter status");
        UserGlobal storage userGlobal = userMapping[addr];
        userGlobal.status = status;
    }
```

Safety advice： ： NONE.

8.3.10. TransferInvestAmount 【security】

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

Audit results： The business logic design of transferinvestamount function in the contract is correct.

Code file： Dawn.sol L381~385

Code information：

```
function transferInvestAmount(uint amt) private {
        require(amt >= usd500 && amt <= usd3000, "between 100 USDT and 3000 USDT");
        require(amt == amt.div(unit100).mul(unit100), "must be multor of 100 USDT");
        TRC20(usdtAddr).transferFrom(msg.sender, address(this), amt);
    }
```

Safety advice： NONE.

8.3.11. Logical design of deposit pledge 【security】

Audit description： Conduct security audit on the logic design of the deposit pledge business in the contract, and check whether the relevant business logic is reasonable.

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

Code file: Dawn.sol L391~442

Code information:

```
 function deposit(uint amt, string memory inviteCode, string memory beCode) public isHuman()
{   //The caller cannot be a contract
            require(donnotimitate(), "no, donnotimitate");
            // require(msg.value >= 1 * ethWei && msg.value <= 15 * ethWei, "between 1 and 15");
            // require(msg.value == msg.value.div(ethWei).mul(ethWei), "invalid msg value");
            transferInvestAmount(amt); //pledge

            UserGlobal storage userGlobal = userMapping[msg.sender]; //obtain UserGlobal Storage
            if (userGlobal.id == 0) {
                  require(!compareStr(inviteCode, "") && bytes(inviteCode).length == 6, "invalid invite
code"); //Validity check of invitation code
                  address beCodeAddr = addressMapping[beCode];
                  require(isUsed(beCode), "beCode not exist"); //Check whether becode already exists
                  require(beCodeAddr != msg.sender, "beCodeAddr can't be self");    //Check whether
becode is your own address
                  require(!isUsed(inviteCode), "invite code is used"); //Check whether the invitation code
has been used
                  registerUser(msg.sender, inviteCode, beCode); //Registered user
                  incInvertCount(beCodeAddr); //Increase the number of invitations
            }
            uint investAmout;
            uint lineAmount;
            if (isLine()) { //Online or not
                  lineAmount = amt; // msg.value;
            } else {
                  investAmout = amt; // msg.value;
            }
            User storage user = userRoundMapping[rid][msg.sender];
            if (user.id != 0) {
                  require(user.freezeAmount.add(user.lineAmount) == 0, "only once invest"); //Only one
investment is allowed
                  user.freezeAmount = investAmout;
                  user.lineAmount = lineAmount;
                  user.level = getLevel(user.freezeAmount); //Get level
                  user.lineLevel                                                                    =
getNodeLevel(user.freezeAmount.add(user.freeAmount).add(user.lineAmount)); //Get level
            } else {
                  user.id = userGlobal.id;
                  user.userAddress = msg.sender;
                  user.freezeAmount = investAmout;
                  user.level = getLevel(investAmout);//Get level
                  user.lineAmount = lineAmount;
                  user.lineLevel                                                                    =
getNodeLevel(user.freezeAmount.add(user.freeAmount).add(user.lineAmount));//Get level
                  user.inviteCode = userGlobal.inviteCode;
                  user.beCode = userGlobal.beCode;
            }

            rInvestCount[rid] = rInvestCount[rid].add(1);
            rInvestMoney[rid] = rInvestMoney[rid].add(amt);
            if (!isLine()) {//Online or not
                  sendFeetoAdmin(amt); //Transfer fee
                  countBonus(user.userAddress);
            } else {
                  lineArrayMapping[rid].push(user.id);
            }
            emit Deposit(msg.sender, amt);
        }
        function transferInvestAmount(uint amt) private {
            require(amt >= usd500 && amt <= usd3000, "between 100 USDT and 3000 USDT");
            require(amt == amt.div(unit100).mul(unit100), "must be multor of 100 USDT");
            TRC20(usdtAddr).transferFrom(msg.sender, address(this), amt);
        }
```

```
    function compareStr(string memory _str, string memory str) public pure returns (bool) {
        if (keccak256(abi.encodePacked(_str)) == keccak256(abi.encodePacked(str))) {
            return true;
        }
        return false;
    }
    function isUsed(string memory code) public view returns (bool) {
        address addr = addressMapping[code];
        return uint(addr) != 0;
    }
    function incInvertCount(address beCodeAddr) private {
        userInvites[beCodeAddr] += 1;
    }
    function sendFeetoAdmin(uint amount) private {
        // devAddr.transfer(amount.div(25));
        TRC20(usdtAddr).transfer(devAddr, amount.div(25));
    }
    function countBonus(address userAddr) private {
        User storage user = userRoundMapping[rid][userAddr];
        if (user.id == 0) {
            return;
        }
        uint scale = getScByLevel(user.level); //level
        user.dayBonusAmount = user.freezeAmount.mul(scale).div(1000);
        user.investTimes = 0;
        UserGlobal memory userGlobal = userMapping[userAddr];
        // if (user.freezeAmount >= 1 ether && user.freezeAmount <= bonuslimit &&
userGlobal.status == 0) {
        if (user.freezeAmount >= usd500 && user.freezeAmount <= bonuslimit &&
userGlobal.status == 0) {
            getaway(user.beCode, user.freezeAmount, scale);
        }
    }
    function getaway(string memory beCode, uint money, uint shareSc) private {
        string memory tmpReferrer = beCode;
        for (uint i = 1; i <= 25; i++) {
            if (compareStr(tmpReferrer, "")) { //Whether the invitation is empty
                break;
            }
            address tmpUserAddr = addressMapping[tmpReferrer];
            UserGlobal storage userGlobal = userMapping[tmpUserAddr];
            User storage calUser = userRoundMapping[rid][tmpUserAddr];

            if (calUser.freezeAmount.add(calUser.freeAmount).add(calUser.lineAmount) == 0) {
                tmpReferrer = userGlobal.beCode;
                continue;
            }

            // uint recommendSc = getRecommendScaleByLevelAndTim(3, i);
            uint recommendSc = getRecommendScaleByLevelAndTim(calUser.level, i);
            uint moneyResult = 0;
            // if (money <= 15 ether) {
            if (money <= usd3000) {
                moneyResult = money;
            } else {
                // moneyResult = 15 ether;
                moneyResult = usd3000;
            }

            if (recommendSc != 0) {
                uint tmpDynamicAmount = moneyResult.mul(shareSc).mul(recommendSc);
                tmpDynamicAmount = tmpDynamicAmount.div(1000).div(100);
                earneth(userGlobal.userAddress, tmpDynamicAmount, calUser.rewardIndex, i);
            }
            tmpReferrer = userGlobal.beCode;
        }
    }
    function earneth(address userAddr, uint dayInvAmount, uint rewardIndex, uint times) private {
        for (uint i = 0; i < 5; i++) {
            AwardData                    storage                    awData                    =
userAwardDataMapping[rid][userAddr][rewardIndex.add(i)];
```

```
                    if (times == 1) {
                            awData.oneInvAmount += dayInvAmount;
                    }
                    if (times == 2) {
                            awData.twoInvAmount += dayInvAmount;
                    }
                    awData.threeInvAmount += dayInvAmount;
            }
    }
```

Safety advice： ： NONE.

8.3.12. Withdraw withdrawal business logic 【security】

Audit description： Conduct security audit on the design of withdraw business logic of withraw in the contract, and check whether the relevant business logic is reasonable.

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

Code file： Dawn.sol L545~563

Code information：

```
        function withdraw() public isHuman() returns (uint) {    //It can only be called by the user address, and the contract and interaction are prohibited
            require(donnotimitate(), "no donnotimitate");
            User storage user = userRoundMapping[rid][msg.sender];
            require(user.id != 0, "user not exist"); //user does not exist
            uint sendMoney = user.freeAmount + user.lineAmount; //Calculate the total withdrawal amount
            bool isEnough = false;
            uint resultMoney = 0;

            (isEnough, resultMoney) = isEnoughBalance(sendMoney); //Check whether the withdrawal quantity is satisfied

            if (resultMoney > 0 && resultMoney <= withdrawLimit) { //Check whether the single
```

```
withdrawal limit is exceeded
                sendMoneyToUser(msg.sender, resultMoney);    //Withdrawal
                emit Withdraw(msg.sender, resultMoney);
                user.freeAmount = 0; //update operation
                user.lineAmount = 0;//update operation
                user.lineLevel = getNodeLevel(user.freezeAmount);//update operation
            }
            return resultMoney;
        }
        function isEnoughBalance(uint sendMoney) private view returns (bool, uint){
            uint balance = usdtbalance();
            if (sendMoney >= balance) {
                return (false, balance);
            } else {
                return (true, sendMoney);
            }
        }
        /// @dev usdt balance of address(this)
        function usdtbalance() public view returns (uint) {
            return TRC20(usdtAddr).balanceOf(address(this));
        }
        function sendMoneyToUser(address userAddress, uint money) private {
            if (money > 0) {
                // userAddress.transfer(money);
                TRC20(usdtAddr).transfer(userAddress, money);
            }
        }
```

Safety advice： NONE.

8.3.13. Christmas business logic design 【Low risk】

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

Audit results： The Christmas business logic in the contract has controllable start and end parameters, and the number of cycles is end. If the end is large, there is a self DOS risk due to the gas upper limit of a single transaction.

Code file: Dawn.sol L569~648

Code information:

```
function christmas(uint start, uint end) external onlyWhitelistAdmin {
        for (uint i = start; i <= end; i++) { //self-dos risk
            address userAddr = indexMapping[i];
            User storage user = userRoundMapping[rid][userAddr];
            UserGlobal memory userGlobal = userMapping[userAddr];
            // if (now.sub(user.lastRwTime) <= 12 hours) {
            if (now.sub(user.lastRwTime) <= bonusInterval) {
                continue;
            }
            user.lastRwTime = now;
            if (userGlobal.status == 1) {
                user.rewardIndex = user.rewardIndex.add(1);
                continue;
            }
            uint bonusSend = 0;
            // if (user.id != 0 && user.freezeAmount >= 1 ether && user.freezeAmount <=
bonuslimit) {
            if (user.id != 0 && user.freezeAmount >= usd500 && user.freezeAmount <=
bonuslimit) {
                if (user.investTimes < 5) {
                    bonusSend += user.dayBonusAmount;
                    user.bonusAmount = user.bonusAmount.add(bonusSend);
                    user.investTimes = user.investTimes.add(1);
                } else {
                    user.freeAmount = user.freeAmount.add(user.freezeAmount);
                    user.freezeAmount = 0;
                    user.dayBonusAmount = 0;
                    user.level = 0;
                }
            }
            uint lineAmount = user.freezeAmount.add(user.freeAmount).add(user.lineAmount);
            // if (lineAmount < 1 ether || lineAmount > withdrawLimit) {
            if (lineAmount < usd500 || lineAmount > withdrawLimit) {
                user.rewardIndex = user.rewardIndex.add(1);
                continue;
            }
            uint inviteSend = 0;
            if (userGlobal.status == 0) {
                AwardData memory awData =
userAwardDataMapping[rid][userAddr][user.rewardIndex];
                user.rewardIndex = user.rewardIndex.add(1);
                uint lineValue = lineAmount.div(unit100); // .div(2);
                if (lineValue >= 30) {
                    inviteSend += awData.threeInvAmount;
                } else {
                    // if (user.lineLevel == 1 && lineAmount >= 1 ether &&
awData.oneInvAmount > 0) {
                    if (user.lineLevel == 1 && lineAmount >= usd500 &&
awData.oneInvAmount > 0) {
                    inviteSend += awData.oneInvAmount.div(30).mul(lineValue).div(2);
                    }
                    // if (user.lineLevel == 2 && lineAmount >= 6 ether &&
(awData.oneInvAmount > 0 || awData.twoInvAmount > 0)) {
                    if (user.lineLevel == 2 && lineAmount >= usd1000 &&
(awData.oneInvAmount > 0 || awData.twoInvAmount > 0)) {
                    inviteSend +=
awData.oneInvAmount.div(30).mul(lineValue).mul(7).div(10);
                    inviteSend +=
awData.twoInvAmount.div(30).mul(lineValue).mul(5).div(7);
                    }
                    // if (user.lineLevel == 3 && lineAmount >= 11 ether &&
awData.threeInvAmount > 0) {
                    if (user.lineLevel == 3 && lineAmount >= usd2100 &&
awData.threeInvAmount > 0) {
                    inviteSend += awData.threeInvAmount.div(30).mul(lineValue);
                    }
                    if (user.lineLevel < 3) {
                    uint fireSc = getFireScByLevel(user.lineLevel);
                    inviteSend = inviteSend.mul(fireSc).div(10);
                    }
                }
            } else if (userGlobal.status == 2) {
                user.rewardIndex = user.rewardIndex.add(1);
            }

            if (bonusSend.add(inviteSend) <= sendLimit) {
                user.inviteAmonut = user.inviteAmonut.add(inviteSend);
                bool isEnough = false;
                uint resultMoney = 0;
                (isEnough, resultMoney) = isEnoughBalance(bonusSend.add(inviteSend));
                if (resultMoney > 0) {
                    uint comfortMoney = resultMoney.div(20);
                    sendMoneyToUser(comfortAddr, comfortMoney);
                    resultMoney = resultMoney.sub(comfortMoney);
                    address sendAddr = address((userAddr));
                    sendMoneyToUser(sendAddr, resultMoney);
                    emit Bonus(sendAddr, resultMoney, bonusSend, inviteSend);
                }
            }
        }
    }
    function isEnoughBalance(uint sendMoney) private view returns (bool, uint){
        uint balance = usdtbalance();
        if (sendMoney >= balance) {
            return (false, balance);
        } else {
            return (true, sendMoney);
        }
    }
    function sendMoneyToUser(address userAddress, uint money) private {
        if (money > 0) {
            // userAddress.transfer(money);
```

```
                TRC20(usdtAddr).transfer(userAddress, money);
          }
```

Safety advice： Check the number of cycles.

8.3.14. Endround business logic design 【security】

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

Audit results： The endround business logic design in the contract is correct.

Code file： Dawn.sol L698~703

Code information：

```
function endRound() external onlyOwner { //Only owner calls are allowed
          require(usdtbalance() < usd500, "contract balance must be lower than 500 usdt"); //Condition
check
          rid++;
          startTime = now.add(period).div(1 days).mul(1 days);
          canSetStartTime = 1;
```

Safety advice： NONE.

8.3.15. Getuserbyaddress business logic 【security】

Audit description： Conduct security audit on the business logic design of getuserbyaddress in the contract and check whether the relevant business logic is reasonable.

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

Code file： Dawn.sol L722~782

Code information：

```
function getUserByAddress(address addr, uint roundId) public view returns (uint[14] memory info,
string memory inviteCode, string memory beCode) {
        require(isWhitelistAdmin(msg.sender) || msg.sender == addr, "Permission denied for view
user's privacy"); //Permission check

        if (roundId == 0) {
            roundId = rid;
        }

        UserGlobal memory userGlobal = userMapping[addr];
        User memory user = userRoundMapping[roundId][addr];
        info[0] = userGlobal.id;
        info[1] = user.lineAmount;
        info[2] = user.freeAmount;
        info[3] = user.freezeAmount;
        info[4] = user.inviteAmonut;
        info[5] = user.bonusAmount;
        info[6] = user.lineLevel;
        info[7] = user.dayBonusAmount;
        info[8] = user.rewardIndex;
        info[9] = user.investTimes;
        info[10] = user.level;
        uint grantAmount = 0;
        // if (user.id > 0 && user.freezeAmount >= 1 ether && user.freezeAmount <= bonuslimit
&&
        //          user.investTimes < 5 && userGlobal.status != 1) {
        if (user.id > 0 && user.freezeAmount >= usd500 && user.freezeAmount <= bonuslimit &&
                user.investTimes < 5 && userGlobal.status != 1) {
            grantAmount += user.dayBonusAmount;
        }
        if (userGlobal.status == 0) {
            uint inviteSend = 0;
            AwardData                    memory                    awData                    =
userAwardDataMapping[rid][user.userAddress][user.rewardIndex];
            uint lineAmount = user.freezeAmount.add(user.freeAmount).add(user.lineAmount);
            if (lineAmount >= usd500) {
                uint lineValue = lineAmount.div(unit100); // .div(2);
                if (lineValue >= 30) {
                    inviteSend += awData.threeInvAmount;
                } else {
                    // if (user.lineLevel == 1 && lineAmount >= 1 ether && awData.oneInvAmount > 0) {
                    if (user.lineLevel == 1 && lineAmount >= usd500 && awData.oneInvAmount > 0) {
                        inviteSend += awData.oneInvAmount.div(30).mul(lineValue).div(2);
                    }
                    if (user.lineLevel == 2 && lineAmount >= usd500 && (awData.oneInvAmount > 0 || awData.twoInvAmount > 0)) {
                        inviteSend += awData.oneInvAmount.div(30).mul(lineValue).mul(7).div(10);
                        inviteSend += awData.twoInvAmount.div(30).mul(lineValue).mul(5).div(7);
                    }
                    if (user.lineLevel == 3 && lineAmount >= usd500 && awData.threeInvAmount > 0) {
                        inviteSend += awData.threeInvAmount.div(30).mul(lineValue);
                    }
                    if (user.lineLevel < 3) {
                        uint fireSc = getFireScByLevel(user.lineLevel); //Level acquisition
                        inviteSend = inviteSend.mul(fireSc).div(10);
                    }
                }
                grantAmount += inviteSend;
            }
        }
        info[11] = grantAmount;
        info[12] = user.lastRwTime;
        info[13] = userGlobal.status;

        return (info, userGlobal.inviteCode, userGlobal.beCode);
    }
```

Safety advice： NONE.

8.3.16. Getuseraddressbyid business logic 【security】

Audit description： Conduct security audit on the business logic design of getuseraddressbyid in the contract and check whether the relevant business logic is reasonable.

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

Code file： Dawn.sol L784~787

Code information：

```
function getUserAddressById(uint id) public view returns (address) {
            require(isWhitelistAdmin(msg.sender), "Permission denied"); //Caller identity check
            return indexMapping[id];
    }
```

Safety advice： NONE.

9. Contract source code

```
/**
 *Submitted for verification at BscScan.com on 2022-05-28
*/

pragma solidity 0.5.8;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
	// Empty internal constructor, to prevent people from mistakenly deploying
	// an instance of this contract, which should be used via inheritance.
	constructor() internal {}
	// solhint-disable-previous-line no-empty-blocks

	function _msgSender() internal view returns (address) {
		return msg.sender;
	}
}

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
	address private _owner;

	/**
	 * @dev Initializes the contract setting the deployer as the initial owner.
	 */
	constructor () internal {
		_owner = _msgSender();
	}

	/**
	 * @dev Throws if called by any account other than the owner.
	 */
	modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _owner = newOwner;
    }
}

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping(address => bool) bearer;
    }

    /**
     * @dev Give an account access to this role.
     */
    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role");
        role.bearer[account] = true;
    }

    /**
     * @dev Remove an account's access to this role.
     */
    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
        role.bearer[account] = false;
    }

    /**
     * @dev Check if an account has this role.
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0), "Roles: account is the zero address");
        return role.bearer[account];
    }
}

/**
 * @title WhitelistAdminRole
 * @dev WhitelistAdmins are responsible for assigning and removing Whitelisted accounts.
 */
contract WhitelistAdminRole is Context, Ownable {
    using Roles for Roles.Role;

    Roles.Role private _whitelistAdmins;

    constructor () internal {
    }

    modifier onlyWhitelistAdmin() {
        require(isWhitelistAdmin(_msgSender()) || isOwner(), "WhitelistAdminRole: caller does not
have the WhitelistAdmin role");
        _;
    }

    function isWhitelistAdmin(address account) public view returns (bool) {
        return _whitelistAdmins.has(account) || isOwner();
    }

    function addWhitelistAdmin(address account) public onlyOwner {
        _whitelistAdmins.add(account);
    }

    function removeWhitelistAdmin(address account) public onlyOwner {
        _whitelistAdmins.remove(account);
    }
}

contract TRC20 {
    function totalSupply() public view returns (uint theTotalSupply);
    function balanceOf(address _owner) public view returns (uint balance);
    function transfer(address _to, uint _value) public returns (bool success);
    function transferFrom(address _from, address _to, uint _value) public returns (bool success);
    function approve(address _spender, uint _value) public returns (bool success);
    function allowance(address _owner, address _spender) public view returns (uint remaining);
    event Transfer(address indexed _from, address indexed _to, uint _value);
    event Approval(address indexed _owner, address indexed _spender, uint _value);
}

contract BigWin is WhitelistAdminRole {
    using SafeMath for *;
    // uint ethWei = 1 ether;
    uint unit100        = 100 * (10 ** 18);
    uint usd500          = 100 * (10 ** 18);
    uint usd1000         = 1000 * (10 ** 18);
    uint usd1100         = 1100 * (10 ** 18);
    uint usd2000         = 2000 * (10 ** 18);
    uint usd2100         = 2100 * (10 ** 18);
    uint usd3000         = 3000 * (10 ** 18);
```

```
    uint usd20000       = 20000 * (10 ** 18);

    address payable private devAddr = address(0x0Ee152aB24453F32A5A3585AF68002FA87dC38eF);
    address payable private comfortAddr = address(0x8170FC3FCb38468a28b04Cf497ffe1c12812f562);
    address public usdtAddr = address(0x55d398326f99059fF775485246999027B3197955); // bsc usdt

    struct User {
        uint id;
        address userAddress;
        uint freeAmount;
        uint freezeAmount;
        uint lineAmount;
        uint inviteAmonut;
        uint dayBonusAmount;
        uint bonusAmount;
        uint level;
        uint lineLevel;
        uint resTime;
        uint investTimes;
        string inviteCode;
        string beCode;
        uint rewardIndex;
        uint lastRwTime;
    }

    struct UserGlobal {
        uint id;
        address userAddress;
        string inviteCode;
        string beCode;
        uint status;
    }
```

```
struct AwardData {
        uint oneInvAmount;
        uint twoInvAmount;
        uint threeInvAmount;
}

event Deposit(address indexed from, uint256 value);
event Withdraw(address indexed to, uint256 value);
event Bonus(address indexed to, uint256 total, uint256 bonusAmt, uint256 inviteAmt);

uint startTime;
uint lineStatus = 0;
mapping(uint => uint) rInvestCount;
mapping(uint => uint) rInvestMoney;
uint period = 1 days;
uint uid = 0;
uint rid = 1;
uint bonusInterval = 12 hours;
mapping(uint => uint[]) lineArrayMapping;
mapping(uint => mapping(address => User)) userRoundMapping;
mapping(address => uint) public userInvites;
mapping(address => UserGlobal) userMapping;
mapping(string => address) addressMapping;
mapping(uint => address) indexMapping;
mapping(uint => mapping(address => mapping(uint => AwardData))) userAwardDataMapping;
uint bonuslimit = 3000 * (10 ** 18); // 15 ether;
uint sendLimit = 20000 * (10 ** 18); // 100 ether;
uint withdrawLimit = 3000 * (10 ** 18); // 15 ether;
uint canImport = 1;
uint canSetStartTime = 1;

modifier isHuman() {
        address addr = msg.sender;
        uint codeLength;
        assembly {codeLength := extcodesize(addr)}
        require(codeLength == 0, "sorry humans only");
        require(tx.origin == msg.sender, "sorry, humans only");
        _;
    }

    constructor () public {
    }

    function() external payable {
    }

    function setUSDTAddr(address addr) external onlyOwner {
        usdtAddr = addr;
    }

    function setDevAddr(address payable dev, address payable comfort) external onlyOwner {
        devAddr = dev;
        comfortAddr = comfort;
    }

    /// @dev for debug
    function setLevel1Amt(uint amt, uint unit) external onlyOwner {
        usd500 = amt;
        unit100 = unit;
    }

    function getLevel(uint value) public view returns (uint) {
        if (value >= usd500 && value <= usd1000) {
            return 1;
        }
        if (value >= usd1100 && value <= usd2000) {
            return 2;
        }
        if (value >= usd2100 && value <= usd3000) {
            return 3;
        }
        return 0;
```

```
}

function getNodeLevel(uint value) public view returns (uint) {
        if (value >= usd500 && value <= usd1000) {
                return 1;
        }
        if (value >= usd1100 && value <= usd2000) {
                return 2;
        }
        if (value >= usd2100) {
                return 3;
        }
        return 0;
}

function getScByLevel(uint level) public pure returns (uint) {
        if (level == 1) {
                return 4;
        }
        if (level == 2) {
                return 6;
        }
        if (level == 3) {
                return 10;
        }
        return 0;
}

function getFireScByLevel(uint level) public pure returns (uint) {
        if (level == 1) {
                return 3;
        }
        if (level == 2) {
                return 6;
        }
        if (level == 3) {
            return 10;
        }
        return 0;
    }

    function getRecommendScaleByLevelAndTim(uint level, uint times) public pure returns (uint){
        if (level == 1 && times == 1) {
            return 40;
        }
        if (level == 2 && times == 1) {
            return 60;
        }
        if (level == 2 && times == 2) {
            return 40;
        }
        if (level == 3) {
            if (times == 1) {
                return 90;
            }
            if (times == 2) {
                return 60;
            }
            if (times == 3) {
                return 40;
            }
            if (times >= 4 && times <= 10) {
                return 8;
            }
            if (times >= 11 && times <= 20) {
                return 4;
            }
            if (times >= 21) {
                return 1;
            }
        }
        return 0;
    }

    function compareStr(string memory _str, string memory str) public pure returns (bool) {
        if (keccak256(abi.encodePacked(_str)) == keccak256(abi.encodePacked(str))) {
            return true;
        }
        return false;
    }

    function verydangerous(uint time) external onlyOwner {
        require(canSetStartTime == 1, "verydangerous, limited!");
        require(time > now, "no, verydangerous");
        startTime = time;
        canSetStartTime = 0;
    }

    function donnotimitate() public view returns (bool) {
        return startTime != 0 && now > startTime;
    }

    function updateLine(uint line) external onlyWhitelistAdmin {
        lineStatus = line;
    }

    function isLine() private view returns (bool) {
        return lineStatus != 0;
    }

    function actAllLimit(uint bonusLi, uint sendLi, uint withdrawLi) external onlyOwner {
        // require(bonusLi >= 15 ether && sendLi >= 100 ether && withdrawLi >= 15 ether,
"invalid amount");
        require(bonusLi >= usd3000 && sendLi >= usd20000 && withdrawLi >= usd3000, "invalid
amount");
        bonuslimit = bonusLi;
        sendLimit = sendLi;
        withdrawLimit = withdrawLi;
    }

    function stopImport() external onlyOwner {
        canImport = 0;
    }

    function actUserStatus(address addr, uint status) external onlyWhitelistAdmin {
        require(status == 0 || status == 1 || status == 2, "bad parameter status");
        UserGlobal storage userGlobal = userMapping[addr];
        userGlobal.status = status;
    }

    function transferInvestAmount(uint amt) private {
        require(amt >= usd500 && amt <= usd3000, "between 100 USDT and 3000 USDT");
        require(amt == amt.div(unit100).mul(unit100), "must be multor of 100 USDT");
        TRC20(usdtAddr).transferFrom(msg.sender, address(this), amt);
    }

    function incInvertCount(address beCodeAddr) private {
        userInvites[beCodeAddr] += 1;
    }

    function deposit(uint amt, string memory inviteCode, string memory beCode) public isHuman() {
        require(donnotimitate(), "no, donnotimitate");
        // require(msg.value >= 1 * ethWei && msg.value <= 15 * ethWei, "between 1 and 15");
        // require(msg.value == msg.value.div(ethWei).mul(ethWei), "invalid msg value");
        transferInvestAmount(amt);

        UserGlobal storage userGlobal = userMapping[msg.sender];
        if (userGlobal.id == 0) {
            require(!compareStr(inviteCode, "") && bytes(inviteCode).length == 6, "invalid invite
code");
            address beCodeAddr = addressMapping[beCode];
            require(isUsed(beCode), "beCode not exist");
            require(beCodeAddr != msg.sender, "beCodeAddr can't be self");
            require(!isUsed(inviteCode), "invite code is used");
            registerUser(msg.sender, inviteCode, beCode);
            incInvertCount(beCodeAddr);
        }
        uint investAmout;
        uint lineAmount;
        if (isLine()) {
            lineAmount = amt; // msg.value;
        } else {
            investAmout = amt; // msg.value;
        }
        User storage user = userRoundMapping[rid][msg.sender];
        if (user.id != 0) {
            require(user.freezeAmount.add(user.lineAmount) == 0, "only once invest");
            user.freezeAmount = investAmout;
            user.lineAmount = lineAmount;
            user.level = getLevel(user.freezeAmount);
            user.lineLevel                                                              =
getNodeLevel(user.freezeAmount.add(user.freeAmount).add(user.lineAmount));
        } else {
            user.id = userGlobal.id;
            user.userAddress = msg.sender;
            user.freezeAmount = investAmout;
            user.level = getLevel(investAmout);
            user.lineAmount = lineAmount;
            user.lineLevel                                                              =
getNodeLevel(user.freezeAmount.add(user.freeAmount).add(user.lineAmount));
            user.inviteCode = userGlobal.inviteCode;
            user.beCode = userGlobal.beCode;
        }

        rInvestCount[rid] = rInvestCount[rid].add(1);
        rInvestMoney[rid] = rInvestMoney[rid].add(amt);
        if (!isLine()) {
            sendFeetoAdmin(amt);
            countBonus(user.userAddress);
        } else {
            lineArrayMapping[rid].push(user.id);
        }
        emit Deposit(msg.sender, amt);
    }

    function importGlobal(address addr, string calldata inviteCode, string calldata beCode) external
onlyWhitelistAdmin {
        require(canImport == 1, "import stopped");
        UserGlobal storage user = userMapping[addr];
        require(user.id == 0, "user already exists");
        require(!compareStr(inviteCode, ""), "empty invite code");
        if (uid != 0) {
            require(!compareStr(beCode, ""), "empty beCode");
        }
        address beCodeAddr = addressMapping[beCode];
        require(beCodeAddr != addr, "beCodeAddr can't be self");
        require(!isUsed(inviteCode), "invite code is used");

        registerUser(addr, inviteCode, beCode);
        if (beCodeAddr != address(0)) {
            incInvertCount(beCodeAddr);
        }
    }

    function helloworld(uint start, uint end, uint isUser) external onlyWhitelistAdmin {
        for (uint i = start; i <= end; i++) {
            uint userId = 0;
            if (isUser == 0) {
                userId = lineArrayMapping[rid][i];
            } else {
                userId = i;
            }
            address userAddr = indexMapping[userId];
            User storage user = userRoundMapping[rid][userAddr];
            if (user.freezeAmount == 0 && user.lineAmount >= usd500 && user.lineAmount <=
usd3000) {
```

```
                user.freezeAmount = user.lineAmount;
                user.level = getLevel(user.freezeAmount);
                user.lineAmount = 0;
                sendFeetoAdmin(user.freezeAmount);
                countBonus(user.userAddress);
            }
        }
    }

    function countBonus(address userAddr) private {
        User storage user = userRoundMapping[rid][userAddr];
        if (user.id == 0) {
            return;
        }
        uint scale = getScByLevel(user.level);
        user.dayBonusAmount = user.freezeAmount.mul(scale).div(1000);
        user.investTimes = 0;
        UserGlobal memory userGlobal = userMapping[userAddr];
        // if (user.freezeAmount >= 1 ether && user.freezeAmount <= bonuslimit &&
userGlobal.status == 0) {
        if (user.freezeAmount >= usd500 && user.freezeAmount <= bonuslimit &&
userGlobal.status == 0) {
            getaway(user.beCode, user.freezeAmount, scale);
        }
    }

    function getaway(string memory beCode, uint money, uint shareSc) private {
        string memory tmpReferrer = beCode;

        for (uint i = 1; i <= 25; i++) {
            if (compareStr(tmpReferrer, "")) {
                break;
            }
            address tmpUserAddr = addressMapping[tmpReferrer];
            UserGlobal storage userGlobal = userMapping[tmpUserAddr];
            User storage calUser = userRoundMapping[rid][tmpUserAddr];
```

```
                if (calUser.freezeAmount.add(calUser.freeAmount).add(calUser.lineAmount) == 0) {
                    tmpReferrer = userGlobal.beCode;
                    continue;
                }

                // uint recommendSc = getRecommendScaleByLevelAndTim(3, i);
                uint recommendSc = getRecommendScaleByLevelAndTim(calUser.level, i);
                uint moneyResult = 0;
                // if (money <= 15 ether) {
                if (money <= usd3000) {
                    moneyResult = money;
                } else {
                    // moneyResult = 15 ether;
                    moneyResult = usd3000;
                }

                if (recommendSc != 0) {
                    uint tmpDynamicAmount = moneyResult.mul(shareSc).mul(recommendSc);
                    tmpDynamicAmount = tmpDynamicAmount.div(1000).div(100);
                    earneth(userGlobal.userAddress, tmpDynamicAmount, calUser.rewardIndex, i);
                }
                tmpReferrer = userGlobal.beCode;
            }
        }

        function earneth(address userAddr, uint dayInvAmount, uint rewardIndex, uint times) private {
            for (uint i = 0; i < 5; i++) {
                AwardData                storage                awData                =
userAwardDataMapping[rid][userAddr][rewardIndex.add(i)];
                if (times == 1) {
                    awData.oneInvAmount += dayInvAmount;
                }
                if (times == 2) {
                    awData.twoInvAmount += dayInvAmount;
                }
                awData.threeInvAmount += dayInvAmount;
            }
        }

        function withdraw() public isHuman() returns (uint) {
            require(donnotimitate(), "no donnotimitate");
            User storage user = userRoundMapping[rid][msg.sender];
            require(user.id != 0, "user not exist");
            uint sendMoney = user.freeAmount + user.lineAmount;
            bool isEnough = false;
            uint resultMoney = 0;

            (isEnough, resultMoney) = isEnoughBalance(sendMoney);

            if (resultMoney > 0 && resultMoney <= withdrawLimit) {
                sendMoneyToUser(msg.sender, resultMoney);
                emit Withdraw(msg.sender, resultMoney);
                user.freeAmount = 0;
                user.lineAmount = 0;
                user.lineLevel = getNodeLevel(user.freezeAmount);
            }
            return resultMoney;
        }

        function setBonusInterval(uint interval) external onlyOwner {
            bonusInterval = interval;
        }

        function christmas(uint start, uint end) external onlyWhitelistAdmin {
            for (uint i = start; i <= end; i++) {
                address userAddr = indexMapping[i];
                User storage user = userRoundMapping[rid][userAddr];
                UserGlobal memory userGlobal = userMapping[userAddr];
                // if (now.sub(user.lastRwTime) <= 12 hours) {
                if (now.sub(user.lastRwTime) <= bonusInterval) {
                    continue;
```

```
            }
            user.lastRwTime = now;
            if (userGlobal.status == 1) {
                user.rewardIndex = user.rewardIndex.add(1);
                continue;
            }
            uint bonusSend = 0;
            // if (user.id != 0 && user.freezeAmount >= 1 ether && user.freezeAmount <=
bonuslimit) {
            if (user.id != 0 && user.freezeAmount >= usd500 && user.freezeAmount <=
bonuslimit) {
                if (user.investTimes < 5) {
                    bonusSend += user.dayBonusAmount;
                    user.bonusAmount = user.bonusAmount.add(bonusSend);
                    user.investTimes = user.investTimes.add(1);
                } else {
                    user.freeAmount = user.freeAmount.add(user.freezeAmount);
                    user.freezeAmount = 0;
                    user.dayBonusAmount = 0;
                    user.level = 0;
                }
            }
            uint lineAmount = user.freezeAmount.add(user.freeAmount).add(user.lineAmount);
            // if (lineAmount < 1 ether || lineAmount > withdrawLimit) {
            if (lineAmount < usd500 || lineAmount > withdrawLimit) {
                user.rewardIndex = user.rewardIndex.add(1);
                continue;
            }
            uint inviteSend = 0;
            if (userGlobal.status == 0) {
                AwardData                memory                awData                =
userAwardDataMapping[rid][userAddr][user.rewardIndex];
                user.rewardIndex = user.rewardIndex.add(1);
                uint lineValue = lineAmount.div(unit100); // .div(2);
                if (lineValue >= 30) {
                    inviteSend += awData.threeInvAmount;
                } else {
                    // if (user.lineLevel == 1 && lineAmount >= 1 ether && awData.oneInvAmount > 0) {
                    if (user.lineLevel == 1 && lineAmount >= usd500 && awData.oneInvAmount > 0) {
                        inviteSend += awData.oneInvAmount.div(30).mul(lineValue).div(2);
                    }
                    // if (user.lineLevel == 2 && lineAmount >= 6 ether && (awData.oneInvAmount > 0 || awData.twoInvAmount > 0)) {
                    if (user.lineLevel == 2 && lineAmount >= usd1000 && (awData.oneInvAmount > 0 || awData.twoInvAmount > 0)) {
                        inviteSend += awData.oneInvAmount.div(30).mul(lineValue).mul(7).div(10);
                        inviteSend += awData.twoInvAmount.div(30).mul(lineValue).mul(5).div(7);
                    }
                    // if (user.lineLevel == 3 && lineAmount >= 11 ether && awData.threeInvAmount > 0) {
                    if (user.lineLevel == 3 && lineAmount >= usd2100 && awData.threeInvAmount > 0) {
                        inviteSend += awData.threeInvAmount.div(30).mul(lineValue);
                    }
                    if (user.lineLevel < 3) {
                        uint fireSc = getFireScByLevel(user.lineLevel);
                        inviteSend = inviteSend.mul(fireSc).div(10);
                    }
                }
            } else if (userGlobal.status == 2) {
                user.rewardIndex = user.rewardIndex.add(1);
            }

            if (bonusSend.add(inviteSend) <= sendLimit) {
                user.inviteAmonut = user.inviteAmonut.add(inviteSend);
                bool isEnough = false;
                uint resultMoney = 0;
                (isEnough, resultMoney) = isEnoughBalance(bonusSend.add(inviteSend));
```

```
                        if (resultMoney > 0) {
                                uint comfortMoney = resultMoney.div(20);
                                sendMoneyToUser(comfortAddr, comfortMoney);
                                resultMoney = resultMoney.sub(comfortMoney);
                                address sendAddr = address((userAddr));
                                sendMoneyToUser(sendAddr, resultMoney);
                                emit Bonus(sendAddr, resultMoney, bonusSend, inviteSend);
                        }
                }
        }
}

function isEnoughBalance(uint sendMoney) private view returns (bool, uint){
        uint balance = usdtbalance();
        if (sendMoney >= balance) {
                return (false, balance);
        } else {
                return (true, sendMoney);
        }
}

function sendFeetoAdmin(uint amount) private {
        // devAddr.transfer(amount.div(25));
        TRC20(usdtAddr).transfer(devAddr, amount.div(25));
}

function sendMoneyToUser(address userAddress, uint money) private {
        if (money > 0) {
                // userAddress.transfer(money);
                TRC20(usdtAddr).transfer(userAddress, money);
        }
}

function isUsed(string memory code) public view returns (bool) {
        address addr = addressMapping[code];
        return uint(addr) != 0;
```

```
    }

    function getUserAddressByCode(string memory code) public view returns (address) {
        require(isWhitelistAdmin(msg.sender), "Permission denied");
        return addressMapping[code];
    }

    function registerUser(address addr, string memory inviteCode, string memory beCode) private {
        UserGlobal storage userGlobal = userMapping[addr];
        uid++;
        userGlobal.id = uid;
        userGlobal.userAddress = addr;
        userGlobal.inviteCode = inviteCode;
        userGlobal.beCode = beCode;

        addressMapping[inviteCode] = addr;
        indexMapping[uid] = addr;
    }

    /// @dev usdt balance of address(this)
    function usdtbalance() public view returns (uint) {
        return TRC20(usdtAddr).balanceOf(address(this));
    }

    function endRound() external onlyOwner {
        require(usdtbalance() < usd500, "contract balance must be lower than 500 usdt");
        rid++;
        startTime = now.add(period).div(1 days).mul(1 days);
        canSetStartTime = 1;
    }

    function donnottouch() public view returns (uint, uint, uint, uint, uint, uint, uint, uint, uint, uint,
uint, uint) {
        return (
        rid,
        uid,
            startTime,
            rInvestCount[rid],
            rInvestMoney[rid],
            bonuslimit,
            sendLimit,
            withdrawLimit,
            canImport,
            lineStatus,
            lineArrayMapping[rid].length,
            canSetStartTime
            );
        }

        function getUserByAddress(address addr, uint roundId) public view returns (uint[14] memory
info, string memory inviteCode, string memory beCode) {
            require(isWhitelistAdmin(msg.sender) || msg.sender == addr, "Permission denied for view
user's privacy");

            if (roundId == 0) {
                roundId = rid;
            }

            UserGlobal memory userGlobal = userMapping[addr];
            User memory user = userRoundMapping[roundId][addr];
            info[0] = userGlobal.id;
            info[1] = user.lineAmount;
            info[2] = user.freeAmount;
            info[3] = user.freezeAmount;
            info[4] = user.inviteAmonut;
            info[5] = user.bonusAmount;
            info[6] = user.lineLevel;
            info[7] = user.dayBonusAmount;
            info[8] = user.rewardIndex;
            info[9] = user.investTimes;
            info[10] = user.level;
            uint grantAmount = 0;
            // if (user.id > 0 && user.freezeAmount >= 1 ether && user.freezeAmount <= bonuslimit
&&
            //          user.investTimes < 5 && userGlobal.status != 1) {
            if (user.id > 0 && user.freezeAmount >= usd500 && user.freezeAmount <= bonuslimit &&
                    user.investTimes < 5 && userGlobal.status != 1) {
                grantAmount += user.dayBonusAmount;
            }
            if (userGlobal.status == 0) {
                uint inviteSend = 0;
                AwardData memory awData =
userAwardDataMapping[rid][user.userAddress][user.rewardIndex];
                uint lineAmount = user.freezeAmount.add(user.freeAmount).add(user.lineAmount);
                if (lineAmount >= usd500) {
                    uint lineValue = lineAmount.div(unit100); // .div(2);
                    if (lineValue >= 30) {
                        inviteSend += awData.threeInvAmount;
                    } else {
                        // if (user.lineLevel == 1 && lineAmount >= 1 ether &&
awData.oneInvAmount > 0) {
                        if (user.lineLevel == 1 && lineAmount >= usd500 &&
awData.oneInvAmount > 0) {
                        inviteSend += awData.oneInvAmount.div(30).mul(lineValue).div(2);
                        }
                        if (user.lineLevel == 2 && lineAmount >= usd500 &&
(awData.oneInvAmount > 0 || awData.twoInvAmount > 0)) {
                        inviteSend +=
awData.oneInvAmount.div(30).mul(lineValue).mul(7).div(10);
                        inviteSend +=
awData.twoInvAmount.div(30).mul(lineValue).mul(5).div(7);
                        }
                        if (user.lineLevel == 3 && lineAmount >= usd500 &&
awData.threeInvAmount > 0) {
                        inviteSend += awData.threeInvAmount.div(30).mul(lineValue);
                        }
                        if (user.lineLevel < 3) {
                        uint fireSc = getFireScByLevel(user.lineLevel);
```

```
                                                inviteSend = inviteSend.mul(fireSc).div(10);
                                        }
                                }
                                grantAmount += inviteSend;
                        }
                }
                info[11] = grantAmount;
                info[12] = user.lastRwTime;
                info[13] = userGlobal.status;

                return (info, userGlobal.inviteCode, userGlobal.beCode);
        }

        function getUserAddressById(uint id) public view returns (address) {
                require(isWhitelistAdmin(msg.sender), "Permission denied");
                return indexMapping[id];
        }

        function getLineUserId(uint index, uint rouId) public view returns (uint) {
                require(isWhitelistAdmin(msg.sender), "Permission denied");
                if (rouId == 0) {
                        rouId = rid;
                }
                return lineArrayMapping[rid][index];
        }
}

/**
* @title SafeMath
* @dev Math operations with safety checks that revert on error
*/
library SafeMath {
        /**
        * @dev Multiplies two numbers, reverts on overflow.
        */
        function mul(uint256 a, uint256 b) internal pure returns (uint256) {
```

CHAINLION

```
        // 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-solidity/pull/522
        if (a == 0) {
                return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "mul overflow");

        return c;
}

/**
* @dev Integer division of two numbers truncating the quotient, reverts on division by zero.
*/
function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "div zero");
        // Solidity only automatically asserts when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
}

/**
* @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
*/
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "lower sub bigger");
        uint256 c = a - b;

        return c;
}

/**
```

```
* @dev Adds two numbers, reverts on overflow.
*/
function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "overflow");

        return c;
}

/**
* @dev Divides two numbers and returns the remainder (unsigned integer modulo),
* reverts when dividing by zero.
*/
function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "mod zero");
        return a % b;
}

/**
* @dev compare two numbers and returns the smaller one.
*/
function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a > b ? b : a;
}
```

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