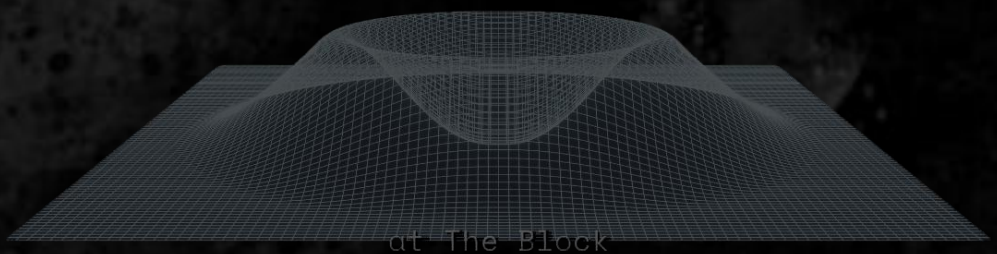
CHAINLION

BINGO

Smart Contract Audit Report

OCT 31th，2022

NO.0C002210310001



CATALOGUE

1. PROJECT SUMMARY

Entry type	Specific description
Entry name	BINGO
Project type	BEP-20
Application platform	BSC
DawnToken	0x58C2Cc04b2859916C5E5683545B349df3d7530B8

2. AUDIT SUMMARY

Entry type	Specific description
Project cycle	OCT/28/2022-OCT/31/2022
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	0
Low risk vulnerability	0

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 **BINGO** 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

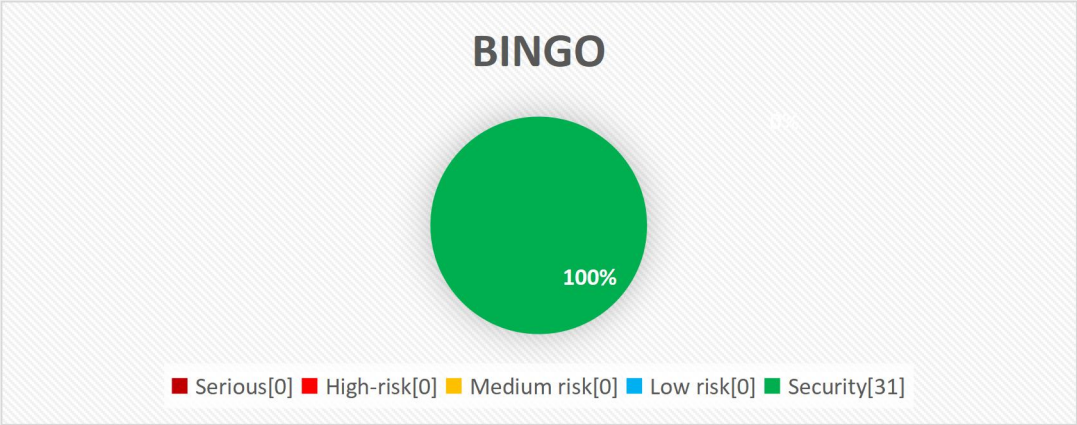
```
Address.sol
BingoToken.sol
Context.sol
IERC20.sol
Ownable.sol
SafeMath.sol
```

6. File hashes

Contract	SHA1 Checksum
Address.sol	EC295C93CB5EE1BC1DD7E1DB327EACB5704F1694
BingoToken.sol	63CE1200FACA09002CF268ECFAA0D5F650C68904

Context.sol	37CCEF725837F7FA2F3DD5446C62A9776B435B6F
IERC20.sol	3A4C9C7E4D7BDC778A551C2601C77FC0A8043614
Ownable.sol	FCAB37D90398E8463C3FBA93801109A46BB3846B
SafeMath.sol	A31FA4118424E37612933C0C4032476574FF54D5

7. Vulnerability distribution



8. Audit content

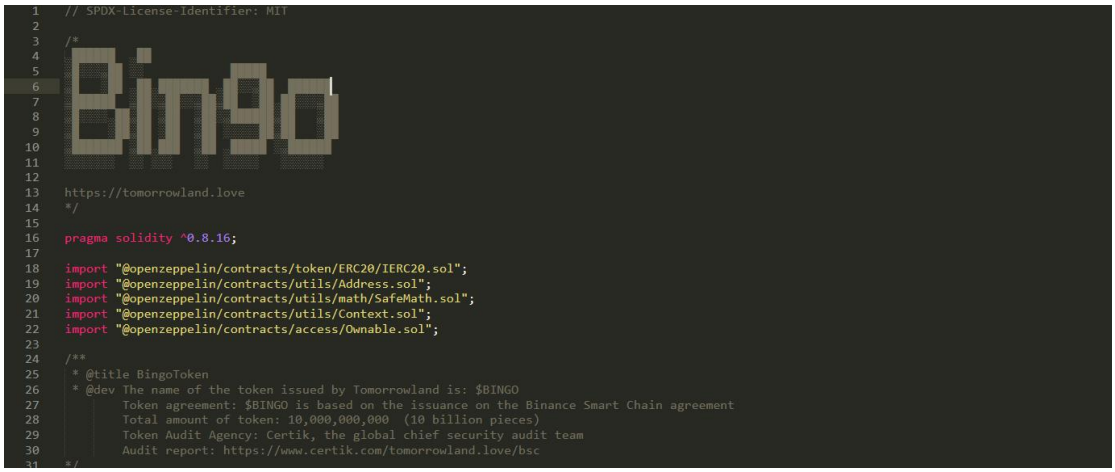
8.1. Coding specification

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 0.8.0, so there is no such security problem.



```
// SPDX-License-Identifier: MIT

/*
 ____  _                 
| __ )(_)_ __   __ _  ___ 
|  _ \| | '_ \ / _` |/ _ \
| |_) | | | | | (_| | (_) |
|____/|_|_| |_|\__, |\___/ 
               |___/       

https://tomorrowland.love
*/

pragma solidity ^0.8.16;

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

/**
 * @title BingoToken
 * @dev The name of the token issued by Tomorrowland is: $BINGO
        Token agreement: $BINGO is based on the issuance on the Binance Smart Chain agreement
        Total amount of token: 10,000,000,000  (10 billion pieces)
        Token Audit Agency: Certik, the global chief security audit team
        Audit report: https://www.certik.com/tomorrowland.love/bsc
*/
```

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.

```
63      /**
64       * @dev constructor
65       */
66      constructor () {
67          //total supply 10,000,000,000
68          _mint(_groupAddr, 600_000_000 ether);
69          _mint(_institutionalFinancingAddr, 400_000_000 ether);
70          _mint(_jackpotPoolAddr, 5_000_000_000 ether);
71          _mint(_IDOAddr, 2_000_000_000 ether);
72          _mint(_clubRewardAddr, 2_000_000_000 ether);
73          _mint(_msgSender(), 2_000_000_000 ether);
74
75          //init buyFeeRateMap
76          _feeRateMap[_jackpotPoolAddr] = 2;
77          _feeRateMap[_clubRewardAddr] = 1;
78          _feeRateMap[_deadAddr] = 2;
79          _feeRateMap[_groupAddr] = 1;
80          _feeRateMap[_transferFeeReceiveAddr] = 2;
81
82          //init whiteList
83          _excludeFee[_groupAddr] = 1;
84          _excludeFee[_institutionalFinancingAddr] = 1;
85          _excludeFee[_jackpotPoolAddr] = 1;
86          _excludeFee[_IDOAddr] = 1;
87          _excludeFee[_clubRewardAddr] = 1;
88          _excludeFee[_transferFeeReceiveAddr] = 1;
89      }
```

Safety advice：NONE.

8.1.4. Key event trigger【security】

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 key operations of SetXXX triggered the

corresponding events through the emit.

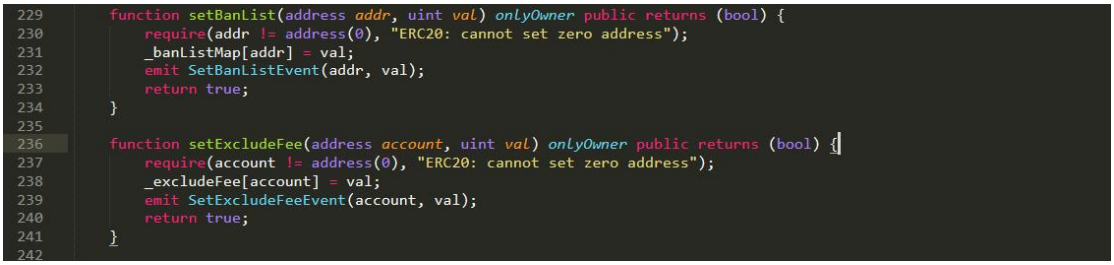
```
function setBanList(address addr, uint val) onlyOwner public returns (bool) {
    require(addr != address(0), "ERC20: cannot set zero address");
    _banListMap[addr] = val;
    emit SetBanListEvent(addr, val);
    return true;
}

function setExcludeFee(address account, uint val) onlyOwner public returns (bool) {
    require(account != address(0), "ERC20: cannot set zero address");
    _excludeFee[account] = val;
    emit SetExcludeFeeEvent(account, val);
    return true;
}

function pauseTransfer(bool isPause) onlyOwner external returns (bool){
    _pauseTransfer = isPause;
    emit PauseTransfer(isPause);
    return true;
}

function setJackpotFee(uint fee) onlyOwner external returns (bool) {
    _feeRateMap[_jackpotPoolAddr] = fee;
    emit SetJackpotFeeEvent(fee);
    return true;
}

function setClubFee(uint fee) onlyOwner external returns (bool) {
    _feeRateMap[_clubRewardAddr] = fee;
    emit SetClubFeeEvent(fee);
    return true;
}

function setDeadFee(uint fee) onlyOwner external returns (bool) {
    _feeRateMap[_deadAddr] = fee;
    emit SetDeadFeeEvent(fee);
    return true;
}
```

Safety advice：NONE.

8.1.5. Address non-zero check【security】

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： It is audited that there is a non-zero check on the address.



```
function setBanList(address addr, uint val) onlyOwner public returns (bool) {
    require(addr != address(0), "ERC20: cannot set zero address");
    _banListMap[addr] = val;
    emit SetBanListEvent(addr, val);
    return true;
}

function setExcludeFee(address account, uint val) onlyOwner public returns (bool) {
    require(account != address(0), "ERC20: cannot set zero address");
    _excludeFee[account] = val;
    emit SetExcludeFeeEvent(account, val);
    return true;
}
```

```
273     function setSwapAddrMap(address addr, uint val) onlyOwner external returns (bool) {
274         require(addr != address(0), "ERC20: cannot set zero address");
275         _swapAddrMap[addr] = val;
276         emit SetSwapAddrMapEvent(addr, val);
277         return true;
278     }
279
280     function setGroupAddr(address addr) onlyOwner external returns (bool) {
281         require(addr != address(0), "ERC20: cannot set zero address");
282         _groupAddr = addr;
283         emit SetGroupAddrEvent(addr);
284         return true;
285     }
286
287     function setInstitutionalFinancingAddr(address addr) onlyOwner external returns (bool) {
288         require(addr != address(0), "ERC20: cannot set zero address");
289         _institutionalFinancingAddr = addr;
290         emit SetInstitutionalFinancingAddrEvent(addr);
291         return true;
292     }
293
294     function setJackpotPoolAddr(address addr) onlyOwner external returns (bool) {
295         require(addr != address(0), "ERC20: cannot set zero address");
296         _jackpotPoolAddr = addr;
297         emit SetJackPotPoolAddrEvent(addr);
298         return true;
299     }
300
301     function setIDOAddr(address addr) onlyOwner external returns (bool) {
302         require(addr != address(0), "ERC20: cannot set zero address");
303         _IDOAddr = addr;
304         emit SetIDOAddrEvent(addr);
305         return true;
306     }
307
308     function setClubRewardAddr(address addr) onlyOwner external returns (bool) {
309         require(addr != address(0), "ERC20: cannot set zero address");
310         _clubRewardAddr = addr;
311         emit SetClubRewardAddrEvent(addr);
312         return true;
313     }
```

Safety advice：NONE .

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.

```
// SPDX-License-Identifier: MIT

/*
Bingo

https://tomorrowland.love
*/

pragma solidity ^0.8.16;

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

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. IsContract contract address judgment【security】

Audit description： Conduct security audit on the check logic of whether the function caller address in the contract is a contract address to check whether there is a risk of being bypassed. For example, judge whether it is a wrong logic of the contract address only according to extcodesize.

Audit results： The isContract business logic design of the check function for whether the address in the contract is the contract address is correct, and no related security risks are found.

Code file： Address.sol 36~42

Code information：

```
/**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
```

```
        *
        * Among others, `isContract` will return false for the following
        * types of addresses:
        *
        *  - an externally-owned account
        *  - a contract in construction
        *  - an address where a contract will be created
        *  - an address where a contract lived, but was destroyed
        * ====
        *
        * [IMPORTANT]
        * ====
        * You shouldn't rely on `isContract` to protect against flash loan attacks!
        *
        * Preventing calls from contracts is highly discouraged. It breaks composability, breaks support for smart wallets
        * like Gnosis Safe, and does not provide security since it can be circumvented by calling from a contract
        * constructor.
        * ====
        */
    function isContract(address account) internal view returns (bool) {
            // This method relies on extcodesize/address.code.length, which returns 0
            // for contracts in construction, since the code is only stored at the end
            // of the constructor execution.

            return account.code.length > 0;    //Check if it is a contract address

            }
```

Safety advice：NONE.

8.3.2. SendValue business logic design【security】

Audit description：Conduct security audit on the sendValue business logic in the contract to check whether there are parameters for validity verification and whether the design of relevant business logic is reasonable.

Audit results： The sendValue function in the contract can only be called internally. The validity of the function parameters has been verified, and the relevant

business logic design is correct.

Code file： Address.sol 60~65

Code information：

```
    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     *
https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {    //Only internal calls are allowed
        require(address(this).balance >= amount, "Address: insufficient balance");    //Check whether the number of assets held is sufficient for transfer

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may have reverted"); //Check whether the call is successful

    }
```

Safety advice： NONE.

8.3.3. FunctionCall business logic design【security】

Audit results： Conduct security audit on the functionCall business logic in the contract to check whether there are parameters for validity verification and whether

the design of relevant business logic is reasonable.

Audit results： The functionCall function in the contract can only be called internally. The validity of the function parameters has been verified, and the relevant business logic design is correct.

Code file： Address.sol 85~87

Code information：

```
    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain `call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     *                                                                    use
https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionCall(target, data, "Address: low-level call failed");
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(
        address target,
        bytes memory data,
```

```
        string memory errorMessage
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, errorMessage);
    }
    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");    //Check whether the assets held are sufficient for transfer
        require(isContract(target), "Address: call to non-contract");

        (bool success, bytes memory returndata) = target.call{value: value}(data);
        return verifyCallResult(success, returndata, errorMessage); //Call result validation
    }
    /**
     * @dev Tool to verifies that a low level call was successful, and revert if it wasn't, either by bubbling the
     * revert reason using the provided one.
     *
     * _Available since v4.3._
     */
    function verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) internal pure returns (bytes memory) {
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly
                /// @solidity memory-safe-assembly
                assembly {
```

```
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
```

Safety advice：NONE.

8.3.4. FunctionDelegateCall logic【security】

Audit description： Conduct security audit on the functionDelegateCall business logic in the contract to check whether there are parameters for validity verification and whether the design of relevant business logic is reasonable.

Audit results： The functionDelegateCall function in the contract can only be called internally. The validity of the function parameters has been verified, and the relevant business logic design is correct.

Code file： Address.sol 174~176

Code information：

```
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
{
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }
    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
```

```
     * _Available since v3.4._
     */
    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract"); //Check whether the target address is a contract address

        (bool success, bytes memory returndata) = target.delegatecall(data);
        return verifyCallResult(success, returndata, errorMessage);    //Verify call results

    }
```

Safety advice： NONE.

8.3.5. RenounceOwnership logic【security】

Audit description： Conduct security audit on the business logic design of the renounceOwnership giving up the contract owner authority in the contract to check whether the function caller authority is checked and whether there are security problems such as logic design defects.

Audit results： The renounceOwnership function in the contract can only be called by the owner of the current contract, and the relevant business logic is designed correctly.

Code file： Ownable.sol 61~63

Code information：

```
    /**
    * @dev Leaves the contract without owner. It will not be possible to call
    * `onlyOwner` functions anymore. Can only be called by the current owner.
    *
    * NOTE: Renouncing ownership will leave the contract without an owner,
    * thereby removing any functionality that is only available to the owner.
```

```
     */
    function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(address(0));    //Update the owner of the contract
    }
    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;      //Staging the old owner address
        _owner = newOwner;    //Update owner address
        emit OwnershipTransferred(oldOwner, newOwner);
    }
    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        _checkOwner();    //Check the caller's identity information
        _;
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if the sender is not the owner.
     */
    function _checkOwner() internal view virtual {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");     //Only contract
owner calls are allowed

        }
```

Safety advice： NONE.

8.3.6. TransferOwnership logic【security】

Audit description： Conduct security audit on the business logic design of

transferOwnership in the contract to update the contract owner's permission, check whether the function caller's permission is checked, and check whether there are security problems such as logic design defects.

Audit results： The transferOwnership function in the contract can only be called by the owner of the current contract, and the relevant business logic design is correct.

Code file： Ownable.sol 69~73

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 virtual onlyOwner { //Only contract owner calls are allowed
        require(newOwner != address(0), "Ownable: new owner is the zero address");    //New owner address non-zero check
        _transferOwnership(newOwner);    //Update the owner of the contract
    }
    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;      //Staging the old owner address
        _owner = newOwner;    //Update owner address
        emit OwnershipTransferred(oldOwner, newOwner);

    }
```

Safety advice： NONE.

8.3.7. Constructor initializes business 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 statement in the requirements document.

Audit results： The total amount of constructor initialization issuance in the contract is 10000000000, and the initial coinage distribution proportion is consistent with the document description.

Code file： BingoToken.sol 66~89

Code information：

```
    using Address for address;
    using SafeMath for uint;

    string private _name = '$Bingo';    //Token name
    string private _symbol = 'BG';      //Token symbol
    uint8 private _decimals = 18;       //Token accuracy
    uint private _totalSupply;          //Total number of tokens issued
    uint private _totalFee;             //Fee

    mapping (address => uint) private _balances;
    mapping (address => mapping (address => uint)) private _allowances;
    mapping (address => uint) public _excludeFee;      //Whitelist address
    mapping (address => uint) public _banListMap;      //Blacklist address
    mapping (address => uint) public _totalReceiveFeeMap;    //Receive Free address
    mapping (address => uint) public _feeRateMap;    //rate
    mapping (address => uint) public _swapAddrMap;    //swap address

    address public _jackpotPoolAddr = 0xa93829524E0213e3FA261C353e2dB400779e2dd6;
//Accumulated bonus pool address
    address public _clubRewardAddr = 0xe7F1EE9d254a1836E7313F5E1571BF02b347c30C;    //
Club Pool Address
    address public _groupAddr = 0x74D1DDBDCFd0068EDebfc4b366dB8cc86430d69B;
//Address of technical operation team
    address public _IDOAddr = 0x2e8E8eeb390c7021d749C23fBF88Cb19B103fa61;          //IDO
address
    address public _institutionalFinancingAddr =
0xF411724b82BaD44d125983409dD407E8285707EB; //Institutional investment address
    address public _transferFeeReceiveAddr = 0xa93829524E0213e3FA261C353e2dB400779e2dd6;
    address public _deadAddr = 0x000000000000000000000000000000000000dEaD;
```

```
    bool _pauseTransfer = true;

    event PauseTransfer(bool isPause);

    /**
     * @dev constructor
     */
    constructor () {
        //total supply 10,000,000,000
        _mint(_groupAddr, 600_000_000 ether);    //6% for technical operation team
        _mint(_institutionalFinancingAddr,    400_000_000    ether);        //Institutional    investment
allocation 4%
        _mint(_jackpotPoolAddr, 5_000_000_000 ether);    //Accumulated bonus pool address 50%
        _mint(_IDOAddr, 2_000_000_000 ether);    //IOD 20%
        _mint(_clubRewardAddr, 2_000_000_000 ether);    // Club trophy pool 20%
        _mint(_msgSender(), 2_000_000_000 ether); //contract owner 20%

        //Initial rate collection proportion
        _feeRateMap[_jackpotPoolAddr] = 2;
        _feeRateMap[_clubRewardAddr] = 1;
        _feeRateMap[_deadAddr] = 2;
        _feeRateMap[_groupAddr] = 1;
        _feeRateMap[_transferFeeReceiveAddr] = 2;

        //Initialize whitelist address
        _excludeFee[_groupAddr] = 1;
        _excludeFee[_institutionalFinancingAddr] = 1;
        _excludeFee[_jackpotPoolAddr] = 1;
        _excludeFee[_IDOAddr] = 1;
        _excludeFee[_clubRewardAddr] = 1;
        _excludeFee[_transferFeeReceiveAddr] = 1;

}
```

Safety advice：NONE.

8.3.8. _ Mint token issuance business logic【security】

Audit description： Conduct security audit on the logic design of the token issuance business in the contract to check whether the token issuance function checks the identity of the caller and whether there is any overflow or business design

defect.

Audit results： The coin function in the contract is only allowed to be called internally, and the address validity is checked, and the relevant logic design is correct.

Code file： BingoToken.sol 91~97

Code information：

```
function _mint(address account, uint amount) internal {    //Only internal calls are allowed
        require(account != address(0), "ERC20: mint to the zero address");    //Mint address non-zero check
        _totalSupply = _totalSupply.add(amount);    //Update of total token issuance
       _balances[account] = _balances[account].add(amount);    //Coining operation
        emit Transfer(address(0), account, amount);

      }
```

Safety advice：：NONE.

8.3.9. Token basic information query logic【security】

Audit description： Conduct security audit on the logic design of token information query business in the contract to check whether there are any defects in the logic design.

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

Code file： BingoToken.sol 98~120

Code information：

```
function name() public view virtual returns (string memory) {
        return _name;      //Token name query
    }

    function symbol() public view virtual returns (string memory) {
```

```
        return _symbol;      //Token symbol query
    }

    function decimals() public view virtual returns (uint8) {
        return _decimals;      //Token precision query
    }

    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;      //Total Token Issuance Query
    }

    function balanceOf(address account) public view virtual override returns (uint256) {
        return _balances[account];      //Address corresponding asset quantity query
    }

    function totalFee() public view returns (uint256) {
        return _totalFee;      //Total Expense Query
    }
```

Safety advice： ： NONE.

8.3.10. Transfer transfer business logic【security】

Audit description： Conduct security audit on the logic design of the transfer business in the contract to check whether there are any defects in the logic design.

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

Code file： BingoToken.sol 122~125

Code information：

```
 function transfer(address recipient, uint amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);    //Call_ Transfer
        return true;
    }
    function _transfer(address sender, address recipient, uint amount) internal{
        require(_pauseTransfer == false || sender == owner(), "ERC20: transfer is paused");
//Check whether transfer is prohibited or the sender address is the owner address of the contract
        require(sender != address(0), "ERC20: transfer from the zero address");    //Check whether
the sender address is zero
        require(recipient != address(0), "ERC20: transfer to the zero address");    //Check whether the
```

```
recipient address is zero
        require(_banListMap[sender] == 0, "ERC20: sender be pulled black");    //Check whether the sender is in the blacklist address
        require(_banListMap[recipient] == 0, "ERC20: recipient be pulled black");    //Check whether the recipient address is in the blacklist address
        _balances[sender] = _balances[sender].sub(amount);    //Update the number of sender address assets (using the Safemath operation, there is no risk of overflow)

        if(recipient == _deadAddr){    //Check whether recipient is a black hole address
            _totalSupply -= amount;    //If it is a black hole address, update it directly_ TotalSupply
            emit Transfer(sender, _deadAddr, amount);
            return;
        }

        uint addValue = amount;       //Total amount of assets involved in the whole transaction
        uint onePercent = amount.div(100);    //Split assets into 100

        if(_swapAddrMap[recipient] == 1 && _excludeFee[sender] != 1){    //Check whether the recipient address is_ SwapAddr address (for sale) and sender is a non transaction fee free address
            uint jackpotFee = onePercent.mul(_feeRateMap[_jackpotPoolAddr]);    //2% of sales slip points are injected into the project prize pool
            uint clubFee = onePercent.mul(_feeRateMap[_clubRewardAddr]);    //1% into the club prize pool
            uint deadFee = onePercent.mul(_feeRateMap[_deadAddr]);    //2% hack into the black hole address and permanently destroy it
            uint groupFee = onePercent.mul(_feeRateMap[_groupAddr]); //1% allocated to the project team

            _totalReceiveFeeMap[_jackpotPoolAddr] = _totalReceiveFeeMap[_jackpotPoolAddr].add(jackpotFee);    //Update charges
            _totalReceiveFeeMap[_clubRewardAddr] = _totalReceiveFeeMap[_clubRewardAddr].add(clubFee);    //Update charges
            _totalReceiveFeeMap[_deadAddr] = _totalReceiveFeeMap[_deadAddr].add(deadFee); //Update charges
            _totalReceiveFeeMap[_groupAddr] = _totalReceiveFeeMap[_groupAddr].add(groupFee);       //Update charges

            _balances[_jackpotPoolAddr] = _balances[_jackpotPoolAddr].add(jackpotFee); //Update asset quantity
            _balances[_clubRewardAddr] = _balances[_clubRewardAddr].add(clubFee); //Update asset quantity
            _balances[_deadAddr] = _balances[_deadAddr].add(deadFee);    //Update asset quantity
            _balances[_groupAddr] = _balances[_groupAddr].add(groupFee);       //Update asset quantity
```

```
                uint currentFee = jackpotFee.add(clubFee).add(deadFee).add(groupFee);     //Calculate
the total amount of fee assets charged
                _totalFee = _totalFee.add(currentFee); //Calculate total cost
                addValue = addValue.sub(currentFee); //Update the actual number of assets to be
transferred after expenses are removed

                _totalSupply -= deadFee;    //Update Total(Removal of destroyed assets)
                _balances[recipient] = _balances[recipient].add(addValue);     //Update the number of
assets held by recipient address
                emit Transfer(sender, recipient, amount);
                return;
            }

            if(_excludeFee[sender] != 1 && _excludeFee[recipient] != 1){    //If the sender is not a free
transaction address and the recipient is not a free transaction address
                uint      transferFee      =      onePercent.mul(_feeRateMap[_transferFeeReceiveAddr]);
//Calculation of transaction fee

                _totalReceiveFeeMap[_transferFeeReceiveAddr]                                          =
_totalReceiveFeeMap[_transferFeeReceiveAddr].add(transferFee);//Update the total fees charged
                _balances[_transferFeeReceiveAddr]                                                    =
_balances[_transferFeeReceiveAddr].add(transferFee); //Update total expense quantity
                _totalFee = _totalFee.add(transferFee); //Calculate total cost
                addValue = addValue.sub(transferFee); //Calculate the amount of the actual transfer

                _balances[recipient] = _balances[recipient].add(addValue); //Update transfer amount
                emit Transfer(sender, recipient, amount);
                return;
            }

            _balances[recipient] = _balances[recipient].add(addValue); //Direct update amount (0%
purchase slip point)
            emit Transfer(sender, recipient, amount);

        }
```

Safety advice： NONE.

8.3.11. Logic design of authorized transfer business【security】

Audit description： Conduct security audit on the logic design of authorized

transfer business in the contract to check whether there are any defects in the logic design.

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

Code file： BingoToken.sol 127~134

Code information：

```
function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];    //Query authorization limit
    }

    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);      //Call_ Approve to perform authorization operations
        return true;
    }
    function increaseAllowance(address spender, uint addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue)); //Increase authorization limit
        return true;
    }

    function decreaseAllowance(address spender, uint subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero")); //Reduce the amount of authorization
        return true;
    }

    function _approve(address owner, address spender, uint amount) internal virtual {
        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.

8.3.12. TransferFrom transfer business【security】

Audit description： Conduct security audit on the logic design of transferFrom transfer business in the contract to check whether there are any defects in the logic design.

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

Code file： BingoToken.sol 136~140

Code information：

```
 function transferFrom(address sender, address recipient, uint amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);    //Call_ Transfer
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance")); //Update authorization limit
        return true;
    }
    function _transfer(address sender, address recipient, uint amount) internal{
        require(_pauseTransfer == false || sender == owner(), "ERC20: transfer is paused"); //Check whether transfer is prohibited or the sender address is the owner address of the contract
        require(sender != address(0), "ERC20: transfer from the zero address");    //Check whether the sender address is zero
        require(recipient != address(0), "ERC20: transfer to the zero address");    //Check whether the recipient address is zero
        require(_banListMap[sender] == 0, "ERC20: sender be pulled black");    //Check whether the sender is in the blacklist address
        require(_banListMap[recipient] == 0, "ERC20: recipient be pulled black");    //Check whether the recipient address is in the blacklist address
        _balances[sender] = _balances[sender].sub(amount);    //Update the number of sender address assets (using the Safemath operation, there is no risk of overflow)

        if(recipient == _deadAddr){    //Check whether recipient is a black hole address
            _totalSupply -= amount;    //If it is a black hole address, update it directly_ TotalSupply
            emit Transfer(sender, _deadAddr, amount);
            return;
        }

        uint addValue = amount;       //Total amount of assets involved in the whole transaction
```

```
        uint onePercent = amount.div(100);    //Split assets into 100

        if(_swapAddrMap[recipient] == 1 && _excludeFee[sender] != 1){    //Check whether the recipient address is_ SwapAddr address (for sale) and sender is a non transaction fee free address
            uint jackpotFee = onePercent.mul(_feeRateMap[_jackpotPoolAddr]);    //2% of sales slip points are injected into the project prize pool
            uint clubFee = onePercent.mul(_feeRateMap[_clubRewardAddr]);    //1% into the club prize pool
            uint deadFee = onePercent.mul(_feeRateMap[_deadAddr]);    //2% hack into the black hole address and permanently destroy it
            uint groupFee = onePercent.mul(_feeRateMap[_groupAddr]); //1% allocated to the project team

            _totalReceiveFeeMap[_jackpotPoolAddr] = _totalReceiveFeeMap[_jackpotPoolAddr].add(jackpotFee);    //Update charges
            _totalReceiveFeeMap[_clubRewardAddr] = _totalReceiveFeeMap[_clubRewardAddr].add(clubFee);    //Update charges
            _totalReceiveFeeMap[_deadAddr] = _totalReceiveFeeMap[_deadAddr].add(deadFee); //Fees for updating authorization
            _totalReceiveFeeMap[_groupAddr] = _totalReceiveFeeMap[_groupAddr].add(groupFee);    //Update charges

            _balances[_jackpotPoolAddr] = _balances[_jackpotPoolAddr].add(jackpotFee); //Update asset quantity
            _balances[_clubRewardAddr] = _balances[_clubRewardAddr].add(clubFee); //Update asset quantity
            _balances[_deadAddr] = _balances[_deadAddr].add(deadFee);    //Update asset quantity
            _balances[_groupAddr] = _balances[_groupAddr].add(groupFee);    //Update asset quantity

            uint currentFee = jackpotFee.add(clubFee).add(deadFee).add(groupFee);    //Calculate the total amount of fee assets charged
            _totalFee = _totalFee.add(currentFee); //Calculate total cost
            addValue = addValue.sub(currentFee); //Update the actual number of assets to be transferred after expenses are removed

            _totalSupply -= deadFee;    //Update total (remove destroyed assets)
            _balances[recipient] = _balances[recipient].add(addValue);    //Update the number of assets held by recipient address
            emit Transfer(sender, recipient, amount);
            return;
        }

        if(_excludeFee[sender] != 1 && _excludeFee[recipient] != 1){    //If the sender is not a free
```

```
transaction address and the recipient is not a free transaction address
                uint transferFee = onePercent.mul(_feeRateMap[_transferFeeReceiveAddr]);
//Calculation of transaction fee
                _totalReceiveFeeMap[_transferFeeReceiveAddr] =
_totalReceiveFeeMap[_transferFeeReceiveAddr].add(transferFee);//Update the total fees charged
                _balances[_transferFeeReceiveAddr] =
_balances[_transferFeeReceiveAddr].add(transferFee); //Update total expense quantity
                _totalFee = _totalFee.add(transferFee); //Calculate total cost
                addValue = addValue.sub(transferFee); //Calculate the amount of the actual transfer

                _balances[recipient] = _balances[recipient].add(addValue); //Update transfer amount
                emit Transfer(sender, recipient, amount);
                return;
            }

            _balances[recipient] = _balances[recipient].add(addValue); //Direct update amount (0%
purchase slip point)
            emit Transfer(sender, recipient, amount);

        }
```

Safety advice：NONE.

9. Appendix:Analysis tools

9.1.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

9.2.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

9.3.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

9.4.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

9.5.Mythril

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

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

9.6.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

10. 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.