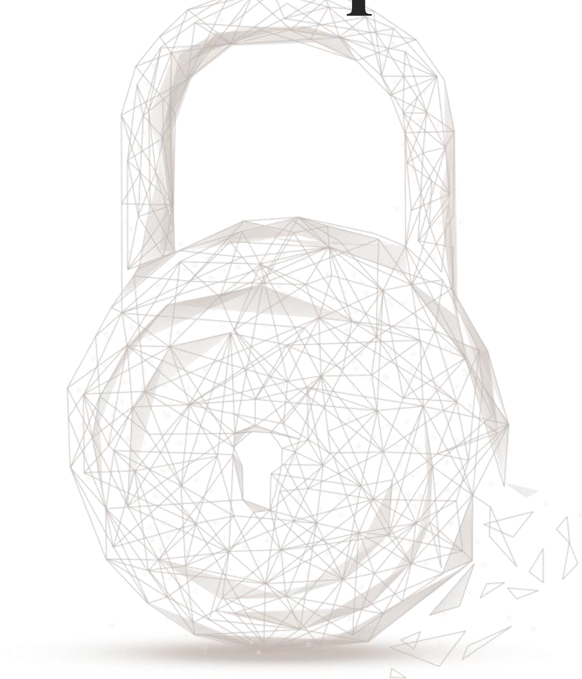
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number: 202107251338

Report Query Name: JGN Defi

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
dJGNToken	0xda656c9dDe4Ae956D50D67f98A437BCd269cde4d	https://bscscan.com/address/0xda656c9dDe4Ae956D50D67f98A437BCd269cde4d#code
Airdrop	0x9fdABF94e4231f7a72CA25F1425AbF3cB318e2C4	https://bscscan.com/address/0x9fdABF94e4231f7a72CA25F1425AbF3cB318e2C4#code

Start Date: 2021.07.23

Completion Date: 2021.07.25

Overall Result: Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass

BEOSIN Blockchain Security

		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Disclaimer: This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project. This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts project JGN Defi, including Coding Standards, Security, and Business Logic. **The JGN Defi project passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

Audit Contents:

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing BNB.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

Check whether the business is secure.

3.1 Business analysis of Contract dJGNToken

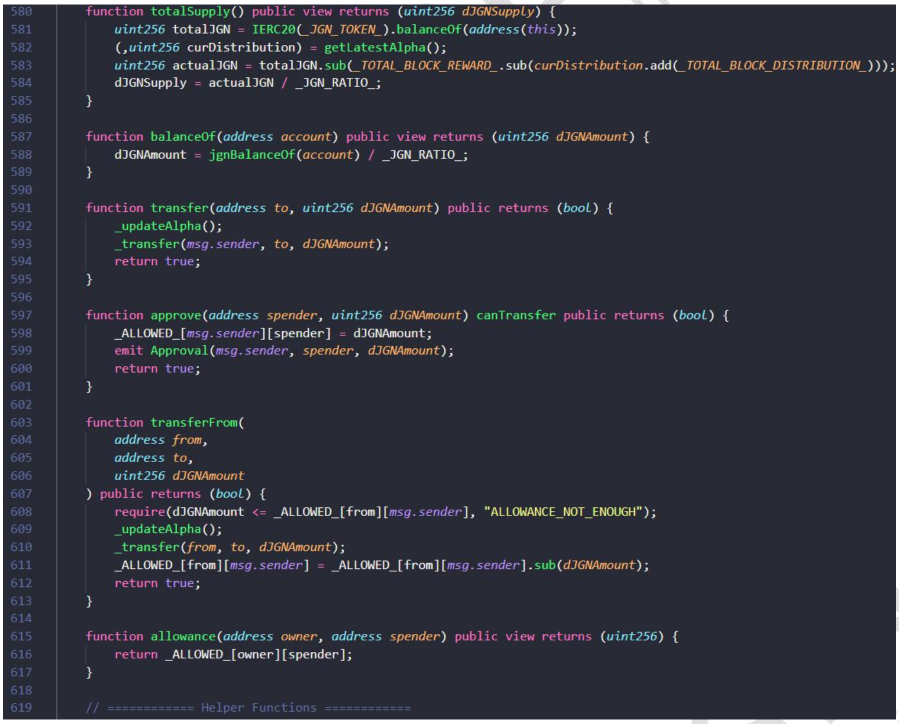
(1) Basic Token Information

Token name	dJGN Membership Token
Token symbol	dJGN
decimals	18
totalSupply	The initial supply is 0
Token type	BEP-20

Table 1 Basic Token Information

(2) BEP-20 Token Standard Functions

- Description: The token contract implements a token which conforms to the BEP-20 Standards. It should be noted that the user can directly call the *approve* function to set the approval value for the specified address, but in order to avoid multiple authorizations, it is recommended that the user resets the authorization value to 0 when calling this function to change the authorization value. Token transfer related functions are only available when the related status is true.



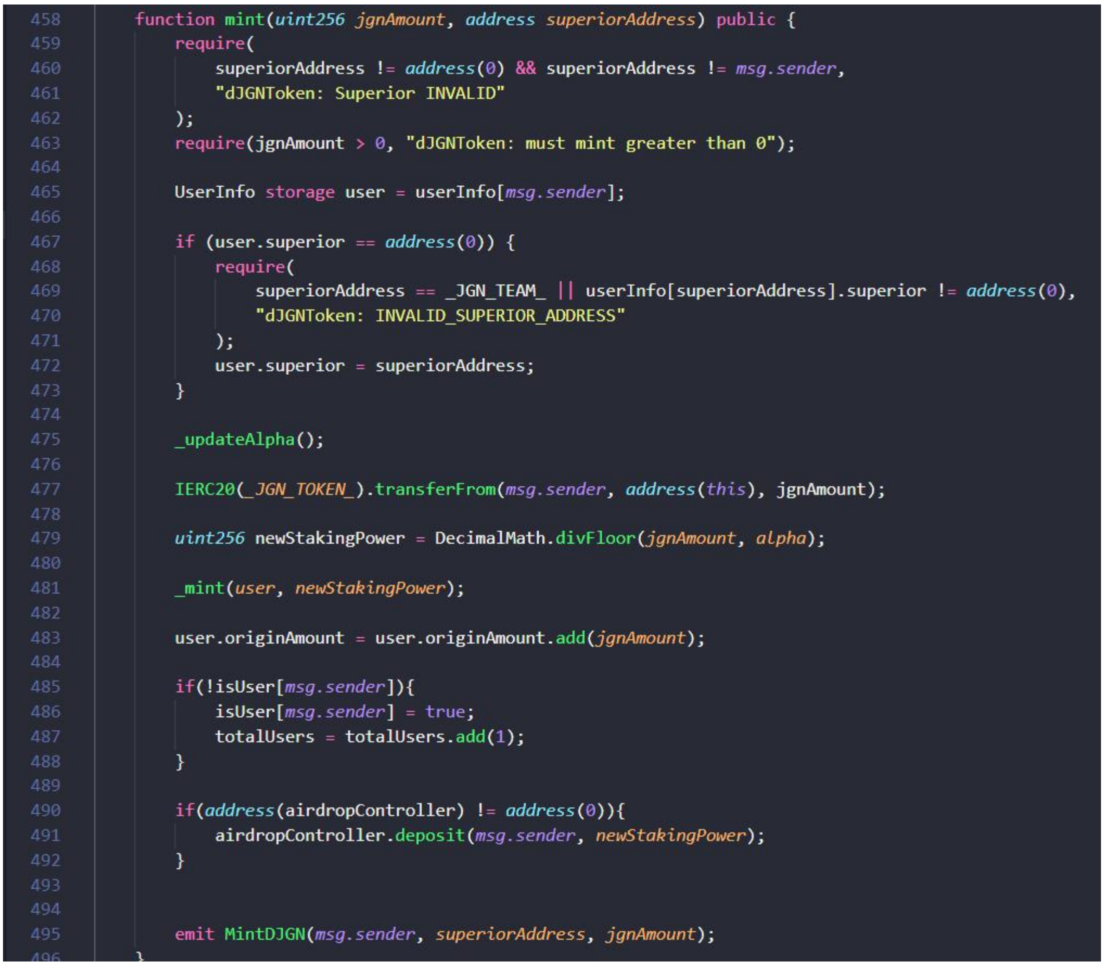
```
    function totalSupply() public view returns (uint256 dJGNSupply) {
        uint256 totalJGN = IERC20(_JGN_TOKEN_).balanceOf(address(this));
        (,uint256 curDistribution) = getLatestAlpha();
        uint256 actualJGN = totalJGN.sub(_TOTAL_BLOCK_REWARD_.sub(curDistribution.add(_TOTAL_BLOCK_DISTRIBUTION_)));
        dJGNSupply = actualJGN / _JGN_RATIO_;
    }

    function balanceOf(address account) public view returns (uint256 dJGNAmount) {
        dJGNAmount = jgnBalanceOf(account) / _JGN_RATIO_;
    }

    function transfer(address to, uint256 dJGNAmount) public returns (bool) {
        _updateAlpha();
        _transfer(msg.sender, to, dJGNAmount);
        return true;
    }

    function approve(address spender, uint256 dJGNAmount) canTransfer public returns (bool) {
        _ALLOWED_[msg.sender][spender] = dJGNAmount;
        emit Approval(msg.sender, spender, dJGNAmount);
        return true;
    }

    function transferFrom(
        address from,
        address to,
        uint256 dJGNAmount
    ) public returns (bool) {
        require(dJGNAmount <= _ALLOWED_[from][msg.sender], "ALLOWANCE_NOT_ENOUGH");
        _updateAlpha();
        _transfer(from, to, dJGNAmount);
        _ALLOWED_[from][msg.sender] = _ALLOWED_[from][msg.sender].sub(dJGNAmount);
        return true;
    }

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

    // =========== Helper Functions ===========
```

Figure 1 source code of BEP-20 functions

- Related functions: *name, symbol, decimals, totalSupply, balanceOf, allowance, transfer, transferFrom, approve*
- Result: Pass

(3) mint function

- Description: The contract implements the *mint* function for user participation in staking mining (requires pre-authorization of this contract). The first call to this function will carry out the registration of the user address, the superior address cannot be 0 and the caller itself, and the staking amount needs to be greater than 0; the internal function *_updateAlpha* will be called before the collateral to update the relevant data, and *_mint* will be called after the collateral to update the relevant data of superior address. If the airdropController address is not 0, the *deposit* function in the airdrop contract will be executed to update the airdrop reward related parameters.



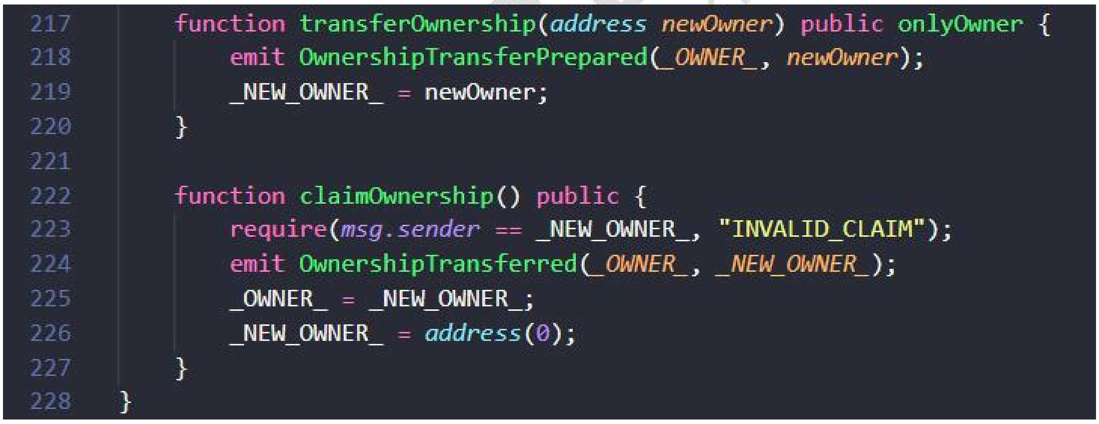
```
    function mint(uint256 jgnAmount, address superiorAddress) public {
        require(
            superiorAddress != address(0) && superiorAddress != msg.sender,
            "dJGNToken: Superior INVALID"
        );
        require(jgnAmount > 0, "dJGNToken: must mint greater than 0");

        UserInfo storage user = userInfo[msg.sender];

        if (user.superior == address(0)) {
            require(
                superiorAddress == _JGN_TEAM_ || userInfo[superiorAddress].superior != address(0),
                "dJGNToken: INVALID_SUPERIOR_ADDRESS"
            );
            user.superior = superiorAddress;
        }

        _updateAlpha();

        IERC20(_JGN_TOKEN_).transferFrom(msg.sender, address(this), jgnAmount);

        uint256 newStakingPower = DecimalMath.divFloor(jgnAmount, alpha);

        _mint(user, newStakingPower);

        user.originAmount = user.originAmount.add(jgnAmount);

        if(!isUser[msg.sender]){
            isUser[msg.sender] = true;
            totalUsers = totalUsers.add(1);
        }

        if(address(airdropController) != address(0)){
            airdropController.deposit(msg.sender, newStakingPower);
        }


        emit MintDJGN(msg.sender, superiorAddress, jgnAmount);
    }
```

Figure 2 source code of *mint*

- Related functions: *mint, transferFrom, deposit*
- Result: Pass

(4) Ownership

- Description: The contract implements *transferOwnership* and *claimOwnership* functions to manage the contract's ownership. *transferOwnership* is used to set the newOwner address and can only be called by the current owner of the contract; The *claimOwnership* function can be called only by the current newOwner to receive the ownership and reset the newOwner address to 0.



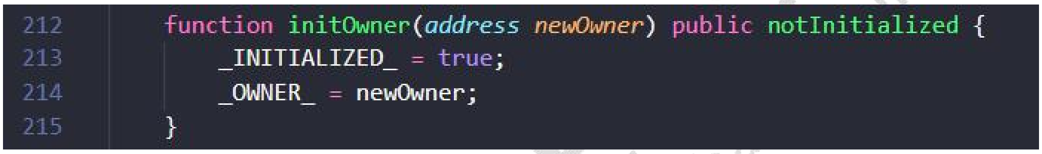
```
    function transferOwnership(address newOwner) public onlyOwner {
        emit OwnershipTransferPrepared(_OWNER_, newOwner);
        _NEW_OWNER_ = newOwner;
    }

    function claimOwnership() public {
        require(msg.sender == _NEW_OWNER_, "INVALID_CLAIM");
        emit OwnershipTransferred(_OWNER_, _NEW_OWNER_);
        _OWNER_ = _NEW_OWNER_;
        _NEW_OWNER_ = address(0);
    }
}
```

Figure 3 source code of *transferOwnership* and *claimOwnership*

- Related functions: *transferOwnership, claimOwnership*
- Result: Pass

(5) Initialize owner

- Description: The contract implements the *initOwner* function to initialize the owner after the contract is deployed and can only be called once. It is recommended to call the contract immediately after it is deployed.



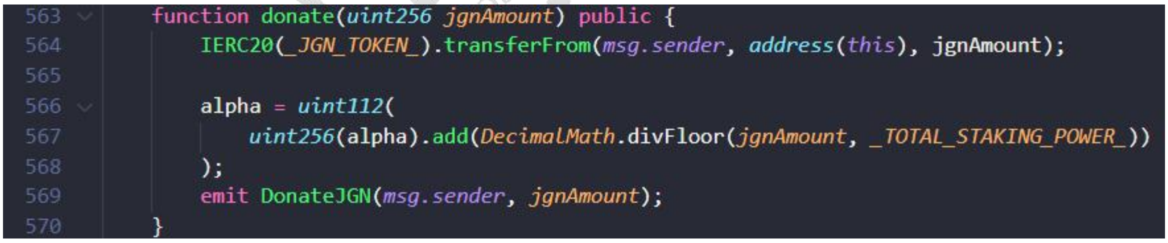
```
    function initOwner(address newOwner) public notInitialized {
        _INITIALIZED_ = true;
        _OWNER_ = newOwner;
    }
```

Figure 4 source code of *initOwner*

- Related functions: *initOwner*
- Result: Pass

(6) Donate

- Description: The contract implements the *donate* function for users to donate tokens to the contract, which will update the value of alpha.



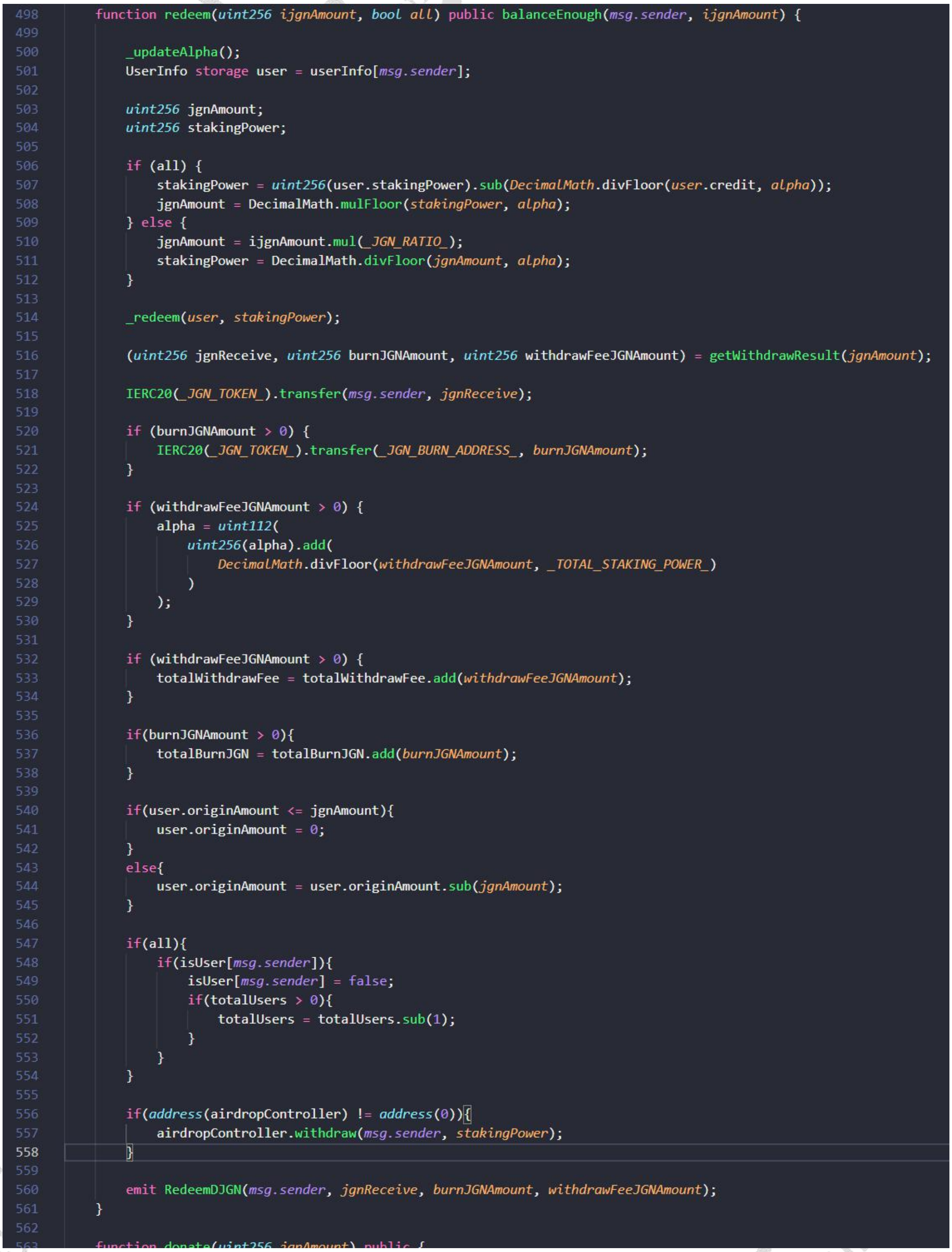
```
    function donate(uint256 jgnAmount) public {
        IERC20(_JGN_TOKEN_).transferFrom(msg.sender, address(this), jgnAmount);

        alpha = uint112(
            uint256(alpha).add(DecimalMath.divFloor(jgnAmount, _TOTAL_STAKING_POWER_))
        );
        emit DonateJGN(msg.sender, jgnAmount);
    }
```

Figure 5 source code of *donate*

- Related functions: *donate*
- Result: Pass

(7) Redeem

- Description: The contract implements the *redeem* function for the user to withdraw the pledged JGN tokens. Before the withdrawal, the internal function *_updateAlpha* is called to update the relevant data, determine whether the user is withdrawing all, call the internal function *_redeem* to update the information about the superior address. Then calculate the actual withdrawal amount, whether destruction and transaction fees are incurred, and make the relevant transfer. If the user withdraws all, the user's identity will be cancelled. If the airdropController address is not 0, the *withdraw* function in the airdrop contract will be executed to update the airdrop reward related parameters.



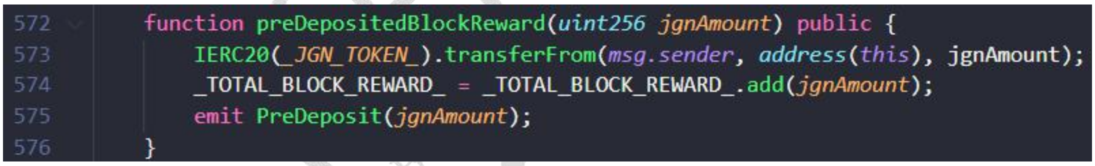
```
    function redeem(uint256 ijgnAmount, bool all) public balanceEnough(msg.sender, ijgnAmount) {

        _updateAlpha();
        UserInfo storage user = userInfo[msg.sender];

        uint256 jgnAmount;
        uint256 stakingPower;

        if (all) {
            stakingPower = uint256(user.stakingPower).sub(DecimalMath.divFloor(user.credit, alpha));
            jgnAmount = DecimalMath.mulFloor(stakingPower, alpha);
        } else {
            jgnAmount = ijgnAmount.mul(_JGN_RATIO_);
            stakingPower = DecimalMath.divFloor(jgnAmount, alpha);
        }

        _redeem(user, stakingPower);

        (uint256 jgnReceive, uint256 burnJGNAmount, uint256 withdrawFeeJGNAmount) = getWithdrawResult(jgnAmount);

        IERC20(_JGN_TOKEN_).transfer(msg.sender, jgnReceive);

        if (burnJGNAmount > 0) {
            IERC20(_JGN_TOKEN_).transfer(_JGN_BURN_ADDRESS_, burnJGNAmount);
        }

        if (withdrawFeeJGNAmount > 0) {
            alpha = uint112(
                uint256(alpha).add(
                    DecimalMath.divFloor(withdrawFeeJGNAmount, _TOTAL_STAKING_POWER_)
                )
            );
        }

        if (withdrawFeeJGNAmount > 0) {
            totalWithdrawFee = totalWithdrawFee.add(withdrawFeeJGNAmount);
        }

        if(burnJGNAmount > 0){
            totalBurnJGN = totalBurnJGN.add(burnJGNAmount);
        }

        if(user.originAmount <= jgnAmount){
            user.originAmount = 0;
        }
        else{
            user.originAmount = user.originAmount.sub(jgnAmount);
        }

        if(all){
            if(isUser[msg.sender]){
                isUser[msg.sender] = false;
                if(totalUsers > 0){
                    totalUsers = totalUsers.sub(1);
                }
            }
        }

        if(address(airdropController) != address(0)){
            airdropController.withdraw(msg.sender, stakingPower);
        }

        emit RedeemDJGN(msg.sender, jgnReceive, burnJGNAmount, withdrawFeeJGNAmount);
    }

    function donate(uint256 jgnAmount) public {
```

Figure 6 source code of *redeem*

- Related functions: *redeem, withdraw*
- Result: Pass

(8) Pre-deposit

- Description: The contract implements a *preDepositedBlockReward* for users to send JGN tokens as reward, this part of JGN tokens will not enter into dJGN related calculations and cannot be withdrawn.



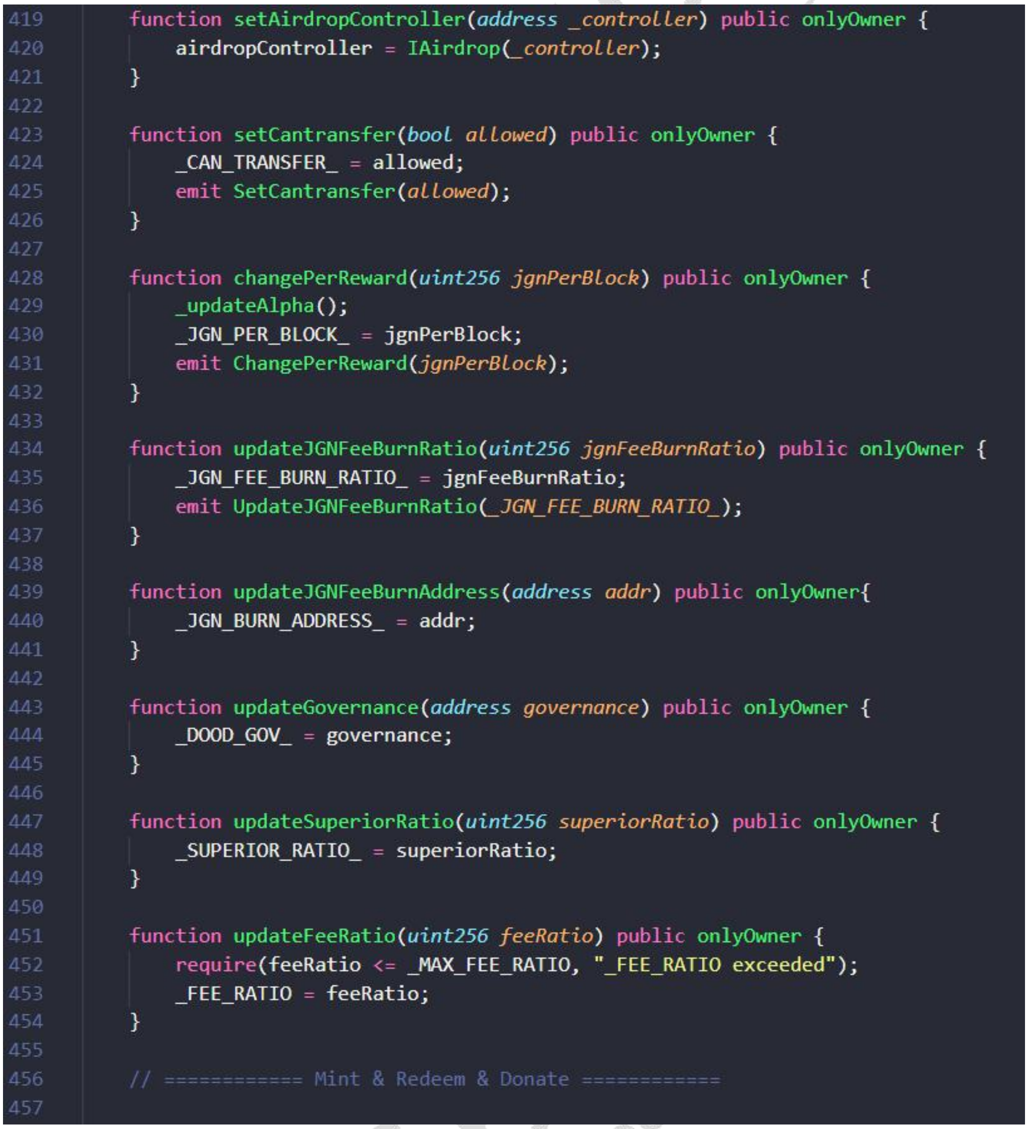
```
    function preDepositedBlockReward(uint256 jgnAmount) public {
        IERC20(_JGN_TOKEN_).transferFrom(msg.sender, address(this), jgnAmount);
        _TOTAL_BLOCK_REWARD_ = _TOTAL_BLOCK_REWARD_.add(jgnAmount);
        emit PreDeposit(jgnAmount);
    }
```

Figure 7 source code of *preDepositedBlockReward*

- Related functions: *preDepositedBlockReward*
- Result: Pass

(9) Contract parameter setting functions

- Description: The contract implements the following functions that only the contract owner can call: The *setAirdropController* function is used to set the address of the airdropController contract; *setCantransfer* to set whether dJGN transfers are allowed; *changePerReward* to change _JGN_PER_BLOCK_; *updateJGNFeeBurnRatio* to change the rate of the destruction fee. *updateJGNFeeBurnAddress* for setting the address to receive tokens when they are destroyed; *updateGovernance* for setting _DOOD_GOV_; *updateSuperiorRatio* for setting the rate of reward for superior addresses; *updateFeeRatio* for setting the rate of transaction fees.



```
    function setAirdropController(address _controller) public onlyOwner {
        airdropController = IAirdrop(_controller);
    }

    function setCantransfer(bool allowed) public onlyOwner {
        _CAN_TRANSFER_ = allowed;
        emit SetCantransfer(allowed);
    }

    function changePerReward(uint256 jgnPerBlock) public onlyOwner {
        _updateAlpha();
        _JGN_PER_BLOCK_ = jgnPerBlock;
        emit ChangePerReward(jgnPerBlock);
    }

    function updateJGNFeeBurnRatio(uint256 jgnFeeBurnRatio) public onlyOwner {
        _JGN_FEE_BURN_RATIO_ = jgnFeeBurnRatio;
        emit UpdateJGNFeeBurnRatio(_JGN_FEE_BURN_RATIO_);
    }

    function updateJGNFeeBurnAddress(address addr) public onlyOwner{
        _JGN_BURN_ADDRESS_ = addr;
    }

    function updateGovernance(address governance) public onlyOwner {
        _DOOD_GOV_ = governance;
    }

    function updateSuperiorRatio(uint256 superiorRatio) public onlyOwner {
        _SUPERIOR_RATIO_ = superiorRatio;
    }

    function updateFeeRatio(uint256 feeRatio) public onlyOwner {
        require(feeRatio <= _MAX_FEE_RATIO, "_FEE_RATIO exceeded");
        _FEE_RATIO = feeRatio;
    }

    // ============ Mint & Redeem & Donate ============

```

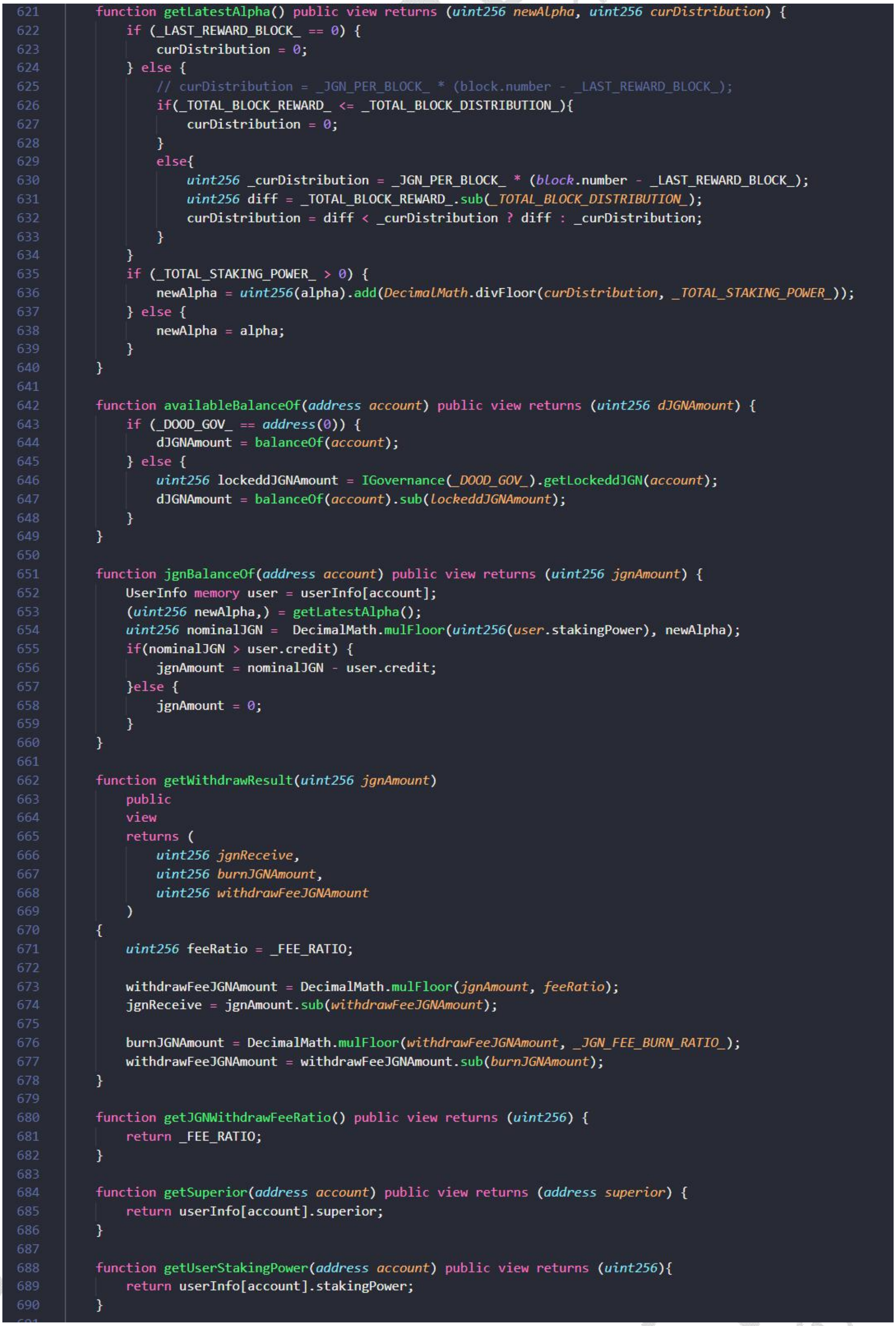
Figure 8 source code of Ownable functions

- Related functions: *setAirdropController, setCantransfer, changePerReward, updateJGNFeeBurnRatio, updateJGNFeeBurnAddress, updateGovernance, updateSuperiorRatio, updateFeeRatio*
- Result: Pass

(10) Related parameter query function

- Description: The contract implements *getLatestAlpha* function to query the latest alpha value; *availableBalanceOf* function to query the available balance of the specified address; *JGNBalanceOf* function to calculate the number of JGN tokens pledged to the contract from the specified address; *getWithdrawResult* function to calculate the actual withdrawal amount based on the input amount; *getJGNWithdrawFeeRatio* function to query the fee ratio of the JGN tokens withdrawn from the specified address; *getSuperior* function to query the superior address; *getWithdrawResult* function is used to calculate the actual number of tokens withdrawn based on the amount entered; *getJGNWithdrawFeeRatio* function is used to query the fee rate for withdrawing JGN tokens;

getSuperior function is used to query the superior address of the specified address; The *getUserStakingPower* function is used to query the collateral power of the specified address.



```
    function getLatestAlpha() public view returns (uint256 newAlpha, uint256 curDistribution) {
        if (_LAST_REWARD_BLOCK_ == 0) {
            curDistribution = 0;
        } else {
            // curDistribution = _JGN_PER_BLOCK_ * (block.number - _LAST_REWARD_BLOCK_);
            if(_TOTAL_BLOCK_REWARD_ <= _TOTAL_BLOCK_DISTRIBUTION_){
                curDistribution = 0;
            }
            else{
                uint256 _curDistribution = _JGN_PER_BLOCK_ * (block.number - _LAST_REWARD_BLOCK_);
                uint256 diff = _TOTAL_BLOCK_REWARD_.sub(_TOTAL_BLOCK_DISTRIBUTION_);
                curDistribution = diff < _curDistribution ? diff : _curDistribution;
            }
        }
        if (_TOTAL_STAKING_POWER_ > 0) {
            newAlpha = uint256(alpha).add(DecimalMath.divFloor(curDistribution, _TOTAL_STAKING_POWER_));
        } else {
            newAlpha = alpha;
        }
    }

    function availableBalanceOf(address account) public view returns (uint256 dJGNAmount) {
        if (_DOOD_GOV_ == address(0)) {
            dJGNAmount = balanceOf(account);
        } else {
            uint256 lockeddJGNAmount = IGovernance(_DOOD_GOV_).getLockeddJGN(account);
            dJGNAmount = balanceOf(account).sub(lockeddJGNAmount);
        }
    }

    function jgnBalanceOf(address account) public view returns (uint256 jgnAmount) {
        UserInfo memory user = userInfo[account];
        (uint256 newAlpha,) = getLatestAlpha();
        uint256 nominalJGN =  DecimalMath.mulFloor(uint256(user.stakingPower), newAlpha);
        if(nominalJGN > user.credit) {
            jgnAmount = nominalJGN - user.credit;
        }else {
            jgnAmount = 0;
        }
    }

    function getWithdrawResult(uint256 jgnAmount)
        public
        view
        returns (
            uint256 jgnReceive,
            uint256 burnJGNAmount,
            uint256 withdrawFeeJGNAmount
        )
    {
        uint256 feeRatio = _FEE_RATIO;

        withdrawFeeJGNAmount = DecimalMath.mulFloor(jgnAmount, feeRatio);
        jgnReceive = jgnAmount.sub(withdrawFeeJGNAmount);

        burnJGNAmount = DecimalMath.mulFloor(withdrawFeeJGNAmount, _JGN_FEE_BURN_RATIO_);
        withdrawFeeJGNAmount = withdrawFeeJGNAmount.sub(burnJGNAmount);
    }

    function getJGNWithdrawFeeRatio() public view returns (uint256) {
        return _FEE_RATIO;
    }

    function getSuperior(address account) public view returns (address superior) {
        return userInfo[account].superior;
    }

    function getUserStakingPower(address account) public view returns (uint256){
        return userInfo[account].stakingPower;
    }
```

Figure 9 source code of query functions

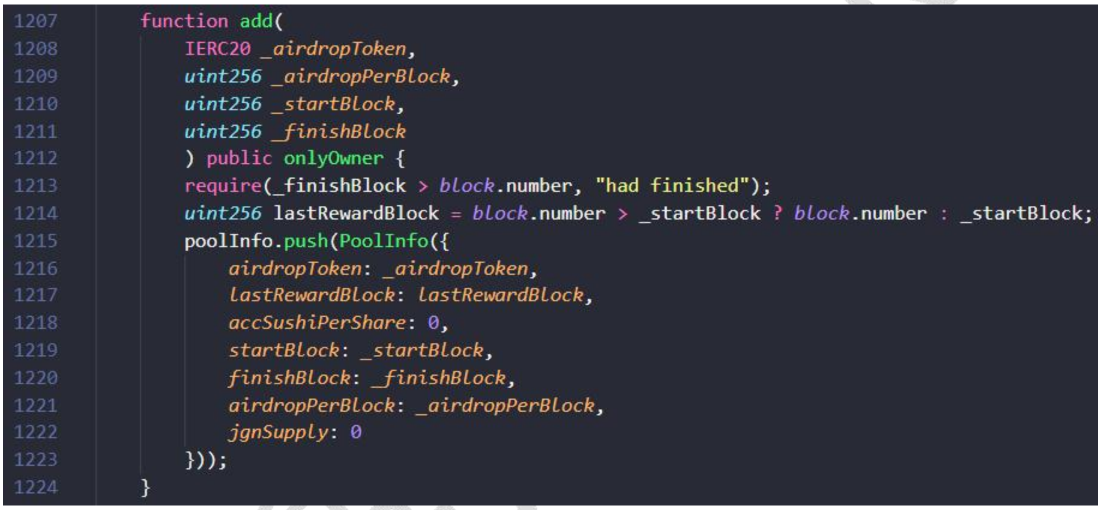
- Related functions: *getLatestAlpha, availableBalanceOf, JGNBalanceOf, getWithdrawResult, getJGNWithdrawFeeRatio, getSuperior, getUserStakingPower*
- Result: Pass

3.2 Business analysis of Contract Token Airdrop

dJGN's collateral arithmetic varies according to its holdings. dJGN token species only *mint* and *redeem* functions update the user airdrop reward calculations in the Airdrop contract. If the dJGN token is opened for transfer, the receiving address can update the data related to the airdrop reward through functions such as *syncdJGN* to get the airdrop reward; however, the data related to the airdrop reward in Airdrop for the transferring address will not be updated and can continue to maintain the same yield as before the transfer. (i.e. the dJGN token holdings decrease while the reward remains unchanged) The project owner declares that dJGN transfers will not be activated and that if they are, the relevant airdrop contract will be voided.

(1) add function

- Description: The contract implements the *add* function for the contract's owner to add new airdrop tokens and set airdrop reward related parameters. Note: Adding duplicate airdrop tokens will cause the reward to be calculated incorrectly, so administrators should be careful to prevent duplicate additions.



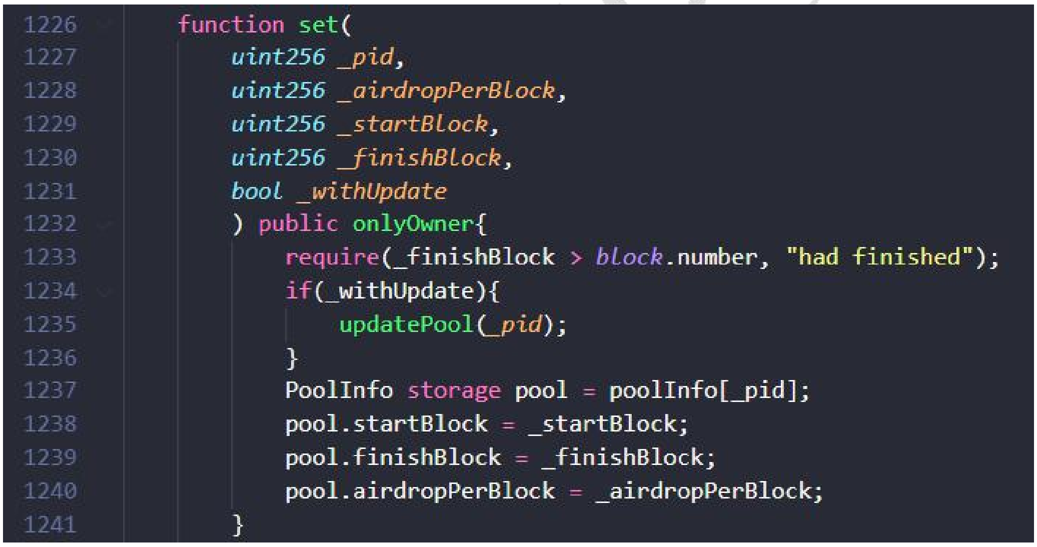
```
    function add(
        IERC20 _airdropToken,
        uint256 _airdropPerBlock,
        uint256 _startBlock,
        uint256 _finishBlock
        ) public onlyOwner {
        require(_finishBlock > block.number, "had finished");
        uint256 lastRewardBlock = block.number > _startBlock ? block.number : _startBlock;
        poolInfo.push(PoolInfo({
            airdropToken: _airdropToken,
            lastRewardBlock: lastRewardBlock,
            accSushiPerShare: 0,
            startBlock: _startBlock,
            finishBlock: _finishBlock,
            airdropPerBlock: _airdropPerBlock,
            jgnSupply: 0
        }));
    }
```

Figure 10 source code of *add* function

- Related functions: *add*
- Result: Pass

(2) set function

- Description: The contract implements the *set* function for the owner of the contract to modify the parameters related to the airdrop token rewards for the specified id, optionally executing the *updatePool* function to update the rewards related data before the modification.



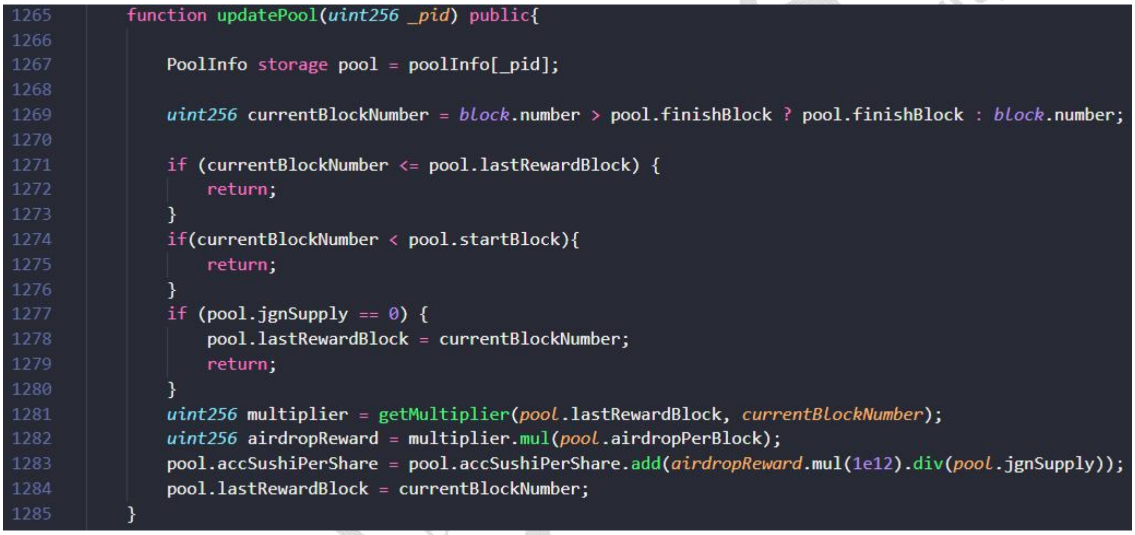
```
    function set(
        uint256 _pid,
        uint256 _airdropPerBlock,
        uint256 _startBlock,
        uint256 _finishBlock,
        bool _withUpdate
        ) public onlyOwner{
            require(_finishBlock > block.number, "had finished");
            if(_withUpdate){
                updatePool(_pid);
            }
            PoolInfo storage pool = poolInfo[_pid];
            pool.startBlock = _startBlock;
            pool.finishBlock = _finishBlock;
            pool.airdropPerBlock = _airdropPerBlock;
        }
```

Figure 11 source code of *set* function

- Related functions: *set, updatePool*
- Result: Pass

(3) updatePool function

- Description: The contract implements *updatePool* function to update the data related to the airdrop token rewards for the specified id.



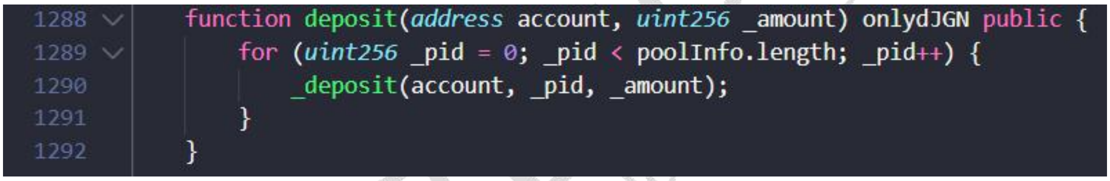
```
    function updatePool(uint256 _pid) public{

        PoolInfo storage pool = poolInfo[_pid];

        uint256 currentBlockNumber = block.number > pool.finishBlock ? pool.finishBlock : block.number;

        if (currentBlockNumber <= pool.lastRewardBlock) {
            return;
        }
        if(currentBlockNumber < pool.startBlock){
            return;
        }
        if (pool.jgnSupply == 0) {
            pool.lastRewardBlock = currentBlockNumber;
            return;
        }
        uint256 multiplier = getMultiplier(pool.lastRewardBlock, currentBlockNumber);
        uint256 airdropReward = multiplier.mul(pool.airdropPerBlock);
        pool.accSushiPerShare = pool.accSushiPerShare.add(airdropReward.mul(1e12).div(pool.jgnSupply));
        pool.lastRewardBlock = currentBlockNumber;
    }
```

Figure 12 source code of *updatePool* function

- Related functions: *updatePool, getMultiplier*
- Result: Pass

(4) deposit function

- Description: The contract implements the *deposit* function to update all the user's drop reward related data (increasing the user's calculation), by calling the internal function *_deposit,* and the updatePool is executed to update the airdrop token data before increasing. If the user's calculated amount is not 0, the previous airdrop rewards are calculated and sent. Only dJGN token contract addresses can be called.



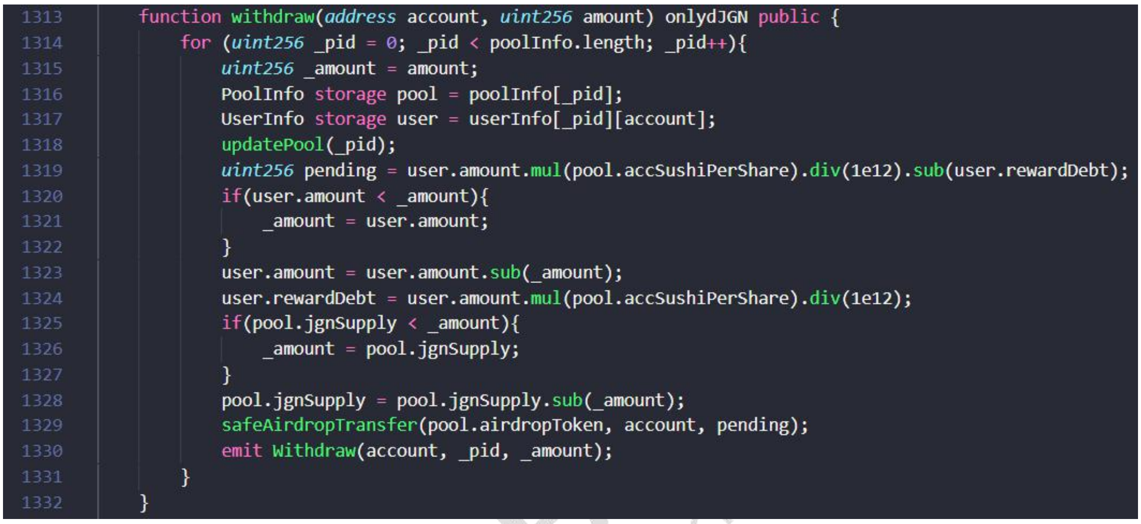
```
function deposit(address account, uint256 _amount) onlydJGN public {
    for (uint256 _pid = 0; _pid < poolInfo.length; _pid++) {
        _deposit(account, _pid, _amount);
    }
}
```

Figure 13 source code of *deposit* function

- Related functions: *deposit, updatePool, safeAirdropTransfer*
- Result: Pass

(5) withdraw function

- Description: The contract implements the *withdraw* function to update all the user's airdrop reward data (reducing the amount of calculations for the user), before reducing the *updatePool* to update the airdrop token data. If the user's calculated amount is not 0, the previous airdrop rewards are calculated and sent. Only dJGN token contract addresses can be called.



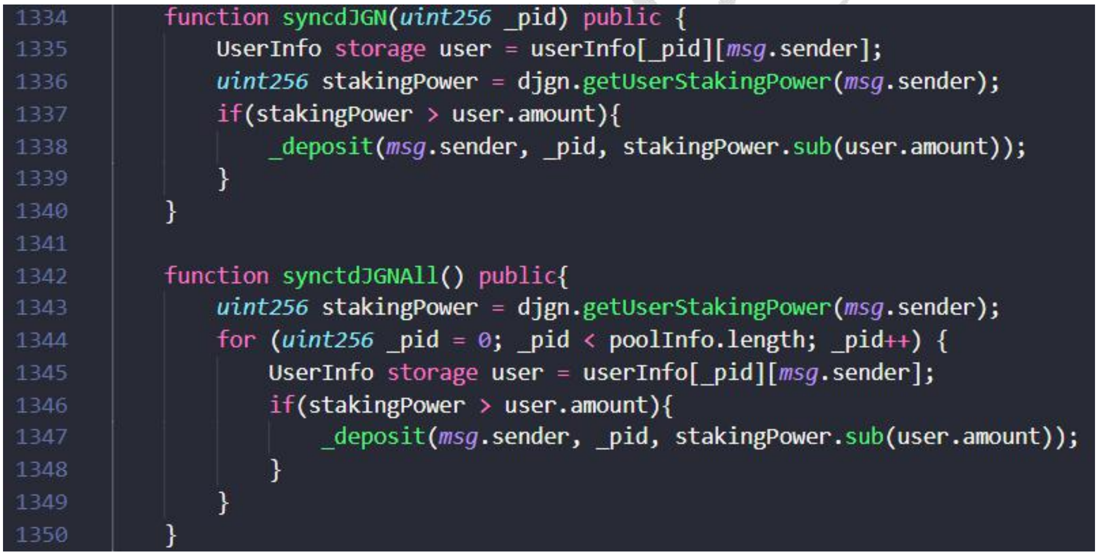
```
function withdraw(address account, uint256 amount) onlydJGN public {
    for (uint256 _pid = 0; _pid < poolInfo.length; _pid++){
        uint256 _amount = amount;
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][account];
        updatePool(_pid);
        uint256 pending = user.amount.mul(pool.accSushiPerShare).div(1e12).sub(user.rewardDebt);
        if(user.amount < _amount){
            _amount = user.amount;
        }
        user.amount = user.amount.sub(_amount);
        user.rewardDebt = user.amount.mul(pool.accSushiPerShare).div(1e12);
        if(pool.jgnSupply < _amount){
            _amount = pool.jgnSupply;
        }
        pool.jgnSupply = pool.jgnSupply.sub(_amount);
        safeAirdropTransfer(pool.airdropToken, account, pending);
        emit Withdraw(account, _pid, _amount);
    }
}
```

Figure 14 source code of *withdraw* function

- Related functions: *withdraw, updatePool, safeAirdropTransfer*
- Result: Pass

(6) sync functions

- Description: The contract implements the *syncdJGN* function for the user to update the reward-related data for their specified airdrop tokens, calling the internal function *_deposit* to update when the user's dJGN collateral arithmetic exceeds the amount of calculations for the specified airdrop tokens. *synctdJGNAll* function for the user to update the reward-related data for all their airdrop tokens, traversing all airdrop tokens and updating only dJGN collateral arithmetic exceeds the computed amount of the corresponding airdrop token.



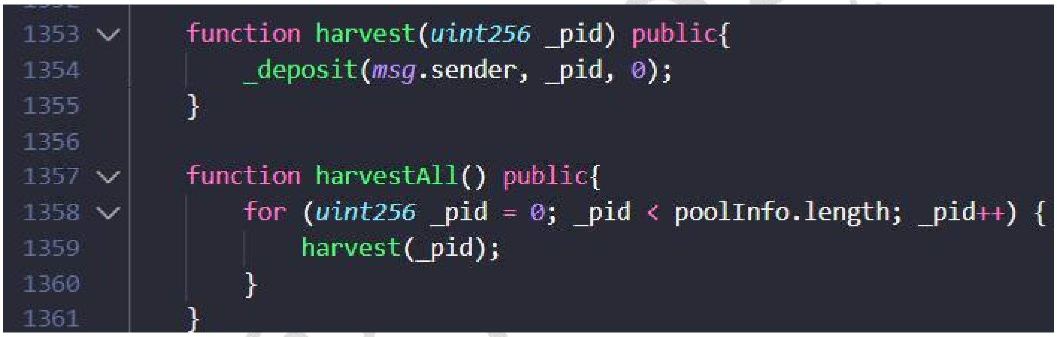
```
    function syncdJGN(uint256 _pid) public {
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 stakingPower = djgn.getUserStakingPower(msg.sender);
        if(stakingPower > user.amount){
            _deposit(msg.sender, _pid, stakingPower.sub(user.amount));
        }
    }

    function synctdJGNAll() public{
        uint256 stakingPower = djgn.getUserStakingPower(msg.sender);
        for (uint256 _pid = 0; _pid < poolInfo.length; _pid++) {
            UserInfo storage user = userInfo[_pid][msg.sender];
            if(stakingPower > user.amount){
                _deposit(msg.sender, _pid, stakingPower.sub(user.amount));
            }
        }
    }
```

Figure 15 source code of sync functions

- Related functions: *syncdJGN, synctdJGNAll, getUserStakingPower*
- Result: Pass

(7) harvest functions

- Description: The contract implements the *harvest* function for the user to receive the airdrop reward for the specified airdrop token, implemented by calling the internal function *_deposit*. The *harvestAll* function is used for the user to receive the airdrop reward for all airdrop tokens.



```
    function harvest(uint256 _pid) public{
        _deposit(msg.sender, _pid, 0);
    }

    function harvestAll() public{
        for (uint256 _pid = 0; _pid < poolInfo.length; _pid++) {
            harvest(_pid);
        }
    }
```

Figure 16 source code of harvest functions

- Related functions: *harvest, harvestAll*
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project JGN Defi. The problems found by the audit team during the audit process have been notified to the project party and reached an agreement on the repair results, the overall audit result of the JGN Defi project's smart contract is **Pass**.

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com