Diamond JGN (dJGN)

Smart Contract Audit

Diamond JGN (dJGN) Audit Summary

Project name : Diamond JGN (dJGN) Contract

Project address: None

Code URL : https://bscscan.com/address/0xda656c9dDe4Ae956D50D67f98A437BCd269cde4d#code

Commit : None

Project target : Diamond JGN (dJGN) Contract Audit

Blockchain : Binance Smart Chain（BSC）

Test result : PASSED

Audit Info

Audit NO : 0X202108040009

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

Diamond JGN (dJGN) Audit

The Diamond JGN (dJGN) team asked us to review and audit their Diamond JGN (dJGN) contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
Diamond JGN (dJGN) Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-08-04

Audit results

Note:

1. The available balance of users in redemption and trading is affected by the interface contract. Currently, the address is 0
2. There are fees, interest rates and other related Settings

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Diamond JGN (dJGN) contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
dJGNToken.sol	f6f3b708c8095a4174d414e20f254b45

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe

Vulnerability	status
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

```
/**
 *Submitted for verification at BscScan.com on 2021-07-19
*/

// File: contracts/intf/IERC20.sol

// This is a file copied from https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/cont
// SPDX-License-Identifier: MIT

pragma solidity 0.6.12;
pragma experimental ABIEncoderV2;


/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    function decimals() external view returns (uint8);

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

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

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

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
```

```
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

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

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

// File: contracts/lib/SafeMath.sol

/**
 * @title SafeMath
 * @author JGN
 *
 * @notice Math operations with safety checks that revert on error
 */
library SafeMath {
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

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

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "DIVIDING_ERROR");
        return a / b;
```

```
    }

    function divCeil(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 quotient = div(a, b);
        uint256 remainder = a - quotient * b;
        if (remainder > 0) {
            return quotient + 1;
        } else {
            return quotient;
        }
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SUB_ERROR");
        return a - b;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "ADD_ERROR");
        return c;
    }

    function sqrt(uint256 x) internal pure returns (uint256 y) {
        uint256 z = x / 2 + 1;
        y = x;
        while (z < y) {
            y = z;
            z = (x / z + z) / 2;
        }
    }
}

// File: contracts/lib/DecimalMath.sol

/**
 * @title DecimalMath
 * @author JGN
 *
 * @notice Functions for fixed point number with 18 decimals
 */
library DecimalMath {
    using SafeMath for uint256;

    uint256 internal constant ONE = 10**18;
    uint256 internal constant ONE2 = 10**36;

    function mulFloor(uint256 target, uint256 d) internal pure returns (uint256) {
        return target.mul(d) / (10**18);
    }

    function mulCeil(uint256 target, uint256 d) internal pure returns (uint256) {
        return target.mul(d).divCeil(10**18);
    }

    function divFloor(uint256 target, uint256 d) internal pure returns (uint256) {
        return target.mul(10**18).div(d);
    }

    function divCeil(uint256 target, uint256 d) internal pure returns (uint256) {
        return target.mul(10**18).divCeil(d);
    }

    function reciprocalFloor(uint256 target) internal pure returns (uint256) {
        return uint256(10**36).div(target);
    }
```

```
    function reciprocalCeil(uint256 target) internal pure returns (uint256) {
        return uint256(10**36).divCeil(target);
    }
}

// File: contracts/lib/InitializableOwnable.sol

/**
 * @title Ownable
 * @author JGN
 *
 * @notice Ownership related functions
 */
contract InitializableOwnable {
    address public _OWNER_;
    address public _NEW_OWNER_;
    bool internal _INITIALIZED_;

    // ============ Events ============

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

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

    // ============ Modifiers ============

    modifier notInitialized() {
        require(!_INITIALIZED_, "JGN_INITIALIZED");
        _;
    }

    modifier onlyOwner() {
        require(msg.sender == _OWNER_, "NOT_OWNER");
        _;
    }

    // ============ Functions ============

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

    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);
    }
}

// File: contracts/lib/SafeERC20.sol



/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
```

```
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;

    function safeTransfer(
        IERC20 token,
        address to,
        uint256 value
    ) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(
        IERC20 token,
        address from,
        address to,
        uint256 value
    ) internal {
        _callOptionalReturn(
            token,
            abi.encodeWithSelector(token.transferFrom.selector, from, to, value)
        );
    }

    function safeApprove(
        IERC20 token,
        address spender,
        uint256 value
    ) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require(
            (value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxin
     * on the return value: the return value is optional (but if data is returned, it must not be fal
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking m
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //  1. The target address is checked to verify it contains contract code
        //  2. The call itself is made, and success asserted
        //  3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) {
            // Return data is optional
            // solhint-disable-next-line max-line-length
```

```
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}

// File: contracts/JGNToken/dJGNToken.sol
interface IGovernance {
    function getLockeddJGN(address account) external view returns (uint256);
}


interface IAirdrop {
    function deposit(address account, uint256 _amount) external;
    function withdraw(address account, uint256 _amount) external;
}


// import "@nomiclabs/buidler/console.sol";


contract dJGNToken is InitializableOwnable {
    using SafeMath for uint256;

    // ============ Storage(ERC20) ============

    string public name = "dJGN Membership Token";
    string public symbol = "dJGN";
    uint8 public decimals = 18;
    mapping(address => mapping(address => uint256)) internal _ALLOWED_;

    // ============ Storage ============

    address public immutable _JGN_TOKEN_;
    address public immutable _JGN_TEAM_;
    address public _DOOD_GOV_;

    bool public _CAN_TRANSFER_;

    // staking reward parameters
    uint256 public _JGN_PER_BLOCK_;
    uint256 public _SUPERIOR_RATIO_ = 0;
    uint256 public constant _JGN_RATIO_ = 100;
    uint256 public _JGN_FEE_BURN_RATIO_;
    address public _JGN_BURN_ADDRESS_;

    uint256 public _FEE_RATIO = 5 * 10**16;
    uint256 public _MAX_FEE_RATIO = 20 * 10**16;

    // accounting
    uint112 public alpha = 10**18; // 1
    uint112 public _TOTAL_BLOCK_DISTRIBUTION_;
    uint32 public _LAST_REWARD_BLOCK_;

    uint256 public _TOTAL_BLOCK_REWARD_;
    uint256 public _TOTAL_STAKING_POWER_;
    mapping(address => UserInfo) public userInfo;

    uint256 public totalUsers;
    mapping(address => bool) public isUser;

    uint256 public totalWithdrawFee;
    uint256 public totalBurnJGN;

    struct UserInfo {
        uint128 stakingPower;
        uint128 superiorSP;
```

```
        address superior;
        uint256 credit;
        uint256 originAmount;
    }

    IAirdrop public airdropController;


    // ============ Events ============

    event MintDJGN(address user, address superior, uint256 mintJGN);
    event RedeemDJGN(address user, uint256 receiveJGN, uint256 burnJGN, uint256 feeJGN);
    event DonateJGN(address user, uint256 donateJGN);
    event SetCantransfer(bool allowed);

    event PreDeposit(uint256 jgnAmount);
    event ChangePerReward(uint256 jgnPerBlock);
    event UpdateJGNFeeBurnRatio(uint256 jgnFeeBurnRatio);

    event Transfer(address indexed from, address indexed to, uint256 amount);
    event Approval(address indexed owner, address indexed spender, uint256 amount);

    // ============ Modifiers ============

    modifier canTransfer() {
        require(_CAN_TRANSFER_, "dJGNToken: not allowed transfer");
        _;
    }

    modifier balanceEnough(address account, uint256 amount) {
        require(availableBalanceOf(account) >= amount, "dJGNToken: available amount not enough");
        _;
    }

    // ============ Constructor ============

    constructor(
        address jgnGov,
        address jgnToken,
        address jgnTeam
    ) public {
        _DOOD_GOV_ = jgnGov;
        _JGN_TOKEN_ = jgnToken;
        _JGN_TEAM_ = jgnTeam;
    }

    // ============ Ownable Functions ============`

    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 ============

    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);
    }

    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

        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 {
        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);
    }

    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);
    }

    // ============ ERC20 Functions ============

    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_DI
        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 ============

    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;
    }

    // ============ Internal Functions ============

    function _updateAlpha() internal {
        (uint256 newAlpha, uint256 curDistribution) = getLatestAlpha();
        uint256 newTotalDistribution = curDistribution.add(_TOTAL_BLOCK_DISTRIBUTION_);
        require(newAlpha <= uint112(-1) && newTotalDistribution <= uint112(-1), "OVERFLOW");
        alpha = uint112(newAlpha);
        _TOTAL_BLOCK_DISTRIBUTION_ = uint112(newTotalDistribution);
        _LAST_REWARD_BLOCK_ = uint32(block.number);
```

```
    }

    function _mint(UserInfo storage to, uint256 stakingPower) internal {
        require(stakingPower <= uint128(-1), "OVERFLOW");
        UserInfo storage superior = userInfo[to.superior];
        uint256 superiorIncreSP = DecimalMath.mulFloor(stakingPower, _SUPERIOR_RATIO_);
        uint256 superiorIncreCredit = DecimalMath.mulFloor(superiorIncreSP, alpha);

        to.stakingPower = uint128(uint256(to.stakingPower).add(stakingPower));
        to.superiorSP = uint128(uint256(to.superiorSP).add(superiorIncreSP));

        superior.stakingPower = uint128(uint256(superior.stakingPower).add(superiorIncreSP));
        superior.credit = uint128(uint256(superior.credit).add(superiorIncreCredit));

        _TOTAL_STAKING_POWER_ = _TOTAL_STAKING_POWER_.add(stakingPower).add(superiorIncreSP);
    }

    function _redeem(UserInfo storage from, uint256 stakingPower) internal {
        from.stakingPower = uint128(uint256(from.stakingPower).sub(stakingPower));

        // superior decrease sp = min(stakingPower*0.1, from.superiorSP)
        uint256 superiorDecreSP = DecimalMath.mulFloor(stakingPower, _SUPERIOR_RATIO_);
        superiorDecreSP = from.superiorSP <= superiorDecreSP ? from.superiorSP : superiorDecreSP;
        from.superiorSP = uint128(uint256(from.superiorSP).sub(superiorDecreSP));

        UserInfo storage superior = userInfo[from.superior];
        uint256 creditSP = DecimalMath.divFloor(superior.credit, alpha);

        if (superiorDecreSP >= creditSP) {
            superior.credit = 0;
            superior.stakingPower = uint128(uint256(superior.stakingPower).sub(creditSP));
        } else {
            superior.credit = uint128(
                uint256(superior.credit).sub(DecimalMath.mulFloor(superiorDecreSP, alpha))
            );
            superior.stakingPower = uint128(uint256(superior.stakingPower).sub(superiorDecreSP));
        }

        _TOTAL_STAKING_POWER_ = _TOTAL_STAKING_POWER_.sub(stakingPower).sub(superiorDecreSP);
    }

    function _transfer(
        address from,
        address to,
        uint256 dJGNAmount
    ) internal canTransfer balanceEnough(from, dJGNAmount) {
        require(from != address(0), "transfer from the zero address");
        require(to != address(0), "transfer to the zero address");
        require(from != to, "transfer from same with to");

        uint256 stakingPower = DecimalMath.divFloor(dJGNAmount * _JGN_RATIO_, alpha);

        UserInfo storage fromUser = userInfo[from];
        UserInfo storage toUser = userInfo[to];

        _redeem(fromUser, stakingPower);
        _mint(toUser, stakingPower);

        emit Transfer(from, to, dJGNAmount);
    }
}
```


Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the

external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-

Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io