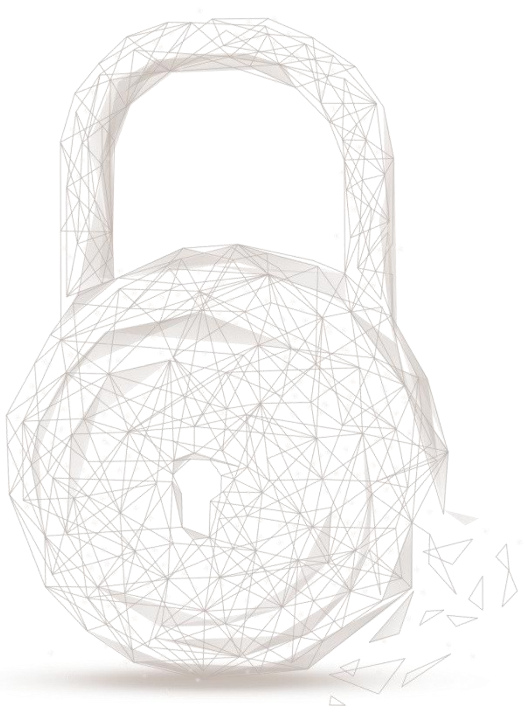
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202012151600

Smart Contract Name：

Juggernaut DeFi (JGN)

Smart Contract Address：

0xC13B7a43223BB9Bf4B69BD68Ab20ca1B79d81C75

Smart Contract Address Link：

https://bscscan.com/address/0xC13B7a43223BB9Bf4B69BD68Ab20ca1B79d81C75

Start Date：2020.12.14

Completion Date：2020.12.15

Overall Result：Pass（Distinction）

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	BEP20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		selfdestruct Function Security	Pass

3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	N/A
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Disclaimer: 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 contract JGN, including Coding Standards, Security, and Business Logic. **JGN contract passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1. Basic Token Information

Token name	Juggernaut DeFi
Token symbol	JGN
decimals	18
totalSupply	Initial total supply is 0(Burnable, mintable without limited)
Token type	BEP20

Table 1 – Basic Token Information

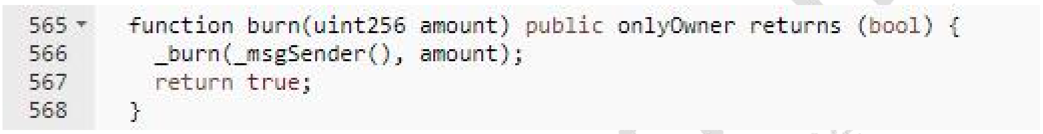
2. Token Vesting Information

N/A

3. Customized Function Audit:

(1) burn function

- Description: As shown in Figure 1, the contract implements the *burn* function to destroy tokens. Only the contract owner can call this function to destroy tokens.

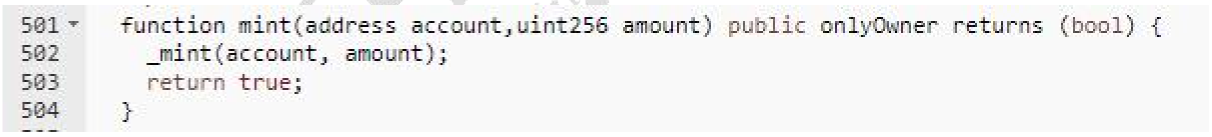


```
function burn(uint256 amount) public onlyOwner returns (bool) {
  _burn(_msgSender(), amount);
  return true;
}
```

Figure 1 burn Function Source Code

(2) mint funciton

- Description: As shown in Figure 2, the contract implements the *mint* function for token issuance, and only the contract owner can call this function for token issuance. The contract owner can call this function to mint a specified amount of tokens to the specified address, and there is no upper limit for minting.



```
function mint(address account,uint256 amount) public onlyOwner returns (bool) {
  _mint(account, amount);
  return true;
}
```

Figure 2 mint Function Source Code

Audited Source Code with Comments:

```
/**
 *Submitted for verification at BscScan.com on 2020-12-14
*/
```

```
pragma solidity 0.5.16;

interface IBEP20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256); // Beosin (Chengdu LianAn) // Declare the
interface of function 'totalSupply'.

    /**
     * @dev Returns the token decimals.
     */
    function decimals() external view returns (uint8); // Beosin (Chengdu LianAn) // Declare the interface of
function 'decimals'.

    /**
     * @dev Returns the token symbol.
     */
    function symbol() external view returns (string memory); // Beosin (Chengdu LianAn) // Declare the
interface of function 'symbol'.

    /**
    * @dev Returns the token name.
    */
    function name() external view returns (string memory); // Beosin (Chengdu LianAn) // Declare the
interface of function 'name'.

    /**
     * @dev Returns the bep token owner.
     */
    function getOwner() external view returns (address); // Beosin (Chengdu LianAn) // Declare the interface
of function 'getOwner'.

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

    /**
     * @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); // Beosin (Chengdu LianAn)
// Declare the interface of function 'transfer'.
```

```
    /**
     * @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); // Beosin (Chengdu LianAn) // Declare the interface of function 'allowance'.

    /**
     * @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); // Beosin (Chengdu LianAn) // Declare the interface of function 'approve'.

    /**
     * @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); // Beosin (Chengdu LianAn) // Declare the interface of function 'transferFrom'.

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value); // Beosin (Chengdu LianAn) // Declare the event 'Transfer'.
    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value); // Beosin (Chengdu
LianAn) // Declare the event 'Approval'.
}

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // Beosin (Chengdu LianAn) // Internal function '_msgSender' for getting the caller address.
    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
    // Beosin (Chengdu LianAn) // Internal function '_msgData' for returning the call data.
    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
```

```
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address private _owner; // Beosin (Chengdu LianAn) // Declare variable '_owner' for storing the
contract owner.

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner); // Beosin
(Chengdu LianAn) // Declare the event 'OwnershipTransferred'.

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender; // Beosin (Chengdu LianAn) // Initialize the ownership address.
        emit OwnershipTransferred(address(0), msgSender); // Beosin (Chengdu LianAn) // Trigger the
OwnershipTransferred event.
    }

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

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner"); // Beosin (Chengdu LianAn) //
Modifier, require that the caller of the modified.
        _;
    }

    /**
     * @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 onlyOwner {
        emit OwnershipTransferred(_owner, address(0)); // Beosin (Chengdu LianAn) // Trigger the event
'OwnershipTransferred'.
        _owner = address(0); // Beosin (Chengdu LianAn) // Transfer ownership to zero address.
    }

    /**
      * @dev Transfers ownership of the contract to a new account (`newOwner`).
      * Can only be called by the current owner.
      */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
      * @dev Transfers ownership of the contract to a new account (`newOwner`).
      */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'newOwner'.
        emit OwnershipTransferred(_owner, newOwner); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
        _owner = newOwner; // Beosin (Chengdu LianAn) // Transfer ownership to 'newOwner'.
    }
}

contract BEP20Token is Context, IBEP20, Ownable {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    mapping (address => uint256) private _balances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_balances' for storing the token balance of corresponding address.

    mapping (address => mapping (address => uint256)) private _allowances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_allowances' for storing the allowance between two addresses.

    uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for storing the total token supply.
    uint8 private _decimals; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for storing the token decimals.
    string private _symbol; // Beosin (Chengdu LianAn) // Declare the variable '_symbol' for storing the token symbol.
    string private _name; // Beosin (Chengdu LianAn) // Declare the variable '_name' for storing the token name.

    constructor() public {
        _name = "Juggernaut DeFi";
        _symbol = "JGN";
        _decimals = 18;
        //_totalSupply = {{TOTAL_SUPPLY}};
        //_balances[msg.sender] = _totalSupply;
        //emit Transfer(address(0), msg.sender, _totalSupply);
```

```
}

/**
  * @dev Returns the bep token owner.
  */
// Beosin (Chengdu LianAn) // Get contract owner.
function getOwner() external view returns (address) {
    return owner();
}

/**
  * @dev Returns the token decimals.
  */
// Beosin (Chengdu LianAn) // Get token decimals.
function decimals() external view returns (uint8) {
    return _decimals;
}

/**
  * @dev Returns the token symbol.
  */
// Beosin (Chengdu LianAn) // Get token symbol.
function symbol() external view returns (string memory) {
    return _symbol;
}

/**
* @dev Returns the token name.
*/
// Beosin (Chengdu LianAn) // Get token name.
function name() external view returns (string memory) {
    return _name;
}

/**
  * @dev See {BEP20-totalSupply}.
  */
// Beosin (Chengdu LianAn) // Get the current total amount of tokens.
function totalSupply() external view returns (uint256) {
    return _totalSupply;
}

/**
  * @dev See {BEP20-balanceOf}.
  */
// Beosin (Chengdu LianAn) // Get the token balance of the specified address.
function balanceOf(address account) external view returns (uint256) {
    return _balances[account];
    }

    /**
     * @dev See {BEP20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    // Beosin (Chengdu LianAn) // Token transfer function.
    function transfer(address recipient, uint256 amount) external returns (bool) {
        _transfer(_msgSender(), recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function
'_transfer' to transfer tokens.
        return true;
    }

    /**
     * @dev See {BEP20-allowance}.
     */
    // Beosin (Chengdu LianAn) // Get the allowance between two corresponding addresses.
    function allowance(address owner, address spender) external view returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {BEP20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    // Beosin (Chengdu LianAn) // 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.
    // Beosin (Chengdu LianAn) // Using function 'increaseAllowance' and 'decreaseAllowance' to alter
allowance is recommended.
    function approve(address spender, uint256 amount) external returns (bool) {
        _approve(_msgSender(), spender, amount); // Beosin (Chengdu LianAn) // Call the internal function
'_approve' to change an allowance.
        return true;
    }

    /**
     * @dev See {BEP20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {BEP20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for `sender`'s tokens of at least
     * `amount`.
     */
    // Beosin (Chengdu LianAn) // Transfer tokens from one address to another.
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool) {
        _transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function
'_transfer' to transfer tokens.
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "BEP20: transfer
amount exceeds allowance")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to set
the allowance which 'sender' allowed to 'msg.sender'.
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {BEP20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    // Beosin (Chengdu LianAn) // Function for updating allowance.
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue)); // Beosin
(Chengdu LianAn) // Call the internal function '_approve' to update allowance.
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {BEP20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    // Beosin (Chengdu LianAn) // Function for updating allowance.
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "BEP20: decreased allowance below zero")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to update allowance.
        return true;
    }

    /**
     * @dev Creates `amount` tokens and assigns them to `msg.sender`, increasing
     * the total supply.
     *
     * Requirements
     *
     * - `msg.sender` must be the token owner
     */
      // Beosin (Chengdu LianAn) // Mint tokens to specified address. Only the contract owner can call this function.
    function mint(address account,uint256 amount) public onlyOwner returns (bool) {
        _mint(account, amount); // Beosin (Chengdu LianAn) // Call the internal function '_mint' to mint tokens.
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    // Beosin (Chengdu LianAn) // Internal function for transferring tokens from 'sender' to 'recipient'.
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "BEP20: transfer from the zero address");    // Beosin (Chengdu LianAn) // The non-zero address check for 'sender'.
        require(recipient != address(0), "BEP20: transfer to the zero address");    // Beosin (Chengdu LianAn) // The non-zero address check for 'recipient'. Avoid losing transferred tokens.
    // Beosin (Chengdu LianAn) // Alter the balances of corresponding addresses and triggers the event 'Transfer' to complete the 'transfer' operation.
        _balances[sender] = _balances[sender].sub(amount, "BEP20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
      * the total supply.
      *
      * Emits a {Transfer} event with `from` set to the zero address.
      *
      * Requirements
      *
      * - `to` cannot be the zero address.
      */
    // Beosin (Chengdu LianAn) // Internal function for minting tokens.
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "BEP20: mint to the zero address"); // Beosin (Chengdu LianAn) // The
non-zero address check for 'account'.

        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Alter the total supply of
token.
        _balances[account] = _balances[account].add(amount); // Beosin (Chengdu LianAn) // Alter the token
balance of 'account'.
        emit Transfer(address(0), account, amount); // Beosin (Chengdu LianAn) // Trigger the event
'Transfer'.
    }

    /**
      * @dev Destroys `amount` tokens from `account`, reducing the
      * total supply.
      *
      * Emits a {Transfer} event with `to` set to the zero address.
      *
      * Requirements
      *
      * - `account` cannot be the zero address.
      * - `account` must have at least `amount` tokens.
      */
    // Beosin (Chengdu LianAn) // Internal functions for destroying tokens.
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "BEP20: burn from the zero address"); // Beosin (Chengdu LianAn) //
The non-zero address check for 'account'.

        _balances[account] = _balances[account].sub(amount, "BEP20: burn amount exceeds balance"); //
Beosin (Chengdu LianAn) // Alter the token balance of 'account'.
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Alter the total supply of
token.
        emit Transfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Trigger the event
'Transfer'.
```

```
    }
// Beosin (Chengdu LianAn) // Function for destroying tokens.
    function burn(uint256 amount) public onlyOwner returns (bool) {
        _burn(_msgSender(), amount); // Beosin (Chengdu LianAn) // Call the internal function '_burn' to
destroy tokens.
        return true;
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    // Beosin (Chengdu LianAn) // Internal function for setting the allowance which 'owner' allowed to
'spender'.
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "BEP20: approve from the zero address"); // Beosin (Chengdu LianAn) //
The non-zero address check for 'owner'.
         require(spender != address(0), "BEP20: approve to the zero address"); // Beosin (Chengdu LianAn) //
The non-zero address check for 'spender'.

        _allowances[owner][spender] = amount; // Beosin (Chengdu LianAn) // Sets the allowance which
'owner' allowed to 'spender' as 'amount'.
        emit Approval(owner, spender, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
    }

    /**
     * @dev Destroys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See {_burn} and {_approve}.
     */
    // Beosin (Chengdu LianAn) // Redundant code, it is recommended to delete.
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount); // Beosin (Chengdu LianAn) // Call the internal function '_burn' to
implement '_burnFrom' operation.
        //_approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "BEP20: burn
amount exceeds allowance")); // Beosin (Chengdu LianAn) // This line of code in the _burnFrom function
cannot be commented out, which will affect the function of the _burnFrom function.
    }
```

```
}
// Beosin (Chengdu LianAn) // Recommend the main contract to inherit 'Pausable' module to grant
owner the authority of pausing all transactions when serious issue occurred.
```

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com