BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number：202101281535

Smart Contract Name：

CYBER USD (CUSD)

Smart Contract Address：

0xf7bcc16f5ccf1c7fa195c22dda8a88329e528649

Smart Contract Address Link：

https://etherscan.io/address/0xf7bcc16f5ccf1c7fa195c22dda8a88329e528649

Start Date：2021.01.22

Completion Date：2021.01.28

Overall Result：Pass（Distinction）

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 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

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 CUSD, including Coding Standards, Security, and Business Logic. **CUSD 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	CYBER USD
Token symbol	CUSD
decimals	18
totalSupply	Initial total supply is 20Millon (Burnable, Mintable without cap)
Token type	ERC20

Table 1 – Basic Token Information

2、Token Vesting Information

N/A

Audited Source Code with Comments:

```
pragma solidity 0.6.1;

/**
 * @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 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 {
    // Beosin (Chengdu LianAn) // Internal function '_msgSender' for getting the caller address.
    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }
}

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
```

```
 */
interface IERC20 {
    // Beosin (Chengdu LianAn) // Define the function interfaces required by ERC20 Token standard.
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @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.
     *
     * > 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);
    // Beosin (Chengdu LianAn) // Declare the events 'Transfer' and 'Approval'.
    /**
    * @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);

    /**
    * @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);
}

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20MinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
```

```
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    uint256 internal _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for storing the total token supply.

    mapping (address => uint256) internal _balances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_balances' for storing the token balance of corresponding address.
    mapping (address => mapping (address => uint256)) internal _allowances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_allowances' for storing the allowance between two addresses.

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view override returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount); // Beosin (Chengdu LianAn) // Call the internal function '_transfer' to transfer tokens.
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }
```

```
    /**
     * @dev See {IERC20-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) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to set the allowance which 'msg.sender' allowed to 'spender'.
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * 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`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override 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, "ERC20: transfer amount exceeds allowance")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to alter the allowance between two addresses.
        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 {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
```

```
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue)); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to increase the allowance between two addresses.
        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 {IERC20-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`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero")); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to decrease the allowance between two addresses.
        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`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'sender'.
```

```
        require(recipient != address(0), "ERC20: transfer to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'recipient'. Avoid losing transferred tokens.

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the token balance of 'sender'.
        _balances[recipient] = _balances[recipient].add(amount); // Beosin (Chengdu LianAn) // Alter the token balance of 'recipient'.
        emit Transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }
    /** @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.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.
        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
        _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.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'account'.

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance"); // Beosin (Chengdu LianAn) // Alter the token balance of 'account'.
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update the total token supply.
```

```
        emit Transfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    }

    /**
     * @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.
     */
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'owner'.
        require(spender != address(0), "ERC20: approve to the zero address"); // Beosin (Chengdu LianAn) // The non-zero address check for 'spender'.

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


}

/**
 * @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 internal _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 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 function must be owner.
        _;
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address"); // Beosin (Chengdu
LianAn) // The non-zero address check for 'newOwner'. Avoid losing ownership.
        emit OwnershipTransferred(_owner, newOwner); // Beosin (Chengdu LianAn) // Trigger the event
'OwnershipTransferred'.
        _owner = newOwner; // Beosin (Chengdu LianAn) // Transfer ownership to 'newOwner'.
    }
}

/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
abstract contract ERC20Mintable is Ownable, ERC20{
    /**
     * @dev See `ERC20._mint`.
     *
     * Requirements:
     *
     * - the caller must have the `MinterRole`.
     */
    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;
```

```
    }
}
abstract contract ERC20Burnable is Ownable, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual  {
        _burn(_msgSender(), amount); // Beosin (Chengdu LianAn) // Call the internal function '_burn' to destroy tokens.
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    function burnFrom(address account, uint256 amount) public virtual  {
        uint256 decreasedAllowance = allowance(account, _msgSender()).sub(amount, "ERC20: burn amount exceeds allowance");

        _approve(account, _msgSender(), decreasedAllowance); // Beosin (Chengdu LianAn) // Call the internal function '_approve' to alter the allowance between two addresses.
        _burn(account, amount); // Beosin (Chengdu LianAn) // Call the internal function '_burn' to destroy tokens.
    }
}

contract CUSDToken is ERC20Burnable , ERC20Mintable{
    string public constant name = "CYBER USD"; // Beosin (Chengdu LianAn) // Declare the constant 'name' for storing the token name. It is 'CYBER USD'.
    string public constant symbol = "CUSD"; // Beosin (Chengdu LianAn) // Declare the constant 'symbol' for storing the token symbol. It is 'CUSD'.
    uint8 public constant decimals = 18; // Beosin (Chengdu LianAn) // Declare the constant 'decimals' for storing the token name. There are 18 decimals.
    address public constant tokenOwner = 0x7EfF41CFcd40255FAE948d26E6c0Ce5f18d01038; // Beosin (Chengdu LianAn) // Declare the constant 'tokenOwner' for storing the address of receiving initial tokens.
    uint256 public constant INIT_TOTALSUPPLY = 20000000; // Beosin (Chengdu LianAn) // Declare the constant 'INIT_TOTALSUPPLY' for storing the initial total token supply.
```

```
    /**
     * @dev Constructor.
     */
    constructor () public {
        _owner = tokenOwner; // Beosin (Chengdu LianAn) // Initialize the contract owner to 'tokenOwner'.
        _totalSupply = INIT_TOTALSUPPLY.mul(10 ** uint256(decimals)); // Beosin (Chengdu LianAn) // Initialize the total token supply.
        _balances[tokenOwner] = _totalSupply; // Beosin (Chengdu LianAn) // Send all tokens to the address 'tokenOwner'.
        emit Transfer(address(0), tokenOwner, _totalSupply); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
        emit OwnershipTransferred(address(0), tokenOwner); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
    }
    // Beosin (Chengdu LianAn) // Batch transfer function, which can only be called by owner.
    function batchTransfer(address[] memory _to, uint256[] memory _value) public onlyOwner returns (bool) {
        require(_to.length > 0);
        require(_to.length <= 150);
        require(_to.length == _value.length);
        for(uint256 k = 0; k < _to.length; k++){
            _transfer(msg.sender, _to[k], _value[k]);
        }
        return true;
    }
}
// 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