BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number：202104121516

Smart Contract Name：

FUN Protocol (FUN Protocol)

Smart Contract Address：

Fill in after deployment

Smart Contract Address Link：

Fill in after deployment

Start Date：2021.04.09

Completion Date：2021.04.12

Overall Result：Pass (Distinction)

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	BEP-20 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 FUN Protocol, including Coding Standards, Security, and Business Logic. **FUN Protocol 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	FUN Protocol
Token symbol	FUN Protocol
decimals	9
totalSupply	1.5 Million (The total supply changes with market value)
Token type	BEP-20

Table 1 – Basic Token Information

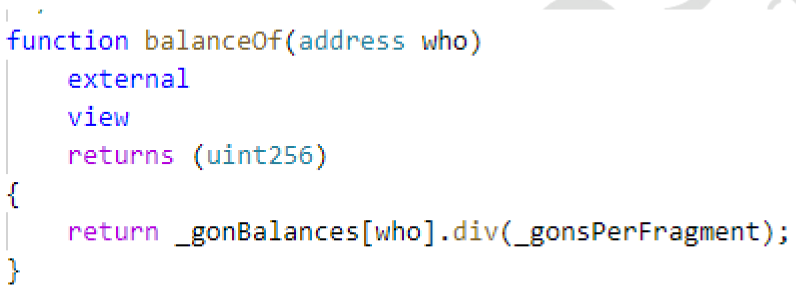
2. Token Vesting Information

N/A

3. Customized Function Audit:

(1) balanceOf function

- Description: As shown in Figure 1 below, The contract implements the function ‘balanceOf’, the caller can this function to query token balance. However, the token balance changes with ‘_gonsPerFragment’, that is, the actual queried token balance is based on the rounding of ‘_gonsPerFragment’ (the total user balance is different with the total supply).



```
function balanceOf(address who)
    external
    view
    returns (uint256)
{
    return _gonBalances[who].div(_gonsPerFragment);
}
```

Figure 1 *balanceOf* Function Source Code

Audited Source Code with Comments

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


/*

MIT License


Copyright (c) 2018 requestnetwork
Copyright (c) 2018 Fragments, Inc.

Copyright (c) 2020 Fun Money


Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:


The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.


THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

*/

pragma solidity >=0.5.17; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

// Beosin (Chengdu LianAn) // The SafeMath library declares functions for safe mathematical operation.
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) {
        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;
    }
}
```

// Beosin (Chengdu LianAn) // The SafeMathInt library declares functions for safe mathematical operation.

```
library SafeMathInt {
    int256 private constant MIN_INT256 = int256(1) << 255;
    int256 private constant MAX_INT256 = ~(int256(1) << 255);


    /**
     * @dev Multiplies two int256 variables and fails on overflow.
     */
    function mul(int256 a, int256 b)
        internal
        pure
        returns (int256)
    {
        int256 c = a * b;


        // Detect overflow when multiplying MIN_INT256 with -1
        require(c != MIN_INT256 || (a & MIN_INT256) != (b & MIN_INT256), "SafeMathInt:
multiplication overflow");
        require((b == 0) || (c / b == a), "SafeMathInt: multiplication overflow");
        return c;
    }


    /**
     * @dev Division of two int256 variables and fails on overflow.
     */
    function div(int256 a, int256 b)
        internal
        pure
        returns (int256)
    {
        // Prevent overflow when dividing MIN_INT256 by -1
        require(b != -1 || a != MIN_INT256, "SafeMathInt: division overflow");

        // Solidity already throws when dividing by 0.
        return a / b;
    }

    /**
     * @dev Subtracts two int256 variables and fails on overflow.
     */
    function sub(int256 a, int256 b)
        internal
        pure
        returns (int256)
    {
        int256 c = a - b;
        require((b >= 0 && c <= a) || (b < 0 && c > a), "SafeMathInt: subtraction overflow");
        return c;
    }

    /**
     * @dev Adds two int256 variables and fails on overflow.
     */
    function add(int256 a, int256 b)
        internal
        pure
        returns (int256)
    {
        int256 c = a + b;
```

```
            require((b >= 0 && c >= a) || (b < 0 && c < a), "SafeMathInt: addition overflow");

            return c;

        }



        /**

          * @dev Converts to absolute value, and fails on overflow.

          */

        function abs(int256 a)

            internal

            pure

            returns (int256)

        {

            require(a != MIN_INT256, "SafeMathInt: abs overflow");

            return a < 0 ? -a : a;

        }

}



contract Ownable {
    address private _owner;



    event OwnershipRenounced(address indexed previousOwner);



    event OwnershipTransferred(

        address indexed previousOwner,

        address indexed newOwner

    );



    /**
      * @dev The Ownable constructor sets the original `owner` of the contract to the sender
```

```
  * account.

  */

constructor() public {

    _owner = msg.sender;

}


/**

  * @return the address of the owner.

  */

function owner() public view returns(address) {

    return _owner;

}


/**

  * @dev Throws if called by any account other than the owner.

  */

modifier onlyOwner() {

    require(isOwner());

    _;

}


/**

  * @return true if `msg.sender` is the owner of the contract.

  */

function isOwner() public view returns(bool) {

    return msg.sender == _owner;

}
```

```
    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0));
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract ERC20Detailed {
        string private _name; // Beosin (Chengdu LianAn) // Declare the variable '_name' for storing the token name.
        string private _symbol; // Beosin (Chengdu LianAn) // Declare the variable '_symbol' for storing the token symbol.
        uint8 private _decimals; // Beosin (Chengdu LianAn) // Declare the variable '_decimals' for storing the token decimals.
        // Beosin (Chengdu LianAn) // Declare the event 'Transfer'.
        event Transfer(
                address indexed from,
        address indexed to,

        uint256 value

);

// Beosin (Chengdu LianAn) // Declare the event 'Approval'.

event Approval(

        address indexed owner,

        address indexed spender,

        uint256 value

);

/**

  * @dev Sets the values for `name`, `symbol`, and `decimals`. All three of

  * these values are immutable: they can only be set once during

  * construction.

  */

// Beosin (Chengdu LianAn) // Constructor function, initialize token basic information.

constructor (string memory name, string memory symbol, uint8 decimals) public {

        _name = name;

        _symbol = symbol;

        _decimals = decimals;

}


/**

  * @dev Returns the name of the token.

  */

function name() public view returns (string memory) {

        return _name;

}
```

```
    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

/**
 * @title Fun ERC20 token
 * @dev
 *          The goal of Fun is to maintain a stable price of 1$.

 *          Based on the Ampleforth protocol.

 */

contract Fun is ERC20Detailed, Ownable {

        using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical
operation. Avoid integer overflow/underflow.

        using SafeMathInt for int256; // Beosin (Chengdu LianAn) // Use the SafeMathInt library for
mathematical operation. Avoid integer overflow/underflow.



        event LogRebase(uint256 indexed epoch, uint256 totalSupply); // Beosin (Chengdu LianAn) // Declare
the event 'LogRebase'.


        // Used for authentication

        address public master; // Beosin (Chengdu LianAn) // Declare the variable 'master' for storing the
management address.

        // Beosin (Chengdu LianAn) // Modifier, require that the caller of the modified function must be
'master' address.

        modifier onlyMaster() {

                require(msg.sender == master, 'only master');

                _;

        }



        // Only the owner can transfer tokens in the initial phase.

        // This is to prevent the AMM listing to happen in an orderly fashion.



        bool public initialDistributionFinished;



        mapping (address => bool) allowTransfer; // Beosin (Chengdu LianAn) // Declare the mapping variable
'allowTransfer' for storing the used to store tradable addresses.

        modifier initialDistributionLock {

                require(initialDistributionFinished || isOwner() || allowTransfer[msg.sender], 'initial distribution
lock');

            _;

        }

        // Beosin (Chengdu LianAn) // Modifier, make a function callable only when the specified address 'to' not equal to zero address and not equal to this contract address.

        modifier validRecipient(address to) {

            require(to != address(0x0), 'zero address');

            require(to != address(this), 'this address');

            _;

        }



        uint256 private constant DECIMALS = 9; // Beosin (Chengdu LianAn) // Declare the constant 'DECIMALS' for storing the token decimals, there are 9 as default.

        uint256 private constant MAX_UINT256 = ~uint256(0); // Beosin (Chengdu LianAn) // Declare the constant 'MAX_UINT256' for storing maximum value of uint256.



        uint256 private constant INITIAL_FRAGMENTS_SUPPLY = 2000000 * 10**DECIMALS; // Beosin (Chengdu LianAn) // Declare the constant 'INITIAL_FRAGMENTS_SUPPLY' for storing initial token total supply.



        // TOTAL_GONS is a multiple of INITIAL_FRAGMENTS_SUPPLY so that _gonsPerFragment is an integer.

        // Use the highest value that fits in a uint256 for max granularity.

        uint256 private constant TOTAL_GONS = MAX_UINT256 - (MAX_UINT256 % INITIAL_FRAGMENTS_SUPPLY);



        // MAX_SUPPLY = maximum integer < (sqrt(4*TOTAL_GONS + 1) - 1) / 2

        uint256 private constant MAX_SUPPLY = ~uint128(0);    // (2^128) - 1



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

        uint256 private _gonsPerFragment; // Beosin (Chengdu LianAn) // Declare the variable
```

```
'_gonsPerFragment' for storing token base.

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


    // This is denominated in Fragments, because the gons-fragments conversion might change before

    // it's fully paid.

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


    /**

     * @dev Notifies Fragments contract about a new rebase cycle.

     * @param supplyDelta The number of new fragment tokens to add into circulation via expansion.

     * @return The total number of fragments after the supply adjustment.

     */

    function rebase(uint256 epoch, int256 supplyDelta)

        external

        onlyMaster

        returns (uint256)

    {

        if (supplyDelta == 0) { // Beosin (Chengdu LianAn) // If the 'supplyDelta' equal to zero, return the
token total supply.

            emit LogRebase(epoch, _totalSupply); // Beosin (Chengdu LianAn) // Trigger the event
'LogRebase'.

            return _totalSupply;

        }


        if (supplyDelta < 0) { // Beosin (Chengdu LianAn) // If 'supplyDelta' is less than zero, update
'_totalSupply' to be the difference between the total supply of tokens and the absolute value of 'supplyDelta'.

            _totalSupply = _totalSupply.sub(uint256(supplyDelta.abs()));

        } else { // Beosin (Chengdu LianAn) // Otherwise, update '_totalSupply' is the sum of the token
total supply and 'supplyDelta'.

                        _totalSupply = _totalSupply.add(uint256(supplyDelta));

                }



                if (_totalSupply > MAX_SUPPLY) { // Beosin (Chengdu LianAn) // If the '_totalSupply' greater
than maximum token total supply, update '_totalSupply' to 'MAX_SUPPLY'.

                        _totalSupply = MAX_SUPPLY;

                }



                _gonsPerFragment = TOTAL_GONS.div(_totalSupply); // Beosin (Chengdu LianAn) // Calculate
the '_gonsPerFragment'.



                emit LogRebase(epoch, _totalSupply); // Beosin (Chengdu LianAn) // Trigger the event
'LogRebase'.

                return _totalSupply;

        }

        // Beosin (Chengdu LianAn) // Initialize contract related information.

        constructor()

                ERC20Detailed("FUN Protocol", "FUN Protocol", uint8(DECIMALS))

                public

        {

                _totalSupply = INITIAL_FRAGMENTS_SUPPLY;

                _gonBalances[msg.sender] = TOTAL_GONS;

                _gonsPerFragment = TOTAL_GONS.div(_totalSupply);



                initialDistributionFinished = false;



                emit Transfer(address(0x0), msg.sender, _totalSupply); // Beosin (Chengdu LianAn) // Trigger the
event 'Transfer'.

        }
```

```
/**
  * @notice Sets a new master
  */
// Beosin (Chengdu LianAn) // Only the owner calls this function to set the 'master' address.
function setMaster(address _master)
        external
        onlyOwner
        returns (uint256)
{
        master = _master;
}


/**
  * @return The total number of fragments.
  */
function totalSupply()
        external
        view
        returns (uint256)
{
        return _totalSupply;
}


/**
  * @param who The address to query.
  * @return The balance of the specified address.
  */
```

```
    function balanceOf(address who)

        external

        view

        returns (uint256)

    {

        return _gonBalances[who].div(_gonsPerFragment);

    }


    /**

     * @dev Transfer tokens to a specified address.

     * @param to The address to transfer to.

     * @param value The amount to be transferred.

     * @return True on success, false otherwise.

     */

    function transfer(address to, uint256 value)

        external

        validRecipient(to)

        initialDistributionLock

        returns (bool)

    {

        uint256 gonValue = value.mul(_gonsPerFragment); // Beosin (Chengdu LianAn) // Declare the local variable 'gonValue' for storing transfer token value.

        // Beosin (Chengdu LianAn) // Alter the token value of 'msg.sender' and 'to'.

        _gonBalances[msg.sender] = _gonBalances[msg.sender].sub(gonValue);

        _gonBalances[to] = _gonBalances[to].add(gonValue);

        emit Transfer(msg.sender, to, value); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.

        return true;

    }
```

```
/**
  * @dev Function to check the amount of tokens that an owner has allowed to a spender.
  * @param owner_ The address which owns the funds.
  * @param spender The address which will spend the funds.
  * @return The number of tokens still available for the spender.
  */
function allowance(address owner_, address spender)
    external
    view
    returns (uint256)
{
    return _allowedFragments[owner_][spender];
}

/**
  * @dev Transfer tokens from one address to another.
  * @param from The address you want to send tokens from.
  * @param to The address you want to transfer to.
  * @param value The amount of tokens to be transferred.
  */
function transferFrom(address from, address to, uint256 value)
    external
    validRecipient(to)
    returns (bool)
{
    _allowedFragments[from][msg.sender] = _allowedFragments[from][msg.sender].sub(value); //
Beosin (Chengdu LianAn) // Alter the allowance between 'from' and 'msg.sender'.
    uint256 gonValue = value.mul(_gonsPerFragment); // Beosin (Chengdu LianAn) // Declare the local
```

```
variable 'gonValue' for recording transfer token value.
            // Beosin (Chengdu LianAn) // Alter the token value of 'to' and 'from'.
            _gonBalances[from] = _gonBalances[from].sub(gonValue);
            _gonBalances[to] = _gonBalances[to].add(gonValue);
            emit Transfer(from, to, value); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.


            return true;
        }


        /**
         * @dev Approve the passed address to spend the specified amount of tokens on behalf of
         * msg.sender. This method is included for ERC20 compatibility.
         * increaseAllowance and decreaseAllowance should be used instead.
         * Changing an allowance with this method brings the risk that someone may transfer both
         * the old and the new allowance - if they are both greater than zero - if a transfer
         * transaction is mined before the later approve() call is mined.
         *
         * @param spender The address which will spend the funds.
         * @param value The amount of tokens to be spent.
         */
        // 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 value)
            external
            initialDistributionLock
            returns (bool)
        {
            _allowedFragments[msg.sender][spender] = value; // Beosin (Chengdu LianAn) // The allowance
```

```
which 'msg.sender' allowed to 'spender' is set to 'value'.

        emit Approval(msg.sender, spender, value); // Beosin (Chengdu LianAn) // Trigger the event
'Approval'.
        return true;

    }



    /**

     * @dev Increase the amount of tokens that an owner has allowed to a spender.

     * This method should be used instead of approve() to avoid the double approval vulnerability

     * described above.

     * @param spender The address which will spend the funds.

     * @param addedValue The amount of tokens to increase the allowance by.

     */

    function increaseAllowance(address spender, uint256 addedValue)

        external

        initialDistributionLock

        returns (bool)

    {

        _allowedFragments[msg.sender][spender] =

            _allowedFragments[msg.sender][spender].add(addedValue); // Beosin (Chengdu LianAn) //
increase the allowance between 'msg.sender' and 'spender'.

        emit Approval(msg.sender, spender, _allowedFragments[msg.sender][spender]); // Beosin (Chengdu
LianAn) // Trigger the event 'Approval'.

        return true;

    }



    /**

     * @dev Decrease the amount of tokens that an owner has allowed to a spender.

     *

     * @param spender The address which will spend the funds.
     * @param subtractedValue The amount of tokens to decrease the allowance by.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue)
        external
        initialDistributionLock
        returns (bool)
    {
        uint256 oldValue = _allowedFragments[msg.sender][spender]; // Beosin (Chengdu LianAn) // Declare the local variable 'oldValue' for recording the current allowance which 'msg.sender' allowed to 'spender'.
        if (subtractedValue >= oldValue) {// Beosin (Chengdu LianAn) // If decrease allowance no less than 'oldValue', the allowance which 'msg.sender' allowed to 'spender' is set to 0.
            _allowedFragments[msg.sender][spender] = 0;
        } else { // Beosin (Chengdu LianAn) // Otherwise, decrease the allowance between 'msg.sender' and 'spender'.
            _allowedFragments[msg.sender][spender] = oldValue.sub(subtractedValue);
        }
        emit Approval(msg.sender, spender, _allowedFragments[msg.sender][spender]); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
        return true;
    }
    // Beosin (Chengdu LianAn) // The function is called by the owner to complete the initialization settings, and the user can trade normally.
    function setInitialDistributionFinished()
        external
        onlyOwner
    {
        initialDistributionFinished = true;
    }
    // Beosin (Chengdu LianAn) // This function is called by the owner to add a tradable address.
    function enableTransfer(address _addr)

        external

        onlyOwner

    {

        allowTransfer[_addr] = 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