BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number：202105071722

Smart Contract Name：

SYL (SYL)

Smart Contract Link：

https://bscscan.com/address/0x7e52a123ed6db6ac872a875552935fbbd2544c86#code

Smart Contract Adress：

0x7e52a123Ed6DB6Ac872A875552935fBbD2544c86

Start Date：2021.05.06

Completion Date：2021.05.07

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

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

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract SYL, including Coding Standards, Security, and Business Logic. **SYL 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	SYL
Token symbol	SYL
decimals	6
totalSupply	Initial supply is 7,000,000,000 (Burnable, the max supply is 10,000,000,000)
Token type	BEP–20

Table 1 – SYL Token Information

2. Token Vesting Information

N/A

Audited Source Code with Comments

BEP20.sol

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

pragma solidity ^0.8.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import   "./IBEP20.sol";
import   "./Ownable.sol";
import   "./SafeMath.sol";

contract BEP20 is Context, IBEP20, Ownable {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // References the "SafeMath" library for
security math operations.
    uint256 constant thirtyDays = 2592000; // Beosin (Chengdu LianAn) // Declare the constant
"thirtyDays" to store a 30-day period.
    mapping (address => uint256) private _balances; // Beosin (Chengdu LianAn) // Declare the mapping
variable "_balances" to store the balance of tokens at the specified address.

    mapping (address => mapping (address => uint256)) private _allowances; // Beosin (Chengdu LianAn) //
Declare mapping variable "_allowances" to store authorization values between corresponding
addresses.

    uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable "_totalSupply" to
store the total amount of tokens.
    uint256 immutable private _maxSupply; // Beosin (Chengdu LianAn) // Declare an immutable variable
```

```
"_maxSupply " to store the upper limit of tokens.
    uint256 private _monthlySupply; // Beosin (Chengdu LianAn) // Declare the variable
"_monthlySupply" to store the number of tokens generated each month.
    uint256 private _maxMonthlySupply; // Beosin (Chengdu LianAn) // Declare the variable
"_maxMonthlySupply" to store the monthly minting limit.
    uint8      private _decimals; // Beosin (Chengdu LianAn) // Declare the variable "_decimals" to store
token decimals.
    string    private _symbol; // Beosin (Chengdu LianAn) // Declare the variable "_symbol" to store token
symbol.
    string    private _name; // Beosin (Chengdu LianAn) // Declare the variable "_name" to store token
name.
    uint256 private _startTime; // Beosin (Chengdu LianAn) // Declare the variable "_startTime" to store
the start of each month's timestamp.
    address private _subventionAdd; // Beosin (Chengdu LianAn) // Declare the variable
"_subventionAdd" to receive monthly tokens minted.

    constructor(address subventionAdd)    {
        _name                              = "SYL"; // Beosin (Chengdu LianAn) // Initialize the token name.
        _symbol                            = "SYL"; // Beosin (Chengdu LianAn) // Initialize the token symbol.
        _decimals                          = 6; // Beosin (Chengdu LianAn) // Initialize the token decimals.
        _totalSupply                       = 7000000000 * (10 ** uint256(_decimals)); //(70%) => 7 MD
        _balances[msg.sender] = _totalSupply; // Beosin (Chengdu LianAn) // Send the initial token to the
contract deployer.
        _startTime                         = 1617141600; // 31 Mars 2021    00:00:00
        _maxMonthlySupply          = 60000000 * (10 ** uint256(_decimals)); // 60 MN SYL
        _maxSupply                       = 10000000000 * (10 ** uint256(_decimals)); // 10 MD
        _subventionAdd                 = subventionAdd; // Beosin (Chengdu LianAn) // Initialize the
"_subventionAdd" address.
        emit Transfer(address(0), msg.sender, _totalSupply); // Beosin (Chengdu LianAn) // Trigger the
"Transfer" event.
    }

    /**
     * @dev Returns the bep token owner.
     */
    function getOwner() public view    override returns (address) {
        return owner();
    }
    /**
     * @dev Returns the subventionAdd.
     */
    function getSubventionAdd() external view returns (address) {
        return _subventionAdd;
    }

    /**
     * @dev Returns the token decimals.
     */
```

```
function decimals() public view    override returns (uint8) {
    return _decimals;
}

/**
  * @dev Returns the token symbol.
  */
function symbol() public view    override returns (string memory) {
    return _symbol;
}

/**
* @dev Returns the token name.
*/
function name() public view    override returns (string memory) {
    return _name;
}

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

/**
  * @dev See {BEP20-balanceOf}.
  */
// Beosin (Chengdu LianAn) // Query the number of tokens for the specified account.
function balanceOf(address account) public view    override returns (uint256) {
    return _balances[account];
}

/**
  * @dev Returns the cap on the token's total supply.
  */
  function maxSupply() public view    returns (uint256) {
        return _maxSupply;
  }

/**
  * @dev Returns the max monthly cap.
  */
  function maxMonthlySupply() public view    returns (uint256) {
      return _maxMonthlySupply;
  }
/**
```

```
     * @dev See {BEP20-monthlySupply}.
     */
    // Beosin (Chengdu LianAn) // Query the supply of tokens cast per month.
    function monthlySupply() external view returns (uint256) {
      return _monthlySupply;
    }
   /**
     * @dev See {BEP20-startTime}.
     */
    // Beosin (Chengdu LianAn) // Query the "_startTime" timestamp.
    function startTime() external view returns (uint256) {
        return _startTime;
    }
    /**
      * @dev See _maxMonthlySupply - _monthlySupply.
      */
      // Beosin (Chengdu LianAn) // Query the number of remaining tokens that can be minted per
month.
      function getRestToBeMinted() external view returns (uint256) {
         return _maxMonthlySupply.sub(_monthlySupply);
      }
   /**
     * @dev See {BEP20-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);
      return true;
   }

   /**
     * @dev See {BEP20-allowance}.
     */
      // Beosin (Chengdu LianAn) // Query the token authorization value of the specified address
"owner" to the "sender".
   function allowance(address owner, address spender) public view   override 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) public virtual    override returns (bool) {
        _approve(_msgSender(), spender, amount);
        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`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual    override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "BEP20: transfer amount exceeds allowance"));
        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.
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        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`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "BEP20:
decreased allowance below zero"));
        return true;
    }

    /**
     * @dev Creates `amount` tokens and assigns them to `subventionAdd`, increasing
     * the total supply and the monthlySupply (will be set to 0 after 30 days from the start time).
     * Every 30 days we can creates at max 60 MN
     *
     * See {ERC20-_mint}.
     */
    function mint(uint256 amount) public onlyOwner returns (bool) {
        if ( block.timestamp > _startTime.add(thirtyDays)){ // Beosin (Chengdu LianAn) // Determine
whether the current time from "_startTime" is more than 30 days.
                _monthlySupply = 0; // Beosin (Chengdu LianAn) // Set the number of tokens per month to
"0".
                _startTime = _startTime.add(thirtyDays); // Beosin (Chengdu LianAn) // Add the timestamp of
"_startTime" to the timestamp of "thirtyDays".
        }
        if ( block.timestamp >= _startTime){ // Beosin (Chengdu LianAn) // Determine whether the current
time is greater than the "_startTime" time.
                require(_monthlySupply.add(amount) <= maxMonthlySupply(), "BEP20: monthly cap exceeded");
// Beosin (Chengdu LianAn) // Determine that the number of tokens per month after minting does not
exceed the monthly limit of tokens.
                require(_totalSupply.add(amount) <= maxSupply(), "BEP20: cap exceeded"); // Beosin (Chengdu
LianAn) // Determine that the total amount of tokens after minting does not exceed the upper limit of
the total amount of tokens.
                _mint(_subventionAdd, amount); // Beosin (Chengdu LianAn) // Minting and sending "amount"
tokens to the beneficiary.
                _monthlySupply = _monthlySupply.add(amount); // Beosin (Chengdu LianAn) // Update the total
amount of minted tokens each month.
        }
        return true;
    }


    /**
      * @dev Destroys `amount` tokens from the caller.
      *
      * See {ERC20-_burn}.
      */
    function burn(uint256 amount) public onlyOwner returns (bool) {
        _burn(_msgSender(), amount);
        return true;
    }

    /**
        * @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 returns (bool) {
                _burnFrom(account, amount);
                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), "BEP20: transfer from the zero address"); // Beosin (Chengdu LianAn) //
"sender" non-zero address check.
        require(recipient != address(0), "BEP20: transfer to the zero address");// Beosin (Chengdu LianAn) //
```

```
"recipient" non-zero address check.
        // condition    chaque mois

        _balances[sender] = _balances[sender].sub(amount, "BEP20: transfer amount exceeds balance"); //
Beosin (Chengdu LianAn) // Update the total number of tokens.
        _balances[recipient] = _balances[recipient].add(amount); // Beosin (Chengdu LianAn) // Update the
account token balance.
        emit Transfer(sender, recipient, amount); // Beosin (Chengdu LianAn) // Trigger the "Transfer"
event.
    }

    /** @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 {
        require(account != address(0), "BEP20: mint to the zero address"); // Beosin (Chengdu LianAn) //
"account" non-zero address check.

        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the total amount of
tokens.
        _balances[account] = _balances[account].add(amount); // Beosin (Chengdu LianAn) // Update the
total number of tokens.
        emit Transfer(address(0), account, amount); // Beosin (Chengdu LianAn) // Trigger the "Transfer"
event.
    }

    /**
      * @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 {
        require(account != address(0), "BEP20: burn from the zero address"); // Beosin (Chengdu LianAn) //
"account" non-zero address check.

        _balances[account] = _balances[account].sub(amount, "BEP20: burn amount exceeds balance"); //
```

```
Beosin (Chengdu LianAn) // Update the number of tokens in the "account".
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update the total amount of
tokens.
        emit Transfer(account, address(0), amount); // Beosin (Chengdu LianAn) // Trigger the "Transfer"
event.
    }
    /**
      * @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), "BEP20: approve from the zero address"); // Beosin (Chengdu LianAn) //
"owner" non-zero address check.
        require(spender != address(0), "BEP20: approve to the zero address"); // Beosin (Chengdu LianAn) //
"spender" non-zero address check.

        _allowances[owner][spender] = amount; // Beosin (Chengdu LianAn) // Update the token
authorization value of "owner" to "spender".
        emit Approval(owner, spender, amount); // Beosin (Chengdu LianAn) // Trigger the "Approval"
event.
    }
    /**
      * @dev Destroys `amount` tokens from `account`.`amount` is then deducted
      * from the caller's allowance.
      *
      * See {_burn} and {_approve}.
      */
    function _burnFrom(address account, uint256 amount) internal {
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "BEP20: burn
amount exceeds allowance"));
        _burn(account, amount);
    }
}
```

Context.sol

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

pragma solidity ^0.8.0;
/*
```

```
  * @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.
  */
abstract contract Context {
        // Beosin (Chengdu LianAn) // Internal function "_msgSender", used to obtain the address of the
function caller.
        function _msgSender() internal view virtual returns (address) {
                return msg.sender;
        }
        // Beosin (Chengdu LianAn) // Internal function "_msgData", used to get the transaction data.
        function _msgData() internal view virtual returns (bytes calldata) {
                this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
                return msg.data;
        }
}
```

IBEP20.sol

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

pragma solidity ^0.8.0;

// Beosin (Chengdu LianAn) // Define the interface functions and events required by the BEP20
standard.
interface IBEP20 {
    /**
      * @dev Returns the amount of tokens in existence.
      */
    function totalSupply() external view returns (uint256);

    /**
      * @dev Returns the token decimals.
      */
    function decimals() external view returns (uint8);

    /**
      * @dev Returns the token symbol.
      */
    function symbol() external view returns (string memory);

    /**
```

```
* @dev Returns the token name.
*/
function name() external view returns (string memory);

/**
  * @dev Returns the bep token owner.
  */
function getOwner() external view returns (address);

/**
  * @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);

    /**
     * @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);
}
```

Ownable.sol

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

pragma solidity ^0.8.0;

/**
  * @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.
  */
  import "./Context.sol";

// Beosin (Chengdu LianAn) // The "Ownable" contract is used to set access control for the contract
```

```
owner.
contract Ownable is Context {
    address private _owner; // Beosin (Chengdu LianAn) // Declare the variable "_owner" to store the
address of the contract owner.

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor ()    {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @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");
        _;
    }

    /**
     * @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));
        _owner = address(0);
    }

    /**
     * @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");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
```

SafeMath.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

/**
 * @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.
 */
// Beosin (Chengdu LianAn) // The "SafeMath" library is used for safe math operations to avoid integer overflow.
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;
    }
}
```
// 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