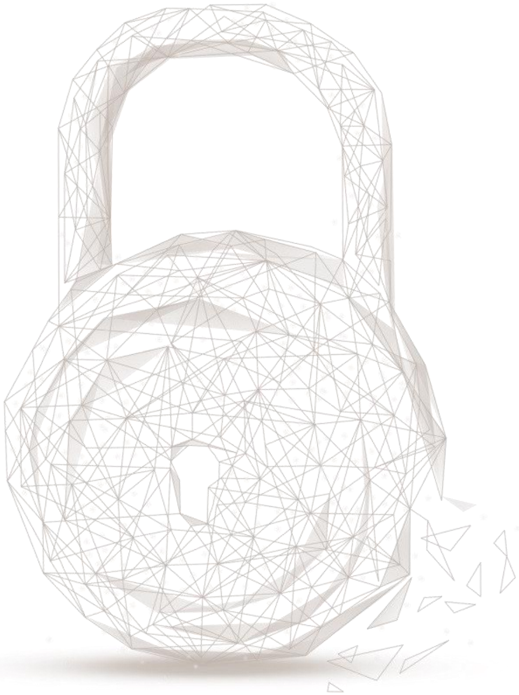
Beosin

成都链安
BEOSIN

Smart contract security audit report



Audit Number：202009102048

Report Query Name: MFI

Smart Contract Info：

Smart Contract Name	**Smart Contract Address**	**Smart Contract Address Link**
MoonPoolCRTReward	TMKYuA8JuJJwrqtrwcayRHADXTYDZNveFn	https://tronscan.org/#/contract/TMKYuA8JuJJwrqtrwcayRHADXTYDZNveFn/code
MoonPoolJSTReward	TVuvY19L6BjiST9bNQrYAasNE6wJCXocnq	https://tronscan.org/#/contract/TVuvY19L6BjiST9bNQrYAasNE6wJCXocnq/code
MoonPoolPearlReward	TUrkZvNhNb9vtT7cD9rchbvFD3CYuyW2n4	https://tronscan.org/#/contract/TUrkZvNhNb9vtT7cD9rchbvFD3CYuyW2n4/code
MoonPoolTAIReward	TQJCpKfhjRXmXLSdNVQbNaVrJcoNTE1CcH	https://tronscan.org/#/contract/TQJCpKfhjRXmXLSdNVQbNaVrJcoNTE1CcH/code
MoonPoolUSDJReward	TWgbZoeDrSPcPF2B6ye291iLenhimpwaZj	https://tronscan.org/#/contract/TWgbZoeDrSPcPF2B6ye291iLenhimpwaZj/code

Start Date：2020.09.09

Completion Date：2020.09.10

Overall Result：Pass

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

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass

		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	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 contracts UniswapRewards, including Coding Standards, Security, and Business Logic. **The UniswapRewards contract passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing TRX.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check the weather the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

Check whether the business is secure.

3.1 Stake Initialization

- Description:

  The "stake-reward" mode of the contract needs to initialize the relevant parameters (*rewardRate*, *lastUpdateTime*, *periodFinish*), call the *notifyRewardAmount* function by the specified reward distribution manager address *rewardDistribution*, and enter the initial reward used to calculate the *rewardRate*, initialize the stake and reward related parameters.

- Related functions: *notifyRewardAmount*, *rewardPerToken*, *lastTimeRewardApplicable*
- Result: Pass

3.2 Stake y tokens

- Description:

  The contract implements the *stake* function to stake the y tokens. The user approve the contract address in advance. By calling the *transferFrom* function in the y contract, the contract address transfers the specified amount of y tokens to the contract address on behalf of the user; This function restricts the user to call only after the "stake-reward" mode is turned on (the specified time is reached); each time this function is called to stake tokens, the reward related data is updated through the modifier *updateReward*; and each call is checked whether the *periodFinish* is reached by the modifier *checkhalve*, and the reward halving operation is performed and the *rewardRate* and the *periodFinish* are updated.

- Related functions: *stake*, *rewardPerToken*, *lastTimeRewardApplicable*, *earned*, *balanceOf*
- Result: Pass

3.3 Withdraw y tokens

- Description:

  The contract implements the *withdraw* function to withdraw the y tokens. By calling the *transfer* function in the y contract, the contract address transfers the specified amount of y tokens to the user; This function restricts the user to call only after the "stake-reward" mode is turned on (the specified time is reached); each time this function is called to stake tokens, the reward related data is updated through the modifier *updateReward*; and each call is checked whether the *periodFinish* is reached by the modifier *checkhalve*, and the reward halving operation is performed and the *rewardRate* and the *periodFinish* are updated.

- Related functions: *withdraw*, *rewardPerToken*, *lastTimeRewardApplicable*, *earned*, *balanceOf*
- Result: Pass

3.4 Withdraw rewards (MFI token)

- Description:

The contract implements the *getReward* function to withdraw the rewards (MFI token). By calling the *transfer* function in the MFI contract, the contract address transfers the specified amount (all rewards of caller) of MFI tokens to the user; When the user withdraws the reward, this function mints 10% of the amount of the reward to the governance management address; This function restricts the user to call only after the "stake-reward" mode is turned on (the specified time is reached); each time this function is called to stake tokens, the reward related data is updated through the modifier *updateReward*; and each call is checked whether the *periodFinish* is reached by the modifier *checkhalve*, and the reward halving operation is performed and the *rewardRate* and the *periodFinish* are updated.

- Related functions: *getReward, rewardPerToken, lastTimeRewardApplicable, earned, balanceOf*

- Result: Pass

3.5 Exit the stake participation

- Description:

The contract implements the *exit* function to close the participation of "stake-reward" mode. Call the *withdraw* function to withdraw all stake y tokens, call the *getReward* function to receive all rewards. The user address cannot get new rewards because the balance of y tokens already staked is empty.

- Related functions: *exit, withdraw, getReward, rewardPerToken, lastTimeRewardApplicable, earned, balanceOf*

- Result: Pass

3.6 Reward related data query function

- Description:

Contract users can query the earliest timestamp between the current timestamp and the *periodFinish* by calling the *lastTimeRewardApplicable* function; calling the *rewardPerToken* function can query the gettable rewards for each stake y token; calling the *earned* function can query the total gettable stake rewards of the specified address.

- Related functions: *lastTimeRewardApplicable, rewardPerToken, earned*

- Result: Pass

Audited Source Code with Comments:

In this project, five smart contracts for the liquidity mining pool were implemented based on the same code architecture, namely *MoonPoolCRTReward, MoonPoolJSTReward, MoonPoolPearlReward*, *MoonPoolTAIReward*, and *MoonPoolUSDJReward*. The implementation codes of these five smart contracts are different in the same position. The specific differences are in the table 1, the following contract source code audit comments are the content of the *MoonPoolCRTReward* contract.

Contract Name	Contract implementation code differences	Note

MoonPoolCRTReward	**// Beosin (Chengdu LianAn) // At line 596 and line 600, contract name and the stake liquidity y token contract address are implemented differently.**
	contract LPTokenCRTWrapper {
	    using SafeMath for uint256;
		The stake liquidity y token contract address: 0x41fdcfd438d2f94fa3acb0891f18128e

	using SafeERC20 for IERC20;
	IERC20 public y = IERC20(0x41fdcfd438d2f94fa3acb0891f18128e27032ddc87); // Stake Token address
	// Beosin (Chengdu LianAn) // At line 626 and line 630, the contract name and the value of the initial token reward ‘initreward’ are implemented differently.
	contract MoonPoolCRTReward is LPTokenCRTWrapper, IRewardDistributionRecipient {
	IERC20 public mfi = IERC20(0x4177fd35cfd3349cb1ca14b9e71b5b7fd373331878); // MFI Token address
	uint256 public constant DURATION = 2 days;
	uint256 public initreward = 750*1e18;
		27032ddc87
		The value of the initial token reward ‘initreward’ is 750*1e18

MoonPoolJSTReward	**// Beosin (Chengdu LianAn) // At line 596 and line 600, contract name and the stake liquidity y token contract address are implemented differently.**
	contract LPTokenJSTWrapper {
	using SafeMath for uint256;
	using SafeERC20 for IERC20;
	IERC20 public y = IERC20(0x4118fd0626daf3af02389aef3ed87db9c33f638ffa); // Stake Token address
	// Beosin (Chengdu LianAn) // At line 626 and line 630, the contract name and the value of the initial token reward ‘initreward’ are implemented differently.
	contract MoonPoolJSTReward is LPTokenJSTWrapper, IRewardDistributionRecipient {
	IERC20 public mfi = IERC20(0x4177fd35cfd3349cb1ca14b9e71b5b7fd373331878); // MFI Token address
	uint256 public constant DURATION = 2 days;
		The stake liquidity y token contract address: 0x4118fd0626daf3af02389aef3ed87db9c33f638ffa
		The value of the initial token reward ‘initreward’ is 250*1e18

	uint256 public initreward = 250*1e18;	

MoonPoolPearlReward	**// Beosin (Chengdu LianAn) // At line 596 and line 600, contract name and the stake liquidity y token contract address are implemented differently.**
	contract LPTokenPearlWrapper {
	using SafeMath for uint256;
	using SafeERC20 for IERC20;
	IERC20 public y = IERC20(0x4148c125e0d3c626842bf7180c85e79f97ae524e91); // Stake Token address
	// Beosin (Chengdu LianAn) // At line 626 and line 630, the contract name and the value of the initial token reward ‘initreward’ are implemented differently.
	contract MoonPoolPearlReward is LPTokenPearlWrapper, IRewardDistributionRecipient {
	IERC20 public mfi = IERC20(0x4177fd35cfd3349cb1ca14b9e71b5b7fd373331878); // MFI Token address
	uint256 public constant DURATION = 2 days;
	uint256 public initreward = 750*1e18;
		The stake liquidity y token contract address: 0x4148c125e0d3c626842bf7180c85e79f97ae524e91
		The value of the initial token reward ‘ initreward ’ is 750*1e18

MoonPoolTAIReward	**// Beosin (Chengdu LianAn) // At line 596 and line 600, contract name and the stake liquidity y token contract address are implemented differently.**
	contract LPTokenTAIWrapper {
	using SafeMath for uint256;
	using SafeERC20 for IERC20;
	IERC20 public y = IERC20(0x41af2c205a7e44f79f680d149d339b733f6d34b6d5); // Stake Token address
	// Beosin (Chengdu LianAn) // At line 626 and line 630, the contract
		The stake liquidity y token contract address: 0x41af2c205a7e44f79f680d149d339b733f6d34b6d5
		The value of the initial token reward ‘ initreward ’ is 750*1e18

	name and the value of the initial token reward ‘initreward’ are implemented differently.
	contract MoonPoolTAIReward is LPTokenTAIWrapper, IRewardDistributionRecipient {
	    IERC20 public mfi = IERC20(0x4177fd35cfd3349cb1ca14b9e71b5b7fd373331878); // MFI Token address
	    uint256 public constant DURATION = 2 days;
	    uint256 public initreward = 750*1e18;	

MoonPoolUSDJReward	**// Beosin (Chengdu LianAn) // At line 596 and line 600, contract name and the stake liquidity y token contract address are implemented differently.**
	contract LPTokenUSDJWrapper {
	    using SafeMath for uint256;
	    using SafeERC20 for IERC20;
	    IERC20 public y = IERC20(0x41834295921a488d9d42b4b3021ed1a3c39fb0f03e); // Stake Token address
	// Beosin (Chengdu LianAn) // At line 626 and line 630, the contract name and the value of the initial token reward ‘initreward’ are implemented differently.
	contract MoonPoolUSDJReward is LPTokenUSDJWrapper, IRewardDistributionRecipient {
	    IERC20 public mfi = IERC20(0x4177fd35cfd3349cb1ca14b9e71b5b7fd373331878); // MFI Token address
	    uint256 public constant DURATION = 2 days;
	    uint256 public initreward = 250*1e18;
		The stake liquidity y token contract address: 0x41834295921a488d9d42b4b3021ed1a3c39fb0f03e
		The value of the initial token reward ‘initreward’ is 250*1e18

Table 1 Contract implementation code differences

```
/*
     _____                    __    __            __    _
   / ___/__ __ ____    / /_/ /    ___    / /_(_)__ __
  _\ \/ // // _ \/ __// _ \/ -_)/ __// / \ \ /
/____/ \_, //_//_/\__//_//_/\__/ \__//_/ /_\_\
         /___/

* Synthetix: MFIReward.sol
*
* https://moonpool.finance/
*
*
* MIT License
* ===========
*
* Copyright (c) 2020 Synthetix
*
* 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
*/

// File: @openzeppelin/contracts/math/Math.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
  * @dev Standard math utilities missing in the Solidity language.
  */
library Math {
        /**
          * @dev Returns the largest of two numbers.
          */
        function max(uint256 a, uint256 b) internal pure returns (uint256) {
            return a >= b ? a : b;
        }

        /**
          * @dev Returns the smallest of two numbers.
          */
        function min(uint256 a, uint256 b) internal pure returns (uint256) {
            return a < b ? a : b;
        }

        /**
          * @dev Returns the average of two numbers. The result is rounded towards
          * zero.
          */
        function average(uint256 a, uint256 b) internal pure returns (uint256) {
            // (a + b) / 2 can overflow, so we distribute
            return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
        }
}

// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
  * @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.
     *
     * _Available since v2.4.0._
     */
    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.
     *
     * _Available since v2.4.0._
     */
    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.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/*
 * @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 { }
	// solhint-disable-previous-line no-empty-blocks
	// 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;
	}
}

// File: @openzeppelin/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @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.
 *
 * 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 {
		_owner = _msgSender();
		emit OwnershipTransferred(address(0), _owner); // Beosin (Chengdu LianAn) // Trigger 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(isOwner(), "Ownable: caller is not the owner"); // Beosin (Chengdu LianAn) // Modifier, require that the caller of the modified function must be owner.
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _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)); // 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); // Beosin (Chengdu LianAn) // Call the internal function '_transferOwnership' to transfer ownership.
    }

    /**
     * @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'. Avoid losing ownership.
        emit OwnershipTransferred(_owner, newOwner); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
        _owner = newOwner; // Beosin (Chengdu LianAn) // Transfer ownership to 'newOwner'.
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    // Beosin (Chengdu LianAn) // Define the function interfaces required by TRC20 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);
    function mint(address account, uint amount) external;

    /**
     * @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);
        // 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);
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.4; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

/**
  * @dev Collection of functions related to the address type
  */
```

```
library Address {
        /**
          * @dev Returns true if `account` is a contract.
          *
          * This test is non-exhaustive, and there may be false-negatives: during the
          * execution of a contract's constructor, its address will be reported as
          * not containing a contract.
          *
          * IMPORTANT: It is unsafe to assume that an address for which this
          * function returns false is an externally-owned account (EOA) and not a
          * contract.
          */
        function isContract(address account) internal view returns (bool) {
                // This method relies in extcodesize, which returns 0 for contracts in
                // construction, since the code is only stored at the end of the
                // constructor execution.

                // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
                // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
                // for accounts without code, i.e. `keccak256('')`
                bytes32 codehash;
                bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
                // solhint-disable-next-line no-inline-assembly
                assembly { codehash := extcodehash(account) }
                return (codehash != 0x0 && codehash != accountHash);
        }

        /**
          * @dev Converts an `address` into `address payable`. Note that this is
          * simply a type cast: the actual underlying value is not changed.
          *
          * _Available since v2.4.0._
          */
        function toPayable(address account) internal pure returns (address payable) {
                return address(uint160(account));
        }

        /**
          * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
          * `recipient`, forwarding all available gas and reverting on errors.
          *
          * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
          * of certain opcodes, possibly making contracts go over the 2300 gas limit
          * imposed by `transfer`, making them unable to receive funds via
          * `transfer`. {sendValue} removes this limitation.
          *
          * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
```

```
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     *
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.




/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.
    using Address for address; // Beosin (Chengdu LianAn) // Use the funcions of Address library to check whether the specified address is contract address.

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

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

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

        function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
                uint256 newAllowance = token.allowance(address(this), spender).add(value);
                callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender,
newAllowance));
        }

        function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
                uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20:
decreased allowance below zero");
                callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender,
newAllowance));
        }

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

                // A Solidity high level call has three parts:
                //    1. The target address is checked to verify it contains contract code
                //    2. The call itself is made, and success asserted
                //    3. The return value is decoded, which in turn checks the size of the returned data.
                // solhint-disable-next-line max-line-length
                require(address(token).isContract(), "SafeERC20: call to non-contract");

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

                if (returndata.length > 0) { // Return data is optional
                // solhint-disable-next-line max-line-length
                require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
            }
        }
}

// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.



contract IRewardDistributionRecipient is Ownable {
        address rewardDistribution; // Beosin (Chengdu LianAn) // Declare the variable 'rewardDistribution'
for storing the reward distribution address.

        function notifyRewardAmount(uint256 reward) external; // Beosin (Chengdu LianAn) // Define the
function interface of 'notifyRewardAmount'.
        // Beosin (Chengdu LianAn) // Modifier, require that the caller should be 'rewardDistribution'.
        modifier onlyRewardDistribution() {
            require(_msgSender() == rewardDistribution, "Caller is not reward distribution");
            _;
        }
        // Beosin (Chengdu LianAn) // The function 'setRewardDistribution' is defined to update the
reward distribution address.
        function setRewardDistribution(address _rewardDistribution)
            external
            onlyOwner
        {
            rewardDistribution = _rewardDistribution;
        }
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.






contract LPTokenCRTWrapper {
        using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for
mathematical operation. Avoid integer overflow/underflow.
        using SafeERC20 for IERC20; // Beosin (Chengdu LianAn) // Use the SafeERC20 library for safe
external TRC20 contract calling.
```

```
    IERC20 public y = IERC20(0x41fdcfd438d2f94fa3acb0891f18128e27032ddc87); // Stake Token address
// Beosin (Chengdu LianAn) // Declare the external Liquidity Pool Token contract.

    uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for
storing the total stake token (y).
    mapping(address => uint256) private _balances; // Beosin (Chengdu LianAn) // Declare the mapping
variable '_balances' for storing the stake token (y) balance of corresponding address.
    // Beosin (Chengdu LianAn) // The function 'totalSupply' is defined to query the total stake token
(y).
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }
    // Beosin (Chengdu LianAn) // The function 'balanceOf' is defined to query the stake token (y)
balance of the specified address 'account'.
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
    // Beosin (Chengdu LianAn) // The function 'stake' is defined to stake y token to this contract.
    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the amount of
total stake token.
        _balances[msg.sender] = _balances[msg.sender].add(amount); // Beosin (Chengdu LianAn) //
Alter the stake token balance of caller.
        y.safeTransferFrom(msg.sender, address(this), amount); // Beosin (Chengdu LianAn) // Call the
'transferFrom' function of specified TRC20 contract via the function 'safeTransferFrom' in SafeERC20
library, this contract delegate the caller transfers the specified amount of stake tokens to this contract.
    }
    // Beosin (Chengdu LianAn) // The function 'withdraw' is defined to withdraw the staked y tokens
to caller itself.
    function withdraw(uint256 amount) public {
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update the amount
oftotal stake token.
        _balances[msg.sender] = _balances[msg.sender].sub(amount); // Beosin (Chengdu LianAn) //
Alter the stake token balance of caller.
        y.safeTransfer(msg.sender, amount); // Beosin (Chengdu LianAn) // Call the 'transfer' function
of specified TRC20 contract via the function 'safeTransfer' in SafeERC20 library, this contract
transfers the specified amount of stake tokens to the function caller.
    }
}

contract MoonPoolCRTReward is LPTokenCRTWrapper, IRewardDistributionRecipient {
    IERC20 public mfi = IERC20(0x4177fd35cfd3349cb1ca14b9e71b5b7fd373331878);    // MFI Token
address    // Beosin (Chengdu LianAn) // Declare the external MFI Token contract.
    uint256 public constant DURATION = 2 days; // Beosin (Chengdu LianAn) // Declare the constant
'DURATION' for storing the fixed duration of one reward calculation period.

    uint256 public initreward = 750*1e18; // Beosin (Chengdu LianAn) // Declare the variable
'initreward' for storing the initial value used for calculating reward rate.
```

```
    uint256 public starttime = 1599310800; //Saturday, 5 September 2020 13:00:00 (UTC)
    uint256 public periodFinish = 0; // Beosin (Chengdu LianAn) // Declare the variable 'periodFinish' for storing the end timestamp of one reward calculation period.
    uint256 public rewardRate = 0; // Beosin (Chengdu LianAn) // Declare the variable 'rewardRate' for storing the reward calculation rate.
    uint256 public lastUpdateTime; // Beosin (Chengdu LianAn) // Declare the variable 'lastUpdateTime' for storing the timestamp of last update time.
    uint256 public rewardPerTokenStored; // Beosin (Chengdu LianAn) // Declare the variable 'rewardPerTokenStored' for storing the gettable reward amount per stake token during the dynamic period.
    address public governance; // Beosin (Chengdu LianAn) // Declare the variable 'governance' for storing the governance management address.
    mapping(address => uint256) public userRewardPerTokenPaid; // Beosin (Chengdu LianAn) // Declare the mapping variable 'userRewardPerTokenPaid' for storing the paid(calculated) reward per stake token.
    mapping(address => uint256) public rewards; // Beosin (Chengdu LianAn) // Declare the mapping variable 'rewards' for storing the total reward amount of corresponding address.
    // Beosin (Chengdu LianAn) // Declare the relevant event.
    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);
    // Beosin (Chengdu LianAn) // Modifier 'updateReward' is defined to update the reward relevant data.
    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken(); // Beosin (Chengdu LianAn) // Call the function 'rewardPerToken' to get the newest gettable reward amount per stake token during the dynamic period.
        lastUpdateTime = lastTimeRewardApplicable(); // Beosin (Chengdu LianAn) // Call the function 'lastTimeRewardApplicable' to get the earliest timestamp between the current time and 'periodFinish'.
        if (account != address(0)) {
            rewards[account] = earned(account); // Beosin (Chengdu LianAn) // Update the reward of 'account'.
            userRewardPerTokenPaid[account] = rewardPerTokenStored; // Beosin (Chengdu LianAn) // Update the paid(calculated) reward per stake token of 'account'.
        }
        _;
    }
    // Beosin (Chengdu LianAn) // Constructor, initialize the governance management address.
    constructor (address g) public{
      governance = g;
    }
    // Beosin (Chengdu LianAn) // The function 'lastTimeRewardApplicable' is defined to return the earliest timestamp between the current time and 'periodFinish'.
    function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }
    // Beosin (Chengdu LianAn) // The function 'rewardPerToken' is defined to calculate the newest
```

```
gettable reward amount per stake token during the dynamic period.
    function rewardPerToken() public view returns (uint256) {
        // Beosin (Chengdu LianAn) // If the total stake token amount is 0, return the current
'rewardPerTokenStored'.
        if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime) // Beosin (Chengdu LianAn) // Calculate the passed time
period after the last update time.
                    .mul(rewardRate) // Beosin (Chengdu LianAn) // Calculate the reward amount
should get during the period.
                    .mul(1e18) // Beosin (Chengdu LianAn) // Token decimals format conversion.
                    .div(totalSupply())
            );
    }
    // Beosin (Chengdu LianAn) // The function 'earned' is defined to query the reward amount of
specified address 'account'.
    function earned(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account])) // Beosin (Chengdu
LianAn) // Calculate the newly gettable reward during period.
                .div(1e18) // Beosin (Chengdu LianAn) // Token decimals format conversion.
                .add(rewards[account]); // Beosin (Chengdu LianAn) // The total reward amount of
'account'.
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'stake', added the modifiers 'updateReward',
'checkhalve' and 'checkStart'.
    // stake visibility is public as overriding LPTokenWrapper's stake() function
    function stake(uint256 amount) public updateReward(msg.sender) checkhalve checkStart{
        require(amount > 0, "Cannot stake 0");
        super.stake(amount); // Beosin (Chengdu LianAn) // Execute the function 'stake' in parent
contract.
        emit Staked(msg.sender, amount); // Beosin (Chengdu LianAn) // Trigger the event 'Staked'.
    }
    // Beosin (Chengdu LianAn) // Rewrite the function 'stake',added the modifiers 'updateReward',
'checkhalve' and 'checkStart'.
    function withdraw(uint256 amount) public updateReward(msg.sender) checkhalve checkStart{
        require(amount > 0, "Cannot withdraw 0");
        super.withdraw(amount); // Beosin (Chengdu LianAn) // Execute the function 'withdraw' in
parent contract.
        emit Withdrawn(msg.sender, amount); // Beosin (Chengdu LianAn) // Trigger the event
'Withdrawn'.
    }
    // Beosin (Chengdu LianAn) // The function 'exit' is defined for callers to withdraw their all stake
```

```
tokens and rewards.
    function exit() external {
        withdraw(balanceOf(msg.sender)); // Beosin (Chengdu LianAn) // Call the function 'withdraw'
to withdraw all stake tokens of caller.
        getReward(); // Beosin (Chengdu LianAn) // Call the function 'getReward' to withdraw all
rewards of caller.
    }
    // Beosin (Chengdu LianAn) // The function 'getReward' is defined to withdraw rewards.
    function getReward() public updateReward(msg.sender) checkhalve checkStart{
        uint256 reward = earned(msg.sender); // Beosin (Chengdu LianAn) // Declare the local variable
'reward' for recording the gettable rewards of caller.
        if (reward > 0) { // Beosin (Chengdu LianAn) // Require that the 'reward' should be greater
than 0.
            rewards[msg.sender] = 0; // Beosin (Chengdu LianAn) // Reset the rewards of caller to 0.
            mfi.safeTransfer(msg.sender, reward); // Beosin (Chengdu LianAn) // Call the 'transfer'
function of specified TRC20 contract via the function 'safeTransfer' in SafeERC20 library, this contract
transfers the specified amount of MFI tokens to the function caller.
            mfi.mint(governance, reward.div(10)); // Beosin (Chengdu LianAn) // Call the function
'mint' of MFI contract to mint tokens to 'governance'.
            emit RewardPaid(msg.sender, reward); // Beosin (Chengdu LianAn) // Trigger the event
'RewardPaid'.
        }
    }
    // Beosin (Chengdu LianAn) // Modifier, do the halve operation and other update operations when
the time reaches the 'periodFinish'.
    modifier checkhalve(){
        if (block.timestamp >= periodFinish) {
            initreward = initreward.mul(50).div(100); // Beosin (Chengdu LianAn) // Do the halve
operation

            mfi.mint(address(this),initreward); // Beosin (Chengdu LianAn) // Call the function 'mint' of
MFI contract to mint tokens to this contract address.

            rewardRate = initreward.div(DURATION); // Beosin (Chengdu LianAn) // Update the
'rewardRate'.
            periodFinish = block.timestamp.add(DURATION); // Beosin (Chengdu LianAn) // Update
the 'periodFinish'.
            emit RewardAdded(initreward); // Beosin (Chengdu LianAn) // Trigger the event
'RewardAdded'.
        }
        _;
    }
    // Beosin (Chengdu LianAn) // Modifier, require that the modified function can only be called when
the time reaches the start time.
    modifier checkStart(){
        require(block.timestamp > starttime,"not start");
        _;
    }
    // Beosin (Chengdu LianAn) // Calculate the 'rewardRate' according to the specified 'reward' and update the 'lastUpdateTime' and 'periodFinish'.
    // Beosin (Chengdu LianAn) // Note: this function only can be called by 'rewardDistribution' to adjust/control the 'rewardRate' and other timestamps.
    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution // Beosin (Chengdu LianAn) // Require that the caller should be 'rewardDistribution'.
        updateReward(address(0)) // Beosin (Chengdu LianAn) // Update the 'rewardPerTokenStored' and 'lastUpdateTime'.
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION); // Beosin (Chengdu LianAn) // Update the 'rewardRate' when the time reaches the 'periodFinish'.
        } else { // Beosin (Chengdu LianAn) // Otherwise, calculate the left gettable rewards and update the corresponding 'rewardRate'.
            uint256 remaining = periodFinish.sub(block.timestamp); // Beosin (Chengdu LianAn) // Calculate the remaining time in this period.
            uint256 leftover = remaining.mul(rewardRate); // Beosin (Chengdu LianAn) // Calculate the left gettable rewards according to the current 'rewardRate'.
            rewardRate = reward.add(leftover).div(DURATION); // Beosin (Chengdu LianAn) // Update the 'rewardRate'.
        }

        mfi.mint(address(this),reward); // Beosin (Chengdu LianAn) // Call the function 'mint' of MFI contract to mint tokens to this contract address.

        lastUpdateTime = block.timestamp; // Beosin (Chengdu LianAn) // Update the 'lastUpdateTime' to the current time.
        periodFinish = block.timestamp.add(DURATION); // Beosin (Chengdu LianAn) // Update the 'periodFinish' to the corresponding time.
        emit RewardAdded(reward); // Beosin (Chengdu LianAn) // Trigger the event 'RewardAdded'.
    }
}
```

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com