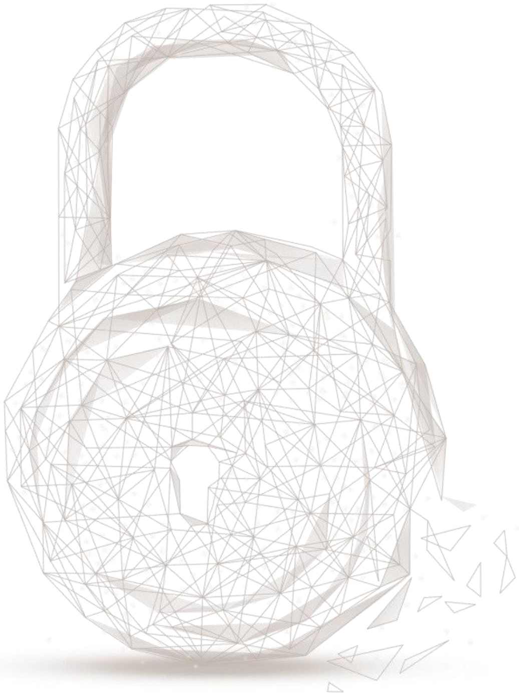
Beosin

成都链安

BEOSIN

Smart contract security audit report



Audit Number：202008191642

Smart Contract Name：

UniswapRewards

Smart Contract Address Link：

https://github.com/dcocos-finance/reward-contract.git

Commit Hash：

7bfe1c273989b9ab60ec0a643fe3fdb7f7175ea1

Start Date：2020.08.13

Completion Date：2020.08.19

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 ETH.
- 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 UNI-V2 tokens

- Description:

  The contract implements the *stake* function to stake the UNI-V2 tokens. The user approve the contract address in advance. By calling the *transferFrom* function in the UNI-V2 contract, the contract address transfers the specified amount of UNI-V2 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 Witudraw UNI-V2 tokens

- Description:

  The contract implements the *withdraw* function to withdraw the UNI-V2 tokens. By calling the *transfer* function in the UNI-V2 contract, the contract address transfers the specified amount of UNI-V2 tokens to the 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*.

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

- Result: Pass

3.4 Witudraw rewards (dCOCOS token)

- Description:

The contract implements the *getReward* function to withdraw the rewards (dCOCOS token). By calling the *transfer* function in the dCOCOS contract, the contract address transfers the specified amount (all rewards of caller) of dCOCOS tokens to the 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: *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 UNI-V2 tokens, call the *getReward* function to receive all rewards. The user address cannot get new rewards because the balance of UNI-V2 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 UNI-V2 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:

```
// Beosin (Chengdu LianAn) // File:IRewardDistributionRecipient.sol
pragma solidity ^0.5.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.

import "@openzeppelin/contracts/ownership/Ownable.sol";


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

        function notifyRewardAmount(uint256 reward) external; // Beosin (Chengdu LianAn) // Decalre the
function interface of 'notifyRewardAmount'.
        // Beosin (Chengdu LianAn) // Modifier, require that the caller of the modified function must be
'rewardDistribution'.
        modifier onlyRewardDistribution() {
                require(_msgSender() == rewardDistribution, "Caller is not reward distribution");
                _;
        }
        // Beosin (Chengdu LianAn) // The function 'setRewardDistribution' is defined to set the manager
address of reward distribution.
```

```
        function setRewardDistribution(address _rewardDistribution)
                external
                onlyOwner // Beosin (Chengdu LianAn) // Require that the caller should be the owner of this contract.
        {
                rewardDistribution = _rewardDistribution; // Beosin (Chengdu LianAn) // Set the 'rewardDistribution' to '_rewardDistribution'.
        }
}


// Beosin (Chengdu LianAn) // File:UniswapRewards.sol
/// @title UniswapRewards Contract
/// For more information about this token please visit https://dcocos.finance
/// @author reedhong
/**
/*
     _____                    __     __            __     _
    / ___/__ __ ____    / /_ / /    ____    / /_ (_)__ __
   _\ \ / // // _ \/ __// _ \/ -_)/ __// / \ \ /
/____/ \_, //_//_/\__//_//_/\__/ \__//_/ /_\_\
          /___/
* Synthetix: CurveRewards.sol
*
* Docs: https://docs.synthetix.io/
*
*
* 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
*/

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

import "@openzeppelin/contracts/math/Math.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/ownership/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./IRewardDistributionRecipient.sol";



contract LPTokenWrapper {
    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 ERC20 contract calling.

    IERC20 public uniswapLP = IERC20(0x16cAC1403377978644e78769Daa49d8f6B6CF565); // Beosin (Chengdu LianAn) // Declare the external Balancer Pool Token contract.

    uint256 private _totalSupply; // Beosin (Chengdu LianAn) // Declare the variable '_totalSupply' for storing the total stake token (UNI-V2).
    mapping(address => uint256) private _balances; // Beosin (Chengdu LianAn) // Declare the mapping variable '_balances' for storing the stake token (UNI-V2) balance of corresponding address.
    // Beosin (Chengdu LianAn) // The function 'totalSupply' is defined to query the total stake token (UNI-V2).
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }
    // Beosin (Chengdu LianAn) // The function 'balanceOf' is defined to query the stake token (UNI-V2) 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 UNI-V2 token to this contract.
    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount); // Beosin (Chengdu LianAn) // Update the total stake token.
        _balances[msg.sender] = _balances[msg.sender].add(amount); // Beosin (Chengdu LianAn) // Alter the stake token balance of caller.
        uniswapLP.safeTransferFrom(msg.sender, address(this), amount); // Beosin (Chengdu LianAn) // Call the 'transferFrom' function of specified ERC20 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 stake UNI-V2 token to caller itself.
    function withdraw(uint256 amount) public {
        _totalSupply = _totalSupply.sub(amount); // Beosin (Chengdu LianAn) // Update the total stake token.
        _balances[msg.sender] = _balances[msg.sender].sub(amount); // Beosin (Chengdu LianAn) // Alter the stake token balance of caller.
        uniswapLP.safeTransfer(msg.sender, amount); // Beosin (Chengdu LianAn) // Call the 'transfer' function of specified ERC20 contract via the function 'safeTransfer' in SafeERC20 library, this contract transfers the specified amount of stake tokens to the function caller.
    }
}

contract UniswapRewards is LPTokenWrapper, IRewardDistributionRecipient {
    using SafeERC20 for IERC20; // Beosin (Chengdu LianAn) // Use the SafeERC20 library for safe external ERC20 contract calling.

    IERC20 public dCOCOS = IERC20(0xa1d0E215a23d7030842FC67cE582a6aFa3CCaB83); // Beosin (Chengdu LianAn) // Declare the external dCOCOS Token contract.
    uint256 public constant DURATION = 7 days; // Beosin (Chengdu LianAn) // Declare the constant 'DURATION' for storing the fixed duration of one reward calculation period.

    uint256 public initreward = 100000000*1e18; // Beosin (Chengdu LianAn) // Declare the variable 'initreward' for storing the initial value used for calculating reward rate.
    uint256 public starttime = 1595865600; //utc+8 2020 07-28 0:00:00
    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.
    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) // 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' and 'checkStart'.
    function withdraw(uint256 amount) public updateReward(msg.sender) 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.
            dCOCOS.safeTransfer(msg.sender, reward); // Beosin (Chengdu LianAn) // Call the 'transfer' function of specified ERC20 contract via the function 'safeTransfer' in SafeERC20 library, this contract transfers the specified amount of dCOCOS tokens to the function caller.
            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
            //dCOCOS.mint(address(this),initreward);
            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'.
        }

        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'.
    }
}
```

Beosin

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com