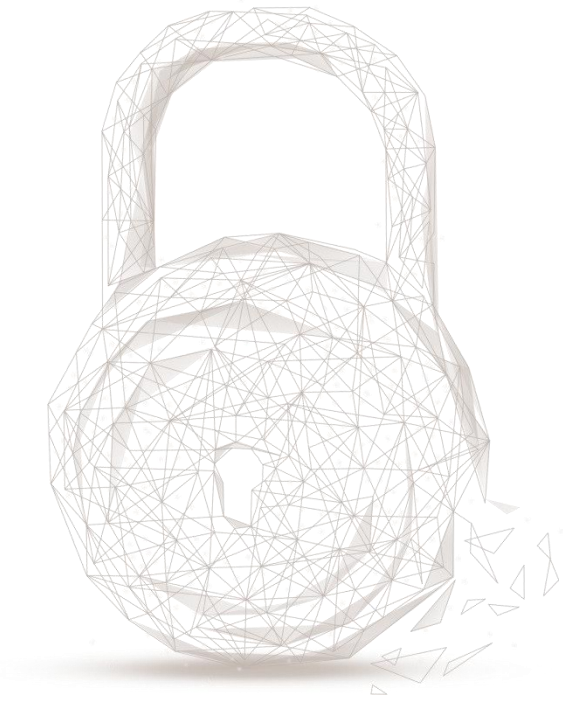
成都链安
BEOSIN

Smart contract security audit report



Audit Number：202008221847

Smart Contract Name：

finNexusLiquidity

Smart Contract Address Link：

https://github.com/FinNexus/finNexusLiquidity

Commit Hash：

dc8b3dce5694c1f069982becd24cd4c75dfc1d16

No	Path
1	/contracts/IStaking.sol
2	/contracts/TokenLiquidity.sol
3	/contracts/TokenPool.sol

Start Date：2020.08.20

Completion Date：2020.08.22

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 finNexusLiquidity,

including Coding Standards, Security, and Business Logic. **The finNexusLiquidity 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 tokens

3.1.1 Stake for caller (beneficiary)

- Description:

  The TokenLiquidity contract implements the *stake* function (actually call the function *_stakeFor*) for the users to deposit tokens to specified token pool. The stake operation requires that the current time should be earlier than the expiration time and the deposit amount should be greater than zero. The gettable stake rewards when the stake keeping period reaches the specified bonus period will be recorded. The user pre-approve this contract (TokenLiquidity) address, by calling the *transferFrom* function in the specified ERC20 Token contract, this contract address as a delegate of caller transfers the specified amount of specified tokens to the specified TokenPool contract (_stakingPool) address.

- Related functions: *stake*, *_stakeFor*, *transferFrom*
- Result: Pass

3.1.1 Stake for user (beneficiary)

- Description:

  The TokenLiquidity contract implements the *stakeFor* function (actually call the function *_stakeFor*) for the users to deposit tokens to specified token pool on behalf of user (beneficiary). The stake operation requires that the current time should be earlier than the expiration time and the deposit amount should be greater than zero. The gettable stake rewards when the stake keeping period reaches the specified bonus period will be recorded. The user pre-approve this contract (TokenLiquidity) address, by calling the *transferFrom* function in the specified ERC20 Token contract, this contract address as a delegate of caller transfers the specified amount of specified tokens to the specified TokenPool contract (_stakingPool) address.

- Related functions: *stakeFor*, *_stakeFor*, *transferFrom*
- Result: Pass

3.2 Withdraw tokens

3.2.1 Withdraw certain amount of deposited tokens

- Description:

  The TokenLiquidity contract implements the *unstake* function for users to withdraw a certain amount of deposited tokens. It is required that the withdraw amount should be greater than zero and the withdraw amount should be less than the total stake amount. The function *computeNewReward* is called to calculate the gettable reward amount. Call the *transfer* function of the specified ERC20 Token contract, the contract _stakingPool transfers tokens to this function caller. And call the *transfer* function of the specified ERC20 Token contract, the contract _distributionPool transfers tokens to this function caller as reward.

- Related functions: *unstake, totalStakedFor, computeNewReward, transfer*
- Result: Pass

3.2.2 Withdraw all of deposited tokens

- Description:

  The TokenLiquidity contract implements the *unstakeAll* function for users to withdraw all deposited tokens. It is required that the withdraw amount should be greater than zero. The function *computeNewReward* is called to calculate the gettable reward amount. Call the *transfer* function of the specified ERC20 Token contract, the contract _stakingPool transfers tokens to this function caller. And call the *transfer* function of the specified ERC20 Token contract, the contract _distributionPool transfers tokens to this function caller as reward.

- Related functions: *unstakeAll, totalStakedFor, computeNewReward, transfer*
- Result: Pass

3.2.3 Query gettable rewards & withdraw certain amount of deposited tokens

- Description:

  The TokenLiquidity contract implements the *unstakeQuery* function for users to query the gettable rewards and withdraw a certain amount of deposited tokens. It is required that the withdraw amount should be greater than zero and the withdraw amount should be less than the total stake amount. The function *computeNewReward* is called to calculate the gettable reward amount. Call the *transfer* function of the specified ERC20 Token contract, the contract _stakingPool transfers tokens to this function caller. And call the *transfer* function of the specified ERC20 Token contract, the contract _distributionPool transfers tokens to this function caller as reward.

- To be confirmed:

  The actual function of the function is different from the function naming. It is recommended to add corresponding query function. If it conforms to the project design, please ignore it.

- Related functions: *unstakeQuery*
- Result: Pass

3.2.4 Withdraw all the specified tokens of contract '_distributionPool'

- Description:

  The TokenLiquidity contract implements the *unlockToken* function for contract owner to withdraw all the specified tokens of contract '_distributionPool'

- Related functions: *unlockToken, transfer*
- Result: Pass

4. Other Audit Suggestions

- *setInitialSharesPerToken* function

  As shown in Figure 1 below, the *setInitialSharesPerToken* function does not do a non-zero check when setting _initialSharesPerToken, which may cause the stake to be unsuccessful.

```
108     function setInitialSharesPerToken(uint init) public onlyOwner {
109         _initialSharesPerToken = init;
110     }
```

Figure 1 *setInitialSharesPerToken* source code

- *_unstake* function

  As shown in the figure below, when the user withdraws stake tokens, the withdrawal amount should be close to the amount of the last stake. Otherwise, when the user has too many stakes and the user withdraws too much at a time, the while loop will cause out of gas to fail the withdrawal operation.

```
220         while (amountLeft > 0) {
221             Stake storage lastStake = accountStakes[accountStakes.length - 1];
222 //            uint256 stakeTimeSec = now.sub(lastStake.timestampSec);
223             //uint256 newStakingShareToBurn = 0;
224             if (lastStake.staking <= amountLeft) {
225                 // fully redeem a past stake
226                 // newStakingShareToBurn = lastStake.stakingShares;
227                 rewardAmount = computeNewReward(rewardAmount, lastStake.stakingShares, lastStake.timestampSec);
228                 amountLeft = amountLeft.sub(lastStake.staking);
229                 accountStakes.length--;
230             } else {
231                 // partially redeem a past stake
232                 uint256 oneRewardAmountAll = computeNewReward(0, lastStake.stakingShares, lastStake.timestampSec);
233                 uint256 oneRewardAmountReal = oneRewardAmountAll.mul(amountLeft).div(lastStake.staking);
234                 rewardAmount += oneRewardAmountReal;
235                 lastStake.stakingShares =  lastStake.stakingShares.mul(lastStake.staking - amountLeft).div(lastStake.staking);
236                 lastStake.staking = lastStake.staking.sub(amountLeft);
237
238                 amountLeft = 0;
239             }
240         }
```

Figure 2 The part of *_unstake* function source code

- Require Error

As shown in Figure 2 below, when the user or contract address calls the *_stakeFor*/*_unstake*/*contributeTokens* function, if the *transfer*/*transferFrom* function in the token contract has no return value, the require judgment will always fail, resulting in unsuccessful function call.

```
153     function _stakeFor(address staker, address beneficiary, uint256 amount) private {
154         require(now < expiration, 'Tokenliquidityity: staking is expired');
155         require(amount > 0, 'Tokenliquidityity: stake amount is zero');
156         require(beneficiary != address(0), 'Tokenliquidityity: beneficiary is zero address');
157
158         uint256 mintedStakingShares = amount.mul(_initialSharesPerToken).div(10000);
159         require(mintedStakingShares > 0, 'Tokenliquidityity: Stake amount is too small');
160
161         //updateAccounting();
162
163         // 1. User Accounting
164         UserTotals storage totals = _userTotals[beneficiary];
165         totals.staking = totals.staking.add(amount);
166         totals.lastAccountingTimestampSec = now;
167
168         Stake memory newStake = Stake(mintedStakingShares, amount, now);
169         _userStakes[beneficiary].push(newStake);
170
171         // interactions
172         require(_stakingPool.token().transferFrom(staker, address(_stakingPool), amount),
173             'Tokenliquidityity: transfer into staking pool failed');
174
175         emit Staked(beneficiary, amount, totalStakedFor(beneficiary), "");
176     }
```

Figure 3 *_stakeFor function source code*

Security advice: The transfer/transferFrom functions in the token contract must have a return value.

- Compiler Warning

  The TokenLiquidity contract specified that the minimum version of the compiler is 0.5.0, and compiling this contract with the 0.5.0 version of the compiler will generate a compiler warning as shown in Figure 1 below.

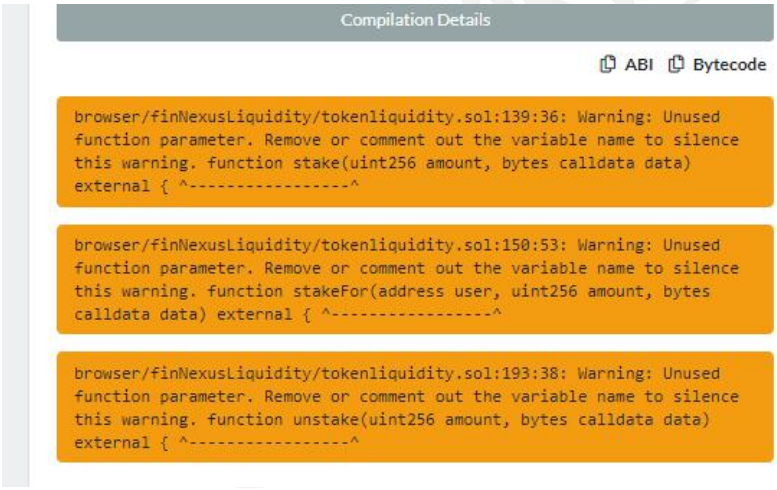


Figure 4 Compiler Warning

Modify Recommendation: It is recommended to modify the code to eliminate the compiler warning.

Audited Source Code with Comments:

```
// Beosin (Chengdu LianAn) // File: IStaking.sol

pragma solidity ^0.5.0;


/**
 * @title Staking interface, as defined by EIP-900.
 * @dev https://github.com/ethereum/EIPs/blob/master/EIPS/eip-900.md
 */
contract IStaking {
    event Staked(address indexed user, uint256 amount, uint256 total, bytes data);
    event Unstaked(address indexed user, uint256 amount, uint256 total, bytes data);

    function stake(uint256 amount, bytes calldata data) external;
    function stakeFor(address user, uint256 amount, bytes calldata data) external;
    function unstake(uint256 amount, bytes calldata data) external;
    function totalStakedFor(address addr) public view returns (uint256);
    function totalStaked() public view returns (uint256);
    function token() external view returns (address);

    /**
     * @return False. This application does not support staking history.
     */
    function supportsHistory() external pure returns (bool) {
        return false;
    }
}
```

```
// Beosin (Chengdu LianAn) // File: TokenPool.sol

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


import "openzeppelin-solidity/contracts/ownership/Ownable.sol";
import "openzeppelin-solidity/contracts/token/ERC20/IERC20.sol";


/**
 * @title A simple holder of tokens.
 * This is a simple contract to hold tokens. It's useful in the case where a separate contract
 * needs to hold multiple distinct pools of the same token.
 */
contract TokenPool is Ownable {
    IERC20 public token; // Beosin (Chengdu LianAn) // Declare the external token contract instance.
    // Beosin (Chengdu LianAn) // Constructor, initialize the external token contract instance address.
    constructor(IERC20   token) public {
        token =   token;
    }
    // Beosin (Chengdu LianAn) // The function 'balance' is defined to get the token balance of this contract address.
    function balance() public view returns (uint256) {
        return token.balanceOf(address(this));
    }
    // Beosin (Chengdu LianAn) // The function 'transfer' is defined to transfer specified amount of tokens under this contract address to a specified address 'to'.
    function transfer(address to, uint256 value) external onlyOwner returns (bool) {
        return token.transfer(to, value);
    }
}


// Beosin (Chengdu LianAn) // File: TokenLiquidity.sol

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


import "openzeppelin-solidity/contracts/math/SafeMath.sol";
import "openzeppelin-solidity/contracts/token/ERC20/IERC20.sol";
import "openzeppelin-solidity/contracts/ownership/Ownable.sol";

import "./IStaking.sol";
import "./TokenPool.sol";


/**
 * @title Token liquidityity
```

```
 * @dev A smart-contract based mechanism to distribute tokens over time, inspired loosely by
 *      Compound and Uniswap.
 *
 *      Distribution tokens are added to a locked pool in the contract and become unlocked over time
 *      according to a once-configurable unlock schedule. Once unlocked, they are available to be
 *      claimed by users.
 *
 *      A user may deposit tokens to accrue ownership share over the unlocked pool. This owner share
 *      is a function of the number of tokens deposited as well as the length of time deposited.
 *      Specifically, a user's share of the currently-unlocked pool equals their "deposit-seconds"
 *      divided by the global "deposit-seconds". This aligns the new token distribution with long
 *      term supporters of the project, addressing one of the major drawbacks of simple airdrops.
 *
 *      More background and motivation available at:
 *      https://github.com/ampleforth/RFCs/blob/master/RFCs/rfc-1.md
 */
contract TokenLiquidity is IStaking, Ownable {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.
    // Beosin (Chengdu LianAn) // Declare the stake relevant events.
    event Staked(address indexed user, uint256 amount, uint256 total, bytes data);
    event Unstaked(address indexed user, uint256 amount, uint256 total, bytes data);
    event TokensClaimed(address indexed user, uint256 amount);
    event Contributed(uint256 indexed amount, uint256 indexed total);

    TokenPool private _stakingPool; // Beosin (Chengdu LianAn) // Declare the external TokenPool contract instance '_stakingPool'.
    TokenPool private _distributionPool; // Beosin (Chengdu LianAn) // Declare the external TokenPool contract instance '_distributionPool'.

    //
    // Time-bonus params
    //
    uint256 public constant BONUS_DECIMALS = 2;
    uint256 public bonusPeriodSec = 0;

    //
    // Global accounting state
    //
    uint256 public totalContribution = 0;
    uint256 public totalClaimed = 0;

    uint256 public totalLockedShares = 0;
    uint256 private _initialSharesPerToken = 0;

    uint256 public expiration;

    //
```

```
    // User accounting state
    //
    // Represents a single stake for a user. A user may have multiple.
    struct Stake {
        uint256 stakingShares;    // the max token B
        uint256 staking; // the token A
        uint256 timestampSec;
    }

    // Caches aggregated values from the User->Stake[] map to save computation.
    // If lastAccountingTimestampSec is 0, there's no entry for that user.
    struct UserTotals {
        uint256 staking; // the token A
        uint256 stakingShareSeconds;
        uint256 lastAccountingTimestampSec;
    }

    // Aggregated staking values per user
    mapping(address => UserTotals) private   userTotals;

    // The collection of stakes for each user. Ordered by timestamp, earliest to latest.
    mapping(address => Stake[]) private _userStakes;


    /**
     * @param stakingToken The token users deposit as stake.
     * @param distributionToken The token users receive as they unstake.
     * @param bonusPeriodSec_ Length of time for bonus to increase linearly to max.
     * @param initialSharesPerToken Number of shares to mint per staking token on first stake.
     * @param _expiration expiration time of this miner poor.
     */
    constructor(IERC20 stakingToken, IERC20 distributionToken,
                uint256 bonusPeriodSec_, uint256 initialSharesPerToken,uint256 _expiration) public {
        // and bonusPeriod to a small value like 1sec.
        require(bonusPeriodSec_ != 0, 'Tokenliquidityity: bonus period is zero');
        require(initialSharesPerToken > 0, 'Tokenliquidityity: initialSharesPerToken is zero');

        _stakingPool = new TokenPool(stakingToken); // Beosin (Chengdu LianAn) // Deploy new 'TokenPool' contract and initialize its token contract address with corresponding 'stakingToken'.
        _distributionPool = new TokenPool(distributionToken); // Beosin (Chengdu LianAn) // Deploy new 'TokenPool' contract and initialize its token contract address with corresponding 'distributionToken'.
        bonusPeriodSec = bonusPeriodSec_; // Beosin (Chengdu LianAn) // Initialize the bonus period times.
        _initialSharesPerToken = initialSharesPerToken; // Beosin (Chengdu LianAn) // Initialize the initial shares per token.
        expiration = _expiration; // Beosin (Chengdu LianAn) // Initialize the expiration time.
    }

    /**
```

```
     * @return The token users deposit as stake.
     */
    function getStakingToken() public view returns (IERC20) {
        return  stakingPool.token();
    }
    // Beosin (Chengdu LianAn) // The function 'setInitialSharesPerToken' is defined to modify the initial shares per token.
    function setInitialSharesPerToken(uint init) public onlyOwner {
         initialSharesPerToken = init;
    }
    function getInitialSharesPerToken() public view returns(uint) {
        return  initialSharesPerToken;
    }
    // Beosin (Chengdu LianAn) // The function 'setExpiration' is defined to modify the expiration time.
    function setExpiration(uint256  expiration)public onlyOwner{
        expiration =  expiration;
    }
    // Beosin (Chengdu LianAn) // The function 'setBonusPeriod' is defined to modify the bonus period.
    function setBonusPeriod(uint period) public onlyOwner{
        bonusPeriodSec = period;
    }

    /**
     * @return The token users receive as they unstake.
     */
    function getDistributionToken() public view returns (IERC20) {
        return _distributionPool.token();
    }

    /**
     * @dev Transfers amount of deposit tokens from the user.
     * @param amount Number of deposit tokens to stake.
     * @param data Not used.
     */
    function stake(uint256 amount, bytes calldata data) external {
        _stakeFor(msg.sender, msg.sender, amount); // Beosin (Chengdu LianAn) // Call the private function '_stakeFor' to deposit for the caller (beneficiary).
    }

    /**
     * @dev Transfers amount of deposit tokens from the caller on behalf of user.
     * @param user User address who gains credit for this stake operation.
     * @param amount Number of deposit tokens to stake.
     * @param data Not used.
     */
    function stakeFor(address user, uint256 amount, bytes calldata data) external {
        _stakeFor(msg.sender, user, amount); // Beosin (Chengdu LianAn) // Call the private function '_stakeFor' to deposit for the user (beneficiary).
```

```
    }

    /**
     * @dev Private implementation of staking methods.
     * @param staker User address who deposits tokens to stake.
     * @param beneficiary User address who gains credit for this stake operation.
     * @param amount Number of deposit tokens to stake.
     */
    function   stakeFor(address staker, address beneficiary, uint256 amount) private {
        require(now < expiration, 'Tokenliquidityity: staking is expired'); // Beosin (Chengdu LianAn) //
Require that the current time cannot exceed the expiration time.
        require(amount > 0, 'Tokenliquidityity: stake amount is zero'); // Beosin (Chengdu LianAn) // Require
that the deposit amount should be greater than 0.
        require(beneficiary != address(0), 'Tokenliquidityity: beneficiary is zero address'); // Beosin (Chengdu
LianAn) // Require that the beneficiary address should not be zero address.

        uint256 mintedStakingShares = amount.mul(  initialSharesPerToken).div(10000); // Beosin (Chengdu
LianAn) // Declare the local variable 'mintedStakingShares' for calculate the gettable share amount of this
deposit.
        require(mintedStakingShares > 0, 'Tokenliquidityity: Stake amount is too small'); // Beosin (Chengdu
LianAn) // Require that the 'mintedStakingShares' should be greater than 0.

        //updateAccounting();

        // 1. User Accounting
        UserTotals storage totals = _userTotals[beneficiary]; // Beosin (Chengdu LianAn) // Declare the
variable 'totals' for getting the stake data of specified address 'beneficiary'.
        totals.staking = totals.staking.add(amount); // Beosin (Chengdu LianAn) // Store/Update the deposited
amount of total stake token.
        totals.lastAccountingTimestampSec = now; // Beosin (Chengdu LianAn) // Update the stake
timestamp.

        Stake memory newStake = Stake(mintedStakingShares, amount, now); // Beosin (Chengdu LianAn) //
Declare the variable 'newStake' for constructing a memory structure instance of this new stake.
        _userStakes[beneficiary].push(newStake); // Beosin (Chengdu LianAn) // Store the stake data.

        // interactions
        require(_stakingPool.token().transferFrom(staker, address(_stakingPool), amount),
            'Tokenliquidityity: transfer into staking pool failed'); // Beosin (Chengdu LianAn) // Call the
'transferFrom' function of the specified ERC20 Token contract, the contract 'TokenLiquidity' as a delegate
of 'staker' transfers tokens to itself.

        emit Staked(beneficiary, amount, totalStakedFor(beneficiary), ""); // Beosin (Chengdu LianAn) //
Trigger the event 'Staked'.
    }

    /**
     * @dev Unstakes a certain amount of previously deposited tokens. User also receives their
```

```
     * alotted number of distribution tokens.
     * @param amount Number of deposit tokens to unstake / withdraw.
     * @param data Not used.
     */
    function unstake(uint256 amount, bytes calldata data) external {
        unstake(amount); // Beosin (Chengdu LianAn) // Call the private function '_unstake' to withdraw a certain amount of deposited tokens of caller.
    }
    // Beosin (Chengdu LianAn) // The function 'unstakeAll' is defined to withdraw all deposited tokens of caller.
    function unstakeAll() external {
        uint amount = totalStakedFor(msg.sender); // Beosin (Chengdu LianAn) // Declare the local variable 'amount' for recording the amount of caller's total deposited tokens.
        unstake(amount); // Beosin (Chengdu LianAn) // Call the private function '_unstake' to withdraw deposited tokens of caller.
    }
    /**
     * @param amount Number of deposit tokens to unstake / withdraw.
     * @return The total number of distribution tokens that would be rewarded.
     */
    function unstakeQuery(uint256 amount) public returns (uint256) {
        return _unstake(amount);
    }

    /**
     * @dev Unstakes a certain amount of previously deposited tokens. User also receives their
     * alotted number of distribution tokens.
     * @param amount Number of deposit tokens to unstake / withdraw.
     * @return The total number of distribution tokens rewarded.
     */
    function _unstake(uint256 amount) private returns (uint256) {
        //updateAccounting();

        // checks
        require(amount > 0, 'Tokenliquidityity: unstake amount is zero'); // Beosin (Chengdu LianAn) // Require that the withdraw amount should be greater than 0.
        require(totalStakedFor(msg.sender) >= amount,
            'Tokenliquidityity: unstake amount is greater than total user stakes'); // Beosin (Chengdu LianAn) // Require that the total stake amount is sufficient to withdraw.

        // 1. User Accounting
        UserTotals storage totals = _userTotals[msg.sender]; // Beosin (Chengdu LianAn) // Declare the variable 'totals' for getting the stake data of caller.
        Stake[] storage accountStakes = _userStakes[msg.sender]; // Beosin (Chengdu LianAn) // Declare the variable 'accountStakes' for getting the stake array data of caller.

        // Redeem from most recent stake and go backwards in time.
        uint256 amountLeft = amount;
```

```
        uint256 rewardAmount = 0; // Beosin (Chengdu LianAn) // Declare the local variable 'rewardAmount' to record the corresponding gettable reward of this part of deposited tokens.
        // Beosin (Chengdu LianAn) // Update (withdraw) the deposited tokens from the most recent stake data.
        while (amountLeft > 0) {
            Stake storage lastStake = accountStakes[accountStakes.length - 1]; // Beosin (Chengdu LianAn) // Declare the variable 'lastStake' for getting the last stake data of caller.
//                  uint256 stakeTimeSec = now.sub(lastStake.timestampSec);
                //uint256 newStakingShareToBurn = 0;
                if (lastStake.staking <= amountLeft) { // Beosin (Chengdu LianAn) // If the deposited token amount of this stake is insufficient(or the same as the withdraw amount) to withdraw.
                    // fully redeem a past stake
                    // newStakingShareToBurn = lastStake.stakingShares;
                    rewardAmount = computeNewReward(rewardAmount, lastStake.stakingShares, lastStake.timestampSec); // Beosin (Chengdu LianAn) // Calculate the gettable reward amount.
                    amountLeft = amountLeft.sub(lastStake.staking); // Beosin (Chengdu LianAn) // Update the 'amountLeft'.
                    accountStakes.length--; // Beosin (Chengdu LianAn) // Delete the last stake data.
                } else { // Beosin (Chengdu LianAn) // Otherwise, update the last stake data.
                    // partially redeem a past stake
                    uint256 oneRewardAmountAll = computeNewReward(0, lastStake.stakingShares, lastStake.timestampSec); // Beosin (Chengdu LianAn) // Calculate the gettable all reward of this stake.
                    uint256 oneRewardAmountReal = oneRewardAmountAll.mul(amountLeft).div(lastStake.staking); // Beosin (Chengdu LianAn) // Calculate the actual amount of the reward corresponding to the specified withdraw amount.
                    rewardAmount += oneRewardAmountReal; // Beosin (Chengdu LianAn) // Update the reward of this withdraw.
                    // Beosin (Chengdu LianAn) // Update the last stake data.
                    lastStake.stakingShares =    lastStake.stakingShares.mul(lastStake.staking - amountLeft).div(lastStake.staking);
                    lastStake.staking = lastStake.staking.sub(amountLeft);

                    amountLeft = 0; // Beosin (Chengdu LianAn) // Update the 'amountLeft' to 0 for ending the while loop.
                }
        }
        totals.staking = totals.staking.sub(amount);
        // interactions
        require(_stakingPool.transfer(msg.sender, amount),
            'Tokenliquidityity: transfer out of staking pool failed'); // Beosin (Chengdu LianAn) // Call the 'transfer' function of the specified ERC20 Token contract, the contract '_stakingPool' transfers tokens to this function caller.
        require(_distributionPool.transfer(msg.sender, rewardAmount),
            'Tokenliquidityity: transfer out of _distribution pool failed'); // Beosin (Chengdu LianAn) // Call the 'transfer' function of the specified ERC20 Token contract, the contract '_distributionPool' transfers tokens to this function caller as reward.
        // Beosin (Chengdu LianAn) // Trigger the relevant events.
        emit Unstaked(msg.sender, amount, totalStakedFor(msg.sender), "");
```

```
        emit TokensClaimed(msg.sender, rewardAmount);

        return rewardAmount;
    }
    // Beosin (Chengdu LianAn) // The function 'totalRewards' is defined to query the total rewards of specified address's all stake tokens.
    function totalRewards(address account) public view returns(uint256){
        Stake[] storage accountStakes =   userStakes[account]; // Beosin (Chengdu LianAn) // Declare the variable 'accountStakes' for getting the stake array data of 'account'.
        uint rewardAmount = 0;
        for (uint256 i = 0; i< accountStakes.length; i++) {
            Stake storage oneStake = accountStakes[i];
            //uint256 stakeTimeSec = now.sub(oneStake.timestampSec);
            rewardAmount = computeNewReward(rewardAmount, oneStake.stakingShares, oneStake.timestampSec);
        }
        return rewardAmount;
    }
    /**
     * @dev Applies an additional time-bonus to a distribution amount. This is necessary to
     *      encourage long-term deposits instead of constant unstake/restakes.
     *      ends at 100% over bonusPeriodSec, then stays at 100% thereafter.
     * @param currentRewardTokens The current number of distribution tokens already alotted for this
     *                            unstake op. Any bonuses are already applied.
     * @param stakeTimeSec Time for which the tokens were staked. Needed to calculate
     *                     the time-bonus.
     * @return Updated amount of distribution tokens to award, with any bonus included on the
     *         newly added tokens.
     */
    function computeNewReward(uint256 currentRewardTokens,
                              uint256 stakingShare,
                              uint256 stakeTimeSec) public view returns (uint256) { //TODO : private???

        stakeTimeSec = now <expiration ? now.sub(stakeTimeSec) : expiration.sub(stakeTimeSec); // Beosin (Chengdu LianAn) // Calculate the actual stake keeping time.
        //uint256 newRewardTokens = stakingShare;
        if (stakeTimeSec >= bonusPeriodSec) {
            return currentRewardTokens.add(stakingShare); // Beosin (Chengdu LianAn) // Return the actual rewards.
        }
        uint256 bonusedReward = stakingShare.mul(stakeTimeSec).div(bonusPeriodSec);
        return currentRewardTokens.add(bonusedReward); // Beosin (Chengdu LianAn) // Return the actual rewards.
    }

    /**
     * @param addr The user to look up staking information for.
```

```
     * @return The number of staking tokens deposited for addr.
     */
    function totalStakedFor(address addr) public view returns (uint256) {
        return   userTotals[addr].staking;
    }

    /**
     * @return The total number of deposit tokens staked globally, by all users.
     */
    function totalStaked() public view returns (uint256) {
        return   stakingPool.balance();
    }

    /**
     * @dev Note that this application has a staking token as well as a distribution token, which
     * may be different. This function is required by EIP-900.
     * @return The deposit token used for staking.
     */
    function token() external view returns (address) {
        return address(getStakingToken());
    }


    /**
     * @return Total number of distribution tokens balance.
     */
    function distributionBalance() public view returns (uint256) {
        return _distributionPool.balance();
    }


    /**
     * @dev distribute token to distribution pool. Publicly callable.
     * @return Number of total distribution tokens.
     */
    function contributeTokens(uint256 amount) public returns (uint256) {
        // interactions
        require(_distributionPool.token().transferFrom(msg.sender, address(_distributionPool), amount), //
Beosin (Chengdu LianAn) // Call the 'transferFrom' function of the specified ERC20 Token contract, the
contract 'TokenLiquidity' as a delegate of caller transfers tokens to itself.
            'Tokenliquidityity: transfer into staking pool failed');

        totalContribution += amount;
        emit Contributed(amount, totalContribution); // Beosin (Chengdu LianAn) // Trigger the event
'Contributed'.
        return totalContribution;
    }
    // Beosin (Chengdu LianAn) // The function 'unlockToken' is defined to transfer out the specified
```

```
tokens of contract '_distributionPool'.
    function unlockToken() public onlyOwner {
        uint amount = _distributionPool.balance();
        require(  distributionPool.transfer(owner(), amount),
            'Tokenliquidityity: transfer   distribution pool failed'); // Beosin (Chengdu LianAn) // Call the
'transfer' function of the specified ERC20 Token contract, the contract '_distributionPool' transfers all
tokens of contract '_distributionPool' to the contract owner.
        return;
    }

}
```

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com