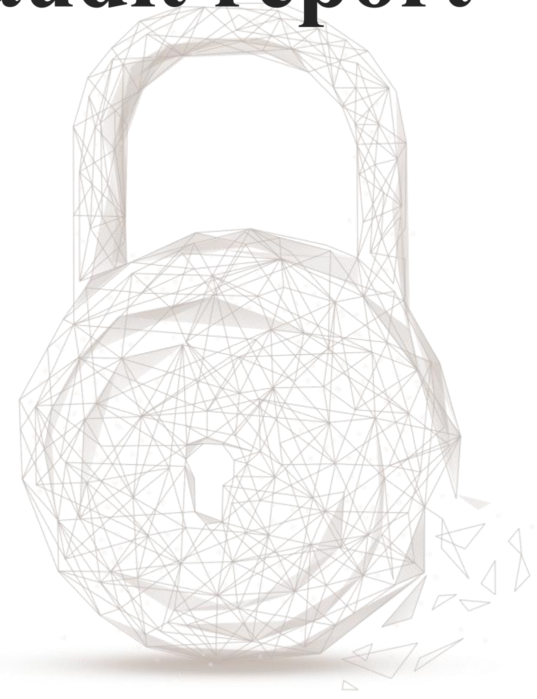
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number: 202108191700

Report Query Name: EKTAStakingPool

Smart Contract Address:

Fill in after deployment

Smart Contract Address Link:

Fill in after deployment

SHA256: 8324c183b0f254f2d06ef121b764f066aec2637ff7270f45bc72c66ad5e355a0

Start Date: 2021.08.13

Completion Date: 2021.08.19

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

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

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract EKTAStakingPool, including Coding Standards, Security, and Business Logic. **The EKTAStakingPool contract passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

Audit Contents:

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 BNB.
- 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 whether 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

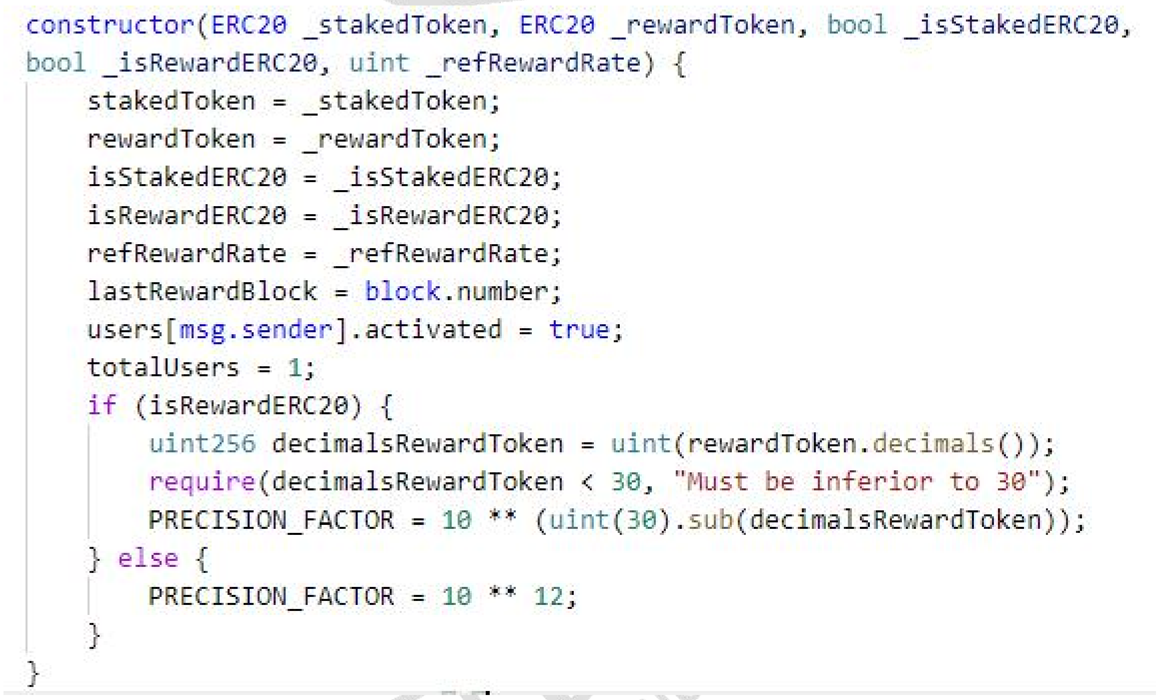
3. Business Security

Check whether the business is secure.

3.1 Business analysis of contract EKTAStakingPool

(1) constructor function

- Description: The *constructor* function initialises the collateral and reward token addresses, sets the referral reward percentage, updates the last reward block, updates the total number of users and sets other relevant data.



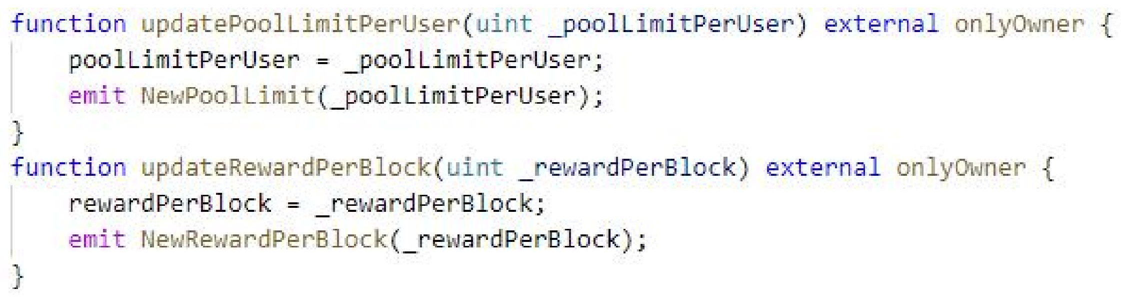
```
constructor(ERC20 _stakedToken, ERC20 _rewardToken, bool _isStakedERC20,
bool _isRewardERC20, uint _refRewardRate) {
    stakedToken = _stakedToken;
    rewardToken = _rewardToken;
    isStakedERC20 = _isStakedERC20;
    isRewardERC20 = _isRewardERC20;
    refRewardRate = _refRewardRate;
    lastRewardBlock = block.number;
    users[msg.sender].activated = true;
    totalUsers = 1;
    if (isRewardERC20) {
        uint256 decimalsRewardToken = uint(rewardToken.decimals());
        require(decimalsRewardToken < 30, "Must be inferior to 30");
        PRECISION_FACTOR = 10 ** (uint(30).sub(decimalsRewardToken));
    } else {
        PRECISION_FACTOR = 10 ** 12;
    }
}
```

Figure 1 source code of *constructor* function

- Related functions: *constructor*
- Result: Pass

(2) Related setting functions

- Description: The *updatePoolLimitPerUser* function is used to set the maximum stake limit; the *updateRewardPerBlock* function is used to set the per-block reward, and the above functions are called via the owner.



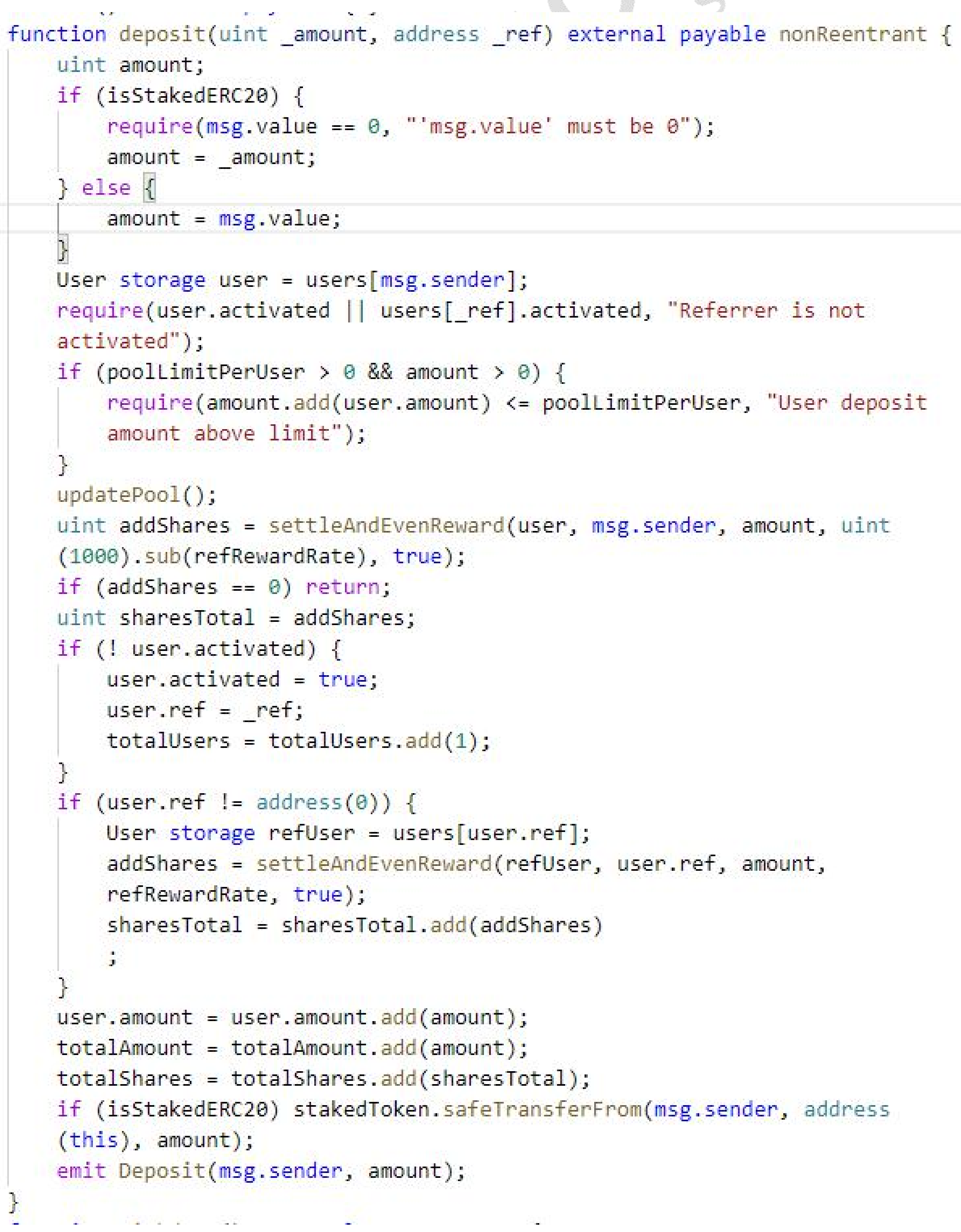
```
function updatePoolLimitPerUser(uint _poolLimitPerUser) external onlyOwner {
    poolLimitPerUser = _poolLimitPerUser;
    emit NewPoolLimit(_poolLimitPerUser);
}
function updateRewardPerBlock(uint _rewardPerBlock) external onlyOwner {
    rewardPerBlock = _rewardPerBlock;
    emit NewRewardPerBlock(_rewardPerBlock);
}
```

Figure 2 source code of *updatePoolLimitPerUser* and *updateRewardPerBlock* functions

- Related functions: *updatePoolLimitPerUser, updateRewardPerBlock*
- Result: Pass

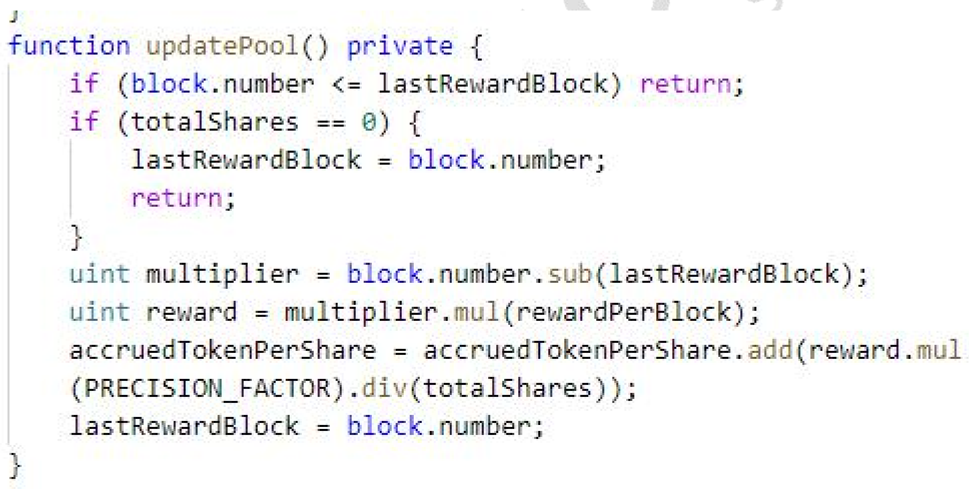
(3) deposit function

- Description: The *deposit* function is used to stake tokens. When calling this function, it is necessary to satisfy that the caller is the staked user or the referrer is the staked user; if the maximum stake limit is set, then each stake needs to determine the amount of stake; the last staked block time and accruedTokenPerShare are updated via the *updatePool* function; after updating the user via *settleAndEvenReward*; If it is the first time to stake, then update the total amount of users; if the referrer address is set, then update the referrer stake shares and referrer reward settlement, and finally update the totalShares.



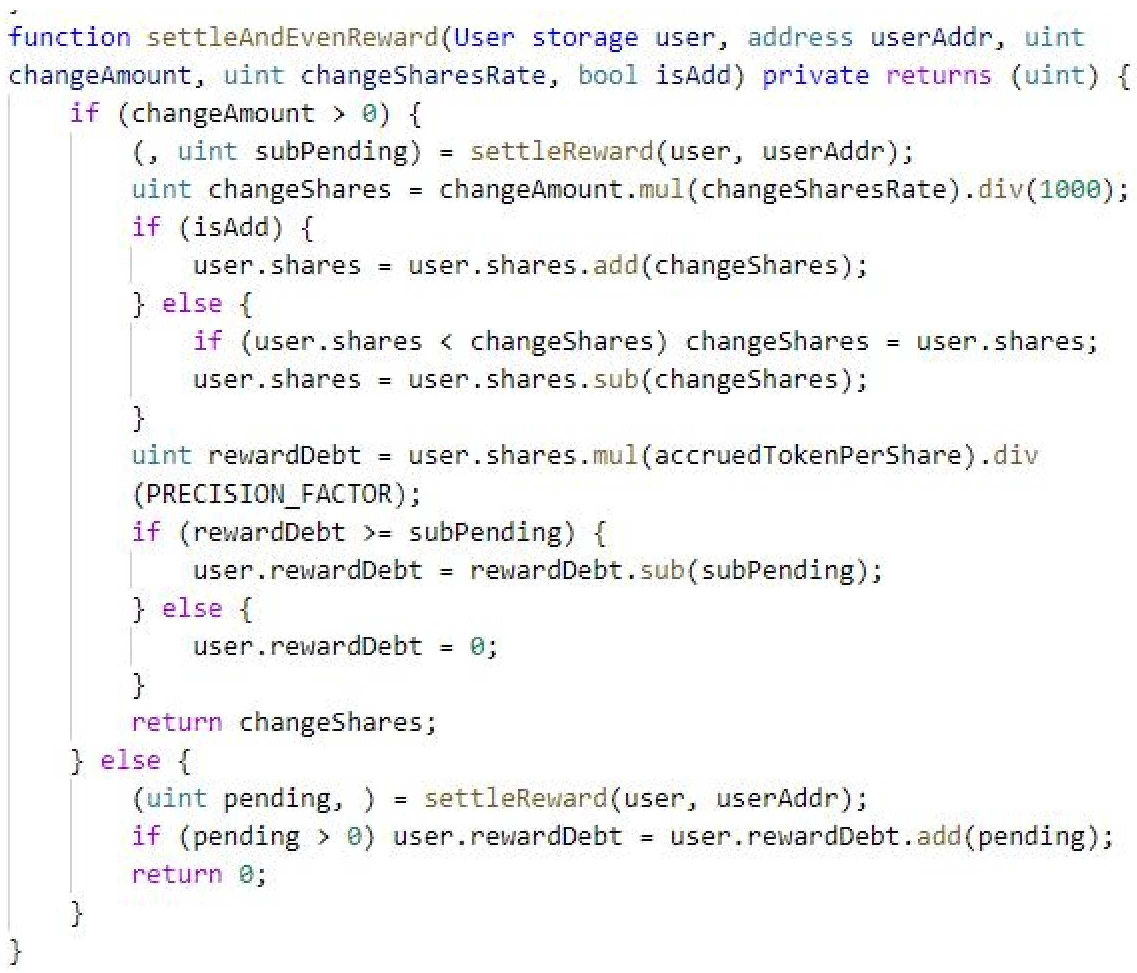
```
function deposit(uint _amount, address _ref) external payable nonReentrant {
    uint amount;
    if (isStakedERC20) {
        require(msg.value == 0, "'msg.value' must be 0");
        amount = _amount;
    } else {
        amount = msg.value;
    }
    User storage user = users[msg.sender];
    require(user.activated || users[_ref].activated, "Referrer is not
    activated");
    if (poolLimitPerUser > 0 && amount > 0) {
        require(amount.add(user.amount) <= poolLimitPerUser, "User deposit
        amount above limit");
    }
    updatePool();
    uint addShares = settleAndEvenReward(user, msg.sender, amount, uint
    (1000).sub(refRewardRate), true);
    if (addShares == 0) return;
    uint sharesTotal = addShares;
    if (! user.activated) {
        user.activated = true;
        user.ref = _ref;
        totalUsers = totalUsers.add(1);
    }
    if (user.ref != address(0)) {
        User storage refUser = users[user.ref];
        addShares = settleAndEvenReward(refUser, user.ref, amount,
        refRewardRate, true);
        sharesTotal = sharesTotal.add(addShares)
        ;
    }
    user.amount = user.amount.add(amount);
    totalAmount = totalAmount.add(amount);
    totalShares = totalShares.add(sharesTotal);
    if (isStakedERC20) stakedToken.safeTransferFrom(msg.sender, address
    (this), amount);
    emit Deposit(msg.sender, amount);
}
```

Figure 3 source code of *deposit* function



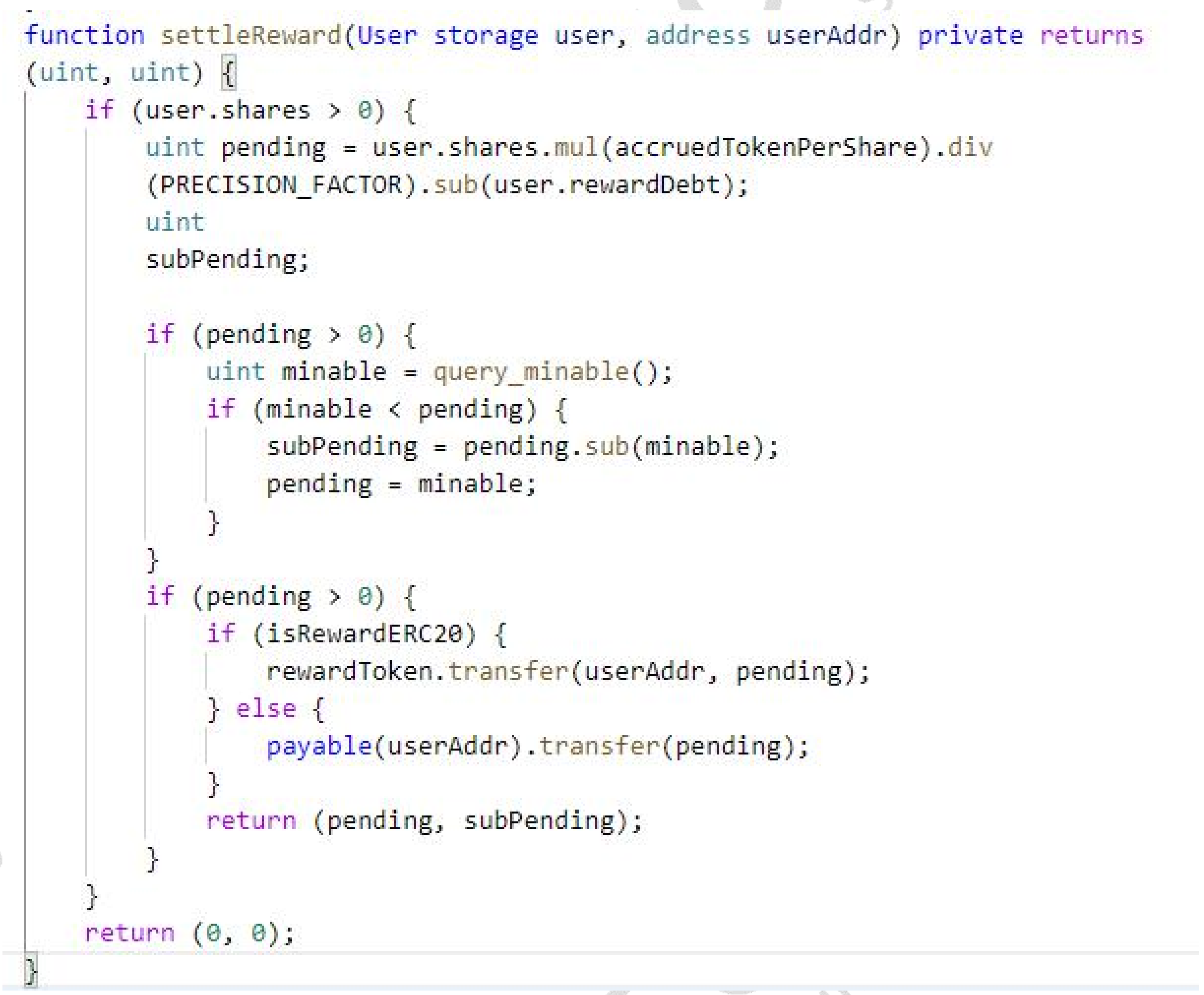
```
function updatePool() private {
    if (block.number <= lastRewardBlock) return;
    if (totalShares == 0) {
        lastRewardBlock = block.number;
        return;
    }
    uint multiplier = block.number.sub(lastRewardBlock);
    uint reward = multiplier.mul(rewardPerBlock);
    accruedTokenPerShare = accruedTokenPerShare.add(reward.mul
    (PRECISION_FACTOR).div(totalShares));
    lastRewardBlock = block.number;
}
```

Figure 4 source code of *updatePool* function



```
function settleAndEvenReward(User storage user, address userAddr, uint
changeAmount, uint changeSharesRate, bool isAdd) private returns (uint) {
    if (changeAmount > 0) {
        (, uint subPending) = settleReward(user, userAddr);
        uint changeShares = changeAmount.mul(changeSharesRate).div(1000);
        if (isAdd) {
            user.shares = user.shares.add(changeShares);
        } else {
            if (user.shares < changeShares) changeShares = user.shares;
            user.shares = user.shares.sub(changeShares);
        }
        uint rewardDebt = user.shares.mul(accruedTokenPerShare).div
        (PRECISION_FACTOR);
        if (rewardDebt >= subPending) {
            user.rewardDebt = rewardDebt.sub(subPending);
        } else {
            user.rewardDebt = 0;
        }
        return changeShares;
    } else {
        (uint pending, ) = settleReward(user, userAddr);
        if (pending > 0) user.rewardDebt = user.rewardDebt.add(pending);
        return 0;
    }
}
```

Figure 5 source code of *settleAndEvenReward* function



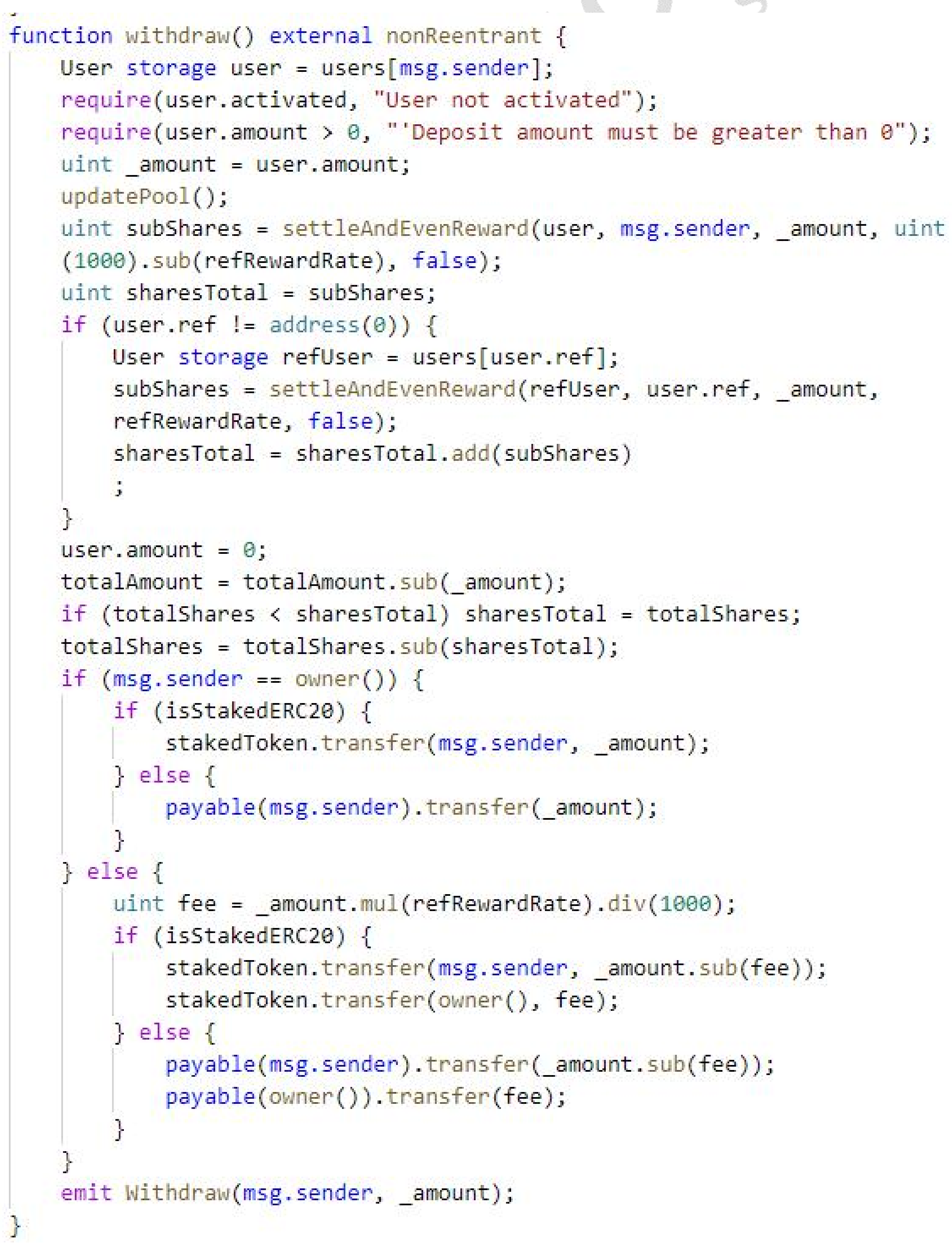
```
function settleReward(User storage user, address userAddr) private returns
(uint, uint) {
    if (user.shares > 0) {
        uint pending = user.shares.mul(accruedTokenPerShare).div
        (PRECISION_FACTOR).sub(user.rewardDebt);
        uint
        subPending;

        if (pending > 0) {
            uint minable = query_minable();
            if (minable < pending) {
                subPending = pending.sub(minable);
                pending = minable;
            }
        }
        if (pending > 0) {
            if (isRewardERC20) {
                rewardToken.transfer(userAddr, pending);
            } else {
                payable(userAddr).transfer(pending);
            }
            return (pending, subPending);
        }
    }
    return (0, 0);
}
```

Figure 6 source code of *settleAndEvenReward* function

- Related functions: *updatePool, transferFrom, deposit, transfer, owner, settleReward*
- Result: Pass

(4) withdraw function

- Description: This function is called by the user to withdraw all the stake tokens. When this function is called, the lastRewardBlock and accruedTokenPerShare are updated via the *updatePool* function; the corresponding reward is sent and the user's stake amount is updated via the *settleAndEvenReward* function; If the caller is the owner, no exit fees will be charged for stake; the user's stake amount is updated via the *settleReward* function to update the referrer's stake and send the reward, and finally update totalShares.



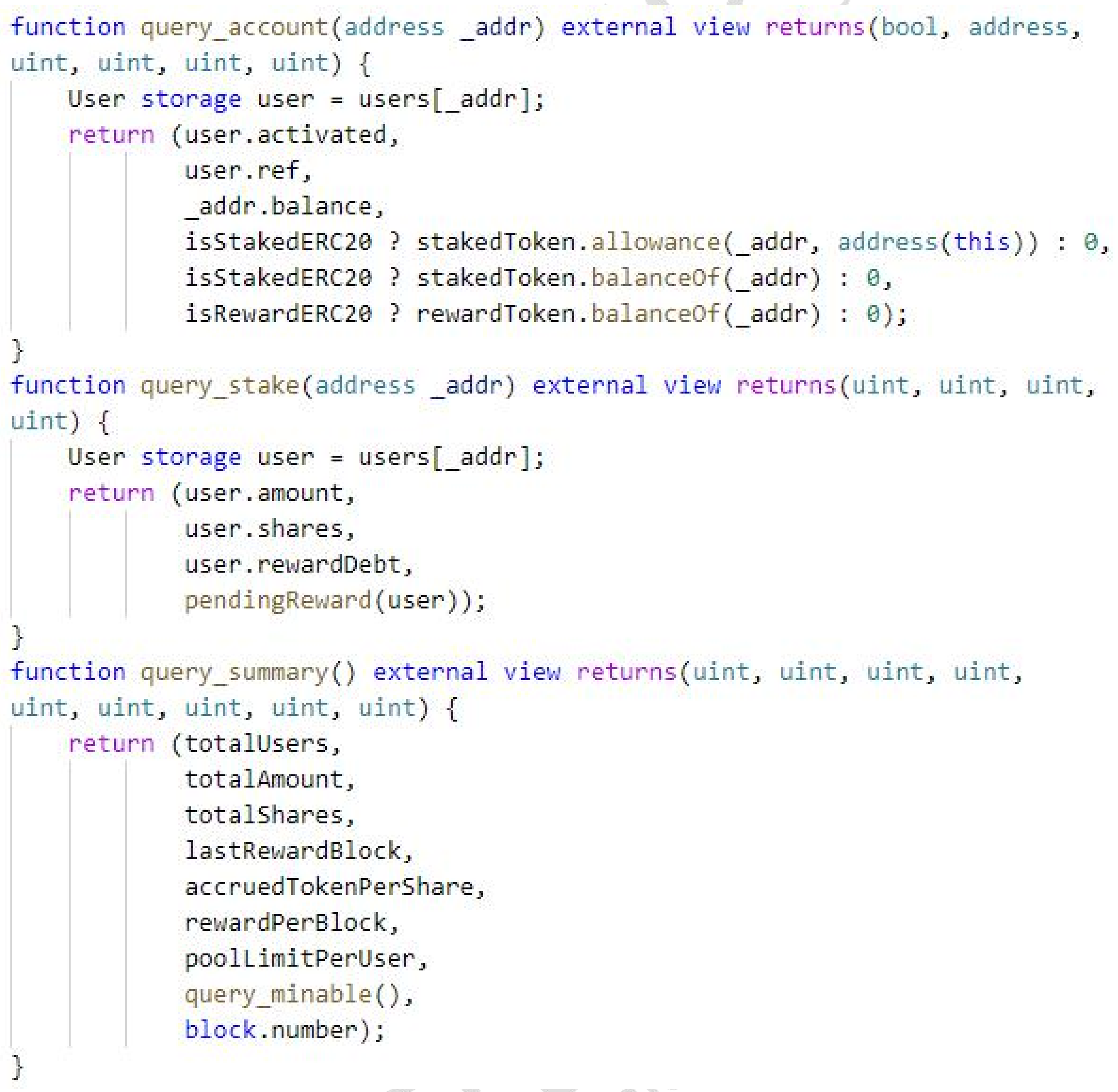
```
function withdraw() external nonReentrant {
    User storage user = users[msg.sender];
    require(user.activated, "User not activated");
    require(user.amount > 0, "'Deposit amount must be greater than 0");
    uint _amount = user.amount;
    updatePool();
    uint subShares = settleAndEvenReward(user, msg.sender, _amount, uint
    (1000).sub(refRewardRate), false);
    uint sharesTotal = subShares;
    if (user.ref != address(0)) {
        User storage refUser = users[user.ref];
        subShares = settleAndEvenReward(refUser, user.ref, _amount,
        refRewardRate, false);
        sharesTotal = sharesTotal.add(subShares)
        ;
    }
    user.amount = 0;
    totalAmount = totalAmount.sub(_amount);
    if (totalShares < sharesTotal) sharesTotal = totalShares;
    totalShares = totalShares.sub(sharesTotal);
    if (msg.sender == owner()) {
        if (isStakedERC20) {
            stakedToken.transfer(msg.sender, _amount);
        } else {
            payable(msg.sender).transfer(_amount);
        }
    } else {
        uint fee = _amount.mul(refRewardRate).div(1000);
        if (isStakedERC20) {
            stakedToken.transfer(msg.sender, _amount.sub(fee));
            stakedToken.transfer(owner(), fee);
        } else {
            payable(msg.sender).transfer(_amount.sub(fee));
            payable(owner()).transfer(fee);
        }
    }
    emit Withdraw(msg.sender, _amount);
}
```

Figure 7 source code of *withdraw* function

- Related functions: *withdraw, settleAndEvenReward, transfer, owner*
- Result: Pass

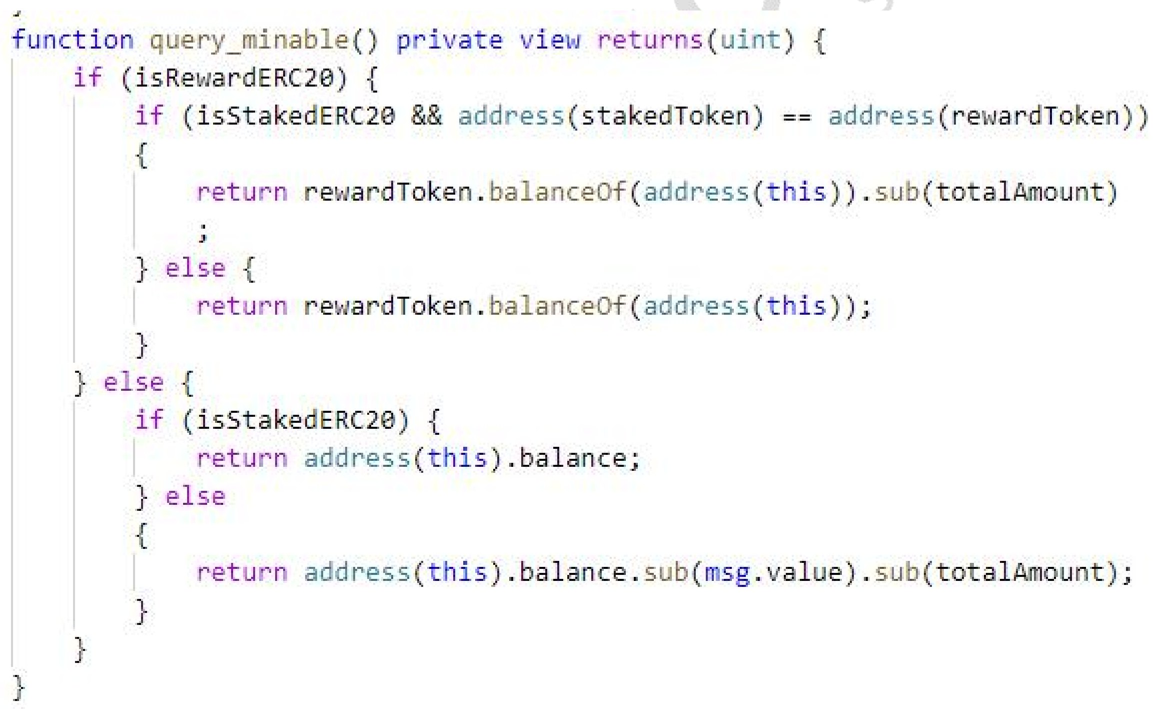
(5) Related query functions

- Description: The *query_account* function is used to query information about the user; *query_summary* queries information about the contract; *query_stake* function queries information about the user's stake. The *query_minable* function is used to query the total number of mineable.



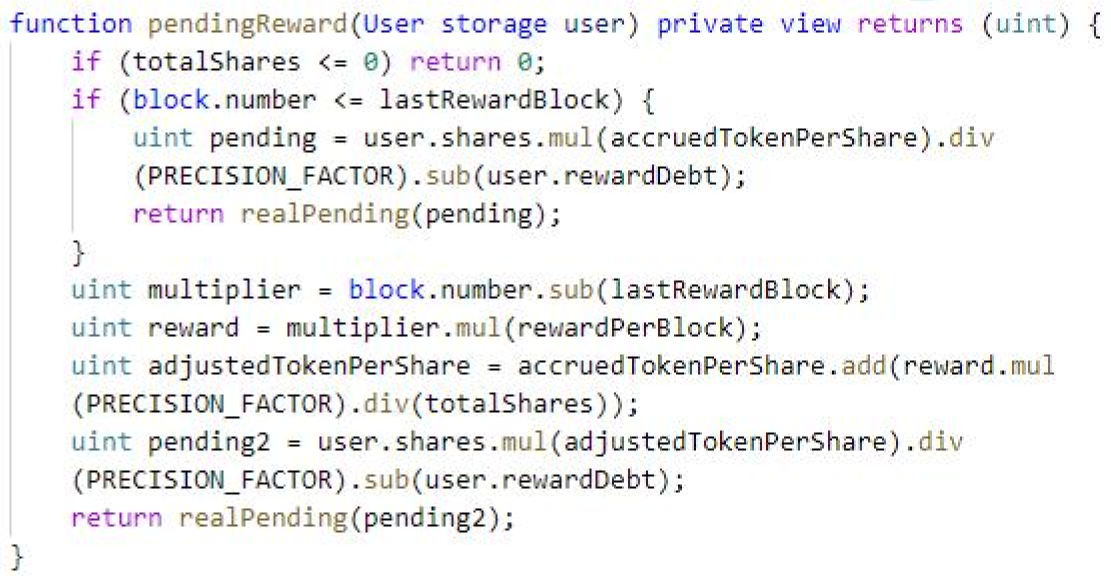
```
function query_account(address _addr) external view returns(bool, address,
uint, uint, uint, uint) {
    User storage user = users[_addr];
    return (user.activated,
            user.ref,
            _addr.balance,
            isStakedERC20 ? stakedToken.allowance(_addr, address(this)) : 0,
            isStakedERC20 ? stakedToken.balanceOf(_addr) : 0,
            isRewardERC20 ? rewardToken.balanceOf(_addr) : 0);
}
function query_stake(address _addr) external view returns(uint, uint, uint,
uint) {
    User storage user = users[_addr];
    return (user.amount,
            user.shares,
            user.rewardDebt,
            pendingReward(user));
}
function query_summary() external view returns(uint, uint, uint, uint,
uint, uint, uint, uint, uint) {
    return (totalUsers,
            totalAmount,
            totalShares,
            lastRewardBlock,
            accruedTokenPerShare,
            rewardPerBlock,
            poolLimitPerUser,
            query_minable(),
            block.number);
}
```

Figure 8 source code of related query functions



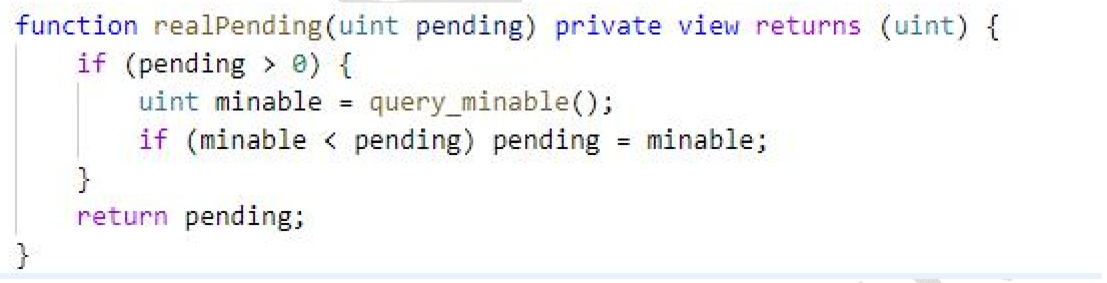
```
function query_minable() private view returns(uint) {
    if (isRewardERC20) {
        if (isStakedERC20 && address(stakedToken) == address(rewardToken))
        {
            return rewardToken.balanceOf(address(this)).sub(totalAmount)
            ;
        } else {
            return rewardToken.balanceOf(address(this));
        }
    } else {
        if (isStakedERC20) {
            return address(this).balance;
        } else
        {
            return address(this).balance.sub(msg.value).sub(totalAmount);
        }
    }
}
```

Figure 9 source code of *query_minable* function



```
function pendingReward(User storage user) private view returns (uint) {
    if (totalShares <= 0) return 0;
    if (block.number <= lastRewardBlock) {
        uint pending = user.shares.mul(accruedTokenPerShare).div
        (PRECISION_FACTOR).sub(user.rewardDebt);
        return realPending(pending);
    }
    uint multiplier = block.number.sub(lastRewardBlock);
    uint reward = multiplier.mul(rewardPerBlock);
    uint adjustedTokenPerShare = accruedTokenPerShare.add(reward.mul
    (PRECISION_FACTOR).div(totalShares));
    uint pending2 = user.shares.mul(adjustedTokenPerShare).div
    (PRECISION_FACTOR).sub(user.rewardDebt);
    return realPending(pending2);
}
```

Figure 10 source code of *pendingReward* function



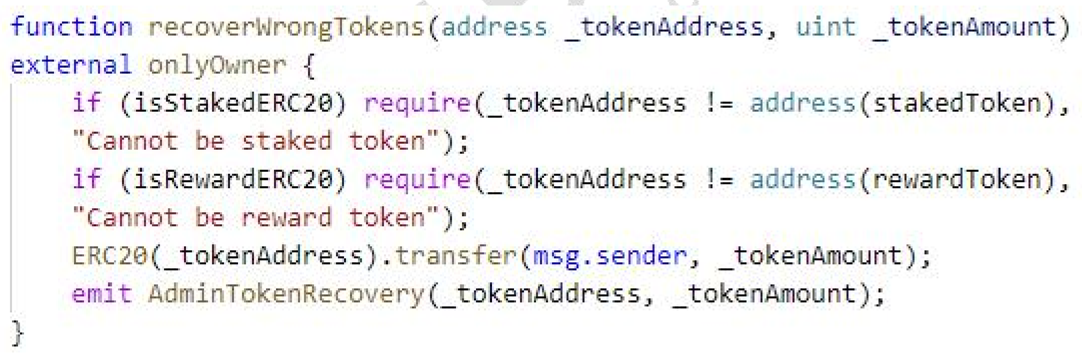
```
function realPending(uint pending) private view returns (uint) {
    if (pending > 0) {
        uint minable = query_minable();
        if (minable < pending) pending = minable;
    }
    return pending;
}
```

Figure 11 source code of *realPending* function

- Related functions: *query_account, query_stake, query_summary, pendingReward, realPending*
- Result: Pass

(6) recoverWrongTokens function

- Description: The *recoverWrongTokens* function is called by the owner to extract tokens other than collateral tokens and reward tokens.



```
function recoverWrongTokens(address _tokenAddress, uint _tokenAmount)
external onlyOwner {
    if (isStakedERC20) require(_tokenAddress != address(stakedToken),
    "Cannot be staked token");
    if (isRewardERC20) require(_tokenAddress != address(rewardToken),
    "Cannot be reward token");
    ERC20(_tokenAddress).transfer(msg.sender, _tokenAmount);
    emit AdminTokenRecovery(_tokenAddress, _tokenAmount);
}
```

Figure 12 source code of *recoverWrongTokens* function

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

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contract EKTAStakingPool. The problems found by the audit team during the audit process have been notified to the project party and reached an agreement on the repair results, the overall audit result of the EKTAStakingPool smart contract is **Pass**. It is important to note that stake tokens avoid the use of non-standard ERC20.

Code:

EKTAStakingPool.sol

```
pragma solidity ^0.8.0;
abstract contract Context {
        function _msgSender() internal view virtual returns (address) {
                return msg.sender;
        }
        function _msgData() internal view virtual returns (bytes calldata) {
                return msg.data;
        }
}
abstract contract Ownable is Context {
        address private _owner;
        event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
        constructor() {
                _setOwner(_msgSender());
        }
        function owner() public view virtual returns (address) {
                return _owner;
        }
        modifier onlyOwner() {
                require(owner() == _msgSender(), "Ownable: caller is not the owner");
                _;
        }
        function renounceOwnership() public virtual onlyOwner {
                _setOwner(address(0));
        }
        function transferOwnership(address newOwner) public virtual onlyOwner {
                require(newOwner != address(0), "Ownable: new owner is the zero address");
                _setOwner(newOwner);
        }
        function _setOwner(address newOwner) private {
                address oldOwner = _owner;
                _owner = newOwner;
                emit OwnershipTransferred(oldOwner, newOwner);
        }
}
interface IERC20 {
        function totalSupply() external view returns (uint256);
        function balanceOf(address account) external view returns (uint256);
        function transfer(address recipient, uint256 amount) external returns (bool);
        function allowance(address owner, address spender) external view returns (uint256);
        function approve(address spender, uint256 amount) external returns (bool);
        function transferFrom(
                address sender,
                address recipient,
                uint256 amount
        ) external returns (bool);
```

```
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
interface IERC20Metadata is IERC20 {
    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
    function decimals() external view returns (uint8);
}
interface ERC20 is IERC20Metadata {
}
library SafeMath {
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b > a) return (false, 0);
            return (true, a - b);
        }
    }
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (a == 0) return (true, 0);
            uint256 c = a * b;
            if (c / a != b) return (false, 0);
            return (true, c);
        }
    }
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a / b);
        }
    }
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b == 0) return (false, 0);
            return (true, a % b);
        }
    }
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        return a + b;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return a - b;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        return a * b;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return a / b;
    }
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return a % b;
    }
    function sub(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b <= a, errorMessage);
            return a - b;
        }
    }
    function div(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b > 0, errorMessage);
            return a / b;
        }
    }
    function mod(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
}
abstract contract ReentrancyGuard {
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;
    uint256 private _status;
    constructor() {
        _status = _NOT_ENTERED;
    }
    modifier nonReentrant() {
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");
        _status = _ENTERED;
        _;
        _status = _NOT_ENTERED;
    }
}
library Address {
    function isContract(address account) internal view returns (bool) {
        uint256 size;
        assembly {
            size := extcodesize(account)
        }
        return size > 0;
    }
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");
        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionCall(target, data, "Address: low-level call failed");
    }
    function functionCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, errorMessage);
    }
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value
    ) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");
        (bool success, bytes memory returndata) = target.call{value: value}(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }
    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");
        (bool success, bytes memory returndata) = target.staticcall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }
    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
    function verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) internal pure returns (bytes memory) {
        if (success) {
            return returndata;
        } else {
            if (returndata.length > 0) {
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}
library SafeERC20 {
    using Address for 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 {
        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) + value;
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender,
newAllowance));
    }
    function safeDecreaseAllowance(
        IERC20 token,
        address spender,
        uint256 value
    ) internal {
        unchecked {
            uint256 oldAllowance = token.allowance(address(this), spender);
            require(oldAllowance >= value, "SafeERC20: decreased allowance below zero");
            uint256 newAllowance = oldAllowance - value;
            _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender,
newAllowance));
        }
    }
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call failed");
        if (returndata.length > 0) {
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
contract YieldFarm is Ownable, ReentrancyGuard {
    using SafeMath for uint;
    using SafeERC20 for ERC20;
    uint PRECISION_FACTOR;
    uint public refRewardRate;
    ERC20 public stakedToken;
    ERC20 public rewardToken;
    bool isStakedERC20;
    bool isRewardERC20;
    uint rewardPerBlock = 1 * (10 ** 18);
    uint poolLimitPerUser;
    uint lastRewardBlock;
    uint accruedTokenPerShare;
    uint totalShares;
    uint totalAmount;
    uint totalUsers;
    struct User {
        bool activated;
        address ref;
        uint amount;
        uint shares;
        uint rewardDebt;
    }
    mapping(address => User) public users;
    event Deposit(address indexed user, uint amount);
    event Withdraw(address indexed user, uint amount);
    event AdminTokenRecovery(address tokenRecovered, uint amount);
    event NewPoolLimit(uint poolLimitPerUser);
    event NewRewardPerBlock(uint rewardPerBlock);
    constructor(ERC20 _stakedToken, ERC20 _rewardToken, bool _isStakedERC20, bool
_isRewardERC20, uint _refRewardRate) {
        stakedToken = _stakedToken;
        rewardToken = _rewardToken;
        isStakedERC20 = _isStakedERC20;
        isRewardERC20 = _isRewardERC20;
        refRewardRate = _refRewardRate;
        lastRewardBlock = block.number;
        users[msg.sender].activated = true;
        totalUsers = 1;
        if (isRewardERC20) {
            uint256 decimalsRewardToken = uint(rewardToken.decimals());
            require(decimalsRewardToken < 30, "Must be inferior to 30");
            PRECISION_FACTOR = 10 ** (uint(30).sub(decimalsRewardToken));
        } else {
            PRECISION_FACTOR = 10 ** 12;
        }
    }
    receive() external payable { }
    function deposit(uint _amount, address _ref) external payable nonReentrant {
        uint amount;
        if (isStakedERC20) {
            require(msg.value == 0, "'msg.value' must be 0");
            amount = _amount;
        } else {
            amount = msg.value;
        }
        User storage user = users[msg.sender];
        require(user.activated || users[_ref].activated, "Referrer is not activated");
        if (poolLimitPerUser > 0 && amount > 0) {
            require(amount.add(user.amount) <= poolLimitPerUser, "User deposit amount above limit");
        }
        updatePool();
        uint addShares = settleAndEvenReward(user, msg.sender, amount, uint(1000).sub(refRewardRate),
true);
        if (addShares == 0) return;
        uint sharesTotal = addShares;
        if (! user.activated) {
            user.activated = true;
            user.ref = _ref;
            totalUsers = totalUsers.add(1);
        }
        if (user.ref != address(0)) {
            User storage refUser = users[user.ref];
            addShares = settleAndEvenReward(refUser, user.ref, amount, refRewardRate, true);
            sharesTotal = sharesTotal.add(addShares);
        }
        user.amount = user.amount.add(amount);
        totalAmount = totalAmount.add(amount);
        totalShares = totalShares.add(sharesTotal);
        if (isStakedERC20) stakedToken.safeTransferFrom(msg.sender, address(this), amount);
        emit Deposit(msg.sender, amount);
    }
    function withdraw() external nonReentrant {
        User storage user = users[msg.sender];
        require(user.activated, "User not activated");
        require(user.amount > 0, "'Deposit amount must be greater than 0");
        uint _amount = user.amount;
        updatePool();
        uint subShares = settleAndEvenReward(user, msg.sender, _amount, uint(1000).sub(refRewardRate),
false);
        uint sharesTotal = subShares;
        if (user.ref != address(0)) {
            User storage refUser = users[user.ref];
            subShares = settleAndEvenReward(refUser, user.ref, _amount, refRewardRate, false);
            sharesTotal = sharesTotal.add(subShares);
        }
        user.amount = 0;
        totalAmount = totalAmount.sub(_amount);
        if (totalShares < sharesTotal) sharesTotal = totalShares;
        totalShares = totalShares.sub(sharesTotal);
        if (msg.sender == owner()) {
            if (isStakedERC20) {
                stakedToken.transfer(msg.sender, _amount);
            } else {
                payable(msg.sender).transfer(_amount);
            }
        } else {
            uint fee = _amount.mul(refRewardRate).div(1000);
            if (isStakedERC20) {
                stakedToken.transfer(msg.sender, _amount.sub(fee));
                stakedToken.transfer(owner(), fee);
            } else {
                payable(msg.sender).transfer(_amount.sub(fee));
                payable(owner()).transfer(fee);
            }
        }
        emit Withdraw(msg.sender, _amount);
    }
    function query_account(address _addr) external view returns(bool, address, uint, uint, uint, uint) {
        User storage user = users[_addr];
        return (user.activated,
                user.ref,
                _addr.balance,
                isStakedERC20 ? stakedToken.allowance(_addr, address(this)) : 0,
                isStakedERC20 ? stakedToken.balanceOf(_addr) : 0,
                isRewardERC20 ? rewardToken.balanceOf(_addr) : 0);
    }
    function query_stake(address _addr) external view returns(uint, uint, uint, uint) {
        User storage user = users[_addr];
        return (user.amount,
                user.shares,
                user.rewardDebt,
                pendingReward(user));
    }
    function query_summary() external view returns(uint, uint, uint, uint, uint, uint, uint, uint, uint) {
        return (totalUsers,
                totalAmount,
                totalShares,
                lastRewardBlock,
                    accruedTokenPerShare,
                    rewardPerBlock,
                    poolLimitPerUser,
                    query_minable(),
                    block.number);
    }
    function recoverWrongTokens(address _tokenAddress, uint _tokenAmount) external onlyOwner {
        if (isStakedERC20) require(_tokenAddress != address(stakedToken), "Cannot be staked token");
        if (isRewardERC20) require(_tokenAddress != address(rewardToken), "Cannot be reward token");
        ERC20(_tokenAddress).transfer(msg.sender, _tokenAmount);
        emit AdminTokenRecovery(_tokenAddress, _tokenAmount);
    }
    function updatePoolLimitPerUser(uint _poolLimitPerUser) external onlyOwner {
        poolLimitPerUser = _poolLimitPerUser;
        emit NewPoolLimit(_poolLimitPerUser);
    }
    function updateRewardPerBlock(uint _rewardPerBlock) external onlyOwner {
        rewardPerBlock = _rewardPerBlock;
        emit NewRewardPerBlock(_rewardPerBlock);
    }
    function settleReward(User storage user, address userAddr) private returns (uint, uint) {
        if (user.shares > 0) {
            uint pending =
user.shares.mul(accruedTokenPerShare).div(PRECISION_FACTOR).sub(user.rewardDebt);
            uint subPending;
            if (pending > 0) {
                uint minable = query_minable();
                if (minable < pending) {
                    subPending = pending.sub(minable);
                    pending = minable;
                }
            }
            if (pending > 0) {
                if (isRewardERC20) {
                    rewardToken.transfer(userAddr, pending);
                } else {
                    payable(userAddr).transfer(pending);
                }
                return (pending, subPending);
            }
        }
        return (0, 0);
    }
    function settleAndEvenReward(User storage user, address userAddr, uint changeAmount, uint
changeSharesRate, bool isAdd) private returns (uint) {
        if (changeAmount > 0) {
            (, uint subPending) = settleReward(user, userAddr);
            uint changeShares = changeAmount.mul(changeSharesRate).div(1000);
```

```
                if (isAdd) {
                    user.shares = user.shares.add(changeShares);
                } else {
                    if (user.shares < changeShares) changeShares = user.shares;
                    user.shares = user.shares.sub(changeShares);
                }
                uint rewardDebt = user.shares.mul(accruedTokenPerShare).div(PRECISION_FACTOR);
                if (rewardDebt >= subPending) {
                    user.rewardDebt = rewardDebt.sub(subPending);
                } else {
                    user.rewardDebt = 0;
                }
                return changeShares;
            } else {
                (uint pending, ) = settleReward(user, userAddr);
                if (pending > 0) user.rewardDebt = user.rewardDebt.add(pending);
                return 0;
            }
        }
        function pendingReward(User storage user) private view returns (uint) {
            if (totalShares <= 0) return 0;
            if (block.number <= lastRewardBlock) {
                uint pending =
user.shares.mul(accruedTokenPerShare).div(PRECISION_FACTOR).sub(user.rewardDebt);
                return realPending(pending);
            }
            uint multiplier = block.number.sub(lastRewardBlock);
            uint reward = multiplier.mul(rewardPerBlock);
            uint adjustedTokenPerShare =
accruedTokenPerShare.add(reward.mul(PRECISION_FACTOR).div(totalShares));
            uint pending2 =
user.shares.mul(adjustedTokenPerShare).div(PRECISION_FACTOR).sub(user.rewardDebt);
            return realPending(pending2);
        }
        function realPending(uint pending) private view returns (uint) {
            if (pending > 0) {
                uint minable = query_minable();
                if (minable < pending) pending = minable;
            }
            return pending;
        }
        function updatePool() private {
            if (block.number <= lastRewardBlock) return;
            if (totalShares == 0) {
                lastRewardBlock = block.number;
                return;
            }
            uint multiplier = block.number.sub(lastRewardBlock);
            uint reward = multiplier.mul(rewardPerBlock);
            accruedTokenPerShare =
accruedTokenPerShare.add(reward.mul(PRECISION_FACTOR).div(totalShares));
            lastRewardBlock = block.number;
      }
      function query_minable() private view returns(uint) {
            if (isRewardERC20) {
                  if (isStakedERC20 && address(stakedToken) == address(rewardToken)) {
                        return rewardToken.balanceOf(address(this)).sub(totalAmount);
                  } else {
                        return rewardToken.balanceOf(address(this));
                  }
            } else {
                  if (isStakedERC20) {
                        return address(this).balance;
                  } else {
                        return address(this).balance.sub(msg.value).sub(totalAmount);
                  }
            }
      }
}
```

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com