BEOSIN

Blockchain Security

Smart contract security audit report

BEOSIN
Blockchain Security

Audit Number：202101221038

Report Query Name: FC-FCTRXBasepool

Smart Contract Info：

Smart Contract Name	**Smart Contract Address**	**Smart Contract Address Link**
FCTRXBasePool	TNyePwQCWkjeZBnoDotWg5bKyL6WKyWq7i	https://tronscan.org/#/contract/TNyePwQCWkjeZBnoDotWg5bKyL6WKyWq7i/code
FCTRXBasePool	TN3HkS9xUm2ZNgLsGvd6qBestuGRLMFuJM	https://tronscan.org/#/contract/TN3HkS9xUm2ZNgLsGvd6qBestuGRLMFuJM/code
FCTRXBasePool	TSxtMPRzfk5hssmGCgB2y7NupjpyERqP39	https://tronscan.org/#/contract/TSxtMPRzfk5hssmGCgB2y7NupjpyERqP39/code
FCTRXBasePool	TAnzS2yuhAFsiUURV83o92Y17JvNNf1EXv	https://tronscan.org/#/contract/TAnzS2yuhAFsiUURV83o92Y17JvNNf1EXv/code
FCTRXBasePool	TAhzUy5QBFWH4B9saULAopFF7AhGDrT2we	https://tronscan.org/#/contract/TAhzUy5QBFWH4B9saULAopFF7AhGDrT2we/code

Start Date：2021.01.08

Completion Date：2021.01.22

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 project FC-FCTRXBasepool, including Coding Standards, Security, and Business Logic. **The FC-FCTRXBasepool project passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Immediate update is required to mitigate security risks. The smart contract is able to function properly only the possible operations mentioned above should be noticed.

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 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. In this project, the contract
- Result: Pass

2.10 Replay Attack

- Description: Check 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

3.1 Business analysis of Contract FCTRXBasePool

(1) Stake initialization function

- Description: As shown in the figure below. The "stake-reward" mode of the contract needs to initialize the relevant parameters (reward ratio *rewardRate*, first update time *lastUpdateTime*, phase completion time *periodFinish*), call the *notifyRewardAmount* function through the specified reward distribution administrator address *rewardDistribution*, and enter the initial reward used to calculate the reward ratio value reward, initialize the stake and reward related parameters. This function can be called by the designated address *rewardDistribution* at any time to control the reward ratio (the reward ratio can also be modified before the stake starts). If the value is too small, the user's income will not match the expectation.

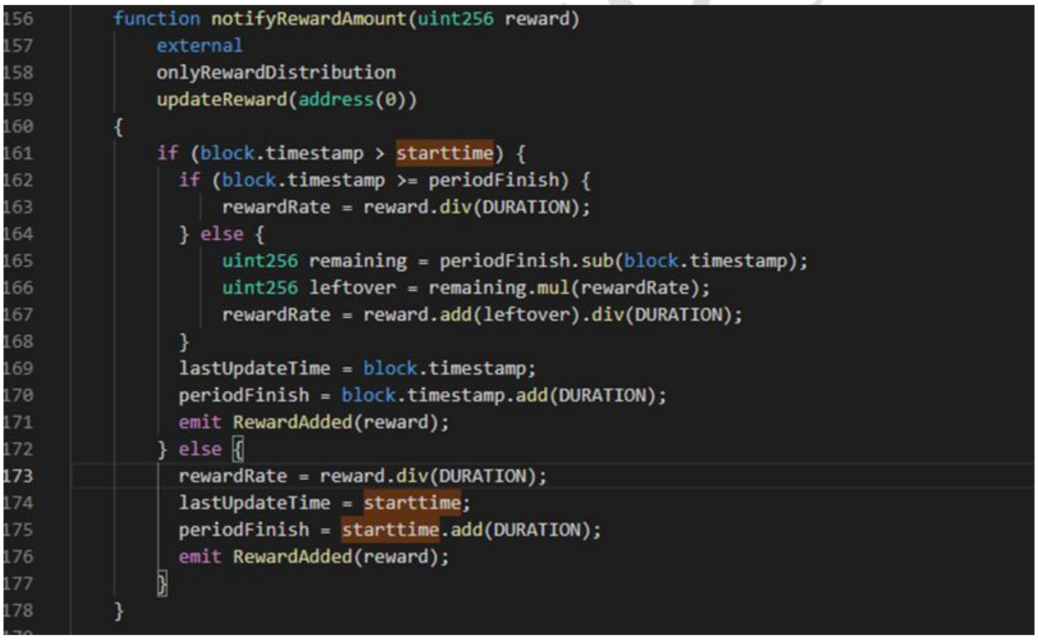

```
function notifyRewardAmount(uint256 reward)
    external
    onlyRewardDistribution
    updateReward(address(0))
{
    if (block.timestamp > starttime) {
      if (block.timestamp >= periodFinish) {
          rewardRate = reward.div(DURATION);
      } else {
          uint256 remaining = periodFinish.sub(block.timestamp);
          uint256 leftover = remaining.mul(rewardRate);
          rewardRate = reward.add(leftover).div(DURATION);
      }
      lastUpdateTime = block.timestamp;
      periodFinish = block.timestamp.add(DURATION);
      emit RewardAdded(reward);
    } else {
      rewardRate = reward.div(DURATION);
      lastUpdateTime = starttime;
      periodFinish = starttime.add(DURATION);
      emit RewardAdded(reward);
    }
}
```

Figure 1 source code of *notifyRewardAmount*

- Related functions: *notifyRewardAmount*
- Result: Pass

(2) Withdrawal of staked tokens

- Description: As shown in the figure below, the contract implements the withdraw function to withdraw the staked tokens. By calling the safetransfer function in the TRC20 contract, the contract address transfers the specified amount of TRC20 tokens to the function caller (user) address; this function restricts the user to call after the stake-reward mode is turned on (when the specified time is reached); each time the function is called to stake tokens, the reward-related data is updated through the modifier updateReward; and the modifier checkStart is used for each call to check whether the phase completion time is reached.

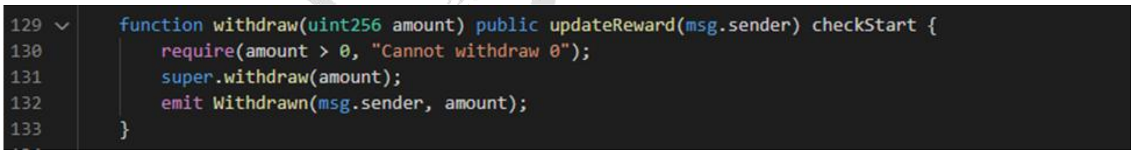

```
function withdraw(uint256 amount) public updateReward(msg.sender) checkStart {
    require(amount > 0, "Cannot withdraw 0");
    super.withdraw(amount);
    emit Withdrawn(msg.sender, amount);
}
```

Figure 2 source code of *withdraw*(FCTRXBasePool contract)

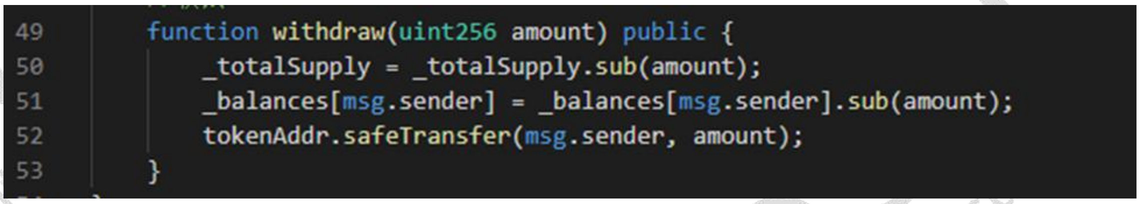
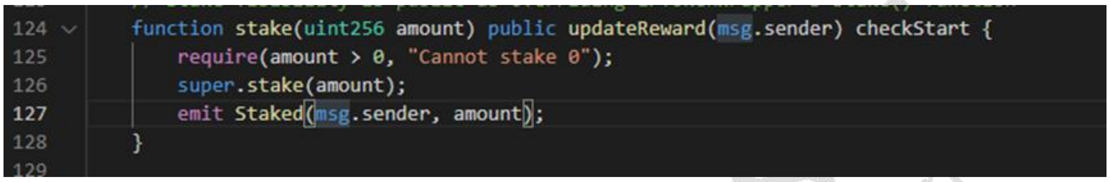

```
function withdraw(uint256 amount) public {
    _totalSupply = _totalSupply.sub(amount);
    _balances[msg.sender] = _balances[msg.sender].sub(amount);
    tokenAddr.safeTransfer(msg.sender, amount);
}
```

Figure 3 source code of *withdraw*(LPtokenWrapper contract)

- Related functions: *withdraw*
- Result: Pass

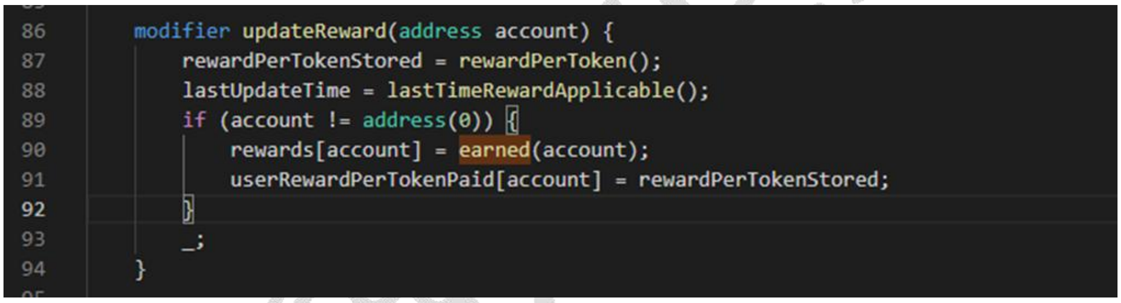
(3) Stake function

- Description: As shown in the figure below, the contract implements the stake function to stakeTRC20 tokens. The user authorizes the contract address in advance. By calling the safeTransferFrom function in the TRC20 contract, the contract address transfers the specified amount of TRC20 tokens to the contract address on behalf of the user; this function limits the user to only It can be called after the "stake-reward" mode is turned on (when the specified time is reached); each time the function is called to deposit tokens, the reward-related data is updated through the modifier updateReward; and the modifier checkStart is used for each call to check whether the phase completion time is reached .



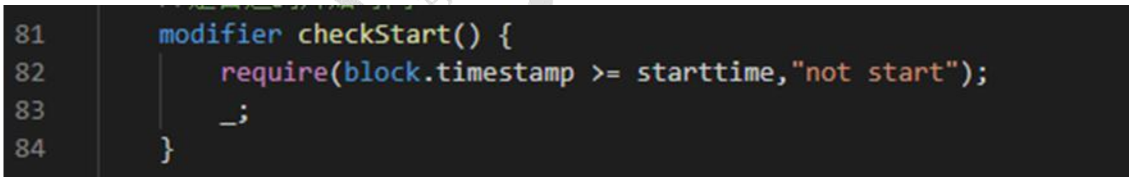
```
    function stake(uint256 amount) public updateReward(msg.sender) checkStart {
        require(amount > 0, "Cannot stake 0");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }
```

Figure 4 source code of *stake* (FCTRXBasePool contract)



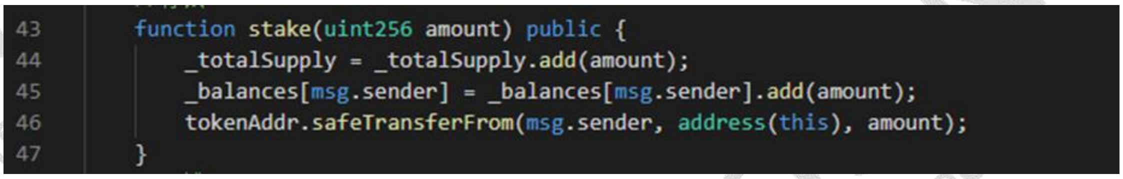
```
    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }
```

Figure 5 source code of *updateReward*



```
    modifier checkStart() {
        require(block.timestamp >= starttime,"not start");
        _;
    }
```

Figure 6 source code of *checkStart*



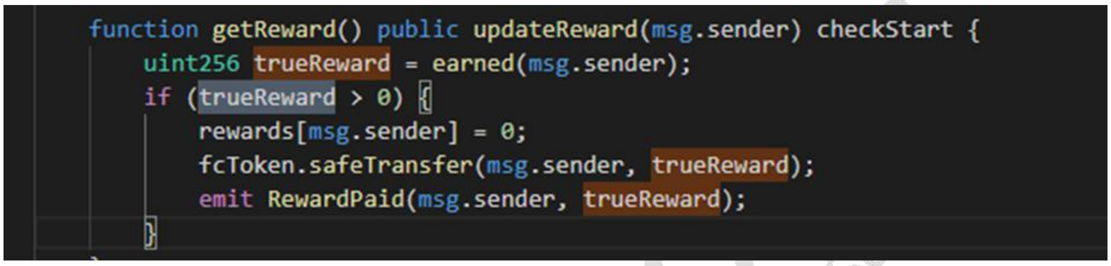
```
    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        tokenAddr.safeTransferFrom(msg.sender, address(this), amount);
    }
```

Figure 7 source code of *stake*(LPtokenWrapper contract)

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

(4) Get reward function

- Description: As shown in the figure below, The contract implements the getReward function to receive stake rewards (FC tokens). By calling the safeTransfer function in the FC contract, the contract address transfers the specified number of FC tokens (the user's all stake rewards) to the function caller (user) address ; This function restricts the user to call only after the "stake-reward" mode is turned on (the specified time is reached); each time this function is called to stake tokens, the reward related data is updated through the modifier updateReward; and each call is through the modifier checkStart Check whether the phase completion time is reached.



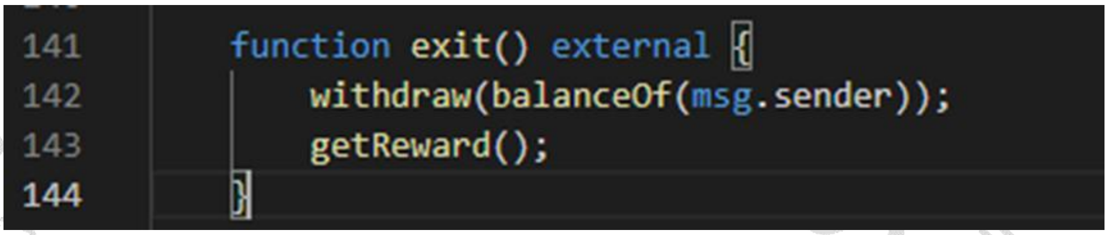
```
    function getReward() public updateReward(msg.sender) checkStart {
        uint256 trueReward = earned(msg.sender);
        if (trueReward > 0) {
            rewards[msg.sender] = 0;
            fcToken.safeTransfer(msg.sender, trueReward);
            emit RewardPaid(msg.sender, trueReward);
        }
```

Figure 8 source code of *getReward*

- Related functions: *getReward, earned*
- Result: Pass

(5) Exit function

- Description: As shown in the figure below, the contract implements the exit function for the caller to withdraw from the stake, call the withdraw function to extract all staked TRC20 tokens, call the getReward function to receive the caller's stake reward (FC token), and end the participation in the "stake-reward" mode. At this time, the user address cannot obtain new stake rewards because the amount of staked TRC20 tokens is empty.



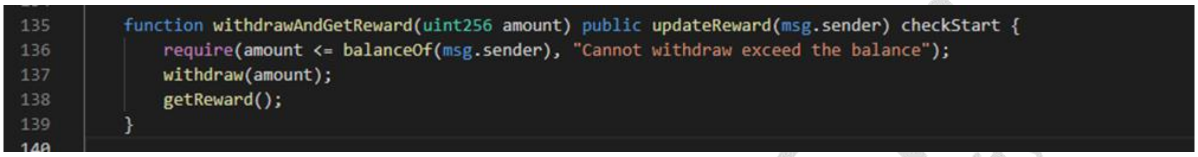
```
    function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
    }
```

Figure 9 source code of *exit*

- Related functions: *exit' withdraw, getReward*
- Result: Pass

(6) Withdraw staked and get staking reward function

- Description: As shown in the figure below, the contract implements the *withdrawAndGetReward* function for the caller to receive the reward while withdrawing the stake, call the *withdraw* function to extract the staked TRC20 tokens, and call the *getReward* function to receive the caller's stake reward. This function restricts users to only calling after the "stake-reward" mode is turned on (the specified time is reached); each time this function is called to deposit tokens, the reward related data is updated through the modifier *updateReward*; and each call is checked by the modifier *checkStart* Whether the phase completion time is reached.



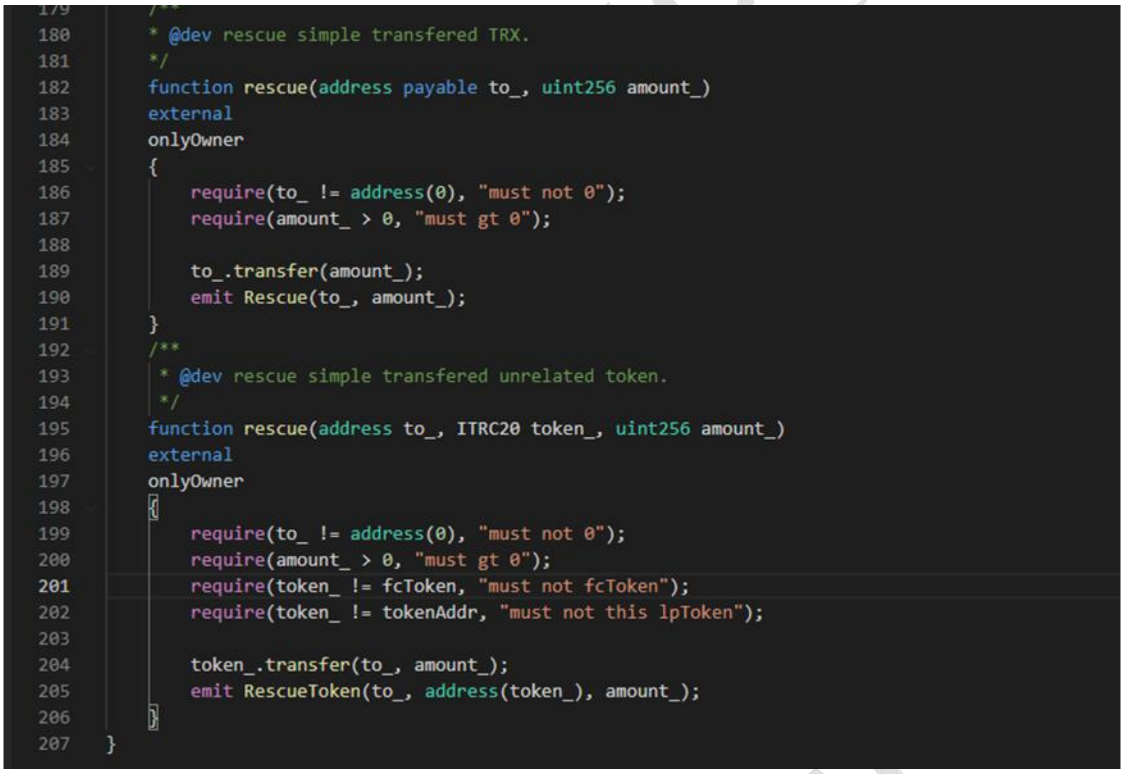
```
    function withdrawAndGetReward(uint256 amount) public updateReward(msg.sender) checkStart {
        require(amount <= balanceOf(msg.sender), "Cannot withdraw exceed the balance");
        withdraw(amount);
        getReward();
    }
```

Figure 10 source code of *withdrawAndGetReward*

- Related functions: *withdrawAndGetReward, withdraw, getReward*
- Result: Pass

(7) Misoperation rescue function

- Description: As shown in the figure below, the contract implements two rescue functions for the owner to withdraw the tokens sent by the user by mistake to the specified address. One is used to withdraw TRX and one is used to withdraw other TRC20 tokens (LP tokens and FC tokens cannot be withdrawn).



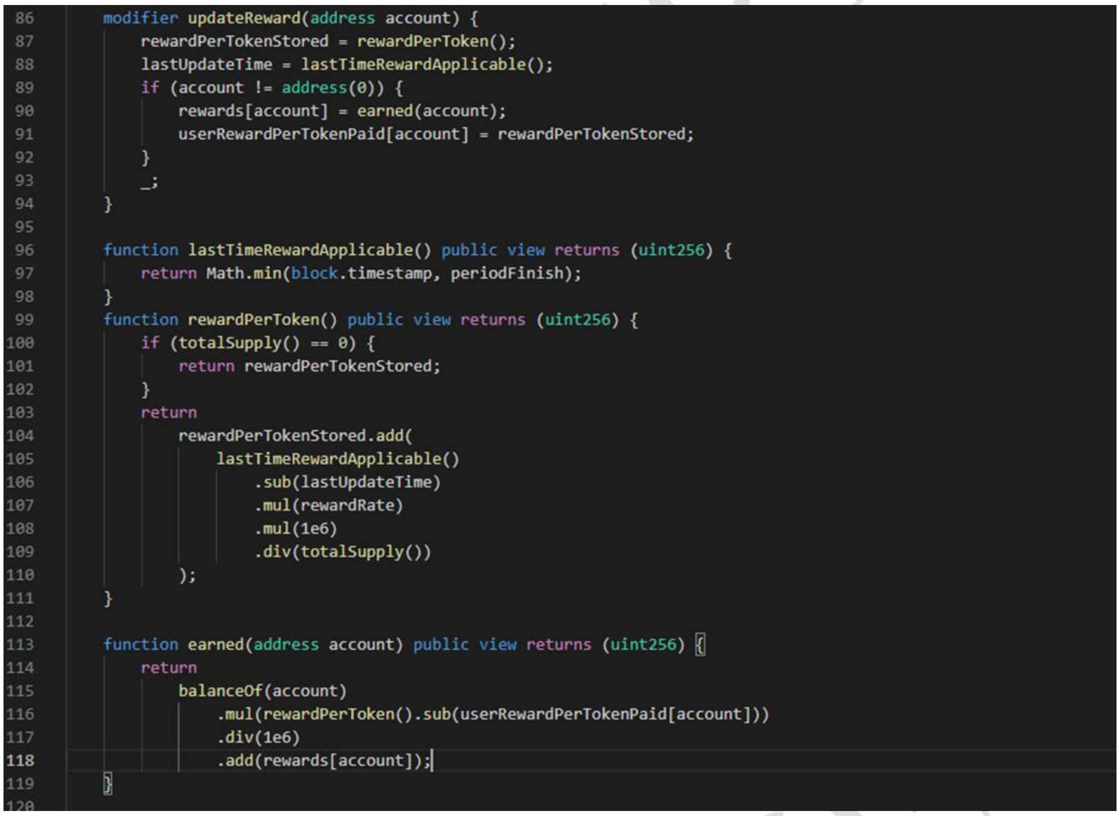
```
    /**
     * @dev rescue simple transfered TRX.
     */
    function rescue(address payable to_, uint256 amount_)
    external
    onlyOwner
    {
        require(to_ != address(0), "must not 0");
        require(amount_ > 0, "must gt 0");

        to_.transfer(amount_);
        emit Rescue(to_, amount_);
    }
    /**
     * @dev rescue simple transfered unrelated token.
     */
    function rescue(address to_, ITRC20 token_, uint256 amount_)
    external
    onlyOwner
    {
        require(to_ != address(0), "must not 0");
        require(amount_ > 0, "must gt 0");
        require(token_ != fcToken, "must not fcToken");
        require(token_ != tokenAddr, "must not this lpToken");

        token_.transfer(to_, amount_);
        emit RescueToken(to_, address(token_), amount_);
    }
}
```

Figure 11 source code of *rescue*

- Related functions: *rescue*
- Result: Pass

(8) Reward related data query function

- Description: As shown in the figure below, contract users can query the earliest time stamp between the current time stamp and the phase completion time by calling the lastTimeRewardApplicable function; calling the rewardPerToken function can query the stake rewards available for each stake token; calling the earned function can query the total stake rewards obtained by the specified address .



```
    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }
    function rewardPerToken() public view returns (uint256) {
        if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime)
                    .mul(rewardRate)
                    .mul(1e6)
                    .div(totalSupply())
            );
    }

    function earned(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
                .div(1e6)
                .add(rewards[account]);
    }
```

Figure 12 source code of *lastTimeRewardApplicable*, *rewardPerToken* and *earned*

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

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project FC-FCTRXBasepool. The problems found by the audit team during the audit process have been notified to the project party and fixed, the overall audit result of the FC-FCTRXBasepool project's smart contract is **Pass**.

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com