BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number：202102261534

Report Query Name: XF

Smart Contract Info：

Smart Contract Name	**Smart Contract Address**	**Smart Contract Address Link**
xFarmer	0xe0fe25eefcfcaddef844fe30b8be1d68ac6b7af3	https://hecoinfo.com/address/0xe0fe25eefcfcaddef844fe30b8be1d68ac6b7af3#code
StakingRewards(XF-XF)	0xA7b4E0c598305FC695b837A3D5E8Cfe121DED34b	https://hecoinfo.com/address/0xA7b4E0c598305FC695b837A3D5E8Cfe121DED34b#code
StakingRewards(XF-USDT)	0xb2D4688598aAd83Be3bF9243487817125E1dE95C	https://hecoinfo.com/address/0xb2D4688598aAd83Be3bF9243487817125E1dE95C#code
StakingRewards(XF-HT)	0x64C9fc836c5EBe8483814F669f0E34085d1cdF4f	https://hecoinfo.com/address/0x64C9fc836c5EBe8483814F669f0E34085d1cdF4f#code
StakingRewards(XF-HUSD)	0x848236841886459a308Fe180FDAF2C3dd83843c6	https://hecoinfo.com/address/0x848236841886459a308Fe180FDAF2C3dd83843c6#code
StakingRewards(XF-HBTC)	0xcb3440B517533c08b84CaBD8Cf8B620A0d131F29	https://hecoinfo.com/address/0xcb3440B517533c08b84CaBD8Cf8B620A0d131F29#code

Start Date：2021.02.26

Completion Date：2021.02.26

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 XF, including Coding Standards, Security, and Business Logic. **The XF project 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 HT.
- 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. All the stake incentive pool codes of the project are the same, only the stake currency is set when the contract is deployed.

3.1 Business analysis of Contract Token XF

(1) Basic Token Information

Token name	xFarmer
Token symbol	XF
decimals	18
totalSupply	The initial supply is 10 thousand, can not be destroyed, mintable, the cap of the total amount is 200 million
Token type	HRC20

Table 1 Basic Token Information

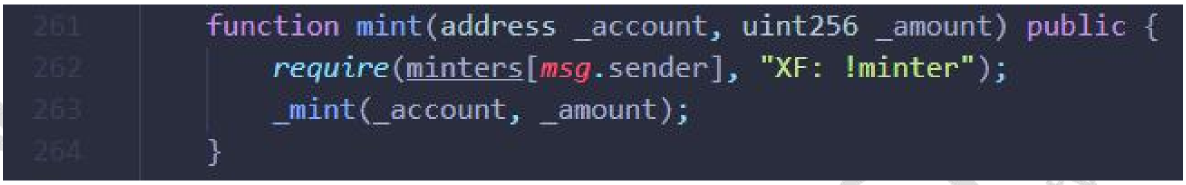
(2) HRC20 Token Standard Functions

- Description: The Token Contract implements a Token which conforms to the HRC20 Standards. It should be noted that the user can directly call the *approve* function to set the approval value for the specified address, but in order to avoid multiple authorizations, it is recommended to use the *increaseAllowance* and *decreaseAllowance* functions when modifying the approval value instead of using the *approve* function directly.
- Related functions: *name, symbol, decimals, totalSupply, balanceOf, allowance, transfer, transferFrom, approve, increaseAllowance, decreaseAllowance*

- Result: Pass

(3) mint function

- Description: As shown in the figure below, the internal function _*mint* is called by minter to mint tokens for the _account address. the cap of minting is 210 million.



```
function mint(address _account, uint256 _amount) public {
    require(minters[msg.sender], "XF: !minter");
    _mint(_account, _amount);
}
```

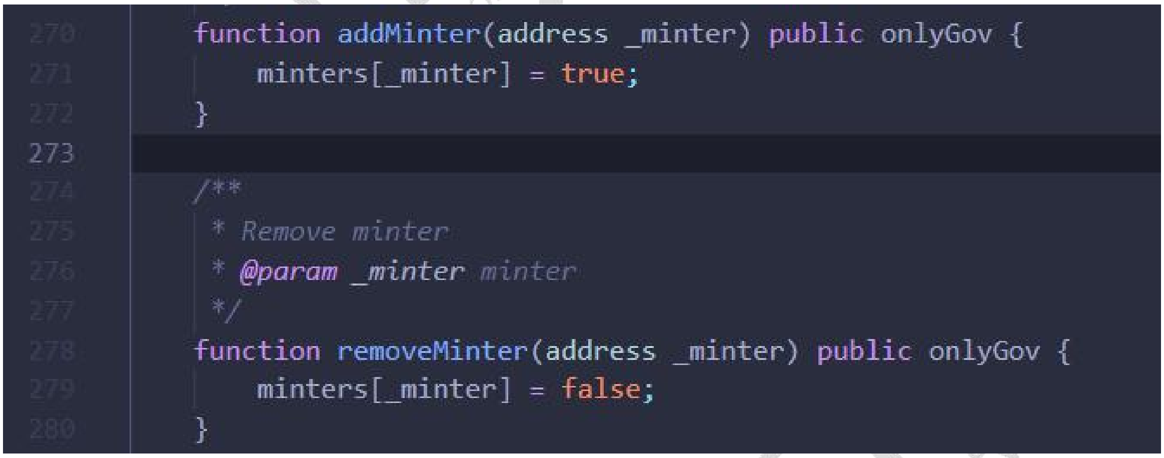
Figure 1 source code of *mint*

- Related functions: *mint*, *_mint*, *_beforeTokenTransfer*

- Result: Pass

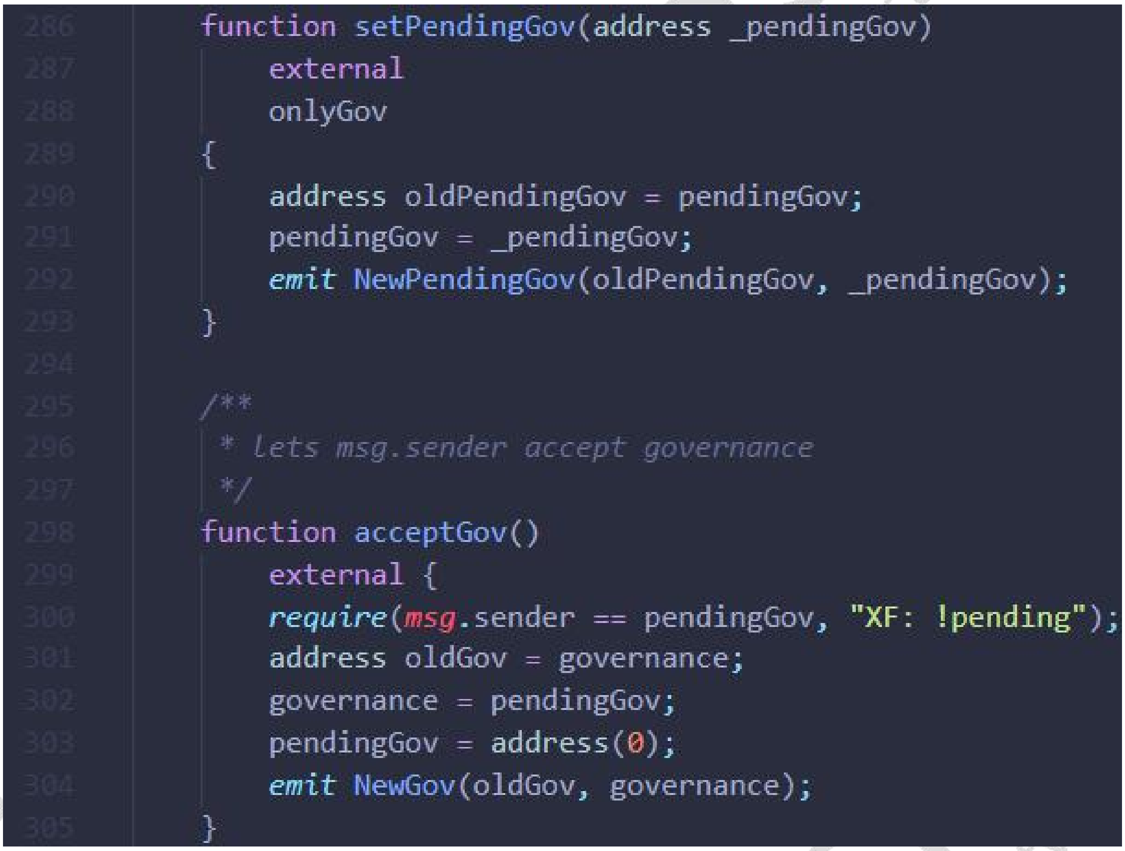
(4) Related governance functions

- Description: The contract implements functions such as *addMinter*, *removeMinter*, *setPendingGov*, *acceptGov* for contract governance. *SetPendingGov* is used to set up the preparatory governance, and *acceptGov* is used to receive governance privileges as the preparatory governance . Contract governance can add or remove contract minter through *addMinter* and *removeMinter.*



```
function addMinter(address _minter) public onlyGov {
    minters[_minter] = true;
}

/**
 * Remove minter
 * @param _minter minter
 */
function removeMinter(address _minter) public onlyGov {
    minters[_minter] = false;
}
```

Figure 2 source code of *addMinter* and *removeMinter*



```
function setPendingGov(address _pendingGov)
    external
    onlyGov
{
    address oldPendingGov = pendingGov;
    pendingGov = _pendingGov;
    emit NewPendingGov(oldPendingGov, _pendingGov);
}

/**
 * lets msg.sender accept governance
 */
function acceptGov()
    external {
    require(msg.sender == pendingGov, "XF: !pending");
    address oldGov = governance;
    governance = pendingGov;
    pendingGov = address(0);
    emit NewGov(oldGov, governance);
}
```

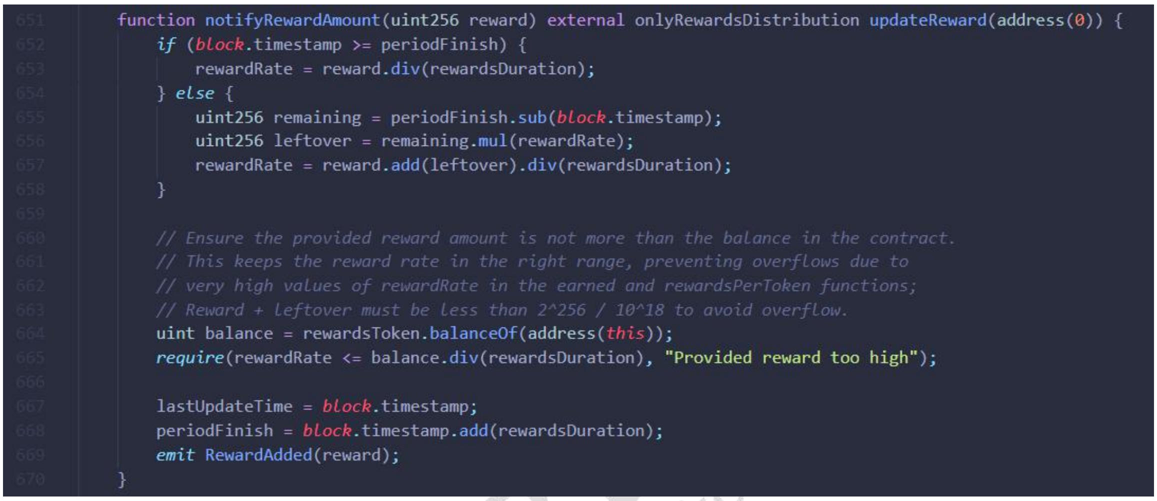
Figure 3 source code of *setPendingGov* and *acceptGov*

- Related functions: *addMinter*, *removeMinter*, *setPendingGov*, *acceptGov*
- Result: Pass

3.2 Business analysis of Contract StakingReward

(1) Initialization

- Description: The "Stake-Award" mode of the contract needs to initialize the relevant parameters (award ratio *rewardRate*, first update time *lastUpdateTime*, phase completion time *periodFinish*), calls the *notifyRewardAmount* function through the specified award allocation administrator address *rewardDistribution*, enter the initial award value reward, used to calculate the award ratio initialize the stake and award related parameters.



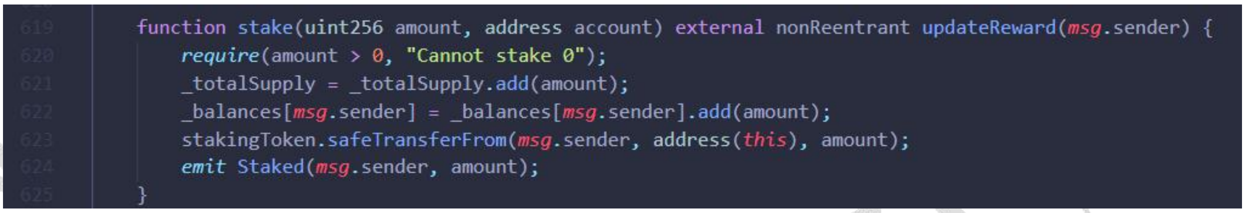
```
function notifyRewardAmount(uint256 reward) external onlyRewardsDistribution updateReward(address(0)) {
    if (block.timestamp >= periodFinish) {
        rewardRate = reward.div(rewardsDuration);
    } else {
        uint256 remaining = periodFinish.sub(block.timestamp);
        uint256 leftover = remaining.mul(rewardRate);
        rewardRate = reward.add(leftover).div(rewardsDuration);
    }

    // Ensure the provided reward amount is not more than the balance in the contract.
    // This keeps the reward rate in the right range, preventing overflows due to
    // very high values of rewardRate in the earned and rewardsPerToken functions;
    // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
    uint balance = rewardsToken.balanceOf(address(this));
    require(rewardRate <= balance.div(rewardsDuration), "Provided reward too high");

    lastUpdateTime = block.timestamp;
    periodFinish = block.timestamp.add(rewardsDuration);
    emit RewardAdded(reward);
}
```

Figure 4 source code of *notifyRewardAmount*

- Related functions: *notifyRewardAmount*
- Result: Pass

(2) Stake function

- Description: The contract implements the *stake* function for the stake token, and the user authorizes the contract address in advance. By calling the *transferFrom* function in the contract, each time the function stake token is called, the reward-related data is updated through the modifier *updateReward*.



```
function stake(uint256 amount, address account) external nonReentrant updateReward(msg.sender) {
    require(amount > 0, "Cannot stake 0");
    _totalSupply = _totalSupply.add(amount);
    _balances[msg.sender] = _balances[msg.sender].add(amount);
    stakingToken.safeTransferFrom(msg.sender, address(this), amount);
    emit Staked(msg.sender, amount);
}
```

Figure 5 source code of *stake*

- Related functions: *stake*, *updateReward*

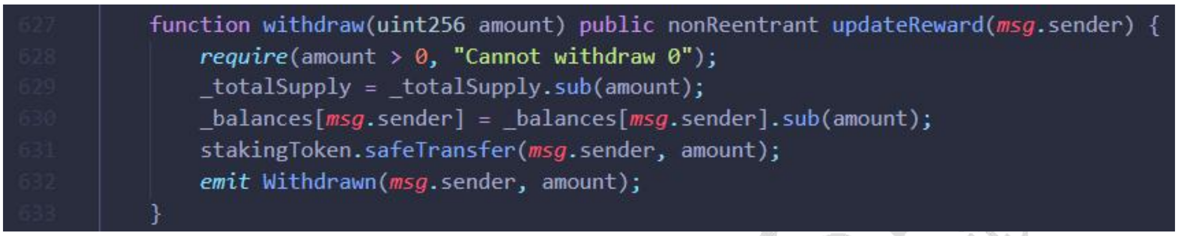
- Security recommendation: parameter *account* is not used in the stake function. Redundant code. It is recommended to delete it.

- Repair result: ignore

- Result: Pass

(3) Withdraw function

- Description: The contract implements the *withdraw* function to extract the staked token. By calling the *transfer* function in the contract, the contract address transfers the specified number of tokens to the function caller (user) address; each time the function is called to extract the token, the reward data is updated through the modifier *updateReward*.



```
function withdraw(uint256 amount) public nonReentrant updateReward(msg.sender) {
    require(amount > 0, "Cannot withdraw 0");
    _totalSupply = _totalSupply.sub(amount);
    _balances[msg.sender] = _balances[msg.sender].sub(amount);
    stakingToken.safeTransfer(msg.sender, amount);
    emit Withdrawn(msg.sender, amount);
}
```

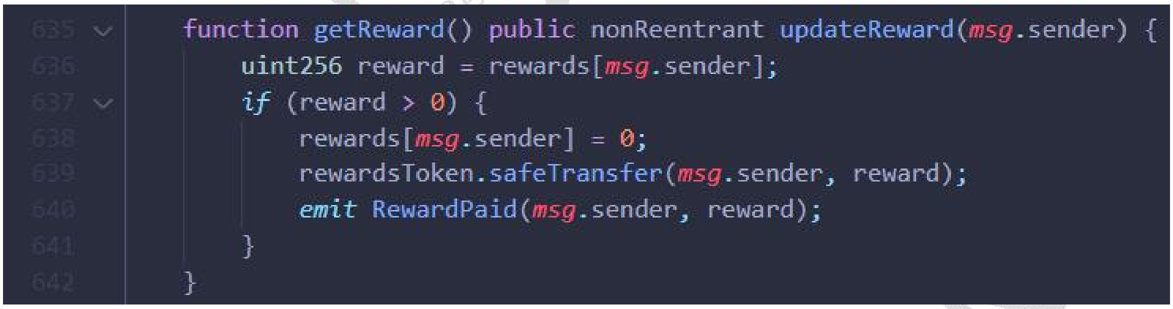
Figure 6 source code of *withdraw*

- Related functions: *withdraw*, *updateReward*

- Result: Pass

(4) Get reward function

- Description: The contract implements the *getReward* function to receive the stake reward. By calling the transfer function in the contract, the contract address transfers the specified number of tokens (all stake rewards of the user) to the function caller (user) address; each time the function stake token is called, the reward-related data is updated through the modifier *updateReward*.



```
function getReward() public nonReentrant updateReward(msg.sender) {
    uint256 reward = rewards[msg.sender];
    if (reward > 0) {
        rewards[msg.sender] = 0;
        rewardsToken.safeTransfer(msg.sender, reward);
        emit RewardPaid(msg.sender, reward);
    }
}
```

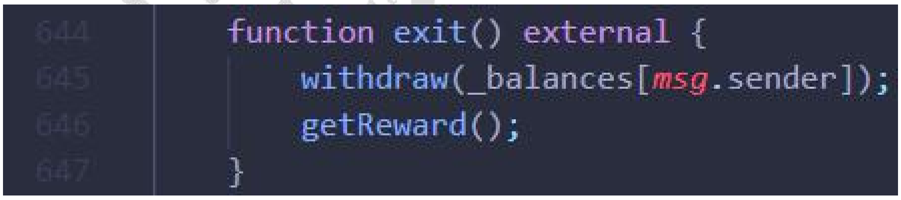
Figure 7 source code of *getReward*

- Related functions: *getReward*, *updateReward*

- Result: Pass

(5) Exit function

- Description: The contract implements the *exit* function for the caller to withdraw from the stake reward participation, calls the *withdraw* function to extract all the staked tokens, calls the *getReward* function to get the caller's stake reward, and ends the "stake-reward" mode participation. At this time, the user address cannot get a new stake reward because the number of staked tokens is empty.



```
    function exit() external {
        withdraw(_balances[msg.sender]);
        getReward();
    }
```

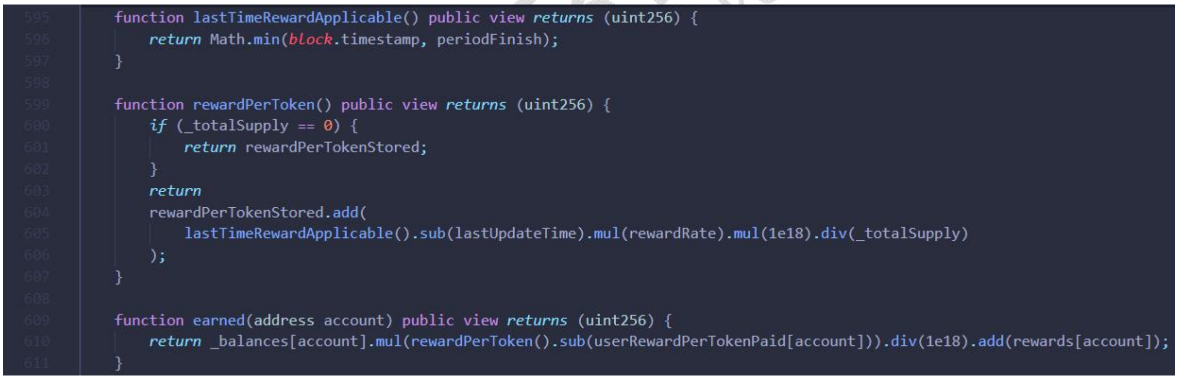
Figure 8 source code of *exit*

- Related functions: *exit*, *withdraw*, *getReward*

- Result: Pass

(6) Related parameter query function

- Description: Contract users can query the earliest timestamps in the current timestamp and phase completion time by calling the *lastTimeRewardApplicable* function; call the *rewardPerToken* function to query the stake rewards available for each stake token; and call the *earned* function to query the total stake awards obtained at the specified address.



```
    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(1e18).div(_totalSupply)
        );
    }

    function earned(address account) public view returns (uint256) {
        return _balances[account].mul(rewardPerToken().sub(userRewardPerTokenPaid[account])).div(1e18).add(rewards[account]);
    }
```

Figure 9 source code of related functions

- 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 XF. 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 XF 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