BEOSIN
Blockchain Security

ICPSwap-Staking pool

Smart Contract Security Audit

No. 202407011039

Jul 1st, 2024

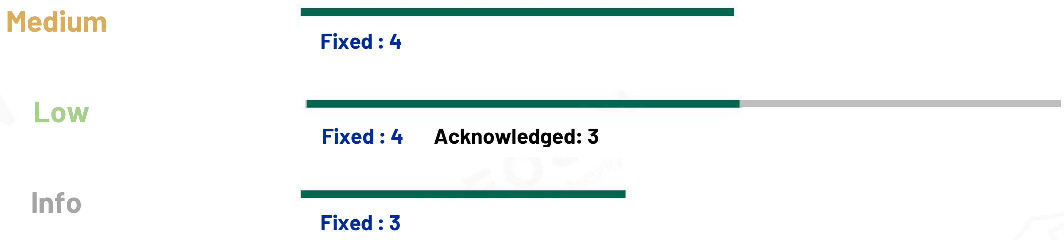
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 4 Medium-risk ,7 Low-risk and 3 Info items were identified in the ICPSwap-Staking pool project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:



- **Project Description:**

Business overview

ICPSwap-Staking pool is a staking and mining project that includes three main functional modules: StakingFeeReceiver, StakingPool, and StakingPoolFactory. The following is an introduction to each function.

StakingFeeReceiver： The Controller has the authority to call the `transferAll` and `transfer` functions to withdraw DIP20, ICRC, and ICP standard tokens stored in this canister. Additionally, the collected reward fees can be claimed to this canister using the `claim` function.

StakingPool:

The actor implements the staking-related logic. Users can call the `deposit` and `depositFrom` functions to transfer a certain amount of staking tokens into this canister, thereby increasing their `stakeTokenBalance`. Users can withdraw the staked principal based on this `stakeTokenBalance` or call the `stake` function to convert their `stakeTokenBalance` into `stakeAmount` to generate rewards.

The project uses the `accPerShare` model for reward distribution. Users with admin permissions can modify the staking period and reward allocation. During the reward period, users can withdraw reward tokens and the staked principal using the `unstake` and `withdraw` functions. After the reward period ends, the admin account can call `liquidation` to clear the users' staked amounts. Subsequently, refunds to users can be processed.

StakingPoolFactory：

This actor is primarily responsible for the lifecycle management of StakingPools, including creating, ending, and deleting StakingPools. The admin can call the `createStakingPool` function to create a new reward-eligible StakingPool. Additionally, it can synchronize the created StakingPool information and the staking user information to this canister, maintaining the authority to manage the reward cycles of the StakingPools.

1 Overview

1.1 Project Overview

Project Name	ICPSwap-Staking pool
Project Language	Motoko
Platform	IC
Code Base	https://github.com/ICPSwap-Labs/ICPSwap-Staking pool/
Commit Id	9138126fa4375abe774fd63ce3532ecf2c4b2bee 0e023e7cc14c410d446c418f0e78cac25c1415ba da9cf48bc83c986ef0689e7c1548bcf1302a0b28 41c6b90764f4c523514d8e5bd9688b2f26dcfd51

1.2 Audit Overview

Audit work duration: May 10, 2024 – Jul 1, 2024

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
Staking pool-01	The project lacks a rollback mechanism	Medium	Fixed
Staking pool-02	Centralized risk	Medium	Fixed
Staking pool-03	The user's principal may be unable to be withdrawn	Medium	Fixed
Staking pool-04	The withdrawRemainingRewardToken is unreasonable	Medium	Fixed
Staking pool-05	The withdrawal of RewardFee is unreasonable	Low	Acknowledged
Staking pool-06	Incorrect setting of the stakingStandard value	Low	Fixed
Staking pool-07	Reward calculation defect	Low	Partially Fixed
Staking pool-08	The stop function does not check the time	Low	Acknowledged
Staking pool-09	Global data is not separated from user updates	Low	Fixed
Staking pool-10	The calculation of the total fee amount is not synchronized	Low	Fixed
Staking pool-11	The _getRewardInterval function does not check the start time	Low	Acknowledged
Staking pool-12	The return value of getInitArgs is incomplete	Info	Fixed
Staking pool-13	Redundant code	Info	Fixed
Staking pool-14	Lack of permission check	Info	Fixed

Finding Details:

[Staking pool-01] The project lacks a rollback mechanism

Severity Level	**Medium**
Type	Business Security
Lines	StakingPool.mo#L950-997
Description	The _harvest function of this project, there is a pattern of modifying variables first and then making external calls or performing state checks. Moreover, in case of failure, these checks or calls do not trigger a state rollback.In the _harvest function after the _totalRewardFee variable is modified, if the execution of _pay failed due to either insufficient actor balance or other exceptional circumstances, it will not be rolled back.

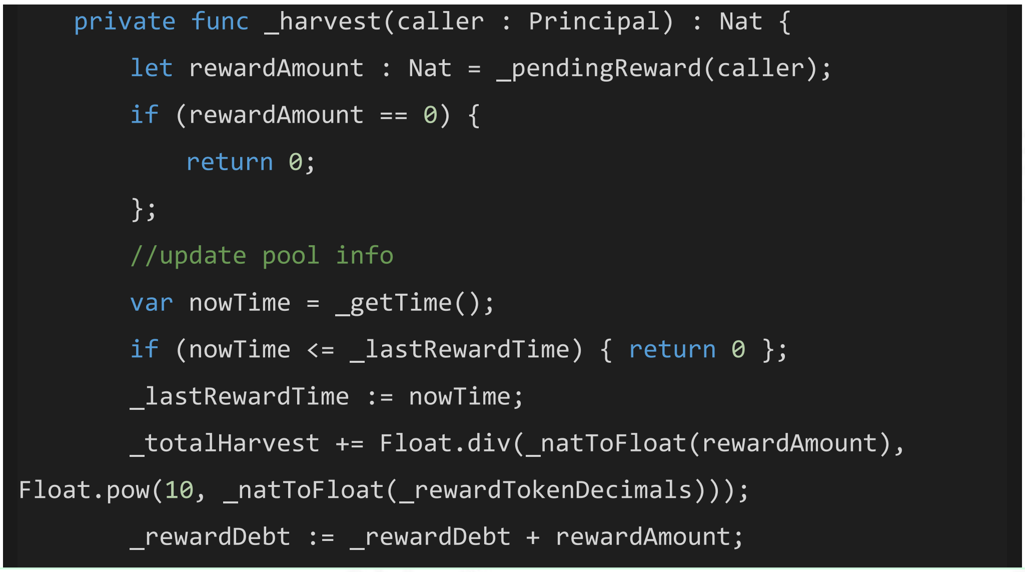
```
    private func _harvest(caller : Principal) : async
Result.Result<Nat, Text> {
        await _updatePool();
        var _userInfo : Types.UserInfo = _getUserInfo(caller);
        var pending : Nat = Nat.sub(Nat.div(Nat.mul(_userInfo.amount,
_poolInfo.accPerShare), _arithmeticFactor), _userInfo.rewardDebt);
        if (pending == 0 or pending < _poolInfo.rewardTokenFee) {
            return #ok(0);
        };
        var rewardAmount = pending;
        let rewardFee : Nat = Nat.div(Nat.mul(rewardAmount,
_rewardFee), 100);
        if (rewardFee > 0) {
            _totalRewardFee += rewardFee;
            rewardAmount := rewardAmount - rewardFee;
        };

        if (rewardAmount < _poolInfo.rewardTokenFee) {
            _totalRewardFee -= rewardFee;
            return #ok(0);
        };
        switch (await _pay(_poolInfo.rewardToken,
Principal.fromActor(this), null, caller, null, rewardAmount -
_poolInfo.rewardTokenFee)) {
```

Recommendation

It is recommended to implement a mechanism across the entire project that enables the rollback of critical variable modifications in case of subsequent exceptions. For instance, consider utilizing the Prim.Trap function for rollback when handling exceptions.

Status

Fixed. The _harvest function will not be called asynchronously; modifying data will be atomic.



```
    private func _harvest(caller : Principal) : Nat {
        let rewardAmount : Nat = _pendingReward(caller);
        if (rewardAmount == 0) {
            return 0;
        };
        //update pool info
        var nowTime = _getTime();
        if (nowTime <= _lastRewardTime) { return 0 };
        _lastRewardTime := nowTime;
        _totalHarvest += Float.div(_natToFloat(rewardAmount),
Float.pow(10, _natToFloat(_rewardTokenDecimals)));
        _rewardDebt := _rewardDebt + rewardAmount;
```

[Staking pool-02] Centralized risk

Severity Level	Medium
Type	Business Security
Lines	StakingPool.mo#L105-120

Description

The admin can call the updateStakingPool function to modify critical parameters of _poolInfo such as staking token related information:token address,decimals,etc.This could lead to inconsistencies between the user's withdrawal and staked tokens, resulting in asset losses for the user.

```
    public shared (msg) func updateStakingPool(params :
Types.UpdateStakingPool) : async Result.Result<Bool, Text> {
        _checkAdminPermission(msg.caller);
        _poolInfo.rewardToken := params.rewardToken;
        _poolInfo.rewardTokenFee := params.rewardTokenFee;
        _poolInfo.rewardTokenSymbol := params.rewardTokenSymbol;
        _poolInfo.rewardTokenDecimals := params.rewardTokenDecimals;
        _poolInfo.stakingToken := params.stakingToken;
        _poolInfo.stakingTokenFee := params.stakingTokenFee;
        _poolInfo.stakingTokenSymbol := params.stakingTokenSymbol;
        _poolInfo.stakingTokenDecimals :=
params.stakingTokenDecimals;

        _poolInfo.startTime := params.startTime;
        _poolInfo.bonusEndTime := params.bonusEndTime;
        _poolInfo.rewardPerTime := params.rewardPerTime;
        return #ok(true);
```

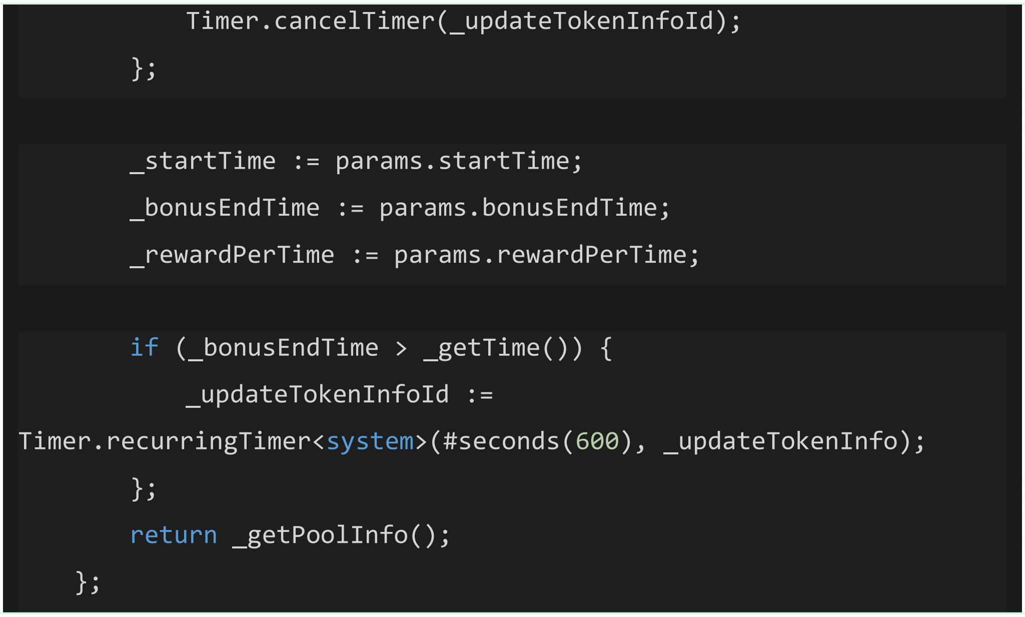
Recommendation

It is recommended to keep stakingToken relevant variables in the pool unchanged,or the reward tokens can be modified before distributing the rewards.

Status

Fixed. The updateStakingPool function will not update the token information.

```
    public shared (msg) func updateStakingPool(params :
Types.UpdateStakingPool) : async
Result.Result<Types.PublicStakingPoolInfo, Text> {
        _checkAdminPermission(msg.caller);

        if (_bonusEndTime <= _getTime()) {
```



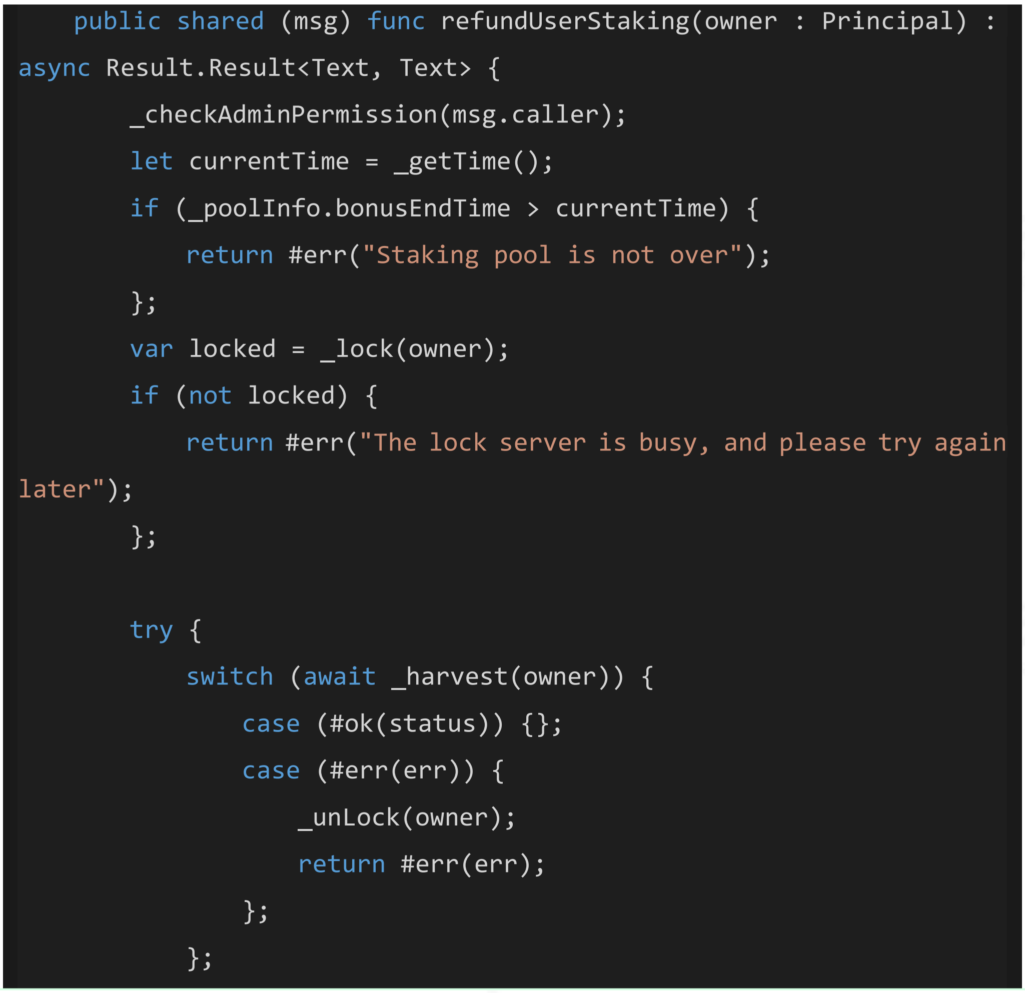
```
            Timer.cancelTimer(_updateTokenInfoId);
        };

        _startTime := params.startTime;
        _bonusEndTime := params.bonusEndTime;
        _rewardPerTime := params.rewardPerTime;

        if (_bonusEndTime > _getTime()) {
            _updateTokenInfoId :=
Timer.recurringTimer<system>(#seconds(600), _updateTokenInfo);
        };
        return _getPoolInfo();
    };
```

[Staking pool-03] The user's principal may be unable to be withdrawn

Severity Level	Medium
Type	Business Security
Lines	StakingPool.mo#L226-297,L539-608

Description

Users can call the withdraw function, and the admin can call the refundUserStaking function to release the staked tokens. The refundUserStaking function calls _harvest to distribute the staking rewards to the user. However, since the function does not check whether the actor has sufficient balance to pay the rewards, if the reward token balance is insufficient, the _harvest function will fail, preventing the user from withdrawing their principal staked amount.



```
    public shared (msg) func refundUserStaking(owner : Principal) :
async Result.Result<Text, Text> {
        _checkAdminPermission(msg.caller);
        let currentTime = _getTime();
        if (_poolInfo.bonusEndTime > currentTime) {
            return #err("Staking pool is not over");
        };
        var locked = _lock(owner);
        if (not locked) {
            return #err("The lock server is busy, and please try again
later");
        };

        try {
            switch (await _harvest(owner)) {
                case (#ok(status)) {};
                case (#err(err)) {
                    _unLock(owner);
                    return #err(err);
                };
            };
```

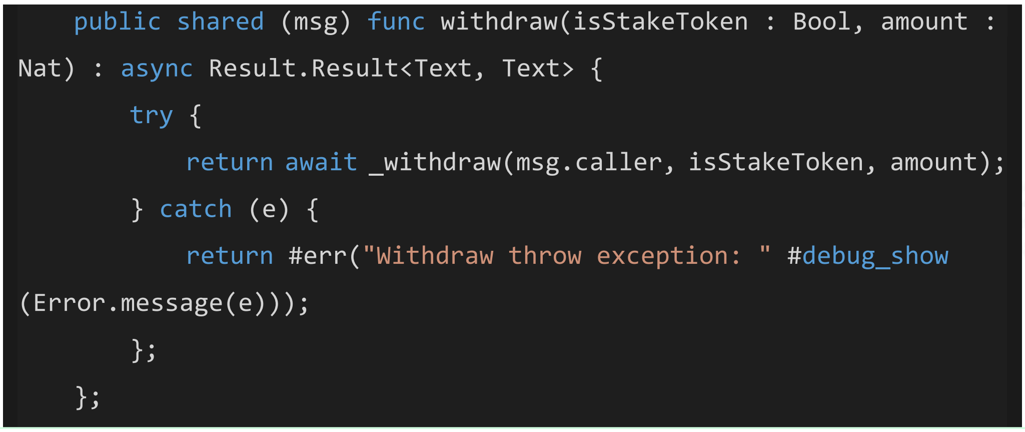
Recommendation

1. Add an emergency withdrawal function to allow users to withdraw their staked principal in urgent situations.

2. Alternatively, when the reward token balance is insufficient, directly distribute the available balance as the reward, instead of failing the _harvest function.

Status

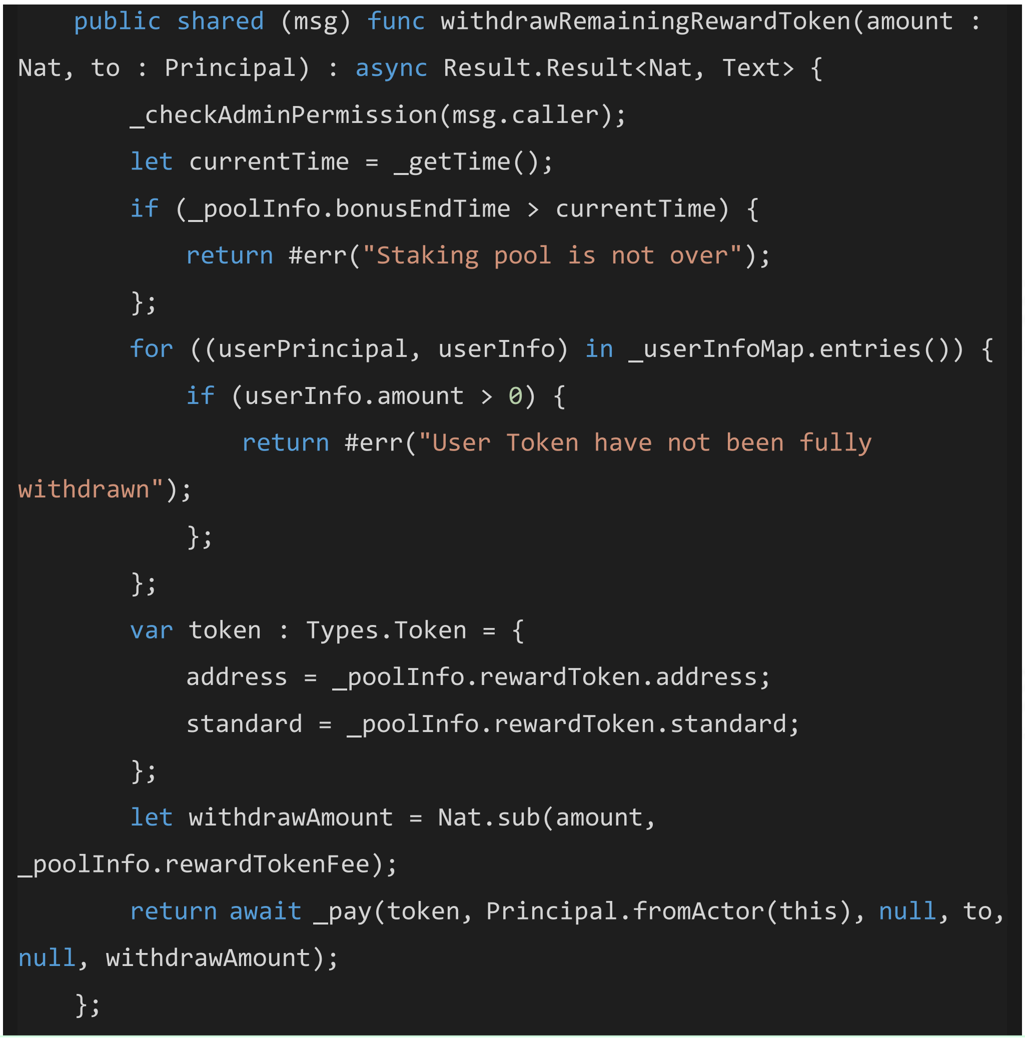
Fixed. The withdraw function now allows for the separate withdrawal of staked principal and reward tokens.



```
    public shared (msg) func withdraw(isStakeToken : Bool, amount : Nat) : async Result.Result<Text, Text> {
        try {
            return await _withdraw(msg.caller, isStakeToken, amount);
        } catch (e) {
            return #err("Withdraw throw exception: " #debug_show (Error.message(e)));
        };
    };
```

[Staking pool-04] The withdrawRemainingRewardToken is unreasonable

Severity Level	**Medium**
Type	Business Security
Lines	StakingPool.mo#L105-120
Description	The `withdrawRemainingRewardToken` function allows the admin to withdraw the remaining funds, but it requires that all users have already withdrawn their staked tokens. However, there may be a scenario where a user's staked token amount is insufficient to cover the transaction fee, causing their tokens to remain locked in the contract. In this case, the admin would be unable to withdraw the remaining rewards.



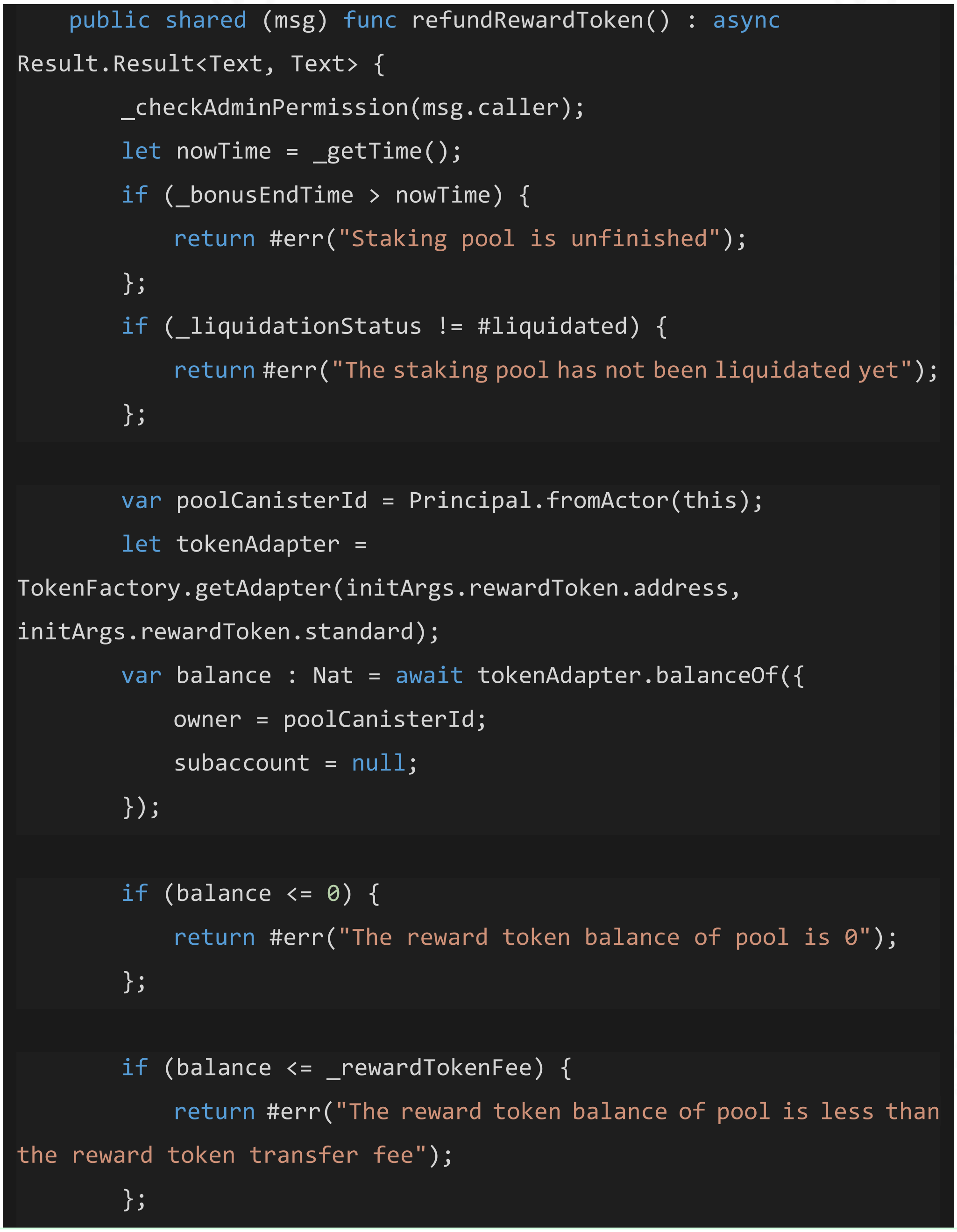
```
    public shared (msg) func withdrawRemainingRewardToken(amount : Nat, to : Principal) : async Result.Result<Nat, Text> {
        _checkAdminPermission(msg.caller);
        let currentTime = _getTime();
        if (_poolInfo.bonusEndTime > currentTime) {
            return #err("Staking pool is not over");
        };
        for ((userPrincipal, userInfo) in _userInfoMap.entries()) {
            if (userInfo.amount > 0) {
                return #err("User Token have not been fully withdrawn");
            };
        };
        var token : Types.Token = {
            address = _poolInfo.rewardToken.address;
            standard = _poolInfo.rewardToken.standard;
        };
        let withdrawAmount = Nat.sub(amount, _poolInfo.rewardTokenFee);
        return await _pay(token, Principal.fromActor(this), null, to, null, withdrawAmount);
    };
```

Recommendation	It is recommended to in the withdraw and `refundUserStaking` functions, when the user's staked balance is insufficient to pay the staking token transaction fee, set `_userInfo.amount` to 0, and synchronize the update of

_userInfo.rewardDebt.

Status

Fixed. The project now uses a liquidation mode, where a refund to the user must be processed before any further calls can be made. Additionally, it will check if the transfer amount is greater than the transaction fee.



```
    public shared (msg) func refundRewardToken() : async
Result.Result<Text, Text> {
        _checkAdminPermission(msg.caller);
        let nowTime = _getTime();
        if (_bonusEndTime > nowTime) {
            return #err("Staking pool is unfinished");
        };
        if (_liquidationStatus != #liquidated) {
            return #err("The staking pool has not been liquidated yet");
        };

        var poolCanisterId = Principal.fromActor(this);
        let tokenAdapter =
TokenFactory.getAdapter(initArgs.rewardToken.address,
initArgs.rewardToken.standard);
        var balance : Nat = await tokenAdapter.balanceOf({
            owner = poolCanisterId;
            subaccount = null;
        });

        if (balance <= 0) {
            return #err("The reward token balance of pool is 0");
        };

        if (balance <= _rewardTokenFee) {
            return #err("The reward token balance of pool is less than
the reward token transfer fee");
        };
```

[Staking pool-05] The withdrawal of RewardFee is unreasonable

<table>
<tr><td><b>Severity Level</b></td><td>Low</td></tr>
<tr><td><b>Type</b></td><td>General Vulnerability</td></tr>
<tr><td><b>Lines</b></td><td>StakingPool.mo#L304-353</td></tr>
<tr><td><b>Description</b></td><td>The _feeReceiverCid can call the withdrawRewardFee function to withdraw the accumulated fee earnings, but when the actor's balance is insufficient to pay the current accumulated effective reward pending, the _feeReceiverCid will not be able to receive any rewards.
<pre><code>   if (not (balance > 0)) {
                _unLock(_feeReceiverCid);
                return #err("The reward token balance of pool is 0");
            };
            var fee = _poolInfo.rewardTokenFee;
            if (not (balance > fee)) {
                _unLock(_feeReceiverCid);
                return #err("The reward token balance of pool is less
than the reward token transfer fee");
            };
            let pending = Nat.sub(_totalRewardFee,
_receivedRewardFee);
            if (not (balance > pending)) {
                _unLock(_feeReceiverCid);
                return #err("The reward token balance of pool is less
than the reward fee");
            }
</code></pre></td></tr>
<tr><td><b>Recommendation</b></td><td>It is recommended that when the balance is insufficient, the actor's remaining balance should be sent to the _feeReceiverCid.</td></tr>
<tr><td><b>Status</b></td><td><b>Acknowledged</b></td></tr>
</table>

[Staking pool-06] Incorrect setting of the stakingStandard value

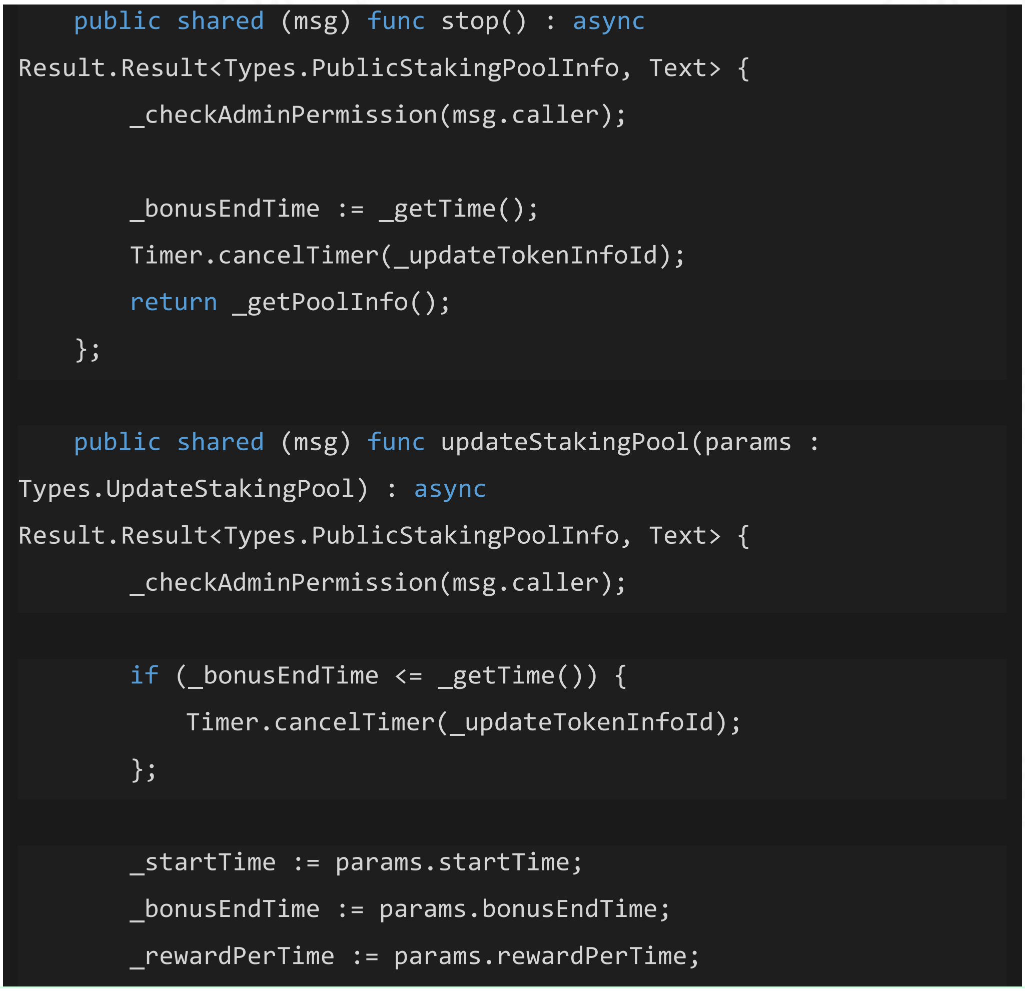
<table>
<tr><td><b>Severity Level</b></td><td>Low</td></tr>
<tr><td><b>Type</b></td><td>Business Security</td></tr>
<tr><td><b>Lines</b></td><td>StakingPool.mo#L105-120</td></tr>
<tr><td><b>Description</b></td><td>The incorrect setting of stakingStandard to <code>_poolInfo.rewardToken.standard</code> in the <code>refundUserStaking</code> and <code>withdraw</code> function will lead to inaccurate recording of the user's <code>_stakingRecordBuffer</code> information.
<pre><code>   to = caller;
                    rewardStandard = _poolInfo.rewardToken.standard;
                    rewardToken = _poolInfo.rewardToken.address;
                    rewardTokenSymbol = _poolInfo.rewardTokenSymbol;
                    rewardTokenDecimals =
_poolInfo.rewardTokenDecimals;
                    stakingStandard = _poolInfo.rewardToken.standard;
</code></pre></td></tr>
<tr><td><b>Recommendation</b></td><td>It is recommended to modify based on business needs.</td></tr>
<tr><td><b>Status</b></td><td><b>Fixed.</b>
<pre><code>   var record = {
            from = Principal.fromActor(this);
            to = caller;
            rewardStandard = initArgs.rewardToken.standard;
            rewardToken = initArgs.rewardToken.address;
            rewardTokenSymbol = _rewardTokenSymbol;
            rewardTokenDecimals = _rewardTokenDecimals;
            stakingStandard = initArgs.stakingToken.standard;
            stakingToken = initArgs.stakingToken.address;
            stakingTokenDecimals = _stakingTokenDecimals;
            stakingTokenSymbol = _stakingTokenSymbol;
</code></pre></td></tr>
</table>

[Staking pool-07] Modifying the stake status or period did not update _accPerShare

Severity Level	Low
Type	Business Security
Lines	StakingPool.mo#L90-114

Description

When modifying the stake status, such as the time period and rewardPerTime, the _harvest function is not called to settle _accPerShare. This omission leads to unclaimed rewards before the modification being affected by _accPerShare and bonusEndTime, resulting in inaccurate final reward amounts.



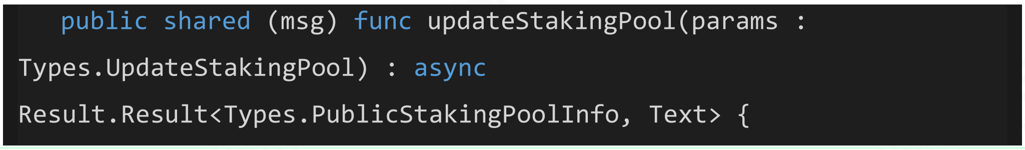
```
    public shared (msg) func stop() : async
Result.Result<Types.PublicStakingPoolInfo, Text> {
        _checkAdminPermission(msg.caller);

        _bonusEndTime := _getTime();
        Timer.cancelTimer(_updateTokenInfoId);
        return _getPoolInfo();
    };

    public shared (msg) func updateStakingPool(params :
Types.UpdateStakingPool) : async
Result.Result<Types.PublicStakingPoolInfo, Text> {
        _checkAdminPermission(msg.caller);

        if (_bonusEndTime <= _getTime()) {
            Timer.cancelTimer(_updateTokenInfoId);
        };

        _startTime := params.startTime;
        _bonusEndTime := params.bonusEndTime;
        _rewardPerTime := params.rewardPerTime;
```

Recommendation

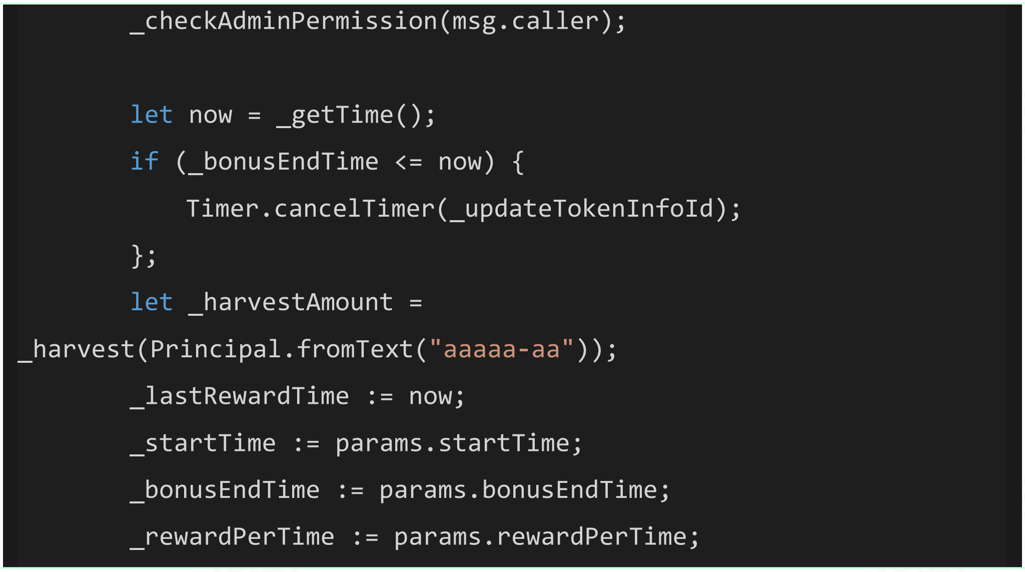
It is recommended to call the _harvest function (with any principal) when invoking the above functions to settle _accPerShare.

Status

Partially Fixed. The stop function has not been modified.



```
   public shared (msg) func updateStakingPool(params :
Types.UpdateStakingPool) : async
Result.Result<Types.PublicStakingPoolInfo, Text> {
```



```
        _checkAdminPermission(msg.caller);

        let now = _getTime();
        if (_bonusEndTime <= now) {
            Timer.cancelTimer(_updateTokenInfoId);
        };
        let _harvestAmount =
_harvest(Principal.fromText("aaaaa-aa"));
        _lastRewardTime := now;
        _startTime := params.startTime;
        _bonusEndTime := params.bonusEndTime;
        _rewardPerTime := params.rewardPerTime;
```

[Staking pool-08] The stop function does not check the time

Severity Level	Low
Type	Business Security
Lines	StakingPool.mo#L90-97
Description	The `stop` function does not validate the input parameters, which may result in extending the reward period if `_getTime` is greater than `_bonusEndTime`. and may result in insufficient contract reward tokens.

```
    public shared (msg) func stop() : async
Result.Result<Types.PublicStakingPoolInfo, Text> {
        _checkAdminPermission(msg.caller);

        _bonusEndTime := _getTime();
        Timer.cancelTimer(_updateTokenInfoId);
        return _getPoolInfo();
    };
```

Recommendation	It is recommended to add time parameter validation in the function.
Status	**Acknowledged.**

[Staking pool-09] Reward calculation defect

Severity Level	Low
Type	Business Security
Lines	StakingPool.mo#L916-934

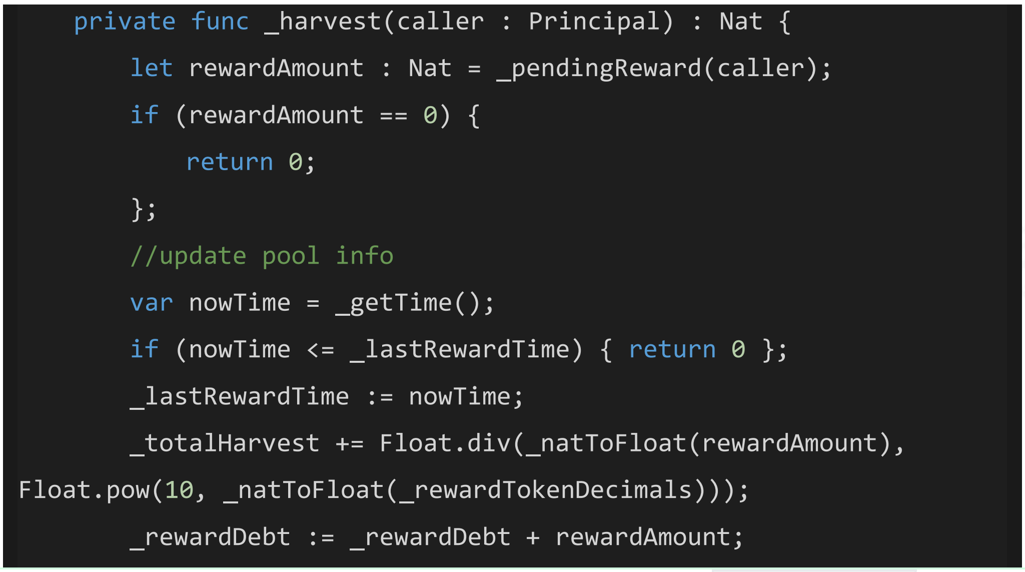
Description

Since the global reward update occurs after the user's status update, reward calculations may be incorrect in certain situations. For example, in the following situation:

User A stakes at 100 seconds, and then User B stakes at 200 seconds. When processing the next transaction, the reward settlement for User A is handled (both at 200 seconds but with User B's transaction occurring before User A's). This results in the `_lastRewardTime` being updated by User B. Consequently, when determining the reward, the following condition is processed in `_harvest:`

`if (nowTime <= _lastRewardTime) { return 0; }`

Since the processing is not separated, the global variable affects User A's reward amount, resulting in it being zero and returning directly. However, User A actually has a staking reward for 100 seconds. When User A then performs an unstake, the `rewardDebt` is modified directly, causing the reward to be nonexistent.

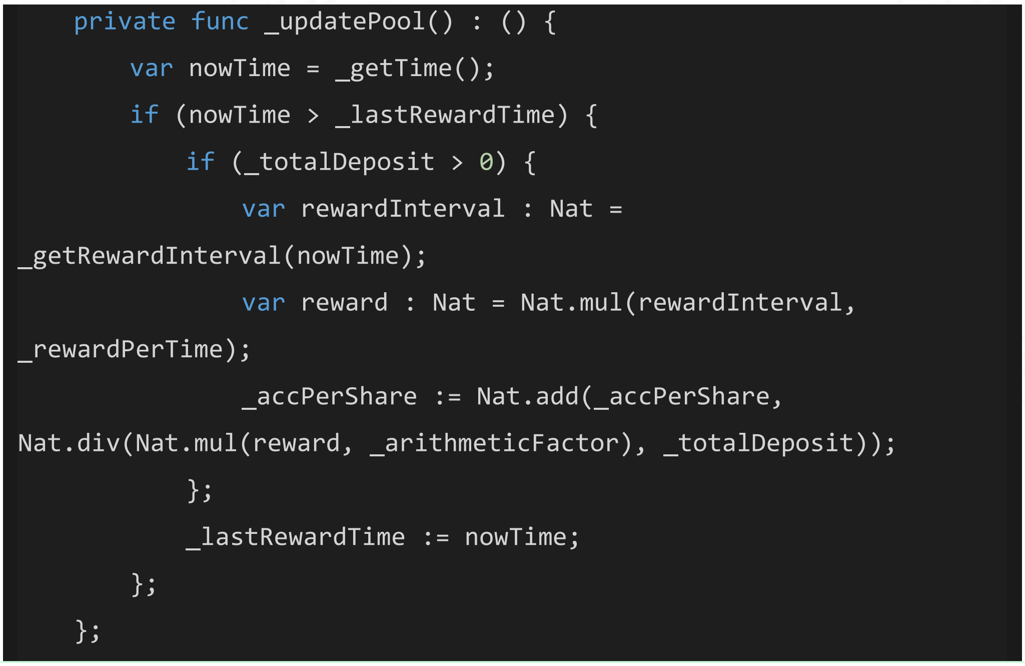


```
    private func _harvest(caller : Principal) : Nat {
        let rewardAmount : Nat = _pendingReward(caller);
        if (rewardAmount == 0) {
            return 0;
        };
        //update pool info
        var nowTime = _getTime();
        if (nowTime <= _lastRewardTime) { return 0 };
        _lastRewardTime := nowTime;
        _totalHarvest += Float.div(_natToFloat(rewardAmount),
Float.pow(10, _natToFloat(_rewardTokenDecimals)));
        _rewardDebt := _rewardDebt + rewardAmount;
```

Recommendation

It is recommended to synchronize the settlement of `rewardAmount` and the update of `_totalRewardFee`. Before settling user rewards, the global reward data should be updated first: `_accPerShare` and `_lastRewardTime`. After that, the user reward data should be updated.

Status **Fixed.**



```
    private func _updatePool() : () {
        var nowTime = _getTime();
        if (nowTime > _lastRewardTime) {
            if (_totalDeposit > 0) {
                var rewardInterval : Nat =
_getRewardInterval(nowTime);
                var reward : Nat = Nat.mul(rewardInterval,
_rewardPerTime);
                _accPerShare := Nat.add(_accPerShare,
Nat.div(Nat.mul(reward, _arithmeticFactor), _totalDeposit));
            };
            _lastRewardTime := nowTime;
        };
    };
```

[Staking pool-10] The calculation of the total fee amount is not synchronized

Severity Level	Low
Type	Business Security
Lines	StakingPool.mo#L986-999

Description

As shown below, according to the code logic, the reward fee is charged in all cases. However, if the remaining reward amount is too low and less than the `rewardTokenFee`, the `totalRewardFee` will not be updated. This is clearly unreasonable because when `rewardAmount < rewardTokenFee`, the user's reward will be deducted, but the `totalRewardFee` will not increase.

```
    private func _pendingReward(user : Principal) : Nat {
        _updatePool();
        var userInfo : Types.UserInfo = _getUserInfo(user);
        var rewardAmount =
Nat.sub(Nat.div(Nat.mul(userInfo.stakeAmount, _accPerShare),
_arithmeticFactor), userInfo.rewardDebt);
        let rewardFee : Nat = Nat.div(Nat.mul(rewardAmount,
initArgs.rewardFee), 1000);
        if (rewardFee > 0) {
            rewardAmount := rewardAmount - rewardFee;
        };
        if (rewardAmount > 0 and rewardFee > 0 and rewardAmount >=
_rewardTokenFee) {
            _totalRewardFee += rewardFee;
        };

        return rewardAmount;
    };
```

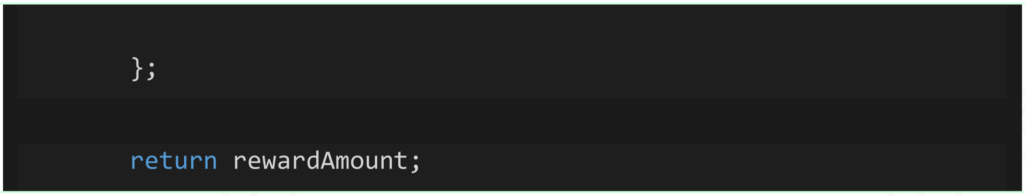
Recommendation

It is recommended to synchronize the settlement of `rewardAmount` and the update of `_totalRewardFee`.

Status

Fixed.

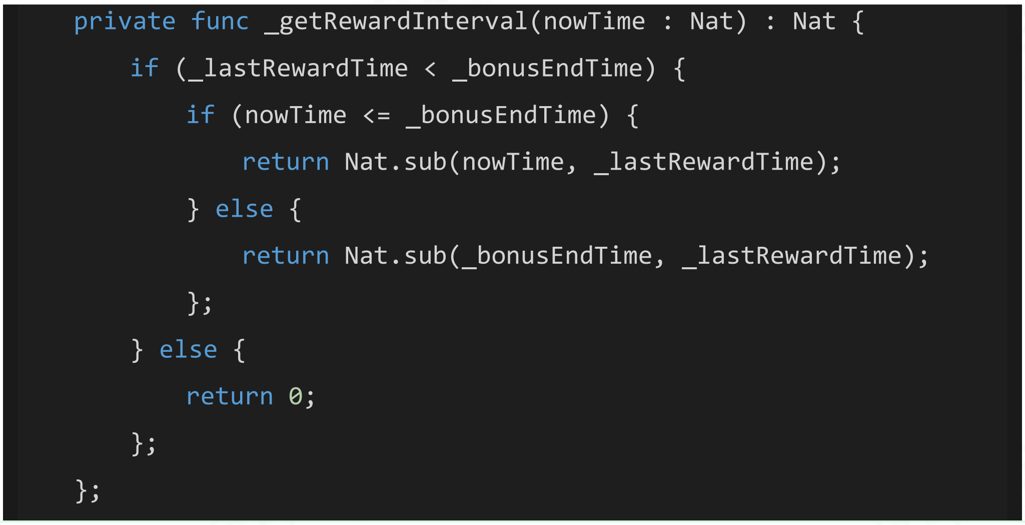
```
        let rewardFee : Nat = Nat.div(Nat.mul(rewardAmount,
initArgs.rewardFee), 1000);
        if (rewardFee > 0) {
            rewardAmount := rewardAmount - rewardFee;
            _totalRewardFee += rewardFee;
```



```
        };

        return rewardAmount;
```

[Staking pool-11] The _getRewardInterval function does not check the start time

<table>
<tr><td><strong>Severity Level</strong></td><td>Low</td></tr>
<tr><td><strong>Type</strong></td><td>Business Security</td></tr>
<tr><td><strong>Lines</strong></td><td>StakingPool.mo#L1013-1024</td></tr>
<tr><td><strong>Description</strong></td><td>The function should include a check to ensure the start time is less than the current time to prevent rewards from being generated when the current time is earlier than the start time.



<pre><code>private func _getRewardInterval(nowTime : Nat) : Nat {
    if (_lastRewardTime < _bonusEndTime) {
        if (nowTime <= _bonusEndTime) {
            return Nat.sub(nowTime, _lastRewardTime);
        } else {
            return Nat.sub(_bonusEndTime, _lastRewardTime);
        };
    } else {
        return 0;
    };
};</code></pre></td></tr>
<tr><td><strong>Recommendation</strong></td><td>It is recommended to add a start time check in the <code>_getRewardInterval</code> function.</td></tr>
<tr><td><strong>Status</strong></td><td><strong>Acknowledged.</strong></td></tr>
</table>

[Staking pool-12] The return value of getInitArgs is incomplete

Severity Level Info

Type Coding Conventions

Lines StakingPool.mo#L154-156

Description The return value of the getInitArgs function is missing the feeReceiverCid.

```
shared (initMsg) actor class StakingPoolController(
    feeReceiverCid : Principal,
    governanceCid : ?Principal,
) = this {
    public query func getInitArgs() : async
Result.Result<{ governanceCid : ?Principal }, Types.Error> {
        #ok({ governanceCid = governanceCid });
    };
```

Recommendation It is recommended to add the return value.

Status **Fixed.**

```
    public query func getInitArgs() : async
Result.Result<{ feeReceiverCid : Principal;
governanceCid : ?Principal }, Types.Error> {
        #ok({ feeReceiverCid = feeReceiverCid; governanceCid =
governanceCid });
    };
```

[Staking pool-13] Redundant code

Severity Level Info

Type Coding Conventions

Lines StakingPool.mo#L63-64

Description The following code is not used in the actor and can be considered for removal.The `stakingBalance` and `rewardBalance` fields in `_ledgerAmount` are unused.

```
    private stable var _ledgerAmount = {
        var claim = 0.00;
        var staking = 0.00;
        var unStaking = 0.00;
        var stakingBalance = 0.00;
        var rewardBalance = 0.00;
    };
```

Recommendation Based on business needs, choose whether need to delete them.

Status **Fixed.**

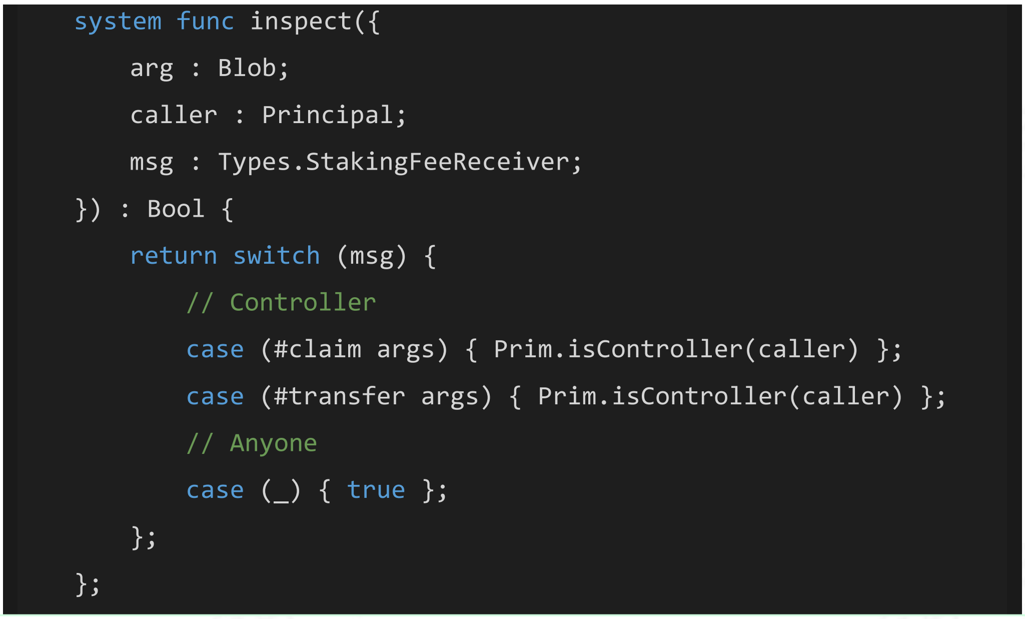
```
    public type UserInfo = {
        var stakeTokenBalance : Nat;
        var rewardTokenBalance : Nat;
        var stakeAmount : Nat;
        var rewardDebt : Nat;
        var lastStakeTime : Nat;
        var lastRewardTime : Nat;
    };
```

[Staking pool-14] Lack of permission check

Severity Level	Info
Type	Coding Conventions
Lines	StakingFeeReceiver.mo#L93-105

Description

According to the code guidelines, it is recommended to add a permission check for transferAll in the system function inspect.



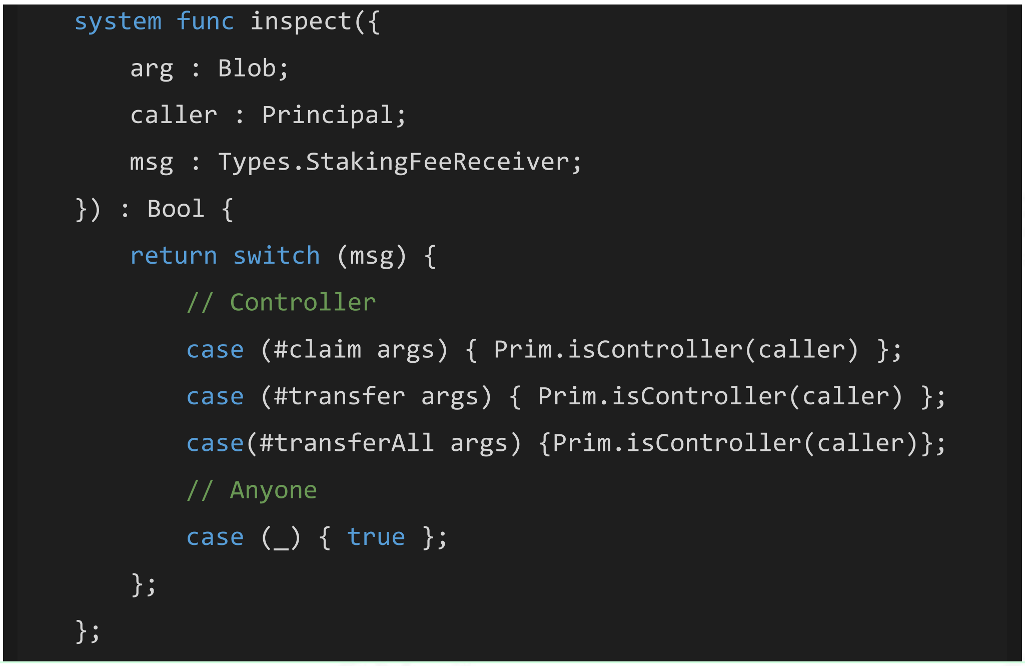
```
system func inspect({
    arg : Blob;
    caller : Principal;
    msg : Types.StakingFeeReceiver;
}) : Bool {
    return switch (msg) {
        // Controller
        case (#claim args) { Prim.isController(caller) };
        case (#transfer args) { Prim.isController(caller) };
        // Anyone
        case (_) { true };
    };
};
```

Recommendation

It is recommended that the project team to following the code guidelines.

Status

Fixed.



```
system func inspect({
    arg : Blob;
    caller : Principal;
    msg : Types.StakingFeeReceiver;
}) : Bool {
    return switch (msg) {
        // Controller
        case (#claim args) { Prim.isController(caller) };
        case (#transfer args) { Prim.isController(caller) };
        case(#transferAll args) {Prim.isController(caller)};
        // Anyone
        case (_) { true };
    };
};
```

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.3 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.4 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		assert Usage
		Cycles Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		Returned Value Security
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com