BEOSIN
Blockchain Security

ICPSwap-Farm

Smart Contract Security Audit

No. 202408011145

Aug 1st, 2024

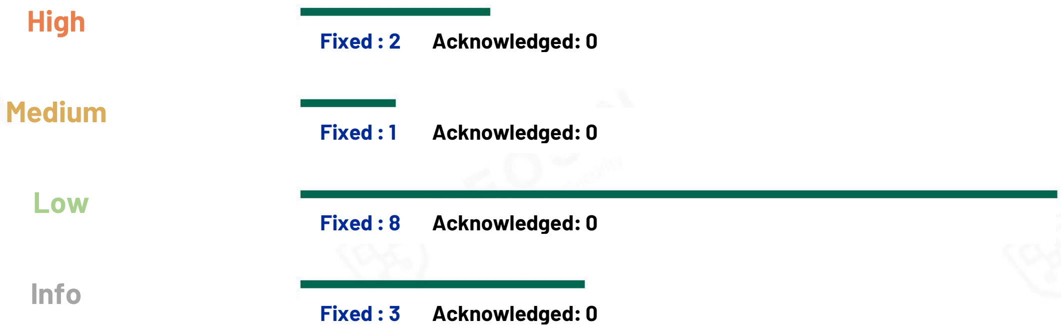
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing,2 High-risk, 1 Medium-risk ,8 Low-risk and 3 Info items were identified in the ICPSwap-Farm 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-Farm implements the function of staking to obtain rewards, including five functional modules: FarmFactory, Farm, FarmFeeReceiver, FarmFeeReceiver,FarmIndex. Below is a separate explanation of the business logic for each module.

FarmFactory：

This actor is mainly responsible for creating a reward farm. The admin can call the `create` function to create a valid reward farm. The validity verification mainly includes: the reward amount `rewardAmount` must be a positive number, the validity of the farm start time `startTime` and end time `endTime`, and the farm duration's validity, the validity of the reward distribution cycle `secondPerCycle` relative to the duration.

Farm:

This actor mainly includes the following three parts of business functions, which will be introduced separately below.

The first part is the pledge-related logic: users can call the `stake` function to pledge the `_swapPoolAct` related liquidity voucher tokens specified in the farm to obtain reward tokens. Since the voucher position contains the price range, if the farm has turned on the price limit, then only Only positions within the valid price range can be pledged, and qualified positions will be transferred to the farm for pledge; later, the user can call the `unstake` function to unstake, and the contract will return the user's pledged tokens and Issue the user's staking rewards; in addition, `feeReceiverCid` can call `withdrawRewardFee` to obtain the staking reward handling fee.

The second part is the logic related to reward calculation: each farm sets a timer, and regularly calls the `_distributeReward` function to update the user's reward in each time period. The total reward amount of the farm in each cycle is fixed. At each reward update, the actor will calculate the overall reward weight based on whether the price limit is turned on and the liquidity of all users' pledged positions, and then based on the liquidity of the user's single pledged position. and time to calculate its reward amount and update it.

The third part is the status management of the farm. There are t four statuses of the farm, namely `#NOT_STARTED,#LIVE, #CLOSED, and #FINISHED`. When there is no user staking in the farm, the admin can call the close function to set the farm status to #CLOSED, and recycle the remaining unallocated reward tokens to the refunder address; the admin can also call the finishManually function to set the farm status to #FINISHED.; the admin can also call the restartManually function to reset the farm status to #LIVE, at which time the user can continue staking, etc.

FarmFactoryValidator：

It mainly implements the parameter validity check for Farm creation, and the caller permission check when setting permissions for Admins, FarmAdmins, FarmControllers, etc. The relevant implementation has not yet been used in the project.

FarmFeeReceiver：

This actor is responsible for the management of pledge fees and is mainly divided into two parts of business logic functions. All logic can only be called by addresses with Controller permissions. The first part is that the Controller can call the claim function to withdraw the pledge fees accumulated in the farm. The second part is that the Controller can call the transfer and transferAll functions to transfer the fee tokens to other addresses. The token standards supported by this actor include DIP20, DIP20-WICP, DIP20-XTC, EXT, ICRC1, ICRC2, ICRC3, and ICP.

Code module update

In the commit with hash 05494329812b95408ca1a71111cdce6d94e4c229, the project team made the following changes to the code.

Farm: Added information on reward tokens. Used the _updateTVL and _updateAPR functions to calculate and update the TVL and APR of assets in the farm. Utilized farmIndex to manage data synchronization for users participating in the farm, as well as farm status and TVL.

Factory: Removed the functionality for synchronizing and updating the Farm data status. The primary logic now focuses on creating Farms.

FarmFactoryValidator: Removed the address check for rewardPool.

FarmIndex: Added this actor for managing data related to users and Farms. Multiple Farms will periodically call _updateUserInfo, _updateStatus, and _updateTVL to synchronize data with FarmIndex. Users can query information about the entire project in FarmIndex.

1 Overview

1.1 Project Overview

Project Name	ICPswap-Farm
Project Language	Motoko
Platform	IC
Code Base	https://github.com/ICPSwap-Labs/icpswap-farm
Commit Id	d76bf2d55ca48f1b1016c2d553f3f30ca9d1a659 2b89ee1905e105ac055389332e85de2a82f5c499 d562ae0d8c78314b5d4396eb47b60ec440824461 9f33f69e773ae82936a63dd2a626a2fdc6a0860c 0a0a78fb0650e7cee6f780d104f8c9170f979f51 c45af75007396da674ee1c780b2a3fbea11db8b8 5121eb23fdc3f3ee3ff7e0b9bb953103e6fd0348 63ef937f3e1738f65cb025083b3b6f9f5b4105a4 05494329812b95408ca1a71111cdce6d94e4c229

1.2 Audit Overview

Audit work duration: May 10, 2024 – May 22, 2024, Jun 19, 2024 - Jun 25, 2024

Update time: Aug 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

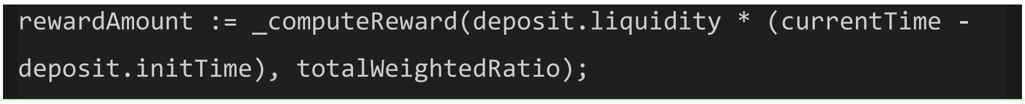
Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems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

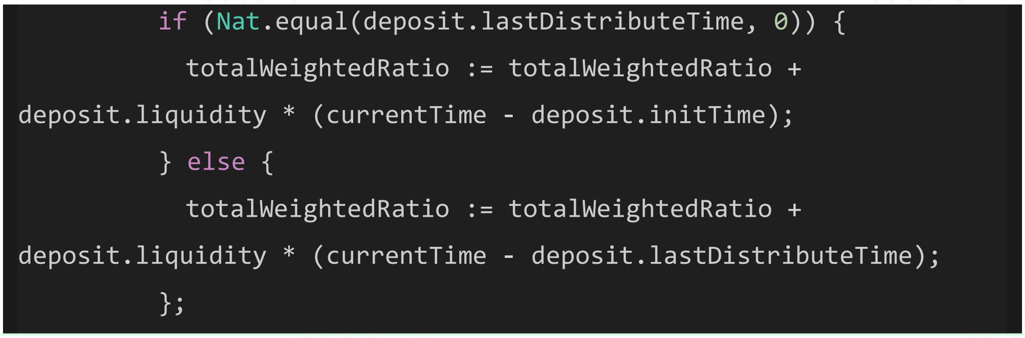
2 Findings

Index	Risk description	Severity level	Status
Farm-01	The reward calculation method is unreasonable	High	Fixed
Farm-02	The amount of tokens extracted by the close function is inaccurate	High	Fixed
Farm-03	Value sets incorrectly	Medium	Fixed
Farm-04	The amount of reward tokens in TVL is not updated in a timely manner	Low	Fixed
Farm-05	The _rewardPerCycle calculate inaccurate	Low	Fixed
Farm-06	The _stakeRecordBuffe record calculate inaccurate	Low	Fixed
Farm-07	The _updateTVL calculate inaccurate	Low	Fixed
Farm-08	Any user can record incorrect data	Low	Fixed
Farm-09	Variable name set incorrect	Low	Fixed
Farm-10	Error in transfer parameters for rewardTokenAdapter	Low	Fixed
Farm-11	Design flaw in the getLiveFarmsByPools function	Low	Fixed
Farm-12	Dependent library version specification error	Info	Fixed
Farm-13	Permission check incomplete	Info	Fixed
Farm-14	Redundant code	Info	Fixed

Finding Details:

[Farm-01] The reward calculation method is unreasonable

Severity Level	High
Type	Business Security
Lines	Farm.mo #L924, 940
Description	In the initial audited code, the formula used to calculate user staking rewards is: deposit.liquidity * (currentTime - deposit.initTime), where deposit.initTime represents the user's staking start time. This leads to an unreasonable situation: even if two users have the same liquidity and the same staking duration within the current period, their rewards for this period will differ due to the different staking start times.  `rewardAmount := _computeReward(deposit.liquidity * (currentTime - deposit.initTime), totalWeightedRatio);`
Recommendation	Update the user's deposit.initTime each time rewards are calculated. Specifically, after updating the rewards, also update the initTime, so that the next reward calculation uses the new initTime.

Status	**Fixed.** This issue has been fixed in commit 3bc1c9b.
	
	`if (Nat.equal(deposit.lastDistributeTime, 0)) {`
	`totalWeightedRatio := totalWeightedRatio + deposit.liquidity * (currentTime - deposit.initTime);`
	`} else {`
	`totalWeightedRatio := totalWeightedRatio + deposit.liquidity * (currentTime - deposit.lastDistributeTime);`
	`};`

[Farm-02] The amount of tokens extracted by the close function is inaccurate

Severity Level	High
Type	Business Security
Lines	Farm.mo #L455,464

Description

The close function is intended to collect unallocated rewards within the actor. Therefore, there are two issues with the close function:

(1) If all users have claimed their rewards, then balance and _totalRewardBalance should be equal, causing the balance check to always fail.

```
if (balance < _totalRewardBalance) {
    _errorLogBuffer.add("InsufficientFunds. balance: " # debug_show
(balance) # " totalRewardBalance: " # debug_show (_totalRewardBalance)
# " . nowTime: " # debug_show (nowTime));
};
```

(2) If there are users who have not claimed their rewards, then balance will be greater than _totalRewardBalance. In this case, the amount that should actually be withdrawn is balance, not _totalRewardBalance..

```
var amount = balance - _rewardTokenFee;
```

Recommendation

It is recommended to add a check in the close function to ensure that all user rewards have been claimed. Additionally, the amount withdrawn in this function should directly be the actor's entire balance.

Status

Fixed. This issue has been fixed in commit 5121eb2. The premise for calling this function is that all users have received rewards, and the amount received this time is the actor's balance.

[Farm-03] Variable value sets incorrectly

<table>
<tr><td><b>Severity Level</b></td><td><b>Medium</b></td></tr>
<tr><td><b>Type</b></td><td>Business Security</td></tr>
<tr><td><b>Lines</b></td><td>Farm.mo#L80</td></tr>
<tr><td><b>Description</b></td><td>The value of _token1AmountLimit is being set to `initArgs.token0AmountLimit` here, which should be set to `initArgs.token1AmountLimit`.
<pre><code> private stable var _positionNumLimit : Nat = 500;
  private stable var _token0AmountLimit : Nat =
initArgs.token0AmountLimit;
  private stable var _token1AmountLimit : Nat =
initArgs.token0AmountLimit;
  private stable var _priceInsideLimit : Bool =
initArgs.priceInsideLimit;
</code></pre></td></tr>
<tr><td><b>Recommendation</b></td><td>It is recommended to replace `initArgs.token0AmountLimit` with `initArgs.token1AmountLimit`.</td></tr>
<tr><td><b>Status</b></td><td><b>Fixed.</b></td></tr>
</table>

[Farm-04] The amount of reward tokens in TVL is not updated in a timely manner

Severity Level	Low
Type	Business Security
Lines	Farm.mo
Description	There are many unreasonable updates of TVL in Farm actor: (1) The TVL (Total Value Locked) is updated exclusively within the _distributeReward function, which is called only once per cycle. Consequently, the recorded TVL may not be accurate, as users can stake and withdraw at any time. (2) The rewardToken.amount within the TVL is not being updated and remains constant at initArgs.totalReward. This is clearly unreasonable.
Recommendation	Regarding issue (1), we believe no changes are necessary for now, as the cycle specified by the FarmFactory actor ranges from half an hour to 12 hours, which should be acceptable. If this is not acceptable, we recommend updating the TVL with every stake and unstake operation. However, this would require iterating through all deposits, potentially consuming a significant amount of cycles. For question (2), we recommend that users update TVL each time they claim rewards. This should also include admin claiming any rewards.
Status	**Fixed**.

[Farm-05] The _rewardPerCycle calculate inaccurate

Severity Level	Low
Type	Business Security
Lines	Farm.mo#L117-119

Description

In the `init` function, an extra 1 is being added to the cumulative `tempRewardTotalCount` used in the calculation of `_rewardPerCycle`. This results in a lower reward per cycle than intended.

```
  _canisterId := ?Principal.fromActor(this);
  var tempRewardTotalCount = SafeUint.Uint512(initArgs.endTime).sub(SafeUint.Uint512(initArgs.startTime)).div(SafeUint.Uint512(initArgs.secondPerCycle)).add(SafeUint.Uint512(1));
  _totalCycleCount := tempRewardTotalCount.val();
  _rewardPerCycle := SafeUint.Uint512(_totalReward).div(tempRewardTotalCount).val();

  let rewardPoolMetadata = switch (await _rewardPoolAct.metadata()) {
    case (#ok(poolMetadata)) { poolMetadata };
    case (#err(code)) {
      {
        key = "";
        token0 = { address = ""; standard = "" };
        token1 = { address = ""; standard = "" };
        fee = 0;
        tick = 0;
        liquidity = 0;
        sqrtPriceX96 = 0;
        maxLiquidityPerTick = 0;
      };
    };
  };
```

Recommendation

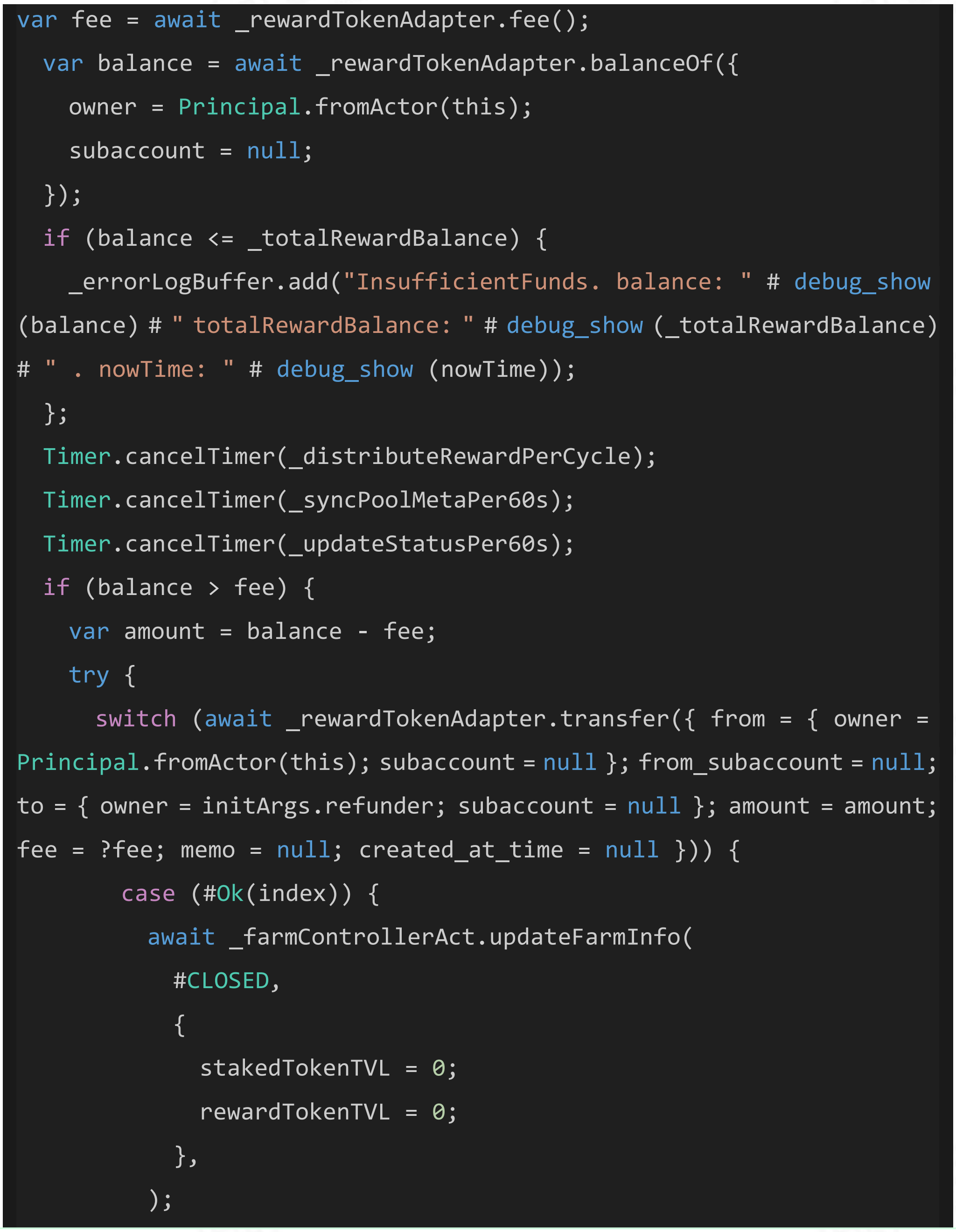
It is recommended that the project team to review and confirm the business logic.

Status

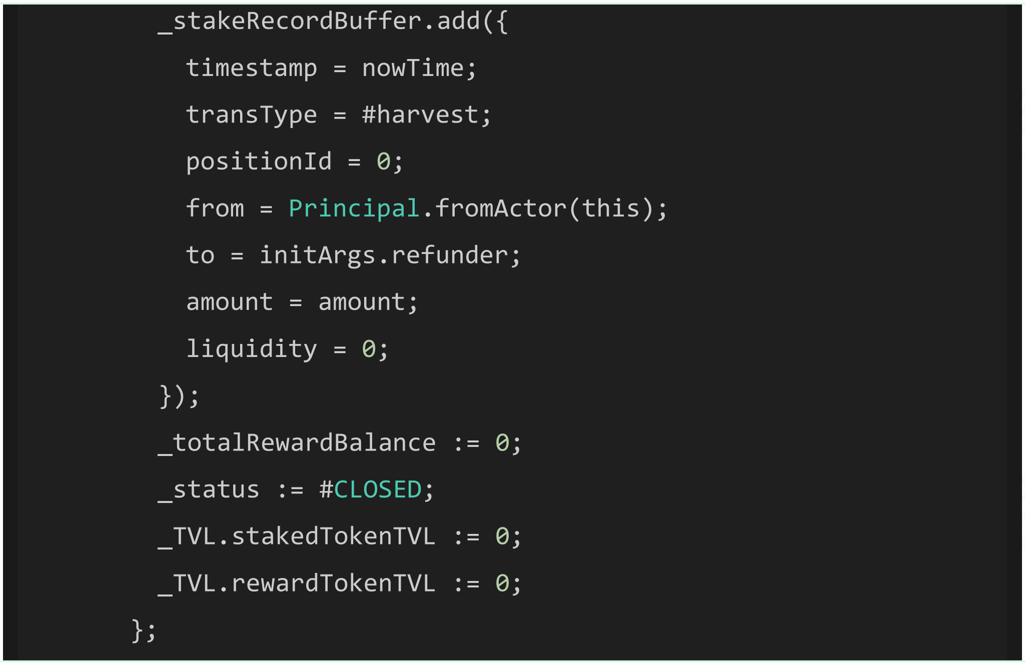
Fixed. The project team removed the addition of 1 when calculating the `_rewardPerCycle`.

[Farm-06] The _stakeRecordBuffe record calculate inaccurate

Severity Level	Low
Type	Business Security
Lines	Farm.mo#L434-472
Description	When there is no staked position, the admin can call the close function to withdraw the remaining undistributed reward tokens to the `refunder` address. However, the Actor mistakenly recorded the amount in the `_stakeRecordBuffer` as the reward quantity after deducting the token transfer fees, resulting in a lower recorded amount than the actual quantity.



```
var fee = await _rewardTokenAdapter.fee();
  var balance = await _rewardTokenAdapter.balanceOf({
    owner = Principal.fromActor(this);
    subaccount = null;
  });
  if (balance <= _totalRewardBalance) {
    _errorLogBuffer.add("InsufficientFunds. balance: " # debug_show
(balance) # " totalRewardBalance: " # debug_show (_totalRewardBalance)
# " . nowTime: " # debug_show (nowTime));
  };
  Timer.cancelTimer(_distributeRewardPerCycle);
  Timer.cancelTimer(_syncPoolMetaPer60s);
  Timer.cancelTimer(_updateStatusPer60s);
  if (balance > fee) {
    var amount = balance - fee;
    try {
      switch (await _rewardTokenAdapter.transfer({ from = { owner =
Principal.fromActor(this); subaccount = null }; from_subaccount = null;
to = { owner = initArgs.refunder; subaccount = null }; amount = amount;
fee = ?fee; memo = null; created_at_time = null })) {
        case (#Ok(index)) {
          await _farmControllerAct.updateFarmInfo(
            #CLOSED,
            {
              stakedTokenTVL = 0;
              rewardTokenTVL = 0;
            },
          );
```



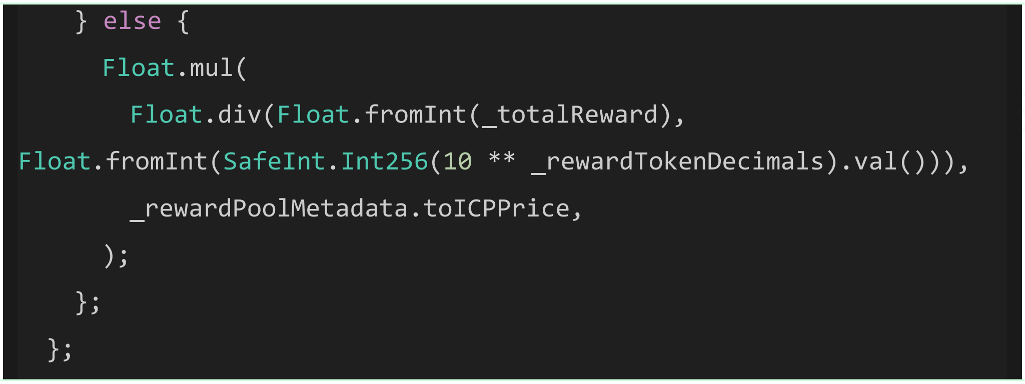
```
  _stakeRecordBuffer.add({
    timestamp = nowTime;
    transType = #harvest;
    positionId = 0;
    from = Principal.fromActor(this);
    to = initArgs.refunder;
    amount = amount;
    liquidity = 0;
  });
  _totalRewardBalance := 0;
  _status := #CLOSED;
  _TVL.stakedTokenTVL := 0;
  _TVL.rewardTokenTVL := 0;
};
```

Recommendation	It is recommended that the project team to replace amount with balance.
Status	**Fixed.** The project team replaces amount with balance.

[Farm-07] The _updateTVL calculate inaccurate

Severity Level	Low
Type	Business Security
Lines	Farm.mo#L960-986
Description	The function `_updateTVL` calculates the TVL (Total Value Locked) of the current reward tokens, referred to as `rewardTokenTVL`, using the `_totalReward` value. However, it's important to note that `_totalReward` represents the total reward quantity for the Farm and remains unchanged throughout the reward distribution process. As a result, the calculated `rewardTokenTVL` may be overestimated and remain constant, potentially deviating from the actual value.

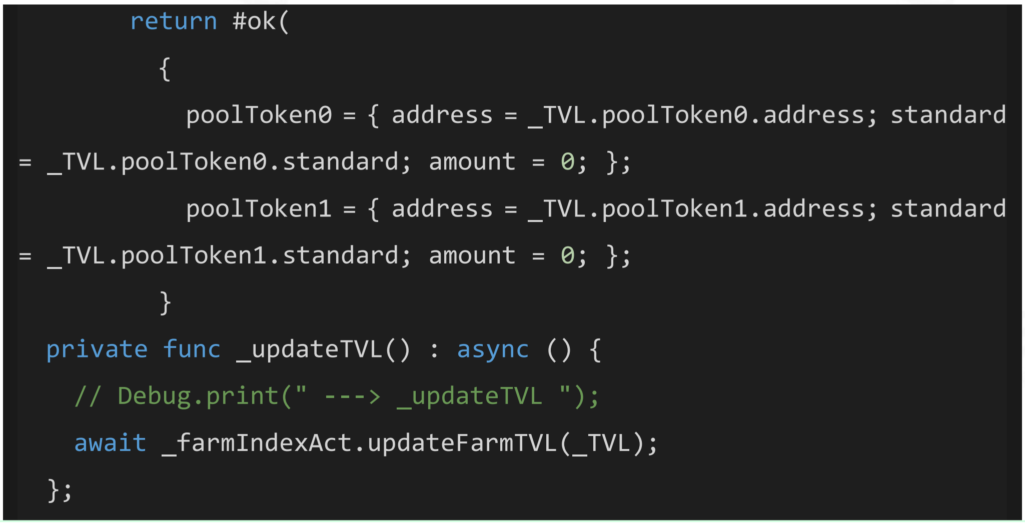
```
private func _updateTVL() {
    _TVL.stakedTokenTVL := if (_poolZeroForOne) {
      Float.add(
        Float.mul(
          Float.div(Float.fromInt(_poolToken0Amount),
Float.fromInt(SafeInt.Int256(10 ** _poolToken0Decimals).val())),
          _poolMetadata.toICPPrice,
        ),
        Float.div(Float.fromInt(_poolToken1Amount),
Float.fromInt(SafeInt.Int256(10 ** _poolToken1Decimals).val())),
      );
    } else {
      Float.add(
        Float.mul(
          Float.div(Float.fromInt(_poolToken1Amount),
Float.fromInt(SafeInt.Int256(10 ** _poolToken1Decimals).val())),
          _poolMetadata.toICPPrice,
        ),
        Float.div(Float.fromInt(_poolToken0Amount),
Float.fromInt(SafeInt.Int256(10 ** _poolToken0Decimals).val())),
      );
    };
    _TVL.rewardTokenTVL := if (Text.equal(initArgs.ICP.address,
initArgs.rewardToken.address)) {
      Float.div(Float.fromInt(_totalReward),
Float.fromInt(SafeInt.Int256(10 ** _rewardTokenDecimals).val()));
```



```
    } else {
      Float.mul(
        Float.div(Float.fromInt(_totalReward),
Float.fromInt(SafeInt.Int256(10 ** _rewardTokenDecimals).val())),
        _rewardPoolMetadata.toICPPrice,
      );
    };
  };
```

Recommendation It is recommended that the project team to replace _totalReward with _totalRewardBalance.

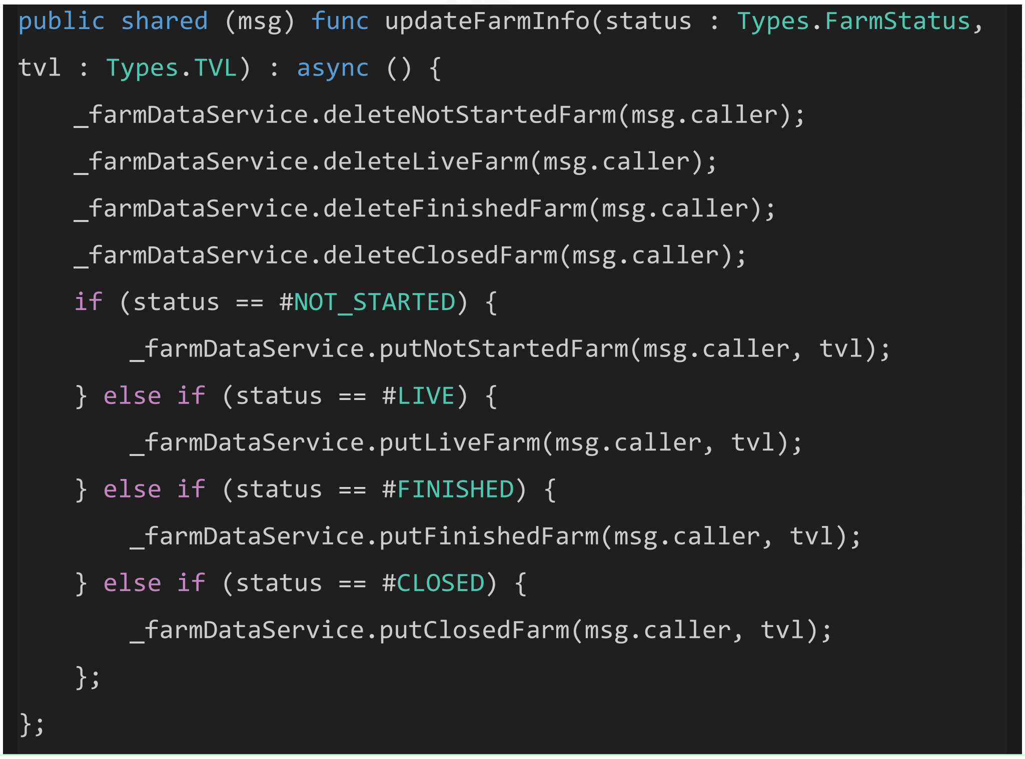
Status **Fixed.** Currently, the project team updates the TVL when modifying asset balances and synchronizes this data with the FarmIndex.



```
        return #ok(
          {
            poolToken0 = { address = _TVL.poolToken0.address; standard
= _TVL.poolToken0.standard; amount = 0; };
            poolToken1 = { address = _TVL.poolToken1.address; standard
= _TVL.poolToken1.standard; amount = 0; };
          }
  private func _updateTVL() : async () {
    // Debug.print(" ---> _updateTVL ");
    await _farmIndexAct.updateFarmTVL(_TVL);
  };
```

[Farm-08] Any user can record incorrect data

Severity Level	Low
Type	General Vulnerability
Lines	FarmController.mo#L134-148
Description	updateFarmInfo in the FarmController actor does not have any calling restrictions, that is, anyone can call this function to record wrong Farm and TVL data, which will not affect the normal Farm, but will affect data query.



```
public shared (msg) func updateFarmInfo(status : Types.FarmStatus,
tvl : Types.TVL) : async () {
    _farmDataService.deleteNotStartedFarm(msg.caller);
    _farmDataService.deleteLiveFarm(msg.caller);
    _farmDataService.deleteFinishedFarm(msg.caller);
    _farmDataService.deleteClosedFarm(msg.caller);
    if (status == #NOT_STARTED) {
        _farmDataService.putNotStartedFarm(msg.caller, tvl);
    } else if (status == #LIVE) {
        _farmDataService.putLiveFarm(msg.caller, tvl);
    } else if (status == #FINISHED) {
        _farmDataService.putFinishedFarm(msg.caller, tvl);
    } else if (status == #CLOSED) {
        _farmDataService.putClosedFarm(msg.caller, tvl);
    };
};
```

Recommendation	It is recommended that the project team that the caller of this function should be restricted to the farm actor created through the create function.
Status	**Fixed.** The project team added a caller's permission check.

[Farm-09] Variable name set incorrect

Severity Level Low

Type Business Security

Lines Farm.mo #L1004-1020

Description In `_computeReward`, rate is expanded by 10e8, but the variable name means it is expanded by 10e9. Of course, this is just an expansion of accuracy and will basically not affect the calculation results.

```
private func _computeReward(weightedRatio : Nat, totalWeightedRatio : Nat) : Nat {
  var excessDecimal = SafeUint.Uint512(100000000);
  var weightedRatioXe9 = SafeUint.Uint512(weightedRatio).mul(excessDecimal);
  // Debug.print("weightedRatioXe9: " # debug_show (weightedRatioXe9.val()));
  var rate = if (totalWeightedRatio == 0) { SafeUint.Uint512(0) } else {
    weightedRatioXe9.div(SafeUint.Uint512(totalWeightedRatio));
  };
  // Debug.print("rate: " # debug_show (rate.val()));

  var reward = SafeUint.Uint512(_rewardPerCycle).mul(rate).div(excessDecimal).val();
  // Debug.print("reward: " # debug_show (reward));
  _totalRewardUnharvested := _totalRewardUnharvested + reward;
  _totalRewardBalance := _totalRewardBalance - reward;
  return reward;
};
```

Recommendation It is recommended that the project team to review and confirm the business logic.

Status **Fixed.** The project team replaces `weightedRatioXe9` with `weightedRatioXe8`.

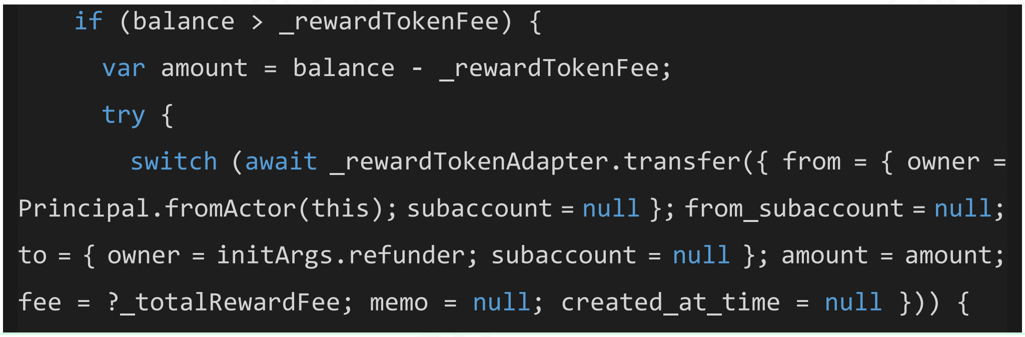
[Farm-10] Error in transfer parameters for rewardTokenAdapter

Severity Level Low

Type Business Security

Lines Farm.mo #L555-558

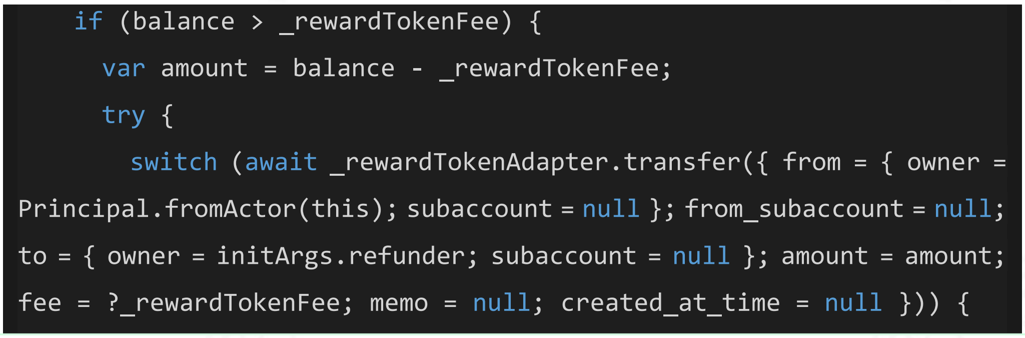
Description In the `close` function, the `rewardTokenAdapter` uses the value of `_totalRewardFee` as the fee for the transfer. If the `rewardTokenAdapter` has a fee that is actually different from `_totalRewardFee`, the transfer will fail. Consequently, when the Farm is marked as closed, the reward tokens will remain in the Farm.



```
    if (balance > _rewardTokenFee) {
      var amount = balance - _rewardTokenFee;
      try {
        switch (await _rewardTokenAdapter.transfer({ from = { owner =
Principal.fromActor(this); subaccount = null }; from_subaccount = null;
to = { owner = initArgs.refunder; subaccount = null }; amount = amount;
fee = ?_totalRewardFee; memo = null; created_at_time = null })) {
```

Recommendation It is recommended to modify the transfer parameter of `rewardTokenAdapter` to `_rewardTokenFee`.

Status Fixed



```
    if (balance > _rewardTokenFee) {
      var amount = balance - _rewardTokenFee;
      try {
        switch (await _rewardTokenAdapter.transfer({ from = { owner =
Principal.fromActor(this); subaccount = null }; from_subaccount = null;
to = { owner = initArgs.refunder; subaccount = null }; amount = amount;
fee = ?_rewardTokenFee; memo = null; created_at_time = null })) {
```

[Farm-11] Design flaw in the getLiveFarmsByPools function

Severity Level Low

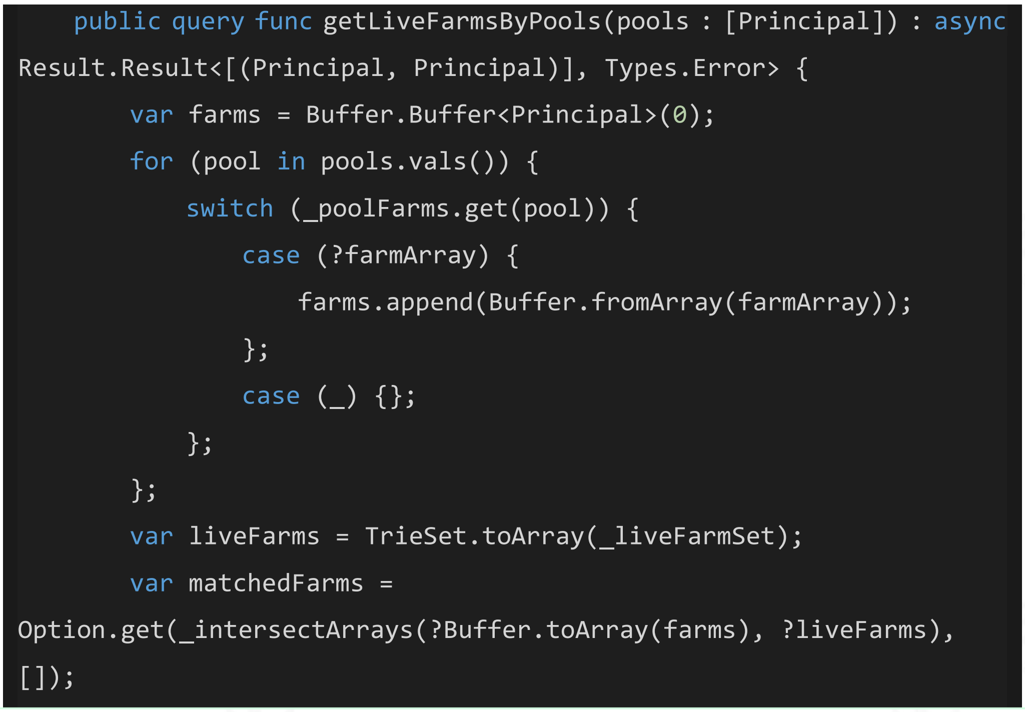
Type Business Security

Lines FarmIndex.mo #L239-262

Description

The _intersectArrays function has a design flaw: when one array has values and the other is null, it takes the non-null array as the result. This does not align with the concept of an intersection, which should result in null. Otherwise, functions like getLiveFarmsByPools will produce inaccurate intersection results in such cases.

The getLiveFarmsByPools function is supposed to correctly compute the intersection of arrays. If one of the arrays is empty, the result will not be as expected.

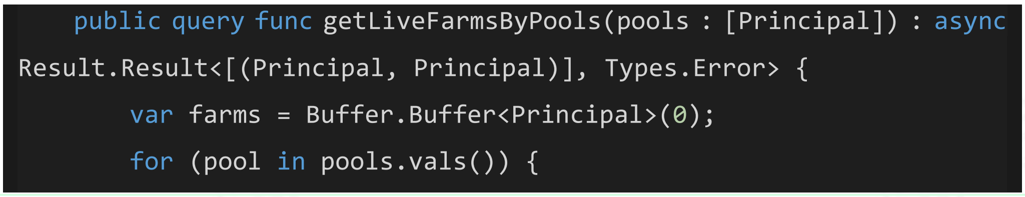


```
    public query func getLiveFarmsByPools(pools : [Principal]) : async
Result.Result<[(Principal, Principal)], Types.Error> {
        var farms = Buffer.Buffer<Principal>(0);
        for (pool in pools.vals()) {
            switch (_poolFarms.get(pool)) {
                case (?farmArray) {
                    farms.append(Buffer.fromArray(farmArray));
                };
                case (_) {};
            };
        };
        var liveFarms = TrieSet.toArray(_liveFarmSet);
        var matchedFarms =
Option.get(_intersectArrays(?Buffer.toArray(farms), ?liveFarms),
[]);
```

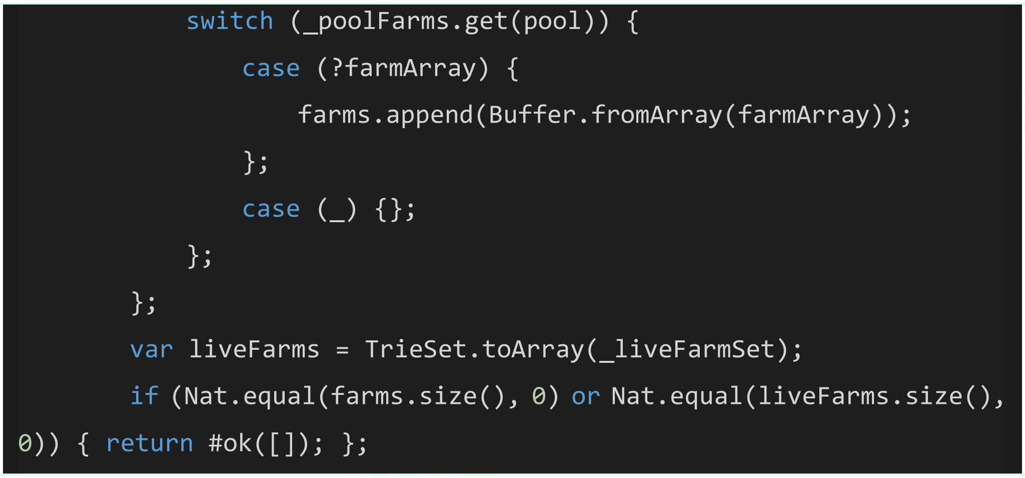
Recommendation

It is recommended that the getLiveFarmsByPools function checks if liveFarms and farms are both not null before calling _intersectArrays. If either is null, return an empty array. Then proceed to call the _intersectArrays function for comparison.

Status **Fixed.**



```
    public query func getLiveFarmsByPools(pools : [Principal]) : async
Result.Result<[(Principal, Principal)], Types.Error> {
        var farms = Buffer.Buffer<Principal>(0);
        for (pool in pools.vals()) {
```



```
            switch (_poolFarms.get(pool)) {
                case (?farmArray) {
                    farms.append(Buffer.fromArray(farmArray));
                };
                case (_) {};
            };
        };
        var liveFarms = TrieSet.toArray(_liveFarmSet);
        if (Nat.equal(farms.size(), 0) or Nat.equal(liveFarms.size(),
0)) { return #ok([]); };
```

[Farm-12] Dependent library version specification error

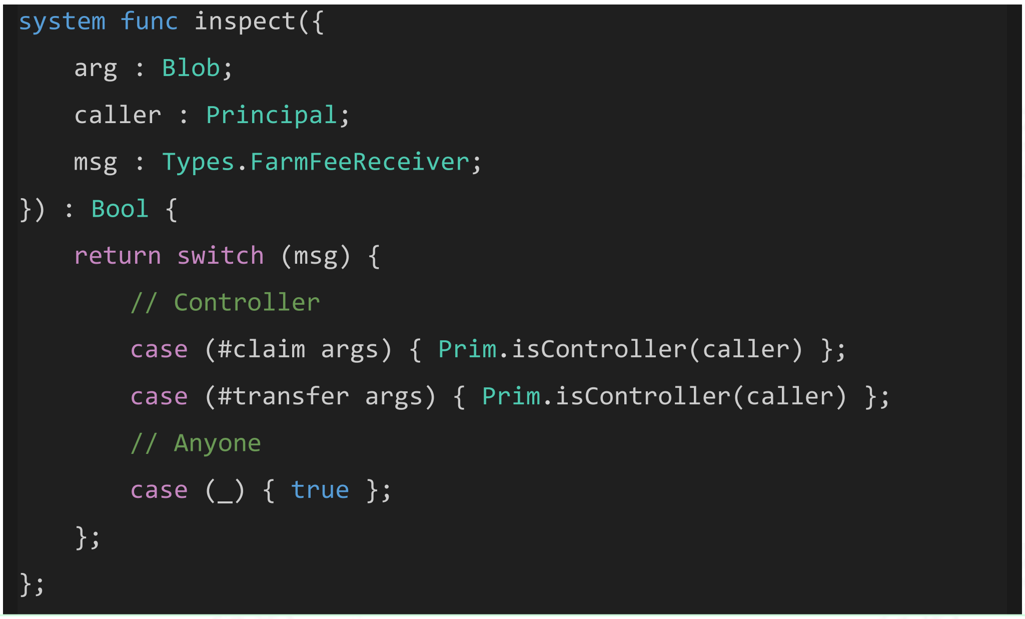
Severity Level	Info
Type	Coding Conventions
Lines	package-set.dhall #L19-23
Description	The `package-set.dhall` file imports version v1.0.3 of the `farm-token-adapter` library. However, the main branch of the library only has versions v1.0.6 and v1.0.7. This indicates a potential issue with importing the correct version.

```
additions =
    [
      { name = "base"
      , repo = "https://github.com/dfinity/motoko-base"
      , version = "moc-0.9.7"
      , dependencies = [] : List Text
      }
      ,{ dependencies = [ "base" ]
      , name = "commons"
      , repo = "git@github.com:farm-Labs/ic-commons-v2.git"
      , version = "v0.0.5"
      }
      ,{ dependencies = [] : List Text
      , name = "token-adapter"
      , repo = "git@github.com:farm-Labs/farm-token-adapter.git"
      , version = "v1.0.3"
      }
      ,{ dependencies = [] : List Text
      , name = "icpswap-v3-service"
      , repo = "git@github.com:ICPSwap-Labs/icpswap-v3-service.git"
      , version = "v3.4.2"
      }
    ]
```

Recommendation	It is recommended that the project team to switch to the appropriate version.
Status	**Fixed.** The project team has updated the library version to v1.0.7.

[Farm-13] Permission check incomplete

Severity Level	Info
Type	Business Security
Lines	FarmFeeReceiver.mo#L105-117
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.FarmFeeReceiver;
}) : 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.** The project team add the permission check of the transferAll.

[Farm-14] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	Farm.mo #L98-109,#114-121,#236-243,#1022-1049
Description	In the `_swapPoolAct` and `_rewardPoolActof` Farm, there is redundant code. The function declares `batchRefreshIncome`, `quote`, and `refreshIncome`, but they are not used.

```
private stable var _swapPoolAct = actor
(Principal.toText(initArgs.pool)) : actor {
  batchRefreshIncome : query (positionIds : [Nat]) -> async
Result.Result<{ totalTokensOwed0 : Nat; totalTokensOwed1 : Nat;
tokenIncome : [(Nat, { tokensOwed0 : Nat; tokensOwed1 : Nat })] },
Types.Error>;
  quote : query (args : Types.SwapArgs) -> async Result.Result<Nat,
Types.Error>;
  metadata : query () -> async Result.Result<Types.PoolMetadata,
Types.Error>;
  getUserPosition : query (positionId : Nat) -> async
Result.Result<Types.UserPositionInfo, Types.Error>;
  transferPosition : shared (from : Principal, to : Principal,
positionId : Nat) -> async Result.Result<Bool, Types.Error>;
  refreshIncome : query (positionId : Nat) -> async
Result.Result<{ tokensOwed0 : Nat; tokensOwed1 : Nat }, Types.Error>;
};
private stable var _rewardPoolAct = actor
(Principal.toText(initArgs.rewardPool)) : actor {
  quote : query (args : Types.SwapArgs) -> async Result.Result<Nat,
Types.Error>;
  metadata : query () -> async Result.Result<Types.PoolMetadata,
Types.Error>;
};
```

The _initLock variable can be deleted because in the init function, there is already a restriction of _init that the function can only be called once, and the caller of the init function must be an authorized account.

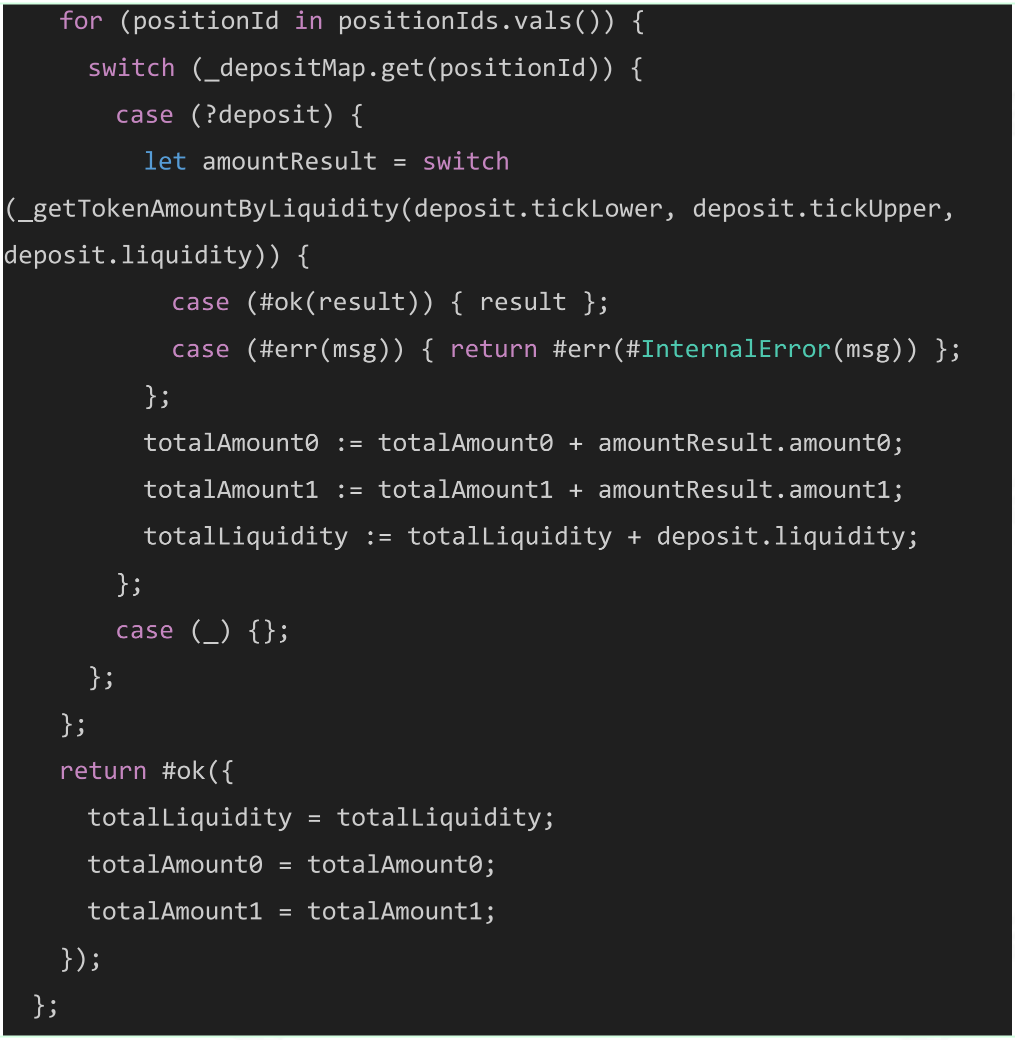
```
private stable var _inited : Bool = false;
  private stable var _initLock : Bool = false;
  public shared (msg) func init() : async () {
    _checkPermission(msg.caller);

    assert (not _inited);
    assert (not _initLock);
    _initLock := true;
```

The previous code has already validated positionTokenAmounts.amount0 < _token0AmountLimit and positionTokenAmounts.amount1 < _token1AmountLimit. There is no need to validate them again.

```
if (_token0AmountLimit != 0 and positionTokenAmounts.amount0 <
_token0AmountLimit) {
      return #err(#InternalError("The quantity of token0 does not reach
the low limit"));
    };
    if (_token1AmountLimit != 0 and positionTokenAmounts.amount1 <
_token1AmountLimit) {
      return #err(#InternalError("The quantity of token1 does not reach
the low limit"));
    };
    if (_token0AmountLimit != 0 and _token1AmountLimit != 0) {
      if (positionTokenAmounts.amount0 < _token0AmountLimit) {
        return #err(#InternalError("The quantity of token0 does not
reach the low limit"));
      };
      if (positionTokenAmounts.amount1 < _token1AmountLimit) {
        return #err(#InternalError("The quantity of token1 does not
reach the low limit"));
      };
    };
```

The private function _getTokenAmounts is not used at all and can be deleted.

```
  private func _getTokenAmounts(positionIds : [Nat]) :
Result.Result<{ totalLiquidity : Nat; totalAmount0 : Int; totalAmount1 :
Int }, Types.Error> {
    if (positionIds.size() == 0) {
      return #ok({ totalLiquidity = 0; totalAmount0 = 0; totalAmount1 =
0 });
    };
    var totalAmount0 : Int = 0;
    var totalAmount1 : Int = 0;
    var totalLiquidity : Nat = 0;
```



```
    for (positionId in positionIds.vals()) {
      switch (_depositMap.get(positionId)) {
        case (?deposit) {
          let amountResult = switch
(_getTokenAmountByLiquidity(deposit.tickLower, deposit.tickUpper,
deposit.liquidity)) {
            case (#ok(result)) { result };
            case (#err(msg)) { return #err(#InternalError(msg)) };
          };
          totalAmount0 := totalAmount0 + amountResult.amount0;
          totalAmount1 := totalAmount1 + amountResult.amount1;
          totalLiquidity := totalLiquidity + deposit.liquidity;
        };
        case (_) {};
      };
    };
    return #ok({
      totalLiquidity = totalLiquidity;
      totalAmount0 = totalAmount0;
      totalAmount1 = totalAmount1;
    });
  };
```

Recommendation	It is recommended to remove the Redundant code.
Status	**Fixed.** The project team has removed the corresponding redundant code.

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