Bird Money Smart Contract Final Audit Report

Project Synopsis

Project Name	**Bird Money**
Platform	Ethereum, Solidity
Github Repo	https://github.com/bird-money/bird-farm-contracts/tree/master/contracts
Deployed Contract	Not Deployed
Total Duration	4 Days
Timeline of Audit	**21st April 2021 to 23rd April 2021**

Contract Details

Total Contract(s)	1
Name of Contract(s)	BirdFarm
Language	Solidity
Commit Hash	**82bdf8f63431221cd878d3625e1bd1ba46f51147**

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	0	05
Medium Severity	01	02
Low Severity	0	05
Information	0	02
Total Found	01	14

Detailed Results

The contract has gone through several stages of the audit procedure that includes structural analysis, automated testing, manual code review etc.

All the issues have been explained and discussed in detail below. Along with the explanation of the issue found during the audit, the recommended way to overcome the issue or improve the code quality has also been mentioned.

A.Contract Name: BirdFarm

Critical Severity Issues

A.1 Similar LP Token Address can be added more than once.

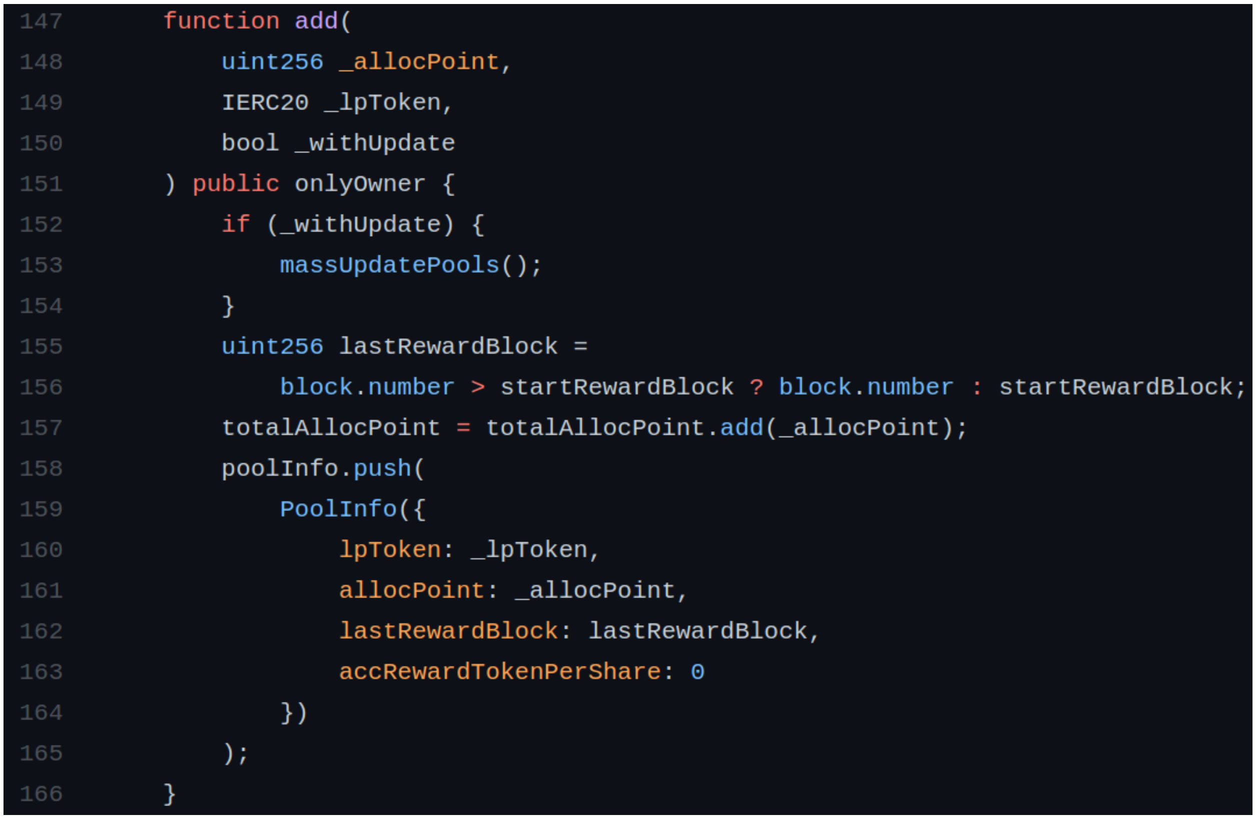
Status: CLOSED

Line no - 147

Explanation:

As per the current architecture of the BirdFarm contract, the **LP Tokens** added in the pool of this contract should not be repeated. Since the address of an **LP Token** plays an imperative role in the calculation of rewards as well as keeping track of specific LP supply in the pool, the presence of a similar LP Token address more than once will break some of the core functionalities of the contract.

However, the **add()** function at Line 147 allows storing a similar LP Token Address more than once. This will lead to an unexpected scenario where different pools will have a similar LP token address.



```
147     function add(
148         uint256 _allocPoint,
149         IERC20 _lpToken,
150         bool _withUpdate
151     ) public onlyOwner {
152         if (_withUpdate) {
153             massUpdatePools();
154         }
155         uint256 lastRewardBlock =
156             block.number > startRewardBlock ? block.number : startRewardBlock;
157         totalAllocPoint = totalAllocPoint.add(_allocPoint);
158         poolInfo.push(
159             PoolInfo({
160                 lpToken: _lpToken,
161                 allocPoint: _allocPoint,
162                 lastRewardBlock: lastRewardBlock,
163                 accRewardTokenPerShare: 0
164             })
165         );
166     }
```

Recommendation:

The argument ***_lpToken*** (*LP token address*) passed in the **add()** function must be checked at the very beginning of the function with a **require statement.**
The **require statement** should be designed to check whether or not the passed lpToken address is already available in the contract. Moreover, the **add()** function should only execute if a new lpToken address is passed, thus eliminating any chances of repeating a similar lpToken in more than one pool.

A.2 safeRewardTokenTransfer function does not execute adequately if Reward Token Balance in the contract is less than the amount of reward tokens to be transferred to a particular user

Line no - 319

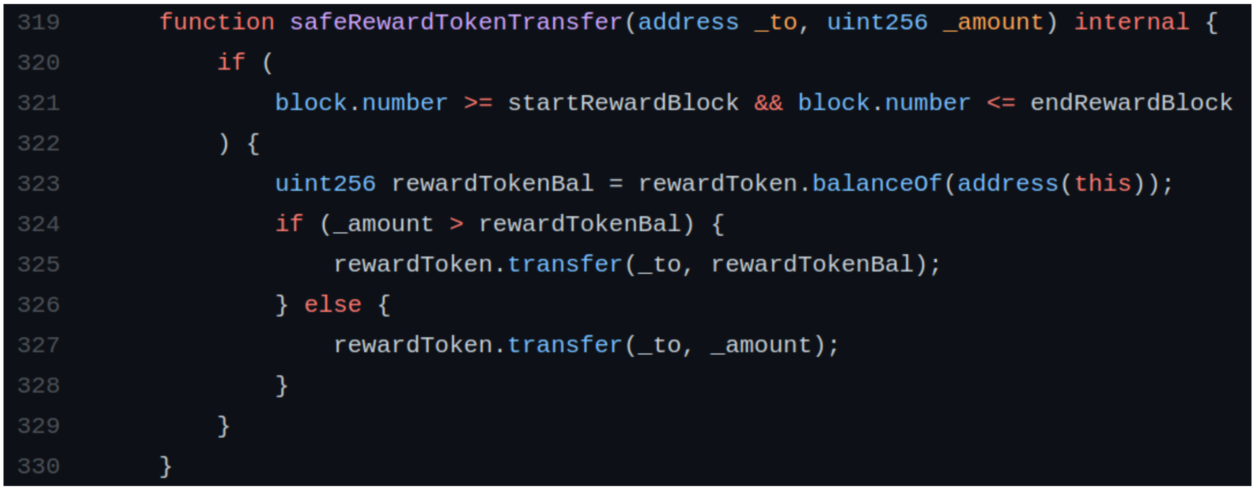
Status: CLOSED

Note: safeRewardTokenTransfer function has been removed from the contract

Explanation:
The **safeRewardTokenTransfer** is designed in a way that it first checks whether or not the BirdFarm contract has more reward token balance than the amount of tokens to be transferred to the user(*Line 324*).

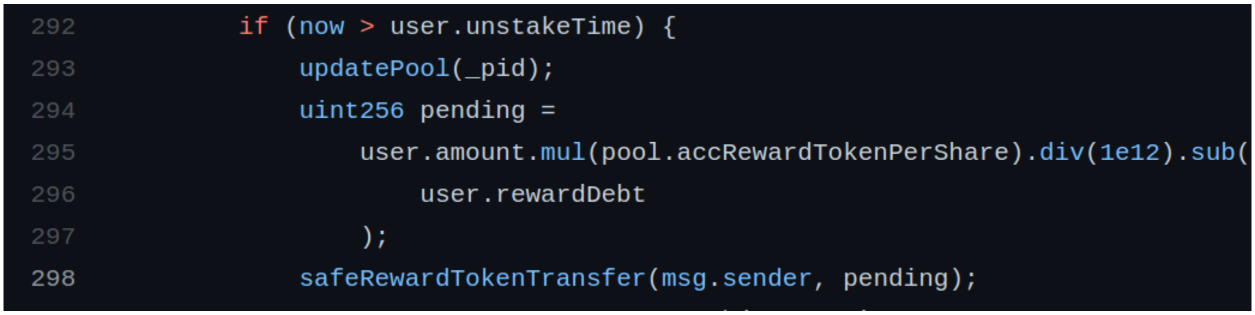
If the BirdFarm contract has less reward token balance than the amount to be transferred, the user gets only the remaining reward tokens in the contract and not the actual amount that was supposed to be transferred(*Line 325*).

However, the major issue in this function is that if the above-mentioned condition is met and the user only gets the remaining reward tokens in BirdFarm contract instead of the actual reward tokens, then the remaining amount of reward tokens that the user didn't receive yet is never stored throughout the function.



```
319     function safeRewardTokenTransfer(address _to, uint256 _amount) internal {
320         if (
321             block.number >= startRewardBlock && block.number <= endRewardBlock
322         ) {
323             uint256 rewardTokenBal = rewardToken.balanceOf(address(this));
324             if (_amount > rewardTokenBal) {
325                 rewardToken.transfer(_to, rewardTokenBal);
326             } else {
327                 rewardToken.transfer(_to, _amount);
328             }
329         }
330     }
```

For instance, if the user is supposed to receive **1000** reward tokens while calling the **withdraw function** after the complete lock-up period is over.
The withdraw function will call the **safeRewardTokenTransfer** function(*Line 298*) and pass the user's address and the reward token amount of 1000 tokens in the **pending variable.**



```
292         if (now > user.unstakeTime) {
293             updatePool(_pid);
294             uint256 pending =
295                 user.amount.mul(pool.accRewardTokenPerShare).div(1e12).sub(
296                     user.rewardDebt
297                 );
298             safeRewardTokenTransfer(msg.sender, pending);
```

However, if the BirdFarm contract has only 800 reward tokens then the **if condition** at *Line 324* will be executed and the user will only receive **800 reward tokens** instead of **1000 reward tokens.**

Now, because of the fact that the **safeRewardTokenTransfer** function doesn't store this information that the user still owes 200 reward tokens will lead to an unexpected scenario where the users receive less reward token than expected.

Is this scenario Intended ?

Recommendation:

If the above-mentioned scenario was not considered while developing the **safeRewardTokenTransfer,** then the function must be updated in such a way that the user gets the actual amount of reward tokens whenever the **safeRewardTokenTrasnfer** function is called.

A.3 Contract State Variables are being updated after External Calls. Leads to a Potential Reentrancy Scenario

Line no - 276-284

Status: CLOSED

Explanation:

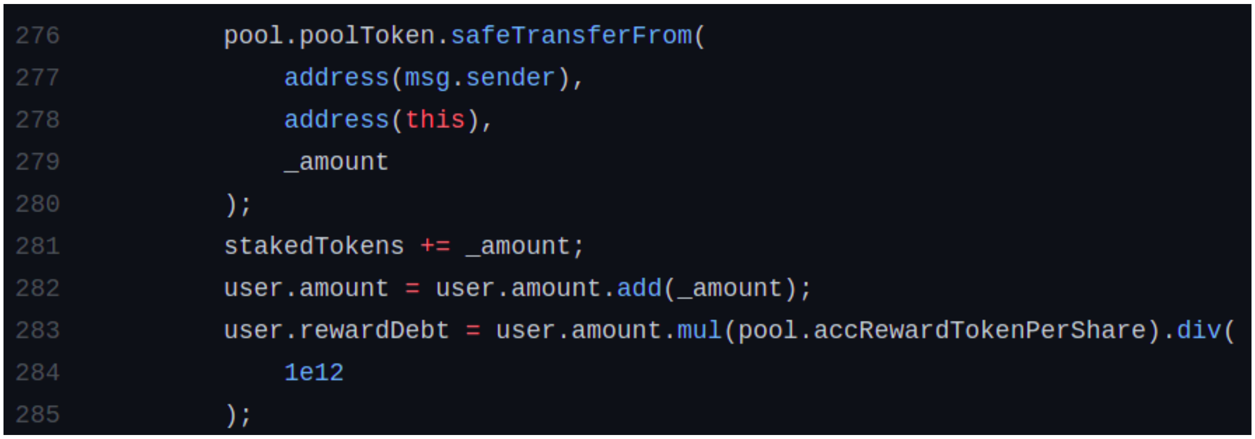
The BirdFarm contract includes quite a few functions that update some of the very imperative state variables of the contract after the external calls are being made.

An external call within a function technically shifts the control flow of the contract to another contract for a particular period of time. Therefore, as per the Solidity Guidelines, any modification of the state variables in the base contract must be performed before executing the external call.
Updating state variables after an external call might lead to a potential re-entrancy scenario.

The following functions in the contract updates the state variables after making an external call

- ***deposit() function** at Line **278-282***

For instance, the **deposit function** makes an external call to the ***poolToken*** contract(*Line 276*) to transfer the amount of token from user to contract.
However the User struct(***state variable in the contract***) is updated(*Line 281-284*) after the external call is made.
This is not considered a secure practice while developing smart contracts in Solidity.



```
276            pool.poolToken.safeTransferFrom(
277                address(msg.sender),
278                address(this),
279                _amount
280            );
281            stakedTokens += _amount;
282            user.amount = user.amount.add(_amount);
283            user.rewardDebt = user.amount.mul(pool.accRewardTokenPerShare).div(
284                1e12
285            );
```

Recommendation:

Modification of any State Variables must be performed before making an external call.

A.4 Unstake Time increases 72 Hours with every new Deposit

Line no - 278

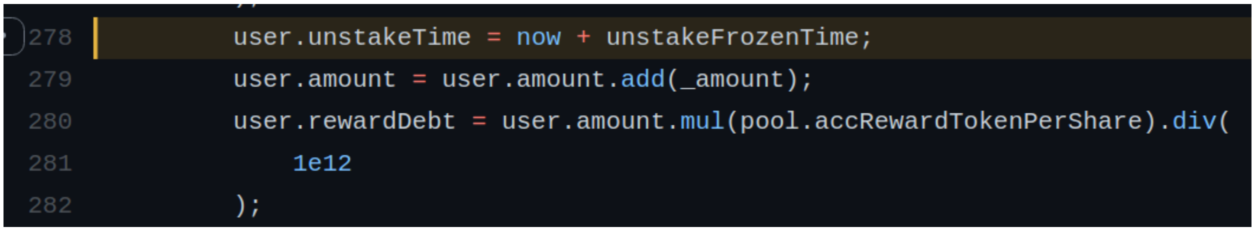
Status: CLOSED

Explanation:

The total lock period for every deposit has been assigned to be 72 hours(*Line no - 73*). It indicates that the user will be able withdraw his/her lp tokens as well as the rewards, after a total duration of 72 hours.

However, as per the current design of the deposit function, the unstake time for a particular user keeps increasing whenever any new deposit is made.

For instance, if a user deposits 1000 LP tokens initially, the unstake time for this user will be 72 hours, i.e., (current time + 72 hours) and the user will be able to withdraw his 1000 LP tokens only after this time period is over.
But if the user once again deposits 500 LP tokens before the unstake time is over, then the unstake time for this user is again updated to 72 hours more for his entire deposit amount. The user will be able to unstake his entire deposit amount of 1500 LP tokens only after the complete unstake period of 72 hours ends.



```
278            user.unstakeTime = now + unstakeFrozenTime;
279            user.amount = user.amount.add(_amount);
280            user.rewardDebt = user.amount.mul(pool.accRewardTokenPerShare).div(
281                1e12
282            );
```

Recommendation:

Is the above-mentioned behaviour intended?

If the above-mentioned issue was not considered during the design of the deposit function, then the function should be modified in a way that the unstake period of each deposit is treated separately so that the users can get a clear idea of the withdrawal time for each of their deposits.

However, if this behaviour is intended, the community should be informed about this deposit & unstake time update functionality beforehand to eliminate any confusion later.

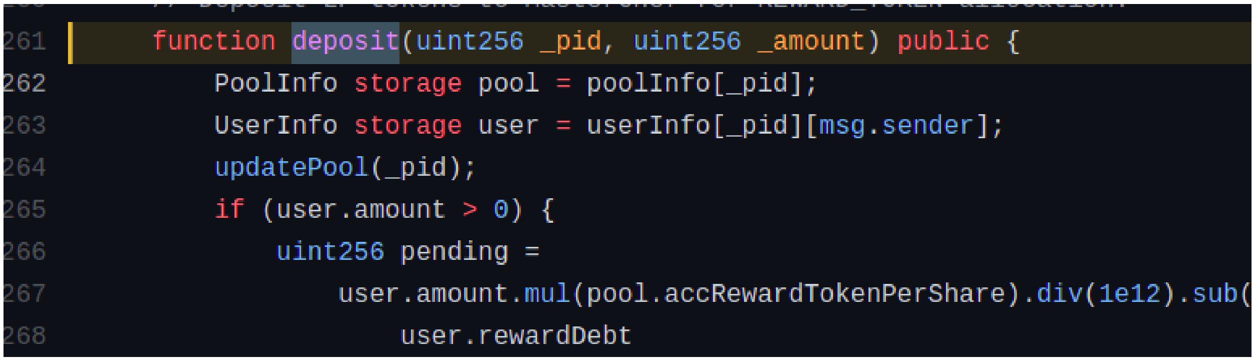
A.5 User is capable of depositing ZERO amount of LP Tokens

Line no - 261-284

Status: CLOSED

Explanation:

As per the current implementation of the **deposit function**, a user is capable of depositing **ZERO** amount of Lp Tokens as well.
Since the user deposit amount plays a significant role in the reward calculation as well as withdrawal procedure, the input validation for the deposit amount must be implemented in the function.
Moreover, allowing users to deposit ZERO amount of tokens doesn't represent a better function design for the deposit function.



```
    function deposit(uint256 _pid, uint256 _amount) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pending =
                user.amount.mul(pool.accRewardTokenPerShare).div(1e12).sub(
                    user.rewardDebt
```

Recommendation:

The **deposit** function should include a **require** statement to validate the ***_amount*** argument passed by the user. The **require** statement must ensure that this argument is either greater than **ZERO** or a **Minimum Deposit Threshold** that can be set by the owner of the contract

---

Medium Severity Issues

A.6 Violation of Check_Effects_Interaction Pattern in the Withdraw function

Line no - 292-305

Status: CLOSED

Explanation:

As per the Check_Effects_Interaction Pattern in Solidity, external calls should be made at the very end of the function and event emission, as well as any state variable modification, must be done before the external call is made.

During the automated testing it was found that the following **functions** does not follow the **Check Effects Interaction Pattern** effectively as event is emitted after the external call is executed.

- ***addRewardTokensToContract***
- ***deposit***

- ***emergencyWithdraw***
- ***harvest***
- ***withdraw***

Recommendation:
Check Effects Interaction Pattern must be followed while implementing external calls in a function.

A.7 updatePool and massUpdatePools functions have been assigned a Public visibility

Line no - 229-234, 237-258

Status: CLOSED

Explanation:
The **updatePool** and **massUpdatePools** functions include imperative functionalities as they deal with updating the reward variables of a given pool.

These functions are called within the contract by some crucial functions like **add(), deposit, withdraw** etc.
However, instead of an **internal visibility,** these functions have been assigned a public visibility.
Since **public** visibility will make the ***updatePool & massUpdatePools function*** accessible to everyone, it would have been a more effective and secure approach to mark these functions as **internal.**
Recommendation:
If both of these functions are only to be called from within the contract, their visibility specifier should be changed from **public** to **internal**.

A.8 Multiplication is being performed on the result of Division

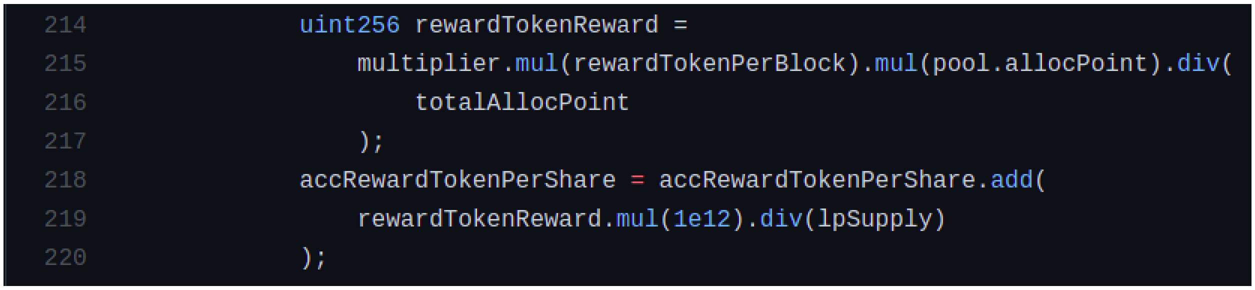
Line no - 214-219, 248-256

Status: Not Considered

Explanation:
During the automated testing of the BirdFarm.sol contract, it was found that some of the functions in the contract are performing multiplication on the result of a Division. Integer Divisions in Solidity might truncate. Moreover, this performing division before multiplication might lead to loss of precision.

The following functions involve division before multiplication in the mentioned lines:

- ***pendingRewardToken*** *at 214-219*
- ***updatePool*** *at 248-256*



```
            uint256 rewardTokenReward =
                multiplier.mul(rewardTokenPerBlock).mul(pool.allocPoint).div(
                    totalAllocPoint
                );
            accRewardTokenPerShare = accRewardTokenPerShare.add(
                rewardTokenReward.mul(1e12).div(lpSupply)
            );
```

Recommendation:

Solidity doesn’t encourage arithmetic operations that involve division before multiplication. Therefore the above-mentioned function should be checked once and redesigned if they do not lead to expected results.

Low Severity Issues

A.9 safeRewardTokenTransfer function should include require statement instead of IF-Else Statement

Line no: 320-322

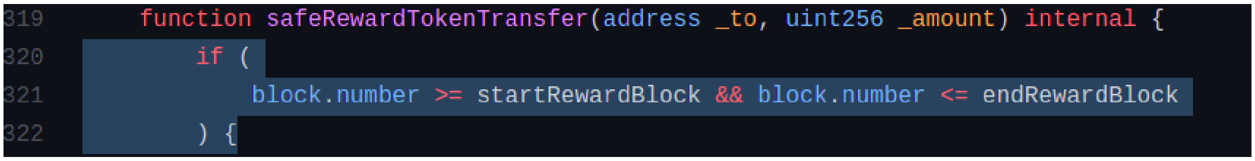
Status: CLOSED

Explanation

The safeRewardTokenTransfer function includes an **if statement** at the very beginning of the function to check whether or not the block.number lies in the valid range of reward blocks.

The function body is only executed if this **IF statement** holds true.

In order to check for such validations in a function, **require statements** are more preferable and effective solidity. While it helps in gas optimizations it also enhances the readability of the code.



```
319     function safeRewardTokenTransfer(address _to, uint256 _amount) internal {
320         if (
321             block.number >= startRewardBlock && block.number <= endRewardBlock
322         ) {
```

Recommendation

Use **require statement** instead of **IF statement** in the above-mentioned function line.

For instance,

require(block.number >= startRewardBlock && block.number <= endRewardBlock,"Error MSG: Block Number doesn't lie in Valid Range");

A.10 External Visibility should be preferred

Status: CLOSED

Explanation:

Those functions that are never called throughout the contract should be marked as ***external*** visibility instead of ***public*** visibility.
This will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as **external** within the contract:

- ***setRewardToken***
- ***setAll***
- ***setUnstakeFrozenTime***
- ***setRewardTokenPerBlock***
- ***setStartRewardBlock***
- ***serEndRewardBlock***
- ***setBonusEndBlock***
- ***add***
- ***set***
- ***deposit***
- ***withdraw***
- ***emergencyWithdraw***
- ***setMigrator***
- ***depositRewardTokens***
- ***withdrawRewardTokens***

A.11 withdraw function should include require statement instead of IF-Else Statement

Line no: 292

Status: CLOSED

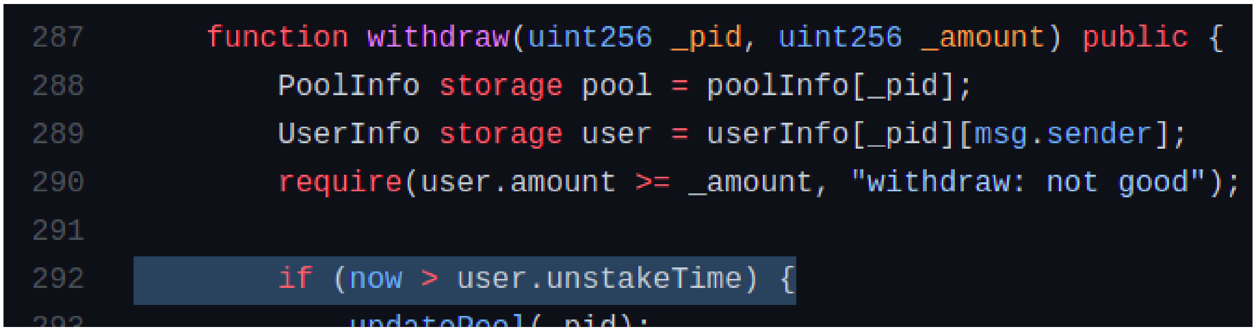
Explanation:

As per the current architecture of the contract, the withdraw function should only enter its function body if 2 imperative conditions are met:

- User Deposit amount is more than ZERO
- Current Time is more than the unstake time

Since these are strict conditions and must be fulfilled before a user can withdraw his/her LP Tokens and reward, **require** statements will be a more effective approach to check such conditions.

While the amount condition is already being checked using the ***require*** statement, the unstake time condition should also be checked with a *require* statement instead of a ***IF-Else*** statement.



```
    function withdraw(uint256 _pid, uint256 _amount) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");

        if (now > user.unstakeTime) {
            updatePool(_pid);
```

Recommendation;
Use **require statement** instead of **IF statement** in the above-mentioned function line. For instance,
require(now> user.unstakeTime,”Error MSG: Unstake Time Not Reached Yet”);

A.12 Return Value of an External Call is never used Effectively

Status: CLOSED

Line no - 325, 327, 345, 349

Explanation:
The external calls made in the above-mentioned lines do return a boolean value that indicates whether or not the external call made was successful.
These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.
However, the BirdFarm contract never uses these return values throughout the contract.

Recommendation:
Effective use of all the return values from external calls must be ensured within the contract.

A.13 No Events emitted after imperative State Variable modification

Line no -118,122,129,133,137

Status: CLOSED

Description:

Functions that update an imperative arithmetic state variable contract should emit an event after the updation.

The following functions modify some crucial arithmetic parameters like ***rewardTokenPerblock, startRewardBlock, endRewardBlock*** etc in the BirdFarm contract but don't emit any event after that:

- ***setUnstakeFrozenTime***
- ***setRewardTokenPerBlock***
- ***setStartRewardBlock***
- ***setEndRewardBlock***

Since there is no event emitted on updating these variables, it might be difficult to track it off chain.

Recommendation:

An event should be fired after changing crucial arithmetic state variables.

Informational

A.14 NatSpec Annotations must be included

Status: CLOSED

Description:

Smart contract does not include the NatSpec annotations adequately.

Recommendation:

Cover by NatSpec all Contract methods.

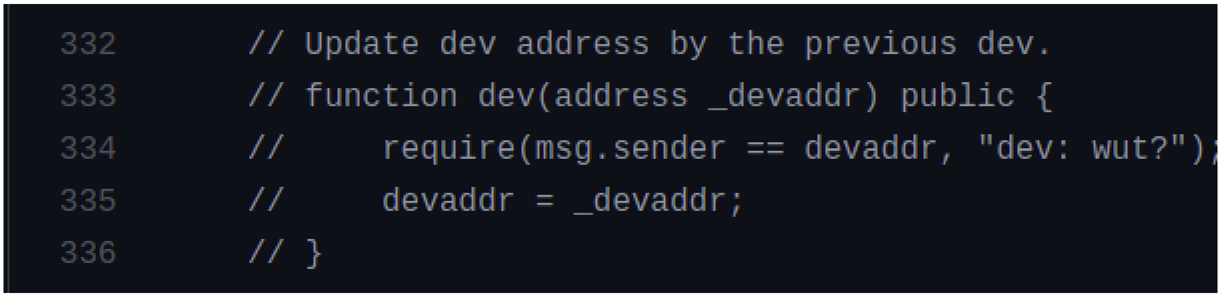
A.15 Commented codes must be wiped-out before deployment

Status: CLOSED

Explanation

The BirdFarmMasterchef contract includes quite a few commented codes regarding a **devAddress state variable(*Line* 51, 85).**

This badly affects the readability of the code.

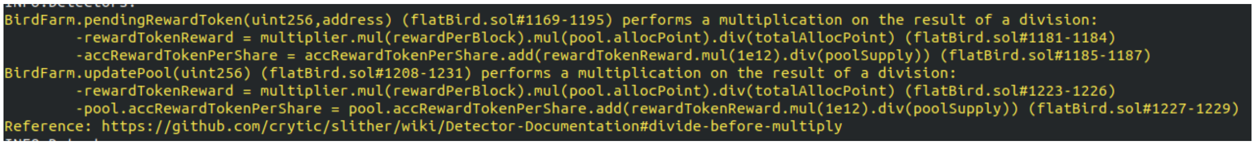


```
        // Update dev address by the previous dev.
        // function dev(address _devaddr) public {
        //     require(msg.sender == devaddr, "dev: wut?");
        //     devaddr = _devaddr;
        // }
```

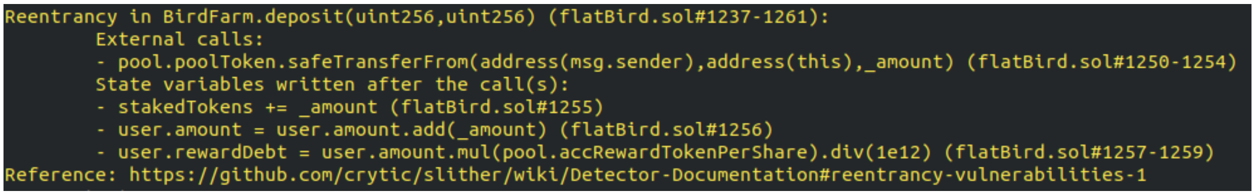
Recommendation:

If these instances of code are not required in the current version of the contract, then the commented codes must be removed before deployment.

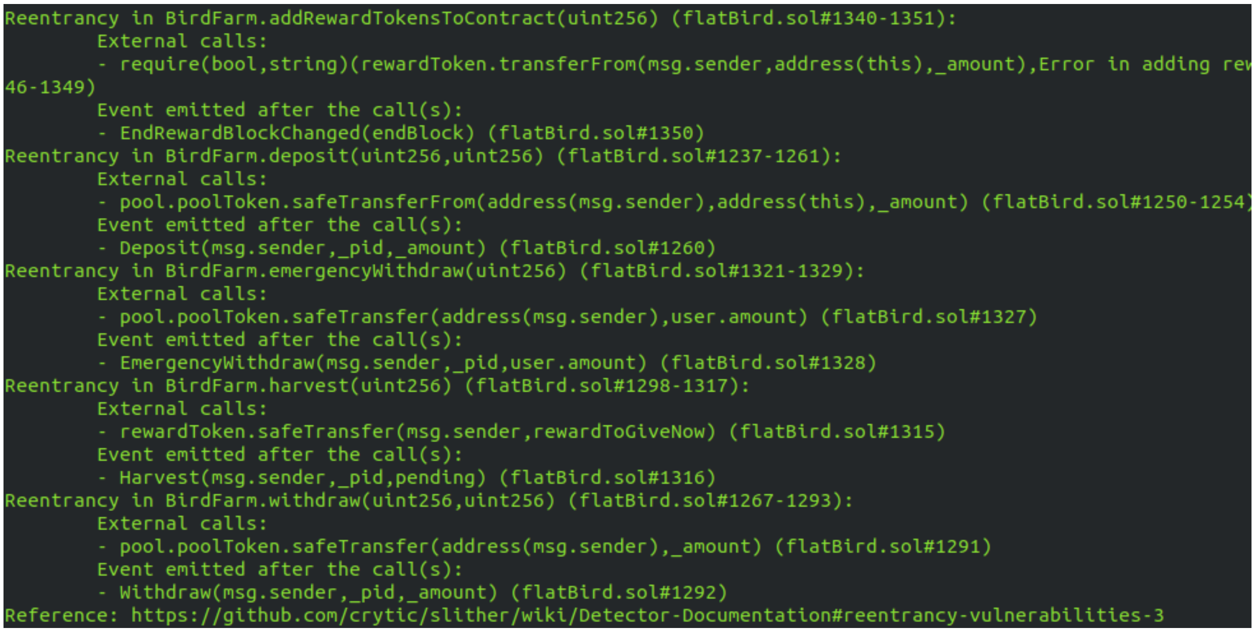
Slither Test Results:



```
BirdFarm.pendingRewardToken(uint256,address) (flatBird.sol#1169-1195) performs a multiplication on the result of a division:
        -rewardTokenReward = multiplier.mul(rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (flatBird.sol#1181-1184)
        -accRewardTokenPerShare = accRewardTokenPerShare.add(rewardTokenReward.mul(1e12).div(poolSupply)) (flatBird.sol#1185-1187)
BirdFarm.updatePool(uint256) (flatBird.sol#1208-1231) performs a multiplication on the result of a division:
        -rewardTokenReward = multiplier.mul(rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (flatBird.sol#1223-1226)
        -pool.accRewardTokenPerShare = pool.accRewardTokenPerShare.add(rewardTokenReward.mul(1e12).div(poolSupply)) (flatBird.sol#1227-1229)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```



```
Reentrancy in BirdFarm.deposit(uint256,uint256) (flatBird.sol#1237-1261):
        External calls:
        - pool.poolToken.safeTransferFrom(address(msg.sender),address(this),_amount) (flatBird.sol#1250-1254)
        State variables written after the call(s):
        - stakedTokens += _amount (flatBird.sol#1255)
        - user.amount = user.amount.add(_amount) (flatBird.sol#1256)
        - user.rewardDebt = user.amount.mul(pool.accRewardTokenPerShare).div(1e12) (flatBird.sol#1257-1259)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```



```
Reentrancy in BirdFarm.addRewardTokensToContract(uint256) (flatBird.sol#1340-1351):
        External calls:
        - require(bool,string)(rewardToken.transferFrom(msg.sender,address(this),_amount),Error in adding rew
46-1349)
        Event emitted after the call(s):
        - EndRewardBlockChanged(endBlock) (flatBird.sol#1350)
Reentrancy in BirdFarm.deposit(uint256,uint256) (flatBird.sol#1237-1261):
        External calls:
        - pool.poolToken.safeTransferFrom(address(msg.sender),address(this),_amount) (flatBird.sol#1250-1254)
        Event emitted after the call(s):
        - Deposit(msg.sender,_pid,_amount) (flatBird.sol#1260)
Reentrancy in BirdFarm.emergencyWithdraw(uint256) (flatBird.sol#1321-1329):
        External calls:
        - pool.poolToken.safeTransfer(address(msg.sender),user.amount) (flatBird.sol#1327)
        Event emitted after the call(s):
        - EmergencyWithdraw(msg.sender,_pid,user.amount) (flatBird.sol#1328)
Reentrancy in BirdFarm.harvest(uint256) (flatBird.sol#1298-1317):
        External calls:
        - rewardToken.safeTransfer(msg.sender,rewardToGiveNow) (flatBird.sol#1315)
        Event emitted after the call(s):
        - Harvest(msg.sender,_pid,pending) (flatBird.sol#1316)
Reentrancy in BirdFarm.withdraw(uint256,uint256) (flatBird.sol#1267-1293):
        External calls:
        - pool.poolToken.safeTransfer(address(msg.sender),_amount) (flatBird.sol#1291)
        Event emitted after the call(s):
        - Withdraw(msg.sender,_pid,_amount) (flatBird.sol#1292)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```