# FourRX MasterChef Smart Contract Final Audit Report

## Project Synopsis

| | |
|---|---|
| **Project Name** | **FOUR RX** |
| **Platform** | Ethereum, Solidity |
| **Github Repo** | https://github.com/FourRX/4rx/blob/master/contracts/Farm/MasterChef.sol |
| **Deployed Contract** | Not Deployed |
| **Total Duration** | 3 Days |
| **Timeline of Audit** | **5th April 2021 to 8th April 2021** |

## Contract Details

| | |
|---|---|
| **Total Contract(s)** | 2 |
| **Name of Contract(s)** | MasterChef |
| **Language** | Solidity |
| **Commit Hash** | **57193fa3acb212cdb827e24513687deabe2bf7af** |

## Contract Vulnerabilities Synopsis

| Issues | Open Issues | Closed Issues |
|---|---|---|
| Critical Severity | 0 | 0 |
| Medium Severity | 1 | 3 |
| Low Severity | 1 | 4 |
| Information | 2 | 0 |
| Total Found | 4 | 7 |

## 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: MasterChef

### Medium Severity Issues

#### A.1 safeFRXTransfer function does not execute adequately if FRX Token Balance in the contract is less than the amount of tokens to be transferred

**Line no - 295 to 302**

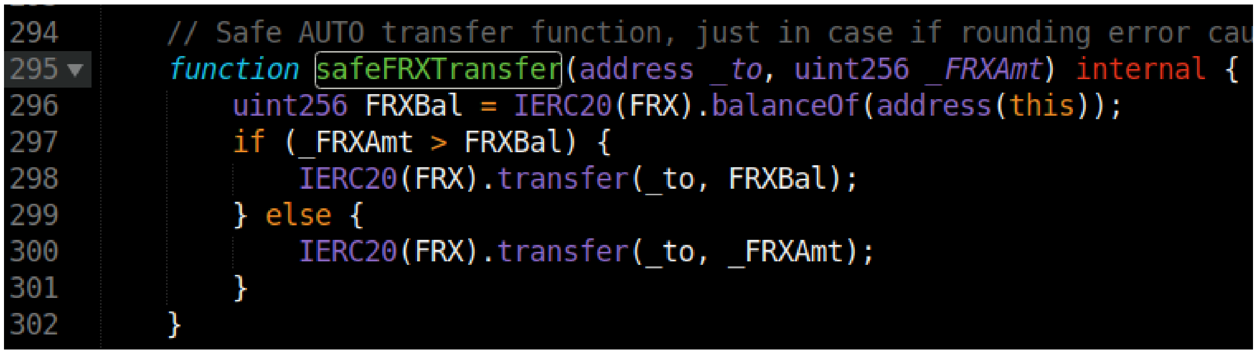
**Status: INTENTIONAL**

***Explanation:***

The **safeFRXTransfer** is designed in a way that it first checks whether or not the MasterChef contract has more FRX token balance than the amount of tokens to be transferred to the user(*Line* 296).

If the MasterChef contract has less reward token balance than the amount to be transferred, the user gets only the remaining FRX tokens in the contract and not the actual amount that was supposed to be transferred(*Line* 298).

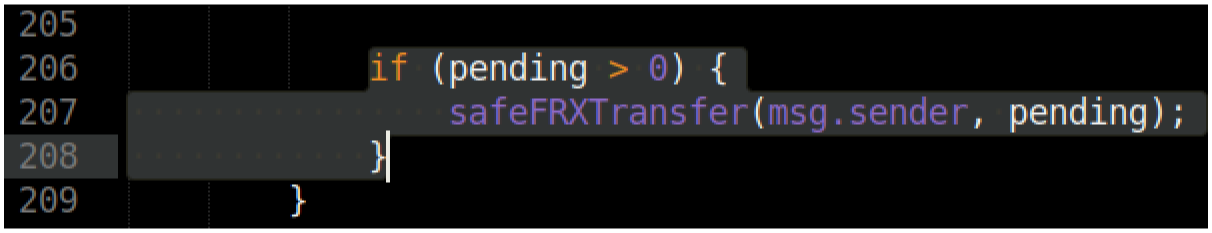
However, the major issue in this function is that if the above-mentioned condition is met and the user only gets the remaining FRX tokens in MasterChef contract instead of the actual sushFRXi tokens that should be transferred, then the remaining amount of tokens that the user didn't receive yet is never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



```
294     // Safe AUTO transfer function, just in case if rounding error cau
295     function safeFRXTransfer(address _to, uint256 _FRXAmt) internal {
296         uint256 FRXBal = IERC20(FRX).balanceOf(address(this));
297         if (_FRXAmt > FRXBal) {
298             IERC20(FRX).transfer(_to, FRXBal);
299         } else {
300             IERC20(FRX).transfer(_to, _FRXAmt);
301         }
302     }
```

For instance, if the user is supposed to receive **1000** FRX tokens while calling the **withdraw or deposit function**.

The deposit function will call the **safeFRXTransfer** function(*Line* 207) and pass the user's address and the pending token amount of 1000 tokens.



```
            if (pending > 0) {
                safeFRXTransfer(msg.sender, pending);
            }
        }
```

However, if the MasterChef contract has only 800 reward tokens then the **if condition** at *Line 297* will be executed and the user will only receive **800 FRX tokens** instead of **1000 FRX tokens.**
Now, because of the fact that the **safeFRXTransfer** function doesn't store this information, the user still owes 200 FRX tokens will lead to an unexpected scenario where the users receive less reward token than expected.

### Was this scenario Intended or Considered during the development of this function?

***Recommendation:***
If the above-mentioned scenario was not considered, then the function must be updated in such a way that the user gets the actual amount of reward tokens whenever the **safeFRXTransfer** function is called. Otherwise, an adequate amount of FRX tokens should always be maintained in the contract.

## A.2 Multiplication is being performed on the result of Division

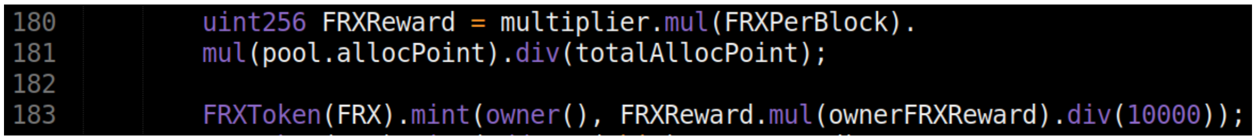
**Line no - 138-139, 181-183, 193**

**Status: OPEN**
***Explanation:***
During the automated testing of the MasterChef.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:

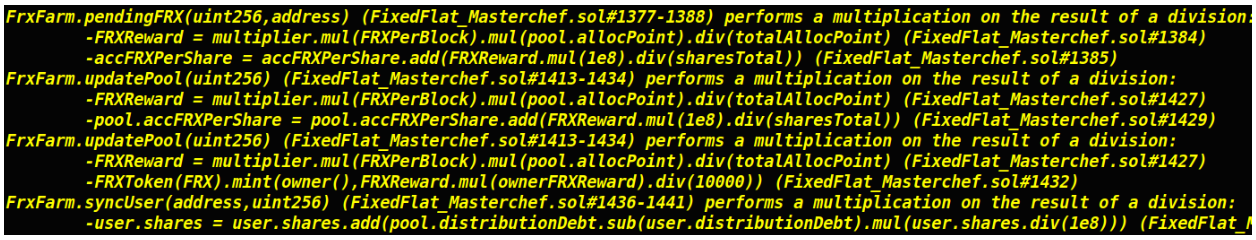
- ***pendingFRX*** at *137*

- ***updatePool** at 180-182*
- ***syncUser** at 192*



```
180         uint256 FRXReward = multiplier.mul(FRXPerBlock).
181         mul(pool.allocPoint).div(totalAllocPoint);
182
183         FRXToken(FRX).mint(owner(), FRXReward.mul(ownerFRXReward).div(10000));
```

**Automated Test Results on Updated Contract:**



```
FrxFarm.pendingFRX(uint256,address) (FixedFlat_Masterchef.sol#1377-1388) performs a multiplication on the result of a division:
        -FRXReward = multiplier.mul(FRXPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (FixedFlat_Masterchef.sol#1384)
        -accFRXPerShare = accFRXPerShare.add(FRXReward.mul(1e8).div(sharesTotal)) (FixedFlat_Masterchef.sol#1385)
FrxFarm.updatePool(uint256) (FixedFlat_Masterchef.sol#1413-1434) performs a multiplication on the result of a division:
        -FRXReward = multiplier.mul(FRXPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (FixedFlat_Masterchef.sol#1427)
        -pool.accFRXPerShare = pool.accFRXPerShare.add(FRXReward.mul(1e8).div(sharesTotal)) (FixedFlat_Masterchef.sol#1429)
FrxFarm.updatePool(uint256) (FixedFlat_Masterchef.sol#1413-1434) performs a multiplication on the result of a division:
        -FRXReward = multiplier.mul(FRXPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (FixedFlat_Masterchef.sol#1427)
        -FRXToken(FRX).mint(owner(),FRXReward.mul(ownerFRXReward).div(10000)) (FixedFlat_Masterchef.sol#1432)
FrxFarm.syncUser(address,uint256) (FixedFlat_Masterchef.sol#1436-1441) performs a multiplication on the result of a division:
        -user.shares = user.shares.add(pool.distributionDebt.sub(user.distributionDebt).mul(user.shares.div(1e8))) (FixedFlat_
```

***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.

## A.3 Contract State Variables are being updated after External Calls. Violation of Check_Effects_Interaction Pattern

**Line no - 294-297**

**Status: Partially CLOSED**

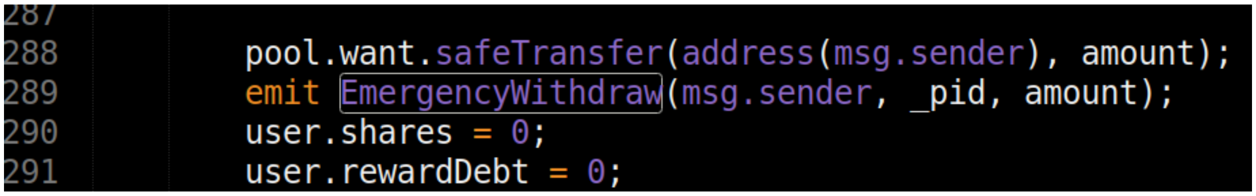
***Explanation:***

The MasterChef 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.

As per the Solidity Guidelines, any modification of any state variables in the base contract must be performed before executing the external call.
Updating state variables after an external call violates the Check-Effects-Interaction Pattern.

The following functions in the contract updates the state variables after making an external call at the line numbers specified below:

- ***emergencyWithdraw** at Line 294-297*



```
287
288         pool.want.safeTransfer(address(msg.sender), amount);
289         emit EmergencyWithdraw(msg.sender, _pid, amount);
290         user.shares = 0;
291         user.rewardDebt = 0;
```

***Recommendation:***
Modification of any State Variables must be performed before making an external call. Check-Effects Interaction Pattern should be followed.

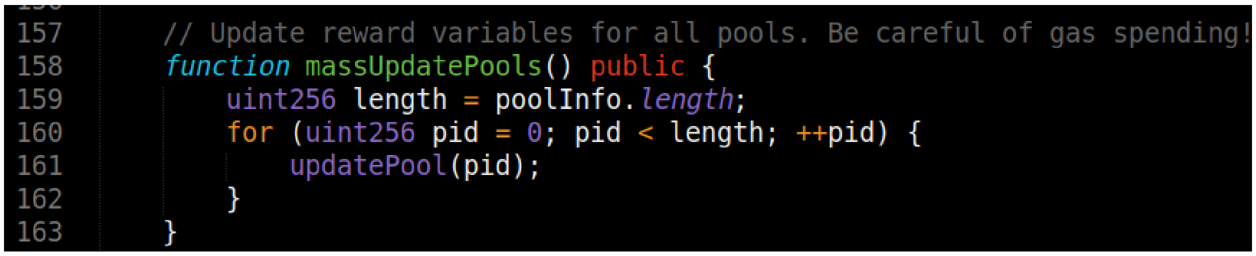
## A.4 ***updatePool*** and ***massUpdatePools*** functions have been assigned a Public visibility

**Line no - 166, 158**

Status: INTENTIONAL

***Explanation:***
The **updatePool** and **massUpdatePools** functions include imperative functionalities as they deal with updating the reward blocks 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.



```
157        // Update reward variables for all pools. Be careful of gas spending!
158        function massUpdatePools() public {
159            uint256 length = poolInfo.length;
160            for (uint256 pid = 0; pid < length; ++pid) {
161                updatePool(pid);
162            }
163        }
```

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**.

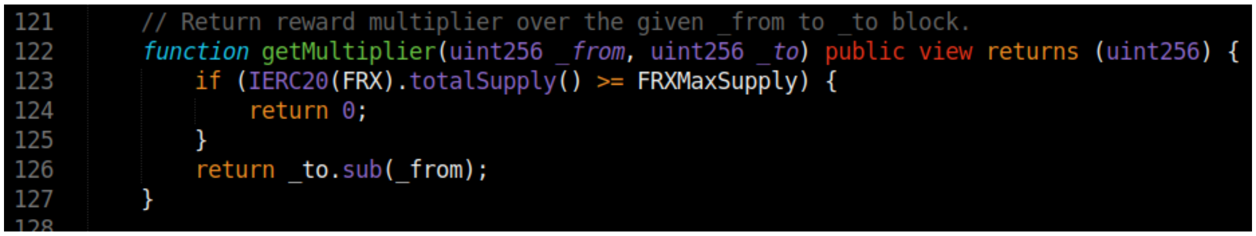
## Low Severity Issues

### A.5 getMultiplier function could be marked as INTERNAL

Line no - 122

Status: CLOSED

***Explanation:***
Functions that are only supposed to be called from within the contract should be marked as **Internal**.
In the Masterchef contract, the ***getMultiplier*** has been assigned a Public visibility while it is being called within other functions of the contract. Moreover, it accepts **block.number** as arguments.



```
    // Return reward multiplier over the given _from to _to block.
    function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
        if (IERC20(FRX).totalSupply() >= FRXMaxSupply) {
            return 0;
        }
        return _to.sub(_from);
    }
```

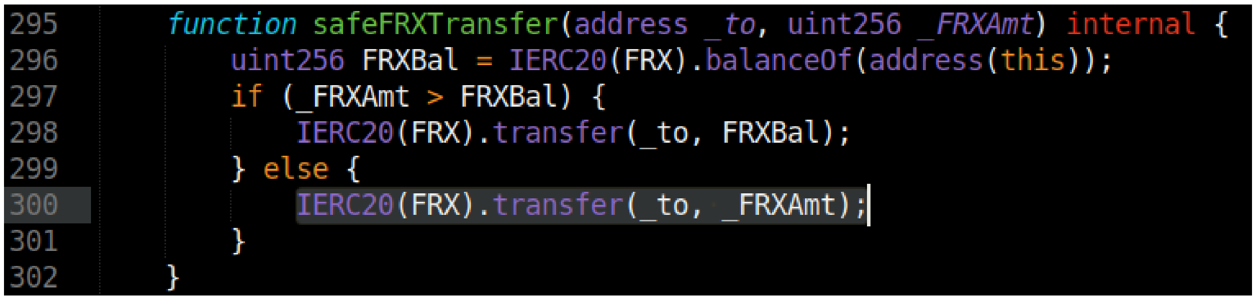
***Recommendation:***
If the **PUBLIC** visibility of the getMultiplier function is not intended, the function should be marked as **INTERNAL.**

## A.6 Return Value of an External Call is never used Effectively

**Line no - 286, 298, 300**

Status: INTENTIONAL

***Explanation:***
The external calls made in the above-mentioned lines return a boolean as well as **uint256** values.
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 MasterChef contract does not use these return values effectively in some instances of the contract.



```
295     function safeFRXTransfer(address _to, uint256 _FRXAmt) internal {
296         uint256 FRXBal = IERC20(FRX).balanceOf(address(this));
297         if (_FRXAmt > FRXBal) {
298             IERC20(FRX).transfer(_to, FRXBal);
299         } else {
300             IERC20(FRX).transfer(_to, _FRXAmt);
301         }
302     }
```

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

## A.7 External Visibility should be preferred

**Status: INTENTIONAL**

***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:

- ***add***
- ***set***
- ***withdrawAll***
- ***emergencyWithdraw***
- ***inCaseTokensGetStuck***

***Recommendation:***
If the PUBLIC visibility of these functions is not intended, the visibility keyword must be modified to EXTERNAL.

## A.8 Constant declaration should be preferred

**Line no- 53 to 63**

**Status: CLOSED**

***Description:***
State variables that are not supposed to change throughout the contract should be declared as **constant.**

***Recommendation***:
The following state variables need to be declared as **constant**, unless the current contract design is intended.

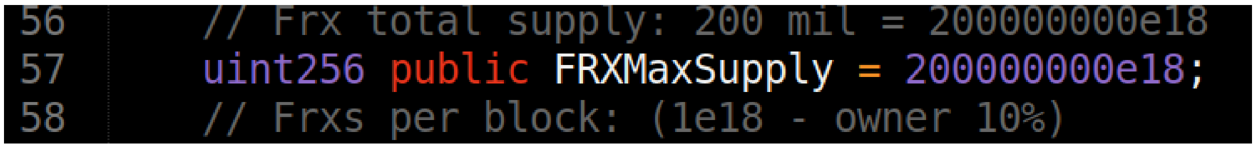
- ***FRXMaxSupply***
- ***FRXPerBlock***
- ***distributionBP***
- ***ownerFRXReward***
- ***startBlock***

### A.9 Too many Digits used

**Line no - 57**

***Description:***

The above-mentioned lines have a large number of digits that makes it difficult to review and reduces the readability of the code



```
56        // Frx total supply: 200 mil = 200000000e18
57        uint256 public FRXMaxSupply = 200000000e18;
58        // Frxs per block: (1e18 - owner 10%)
```

.

***Recommendation:***

Ether Suffix couldshutt be used to symbolize the 10^18 zeros.

## Informational

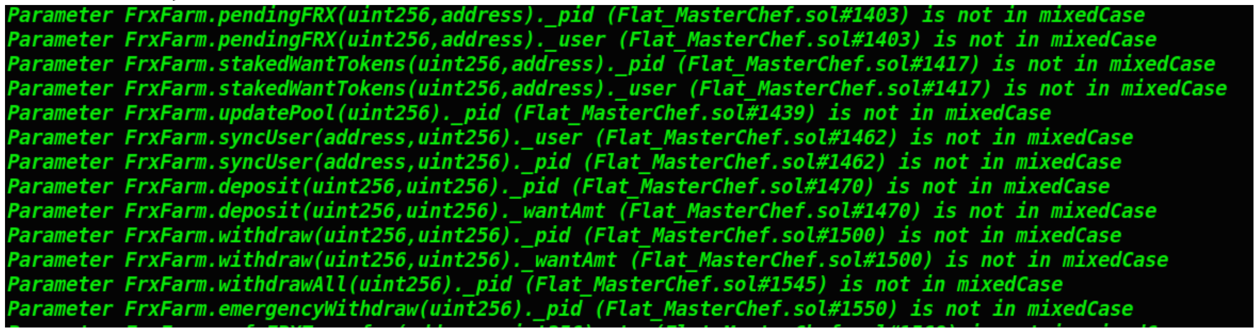
### A.10 Coding Style Issues in the Contract

**Status: Not Considered**

***Explanation:***

Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

During the automated testing, it was found that the FrxFarm contract had quite a few code style issues.



```
Parameter FrxFarm.pendingFRX(uint256,address)._pid (Flat_MasterChef.sol#1403) is not in mixedCase
Parameter FrxFarm.pendingFRX(uint256,address)._user (Flat_MasterChef.sol#1403) is not in mixedCase
Parameter FrxFarm.stakedWantTokens(uint256,address)._pid (Flat_MasterChef.sol#1417) is not in mixedCase
Parameter FrxFarm.stakedWantTokens(uint256,address)._user (Flat_MasterChef.sol#1417) is not in mixedCase
Parameter FrxFarm.updatePool(uint256)._pid (Flat_MasterChef.sol#1439) is not in mixedCase
Parameter FrxFarm.syncUser(address,uint256)._user (Flat_MasterChef.sol#1462) is not in mixedCase
Parameter FrxFarm.syncUser(address,uint256)._pid (Flat_MasterChef.sol#1462) is not in mixedCase
Parameter FrxFarm.deposit(uint256,uint256)._pid (Flat_MasterChef.sol#1470) is not in mixedCase
Parameter FrxFarm.deposit(uint256,uint256)._wantAmt (Flat_MasterChef.sol#1470) is not in mixedCase
Parameter FrxFarm.withdraw(uint256,uint256)._pid (Flat_MasterChef.sol#1500) is not in mixedCase
Parameter FrxFarm.withdraw(uint256,uint256)._wantAmt (Flat_MasterChef.sol#1500) is not in mixedCase
Parameter FrxFarm.withdrawAll(uint256)._pid (Flat_MasterChef.sol#1545) is not in mixedCase
Parameter FrxFarm.emergencyWithdraw(uint256)._pid (Flat_MasterChef.sol#1550) is not in mixedCase
```

**Recommendation:**

Therefore, it is highly recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

## A.11 NatSpec Annotations must be included

**Status: Not Considered**

***Description:***
The smart contracts do not include the NatSpec annotations adequately.

***Recommendation:***
Cover by NatSpec all Contract methods.

## Farming

### Audit Categories and Results:

| No. | Categories | Subitems | Results |
|---|---|---|---|
| 1 | Coding Conventions | ERC20 Token Standards | Pass |
| | | Compiler Version Security | Pass |
| | | Visibility Specifiers | Pass |
| | | Gas Consumption | Pass |
| | | SafeMath Features | Pass |
| | | Fallback Usage | Pass |
| | | tx.origin Usage | Pass |
| | | Deprecated Items | Pass |
| | | Redundant Code | Pass |
| | | Overriding Variables | Pass |
| 2 | Function Call Audit | Authorization of Function Call | Pass |
| | | Low-level Function (call/delegate call) Security | Pass |

| | | | |
|---|---|---|---|
| | | Returned Value Security | Pass |
| | | selfdestruct Function Security | Pass |
| 3 | Business Security | Access Control of Owner | Pass |
| | | Business Logics | Pass |
| | | Business Implementations | Pass |
| 4 | Integer Overflow/underflow | - | Pass |
| 5 | Reentrancy | - | Pass |
| 6 | Exceptional Reachable state | - | Pass |
| 7 | Transaction-Ordering Dependence | - | Pass |
| 8 | Block Properties Dependence | - | Pass |
| 9 | Pseudo-random Number Generator (PRNG) | - | Pass |
| 10 | DoS (Denial of Service) | - | Pass |
| 11 | Token Vesting Implementation | - | N/A |
| 12 | Fake Deposit | - | Pass |
| 13 | Event security | - | Pass |
| 14 | Token Distribution | | Pass |
| 15 | Fees | | Pass |
| 16 | Reward Distribution | | Pass |
| 17 | Pairs | | Pass |