FourRX Finance Smart Contract Preliminary Audit Report

Project Synopsis

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

Contract Details

Total Contract(s)	4
Name of Contract(s)	Pools, SponsorPool, ReferralPool,PercentageCalculator
Language	Solidity
Commit Hash	**f7d395e86028056ba5e88ee50ddbd933a1a0779d**

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	0	0
Medium Severity	1	0
Low Severity	0	0
Informational	2	0
Total Found	0	0

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: FourRX Finance

<u>High Severity Issues</u>
None Found

<u>Medium Severity Issues</u>
None Found

<u>Low Severity Issues</u>
None Found

B. Contract Names: Pool, ReferralPool, SponsorPool, PercentageCalculator

High Severity Issues

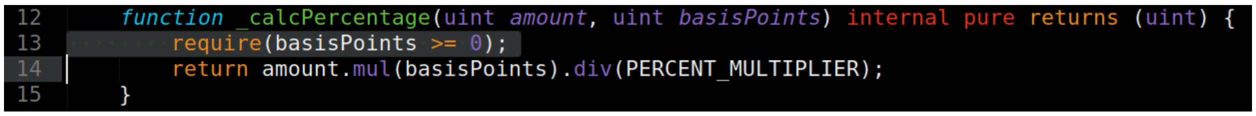
B.1 calcPercentage allows passing ZERO as its second argument

Contract Name - PercentageCalculator

Line no - 13

Description:

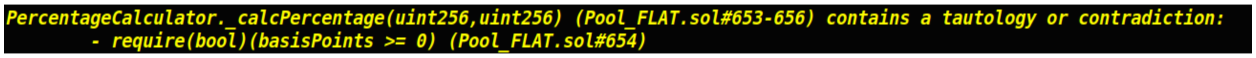
The ***calcPercentage*** function includes a require statement at Line 13 that allows the **basisPoints** arguments to be **equal to ZERO**.



```
12    function _calcPercentage(uint amount, uint basisPoints) internal pure returns (uint) {
13        require(basisPoints >= 0);
14        return amount.mul(basisPoints).div(PERCENT_MULTIPLIER);
15    }
```

Since ***basisPoints*** is an imperative argument and is being used in the calculation of percentage value in multiple instances in the contract, this **require** statement can lead to unwanted scenarios.

During the automated testing of the contract, a similar warning was found as well:



```
PercentageCalculator._calcPercentage(uint256,uint256) (Pool_FLAT.sol#653-656) contains a tautology or contradiction:
	- require(bool)(basisPoints >= 0) (Pool_FLAT.sol#654)
```

Recommendation:

If the above-mentioned function design is not intended, **require statement** of the **calcPercentage** should be updated as follows:

require(basisPoints > 0, “Basis Point cannot be ZERO”)

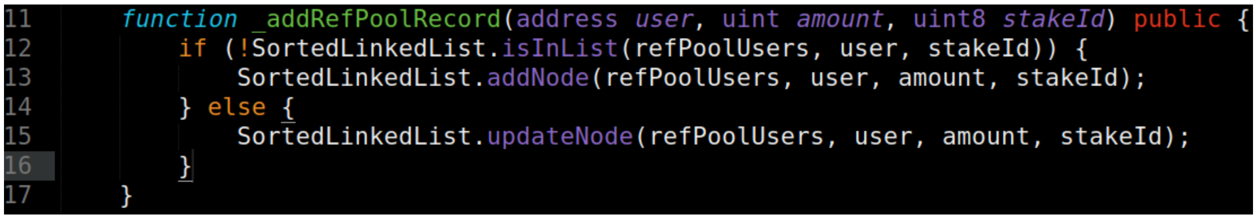
Medium Severity Issues

B.2 Function design of ***_addSponsorPoolRecord*** does not match to ***_addRefPoolRecord***.

Contract: ReferralPool, SponsorPool

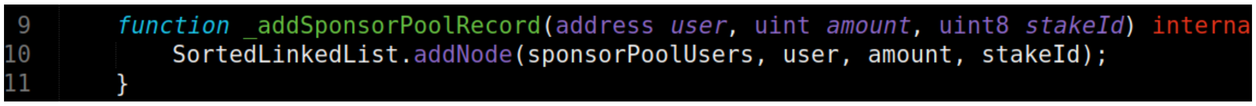
Line no - 11-17, 9-11

The function *_addRefPoolRecord*, in the ReferralPool contract, includes a **IF_ELSE** mechanism to check whether or not the provided information is already present in the list(using SortedLinkedList.isInList function). It only uses the **addNode** function if the information is not present already. Otherwise, it uses the **updateNode** function to store the data.



```
    function _addRefPoolRecord(address user, uint amount, uint8 stakeId) public {
        if (!SortedLinkedList.isInList(refPoolUsers, user, stakeId)) {
            SortedLinkedList.addNode(refPoolUsers, user, amount, stakeId);
        } else {
            SortedLinkedList.updateNode(refPoolUsers, user, amount, stakeId);
        }
    }
```

However, no such IF_ELSE mechanism was found in the _addSponsorPoolRecord function in the ***SponsorPool*** contract despite the fact that both the functions perform almost similar tasks of adding records to their respective pools.



```
    function _addSponsorPoolRecord(address user, uint amount, uint8 stakeId) interna
        SortedLinkedList.addNode(sponsorPoolUsers, user, amount, stakeId);
    }
```

IS THIS INTENDED?

Recommendation:

If the above mentioned scenario is not intended, then the _addSponsorPoolRecord function in SponsorPool contract should also include IF_ELSE mechanism and use updateNode function to store already available information.

Low Severity Issues

None Found

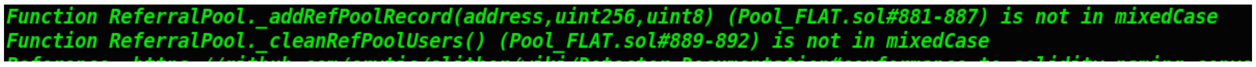
Informational

A.5 Coding Style Issues in the Contract

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 ReferralPool contract had quite a few code style issues.



```
Function ReferralPool._addRefPoolRecord(address,uint256,uint8) (Pool_FLAT.sol#881-887) is not in mixedCase
Function ReferralPool._cleanRefPoolUsers() (Pool_FLAT.sol#889-892) 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.6 NatSpec Annotations must be included

Description:

The smart contracts do not include the NatSpec annotations adequately.

Recommendation:

Cover by NatSpec all Contract methods.