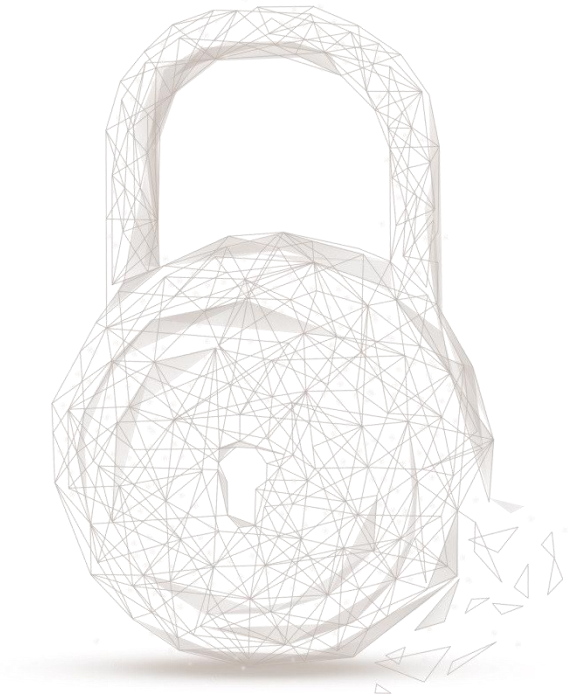
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202012120933

Report Query Name: Farm

Smart Contract Name And Address Link：

None

Start Date：2020.12.10

Completion Date：2020.12.12

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass

		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts OnXFarm, including Coding Standards, Security, and Business Logic. **The OnXFarm contract pass all audit items. The overall result is Pass.** The smart contract is able to function properly.

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.

- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check the weather the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

Check whether the business is secure.

3.1 Business analysis of Contract OnXFarm

(1) add Function

- Description: As shown in Figure 1 below, the contract implements the *add* function to add the Pool. The contract owner can call this function to add the Pool for the user to stake for getting the reward and store the pool-related information.

```
1006    // Add a new token or LP to the pool. Can only be called by the owner.
1007    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do!
1008    function add(
1009        uint256 _allocPoint,
1010        IERC20 _token,
1011        bool _withUpdate
1012    ) public onlyOwner {
1013        if (_withUpdate) {
1014            massUpdatePools();
1015        }
1016        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1017        totalAllocPoint = totalAllocPoint.add(_allocPoint);
1018        poolInfo.push(PoolInfo({
1019          token: _token,
1020          allocPoint: _allocPoint,
1021          lastRewardBlock: lastRewardBlock,
1022          accOnXPerShare: 0
1023        }));
1024    }
```

Figure 1 add Function Source Code

- Related functions: *add*, *massUpdatePools*
- Result: Pass

(2) set Function

- Description: As shown in Figure 2 below, contract implements *set* function to set the reward allocation point of the specified pool, the contract owner can call this function to set the reward allocation point of the specified pool. After the pool reward allocation point is modified, it will affect the value of OnX rewards when users withdraw or deposit tokens.

```
1026    // Update the given pool's OnX allocation point. Can only be called by the owner.
1027    function set(
1028        uint256 _pid,
1029        uint256 _allocPoint,
1030        bool _withUpdate
1031    ) public onlyOwner {
1032        if (_withUpdate) {
1033            massUpdatePools();
1034        }
1035        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
1036        poolInfo[_pid].allocPoint = _allocPoint;
1037    }
```

Figure 2 set Function Source Code

- Related functions: *set*, *massUpdatePools*
- Result: Pass

(3) getTotalRewardInfoInSameCommonDifference Function

- Description: As shown in Figure 3 below, contract implements the *getTotalRewardInfoInSameCommonDifference* function to return reward between the given _from and _to block. The reward decreases in equal steps in each cycle.

```
1039    // (_from, _to)
1040    function getTotalRewardInfoInSameCommonDifference(
1041        uint256 _from,
1042        uint256 _to,
1043        uint256 _onxInitBlock,
1044        uint256 _bulkBlockSize,
1045        uint256 _commonDifference
1046    ) public view returns (uint256 totalReward) {
1047        if (_to <= startBlock || maxRewardBlockNumber <= _from) {
1048            return 0;
1049        }
1050        if (_from < startBlock) {
1051            _from = startBlock;
1052        }
1053        if (maxRewardBlockNumber < _to) {
1054            _to = maxRewardBlockNumber;
1055        }
1056        uint256 currentBulkNumber = _to.sub(startBlock).div(_bulkBlockSize).add(
1057            _to.sub(startBlock).mod(_bulkBlockSize) > 0 ? 1 : 0
1058        );
1059        if (currentBulkNumber < 1) {
1060            currentBulkNumber = 1;
1061        }
1062        uint256 fromBulkNumber = _from.sub(startBlock).div(_bulkBlockSize).add(
1063            _from.sub(startBlock).mod(_bulkBlockSize) > 0 ? 1 : 0
1064        );
1065        if (fromBulkNumber < 1) {
1066            fromBulkNumber = 1;
1067        }
1068        if (fromBulkNumber == currentBulkNumber) {
1069            return _to.sub(_from).mul(_onxInitBlock.sub(currentBulkNumber.sub(1).mul(_commonDifference)));
1070        }
1071        uint256 lastRewardBulkLastBlock = startBlock.add(_bulkBlockSize.mul(fromBulkNumber));
1072        uint256 currentPreviousBulkLastBlock = startBlock.add(_bulkBlockSize.mul(currentBulkNumber.sub(1)));
1073        {
1074            uint256 tempFrom = _from;
1075            uint256 tempTo = _to;
1076            totalReward = tempTo
1077            .sub(tempFrom > currentPreviousBulkLastBlock ? tempFrom : currentPreviousBulkLastBlock)
1078            .mul(_onxInitBlock.sub(currentBulkNumber.sub(1).mul(_commonDifference)));
1079            if (lastRewardBulkLastBlock > tempFrom && lastRewardBulkLastBlock <= tempTo) {
1080                totalReward = totalReward.add(
1081                    lastRewardBulkLastBlock.sub(tempFrom).mul(
1082                        _onxInitBlock.sub(fromBulkNumber > 0 ? fromBulkNumber.sub(1).mul(_commonDifference) : 0)
1083                    )
1084                );
1085            }
1086        }
1087        {
1088            // avoids stack too deep errors
1089            uint256 tempOnXInitBlock = _onxInitBlock;
1090            uint256 tempBulkBlockSize = _bulkBlockSize;
1091            uint256 tempCommonDifference = _commonDifference;
1092            if (currentPreviousBulkLastBlock > lastRewardBulkLastBlock) {
1093                uint256 tempCurrentPreviousBulkLastBlock = currentPreviousBulkLastBlock;
1094                // sum( [fromBulkNumber+1, currentBulkNumber] )
1095                // 1/2 * N *( a1 + aN)
1096                uint256 N = tempCurrentPreviousBulkLastBlock.sub(lastRewardBulkLastBlock).div(tempBulkBlockSize);
1097                if (N > 1) {
1098                    uint256 a1 = tempBulkBlockSize.mul(
1099                        tempOnXInitBlock.sub(
1100                            lastRewardBulkLastBlock.sub(startBlock).mul(tempCommonDifference).div(tempBulkBlockSiz
1101                        )
1102                    );
1103                    uint256 aN = tempBulkBlockSize.mul(
1104                        tempOnXInitBlock.sub(
1105                            tempCurrentPreviousBulkLastBlock.sub(startBlock).div(tempBulkBlockSize).sub(1).mul(
1106                                tempCommonDifference
1107                            )
1108                        )
1109                    );
1110                    totalReward = totalReward.add(N.mul(a1.add(aN)).div(2));
1111                } else {
1112                    totalReward = totalReward.add(
1113                        tempBulkBlockSize.mul(tempOnXInitBlock.sub(currentBulkNumber.sub(2).mul(tempCommonDifferen
1114                    );
1115                }
1116            }
1117        }
1118    }
```

Figure 3 getTotalRewardInfoInSameCommonDifference Function Source code

- Related functions: *getTotalRewardInfoInSameCommonDifference*
- Result: Pass

(4) getTolalRewardInfo Function

- Description: As shown in Figure 4 below, contract implements *getTolalRewardInfo* function to return reward between the given _from and _to block, this function mainly distinguishes before and after *bonusEndBlock*, and chooses different reward calculation parameters according to the difference before and after *bonusEndBlock*.

```
1120    // Return total reward over the given _from to _to block.
1121    function getTotalRewardInfo(uint256 _from, uint256 _to) public view returns (uint256 totalReward) {
1122        if (_to <= bonusEndBlock) {
1123            totalReward = getTotalRewardInfoInSameCommonDifference(
1124                _from,
1125                _to,
1126                onxStartBlock,
1127                bonusBeforeBulkBlockSize,
1128                bonusBeforeCommonDifference
1129            );
1130        } else if (_from >= bonusEndBlock) {
1131            totalReward = getTotalRewardInfoInSameCommonDifference(
1132                _from,
1133                _to,
1134                onxBonusEndBlock,
1135                bonusEndBulkBlockSize,
1136                bonusEndCommonDifference
1137            );
1138        } else {
1139            totalReward = getTotalRewardInfoInSameCommonDifference(
1140                _from,
1141                bonusEndBlock,
1142                onxStartBlock,
1143                bonusBeforeBulkBlockSize,
1144                bonusBeforeCommonDifference
1145            )
1146            .add(
1147                getTotalRewardInfoInSameCommonDifference(
1148                    bonusEndBlock,
1149                    _to,
1150                    onxBonusEndBlock,
1151                    bonusEndBulkBlockSize,
1152                    bonusEndCommonDifference
1153                )
1154            );
1155        }
1156    }
```

Figure 4 getTolalRewardInfo Function Source Code

- Related functions: *getTolalRewardInfo*, *getTotalRewardInfoInSameCommonDifference*
- Result: Pass

(5) updatePool Function

- Description: As shown in Figure 5 below, contract implements *updatePool* function to update pool OnX rewards and information of current block. Any user can call this function to update latest pool OnX rewards and information, and call *mint* function to mint all OnX rewards generated after last block update

to this contract address. 5% of the calculated amount of tokens to be minted will be sent to *devAddr* address, and 2% will be sent to *insAddr* address. In addition, anyone can update all pools at once through calling the *massUpdatePools* function.

```
1180    // Update reward variables of the given pool to be up-to-date.
1181    function updatePool(uint256 _pid) public {
1182        PoolInfo storage pool = poolInfo[_pid];
1183        if (block.number <= pool.lastRewardBlock) {
1184            return;
1185        }
1186        uint256 lpSupply = pool.token.balanceOf(address(this));
1187        if (lpSupply == 0) {
1188            pool.lastRewardBlock = block.number;
1189            return;
1190        }
1191        if (pool.lastRewardBlock >= maxRewardBlockNumber) {
1192            return;
1193        }
1194        uint256 totalReward = getTotalRewardInfo(pool.lastRewardBlock, block.number);
1195        uint256 onxReward = totalReward.mul(pool.allocPoint).div(totalAllocPoint);
1196        onx.mintTo(devAddr, onxReward.div(20)); // 5% OnX sent to dev addr every harvest
1197        onx.mintTo(insAddr, onxReward.div(50)); // 2% OnX sent to insurance fund addr every harvest
1198        onx.mintTo(address(this), onxReward);
1199        pool.accOnXPerShare = pool.accOnXPerShare.add(onxReward.mul(accOnXPerShareMultiple).div(lpSupply));
1200        pool.lastRewardBlock = block.number;
1201    }
```

Figure 5 updatePool Function Source Code

- Related functions: *updatePool*, *getTotalRewardInfo*
- Result: Pass

(6) deposit Function

- Description: As shown in Figure 6 below, the contract implements the *deposit* function for users to stake tokens, the user pre-approves this contract address and then calls this function to deposit tokens(require the pool is exist). Update the pool information when the user is deposited, if the user has previous deposit, calculate the user's previous deposit reward and send the reward to the user address.

```
1203    // Deposit tokens to OnXFarmTest for OnX allocation.
1204    function deposit(uint256 _pid, uint256 _amount) public {
1205        PoolInfo storage pool = poolInfo[_pid];
1206        UserInfo storage user = userInfo[_pid][msg.sender];
1207        updatePool(_pid);
1208        if (user.amount > 0) {
1209            uint256 pending = user.amount.mul(pool.accOnXPerShare).div(accOnXPerShareMultiple).sub(
1210                user.rewardDebt
1211            );
1212            if (pending > 0) {
1213                safeOnXTransfer(msg.sender, pending);
1214            }
1215        }
1216        pool.token.safeTransferFrom(address(msg.sender), address(this), _amount);
1217        user.amount = user.amount.add(_amount);
1218        user.rewardDebt = user.amount.mul(pool.accOnXPerShare).div(accOnXPerShareMultiple);
1219        emit Deposit(msg.sender, _pid, _amount);
1220    }
```

Figure 6 deposit Function Source Code

- Related functions: *deposit, updatePool, safeOnXTransfer, safeTransferFrom*
- Result: Pass

(7) withdraw Function

- Description: As shown in Figure 7 below, the contract implements the *withdraw* function for users to withdraw deposit tokens and OnX rewards, the user can call this function to withdraw the specified amount of deposited tokens and all OnX rewards in the current block. Update pool information when users withdraw deposited tokens and OnX rewards, and transfer the specified deposited tokens and OnX rewards to the user address and update the user deposit information.

```
1222    // Withdraw tokens from OnXFarmTest
1223    function withdraw(uint256 _pid, uint256 _amount) public {
1224        PoolInfo storage pool = poolInfo[_pid];
1225        UserInfo storage user = userInfo[_pid][msg.sender];
1226        require(user.amount >= _amount, 'withdraw: not good');
1227        updatePool(_pid);
1228        uint256 pending = user.amount.mul(pool.accOnXPerShare).div(accOnXPerShareMultiple).sub(
1229            user.rewardDebt
1230        );
1231        if (pending > 0) {
1232            safeOnXTransfer(msg.sender, pending);
1233        }
1234        if (_amount > 0) {
1235            user.amount = user.amount.sub(_amount);
1236            pool.token.safeTransfer(address(msg.sender), _amount);
1237        }
1238        user.rewardDebt = user.amount.mul(pool.accOnXPerShare).div(accOnXPerShareMultiple);
1239        emit Withdraw(msg.sender, _pid, _amount);
1240    }
```

Figure 7 withdraw Function Source Code

- Related functions: *withdraw, updatePool, safeOnXTransfer, safeTransfer*
- Result: Pass

(8) emergencyWithdraw Function

- Description: As shown in Figure 8 below, the contract implements the *emergencyWithdraw* function for users to withdraw deposited tokens. The user can call this function to withdraw all deposited tokens in the pool. Update user deposit information and transfer all deposited tokens to the user address(Note: calling this function cannot get any deposit rewards).

```
1242    // Withdraw without caring about rewards. EMERGENCY ONLY.
1243    function emergencyWithdraw(uint256 _pid) public {
1244        PoolInfo storage pool = poolInfo[_pid];
1245        UserInfo storage user = userInfo[_pid][msg.sender];
1246        pool.token.safeTransfer(address(msg.sender), user.amount);
1247        emit EmergencyWithdraw(msg.sender, _pid, user.amount);
1248        user.amount = 0;
1249        user.rewardDebt = 0;
1250    }
```

Figure 8 emergencyWithdraw Function Source Code

- Related functions: *emergencyWithdraw, safeTransfer*
- Result: Pass

(9) pendingOnX function

- Description: As shown in Figure 9 below, the contract implements the *pendingOnX* function for users to query the number of OnX rewards that can be obtained.

```
1158    // View function to see pending OnX on frontend.
1159    function pendingOnX(uint256 _pid, address _user) external view returns (uint256) {
1160        PoolInfo storage pool = poolInfo[_pid];
1161        UserInfo storage user = userInfo[_pid][_user];
1162        uint256 accOnXPerShare = pool.accOnXPerShare;
1163        uint256 lpSupply = pool.token.balanceOf(address(this));
1164        if (block.number > pool.lastRewardBlock && lpSupply != 0 && pool.lastRewardBlock < maxRewardBlockNumber) {
1165            uint256 totalReward = getTotalRewardInfo(pool.lastRewardBlock, block.number);
1166            uint256 onxReward = totalReward.mul(pool.allocPoint).div(totalAllocPoint);
1167            accOnXPerShare = accOnXPerShare.add(onxReward.mul(accOnXPerShareMultiple).div(lpSupply));
1168        }
1169        return user.amount.mul(accOnXPerShare).div(accOnXPerShareMultiple).sub(user.rewardDebt);
1170    }
```

Figure 9 pendingOnX Function Source Code

- Related functions: *pendingOnX, getTotalRewardInfo*
- Result: Pass

(10) Parameter Related Function

- Description: As shown in Figures 10 and 11, the contract implements the functions *changeDevAddr and changeInsuranceAddr* to set pool related parameters. The contract owner can call these functions to set the *devAddr* and *insAddr*.

```
1262    // Update dev address by the previous dev or governance
1263    function changeDevAddr(address _devAddr) public {
1264        require(msg.sender == devAddr, 'bruh');
1265        devAddr = _devAddr;
1266    }
```

Figure 10 changeDevAddr Function Source Code

```
1268    // Update insurance address by previous dev or governance
1269    function changeInsuranceAddr(address _insAddr) public {
1270        require(msg.sender == insAddr, 'nah');
1271        insAddr = _insAddr;
1272    }
```

Figure 11 changeInsuranceAddr Function Source Code

- Related functions: *changeDevAddr*, *changeInsuranceAddr*
- Result: Pass

4. Conclusion

Beosin(Chengdu LianAn) conducted a detailed audit on the design and code implementation of the smart contracts OnXFarm. All problems found during the audit have been notified to the project party, and the project party believes that no repair is needed. The overall audit result of the smart contracts OnXFarm is **Pass**.

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com