StakingRewards Smart Contract Preliminary Audit Report

Project Synopsis

Project Name	**MahaDao Audit**
Platform	Ethereum, Solidity
Github Repo	https://github.com/MahaDAO/arthcoin-v2/blob/master/contracts/Staking/StakingRewards.sol
Deployed Contract	Not Deployed
Total Duration	15 Days
Timeline of Audit	**15th April 2021 to 02nd April 2021**

Contract Details

Total Contract(s)	1
Name of Contract(s)	StakingRewards
Language	Solidity
Commit Hash	**8bcf83f8d6a3d5675d400ec63acbf079ba638bed**

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	1	0
Medium Severity	2	0
Low Severity	6	0
Informational	2	0
Total Found	11	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: ArthPool

High Severity Issues

A.1 _arthConroller is never initialized

Line no - 56, 458

Explanation:

The ***_arthController*** state variable is never initialized throughout the contract.

```
55      IARTHController private _arthController;
56
```

However, it is used in the ***crBoostMultiplier*** function at line **458** to call the ***getGlobalCollateralRatio*** function.

Since the **_arthController** is never initialized, it will lead to an unexpected scenario that will adversely affect the intended behaviour of the function.

```
453     function crBoostMultiplier() public view returns (uint256) {
454         uint256 multiplier =
455             uint256(_MULTIPLIER_BASE).add(
456                 (
457                     uint256(_MULTIPLIER_BASE).sub(
458                         _arthController.getGlobalCollateralRatio()
459                     )
460                 )
461                     .mul(crBoostMaxMultiplier.sub(_MULTIPLIER_BASE))
462                     .div(_MULTIPLIER_BASE)
463             );
464         return multiplier;
465     }
```

Recommendation:

_arthController must be initialized adequately before being used in a particular function.

Medium Severity Issues

A.2 State Variables Updated After External Call. Violation of Check_Effects_Interaction Pattern

Line no -524-539, 576-593,497-498, 277-279,208-209, 263-265

Explanation:

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

However, during the automated testing, it was found that some of the functions in the StakingRewards contract violate this ***Check-Effects-Interaction*** pattern at the above-mentioned lines.

Recommendation:

Modification of any State Variables must be performed before making an external call. **Check Effects Interaction Pattern** must be followed while implementing external calls in a function.

A.3 for Loop in *withdrawLocked* function is extremely costly

Line no - 227

Description:

The ***for loop*** in the **withdrawLocked function** includes state variables like **.length** of a non-memory array in the condition of the for loops.

```
218     function withdrawLocked(bytes32 kekId)
219         external
220         override
221         nonReentrant
222         updateReward(msg.sender)
223     {
224         LockedStake memory thisStake;
225         thisStake.amount = 0;
226         uint256 theIndex;
227         for (uint256 i = 0; i < _lockedStakes[msg.sender].length; i++) {
228             if (kekId == _lockedStakes[msg.sender][i].kekId) {
229                 thisStake = _lockedStakes[msg.sender][i];
230                 theIndex = i;
231                 break;
232             }
233         }
```

As a result, these state variables consume a lot more extra gas for every iteration of the loop.

Recommendation:

It's quite effective to use a local variable instead of a state variable like .***length*** in a loop.

For instance,

```
local_variable = _lockedStakes[msg.sender].length;
 for (uint256 i = 0; i < local_variable; i++) {
        if (kekId == _lockedStakes[msg.sender][i].kekId) {
           thisStake = _lockedStakes[msg.sender][i];
           theIndex = i;
           break;
        }
     }
```

Low Severity Issues

A.4 getReward function should include require statement instead of IF-Else Statement

Line no: 495-499

Explanation

The **getReward** function includes an **if statement** at the very beginning of the function to check whether or not the ***reward amount*** for a particular user is more than Zero.

Most importantly, this is a strict check and the function body is only executed if this **IF statement** holds true.

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

```
493     function getReward() public override nonReentrant updateReward
494         uint256 reward = rewards[msg.sender];
495         if (reward > 0) {
496             rewards[msg.sender] = 0;
497             rewardsToken.transfer(msg.sender, reward);
498             emit RewardPaid(msg.sender, reward);
499         }
500     }
```

Recommendation

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

For instance,

require(reward > 0,"Error MSG:Reward Amount for this address is ZERO");

A.5 External Visibility should be preferred

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:

- ***lockedBalanceOf***
- ***getReward***

Recommendation:

If the public visibility of these functions is not intended, the visibility keyword must be modified to external.

A.6 Comparison to boolean Constant

Line no: 237, 521, 549

Description:

Boolean constants can directly be used in conditional statements or require statements.
Therefore, it's not considered a better practise to explicitly use **TRUE or FALSE** in the **require** statements.

```
235         require(
236             block.timestamp >= thisStake.endingTimestamp ||
237                 isLockedStakes == true,
238             'Stake is still locked!'
239         );
240
```

Recommendation:

The equality to boolean constants must be removed from the above-mentioned line.

A.7 Return Value of an External Call is never used Effectively

Line no - 277, 497

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 StakingRewards contract never uses these return values throughout the contract.

```
275         require(tokenAddress != address(stakingToken));
276
277         IERC20(tokenAddress).transfer(ownerAddress, tokenAmount);
278
```

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

A.8 No Events emitted after imperative State Variable modification

Line no - 356, 360

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 ***ownerAddress, timelockAddress, rewardRate*** etc, in the StakingReward contract but do not emit an event after that:

- ***setOwnerAndTimelock***
- ***setRewardRate***

Since there is no event emitted on updating this variable, it might be difficult to track it off-chain.

Recommendation:
An event should be fired after updating the ***rewardRate*** variable.

A.9 Absence of Error messages in Require Statements

Line no - 275

Description:
The ***recoverERC20*** includes a **require** statement in the StakingRewards.sol contract that does not include an error message.

```
270     function recoverERC20(address tokenAddress, uint256 tokenAmount)
271         external
272         onlyByOwnerOrGovernance
273     {
274         // Admin cannot withdraw the staking token from the contract
275         require(tokenAddress != address(stakingToken));
```

While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

Recommendation:
Error Messages must be included in every require statement in the contract

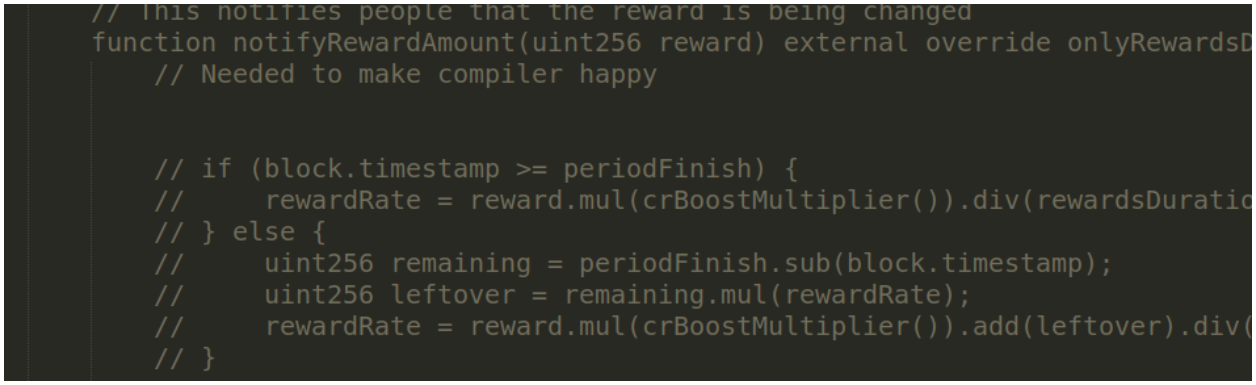
Informational

A.10 NatSpec Annotations must be included

Description:
A smart contract does not include the NatSpec annotations adequately.
Recommendation:
Cover by NatSpec all Contract methods.

A.11 Commented codes must be wiped out before deployment

Explanation
The StakingReward.sol contract includes quite a few commented codes at the end of the contract.
This badly affects the readability of the code.



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.

AutoMated Test Results

```
        - Ownable.transferOwnership(address) (FlatStakes.sol#144-151)
grantRole(bytes32,address) should be declared external:
        - AccessControl.grantRole(bytes32,address) (FlatStakes.sol#1754-1761)
revokeRole(bytes32,address) should be declared external:
        - AccessControl.revokeRole(bytes32,address) (FlatStakes.sol#1772-1779)
renounceRole(bytes32,address) should be declared external:
        - AccessControl.renounceRole(bytes32,address) (FlatStakes.sol#1795-1806)
lockedBalanceOf(address) should be declared external:
        - StakingRewards.lockedBalanceOf(address) (FlatStakes.sol#2362-2364)
getReward() should be declared external:
        - StakingRewards.getReward() (FlatStakes.sol#2387-2394)
```

```
Reentrancy in StakingRewards._stake(address,uint256) (FlatStakes.sol#2408-2434):
        External calls:
        - TransferHelper.safeTransferFrom(address(stakingToken),msg.sender,address(this),amount) (FlatStakes.sol
        State variables written after the call(s):
        - _boostedBalances[who] = _boostedBalances[who].add(amount) (FlatStakes.sol#2431)
        - _stakingTokenBoostedSupply = _stakingTokenBoostedSupply.add(amount) (FlatStakes.sol#2427)
        - _stakingTokenSupply = _stakingTokenSupply.add(amount) (FlatStakes.sol#2426)
Reentrancy in StakingRewards._stakeLocked(address,uint256,uint256) (FlatStakes.sol#2436-2488):
        External calls:
        - TransferHelper.safeTransferFrom(address(stakingToken),msg.sender,address(this),amount) (FlatStakes.sol
        State variables written after the call(s):
        - _boostedBalances[who] = _boostedBalances[who].add(boostedAmount) (FlatStakes.sol#2485)
        - _stakingTokenBoostedSupply = _stakingTokenBoostedSupply.add(boostedAmount) (FlatStakes.sol#2479-2481)
        - _stakingTokenSupply = _stakingTokenSupply.add(amount) (FlatStakes.sol#2478)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
StakingRewards.withdrawLocked(bytes32).theIndex (FlatStakes.sol#2120) is a local variable never initialized
StakingRewards.withdrawLocked(bytes32).thisStake (FlatStakes.sol#2118) is a local variable never initialized
```

```
StakingRewards.withdrawLocked(bytes32).theIndex (FlatStakes.sol#2120) is a local variable never initialized
StakingRewards.withdrawLocked(bytes32).thisStake (FlatStakes.sol#2118) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
StakingRewards.recoverERC20(address,uint256) (FlatStakes.sol#2164-2174) ignores return value by IERC20(tokenAddress).transfer(ownerAddress,tokenAmount
) (FlatStakes.sol#2171)
StakingRewards.getReward() (FlatStakes.sol#2387-2394) ignores return value by rewardsToken.transfer(msg.sender,reward) (FlatStakes.sol#2391)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
StakingRewards.withdrawLocked(bytes32) (FlatStakes.sol#2112-2161) compares to a boolean constant:
        -require(bool,string)(block.timestamp >= thisStake.endingTimestamp || isLockedStakes == true,Stake is s
)
StakingRewards._stake(address,uint256) (FlatStakes.sol#2408-2434) compares to a boolean constant:
        -require(bool,string)(greylist[who] == false,address has been greylisted) (FlatStakes.sol#2415)
StakingRewards._stakeLocked(address,uint256,uint256) (FlatStakes.sol#2436-2488) compares to a boolean constant:
        -require(bool,string)(greylist[who] == false,address has been greylisted) (FlatStakes.sol#2443)
```