THE ARCADIA GROUP

Audit of The FaaSPool ValueDeFi Contracts

a report of findings by

Van Cam Pham, PhD

innovative fortuna iuvat

November 4th, 2020

Table of Contents

Document Info

Client	ValueDeFi Finance
Title	Smart Contract Audit of FaaSPool Contracts
Auditors	Van Cam Pham, PhD
Reviewed By	Joel Farris
Approved By	Rasikh Morani

Contact

For more information on this report, contact The Arcadia Media Group Inc.

Rasikh Morani
(972) 543-3886
rasikh@arcadiamgroup.com
https://t.me/thearcadiagroup

Executive Summary

A Representative Party of the ValueDeFi Finance ("ValueDeFi") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following FaaSPool smart contracts on the ValueDeFi repo at Commit #f79b7baf41cf2d1347336e067a9d2c1f00718385.

FaaSPool.sol

Arcadia completed the review using various methods primarily consisting of dynamic and static analysis. This process included a line by line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

Findings

1. Using `uint256/uint` instead of `uint8`

- FAAS-1
- Severity: Medium
- Likelihood: Low
- Impact: Low

- Target: FaaSPool.sol
- Category: Low
- Finding Type: Static
- Lines: 21-24

In the FaaSPool contract, the data structure `UserInfo` takes `uint8` data type for typing pool ID. This would cause overflow, thus reverting rewarding transactions, if there are more than 256 pools in the pool.

```
 struct UserInfo {
        uint amount;
        mapping(uint8 => uint) rewardDebt;
        mapping(uint8 => uint) accumulatedEarned; // will accumulate every time
user harvest
        mapping(uint8 => uint) lockReward;
        mapping(uint8 => uint) lockRewardReleased;
        uint lastStakeTime;
    }
```

Action Recommended: While it seems impractical to have more than 256 pools, the data structure should use `uint256/uint` type for pool identification indexing. Intuitively, the use of uint8 can save gas cost for contract deployment, but in reality, the use of `uint8` and `uint256/uint` takes up the same memory space in the deployment contract. It is then recommended to change `uint8` to `uint256/uint`.

```
struct UserInfo {
        uint amount;
        mapping(uint => uint) rewardDebt;
        mapping(uint => uint) accumulatedEarned; // will accumulate every time
user harvest
        mapping(uint => uint) lockReward;
        mapping(uint => uint) lockRewardReleased;
        uint lastStakeTime;
```

```
    }
```

Review of the issue: We have reviewed the deployed FaaSPool contract on etherscan, and verified that the issue was resolved by the team.

2. Input parameter value range checking

- FAAS-2
- Severity: Medium
- Impact: Low
- Target: FaaSPool.sol
- Category: Informational
- Finding Type: Static
- Lines: 76-95, 97-103

Functions addRewardPool should have value range checking for input parameters `_rewardPerBlock` and `_lockRewardPercent`. Even though `rewardPerBlock` can be adjusted per pool via function `updateRewardPool`, having value range checking in the function is still recommended in order to avoid an unexpectedly high inflation. Furthermore, `_lockRewardPercent` should at least be less than 100. Any manual input mistake that puts that value for this parameter above 100 would lead to incorrect functioning of the contract.

A similar issue is found in function `updateRewardPool`, which updates the `_rewardPerBlock` and `_endRewardBlock`.

- `_endRewardBlock` in function `updateRewardPool` should be checked for its value range greater than the current block number. Any mistake that puts `_endRewardBlock` greater than block number will cease the rewarding in the updated pool.

```
    function addRewardPool(IERC20 _rewardToken, uint256 _startBlock, uint256 _endRewardBlock, uint256 
_rewardPerBlock,
        uint256 _lockRewardPercent, uint256 _startVestingBlock, uint256 _endVestingBlock) public 
onlyController {
        require(_startVestingBlock <= _endVestingBlock, "sVB>eVB");
        _startBlock = (block.number > _startBlock) ? block.number : _startBlock;
        require(_startBlock < _endRewardBlock, "sB>=eB");
        updateReward();
        rewardPoolInfo.push(RewardPoolInfo({
            rewardToken : _rewardToken,
            lastRewardBlock : _startBlock,
            endRewardBlock : _endRewardBlock,
            rewardPerBlock : _rewardPerBlock,
            accRewardPerShare : 0,
            lockRewardPercent : _lockRewardPercent,
            startVestingBlock : _startVestingBlock,
            endVestingBlock : _endVestingBlock,
            numOfVestingBlocks: _endVestingBlock - _startVestingBlock,
            totalPaidRewards: 0,
            totalLockedRewards: 0
            }));
    }

function updateRewardPool(uint8 _pid, uint256 _endRewardBlock, uint256 _rewardPerBlock) public
onlyController {
        updateReward(_pid);
        RewardPoolInfo storage rewardPool = rewardPoolInfo[_pid];
        require(block.number <= rewardPool.endRewardBlock, "late");
        rewardPool.endRewardBlock = _endRewardBlock;
        rewardPool.rewardPerBlock = _rewardPerBlock;
    }
```

Action Recommended: Add value range checking for the input parameters in the two functions `addRewardPool` and `updateRewardPool`.

Review at commit #454f7ffe97da1d9285f1ee8958819e3561846b40: The issue was fixed by the team.

3. Gas cost in function updateRewardPool

- FAAS-3
- Severity: Low
- Likelihood: Low
- Impact: Low

- Target: FaaSPool.sol
- Category: Informational
- Finding Type: Static
- Lines 97-103

In the function, `updateRewardPool`, the function will revert if the block number (at transaction time) is greater than the pool reward end block. If so, the transaction still costs gas for executing function `updateReward` even though any state update in the function `updateReward` will be reverted as the whole transaction will be reverted.

In order to minimize gas cost In this case scenario, the statement `require(block.number <= rewardPool.endRewardBlock, "late");` should be called before the function `updateReward` gets executed.

```
function updateRewardPool(uint8 _pid, uint256 _endRewardBlock, uint256 _rewardPerBlock) public
onlyController {
        updateReward(_pid);
```

```
       RewardPoolInfo storage rewardPool = rewardPoolInfo[_pid];
       require(block.number <= rewardPool.endRewardBlock, "late");
       rewardPool.endRewardBlock = _endRewardBlock;
       rewardPool.rewardPerBlock = _rewardPerBlock;
   }
```

Action Recommended: Rearrange the function to put the `require` statement before the `updateReward` function call.

```
function updateRewardPool(uint8 _pid, uint256 _endRewardBlock, uint256 _rewardPerBlock) public
onlyController {
       RewardPoolInfo storage rewardPool = rewardPoolInfo[_pid];
       require(block.number <= rewardPool.endRewardBlock, "late");
       require(_endRewardBlock > block.number, "reward ceasing");
       updateReward(_pid);
       rewardPool.endRewardBlock = _endRewardBlock;
       rewardPool.rewardPerBlock = _rewardPerBlock;
   }
```

Review at commit #454f7ffe97da1d9285f1ee8958819e3561846b40: The issue was fixed by the team.

4. Function pendingReward does not take into account locked rewards

- FAAS-4
- Severity: low
- Likelihood: Low
- Impact: Low

- Target: FaaSPool.sol
- Category: Low
- Finding Type: Dynamic
- Lines: 105-118

In the FaaSPool contract, function pendingReward currently does not take into account user locked rewards. In practice, a user would like to check the total rewards that the user will receive when the user calls that function.

```
 function pendingReward(uint8 _pid, address _account) public override view returns (uint _pending) {
       UserInfo storage user = userInfo[_account];
       RewardPoolInfo storage rewardPool = rewardPoolInfo[_pid];
       uint _accRewardPerShare = rewardPool.accRewardPerShare;
       uint lpSupply = balanceOf(address(this));
       uint _endRewardBlockApplicable = block.number > rewardPool.endRewardBlock ?
rewardPool.endRewardBlock : block.number;
       if (_endRewardBlockApplicable > rewardPool.lastRewardBlock && lpSupply != 0) {
           uint _numBlocks = _endRewardBlockApplicable.sub(rewardPool.lastRewardBlock);
```

```
            uint _incRewardPerShare = _numBlocks.mul(rewardPool.rewardPerBlock).mul(1e18).div(lpSupply);
            _accRewardPerShare = _accRewardPerShare.add(_incRewardPerShare);
        }
        _pending = user.amount.mul(_accRewardPerShare).div(1e18).sub(user.rewardDebt[_pid]);
    }
```

Action Recommended: The function should return pending reward and locked rewards.

Review at commit #454f7ffe97da1d9285f1ee8958819e3561846b40: As discussed with the team, this is intended as there is a separate function for reading locked rewards.

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.