THE ARCADIA GROUP

Audit of VOX.Finance

Audit of The Vox Finance Contracts

a report of findings by

Van Cam Pham, PhD

innovative fortuna iuvat

October 30th, 2020

Table of Contents

Document Info

Client	Vox Finance
Title	Smart Contract Audit of Vox 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 Vox Finance ("Vox") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following Vox Finance smart contracts on the Vox Finance repo at Commit #8777a0dfbdb125c27880ea8df55840c4cfbcc73a.

VoxDevTimelock.sol
VoxPopuliToken.sol
Timelock.sol
VoxMaster.sol
VoxToken.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.

Conclusion

While most of the findings do not require any immediate action, some may require additional disclosure and communication to the end-users for clarity, additional code review and audits should be completed before launch to be maximally effective and to lower end-user risk.

Findings

1. Constructor input parameters

- VOX-1
- Severity: Low
- Likelihood: Low
- Impact: Low

- Target: VoxToken.sol
- Category: Low
- Finding Type: Dynamic
- Lines: 26-28

In the VoxToken contract, the function `setBurnrate` takes `burnrate_` as its input. This input parameter should be checked for its value range. This is to avoid any mistake when making a transaction to set the burn rate of the VoxToken contract.

```
 function setBurnrate(uint8 burnrate_) public onlyOwner {
        _setupBurnrate(burnrate_);
    }
```

Action Recommended: Add checking for the value range of burn rate in the `setBurnrate` function.

```
function setBurnrate(uint8 burnrate_) public onlyOwner {
       require(minBurnrate <= burnrate_ && burnrate_ <= maxBurnrate, "burn rate
must be in valid range");
    }
```

2. Events for add and remove from the whitelist

- VOX-2
- Severity: Medium
- Impact: Low

- Target: VoxToken.sol
- Category: Informational
- Finding Type: Static
- Lines: 30-36

Functions addWhitelistedAddress and removeWhitelistedAddress should have events emitted. This is to ease the listing of which addresses are added/removed from the whitelisted addresses as those addresses are currently stored in a mapping.

```
    function addWhitelistedAddress(address _address) public onlyOwner {
        _whitelistedAddresses[_address] = true;
    }

    function removeWhitelistedAddress(address _address) public onlyOwner {
        _whitelistedAddresses[_address] = false;
    }
```

Action Recommended: Add events to the VoxToken contract, and emit an event instance whenever an address is added or removed from the whitelisted address list. As the VoxToken contract is already deployed, it's a good practice if the team can publish the list of whitelisted addresses to the community.

3. Contract owner and minter

- VOX-3
- Severity: Medium
- Likelihood: Low
- Impact: Low

- Target: VoxPopuliToken.sol
- Category: Informational
- Finding Type: Static

The owner of VoxPopuliToken should be transferred to VoxMaster right after VoxPopuliToken is deployed.

Action Recommended: The VoxPopuliToken is already live. The issue can be considered as mitigated as the current owner in the mainnet is VoxMaster.

4. Input parameter value range validation

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

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

In the VoxMaster contract, the input parameters for the contract constructor should have value range check `_startBlock` and `_bonusEndBlock`. This is a good common standard to avoid bad input parameters that could lead to redeployment of the contract, which is costly.

```
constructor(
        VoxToken _vox,
        VoxPopuliToken _populi,
        address _devaddr,
        uint256 _voxPerBlock,
        uint256 _startBlock,
        uint256 _bonusEndBlock
    ) public {
        vox = _vox;
        populi = _populi;
        devaddr = _devaddr;
        voxPerBlock = _voxPerBlock;
        bonusEndBlock = _bonusEndBlock;
        startBlock = _startBlock;

        // staking pool
        poolInfo.push(PoolInfo({
            lpToken: _vox,
            allocPoint: 1000,
            lastRewardBlock: startBlock,
            accVoxPerShare: 0
        }));

        totalAllocPoint = 1000;
    }
```

Action Recommended: Adding input parameter value range validation for `_startBlock` and `_bonusEndBlock` should solve the issue.

5. Gas cost for function `updateStakingPool`

- VOX-5
- Severity: Medium
- Likelihood: Low
- Impact: Low
- Target: VoxMaster.sol
- Category: Medium
- Finding Type: Static
- Lines: 175, 193, 197-208

In the VoxMaster contract, the `updateStakingPool` function is to adjust the allocation point for the VOX staking pool (pool Id 0) to maintain the staking pool reward at 25% of the total farming rewards. However, the use of this function is gas costly as it has to loop over all the pools every time a pool is added or updated.

```
function updateStakingPool() internal {
        uint256 length = poolInfo.length;
        uint256 points = 0;
        for (uint256 pid = 1; pid < length; ++pid) {
            points = points.add(poolInfo[pid].allocPoint);
        }
        if (points != 0) {
            points = points.div(3);
            totalAllocPoint = totalAllocPoint.sub(poolInfo[0].allocPoint).add(points);
            poolInfo[0].allocPoint = points;
        }
    }
```

Action Recommended: The function only adjusts the total allocation point of all pools and allocation point for pool 0. This can be done by adjusting these values in the functions `set` and `add` pools. The following code is a suggestion for adjusting the function `add` to update the allocation point for pool 0 without looping over all other pools.

```
function add(
        uint256 _allocPoint,
        IERC20 _lpToken,
        bool _withUpdate
    ) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock
            ? block.number
            : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(
            PoolInfo({
                lpToken: _lpToken,
                allocPoint: _allocPoint,
                lastRewardBlock: lastRewardBlock,
                accVoxPerShare: 0
            })
        );
        uint256 sumOfOthers = totalAllocPoint.sub(poolInfo[0].allocPoint);
        if (sumOfOthers != 0) {
            uint256 adjustedPoint = sumOfOthers.div(3);
            totalAllocPoint = totalAllocPoint.sub(poolInfo[0].allocPoint).add(adjustedPoint);
            poolInfo[0].allocPoint = adjustedPoint;
        }
    }
```

6. Migration might fail

- VOX-6
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: VoxMaster.sol
- Category: LP Token migration
- Finding Type: Dynamic
- Lines: 220-229

In the VoxMaster contract, the function `migrate` is to migrate the liquidity token from VoxMaster to another contract. After the transferring of liquidity token, the function verifies that the balance of liquidity token in the destination contract must be equal to the migrated liquidity token amount. Line 227 can fail if any one sends any liquidity token amount to the destination contract before the migration transaction gets executed.

```
function migrate(uint256 _pid) public {
       require(address(migrator) != address(0), "migrate: no migrator");
       PoolInfo storage pool = poolInfo[_pid];
       IERC20 lpToken = pool.lpToken;
       uint256 bal = lpToken.balanceOf(address(this));
```

```
        lpToken.safeApprove(address(migrator), bal);
        IERC20 newLpToken = migrator.migrate(lpToken);
        require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");
        pool.lpToken = newLpToken;
    }
```

Action Recommended: The function can mitigate the issue by checking the difference between the balance of liquidity token before and after the migrate function gets called.

Here’s a possible solution.

As VoxMaster is already deployed, before making the migration transaction, the team should ensure that there is no liquidity token sent to VoxMaster in the new liquidity token contract.

```
function migrate(uint256 _pid, IERC20 newLpToken) public {
        require(address(migrator) != address(0), "migrate: no migrator");
        PoolInfo storage pool = poolInfo[_pid];
        IERC20 lpToken = pool.lpToken;
        uint256 bal = lpToken.balanceOf(address(this));
        lpToken.safeApprove(address(migrator), bal);
        uint256 balBefore = newLpToken.balanceOf(address(this));
        migrator.migrate(lpToken);
        uint256 balAfter = newLpToken.balanceOf(address(this));
        require(bal == balAfter.sub(balBefore), "migrate: bad");
        pool.lpToken = newLpToken;
    }
```

7. Function `setDevFundDivRate` should have value range validity.

- VOX-7
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Target: VoxMaster.sol
- Category: Input parameters
- Finding Type: Static
- Lines: 405-430

In the `VoxMaster` contract, the function `setDevFundDivRate` should have a value range validity check to avoid any mistake that could result in `devFundDivRate` too low, which would cause high inflation for the development fund.

```
function setDevFundDivRate(uint256 _devFundDivRate) public onlyOwner {
        require(_devFundDivRate > 0, "!devFundDivRate-0");
        devFundDivRate = _devFundDivRate;
```

```
    }
```

Action Recommended: Add a value range validity check that has a minimum and maximum value for `devFundDivRate`.

8. Development fund reward is not part of the amount of Vox minted per block

- VOX-8
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Target: VoxMaster.sol
- Category: Development Fund Reward
- Finding Type: Static
- Lines: 269-290

In the `VoxMaster` contract, the function `updatePool` updates the rewards for a pool based on the last reward block, the current block, and the amount of VOX minted per block. The development fund reward is also minted based on the development fund rate and the amount of VOX minted per block.
As a common practice in most farming projects, the development fund reward should be pulled from the amount of VOX minted per block. However, this is not the case in the function `updatePool`, which generates development fund reward as an additional VOX amount minted per block. This means the VOX minted per block in parameter `voxPerBlock` is not accounting for the development fund.

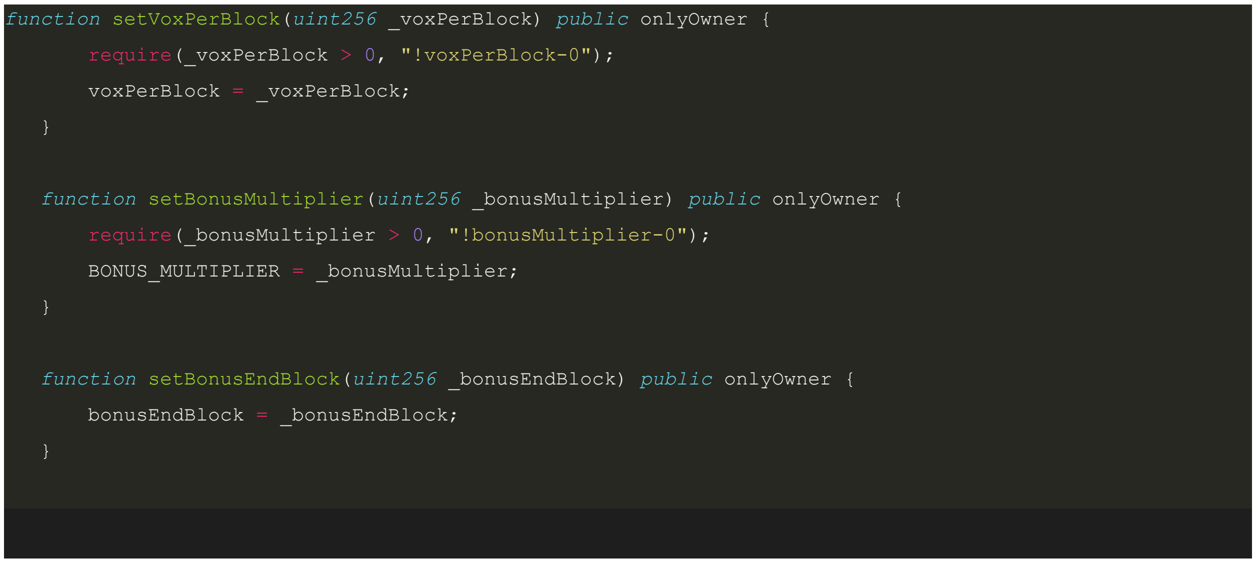
```
uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
        uint256 voxReward = multiplier
            .mul(voxPerBlock)
            .mul(pool.allocPoint)
            .div(totalAllocPoint);
        vox.mint(devaddr, voxReward.div(devFundDivRate));
        vox.mint(address(this), voxReward);
```

Action Recommended: If this is the tokenomics specification of Vox token, no action is needed. However, the Vox Finance team should publish to the community that the parameter `voxPerBlock` in the `VoxMaster` contract does not account for the development fund. It's also good if the actual amount of VOX minted per block (development fund included) is shown on the VOX Finance website.

9. Function `setVoxPerBlock, setBonusMultiplier, setBonusEndBlock` should have value range validity.

- VOX-9
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: VoxMaster.sol
- Category: Input parameters
- Finding Type: Static
- Lines: 564-567, 569-572, 574-576

This issue is similar to VOX-7. Those functions should have a value range validity check for input parameters.



```
function setVoxPerBlock(uint256 _voxPerBlock) public onlyOwner {
        require(_voxPerBlock > 0, "!voxPerBlock-0");
        voxPerBlock = _voxPerBlock;
    }

    function setBonusMultiplier(uint256 _bonusMultiplier) public onlyOwner {
        require(_bonusMultiplier > 0, "!bonusMultiplier-0");
        BONUS_MULTIPLIER = _bonusMultiplier;
    }

    function setBonusEndBlock(uint256 _bonusEndBlock) public onlyOwner {
        bonusEndBlock = _bonusEndBlock;
    }
```

Action Recommended: Add a value range validity check that has a minimum and maximum value for those functions.

10. Function `claim` can save gas cost

- VOX-7
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: VoxMaster.sol
- Category: Input parameters
- Finding Type: Static
- Lines: 405-430

In the `VoxMaster` contract, the function `claim` can save gas costs a bit by using a local variable for user pending rewards. In the current code, a non-zero value is written to the data storage `pendingRewards` of users. The data field is reset to 0 right after transferring the reward to the user. Writing a non-zero value to storage can cost high gas while writing a zero value to storage is much cheaper in terms of gas consumption. Function `claimAndStake` also has the same issue.

```
function claim(uint256 _pid) public {
      require(_pid != 0, "please claim staking rewards on stake page");
      PoolInfo storage pool = poolInfo[_pid];
      UserInfo storage user = userInfo[_pid][msg.sender];
      updatePool(_pid);
      uint256 pending = user.amount.mul(pool.accVoxPerShare).div(1e12).sub(
          user.rewardDebt
      );
      if (pending > 0 || user.pendingRewards > 0) {
          user.pendingRewards = user.pendingRewards.add(pending);
          safeVoxTransfer(msg.sender, user.pendingRewards);
          emit Claim(msg.sender, _pid, user.pendingRewards);
          user.pendingRewards = 0;
      }
      user.rewardDebt = user.amount.mul(pool.accVoxPerShare).div(1e12);
  }
```

Action Recommended: Using a local variable for pending rewards while setting `pendingRewards` data field to 0 directly can save gas cost. Here's a suggestion:

```
function claim(uint256 _pid) public {
      require(_pid != 0, "please claim staking rewards on stake page");
      PoolInfo storage pool = poolInfo[_pid];
      UserInfo storage user = userInfo[_pid][msg.sender];
      updatePool(_pid);
      uint256 pending = user.amount.mul(pool.accVoxPerShare).div(1e12).sub(
          user.rewardDebt
      );
      if (pending > 0 || user.pendingRewards > 0) {
          int256 pendingRewards = user.pendingRewards.add(pending);
          user.pendingRewards = 0;
          safeVoxTransfer(msg.sender, pendingRewards);
          emit Claim(msg.sender, _pid, pendingRewards);
      }
      user.rewardDebt = user.amount.mul(pool.accVoxPerShare).div(1e12);
  }
```

Disclaimer