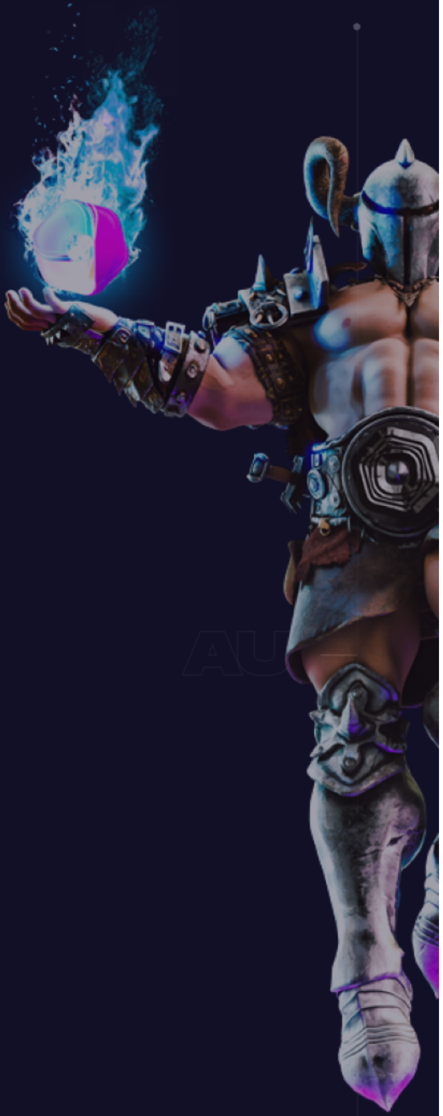
UnoRe

Insured Audit: Code Review & Protocol Security Report

VOX FINANCE



Protocol

Vox Finance

Date

15th May 2023

The UnoRe security research team has completed an initial time-boxed security review of the **Vox Finance** protocol, with a focus on the security aspects of the application's implementation.

Disclaimer

This report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all the vulnerabilities are fixed - upon a decision of the Customer.

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts.

Document Changelog:

24th March 2023	**Initial Pre-Triage Insured Audit Report**
20th April 2023	**Active Monitoring Notes**
15th May 2023	**Finance Post-Triage Insured Audit Report**

Technical Overview

The Vox Finance protocol allows holders of the VOX token to lock their tokens into the VoxStakingPool or the VoxLiquidityFarm in exchange for rewards. The VoxStakingPool has a minimumLock of 2 weeks and a maximumLock of 52 weeks, while the VoxLiquidityFarm does not have locking periods. Both of the contracts have a different withdrawalFee that is taken from the user. However, it is important to note that the fee can be changed by the owner and set to withdrawalFeeMax.

The VOX token has a 4.0 % fee on each buy and sell transaction which is distributed between marketingWallet, liquidityPool and a part of it is burned. More documentation and information about the Tokenomics can be found here.

Threat Model

Roles & Actors

1. Users - able to stake their VOX tokens or deposit them to VoxLiquidityFarm.
2. Owner - able to set critical parameters like withdrawalFee, rewardsDuration, setTreasury, recoverERC20, setLockingPeriods. It can also add and remove

addresses that are ExcludedFromFees and ExcludedMaxTransactionAmount. The owner has extensive access to functions that are restricted or use the onlyOwner modifier.

3. SwapManager - able to addLiquidity, buyAndBurn VOX tokens, and it is approved to swap tokens for ETH. Also, it is ExcludedFromFees and ExcludedMaxTransactionAmount.

4. Marketing Wallet - receives 50% of each fee charged on buy/sell transactions.

Internal Security QA

1. What in the protocol has value in the market? The VOX tokens that are locked in the contract and rewardsToken.

2. What is the worst thing that can happen to the protocol? If the protocol is put into DoS state or locked tokens are stolen.

3. In what case can the protocol/users lose money? If an attacker is able to drain the VoxStakingPool / VoxLiquidityFarm or is able to claim the rewards of other users because of miscalculations.

Severity classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack
Likelihood - the chance that a particular vulnerability gets discovered and exploited
Severity - the overall criticality of the risk

Security Review Summary

Review commit hash - 9c94722e32965b6298d885f6d323fc55bfa8a0e4

Audit Scope

The following smart contracts were in scope of the audit:

- VoxLiquidityFarm.sol

- VoxStakingPool.sol
- VoxSwapManager.sol
- VoxToken.sol
- VoxTokenAirdrop.sol
- VoxVestingWallet.sol

The following number of issues were found, categorized by their severity:

- Critical & High: 1 issues
- Medium: 3 issues
- Low: 8 issues
- Informational: 12 issues

Note: The above summary of report findings at the Pre-Triage stage, most of these issues were addressed/fixed in consecutive stages.

Summary Table of Our Findings

ID	Title	Severity	Status
[H-01]	There is no slippage control in addLiquidity and swapToWeth methods, which expose strategy to sandwich attack	High	Fixed
[M-01]	Owner can steal all of the stakingToken	Medium	Fixed
[M-02]	notifyRewardAmount can lead to loss of yields for the users	Medium	Fixed
[M-03]	setRewardsDuration allows setting near zero or enormous rewardsDuration, which breaks reward logic	Medium	Confirmed
[L-01]	Check array arguments have the same length	Low	Confirmed
[L-02]	Use two-step ownership transfer approach	Low	Confirmed
[L-03]	Avoid using tx.origin for validation	Low	Confirmed
[L-04]	Missing 0 address check	Low	Confirmed
[L-05]	Handle 0 reward case	Low	Confirmed
[L-06]	Set bounds for multiplier	Low	Confirmed
[L-07]	Transactions may revert because of a deadline	Low	Confirmed

[L-08]	Add a timelock to restricted to owner functions that set critical values	Low	Confirmed
[I-01]	Using SafeMath when compiler is ^0.8.0	Informational	
[I-02]	NatSpecs are incomplete	Informational	
[I-03]	Make use of Solidity time units	Informational	
[I-04]	Use custom errors instead of require statements with string error	Informational	
[I-05]	Not used events can be removed	Informational	
[I-06]	Unclear error message	Informational	
[I-07]	CEI pattern is not followed	Informational	
[I-08]	Variables can be turned into an immutable	Informational	
[I-09]	Most setter functions do not emit events	Informational	
[I-10]	Improper naming	Informational	
[I-11]	Contracts are not inheriting their interfaces	Informational	
[I-12]	Solidity safe pragma best practices are not used	Informational	

Triage Fix Comments

[H-01] There is no slippage control in addLiquidity and swapToWeth methods, which expose the strategy to sandwich attack
https://arbiscan.io/address/0xa0eebb0e5c3859a1c5412c2380c074f2f6725e2e#readContract

[M-01] Updated in repository and ownership on live contract has been renounced:
https://github.com/voxfinance/vox2.0-protocol/commit/8876fe77553ea7417d10cc63947ba21c9dc323a6
https://arbiscan.io/tx/0x3f5ad1b1a850902e89cf648641f9e60e60e7fa2e61555c9607104dac2fd6171c

[M-02] notifyRewardAmount can lead to loss of yields for the users, fixed in repository and renounced on live contract

https://github.com/voxfinance/vox2.0-protocol/commit/3a667d9ebe5aff7685276fb0e4162564f20cd592
https://github.com/voxfinance/vox2.0-protocol/commit/d720a946ef1330b974b341888836b3e486e5faeb
https://arbiscan.io/tx/0x2e95894ae40944d2e290aa3d0e7e11f9dbd5ed0b9c9947fbb0c41796a700d0f8

[M-03] setRewardsDuration allows setting near zero or enormous rewardsDuration, which breaks reward logic:
Live contracts were already renounced, repo changes (includes a missing semi-colon from a previous commit)
https://github.com/voxfinance/vox2.0-protocol/commit/85614cbb43b306845acfd5c4324d449294dfa0e0

[L-01] Check array arguments have the same length.
https://github.com/voxfinance/vox2.0-protocol/blob/main/VoxTokenAirdrop.sol
Line 26 includes this check

[L-02] Use two-step ownership transfer approach
Added to repository:
https://github.com/voxfinance/vox2.0-protocol/commit/847bb63cb997985417fd28f3762810cfcfdc9159

[L-03] Avoid using tx.origin for validation
https://github.com/voxfinance/vox2.0-protocol/commit/598e750f6671ded1108a0bdd2b3236910a8c7de2

[L-04] Missing 0 address check
https://github.com/voxfinance/vox2.0-protocol/commit/bceeb35964dbc0061121a693daf9fe949d6c8f83

[L-05] Handle 0 reward case
https://github.com/voxfinance/vox2.0-protocol/commit/1c387b9d89b6d68ae318c8bf83161a15c2098326

[L-06] Set bounds for multiplier
https://github.com/voxfinance/vox2.0-protocol/commit/d4fc7b1bab1d168a9df5a089a67a55fa78a099b5

[L-07] Transactions may revert because of a deadline
https://github.com/voxfinance/vox2.0-protocol/commit/72d407a0ebd9894801eb93ee103b5ea19589f821

[L-08] Add a timelock to restricted functions that set critical values

We have transferred ownership of VoxToken to a Safe instance with 2-of-3 signing policy, thus ensuring that the functions can not be changed without approval from the entire project team. We will look into implementing a timelock function in the future. All other contracts for staking have been renounced.
https://arbiscan.io/tx/0x40aa14e4ad41bccf7598f5c2414059b60539883ea8dd38892c9355bebd2155ac

Centralization Risk Areas

We have also identified several key areas within the protocol which contains centralization risks which needs to be made aware to the community and have highlighted them below:

VoxLiquidityFarm (VoxLiquidityFarm | Address 0x87195340478b792cfb0986450c39b64846867716 | Arbiscan)

1. renounceOwnership (ownable.sol)
2. transferOwnership (ownable.sol)
3. setPaused (Pausable.sol)
4. setTreasury (VoxLiquidityFarm.sol)
5. setStakingPool (VoxLiquidityFarm.sol)
6. recoverERC20 (VoxLiquidityFarm.sol)

VoxStakingPool (VoxStakingPool | Address 0x0B21cfbe22b5730f050c2787379a8263FCCd276b | Arbiscan)

1. renounceOwnership (ownable.sol)
2. transferOwnership (ownable.sol)
3. setPaused (Pausable.sol)
4. recoverERC20 (VoxStakingPool.sol)
5. setTreasury (VoxStakingPool.sol)

VoxSwapManager (VoxSwapManager | Address 0xe84713bE6d41475429bA65A6092973595b7b286A | Arbiscan)

1. renounceOwnership (ownable.sol)
2. transferOwnership (ownable.sol)
3. recover(VoxSwapManager.sol)

VoxToken (Vox Finance: VOX2.0 Token | Address 0xa0eebb0e5c3859a1c5412c2380c074f2f6725e2e | Arbiscan)

1. renounceOwnership (ownable.sol)
2. transferOwnership (ownable.sol)
3. enable Trading (VoxToken.sol)

4. removeLimits(VoxToken.sol)
5. disableTransferDelay(VoxToken.sol)
6. updateSwapTokensAtAmount (VoxToken.sol)
7. recover (VoxToken.sol)

VoxTokenAirdrop(VoxTokenAirdrop | Address 0x3279C1D0a34D60B84BCcba55EE08d220032958aF | Arbiscan)

1. renounceOwnership (ownable.sol)
2. transferOwnership (ownable.sol)
3. setToken (VoxTokenAirdrop.sol)
4. sendBatch(VoxTokenAirdrop.sol)

Initial Report Detailed Findings

[H-01] There is no slippage control in addLiquidity and swapToWeth methods, which expose the strategy to sandwich attack

Severity

Impact: High, as VoxToken contract will lose money due to sandwich attacks
Likelihood: Medium, since MEV is very prominent, the chance of that happening is pretty high

Description

File: VoxSwapManager.sol

We can see the following code in these functions:

Function: addLiquidity

```
router.addLiquidity(
            address(vox),
            vox.weth(),
            voxAmount,
            wethAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            owner(),
            block.timestamp
        );
```

Function: swapToWeth

```
 router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            voxAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
```

```
```

Function: buyAndBurn

```
    router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
        wethAmount,
        0,
        path,
        address(this),
        block.timestamp
     );
```

The “0”s here are the value of the amountOutMin argument which is used for slippage tolerance. 0 value here essentially means 100% slippage tolerance. This is a very easy target for MEV and bots to do a flash loan sandwich attack on each of the strategy’s swaps, resulting in a very big slippage on each trade. 100% slippage tolerance can be exploited in a way that the strategy (so the vault and the users) receive much less value than it should have. This can be done on every trade if the trade transaction goes through a public mempool.

Recommendations

Add a protection parameter to the above-mentioned functions, so that the VoxToken contract can specify the minimum out amount.

[M-01] Owner can steal all of the stakingToken

Severity

Impact: High, as all of the staked tokens can be withdrawn
Likelihood: Low, as it requires a malicious/compromised owner

Description

The recoverERC20 function inside VoxStakingPool rightfully checks if the passed tokenAddress is different from the rewardsToken address. However, it does not check if it is not the same as the stakingToken address which should be the case as can be seen from the comment:

```
  function recoverERC20(address tokenAddress, uint tokenAmount)
      external
      onlyOwner {
      // Cannot recover the staking token or the rewards token
      require(
          tokenAddress != address(rewardsToken),
          "Cannot withdraw the staking or rewards tokens"
      );
      ..
  }
```

This could be exploited by a malicious or compromised owner. This admin privilege allows the owner to sweep the staking tokens, potentially harming depositors by rug-pulling.

Recommendations

Add an additional check inside the require statement:

`tokenAddress != address(stakingToken)`

[M-02] notifyRewardAmount can lead to loss of yields for the users

Severity

Impact: High, because users` yield can be manipulated
Likelihood: Low, this is restricted function and only the owner can call it

Description

The notifyRewardAmount function takes a reward amount and extends the periodFinish to now + rewardsDuration:

periodFinish = block.timestamp.add(rewardsDuration);

It rebases the leftover rewards and the new reward over the rewardsDuration period.

```
    function recoverERC20(address tokenAddress, uint tokenAmount)
        external
        onlyOwner {
        // Cannot recover the staking token or the rewards token
        require(
            tokenAddress != address(rewardsToken),
            "Cannot withdraw the staking or rewards tokens"
        );
        ..
    }

```

This can lead to a dilution of the reward rate and rewards being dragged out forever by malicious new reward deposits.

Let's take a look at the following example:

1. For the sake of the example, imagine the current rewardRate is 1000 rewards / rewardsDuration.

2. When 10% of rewardsDuration has passed, a malicious owner calls notifyRewards with reward = 0.

3. The new rewardRate = 0 + 900 / rewardsDuration, which means the rewardRate just dropped by 10%.

4. This can be repeated infinitely. After another 10% of reward time passed, they trigger notifyRewardAmount(0) to reduce it by another 10% again: rewardRate = 0 + 720 / rewardsDuration.

The rewardRate should never decrease by a notifyRewardAmount call.

Recommendations

There are two potential fixes to this issue:

1. If the periodFinish is not changed at all and not extended on every notifyRewardAmount call. The rewardRate should just increase by rewardRate += reward / (periodFinish - block.timestamp).

2. Keep the rewardRate constant but extend periodFinish time by += reward / rewardRate.

[M-03] setRewardsDuration allows setting near zero or enormous rewardsDuration, which breaks reward logic

Severity

Impact: High, as it breaks reward logic
Likelihood: Low, as it requires an error from the owner's side or a compromised/malicious owner

Description

File: VoxStakingPool.sol

notifyRewardAmount method will be inoperable if rewardsDuration is set to zero. It will cease to produce meaningful results if rewardsDuration be too small or too big.

The setter does not control the value, allowing zero/near zero/enormous duration:

```
    function setRewardsDuration(uint _rewardsDuration) external restricted {
        require(
            block.timestamp > periodFinish,
            "Previous rewards period must be complete before changing the duration for the
new period"
        );
        rewardsDuration = _rewardsDuration;
        emit RewardsDurationUpdated(rewardsDuration);
    }
```

Division by the duration is used in notifyRewardAmount:

```
if (block.timestamp >= periodFinish) {
    rewardRate = reward.div(rewardsDuration);
```

Recommendations

Check for min and max range in the rewardsDuration setter, as too small or too big rewardsDuration breaks the logic.

[L-01] Check array arguments have the same length

When the sendBatch function is called inside VoxTokenAirdrop, two array-type arguments are passed. Validate that the arguments have the same length so you do not get unexpected errors if they don't.

[L-02] Use two-step ownership transfer approach

The owner role is crucial for the protocol as there are a lot of functions with the onlyOwner and the restricted modifiers. Make sure to use a two-step ownership transfer approach by using Ownable2Step from OpenZeppelin as opposed to Ownable as it gives you the security of not unintentionally sending the owner role to an address you do not control. Also, consider using only onlyOwner modifiers instead of using both onlyOwner and restricted modifiers because they are basically the same and using both only creates confusion.

[L-03] Avoid using tx.origin for validation

Inside VoxToken.sol, the following require statement is used:

```
  require(
     _holderLastTransferTimestamp[tx.origin] <
     block.number,
     "_transfer:: Transfer Delay enabled.  Only one purchase per block allowed."
  );
```

This can be easily bypassed if the function is called by a contract. Use msg.sender instead of tx.origin.

[L-04] Missing 0 address check

In VoxStakingPool's constructor we can see that there is a 0 address check for stakingToken but such check is missing for rewardsToken.

```
constructor(
        address _rewardsToken,
        address _stakingToken
    ) {
        rewardsToken = IERC20(_rewardsToken);
        if (_stakingToken != address(0)) {
            stakingToken = IERC20(_stakingToken);
        }
    }
```

Consider adding a 0 address check for rewardsToken as well.

[L-05] Handle 0 reward case

In getReward a check is missing if the rewards are equal to 0. Consider adding the following check with a custom error:

```
function getReward() public nonReentrant updateReward(msg.sender) {
        uint reward = rewards[msg.sender];
+        if(reward == 0) revert ZeroRewards();
        if (reward > 0) {
            rewards[msg.sender] = 0;
            rewardsToken.safeTransfer(msg.sender, reward);
            emit RewardPaid(msg.sender, reward);
        }

```

[L-06] Set bounds for multiplier

In setMultiplier the owner of the contract can set a new value for the multiplier. However, there might be a problem if there is a compromised or malicious owner. Set a max bound in setMultiplier.

[L-07] Transactions may revert because of a deadline

In the VoxSwapManager, the router.addLiquidity is called and the block.timestamp is passed as deadline. This means that if the execution takes longer than the current timestamp, the transaction will revert as it can be seen from the Uniswap documentation. It is the same for router.swapExactTokensForTokensSupportingFeeOnTransferTokens and router.swapExactTokensForTokensSupportingFeeOnTransferTokens. Consider changing it to block.timestamp + 2 minutes, for example, to give it a bit of tolerance.

[L-08] Add a timelock to restricted functions that set critical values

It is a good practice to give time for users to react and adjust to critical changes. A timelock provides more guarantees and reduces the level of trust required, thus decreasing the risk for users. It also indicates that the project is legitimate. Here, no timelock capabilities seem to be used. We believe this impacts multiple users enough to make them want to react/be notified ahead of time.

Consider adding a timelock to functions like: setWithdrawalFee, setLockingPeriod, etc.