Audit of The VoxVault Contracts

a report of findings by

Van Cam Pham, PhD

innovative fortuna iuvat

December 7th, 2020

Table of Contents

Document Info

Client	Vox Finance
Title	Smart Contract Audit of Vox Finance Vault 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 Vault smart contracts on the Vox Protocol repo at Commit #d437b0cd71c4d21cb57609c011c23a0d0ea71108.

VoxVault.sol
strategies/VoxStrategyBase.sol
strategies/VoxStrategyCurveBase.sol
strategies/curve/VoxStrategyCurve3CRV.sol
strategies/curve/VoxStrategyCurveSCRV.sol
strategies/curve/VoxStrategyCurveTBTC.sol

After a first review and report and discuss our findings with the team. The team fixed all issues found by our findings and we had a second view on the code at commit #36b2609858d14a94c0da5870d500f0b41c4219d5.

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. Vault should be inactive by default after contract deployment

- Vault-1
- Severity: low
- Likelihood: low
- Impact: low

- Target: VoxVault.sol
- Category: low
- Finding Type: Dynamic
- Lines: 236-250

Contract VoxVault should be inactive by default and only become active when the vault strategy is correctly configured.

```
    bool public isActive = true;
```

Action Recommended:

- Change `isActive` contract field to false by default.

Review of the code at commit 36b2609858d14a94c0da5870d500f0b41c4219d5:

- `isActive` is set to false by default in the commit.

2. TimeLock contract address is set without validation

- Vault-2
- Severity: Medium
- Impact: Medium

- Target: VoxVault.sol
- Category: TimeLock
- Finding Type: Dynamic
- Lines: 53-66, 96-99

Function constructor of `VoxVault` should validate the `_timelock` contract for its time lock delay. This is to ensure that the admin of the time lock contract cannot make any changes to the `VoxVault` contract before a minimum time lock delay.

This issue is also applied to the function `setTimelock` in the `VoxVault` contract.

```
constructor(address _token, address _vox, address _governance, address _treasury, address
_timelock)
        public
        ERC20(
            string(abi.encodePacked("voxie ", ERC20(_token).name())),
            string(abi.encodePacked("v", ERC20(_token).symbol()))
        )
    {
        _setupDecimals(ERC20(_token).decimals());
        token = IERC20(_token);
        vox = IERC20(_vox);
        governance = _governance;
        treasury = _treasury;
        timelock = _timelock;
    }
function setTimelock(address _timelock) public {
        require(msg.sender == timelock, "!timelock");
        timelock = _timelock;
    }
```

Action Recommended: Add time lock contract validation for minimum time lock delay.

Review of the code at commit 36b2609858d14a94c0da5870d500f0b41c4219d5:

- Validation for time lock is added to the constructor and `setTimelock` function of VoxVault contract.

```
constructor(address _underlying, address _vox, address _governance, address _treasury, address
_timelock)
        public
        ERC20(
            string(abi.encodePacked("voxie ", ERC20(_underlying).name())),
            string(abi.encodePacked("v", ERC20(_underlying).symbol()))
        )
```

```
    {
        require(address(_underlying) != address(_vox), "!underlying equal to vox");
        require(ITimelock(_timelock).delay() >= minTimelockInterval, "!timelock min");
       …
}

function setTimelock(address _timelock) public isTimelock {
        require(ITimelock(_timelock).delay() >= minTimelockInterval, "!timelock min");
        timelock = _timelock;
    }
```

3. Storage field `issued` of the contract seems to be redundant

- Vault-3
- Severity: Low
- Impact: Low
- Target: VoxVault.sol
- Category: Informational
- Finding Type: Dynamic

Storage field `issued` of the VoxVault contract is redundant because it is the vault token balance.

However, as discussed with the Vox team, the Vox Vault token can be transferred and tradable. Therefore, the value stored in _balances for the Vault token balance is different from the issued shares to users who deposit to the vault. The Vault however requires users to have enough shares in their wallet in order to withdraw the deposit. This is valid and correctly implemented in the contract.

Action Recommended: No action is needed for this issue.

4. Code simplification for function `deposit`

- Vault-4
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: VoxVault.sol
- Category: Informational
- Finding Type: Dynamic
- Lines: 224-232

The code segment in the `deposit` function of `VoxVault` contract that computes the amount of shares issued for the user can be simplified.

```
if (totalSupply() == 0) {
            if (tiers[msg.sender] > 0) {
                uint256 userMultiplier = tiers[msg.sender].add(tierBase);
                shares = _amount.mul(userMultiplier).div(tierBase);
            } else {
                shares = _amount;
            }
        } else {
            if (tiers[msg.sender] > 0) {
                uint256 userMultiplier = tiers[msg.sender].add(tierBase);
                shares =
(_amount.mul(userMultiplier).div(tierBase).mul(totalSupply())).div(_pool);
            } else {
                shares = (_amount.mul(totalSupply())).div(_pool);
            }
        }
```

Action Recommended: The code can be simplified as follows:

```
if (totalSupply() == 0) {
            uint256 userMultiplier = tiers[msg.sender].add(tierBase);
            shares = _amount.mul(userMultiplier).div(tierBase);
        } else {
            uint256 userMultiplier = tiers[msg.sender].add(tierBase);
            shares = (_amount.mul(userMultiplier).div(tierBase).mul(totalSupply())).div(_pool);
        }
```

Review of the code at commit 36b2609858d14a94c0da5870d500f0b41c4219d5

- The code is simplified.

5. Constructor's input token should not be Vox token

- Vault-5
- Severity: Low
- Likelihood: Low
- Impact: Low

- Target: VoxVault.sol
- Category: Informational
- Finding Type: Dynamic
- Lines: 53-66

In the `VoxVault` constructor, the input `_token` must be different than `_vox` token so that deposited token will not be accounted as rewards in `withdraw` function.

Action Recommended: Add validation checking that `_token` must be different than `_vox`.

Review of the code at commit 36b2609858d14a94c0da5870d500f0b41c4219d5

- The updated code has been added for validation the tokens input.

```
require(address(_underlying) != address(_vox), "!underlying equal to vox");
```

6. Tiers for rewards boost is too small

- Vault-6
- Severity: Low
- Likelihood: Low
- Impact: Low

- Target: VoxVault.sol
- Category: Informational
- Finding Type: Dynamic

In the `VoxVault` contract, `tiers` range from 1 to 5 while `tierbase` is 100. This means the users who purchase multipliers to boosts their rewards from vaults only get a small extra rewards (from 1 to 5 %) compared to those who do not purchase multipliers.

However, as we discussed with the team, this is intended by design of the vault.

Action Recommended: No action recommended for this issue.

7. Reentrancy safeguard should be applied to deposit and withdraw functions

- Vault-7
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Target: VoxVault.sol
- Category: Informational
- Finding Type: Dynamic

At the time the contract is audited, we do not find any issue potentially caused by reentrancy for functions `deposit` and `withdraw`. However, as for safety even if there is any changes/updates in the `underlying` token of the vault (which could might cause reentrancy issues to the vault), we recommend to add reentrancy safeguards to the two functions in order to any potential issue caused by future update of the underlying token.

Action Recommended: Add reentrancy safeguard to the two functions `deposit` and `withdraw`.

Review of the code at commit 36b2609858d14a94c0da5870d500f0b41c4219d5

- The updated code has been added for reentrancy safeguards for `deposit` and `withdraw` functions.

```
      function deposit(uint256 _amount) public nonReentrant {
        require(!address(msg.sender).isContract() && msg.sender == tx.origin, "!no
contract");
        require(isActive, 'vault is not active');
        ..
      }

      function withdraw(uint256 _amount) public nonReentrant {
       …
      }
```

8. Lack of unit-tests for deposits and withdraws

- Vault-8
- Severity: Medium
- Likelihood: Medium

- Target: VoxVault.sol
- Category: Informational
- Finding Type: Dynamic

- Impact: Medium

The VoxVault contract lacks unit tests, especially unit tests for deposits and withdraws to ensure that:

- Any user should be able to withdraw the corresponding deposit
- Any user should be able to transfer shares to others
- User without enough issued shares should not be able to withdraw all of the user's deposit

Action Recommended: Add unit tests to the `deposit` and `withdraw` functions to ensure the above functionalities.

Unit-test update: As requested from the Vox team, the auditing team has written unit-tests for the `deposit` and `withdraw` functions for the above functionalities. The unit-tests have been successfully passed against the latest snapshot of the Ethereum mainnet using ganache-cli fork. Unit-tests are pushed at commit #6f16ba807877d0a8fdc6dab1ee2a5aafa4c410cd

9. Timelock contract for VoxStrategyBase needs to be checked for its delay

- Vault-9
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: VoxStrategyBase.sol
- Category: TimeLock
- Finding Type: Dynamic

Function constructor of `VoxStrategyBase` should validate the `_timelock` contract for its time lock delay. This is to ensure that the admin of the time lock contract cannot make any changes to the `VoxStrategyBase` contract before a minimum time lock delay.

This issue is also applied to the function `setTimelock` in the `VoxStrategyBase` contract.

Action Recommended: Add time lock contract validation for minimum time lock delay.

Review of the code at commit 36b2609858d14a94c0da5870d500f0b41c4219d5

- Validation for time lock is added to the constructor and `setTimelock` function of `VoxStrategyBase` contract.

10. Value range validity for `setTreasuryFee`, `setDeveloperFee` and `setStrategyFee`

- Vault-10
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Target: VoxStrategyBase.sol
- Category: Value Range
- Finding Type: Dynamic
- Lines: 99-102, 109-112, 104-107

Functions `setTreasuryFee`, `setDeveloperFee` and `setStrategyFee` should have value range checking for input parameters.

Action Recommended: Value range checking for the input parameters of the functions.

Review of the code at commit 36b2609858d14a94c0da5870d500f0b41c4219d5

- Value range checks have been added to the input parameters of the functions.

11. setVault needs to verify the underlying token

- Vault-11
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Target: VoxStrategyBase.sol
- Category: Vault underlying token
- Finding Type: Dynamic
- Lines: 139-142

Function `setVault` needs to verify that the wanted token of the vault is the underlying token of the strategy.

```
function setVault(address _vault) external {
        require(msg.sender == timelock, "!timelock");
        vault = _vault;
    }
```

Action Recommended: Add checking for matching of the underlying token and the wanted token.

Review of the code at commit 36b2609858d14a94c0da5870d500f0b41c4219d5

- Validity check has been added

```
function setVault(address _vault) external isTimelock {
      require(IVault(_vault).underlying() == address(underlying), 'vault does not
support this underlying');
      vault = _vault;
   }
```

12. Withdraw and withdrawAll functions can avoid gas consumption

- Vault-12
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: VoxStrategyBase.sol
- Category: Gas
- Finding Type: Dynamic
- Lines: 156-175

The two functions `withdraw` and `withdrawAll` functions can move `require(vault != address(0), "!vault");` to the beginning of the function to avoid gas consumption in case the vault is not set yet.

```
function withdraw(uint256 _amount) external {
      require(msg.sender == vault, "!vault");
      uint256 _balance = IERC20(underlying).balanceOf(address(this));
      if (_balance < _amount) {
          _amount = _withdrawSome(_amount.sub(_balance));
          _amount = _amount.add(_balance);
      }

      require(vault != address(0), "!vault"); // additional protection so we don't burn the
funds
      IERC20(underlying).safeTransfer(vault, _amount);
   }
```

```
    // Withdraw all funds, normally used when migrating strategies
    function withdrawAll() external restricted returns (uint256 balance) {
        _withdrawAll();
        balance = IERC20(underlying).balanceOf(address(this));

        require(vault != address(0), "!vault"); // additional protection so we don't burn the
funds
        IERC20(underlying).safeTransfer(vault, balance);
    }
```

Action Recommended: Move the `require`s to the beginning of the functions.

Review of the code at commit 36b2609858d14a94c0da5870d500f0b41c4219d5

- The required statements have been moved to the beginning of the functions.

13. More minor issues

- In the `restricted` modifier, `msg.sender == tx.origin` should be used combined with `!isContract` checking the origin of the transaction is not a smart contract (preventing malicious smart contracts from withdrawing from the gauge).
- Code readability: `require(msg.sender == governance, "!governance");` and `require(msg.sender == timelock, "!timelock");` used multiple times repeatedly, it is good practice to have modifiers for those checks.

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.