Audit of cVault.Finance

Audit of cVault.Finance

a report of findings by

Connor Martin

&

Van Cam Pham, PhD

innovative fortuna iuvat

October 7th, 2020

Table of Contents

Document Info

Client	CORE
Title	Smart Contract Audit
Auditors	Van Cam Pham, PhD & Connor Martin
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 cVault Decentralized Organization ("cVault") engaged The Arcadia Group ("Arcadia"), a software development, research and security company, to conduct a review of the following cVault Finance Smart Contracts ("cVault Core-v1") on the cVault-finance/CORE-v1 repo at Commit #0789e77345c602ed74a5ce3786fb93e3b334fc67.

CORE.sol
NBUNIERC20.sol,
CoreVault.sol
FeeApprover.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 above to not require any immediate action, it is strongly recommended in the future that code-reviews and audits be done prior to launch to avoid situations where potentially risky issues could be public facing with limited recourse.

Noting that, we found auditing the CORE project to be a smooth experience, and we commend the CORE team for their availability in communication.

Findings

1. Pragma Inheritance

- CORE-1
- Severity: Informational
- Likelihood: Low
- Impact: Low

- Target: All Contracts
- Category: Informational
- CWE-664
- Finding Type: Static

In the Core-v1 Contracts, a command pragma is not set (SWC-103), this allows for the introduction of solidity version specific bugs.

Action Recommended: Normalize Solidity version across contracts for future releases

FeeApprover.sol

2. State Visibility

- CORE-2
- Severity: Low
- Likelihood: Low
- Impact: Low

- Target: FeeApprover.sol
- Category: Informational
- CWE-710
- Finding Type: Static
- Lines: 32, 35, 36, 39, 65

In the Core-v1 Contracts, variable visibility is not set for multiple variables in the FeeApprover.sol contract. A public variable is easily accessible to users of the contract, and while no specific issues were found with the specific variables identified, it is best practice to utilize the most restrictive visibility as possible, unless it is specifically needing to be public, in which case specifically set it to public.

```
    address tokenUniswapPair;
...
    address coreTokenAddress;

...
    address coreVaultAddress;

...
```

```
    bool paused;
...
    uint minFinney; // 2x for $ liq amount
```

Action Recommended: In future versions of the FeeApprover.sol contract, specifically set the visibility status for each variable, even if it's intended to be public. While this is not an issue that requires immediate attention, it is important to do this for code quality, health and review purposes.

3. Unused Function

- CORE-3
- Severity: Notice
- Likelihood: N/A
- Impact: Low

- Target: FeeApprover.sol
- Category: Informational
- Finding Type: Static
- Lines: 94

In the Core-v1 Contract, FeeApprover.sol, there is an unused variable which may be used in the future, it is recommended at this time on future deployments if the variable is still unused, to remove it as unutilized code. As the contract has already been deployed this has no immediate action but should be kept in mind for future deployments.

```
        address recipient, // unusued maybe use din future
```

Action Recommended: In future versions of the FeeApprover.sol contract, remove variable as unutilized code. As the contract has already been deployed this has no immediate action but should be kept in mind for future deployments.

4. NoFee Whitelist is Adjustable By Contract Owner

- CORE-4
- Severity: Medium
- Likelihood: Low

- Target: FeeApprover.sol
- Category: Security
- Finding Type: Dynamic

- Impact: Low
- Lines: 110

In the Core-v1 Contract, FeeApprover.sol, the editnoFeeList Function & noFeeList state variable is freely adjustable by the contract owner. This gives the owner the ability to add any address to have no fee transfers. At the current time, the contract owner is set to a 2/2 multisig which somewhat mitigates the problem.

```
mapping (address => bool) public noFeeList;


    function setCoreVaultAddress(address _coreVaultAddress) public
onlyOwner {
        coreVaultAddress = _coreVaultAddress;
        noFeeList[coreVaultAddress] = true;
    }

    function editNoFeeList(address _address, bool noFee) public onlyOwner {
        _editNoFeeList(_address,noFee);
    }
    function _editNoFeeList(address _address, bool noFee) internal{
        noFeeList[_address] = noFee;
    }
```

Action Recommended: Upon initialization of planned governance roadmap, contract ownership should be transferred to governance as soon as it possible (when it is safe to do so).

Core.sol

5. Inherited Unused Voting Code

- CORE-5
- Severity: Medium
- Likelihood: Low
- Impact: Low
- Target: Core.sol
- Category: Code Clarity
- Finding Type: Dynamic
- Lines: *

In the Core-v1 Contracts, inherited unused voting functionality from SushiSwap is present, the vulnerability consists of the following issue, when a token is transferred from one address to another address, the sender's vote is not moved to the recipient, which means the voting power remains in the hands of the sender even after token transfer is done. While it does not pose a threat as it's unused to limit potential future risk/mistakes, the voting code should be removed in

any future deployments of the ERC-20 token (i.e. wrapped tokens, or a future extended ERC-20 contract.

```
      /// @notice An event thats emitted when an account changes its
delegate
    event DelegateChanged(address indexed delegator, address indexed
fromDelegate, address indexed toDelegate);

    /// @notice An event thats emitted when a delegate account's vote
balance changes
    event DelegateVotesChanged(address indexed delegate, uint
previousBalance, uint newBalance);

    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
     * @param delegator The address to get delegatee for
     */
    function delegates(address delegator)
        external
        view
        returns (address)
    {
        return _delegates[delegator];
    }

   /**
    * @notice Delegate votes from `msg.sender` to `delegatee`
    * @param delegatee The address to delegate votes to
    */
    function delegate(address delegatee) external {
        return _delegate(msg.sender, delegatee);
    }
```

Action Recommended: As the ERC-20 contract has already been deployed there is very little immediate action possible, however in the event of any redeployments or new token deployments, the unused and vulnerable voting code should be removed from the token contract before any new deployments.

NBUNIERC20.sol

6. Inherited Vulnerable Voting Code

- CORE-6
- Severity: Medium
- Likelihood: Low
- Impact: Low

- Target: NBUNIERC20.sol
- Category: Code Clarity
- Finding Type: Dynamic
- Lines: *

This issue is a duplicate of CORE-5 except affecting the NBUNIERC20.sol contract
Action Recommended: As the ERC-20 contract has already been deployed there is very little immediate action possible, however in the event of any redeployments or new token deployments, the unused and vulnerable voting code should be removed from the token contract before any new deployments.

7. Code Structure

- CORE-7
- Severity: Notice
- Likelihood: Notice
- Impact: Notice

- Target: NBUNIERC20.sol
- Category: Code Clarity
- Finding Type: Dynamic
- Lines: *

Action Recommended: Ensure that you try to order your storage variables and struct members such that they can be packed tightly. For example, declaring your storage variables in the order of uint128, uint128, uint256 instead of uint128, uint256, uint128, as the former will only take up two slots of storage whereas the latter will take up three. This is a good way to improve codebase health, gas cost and clarity.

8. Token Uniswap Pair Location

- CORE-8
- Severity: Medium
- Likelihood: Low
- Impact: Low

- Target: FeeApprover.sol & NBUNIERC20.sol
- Category: Code Clarity
- Finding Type: Dynamic
- Lines: 382

The tokenUniswapPair state variable is obtained through uniswap but it appears independently in two different contracts (FeeApprover.sol and NBUNIERC20.sol). This function should check whether tokenUniswapPair is the same in FeeApprover.sol and NBUNIERC20.sol

```
    function initialize(
        address _COREAddress,
        address _WETHAddress,
        address _uniswapFactory
    ) public initializer {
        OwnableUpgradeSafe.__Ownable_init();
        coreTokenAddress = _COREAddress;
        WETHAddress = _WETHAddress;
        tokenUniswapPair =
IUniswapV2Factory(_uniswapFactory).getPair(WETHAddress,coreTokenAddress);
        feePercentX100 = 10;
        paused = true; // We start paused until sync post LGE happens.
        _editNoFeeList(0xC5cacb708425961594B63eC171f4df27a9c0d8c9, true);
// corevault proxy
        _editNoFeeList(tokenUniswapPair, true);
        sync();
        minFinney = 5000;
    }
```

Action Recommended: Add sanity check to ensure that the trading pair is the same across both contracts. This does not impact current contracts much as the pairs have been set correctly here, but to avoid unnecessary issues in the future, a sanity check would greatly improve code reusability.

9. Redundant Check

- CORE-9
- Severity: Notice
- Likelihood: Notice
- Impact: Notice
- Target: NBUNIERC20.sol
- Category: Code Clarity
- Finding Type: Dynamic
- Lines: 448

There is a redundant check for feeDistributor in the contract, it is not an issue, however the overall deployment and utilization cost will be lowered if this redundant check is removed.

```
if(transferToFeeDistributorAmount > 0 && feeDistributor != address(0)){
            _balances[feeDistributor] =
_balances[feeDistributor].add(transferToFeeDistributorAmount);
            emit Transfer(sender, feeDistributor,
transferToFeeDistributorAmount);
```

```
            if(feeDistributor != address(0)){

ICoreVault(feeDistributor).addPendingRewards(transferToFeeDistributorAmount
);
            }
        }
    }
```

Action Recommended: Remove redundant check when possible.

10. Readibility

- CORE-10
- Severity: Notice
- Likelihood: Notice
- Impact: Notice
- Target: NBUNIERC20.sol
- Category: Readibility
- Finding Type: Dynamic
- Lines: 221

```
mapping (address => uint)  public ethContributed;
```

Action Recommended: Mapping should be moved to a more appropriate location in the contract.

CoreVault.sol

11. Gas Optimization

- CORE-11
- Severity: Low
- Likelihood: Low
- Impact: Low

- Target: NBUNIERC20.sol
- Category: Gas
- Finding Type: Dynamic
- Lines: 133

The add function could potentially run out of gas as the for-loop can cost a large amount of gas in the event there are a large number of pools. While it is unlikely that CORE will have that number of pools in the near future, it is better to have future-proofed than to not have.

```
for (uint256 pid = 0; pid < length; ++pid) {
            require(poolInfo[pid].token != _token,"Error pool already
added");
        }
```

Action Recommended: Add a mapping or other solution that searches for the pool, in order to minimize the gas cost for the for-loop.

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 for damages resulting from the use of the provided information. Additionally Arcadia would like to emphasize that 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 of time.