Security Audit of cVault Lending Smart Contracts

a report of findings by

Arcadia

Jul 14th, 2021

Table of Contents

Document Info **1**

Contact **2**

Executive Summary **3**

Findings **4**

Constructor function approves the contract for itself 4

Function getInterest does not compute interest precisely 6

Liquidation trigger incentives 6

Recommendations **7**

There should be a function to change profiteer address 7

Ensure floor price of tokens 8

Conclusion **8**

Disclaimer **8**

Document Info

Client	cVault Finance
Title	Security Audit of cVault Finance’s Lending Smart Contracts
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 cVault Finance ("CORE") engaged The Arcadia Group ("Arcadia"), a software development, research, and security company, to conduct a review of the following cVault Finance Lending smart contracts on the cVault Finance Lending repo at Commit #4948210cd99149799e917a9600ca2b3965ee2157.

The audit scope includes the following files:

SafeBurn.sol
PawnShop.sol

Arcadia completed this security 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.

There were 05 issues found, 00 of which were deemed to be ‘critical’, and 02 of which were rated as ‘high’.

Severity Rating	Number Of Original Occurrences	Number Of Remaining Occurrences
Critical	00	00
High	02	02
Medium	01	01
Low	02	02
Notice	00	00
Informational	00	00

Findings

1. Constructor function approves the contract for itself

- CLEND-1
- Severity: High
- Likelihood: High
- Impact: High
- Target: PawnShop.sol
- Category: Token Approval
- Finding Type: Dynamic

The current implementation of the `constructor` approves the `CORE` and `coreDAO` tokens for the contract itself, which is not useful at all. Instead, the constructor should approve the `CORE` and `coreDAO` tokens for the uniswap router.

Without approval for the uniswap router, function `_trade` would fail, which causes the `liquidate` function to fail.

```
constructor (IERC20 _CORE, uint256 _coreFloor, IERC20 _coreDAO, uint256 _daoFloor)
public {
       require(_CORE != IERC20(0) && _coreDAO != IERC20(0), "PawnShop::constructor:
Invalid Configuration");
       CORE = _CORE;
       coreFloor = _coreFloor;
       coreDAO = _coreDAO;
       daoFloor = _daoFloor;
       profiteer = msg.sender;
       // Approve Liquidations
       _CORE.approve(address(this), type(uint256).max);
       _coreDAO.approve(address(this), type(uint256).max);
   }



function _trade(address token, uint256 amount) internal returns (uint256) {
       address[] memory path = _getLiquidationPath(token);

       uint256[] memory amounts = uniswapRouter.swapExactTokensForTokens(
           amount,
```

```
            0, // profit already ensured by _liquidate
            path,
            address(this),
            block.timestamp
        );

        return amounts[amounts.length - 1];
    }
```

Action Recommended:

- The constructor should approve the `CORE` and `coreDAO` tokens for uniswapRouter

```
constructor (IERC20 _CORE, uint256 _coreFloor, IERC20 _coreDAO, uint256 _daoFloor)
public {
        require(_CORE != IERC20(0) && _coreDAO != IERC20(0), "PawnShop::constructor:
Invalid Configuration");
        CORE = _CORE;
        coreFloor = _coreFloor;
        coreDAO = _coreDAO;
        daoFloor = _daoFloor;
        profiteer = msg.sender;
        // Approve Liquidations
        _CORE.approve(address(uniswapRouter), type(uint256).max);
        _coreDAO.approve(address(uniswapRouter), type(uint256).max);
    }
```

2. Function getInterest does not compute interest precisely

- CLEND-2
- Severity: High
- Impact: High
- Target: PawnShop.sol
- Category: Interest Calculation
- Finding Type: Dynamic

Function `getInterest` does not compute lending interest precisely. The current implementation computes interest rounded to hours. It means a user can borrow at minute 01 and pay back the loan at minute 59 without having to pay any interest.

```
function getInterest(uint256 amount, uint256 updated) public view override returns
(uint256) {
       if (updated == 0) return 0;
       else return ((block.timestamp - updated) / 1
hours).mul(hourlyInterest).mul(amount) / 1 ether;
   }
```

Action Recommended: The following code is a suggestion for calculating interest precisely. Multiplications should be done before divisions.

```
function getInterest(uint256 amount, uint256 updated) public view override returns
(uint256) {
      if (updated == 0) return 0;
      else return (block.timestamp - updated).mul(hourlyInterest).mul(amount).div(1
hours) / 1 ether;
   }
```

3. Liquidation trigger incentives

- CLEND-3
- Severity: Medium
- Impact: Medium
- Target: PawnShop.sol
- Category: Liquidation incentive
- Finding Type: Dynamic

The current liquidation implementation does not incentivize anyone to run bots to trigger liquidation transactions. The team should consider either:

1. Creating incentives for running bots to trigger liquidation of transactions

2. Ensuring that there will be at least one bot monitoring the contracts to ensure liquidations are triggered on time.

```
function liquidate(address account, bool isCORE) external override {
        (IERC20 token, uint256 rate) = isCORE ? (CORE, coreFloor) : (coreDAO,
daoFloor);
        Loan memory _loan = loans[account][token];

        uint256 amount = getLoanWithInterest(_loan.amount, _loan.updated);
        uint256 collateralAmount = _loan.collateralAmount;

        require(amount >= (collateralAmount.mul(rate) / 1
ether).mul(liquidationThreshold) / 1 ether, "PawnShop::liquidate: Loan cannot be
liquidated!");

        uint256 liquidationAmount = _liquidate(collateralAmount, isCORE);

        delete loans[account][token];

        emit Liquidation(msg.sender, account, collateralAmount, amount,
liquidationAmount, isCORE);
    }
```

Recommendations

4. There should be a function to change profiteer address

- CLEND-4
- Severity: Low
- Impact: Low
- Target: PawnShop.sol
- Category: Profiteer Address
- Finding Type: Dynamic

The contract should have a function allowing governance to change the profiteer address in case the profiteer address is exploited (low probability).

```
function changeProfiteer(address _profiteer) external onlyGovernance {
      require(_profiteer != address(0), "PawnShop::changeProfiteer: _profiteer must
be not zero");
      profiteer = _profiteer;
   }
```

5. Ensure floor price of tokens

- CLEND-5
- Severity: Low
- Impact: Low
- Target: PawnShop.sol
- Category: Token Price Floor
- Finding Type: Dynamic

This issue is related to the deployment inputs of the contract. It is a must to ensure that floor prices of the `CORE` and `coreDAO` tokens must be the minimum price of CORE and coreDAO tokens.

Conclusion

Arcadia identified issues that occurred at hash #4948210cd99149799e917a9600ca2b3965ee2157.

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.