# Smart contract security audit

# Collective Finance

v.1.6

## Table of Contents

# 1.0 Introduction

## 1.1 Project engagement

During March of 2023, Collective Finance team engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. Collective Finance provided CTDSec with access to their code repository and whitepaper.

## 1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the network's fast-paced and rapidly changing environment, we at CTDSec recommend that Collective Finance team put in place a bug bounty program to encourage further and active analysis of the smart contract.

# 2.0 Coverage

## 2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the Collective Finance contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:

Source file:

BSC:

CLF Token - https://bscscan.com/token/0xa76c73f3b2ae69ee40d6bf768e8f1036957d11ff#code

GLD Token - https://bscscan.com/token/0xb9d162d5e7c385e4104737d012c2979536f1b664#code

Presale - https://bscscan.com/address/0xd55383823714cbc4713cab57728887a4763c0e2e#code

Arbitrum:

CGLD - https://arbiscan.io/address/0x0d702ebdef2c47eb33951098db4f06bd8cca8105#code

Presale - https://arbiscan.io/address/0xda3f92b76d837387b142c831317158a9f7927edd#code

CLF - https://arbiscan.io/token/0x2207d6aaebf4e80ae9de54ba13fff62c1e26d481

Fixed versions:

https://bscscan.com/address/0xb67fe4d73c30e5ef435d4af06adfff4c08e45aad#code

https://bscscan.com/address/0x8f3405799eea3a5fbfe3f9faa9839e1c6947a604#code

## 2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

| № | Issue description. | Checking status |
|---|---|---|
| 1 | Compiler warnings. | PASSED |
| 2 | Race conditions and Reentrancy. Cross-function race conditions. | PASSED |
| 3 | Possible delays in data delivery. | PASSED |
| 4 | Oracle calls. | PASSED |
| 5 | Front running. | PASSED |
| 6 | Timestamp dependence. | PASSED |
| 7 | Integer Overflow and Underflow. | FIXED ISSUES |
| 8 | DoS with Revert. | PASSED |
| 9 | DoS with block gas limit. | PASSED |
| 10 | Methods execution permissions. | PASSED |
| 11 | Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial | PASSED |

| | | |
|---|---|---|
| | losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc. | |
| 12 | The impact of the exchange rate on the logic. | PASSED |
| 13 | Private user data leaks. | PASSED |
| 14 | Malicious Event log. | PASSED |
| 15 | Scoping and Declarations. | PASSED |
| 16 | Uninitialized storage pointers. | PASSED |
| 17 | Arithmetic accuracy. | PASSED |
| 18 | Design Logic. | PASSED |
| 19 | Cross-function race conditions. | PASSED |
| 20 | Safe Zeppelin module. | PASSED |
| 21 | Fallback function security. | PASSED |
| 22 | Overpowered functions / Wrong Owner privileges | FIXED ISSUES |

# 3.0 Security Issues

## 3.1 High severity issues [0]

No high severity issues found.

## 3.2 Medium severity issues [2]

**Contract: CGoldToken.sol**

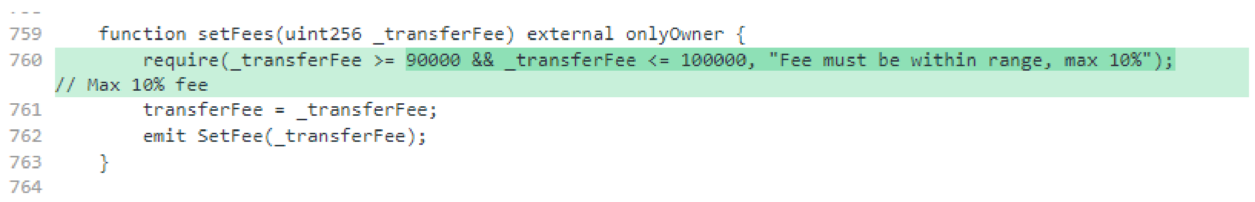
1. Require doesn't emit any error message

**Function**: setFees()

**Issue**: If the transfer fees aren't set less than 100000 could produce a subtraction overflow in uint256 fee = amount.sub(remAmount) on _transferFrom().

**Solution**: Add require _transferFee <= 100000.

**Fix**: Require is correctly set now:



```
    function setFees(uint256 _transferFee) external onlyOwner {
        require(_transferFee >= 90000 && _transferFee <= 100000, "Fee must be within range, max 10%");
// Max 10% fee
        transferFee = _transferFee;
        emit SetFee(_transferFee);
    }
```

**Contract: Sale.sol**

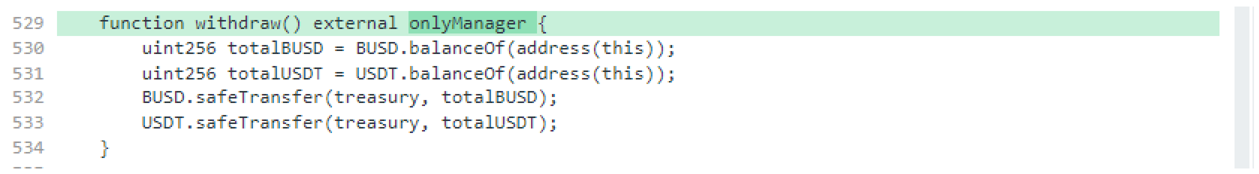
2. Withdraw function privileges are not correctly checked

**Function**: withdraw()

**Issue**: Any user can withdraw ERC20 tokens and not only the owner as expected.

**Solution**: Add to function external onlyManager.

**Fix**: Privileges are correctly set to onlyManager now:


```
    function withdraw() external onlyManager {
        uint256 totalBUSD = BUSD.balanceOf(address(this));
        uint256 totalUSDT = USDT.balanceOf(address(this));
        BUSD.safeTransfer(treasury, totalBUSD);
        USDT.safeTransfer(treasury, totalUSDT);
    }
```

## 3.3 Low severity issues [2]

**Contract: CGoldToken.sol**

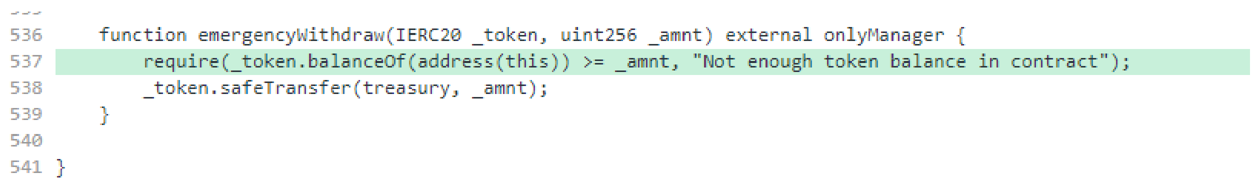
1. Require doesn't emit any error message

**Function**: setFees()

**Issue**: All requires in solidity should include a error message to have better error handling.

**Solution**: Add require(_transferFee >= 90000, "_error_message")

**Fix**: Error message is included now:


```
    function emergencyWithdraw(IERC20 _token, uint256 _amnt) external onlyManager {
        require(_token.balanceOf(address(this)) >= _amnt, "Not enough token balance in contract");
        _token.safeTransfer(treasury, _amnt);
    }

}
```

**Contract: Sale.sol**

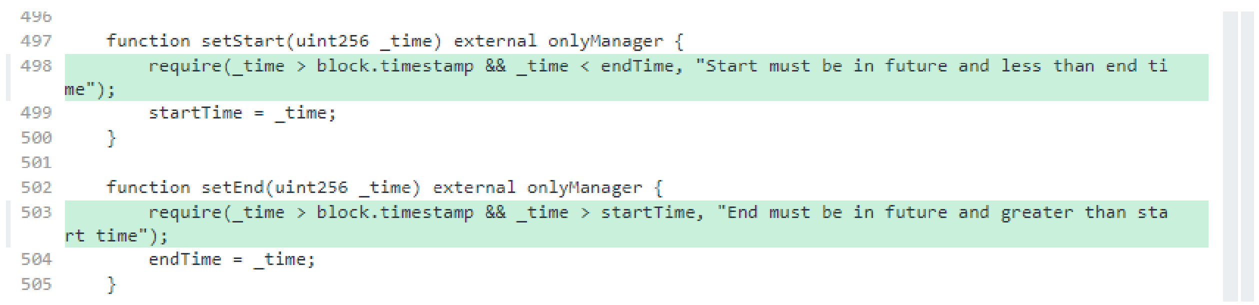
2. Start and endtime aren't checked at the sale

**Functions: setStart() and setEnd()**

**Issue:** Contract is not checking start and endtiem during the sale and can mislead to business errors.

**Solution:** Add require startTime > block.timestamp and startTime < endTime & endTime > block.timestamp and endTime > startTime.

**Fix:** Requires are correctly set now:


```
    function setStart(uint256 _time) external onlyManager {
        require(_time > block.timestamp && _time < endTime, "Start must be in future and less than end time");
        startTime = _time;
    }

    function setEnd(uint256 _time) external onlyManager {
        require(_time > block.timestamp && _time > startTime, "End must be in future and greater than start time");
        endTime = _time;
    }
```

## 3.4 Informational issues [1]

**Contract: Sale.sol**

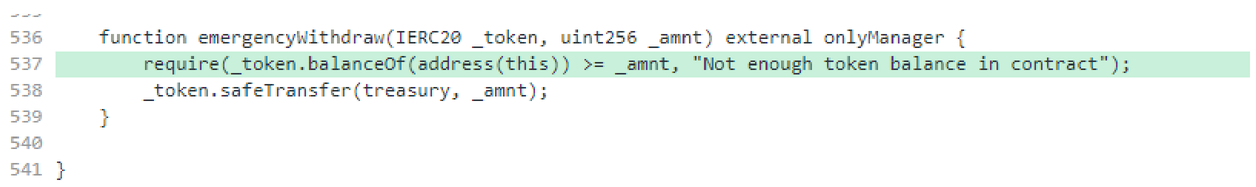
1. Amount not checked before withdraw of tokens

**Functions:** emergencyWithdraw()

**Issue:** Contract is not checking the amount of tokens that are going to be withdrawn.

**Solution:** Add a required **IERC20.balanceOf(address(this)) > amount**.

**Fix**: Balance check is included at the require now:

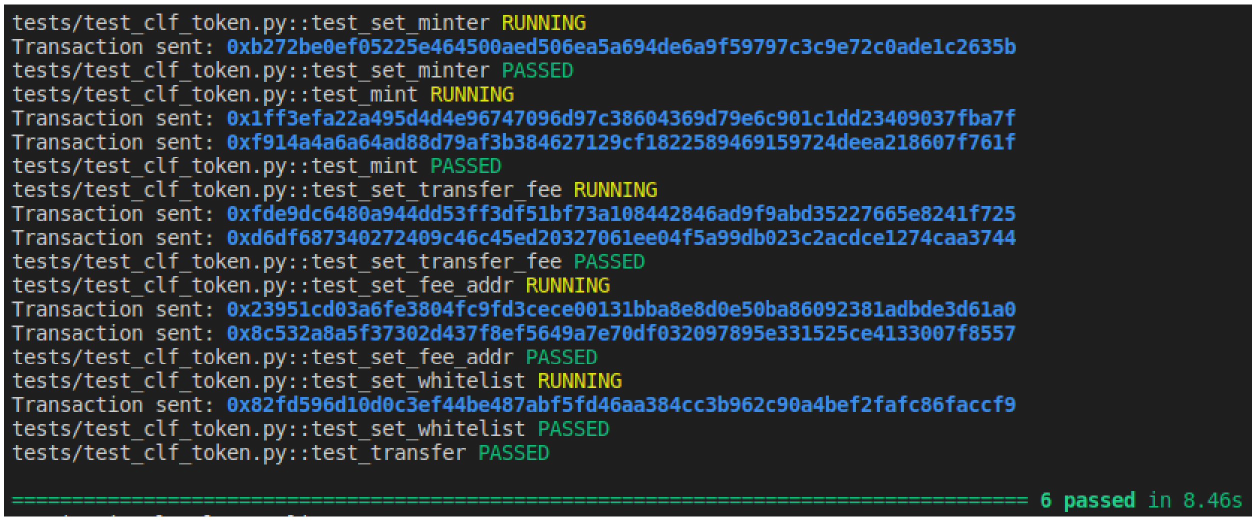

```
    function emergencyWithdraw(IERC20 _token, uint256 _amnt) external onlyManager {
        require(_token.balanceOf(address(this)) >= _amnt, "Not enough token balance in contract");
        _token.safeTransfer(treasury, _amnt);
    }

}
```

# 4.0 Testing coverage - python

During the testing phase, custom use cases were written to cover all the logic of contracts in python language. *<u>*Check "5 Annexes" to see the testing code.</u>*

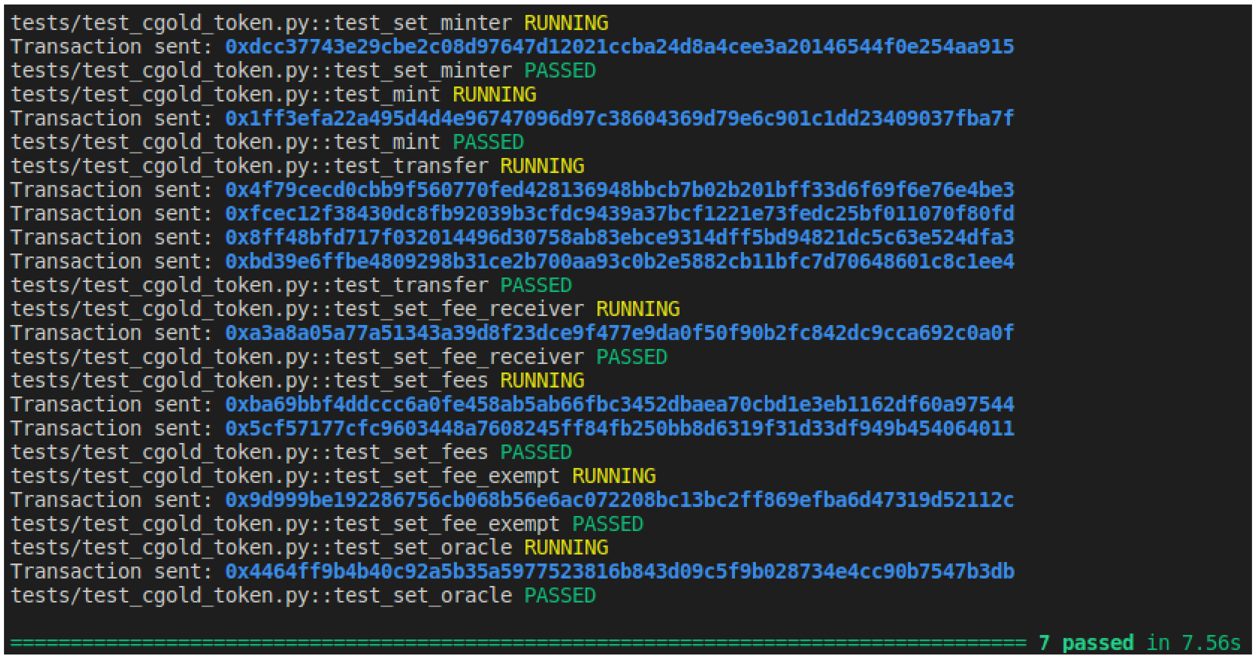
**<u>CLF TOKEN testing:</u>**



```
tests/test_clf_token.py::test_set_minter RUNNING
Transaction sent: 0xb272be0ef05225e464500aed506ea5a694de6a9f59797c3c9e72c0ade1c2635b
tests/test_clf_token.py::test_set_minter PASSED
tests/test_clf_token.py::test_mint RUNNING
Transaction sent: 0x1ff3efa22a495d4d4e96747096d97c38604369d79e6c901c1dd23409037fba7f
Transaction sent: 0xf914a4a6a64ad88d79af3b384627129cf1822589469159724deea218607f761f
tests/test_clf_token.py::test_mint PASSED
tests/test_clf_token.py::test_set_transfer_fee RUNNING
Transaction sent: 0xfde9dc6480a944dd53ff3df51bf73a108442846ad9f9abd35227665e8241f725
Transaction sent: 0xd6df687340272409c46c45ed20327061ee04f5a99db023c2acdce1274caa3744
tests/test_clf_token.py::test_set_transfer_fee PASSED
tests/test_clf_token.py::test_set_fee_addr RUNNING
Transaction sent: 0x23951cd03a6fe3804fc9fd3cece00131bba8e8d0e50ba86092381adbde3d61a0
Transaction sent: 0x8c532a8a5f37302d437f8ef5649a7e70df032097895e331525ce4133007f8557
tests/test_clf_token.py::test_set_fee_addr PASSED
tests/test_clf_token.py::test_set_whitelist RUNNING
Transaction sent: 0x82fd596d10d0c3ef44be487abf5fd46aa384cc3b962c90a4bef2fafc86faccf9
tests/test_clf_token.py::test_set_whitelist PASSED
tests/test_clf_token.py::test_transfer PASSED

===================================================================================== 6 passed in 8.46s
```

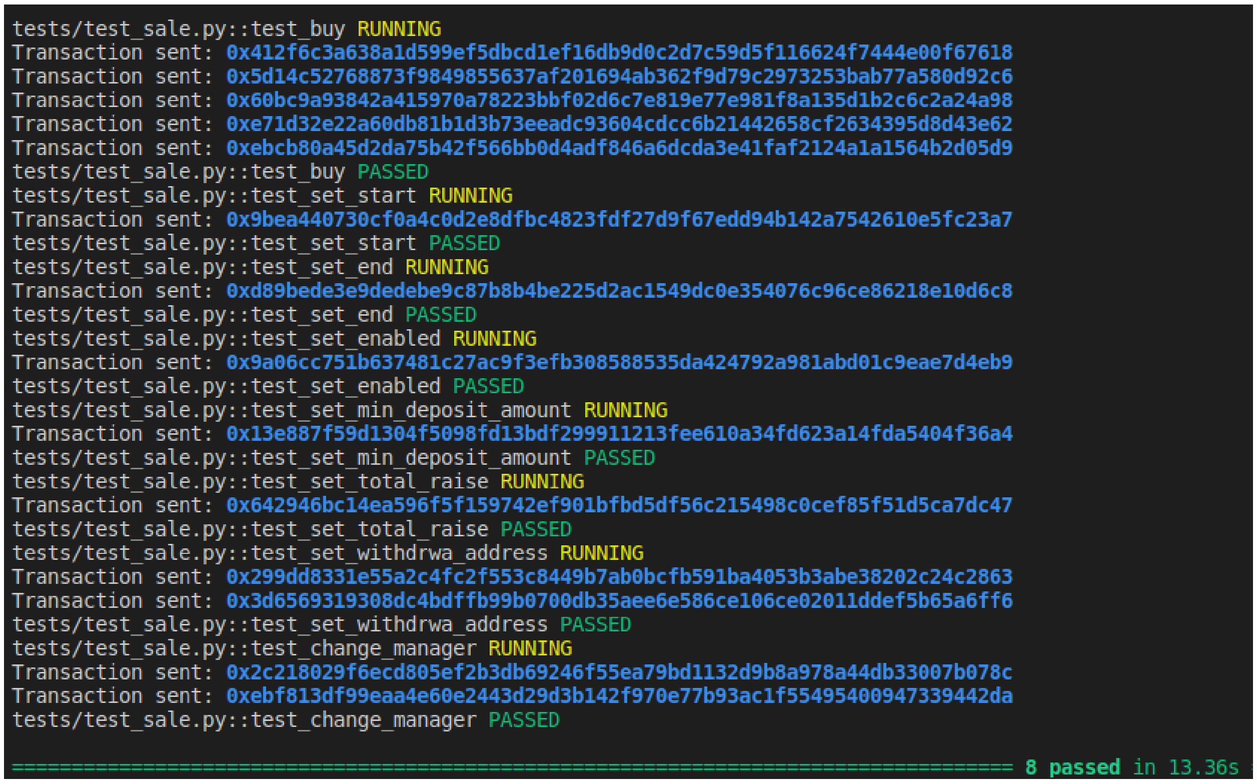
CGLD Token testing:



```
tests/test_cgold_token.py::test_set_minter RUNNING
Transaction sent: 0xdcc37743e29cbe2c08d97647d12021ccba24d8a4cee3a20146544f0e254aa915
tests/test_cgold_token.py::test_set_minter PASSED
tests/test_cgold_token.py::test_mint RUNNING
Transaction sent: 0x1ff3efa22a495d4d4e96747096d97c38604369d79e6c901c1dd23409037fba7f
tests/test_cgold_token.py::test_mint PASSED
tests/test_cgold_token.py::test_transfer RUNNING
Transaction sent: 0x4f79cecd0cbb9f560770fed428136948bbcb7b02b201bff33d6f69f6e76e4be3
Transaction sent: 0xfcec12f38430dc8fb92039b3cfdc9439a37bcf1221e73fedc25bf011070f80fd
Transaction sent: 0x8ff48bfd717f032014496d30758ab83ebce9314dff5bd94821dc5c63e524dfa3
Transaction sent: 0xbd39e6ffbe4809298b31ce2b700aa93c0b2e5882cb11bfc7d70648601c8c1ee4
tests/test_cgold_token.py::test_transfer PASSED
tests/test_cgold_token.py::test_set_fee_receiver RUNNING
Transaction sent: 0xa3a8a05a77a51343a39d8f23dce9f477e9da0f50f90b2fc842dc9cca692c0a0f
tests/test_cgold_token.py::test_set_fee_receiver PASSED
tests/test_cgold_token.py::test_set_fees RUNNING
Transaction sent: 0xba69bbf4ddccc6a0fe458ab5ab66fbc3452dbaea70cbd1e3eb1162df60a97544
Transaction sent: 0x5cf57177cfc9603448a7608245ff84fb250bb8d6319f31d33df949b454064011
tests/test_cgold_token.py::test_set_fees PASSED
tests/test_cgold_token.py::test_set_fee_exempt RUNNING
Transaction sent: 0x9d999be192286756cb068b56e6ac072208bc13bc2ff869efba6d47319d52112c
tests/test_cgold_token.py::test_set_fee_exempt PASSED
tests/test_cgold_token.py::test_set_oracle RUNNING
Transaction sent: 0x4464ff9b4b40c92a5b35a5977523816b843d09c5f9b028734e4cc90b7547b3db
tests/test_cgold_token.py::test_set_oracle PASSED

===================================================================================== 7 passed in 7.56s
```

Sale contract Testing:



```
tests/test_sale.py::test_buy RUNNING
Transaction sent: 0x412f6c3a638a1d599ef5dbcd1ef16db9d0c2d7c59d5f116624f7444e00f67618
Transaction sent: 0x5d14c52768873f9849855637af201694ab362f9d79c2973253bab77a580d92c6
Transaction sent: 0x60bc9a93842a415970a78223bbf02d6c7e819e77e981f8a135d1b2c6c2a24a98
Transaction sent: 0xe71d32e22a60db81b1d3b73eeadc93604cdcc6b21442658cf2634395d8d43e62
Transaction sent: 0xebcb80a45d2da75b42f566bb0d4adf846a6dcda3e41faf2124a1a1564b2d05d9
tests/test_sale.py::test_buy PASSED
tests/test_sale.py::test_set_start RUNNING
Transaction sent: 0x9bea440730cf0a4c0d2e8dfbc4823fdf27d9f67edd94b142a7542610e5fc23a7
tests/test_sale.py::test_set_start PASSED
tests/test_sale.py::test_set_end RUNNING
Transaction sent: 0xd89bede3e9dedebe9c87b8b4be225d2ac1549dc0e354076c96ce86218e10d6c8
tests/test_sale.py::test_set_end PASSED
tests/test_sale.py::test_set_enabled RUNNING
Transaction sent: 0x9a06cc751b637481c27ac9f3efb308588535da424792a981abd01c9eae7d4eb9
tests/test_sale.py::test_set_enabled PASSED
tests/test_sale.py::test_set_min_deposit_amount RUNNING
Transaction sent: 0x13e887f59d1304f5098fd13bdf299911213fee610a34fd623a14fda5404f36a4
tests/test_sale.py::test_set_min_deposit_amount PASSED
tests/test_sale.py::test_set_total_raise RUNNING
Transaction sent: 0x642946bc14ea596f5f159742ef901bfbd5df56c215498c0cef85f51d5ca7dc47
tests/test_sale.py::test_set_total_raise PASSED
tests/test_sale.py::test_set_withdrwa_address RUNNING
Transaction sent: 0x299dd8331e55a2c4fc2f553c8449b7ab0bcfb591ba4053b3abe38202c24c2863
Transaction sent: 0x3d6569319308dc4bdffb99b0700db35aee6e586ce106ce02011ddef5b65a6ff6
tests/test_sale.py::test_set_withdrwa_address PASSED
tests/test_sale.py::test_change_manager RUNNING
Transaction sent: 0x2c218029f6ecd805ef2b3db69246f55ea79bd1132d9b8a978a44db33007b078c
Transaction sent: 0xebf813df99eaa4e60e2443d29d3b142f970e77b93ac1f55495400947339442da
tests/test_sale.py::test_change_manager PASSED

==================================================================================== 8 passed in 13.36s
```

## 5.0 Annexes

CLF Token:

```python
from brownie import reverts, CLFToken

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    ONE_ETH,
    get_account,
    get_owner_role,
)

from scripts.deploy import (
    deploy_clf_token
)

def test_set_minter(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    token = deploy_clf_token(owner)

    # assert
    with reverts():
        token.setMinter(other, True, {"from": other})
    assert token.minter(other) == False
    token.setMinter(other, True, {"from": owner})
    assert token.minter(other) == True

def test_mint(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    token = deploy_clf_token(owner)

    # assert
    with reverts("CLF: not minter"):
        token.mint(extra, 100, {"from": other})
    token.setMinter(other, True, {"from": owner})
    with reverts("CLF: Maximum supply is 10 million"):
        token.mint(extra, ONE_ETH * 10000001, {"from": other})

    assert token.balanceOf(extra) == 0
    token.mint(extra, ONE_ETH, {"from": other})
    assert token.balanceOf(extra) == ONE_ETH

def test_set_transfer_fee(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    token = deploy_clf_token(owner)

    # assert
    with reverts():
        token.setTransferFee(10, {"from": other})
    with reverts("CLF: transfer fee should be less than 15%"):
        token.setTransferFee(20, {"from": owner})

    assert token.transferFee() == 5
    token.setTransferFee(10, {"from": owner})
    assert token.transferFee() == 10

def test_set_fee_addr(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    token = deploy_clf_token(owner)

    # assert
    with reverts():
        token.setFeeAddress(extra, {"from": other})
    with reverts("CLF: fee address can't be zero address"):
        token.setFeeAddress(ZERO_ADDRESS, {"from": owner})

    assert token.feeAddress() == owner
    token.setFeeAddress(extra, {"from": owner})
    assert token.feeAddress() == extra
```

```python
def test_set_whitelist(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    token = deploy_clf_token(owner)

    # assert
    with reverts():
        token.setWhitelist(extra, True, {"from": other})

    assert token.whitelist(extra) == False
    token.setWhitelist(extra, True, {"from": owner})
    assert token.whitelist(extra) == True

def test_transfer(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    token = deploy_clf_token(owner)

    # assert

    # transfer with fees
    assert token.balanceOf(other) == 0
    assert token.balanceOf(owner) == token.totalSupply()
    fee = (ONE_ETH * token.transferFee()) / 100
    tx = token.transfer(other, ONE_ETH, {"from": owner})
    assert token.balanceOf(other) == ONE_ETH - fee
    assert token.balanceOf(owner) == token.totalSupply() - (ONE_ETH - fee)
    assert tx.events["Transfer"] is not None
    assert len(tx.events["Transfer"]) == 2
    # transfer without fees -> to
    token.setWhitelist(extra, True, {"from": owner})
    assert token.balanceOf(extra) == 0
    assert token.balanceOf(owner) == token.totalSupply() - (ONE_ETH - fee)
    tx2 = token.transfer(extra, ONE_ETH, {"from": owner})
    assert token.balanceOf(extra) == ONE_ETH
    assert token.balanceOf(owner) == token.totalSupply() - (ONE_ETH - fee)
- ONE_ETH
    assert tx2.events["Transfer"] is not None
    assert len(tx2.events["Transfer"]) == 1
    # transfer without fees -> from
    assert token.balanceOf(extra) == ONE_ETH
    assert token.balanceOf(other) == ONE_ETH - fee
    tx3 = token.transfer(other, ONE_ETH, {"from": extra})
    assert token.balanceOf(extra) == 0
    assert token.balanceOf(other) == ONE_ETH + (ONE_ETH - fee)
    assert tx3.events["Transfer"] is not None
    assert len(tx3.events["Transfer"]) == 1
```

<u>CGLD Token:</u>

```
from brownie import reverts, C_Gold

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    ONE_ETH,
    get_account
)

from scripts.deploy import (
    deploy_cgold_token
)

def test_set_minter(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    token = deploy_cgold_token(owner)

    # assert
    with reverts():
        token.setMinter(other, {"from": other})
    assert token.minter() == ZERO_ADDRESS
    token.setMinter(other, {"from": owner})
    assert token.minter() == other

def test_mint(only_local):
    # arrage
    owner = get_account(0)
```

```python
    other = get_account(1)
    extra = get_account(2)
    token = deploy_cgold_token(owner)

    # assert
    with reverts("Caller is not admin"):
        token.mint(extra, 100, {"from": other})
    token.setMinter(other, {"from": owner})

    assert token.balanceOf(extra) == 0
    token.mint(extra, ONE_ETH, {"from": other})
    assert token.balanceOf(extra) == ONE_ETH

def test_transfer(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    token = deploy_cgold_token(owner)

    # assert
    with reverts("Transfer To Zero"):
        token.transfer(ZERO_ADDRESS, 100, {"from": owner})
    with reverts("Transfer Amt Zero"):
        token.transfer(other, 0, {"from": owner})

    # mint some tokens
    token.mint(owner, 1000, {"from": owner})
    token.mint(other, 500, {"from": owner})
    token.mint(extra, 100, {"from": owner})

    # transfer without fees FROM
    tx = token.transfer(other, 100, {"from": owner})
    assert tx.events["Transfer"] is not None
    assert token.balanceOf(owner) == 900
    assert token.balanceOf(other) == 600
    # transfer with fees
    token.setFees(90000, {"from": owner})
    tx_1 = token.transfer(extra, 100, {"from": other})
    assert tx_1.events["Transfer"] is not None
    assert len(tx_1.events["Transfer"]) == 2
    assert token.balanceOf(other) == 500
    assert token.balanceOf(extra) == 190

    # transfer without fees TO
    token.setFees(100000, {"from": owner})
    tx_2 = token.transfer(owner, 100, {"from": other})
    assert tx_2.events["Transfer"] is not None
    assert len(tx_2.events["Transfer"]) == 1
    assert token.balanceOf(other) == 400
    assert token.balanceOf(owner) == 1010 # 900 + 100 (amount) + 10 (tx_1
fee)

    # transfer without enought balance
    with reverts("Insufficient Balance"):
        token.transfer(other, 250, {"from": extra})

    # transfer with incorrect fees
    token.setFees(200000, {"from": owner})
    with reverts():
        token.transfer(extra, 100, {"from": other})

def test_set_fee_receiver(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    token = deploy_cgold_token(owner)

    # assert
    with reverts():
        token.setFeeReceiver(other, {"from": other})
    assert token.feeReceiver() == owner
    token.setFeeReceiver(other, {"from": owner})
    assert token.feeReceiver() == other

def test_set_fees(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    token = deploy_cgold_token(owner)

    # assert
    with reverts():
        token.setFees(100000, {"from": other})
    with reverts():
        token.setFees(80000, {"from": owner})
    assert token.transferFee() == 100000
    new_fee = 95000
    token.setFees(95000, {"from": owner}) # 5% fee
    assert token.transferFee() == new_fee

def test_set_fee_exempt(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    token = deploy_cgold_token(owner)

    # assert
    with reverts():
        token.setFeeExempt(other, True, {"from": other})
    assert token.feeExempt(other) == False
    token.setFeeExempt(other, True, {"from": owner})
    assert token.feeExempt(other) == True

def test_set_oracle(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    token = deploy_cgold_token(owner)

    # assert
    with reverts():
        token.setOracle(other, {"from": other})
    assert token.oracle() == ZERO_ADDRESS
    token.setOracle(other, {"from": owner})
    assert token.oracle() == other
```

Sale contracts:

```
from brownie import reverts, Sale

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    ONE_ETH,
    evm_increase_time,
    get_timestamp,
    get_account,
)

from scripts.deploy import (
    deploy_sale,
    deploy_test_erc20
)

def test_buy(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)
    usdt_mock = deploy_test_erc20(owner)
    busd_mock = deploy_test_erc20(owner)
    sale = Sale.deploy(get_timestamp(), owner,
        usdt_mock.address, busd_mock.address, {"from": owner})

    # assert
    with reverts("Sale Not Open"):
        sale.buywithBUSD(ONE_ETH, {"from": other})

    evm_increase_time(86400 * 2) # increase time two days to open sale

    with reverts("Should deposit more than minimum deposit amount per one
time"):
        sale.buywithBUSD(ONE_ETH, {"from": other})
    sale.setMinDepositAmount(ONE_ETH, {"from": owner})
    sale.setTotalRaise(ONE_ETH, {"from": owner})
    with reverts("Total Cap Reached"):
        sale.buywithBUSD(ONE_ETH * 2, {"from": other})
    sale.setTotalRaise(500000 * ONE_ETH, {"from": owner})
    with reverts("Insufficient balance"):
        sale.buywithBUSD(ONE_ETH, {"from": other})
    busd_mock.mint(other, ONE_ETH * 2, {"from": other})
    with reverts("Insufficient allowance"):
        sale.buywithBUSD(ONE_ETH, {"from": other})

    # join sale
    busd_mock.approve(sale.address, ONE_ETH, {"from": other})
    assert sale.users(other) == 0
    assert sale.total_deposited() == 0
    sale.buywithBUSD(ONE_ETH, {"from": other})
    assert sale.users(other) == ONE_ETH
    assert sale.total_deposited() == ONE_ETH

    # another join sale
    busd_mock.mint(extra, ONE_ETH, {"from": extra})
    busd_mock.approve(sale.address, ONE_ETH, {"from": extra})
    assert sale.users(extra) == 0
    assert sale.total_deposited() == ONE_ETH
    sale.buywithBUSD(ONE_ETH, {"from": extra})
    assert sale.users(extra) == ONE_ETH
    assert sale.total_deposited() == ONE_ETH * 2

def test_set_start(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    sale = deploy_sale(owner)

    # assert
    with reverts():
        sale.setStart(get_timestamp(2), {"from": other})

    new_time = get_timestamp(2)
    assert sale.startTime() != new_time
    sale.setStart(new_time, {"from": owner})
    assert sale.startTime() == new_time

def test_set_end(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    sale = deploy_sale(owner)

    # assert
    with reverts():
        sale.setEnd(get_timestamp(2), {"from": other})

    new_time = get_timestamp(10)
    assert sale.endTime() != new_time
    sale.setEnd(new_time, {"from": owner})
    assert sale.endTime() == new_time

def test_set_enabled(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    sale = deploy_sale(owner)

    # assert
    with reverts():
        sale.setEnabled(False, {"from": other})

    assert sale.enabled() == True
    sale.setEnabled(False, {"from": owner})
    assert sale.enabled() == False

def test_set_min_deposit_amount(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    sale = deploy_sale(owner)

    # assert
    with reverts():
        sale.setMinDepositAmount(ONE_ETH, {"from": other})

    assert sale.minDepositAmount() == 50 * ONE_ETH
    sale.setMinDepositAmount(ONE_ETH, {"from": owner})
    assert sale.minDepositAmount() == ONE_ETH

def test_set_total_raise(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    sale = deploy_sale(owner)

    # assert
    with reverts():
        sale.setTotalRaise(10 * ONE_ETH, {"from": other})

    assert sale.totalraiseCapPublic() == 500000 * ONE_ETH
    sale.setTotalRaise(10 * ONE_ETH, {"from": owner})
    assert sale.totalraiseCapPublic() == 10 * ONE_ETH

def test_set_withdrwa_address(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    sale = deploy_sale(owner)

    # assert
    with reverts():
        sale.setWithdrawAddress(other, {"from": other})
    with reverts("Zero address"):
        sale.setWithdrawAddress(ZERO_ADDRESS, {"from": owner})

    assert sale.treasury() == owner
    sale.setWithdrawAddress(other, {"from": owner})
    assert sale.treasury() == other

def test_change_manager(only_local):
    # arrage
    owner = get_account(0)
    other = get_account(1)
    sale = deploy_sale(owner)

    # assert
    with reverts():
        sale.changeManager(other, {"from": other})
    with reverts("Zero address"):
        sale.changeManager(ZERO_ADDRESS, {"from": owner})

    assert sale.manager() == owner
    sale.changeManager(other, {"from": owner})
    assert sale.manager() == other
```

# 6.0 Summary of the audit

All issues were solved by the development team, contracts are safe to be deployed.