Audit Report of yield Farming Contract

Audited by Muhammad Talha

Yield farming Address:
https://goerli.etherscan.io/address/0x337c6d67ef64984cd919f0640711a54d1e61e822#code
Yield farming Address V1 after changes:
https://goerli.etherscan.io/address/0xd03f2870c6aedcae751ae3ed383535c5ad7cc0b0

Total : 11
Major changes or Errors: 5
Minor changes:7
Fixed : all

These are the following bugs, issues and minor changes need to be fixed in tis contracts

```
Icomptroller comptroller; // add visibility
IcToken cDai; // add visibility
IERC20 dai; // add visibility
```

1_;// audit: define visibility of Icomptroller comptroller variable(instance).
2_:// audit: define visibility of IcToken cDai variable(instance).
3_:// audit: define visibility of IERC20 dai; variable(instance).

```
IERC20 dai; // add visibility

uint borrowFactor = 70;// add visibility
address public owner;
```

4_:// audit: define visibility of borrowFactor variable(instance) and add the uint256 instead uint that would be best practice.

```
function openPosition(uint initialAmount) external {

    //the amount should be send from the owner to the contract first
    //need a require check that owner have that specific amount of token
    // add require statement initialAmount != 0
    //add require onlyOwner can use this for himself
```

5_://audit: need a required statement to check only the owner can call this function otherwise anyone else can cause this function.

Explanation: in this function we are using the already sent token in the contract because anyone can call this function and can do transactions with those tokens.Or any one can call withdraw function so it will make the contract balance zero so it can cause problem for the owner so Owner needs to send tokens at the time of function call to the contract and then use them further.

6_://audit:add required statement to check the initialAmount != 0;

7_://audit: first we have to check the balance of the tokens we are using, it will >= to initialAmount.

```
    uint nextCollateralAmount = initialAmount;
    for(uint i = 0; i < 5; i++) {
        nextCollateralAmount = _supplyAndBorrow(nextCollateralAmount); // put this _supplyAndBorrow
    }
}
```

8_://audit: need to check that the _supplyAndBorrow != 0 Otherwise break the loop there to work properly or after multiplication to zero the whole value will be zero and the whole function value will work on zero value throughout all the iterations.

```
function closePosition() external {
    //add require onlyOwner can use this for himself
    uint balanceBorrow = cDai.borrowBalanceCurrent(address(this));
   //  add require statement that  balanceBorrow != 0
    dai.approve(address(cDai), balanceBorrow);

    cDai.repayBorrow(balanceBorrow);
    uint balancecDai = cDai.balanceOf(address(this));
    //add require statement that balancecDai != 0;
    cDai.redeem(balancecDai);
}
```

9_://audit: add require onlyOwner can use this for himself otherwise anyone can call this function and make it close.

10_://audit: add required statement that balanceBorrow != 0 in close position Function
Explanation: we are getting it from the third party contract. If somehow it isn't working and doesn't return anything so we should know by adding a proper error msg.

```
function withdrawContractBalance(IERC20 tokenAddress,uint256 _amount) external { // anyone can withdraw
    require(tokenAddress.balanceOf(address(this)) >= _amount,
        "Contract doesent have sufficient amount of tokens to withdraw.");
    tokenAddress.transfer(owner,_amount);
}
```

11_://audit: add require statement that only owner can call this function
Explanation: withdraw function can be called by anyone and can make the contract balance zero so the owner couldn't be able to make the openPosition function work properly.