HALBORN

EasyFi Lending Contracts

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 6th, 2021 - October 8th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/07/2021	Alessandro Cara
0.1	Document Edit	10/07/2021	Alessandro Cara
0.3	Document Edit	10/08/2021	Gokberk Gulgun
0.4	Final Draft	10/08/2021	Alessandro Cara
0.5	Document Review	10/08/2021	Gabi Urrutia
1.0	Remediation Plan	10/11/2021	Alessandro Cara
1.1	Remediation Plan Review	10/12/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Alessandro Cara	Halborn	Alessandro.Cara@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

EasyFi engaged Halborn to conduct a security assessment on their smart contracts beginning on October 6th, 2021, and ending October 8th, 2021. The security assessment was scoped to the lending smart contracts which were made available to Halborn on GitHub. These smart contracts were a fork of Compound Finance contracts with only minor changes. EasiFy requested that these changes were reviewed before the release to production.

1.2 AUDIT SUMMARY

The team at Halborn was provided one week for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart contracts security expert, with experience in advanced penetration testing, smart contract hacking, and a deep knowledge in multiple blockchain protocols.

The purpose of this audit was the following:

- Ensure that smart contract functions are intended
- Identify potential security issues with the smart contracts

Halborn identified several security issues within the code base. The most severe of the issues stemmed from the use of an outdated Chainlink oracle API which does not allow to perform a number of checks on the returned price data. This could lead to logic flaws in the smart contract set.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverable set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure contract development.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Truffle,Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.

4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The security assessment was scoped to the smart contract folders available at https://github.com/easyfinetwork/defi/tree/main/contracts/lending and the following commit IDs:

- Commit 1 ID: 0fa88aad33ad27c47bf270cd97dfa251fc72156b
- Commit 2 ID: 4c24118e5d67f77f6ad5a85b208707e874719be6

In more details, the modified contracts were the following:

- contracts/lending/contracts/Governance/USDC.sol
- contracts/lending/contracts/ErrorReporter.sol
- contracts/lending/contracts/CTokenInterfaces.sol
- contracts/lending/contracts/CToken.sol
- contracts/lending/contracts/CEther.sol
- contracts/lending/contracts/CErc20Delegator.sol
- contracts/lending/contracts/CErc20.sol
- contracts/lending/contracts/SimplePriceOracle.sol

The fixes implemented by EasyFi can be found here: Fixes

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	9

LIKELIHOOD

IMPACT

		(HAL-01)		
(HAL-07) (HAL-08)	(HAL-02)			
(HAL-03) (HAL-04) (HAL-05) (HAL-06) (HAL-09) (HAL-10) (HAL-11)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - MISUSE OF AN ORACLE	Medium	SOLVED - 10/11/2021
HAL02 - FLOATING PRAGMA	Low	SOLVED - 10/11/2021
HAL03 - USAGE OF ASSERT	Informational	SOLVED - 10/11/2021
HAL04 - LACK OF DUPLICATE AGGREGATOR CHECK	Informational	ACKNOWLEDGED - 10/11/2021
HAL05 - LACK OF ZERO ADDRESS CHECK	Informational	ACKNOWLEDGED - 10/11/2021
HAL06 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 10/11/2021
HAL07 - MISSING EVENTS EMISSION	Informational	SOLVED - 10/11/2021
HAL08 - EXPERIMENTAL FEATURES ENABLED	Informational	ACKNOWLEDGED - 10/11/2021
HAL09 - UNUSED CODE SECTION	Informational	SOLVED - 10/11/2021
HAL10 - USAGE OF ABI ENCODEPACKED FUNCTION	Informational	ACKNOWLEDGED - 10/11/2021
HAL11 - USE OF INLINE ASSEMBLY	Informational	ACKNOWLEDGED - 10/11/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISUSE OF AN ORACLE - MEDIUM

Description:

SimplePriceOracle.sol is calling latestAnswer to get the last asset price. This method will return the last value, but it will not allow to check if the data is fresh. On the other hand, calling the method latestRoundData allows to run additional validation.

Code Location:

SimplePriceOracle.sol Lines# 44-142

Listing 1

```
1   function getUnderlyingPrice(CToken cToken) public view returns (
        uint256) {
2         //if(!isStockAsset[_asset]){
3         if (compareStrings(cToken.symbol(), "cETH")) {
4             address _aggregator = aggregator[bnbUnderlying];
5             (MathError error, uint256 price) = mulUInt(
6                 uint256(AccessControlledAggregator(_aggregator).
                      latestAnswer()),
7                 10**10
8             );
9             assert(error == MathError.NO_ERROR);
10            return price;
11        }
12        [Redacted for brevity]
```

Recommendation:

It is recommended to use the latestRoundData function to retrieve an asset's price instead. Checks on the return data should be introduced with proper revert messages if the price is stale or the round is **uncomplete**. For example:

Listing 2

```
        (
            roundId,
            rawPrice,
            ,
            updateTime,
            answeredInRound
         ) = AccessControlledAggregator(_aggregator).
             latestRoundData();
         require(rawPrice > 0, "Chainlink price <= 0");
         require(updateTime != 0, "Incomplete round");
         require(answeredInRound >= roundId, "Stale price");
```

Remediation Plan:

SOLVED : EasyFi team replaced the API latestAnswer with latestRoundData in all instances.

3.2 (HAL-02) FLOATING PRAGMA - LOW

Description:

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively or recently released pragma versions may have unknown security vulnerabilities.

Listing 3

```
1 pragma solidity ^0.5.16;
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider locking the pragma version as it is not recommended to use a floating pragma in production. Apart from just locking the pragma version in the code, the sign (>=) needs to be removed. It is possible to lock the pragma by fixing the version both in truffle-config.js for the Truffle framework or in hardhat.config.js for the HardHat framework.

Remediation Plan:

SOLVED : EasyFi team amended all smart contracts to lock the pragma version.

3.3 (HAL-03) USAGE OF ASSERT - INFORMATIONAL

Description:

The **assert** function should only be used to test for internal errors, and to check invariants according to **https://solidity.readthedocs.io/en/latest/control-structures.html#id4**.

Code Location:

SimplePriceOracle.sol Lines# 44-188

Listing 4: SimplePriceOracle.sol

```
64     function getUnderlyingPrice(CToken cToken) public view returns
           (uint256) {
65         //if(!isStockAsset[_asset]){
66         if (compareStrings(cToken.symbol(), "cETH")) {
67             address _aggregator = aggregator[bnbUnderlying];
68             (MathError error, uint256 price) = mulUInt(
69                 uint256(AccessControlledAggregator(_aggregator).
                       latestAnswer()),
70                 10**10
71             );
72             assert(error == MathError.NO_ERROR);
73             return price;
74         } else {
75             address _aggregator = aggregator[
76                 address(CErc20(address(cToken)).underlying())
77             ];
78             uint8 underlyingDecimals = EIP20Interface(
79                 address(CErc20(address(cToken)).underlying())
80             ).decimals();
81             if (underlyingDecimals == 18) {
82                 (MathError error, uint256 price) = mulUInt(
83                     uint256(
84                         AccessControlledAggregator(_aggregator).
                               latestAnswer()
85                     ),
```

```
86                      10**10
87                  );
88                  assert(error == MathError.NO_ERROR);
89                  return price;
90              }
91              if (underlyingDecimals == 6) {
92                  (MathError error, uint256 price) = mulUInt(
93                      uint256(
94                          AccessControlledAggregator(_aggregator).
                               latestAnswer()
95                      ),
96                      10**22
97                  );
98                  assert(error == MathError.NO_ERROR);
99                  return price;
100             }
101             if (underlyingDecimals == 8) {
102                 (MathError error, uint256 price) = mulUInt(
103                     uint256(
104                         AccessControlledAggregator(_aggregator).
                               latestAnswer()
105                     ),
106                     10**20
107                 );
108                 assert(error == MathError.NO_ERROR);
109                 return price;
110             }
111         }
112     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The usage of **assert** should be reviewed and if it is inappropriate, the exception should be changed with **revert**. It is recommended to use **require()** function instead of **assert()** function. This change will improve

code readability. The example code snippet can be seen below.

Listing 5: SimplePriceOracle.sol

```
64     function getUnderlyingPrice(CToken cToken) public view returns
           (uint256) {
65         //if(!isStockAsset[_asset]){
66         if (compareStrings(cToken.symbol(), "cETH")) {
67             address _aggregator = aggregator[bnbUnderlying];
68             (MathError error, uint256 price) = mulUInt(
69                 uint256(AccessControlledAggregator(_aggregator).
                       latestAnswer()),
70                 10**10
71             );
72             require(error == MathError.NO_ERROR, "Error with
                   calculation");
73             return price;
```

Remediation Plan:

SOLVED : assert was replaced with require within the SimplePriceOracle.sol smart contract.

3.4 (HAL-04) LACK OF DUPLICATE AGGREGATOR CHECK - INFORMATIONAL

Description:

During the manual code review, it has been observed that the duplicate aggregators have not been checked on the constructor. Although the function is authorized by a contract deployer, the duplicate aggregator should be checked on the related function.

Code Location:

SimplePriceOracle.sol Lines# 17

Listing 6

```
1     constructor(address[] memory _assets, address[] memory
          _aggregators)
2         public
3     {
4         admin = msg.sender;
5         for (uint256 i = 0; i < _assets.length; i++) {
6             aggregator[_assets[i]] = _aggregators[i];
7         }
8     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider to eliminate duplicate variables on the **aggregators** array.

Remediation Plan:

ACKNOWLEDGED: EasyFi team informed that the data will be passed by admin at time of deployment, where duplication will be taken care.

3.5 (HAL-05) LACK OF ZERO ADDRESS CHECK - INFORMATIONAL

Description:

Some addresses used in constructors and functions are not being validated. Every address should be validated and checked that is different than zero. Aggregators and assets should not be equal to zero.

Code Location:

SimplePriceOracle.sol Lines# 17

Listing 7

```
1     constructor(address[] memory _assets, address[] memory
          _aggregators)
2         public
3     {
4         admin = msg.sender;
5         for (uint256 i = 0; i < _assets.length; i++) {
6             aggregator[_assets[i]] = _aggregators[i];
7         }
8     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to validate that every address input is different than zero.

Remediation Plan:

ACKNOWLEDGED: EasyFi team informed that the data will be passed by admin at time of deployment, where zero address check will be taken care.

3.6 (HAL-06) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function. The public function uses 496 gas, while the external function uses only 261. Also, methods do not necessarily have to be public if they are only called within the contract - in such case they should be marked internal.

Code Location:

Below are smart contract and Its corresponding functions affected:

SimplePriceOracle.sol:
addAsset, addStockAsset, getUnderlyingPrice, getUnderlyingPrice1

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practice, you should use external if you expect that the function will only be called externally and use public if you need to call the function internally. To sum up, everyone can access public functions, external functions can only be accessed externally and

internal functions can only be called within the contract.

Remediation Plan:

SOLVED : EasyFi team replaced public with external in the recommended functions.

3.7 (HAL-07) MISSING EVENTS EMISSION - INFORMATIONAL

Description:

It has been observed that important functionality is missing emitting event for some functions on the SimplePriceOracle.sol contract. These functions should emit events. Events are a method of informing the transaction initiator about the actions taken by the called function. An events logs its emitted parameters in a specific log history, which can be accessed outside of the contract using some filter parameters.

Code Location:

SimplePriceOracle.sol Lines# 34-39

Listing 8: SimplePriceOracle.sol (Lines 34,39)

```
34      function addAsset(address _asset, address _aggregator) public
            onlyAdmin {
35          require(aggregator[_asset] == address(0), "Already added")
                ;
36          aggregator[_asset] = _aggregator;
37      }
38
39      function addStockAsset(address _asset, uint256 _price) public
            onlyAdmin {
40          isStockAsset[_asset] = true;
41          assetPrice[_asset] = _price;
42      }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

For best security practices, consider declaring events as often as possible at the end of a function. Events can be used to detect the end of the operation.

Remediation Plan:

SOLVED : EasyFi team implemented Event emission where recommended.

3.8 (HAL-08) EXPERIMENTAL FEATURES ENABLED - INFORMATIONAL

Description:

The use of experimental features could be dangerous on live deployments. The experimental ABI encoder does not handle non-integer values shorter than 32 bytes properly. This applies to bytesNN types, bool, enum and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside abi.encode(...) as arguments in external function calls or in event data without prior assignment to a local variable. Using return does not trigger the bug. The types bytesNN and bool will result in corrupted data while enum might lead to an invalid revert.

Furthermore, arrays with elements shorter than 32 bytes may not be handled correctly even if the base type is an integer. Encoding such arrays in the way described above can lead to other data in the encoding being overwritten if the number of elements encoded is not a multiple of the number of elements that fit a single slot. If nothing follows the array in the encoding (note that dynamically-sized arrays are always encoded after statically-sized arrays with statically-sized content), or if only a single array is encoded, no other data is overwritten. There are known bugs that are publicly released while using this feature. However, the bug only manifests itself when all the following conditions are met:

1. Storage data involving arrays or structs is sent directly to an external function call, to abi.encode or to event data without prior assignment to a local (memory) variable.
2. There is an array that contains elements with size less than 32 bytes or a struct that has elements that share a storage slot or members of type bytesNN shorter than 32 bytes.

In addition to that, in the following situations, the code is NOT affected:

1. If all the structs or arrays only use uint256 or int256 types.

2. If you only use integer types (that may be shorter) and only encode at most one array at a time.
3. If you only return such data and do not use it in abi.encode, external calls or event data.

ABIEncoderV2 is enabled to allow to pass the struct type into a function in both web3 and another contract. Naturally, any bug can have wildly varying consequences depending on the program control flow, but it is expected expected that this is more likely to lead to malfunction than exploitability. The bug, when triggered, will under certain circumstances send corrupt parameters on method invocations to other contracts.

Reference:

https://blog.ethereum.org/2019/03/26/solidity-optimizer-and-abiencoderv2-bug/

Code Location:

Listing 9

```
1 pragma experimental ABIEncoderV2;
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

When possible, do not use experimental features in the final live deployment. Validate and check that all the conditions above are true for integers and arrays (i.e. all using uint256).

Remediation Plan:

ACKNOWLEDGED: EasiFy team limited the use of experimental features to only the necessary items.

3.9 (HAL-09) UNUSED CODE SECTION - INFORMATIONAL

Description:

In the smart contract SimplePriceOracle.sol, the code of the getUnderlyingPrice1 function is the same as the getUnderlyingPrice function. Therefore, the stated code base should be deleted from the repository. Additionally, a large commented block of code was found.

Code Location:

SimplePriceOracle.sol Lines# 94-142

Listing 10

```
1     function getUnderlyingPrice1(CToken cToken) public view
          returns (uint256, address) {
2         if (compareStrings(cToken.symbol(), "cETH")) {
3             address _aggregator = aggregator[bnbUnderlying];
4             (MathError error, uint256 price) = mulUInt(
5                 uint256(AccessControlledAggregator(_aggregator).
                      latestAnswer()),
6                 10**10
7             );
8             assert(error == MathError.NO_ERROR);
9             return (price, _aggregator);
10        } else {
11            address _aggregator = aggregator[
12                address(CErc20(address(cToken)).underlying())
13            ];
14            uint8 underlyingDecimals = EIP20Interface(
15                address(CErc20(address(cToken)).underlying())
16            ).decimals();
17            return (underlyingDecimals, _aggregator);
18            // if (underlyingDecimals == 18) {
19            //     (MathError error, uint256 price) = mulUInt(
20            //         uint256(
21            //             AccessControlledAggregator(_aggregator)
                  .latestAnswer()
```

```
22              //          ),
23              //          10**10
24              //      );
25              //      assert(error == MathError.NO_ERROR);
26              //      return price;
27              // }
28              // if (underlyingDecimals == 6) {
29              //      (MathError error, uint256 price) = mulUInt(
30              //          uint256(
31              //              AccessControlledAggregator(_aggregator)
                    .latestAnswer()
32              //          ),
33              //          10**22
34              //      );
35              //      assert(error == MathError.NO_ERROR);
36              //      return price;
37              // }
38              // if (underlyingDecimals == 8) {
39              //      (MathError error, uint256 price) = mulUInt(
40              //          uint256(
41              //              AccessControlledAggregator(_aggregator)
                    .latestAnswer()
42              //          ),
43              //          10**20
44              //      );
45              //      assert(error == MathError.NO_ERROR);
46              //      return price;
47              // }
48          }
49      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

As stated on the description, `getUnderlyingPrice1` should be deleted from the contract. This will reduce gas cost and code complexity.

Remediation Plan:

SOLVED: EasyFi team amended the code base to remove duplicates and redundant code.

3.10 (HAL-10) USAGE OF ABI ENCODEPACKED FUNCTION - INFORMATIONAL

Description:

Using **abi.encodePacked()** with multiple variable length arguments can, in certain situations, lead to a hash collision.

Code Location:

SimplePriceOracle.sol Lines# 190-198

Listing 11: SimplePriceOracle.sol

```
190     function compareStrings(string memory a, string memory b)
191         internal
192         pure
193         returns (bool)
194     {
195         return (keccak256(abi.encodePacked((a))) ==
196             keccak256(abi.encodePacked((b))));
197     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

When using **abi.encodePacked()**, it's crucial to ensure that a matching signature cannot be achieved using different parameters. To do so, either do not allow users access to parameters used in **abi.encodePacked()**, or use fixed length arrays. Alternatively, **abi.encode()** can be used instead.

Remediation Plan:

ACKNOWLEDGED: EasyFi team confirmed that testing has been done to ensure that the functionality did not expose any security risk.

3.11 (HAL-11) USE OF INLINE ASSEMBLY - INFORMATIONAL

Description:

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This discards several important safety features in Solidity. Inline assembly is used in the USDC.sol contract.

Governance/USDC.sol Lines# 296-301

Listing 12

```
1     function getChainId() internal pure returns (uint) {
2         uint256 chainId;
3         assembly { chainId := chainid() }
4         return chainId;
5     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

When possible, do not use inline assembly because it is a manner to access to the EVM (Ethereum Virtual Machine) at a low level. An attacker could bypass many important safety features of Solidity.

Remediation Plan:

ACKNOWLEDGED: EasyFi team confirmed that the USDC.sol contract is only a testing contract and that the actual USDC contract (available at https://etherscan.io/token/0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48) will be used in production instead.

AUTOMATED TESTING

4.1 STATIC ANALYSIS RESULTS

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.

Results:

```
zlion@testingvm:~/Desktop/EasyFi/defi-4c24118e5d67f77f6ad5a85b208707e874719be6/contracts/lending$ slither contracts/SimplePriceOracle.sol
INFO:Detectors:
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transfer(address,uint256) (contracts/EIP20NonStandardInterface.sol#34)
EIP20NonStandardInterface (contracts/EIP20NonStandardInterface.sol#8-70) has incorrect ERC20 function interface:EIP20NonStandardInterface.transferFrom(address,address,uint256) (contracts/EIP20NonStandardInterface.sol#48)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-erc20-interface
INFO:Detectors:
CToken.accrueInterest() (contracts/CToken.sol#385-471) uses a dangerous strict equality:
        - accrualBlockNumberPrior == currentBlockNumber (contracts/CToken.sol#391)
CToken.accrueInterest() (contracts/CToken.sol#385-471) uses a dangerous strict equality:
        - require(bool,string)(mathErr == MathError.NO_ERROR,could not calculate block delta) (contracts/CToken.sol#408)
CToken.balanceOfUnderlying(address) (contracts/CToken.sol#190-195) uses a dangerous strict equality:
        - require(bool,string)(mErr == MathError.NO_ERROR,balance could not be calculated) (contracts/CToken.sol#193)
CToken.borrowBalanceStored(address) (contracts/CToken.sol#271-275) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,borrowBalanceStored: borrowBalanceStoredInternal failed) (contracts/CToken.sol#273)
CarefulMath.divUInt(uint256,uint256) (contracts/CarefulMath.sol#41-47) uses a dangerous strict equality:
        - b == 0 (contracts/CarefulMath.sol#42)
CToken.exchangeRateStored() (contracts/CToken.sol#328-332) uses a dangerous strict equality:
        - require(bool,string)(err == MathError.NO_ERROR,exchangeRateStored: exchangeRateStoredInternal failed) (contracts/CToken.sol#330)
CToken.exchangeRateStoredInternal() (contracts/CToken.sol#339-370) uses a dangerous strict equality:
        - _totalSupply == 0 (contracts/CToken.sol#341)
CToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/CToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(accrualBlockNumber == 0 && borrowIndex == 0,market may only be initialized once) (contracts/CToken.sol#32)
CToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/CToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting comptroller failed) (contracts/CToken.sol#40)
CToken.initialize(ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/CToken.sol#25-56) uses a dangerous strict equality:
        - require(bool,string)(err == uint256(Error.NO_ERROR),setting interest rate model failed) (contracts/CToken.sol#48)
CToken.liquidateBorrowFresh(address,address,uint256,CTokenInterface) (contracts/CToken.sol#964-1033) uses a dangerous strict equality:
        - require(bool,string)(amountSeizeError == uint256(Error.NO_ERROR),LIQUIDATE_COMPTROLLER_CALCULATE_AMOUNT_SEIZE_FAILED) (contracts/CToken.sol#1009)
CToken.liquidateBorrowFresh(address,address,uint256,CTokenInterface) (contracts/CToken.sol#964-1033) uses a dangerous strict equality:
        - require(bool,string)(seizeError == uint256(Error.NO_ERROR),token seizure failed) (contracts/CToken.sol#1023)
CToken.mintFresh(address,uint256) (contracts/CToken.sol#506-571) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_EXCHANGE_CALCULATION_FAILED) (contracts/CToken.sol#545)
CToken.mintFresh(address,uint256) (contracts/CToken.sol#506-571) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_TOTAL_SUPPLY_CALCULATION_FAILED) (contracts/CToken.sol#553)
CToken.mintFresh(address,uint256) (contracts/CToken.sol#506-571) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,MINT_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED) (contracts/CToken.sol#556)
Exponential.mulExp(ExponentialNoError.Exp,ExponentialNoError.Exp) (contracts/Exponential.sol#135-155) uses a dangerous strict equality:
        - assert(bool)(err2 == MathError.NO_ERROR) (contracts/Exponential.sol#152)
CarefulMath.mulUInt(uint256,uint256) (contracts/CarefulMath.sol#24-36) uses a dangerous strict equality:
        - a == 0 (contracts/CarefulMath.sol#25)
ExponentialNoError.mul_(uint256,uint256,string) (contracts/ExponentialNoError.sol#150-157) uses a dangerous strict equality:
        - a == 0 || b == 0 (contracts/ExponentialNoError.sol#151)
ExponentialNoError.mul_(uint256,uint256,string) (contracts/ExponentialNoError.sol#150-157) uses a dangerous strict equality:
        - require(bool,string)(c / a == b,errorMessage) (contracts/ExponentialNoError.sol#155)
CToken.repayBorrowFresh(address,address,uint256) (contracts/CToken.sol#861-928) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED) (contracts/CToken.sol#910)
CToken.repayBorrowFresh(address,address,uint256) (contracts/CToken.sol#861-928) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,REPAY_BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED) (contracts/CToken.sol#913)
CToken.seizeInternal(address,address,address,uint256) (contracts/CToken.sol#1071-1134) uses a dangerous strict equality:
        - require(bool,string)(vars.mathErr == MathError.NO_ERROR,exchange rate math error) (contracts/CToken.sol#1099)
CToken.transfer(address,uint256) (contracts/CToken.sol#135-137) uses a dangerous strict equality:
        - transferTokens(msg.sender,msg.sender,dst,amount) == uint256(Error.NO_ERROR) (contracts/CToken.sol#136)
CToken.transferFrom(address,address,uint256) (contracts/CToken.sol#146-148) uses a dangerous strict equality:
        - transferTokens(msg.sender,src,dst,amount) == uint256(Error.NO_ERROR) (contracts/CToken.sol#147)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
```

```
INFO:Detectors:
Reentrancy in CToken.liquidateBorrowInternal(address,uint256,CTokenInterface) (contracts/CToken.sol#938-953):
        External calls:
        - error = cTokenCollateral.accrueInterest() (contracts/CToken.sol#945)
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
                - allowed = comptroller.liquidateBorrowAllowed(address(this),address(cTokenCollateral),liquidator,borrower,repayAmount) (contracts/CToken.sol#966)
                - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (contracts/CToken.sol#1073)
                - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (contracts/CToken.sol#863)
                - seizeError = cTokenCollateral.seize(liquidator,borrower,seizeTokens) (contracts/CToken.sol#1019)
        State variables written after the call(s):
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
                - totalBorrows = vars.totalBorrowsNew (contracts/CToken.sol#918)
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
                - totalInterestReserve = vars.totalInterestReserveNew (contracts/CToken.sol#1118)
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
                - totalReserves = vars.totalReservesNew (contracts/CToken.sol#1117)
Reentrancy in CToken.redeemFresh(address,uint256,uint256) (contracts/CToken.sol#623-717):
        External calls:
        - allowed = comptroller.redeemAllowed(address(this),redeemer,vars.redeemTokens) (contracts/CToken.sol#663)
        State variables written after the call(s):
        - totalSupply = vars.totalSupplyNew (contracts/CToken.sol#706)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
SimplePriceOracle.getUnderlyingPrice(CToken).price_scope_6 (contracts/SimplePriceOracle.sol#82) is a local variable never initialized
CToken.seizeInternal(address,address,address,uint256).vars (contracts/CToken.sol#1083) is a local variable never initialized
CToken.mintFresh(address,uint256).vars (contracts/CToken.sol#518) is a local variable never initialized
SimplePriceOracle.assetPrices(address).error_scope_3 (contracts/SimplePriceOracle.sol#138) is a local variable never initialized
SimplePriceOracle.assetPrices(address).error_scope_5 (contracts/SimplePriceOracle.sol#148) is a local variable never initialized
SimplePriceOracle.assetPrices(address).price_scope_4 (contracts/SimplePriceOracle.sol#138) is a local variable never initialized
SimplePriceOracle.getUnderlyingPrice(CToken).error_scope_1 (contracts/SimplePriceOracle.sol#62) is a local variable never initialized
CToken.borrowFresh(address,uint256).vars (contracts/CToken.sol#763) is a local variable never initialized
SimplePriceOracle.assetPrices(address).price_scope_6 (contracts/SimplePriceOracle.sol#148) is a local variable never initialized
SimplePriceOracle.getUnderlyingPrice(CToken).price_scope_2 (contracts/SimplePriceOracle.sol#62) is a local variable never initialized
CToken._addInterestReserveFresh(uint256).actualAddAmount (contracts/CToken.sol#1456) is a local variable never initialized
SimplePriceOracle.getUnderlyingPrice(CToken).error_scope_5 (contracts/SimplePriceOracle.sol#82) is a local variable never initialized
CToken._addReservesFresh(uint256).actualAddAmount (contracts/CToken.sol#1329) is a local variable never initialized
SimplePriceOracle.getUnderlyingPrice(CToken).error_scope_3 (contracts/SimplePriceOracle.sol#72) is a local variable never initialized
CToken.redeemFresh(address,uint256,uint256).vars (contracts/CToken.sol#626) is a local variable never initialized
SimplePriceOracle.getUnderlyingPrice(CToken).price_scope_4 (contracts/SimplePriceOracle.sol#72) is a local variable never initialized
SimplePriceOracle.assetPrices(address).error_scope_1 (contracts/SimplePriceOracle.sol#128) is a local variable never initialized
CToken.repayBorrowFresh(address,address,uint256).vars (contracts/CToken.sol#873) is a local variable never initialized
SimplePriceOracle.assetPrices(address).price_scope_2 (contracts/SimplePriceOracle.sol#128) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
CErc20.initialize(address,ComptrollerInterface,InterestRateModel,uint256,string,string,uint8) (contracts/CErc20.sol#25-38) ignores return value by EIP20Interface(underlying).totalSupply() (contracts/CErc20.sol#37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Exponential.divScalarByExpTruncate(uint256,ExponentialNoError.Exp).fraction (contracts/Exponential.sol#124) shadows:
        - ExponentialNoError.fraction(uint256,uint256) (contracts/ExponentialNoError.sol#192-194) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

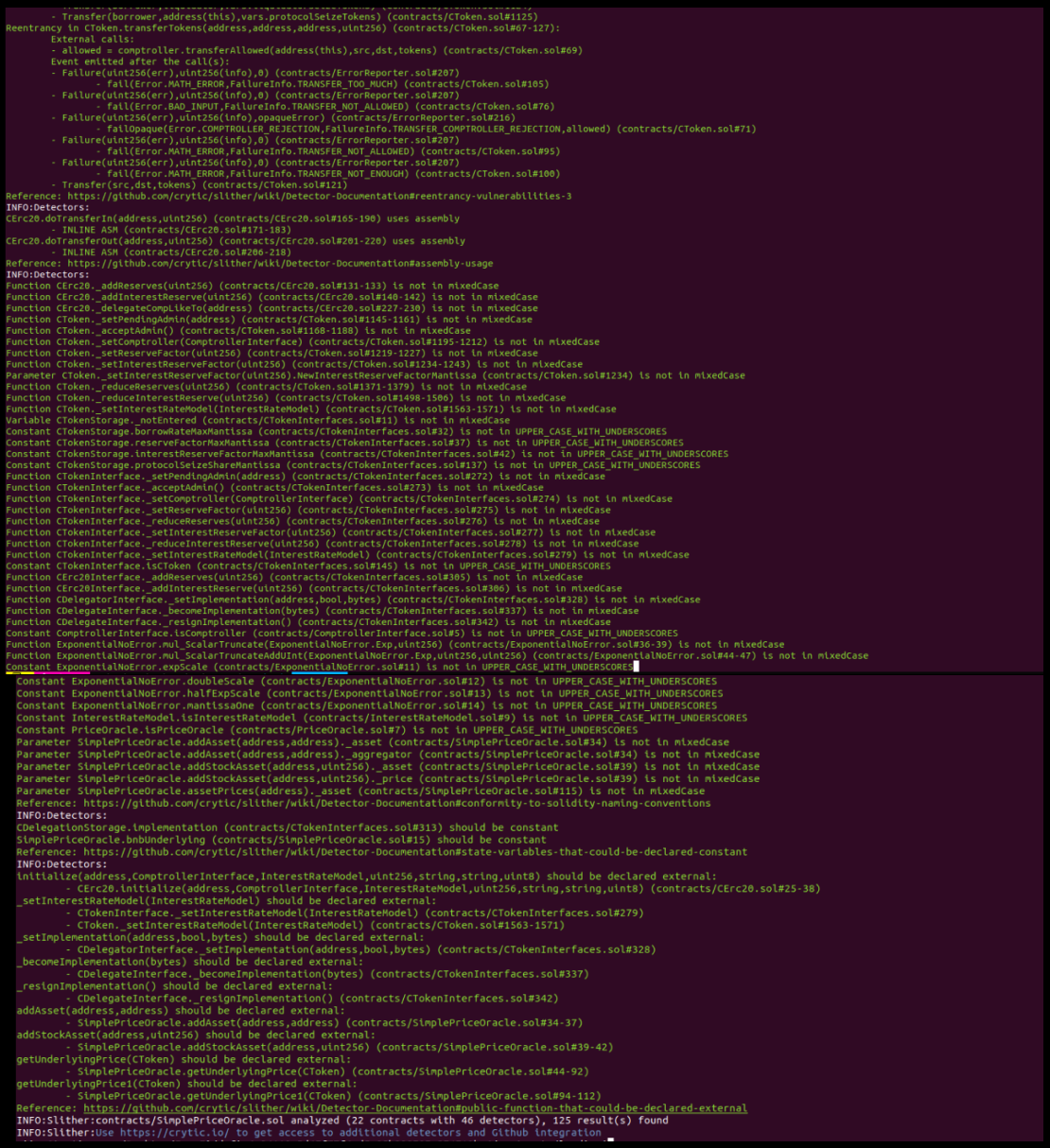
AUTOMATED TESTING

```
INFO:Detectors:
Reentrancy in CToken.borrowFresh(address,uint256) (contracts/CToken.sol#746-810):
        External calls:
        - allowed = comptroller.borrowAllowed(address(this),borrower,borrowAmount) (contracts/CToken.sol#748)
        State variables written after the call(s):
        - accountBorrows[borrower].principal = vars.accountBorrowsNew (contracts/CToken.sol#798)
        - accountBorrows[borrower].interestIndex = borrowIndex (contracts/CToken.sol#799)
        - totalBorrows = vars.totalBorrowsNew (contracts/CToken.sol#800)
Reentrancy in CToken.mintFresh(address,uint256) (contracts/CToken.sol#506-571):
        External calls:
        - allowed = comptroller.mintAllowed(address(this),minter,mintAmount) (contracts/CToken.sol#508)
        State variables written after the call(s):
        - accountTokens[minter] = vars.accountTokensNew (contracts/CToken.sol#560)
        - totalSupply = vars.totalSupplyNew (contracts/CToken.sol#559)
Reentrancy in CToken.redeemFresh(address,uint256,uint256) (contracts/CToken.sol#623-717):
        External calls:
        - allowed = comptroller.redeemAllowed(address(this),redeemer,vars.redeemTokens) (contracts/CToken.sol#663)
        State variables written after the call(s):
        - accountTokens[redeemer] = vars.accountTokensNew (contracts/CToken.sol#707)
Reentrancy in CToken.repayBorrowFresh(address,address,uint256) (contracts/CToken.sol#861-928):
        External calls:
        - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (contracts/CToken.sol#863)
        State variables written after the call(s):
        - accountBorrows[borrower].principal = vars.accountBorrowsNew (contracts/CToken.sol#916)
        - accountBorrows[borrower].interestIndex = borrowIndex (contracts/CToken.sol#917)
        - totalBorrows = vars.totalBorrowsNew (contracts/CToken.sol#918)
Reentrancy in CToken.seizeInternal(address,address,address,uint256) (contracts/CToken.sol#1071-1134):
        External calls:
        - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (contracts/CToken.sol#1073)
        State variables written after the call(s):
        - accountTokens[borrower] = vars.borrowerTokensNew (contracts/CToken.sol#1120)
        - accountTokens[liquidator] = vars.liquidatorTokensNew (contracts/CToken.sol#1121)
        - totalInterestReserve = vars.totalInterestReserveNew (contracts/CToken.sol#1118)
        - totalReserves = vars.totalReservesNew (contracts/CToken.sol#1117)
        - totalSupply = vars.totalSupplyNew (contracts/CToken.sol#1119)
Reentrancy in CToken.transferTokens(address,address,address,uint256) (contracts/CToken.sol#67-127):
        External calls:
        - allowed = comptroller.transferAllowed(address(this),src,dst,tokens) (contracts/CToken.sol#69)
        State variables written after the call(s):
        - accountTokens[src] = srcTokensNew (contracts/CToken.sol#112)
        - accountTokens[dst] = dstTokensNew (contracts/CToken.sol#113)
        - transferAllowances[src][spender] = allowanceNew (contracts/CToken.sol#117)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
INFO:Detectors:
Reentrancy in CToken.borrowFresh(address,uint256) (contracts/CToken.sol#746-810):
        External calls:
        - allowed = comptroller.borrowAllowed(address(this),borrower,borrowAmount) (contracts/CToken.sol#748)
        Event emitted after the call(s):
        - Borrow(borrower,borrowAmount,vars.accountBorrowsNew,vars.totalBorrowsNew) (contracts/CToken.sol#803)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.MATH_ERROR,FailureInfo.BORROW_NEW_TOTAL_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)) (contracts/CToken.sol#782)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.MATH_ERROR,FailureInfo.BORROW_NEW_ACCOUNT_BORROW_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)) (contracts/CToken.sol#777)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.BORROW_COMPTROLLER_REJECTION,allowed) (contracts/CToken.sol#750)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.MATH_ERROR,FailureInfo.BORROW_ACCUMULATED_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)) (contracts/CToken.sol#772)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.MARKET_NOT_FRESH,FailureInfo.BORROW_FRESHNESS_CHECK) (contracts/CToken.sol#755)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.TOKEN_INSUFFICIENT_CASH,FailureInfo.BORROW_CASH_NOT_AVAILABLE) (contracts/CToken.sol#760)
Reentrancy in CToken.liquidateBorrowFresh(address,address,uint256,CTokenInterface) (contracts/CToken.sol#964-1033):
        External calls:
        - allowed = comptroller.liquidateBorrowAllowed(address(this),address(cTokenCollateral),liquidator,borrower,repayAmount) (contracts/CToken.sol#966)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.INVALID_CLOSE_AMOUNT_REQUESTED,FailureInfo.LIQUIDATE_CLOSE_AMOUNT_IS_ZERO),0) (contracts/CToken.sol#988)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.INVALID_CLOSE_AMOUNT_REQUESTED,FailureInfo.LIQUIDATE_CLOSE_AMOUNT_IS_UINT_MAX),0) (contracts/CToken.sol#993)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.MARKET_NOT_FRESH,FailureInfo.LIQUIDATE_COLLATERAL_FRESHNESS_CHECK),0) (contracts/CToken.sol#978)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.LIQUIDATE_COMPTROLLER_REJECTION,allowed),0) (contracts/CToken.sol#968)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.MARKET_NOT_FRESH,FailureInfo.LIQUIDATE_FRESHNESS_CHECK),0) (contracts/CToken.sol#973)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.INVALID_ACCOUNT_PAIR,FailureInfo.LIQUIDATE_LIQUIDATOR_IS_BORROWER),0) (contracts/CToken.sol#983)
Reentrancy in CToken.liquidateBorrowFresh(address,address,uint256,CTokenInterface) (contracts/CToken.sol#964-1033):
        External calls:
        - allowed = comptroller.liquidateBorrowAllowed(address(this),address(cTokenCollateral),liquidator,borrower,repayAmount) (contracts/CToken.sol#966)
        - (repayBorrowError,actualRepayAmount) = repayBorrowFresh(liquidator,borrower,repayAmount) (contracts/CToken.sol#998)
                - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (contracts/CToken.sol#863)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - (repayBorrowError,actualRepayAmount) = repayBorrowFresh(liquidator,borrower,repayAmount) (contracts/CToken.sol#998)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - (repayBorrowError,actualRepayAmount) = repayBorrowFresh(liquidator,borrower,repayAmount) (contracts/CToken.sol#998)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(TokenErrorReporter.Error(repayBorrowError),FailureInfo.LIQUIDATE_REPAY_BORROW_FRESH_FAILED),0) (contracts/CToken.sol#1000)
        - RepayBorrow(payer,borrower,vars.actualRepayAmount,vars.accountBorrowsNew,vars.totalBorrowsNew) (contracts/CToken.sol#921)
                - (repayBorrowError,actualRepayAmount) = repayBorrowFresh(liquidator,borrower,repayAmount) (contracts/CToken.sol#998)
```

AUTOMATED TESTING

```
Reentrancy in CToken.liquidateBorrowFresh(address,address,uint256,CTokenInterface) (contracts/CToken.sol#964-1033):
        External calls:
        - allowed = comptroller.liquidateBorrowAllowed(address(this),address(cTokenCollateral),liquidator,borrower,repayAmount) (contracts/CToken.sol#966)
        - (repayBorrowError,actualRepayAmount) = repayBorrowFresh(liquidator,borrower,repayAmount) (contracts/CToken.sol#998)
                - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (contracts/CToken.sol#863)
        - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (contracts/CToken.sol#1017)
                - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (contracts/CToken.sol#1073)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (contracts/CToken.sol#1017)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (contracts/CToken.sol#1017)
        - InterestReserveAdded(address(this),vars.protocolSeizeAmount,vars.totalInterestReserveNew) (contracts/CToken.sol#1127)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (contracts/CToken.sol#1017)
        - ReservesAdded(address(this),vars.protocolSeizeAmount,vars.totalReservesNew) (contracts/CToken.sol#1126)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (contracts/CToken.sol#1017)
        - Transfer(borrower,liquidator,vars.liquidatorSeizeTokens) (contracts/CToken.sol#1124)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (contracts/CToken.sol#1017)
        - Transfer(borrower,address(this),vars.protocolSeizeTokens) (contracts/CToken.sol#1125)
                - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (contracts/CToken.sol#1017)
Reentrancy in CToken.liquidateBorrowFresh(address,address,uint256,CTokenInterface) (contracts/CToken.sol#964-1033):
        External calls:
        - allowed = comptroller.liquidateBorrowAllowed(address(this),address(cTokenCollateral),liquidator,borrower,repayAmount) (contracts/CToken.sol#966)
        - (repayBorrowError,actualRepayAmount) = repayBorrowFresh(liquidator,borrower,repayAmount) (contracts/CToken.sol#998)
                - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (contracts/CToken.sol#863)
        - seizeError = seizeInternal(address(this),liquidator,borrower,seizeTokens) (contracts/CToken.sol#1017)
                - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (contracts/CToken.sol#1073)
        - seizeError = cTokenCollateral.seize(liquidator,borrower,seizeTokens) (contracts/CToken.sol#1019)
        Event emitted after the call(s):
        - LiquidateBorrow(liquidator,borrower,actualRepayAmount,address(cTokenCollateral),seizeTokens) (contracts/CToken.sol#1026)
Reentrancy in CToken.liquidateBorrowInternal(address,uint256,CTokenInterface) (contracts/CToken.sol#938-953):
        External calls:
        - error = cTokenCollateral.accrueInterest() (contracts/CToken.sol#945)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - (fail(TokenErrorReporter.Error(error),FailureInfo.LIQUIDATE_ACCRUE_COLLATERAL_INTEREST_FAILED),0) (contracts/CToken.sol#948)
Reentrancy in CToken.liquidateBorrowInternal(address,uint256,CTokenInterface) (contracts/CToken.sol#938-953):
        External calls:
        - error = cTokenCollateral.accrueInterest() (contracts/CToken.sol#945)
        - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
                - allowed = comptroller.liquidateBorrowAllowed(address(this),address(cTokenCollateral),liquidator,borrower,repayAmount) (contracts/CToken.sol#966)
                - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (contracts/CToken.sol#1073)
                - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (contracts/CToken.sol#863)
                - seizeError = cTokenCollateral.seize(liquidator,borrower,seizeTokens) (contracts/CToken.sol#1019)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
        - InterestReserveAdded(address(this),vars.protocolSeizeAmount,vars.totalInterestReserveNew) (contracts/CToken.sol#1127)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
        - LiquidateBorrow(liquidator,borrower,actualRepayAmount,address(cTokenCollateral),seizeTokens) (contracts/CToken.sol#1026)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
        - RepayBorrow(payer,borrower,vars.actualRepayAmount,vars.accountBorrowsNew,vars.totalBorrowsNew) (contracts/CToken.sol#921)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
        - ReservesAdded(address(this),vars.protocolSeizeAmount,vars.totalReservesNew) (contracts/CToken.sol#1126)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
        - Transfer(borrower,liquidator,vars.liquidatorSeizeTokens) (contracts/CToken.sol#1124)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
        - Transfer(borrower,address(this),vars.protocolSeizeTokens) (contracts/CToken.sol#1125)
                - liquidateBorrowFresh(msg.sender,borrower,repayAmount,cTokenCollateral) (contracts/CToken.sol#952)
Reentrancy in CToken.mintFresh(address,uint256) (contracts/CToken.sol#506-571):
        External calls:
        - allowed = comptroller.mintAllowed(address(this),minter,mintAmount) (contracts/CToken.sol#508)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - (failOpaque(Error.MATH_ERROR,FailureInfo.MINT_EXCHANGE_RATE_READ_FAILED,uint256(vars.mathErr)),0) (contracts/CToken.sol#522)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - (failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.MINT_COMPTROLLER_REJECTION,allowed),0) (contracts/CToken.sol#510)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - (fail(Error.MARKET_NOT_FRESH,FailureInfo.MINT_FRESHNESS_CHECK),0) (contracts/CToken.sol#515)
        - Mint(minter,vars.actualMintAmount,vars.mintTokens) (contracts/CToken.sol#563)
        - Transfer(address(this),minter,vars.mintTokens) (contracts/CToken.sol#564)
Reentrancy in CToken.redeemFresh(address,uint256,uint256) (contracts/CToken.sol#623-717):
        External calls:
        - allowed = comptroller.redeemAllowed(address(this),redeemer,vars.redeemTokens) (contracts/CToken.sol#663)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.MATH_ERROR,FailureInfo.REDEEM_NEW_TOTAL_SUPPLY_CALCULATION_FAILED,uint256(vars.mathErr)) (contracts/CToken.sol#680)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.REDEEM_COMPTROLLER_REJECTION,allowed) (contracts/CToken.sol#665)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.TOKEN_INSUFFICIENT_CASH,FailureInfo.REDEEM_TRANSFER_OUT_NOT_POSSIBLE) (contracts/CToken.sol#690)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.MATH_ERROR,FailureInfo.REDEEM_NEW_ACCOUNT_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)) (contracts/CToken.sol#685)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.MARKET_NOT_FRESH,FailureInfo.REDEEM_FRESHNESS_CHECK) (contracts/CToken.sol#670)
        - Redeem(redeemer,vars.redeemAmount,vars.redeemTokens) (contracts/CToken.sol#711)
        - Transfer(redeemer,address(this),vars.redeemTokens) (contracts/CToken.sol#710)
Reentrancy in CToken.repayBorrowFresh(address,address,uint256) (contracts/CToken.sol#861-928):
        External calls:
        - allowed = comptroller.repayBorrowAllowed(address(this),payer,borrower,repayAmount) (contracts/CToken.sol#863)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - (failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.REPAY_BORROW_COMPTROLLER_REJECTION,allowed),0) (contracts/CToken.sol#865)
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - (fail(Error.MARKET_NOT_FRESH,FailureInfo.REPAY_BORROW_FRESHNESS_CHECK),0) (contracts/CToken.sol#870)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - (failOpaque(Error.MATH_ERROR,FailureInfo.REPAY_BORROW_ACCUMULATED_BALANCE_CALCULATION_FAILED,uint256(vars.mathErr)),0) (contracts/CToken.sol#881)
        - RepayBorrow(payer,borrower,vars.actualRepayAmount,vars.accountBorrowsNew,vars.totalBorrowsNew) (contracts/CToken.sol#921)
Reentrancy in CToken.seizeInternal(address,address,address,uint256) (contracts/CToken.sol#1071-1134):
        External calls:
        - allowed = comptroller.seizeAllowed(address(this),seizerToken,liquidator,borrower,seizeTokens) (contracts/CToken.sol#1073)
        Event emitted after the call(s):
        - Failure(uint256(err),uint256(info),0) (contracts/ErrorReporter.sol#207)
                - fail(Error.INVALID_ACCOUNT_PAIR,FailureInfo.LIQUIDATE_SEIZE_LIQUIDATOR_IS_BORROWER) (contracts/CToken.sol#1080)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.COMPTROLLER_REJECTION,FailureInfo.LIQUIDATE_SEIZE_COMPTROLLER_REJECTION,allowed) (contracts/CToken.sol#1075)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.MATH_ERROR,FailureInfo.LIQUIDATE_SEIZE_BALANCE_DECREMENT_FAILED,uint256(vars.mathErr)) (contracts/CToken.sol#1092)
        - Failure(uint256(err),uint256(info),opaqueError) (contracts/ErrorReporter.sol#216)
                - failOpaque(Error.MATH_ERROR,FailureInfo.LIQUIDATE_SEIZE_BALANCE_INCREMENT_FAILED,uint256(vars.mathErr)) (contracts/CToken.sol#1109)
        - InterestReserveAdded(address(this),vars.protocolSeizeAmount,vars.totalInterestReserveNew) (contracts/CToken.sol#1127)
        - ReservesAdded(address(this),vars.protocolSeizeAmount,vars.totalReservesNew) (contracts/CToken.sol#1126)
        - Transfer(borrower,liquidator,vars.liquidatorSeizeTokens) (contracts/CToken.sol#1124)
        - Transfer(borrower,address(this),vars.protocolSeizeTokens) (contracts/CToken.sol#1125)
```



According to the static analysis results, some of the findings found by Slither were considered as false positives while some of these findings were not real security concerns.

THANK YOU FOR CHOOSING

// HALBORN