Based Loans
Findings & Analysis Report

2021-05-27

Overview

ABOUT C4

Code 432n4 (C4) is an open organization that consists of security researchers, auditors, developers, and individuals with domain expertise in the area of smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of Based Loans' smart contract system written in Solidity. The code contest took place between April 2 and April 7, 2021.

WARDENS

6 Wardens contributed reports to the Based Loans code contest:

- cmichel
- gpersoon
- OxRajeev

- Thunder
- shw
- toastedsteaksandwich

This contest was judged by Cem.

Final report assembled by ninek and sockdrawermoney.

Summary

The C4 analysis yielded an aggregated total of 31 unique vulnerabilities. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity, 1 received a risk rating in the category of MEDIUM severity, and 13 received a risk rating in the category of LOW severity.

C4 analysis also identified 15 non-critical recommendations.

Scope

The code under review can be found within the C4 code contest repository and comprises 31 smart contracts written in the Solidity programming language.

Severity Criteria

C4 assesses severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into 3 primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings

[H-01] UNISWAPCONFIG GETTERS RETURN WRONG TOKEN CONFIG IF TOKEN CONFIG DOES NOT EXIST

The `UniswapConfig.getTokenConfigBySymbolHash` function does not work as `getSymbolHashIndex` returns `0` if there is no config token for that symbol (uninitialized map value), but the outer function implements the non-existence check with `-1`.

The same issue occurs also for:

- `getTokenConfigByCToken`
- `getTokenConfigByUnderlying`

When encountering a non-existent token config, it will always return the token config of the first index (index 0) which is a valid token config for a completely different token.

This leads to wrong oracle prices for the actual token which could in the worst case be used to borrow more tokens at a lower price or borrow more tokens by having a higher collateral value, essentially allowing undercollateralized loans that cannot be liquidated.

Recommend fixing the non-existence check.

ghoul-sol (Based Loans) confirmed:

> Addressed in this PR

[H-02] UINT(-1) INDEX FOR NOT FOUND

Functions `getTokenConfigBySymbolHash`, `getTokenConfigByCToken` and `getTokenConfigByUnderlying` check returned index against max uint: index != uint(-1)

-1 should indicate that the index is not found, however, a default value for an uninitialized uint is 0, so it is impossible to get -1. What is even weirder is that 0 will be returned for non-existing configs but 0 is a valid index for the 1st config.

One of the solutions would be to reserve 0 for a not found index and use it when searching in mappings. Then normal indexes should start from 1. Another solution would be to introduce a new mapping with a boolean value that indicates if this index is initialized or not but this may be a more gas costly way.

ghoul-sol (Based Loans) confirmed:

> `UniswapConfig` has been refactored. Index 0 is considered a non-existent config and all comparison are against that value.

Medium Risk Findings

[M-01] REWARD RATES CAN BE CHANGED THROUGH FLASH BORROWS

The rewards per market are proportional to their `totalBorrows` which can be changed by a large holder who deposits lots of collateral, takes out a huge borrow in the market, updates the rewards, and then unwinds the position. They'll only pay gas fees as the borrow / repay can happen in the same block.

The `Comptroller.refreshCompSpeeds` function only checks that the single transaction is called from an EOA, but miners (or anyone if a miner offers services like flash bundles for

flashbots) can still run flash-loan-like attacks by first sending a borrow tx increasing the totalBorrows, then the `refreshCompSpeeds` transaction, and then the repay of the borrow, as miners have full control over the transaction order of the block.

The new rate will then persist until the next call to `refreshCompSpeeds`.

Attackers have an incentive to drive up the rewards in markets they are a large supplier/borrower in.

The increased rewards that the attacker receives are essentially stolen from other legitimate users.

Recommend making it an admin-only function or use a time-weighted total borrow system similar to Uniswap's price oracles.

ghoul-sol (Based Loans) confirmed:

> Restricting `Comptroller.refreshCompSpeeds` function to admin only would centralize an ability to update speeds. A better solution may be a bot that keeps track of markets utilizations and updates speeds when needed. That will also give a way to community to participate.
>
> Also, higher rewards would mean that all participants are getting them and that would bring even more liquidity to the given market and decrease attackers earnings. Attacker could keep moving the liquidity from market to market but everyone would follow quite quickly. If that actually happens, admin has a way to stop the rewards and make `refreshCompSpeeds` admin-only function as last resolution because comptroller is using proxy pattern.

Low Risk Findings

[G-01] REQUIRENOERROR CAN BE OPTIMIZED

The function `requireNoError` of Cether.sol contains 2 checks on *errCode == uint(Error.NO*ERROR)_.

After the first check it returns. After this errCode == uint(Error.NO_ERROR) will never be true, so doesn't have to be checked.

Recommend replacing *require(errCode == uint(Error.NO*ERROR), string(fullMessage));_ with *require(false, string(fullMessage));*

Note: Solidity 8.4 has new error handling functionality which could replace the logic of `requireNoError`

ghoul-sol (Based Loans) confirmed:

> It's added to our backlog. Thanks!

[L-01] NO ACCOUNT EXISTENCE CHECK FOR LOW-LEVEL CALL IN CETHER.SOL

Low-level calls call/delegatecall/staticcall return true even if the account called is non-existent (per EVM design). Account existence must be checked prior to calling.

The `doTransferOut()` function was changed from using a `transfer()` function (which reverts) to a `call()` function (which returns a boolean), however there is no account existence check for the destination address to. If it doesn't exist, for some reason, `call` will still return true (not throw an exception) and successfully pass the return value check on the next line.

The checked call paths don't seem vulnerable because they use `msg.sender/admin` and not a user-controlled address, but this may be a risk if used later in other contexts. Hence rating as low-risk.

For reference, see this related high-risk severity finding from Trail of Bit's audit of Hermez Network.

Recommend checking for account-existence before the `call()` to make this safely extendable to user-controlled address contexts in future.

ghoul-sol (Based Loans) confirmed:

> Recommended fix has been implemented.

[L-02] SWEEPTOKEN() FUNCTION REMOVED IN CERC20.SOL FROM ORIGINAL COMPOUND CODE

The `sweepToken()` function in the original Compound code whose specified purpose was to recover accidentally sent ERC20 tokens to contract has been removed.

The original code comment says: “A public function to sweep accidental ERC-20 transfers to this contract. Tokens are sent to admin (timelock).” This safety measure is helpful given the number/value of accidentally stuck tokens that are sent to contracts by mistake.

Tokens accidentally sent to this contract will be stuck leading to fund loss for sender.

Recommend retaining this function unless there is a specific reason to remove it here.

Comments:

ghoul-sol (Based Loans) confirmed:

> Fixed as recommended, thanks!

[L-03] ALL EXCEPT ONE COMPTROLLER VERIFY FUNCTIONS DO NOT VERIFY ANYTHING IN COMPTROLLER.SOL/CTOKEN.SOL

Six of the seven Comptroller verify functions do nothing. Not sure why their calls in CToken.sol have been uncommented from the original Compound version.

Except `redeemVerify()`, six other verify functions `transferVerify()`, `mintVerify()`, `borrowVerify()`, `repayBorrowVerify()`, `liquidateBorrowVerify()` and `seizeVerify()` have no logic except accessing state variables to not be marked pure. Calls to these functions were commented out in the original Compound code’s CToken.sol but have been uncommented here.

Given that they do not implement any logic, the protocol should not be making any assumptions about any defence provided from their unimplemented verification logic.

There are a number of dummy functions whose comments say “// Shh - currently unused”.

Recommend adding logic to implement verification if that is indeed assumed to be implemented but is actually not. Otherwise, comment call sites.

ghoul-sol (Based Loans) confirmed:

> Fixed by commenting unused functions.

[L-04] FLOATING PRAGMA USED IN UNISWAP*.SOL

Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the floating pragma, i.e. by not using ^ in pragma solidity ^0.6.10, ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs.

For reference, see https://swcregistry.io/docs/SWC-103

Recommend removing ^ in “pragma solidity ^0.6.10” and change it to “pragma solidity 0.6.12” to be consistent with the rest of the contracts.

ghoul-sol (Based Loans) confirmed:

> Fixed as recommended.

[L-05] MISSING INPUT VALIDATION MAY SET COMP TOKEN TO ZERO-ADDRESS IN COMPTROLLER.SOL

Function `_setCompAddress()` is used by admin to change the COMP token address. However, there is no zero-address validation on the parameter. This may accidentally set COMP token address to zero-address but it can be reset by the admin. Any interim transactions might hit exceptional behavior.

Recommend adding zero-address check to `_comp` parameter of `_setCompAddress()`.

ghoul-sol (Based Loans) commented:

> Added to our backlog for future refactoring, thanks!

[L-06] MISSING ZERO/THRESHOLD CHECK FOR MAXASSETS

A zero or some minimum threshold check is missing for `newMaxAssets` parameter of `_setMaxAssets()` function which is used by admin to set the maximum number of assets that controls how many markets can be entered.

If accidentally set to 0 then all users cannot enter any market which will significantly affect protocol operations.

Recommend adding zero/threshold check to `newMaxAssets` parameter.

ghoul-sol (Based Loans) commented:

> Added to backlog, however, it's a non-critical issue.

cemozerr (Judge) commented:

> Rating this as low risk as it could pose a situation where users can not enter any markets.

[L-07] USAGE OF ADDRESS.TRANSFER

The `transfer` function is used in `Maximillion.sol` to send ETH to an account.

It is performed with a fixed amount of GAS and might fail if GAS costs change in the future or if a smart contract's fallback function handler is complex.

Consider using the lower-level `.call{value: value}` instead and checking its success return value.

ghoul-sol (Based Loans) confirmed:

> `Maximillion.sol` is not being used and will be deleted.

[L-08] UNBOUNDED ITERATION ON REFRESHCOMPSPEEDSINTERNAL

The `Comptroller.refreshCompSpeedsInternal` function iterates over all markets and does expensive computations like updating all borrower / supply indices.

When the total number of markets is high, this iteration could exceed the total block gas amount breaking the functionality and making it impossible to update the reward distribution speed.

Keep the number of markets low and/or adjust the function to be processable in several transactions.

ghoul-sol (Based Loans) commented:

> While true, estimated gas to update speed for 50 markets is `3377184` gas. Current block gas limit is `14,999,986`, that means we could in theory, get away with updating as many as 222 markets. This is definitely something to keep in mind along the way, however, in my opinion it's a non-critical issue, low at most.

cemozerr (Judge) commented:

> I will rate this as low risk, as it won't be an issue until there are many markets, and does not pose a major risk to user funds.

[L-09] UINT[] MEMORY PARAMETER IS TRICKY

Using memory array parameters (e.g. uint[] memory) as function parameters can be tricky in Solidity, because an attack is possible with a very large array which will overlap with other parts of the memory. See proof of concept below.

The function `propose` of GovernorAlpha.sol seems most vulnerable because this function does not check the validity of the array lengths.

Most other functions do a loop over the array, which will fail with a large array (due to out of gas).

The following functions use a [] memory parameter:

- .\Comptroller.sol: `enterMarkets`, `claimComp`, `claimComp`, `_addCompMarkets`
- .\Governance\GovernorAlpha.sol: `propose`
- .\UniswapOracle\UniswapAnchoredView.sol: `addTokens`
- .\UniswapOracle\UniswapConfig.sol: `_addTokensInternal`

This an example to show the exploit:

```
// based on https://github.com/paradigm-operations/paradigm-ctf-2021/blob/master/swap
pragma solidity ^0.4.24; // only works with low solidity version

contract test{
    struct Overlap {
        uint field0;
    }
    event log(uint);

  function mint(uint[] memory amounts) public  returns (uint) {   // this can be in ar
        Overlap memory v;
        v.field0 = 1234;
        emit log(amounts[0]); // would expect to be 0 however is 1234
        return 1;
    }

  function go() public { // this part requires the low solidity version
      uint x=0x8000000000000000000000000000000000000000000000000000000000000000; // 2^2
      bytes memory payload = abi.encodeWithSelector(this.mint.selector, 0x20, x);
      bool success=address(this).call(payload);
  }
}
```

Recommend adding checks on the size of the array parameters to make sure they are not absurdly long.

ghoul-sol (Based Loans) confirmed:

> As mentioned, this applies to either admin functions or ones that are using a loop. The `propose` function is used by governance so its outcome will be tested on forked network before voting. I don't see an immediate threat from this solidity bug but we'll keep it in mind.

[L-10] CAREFULMATH / SAFE MATH NOT ALLWAYS USED

CarefulMath is used in most calculations, however it isn't always used.

Recommend double checking to see if safe math functions really are not necessary and otherwise add a comment.

ghoul-sol (Based Loans) commented:

> Agreed, however, in my opinion it's a non-critical issue.

cemozerr (Judge) commented:

> I will rate this as low risk instead of non-critical because although the warden here might not have spotted any real issues, unless CarefulMath / SafeMath is not always used, there might be hidden underflow/overflow bugs.

[L-11] USE 'RECEIVE' WHEN EXPECTING ETH AND EMPTY CALL DATA

Contract CEther fallback function was refactored to be compatible with the Solidity 0.6 version:

```
/**
* @notice Send Ether to CEther to mint
*/
fallback () external payable {
    (uint err,) = mintInternal(msg.value);
    requireNoError(err, "mint failed");
}
```

From Solidity 0.6 documentation:

> "The unnamed function commonly referred to as "fallback function" was split up into a new fallback function that is defined using the fallback keyword and a receive ether function defined using the receive keyword. If present, the receive ether function is called whenever the call data is empty (whether or not ether is received). This function is implicitly payable. The new fallback function is called when no other function matches (if the receive ether function does not exist then this includes calls with empty call data). You can make this function payable or not. If it is not payable then transactions not matching any other function which send value will revert. You should only need to implement the new fallback function if you are following an upgrade or proxy pattern."

In this case, "receive" may be more suitable as the function is expecting to receive ether and empty call data.

Recommend replacing "fallback" with "receive".

ghoul-sol (Based Loans) confirmed:

> Fixed as recommended

[L-12] ALLOW BORROWCAP TO BE FILLED FULLY

Here the condition should be '<=', not '<' to allow filling the cap fully:

```
require(nextTotalBorrows < borrowCap, "market borrow cap reached");
```

```
require(nextTotalBorrows <= borrowCap, "market borrow cap reached");
```

ghoul-sol (Based Loans) commented:

> Added to backlog.

Non-Critical Findings

[N-01] OUTDATED COMPILER

The project is using Solidity compiler version 0.6.12 which was released in July 2020, while the latest compiler version is 0.8.4. Using such an older version makes the project susceptible to any compiler bugs fixed or dangerous features deprecated since then, and also prevents it from leveraging the newly introduced features.

It may be recognized that this is harder for this project because it is making modifications to an existing older project (Compound) which uses compiler version 0.5.x.

Given Solidity's fast release cycle, consider using a more recent version of the compiler, such as version 0.7.6.

Given that the project is already going from original Compound's 0.5.x to 0.6.x, it may as well go to 0.7.x version. This may involve a few more breaking changes for changing from 0.6.x to 0.7.x,

but there don't seem to be that many language-level breaking features (see https://github.com/ethereum/solidity/releases/tag/v0.7.0)

Comments:

ghoul-sol (Based Loans) commented:

> Using latest solidity version is best practice. However, upgrading to 0.7.x or 0.8.x requires significant refactoring and any braking changes in solidity could potentially introduce bugs. Also, upgrading at this stage of the project would delay launch further and may require another audit.

cemozerr (Judge) commented:

> I'm changing the severity of the issue to non-significant as Based Loans is a fork of Compound codebase, and there are no compiler-related bugs in Compound codebase AFAIK.

[N-02] MISSED NATSPEC @PARAM FOR NEWLY INTRODUCED PARAMETER DISTRIBUTEALL

The `distributeSupplierComp()` function has been modified to take in a third parameter which is a boolean `distributeAll`. But the corresponding NatSpec comments for the function have not been updated to add this new parameter. This could lead to minor confusion where NatSpec is consulted.

Recommend adding @param for `distributeAll` parameter.

ghoul-sol (Based Loans) confirmed:

> Fixed as recommended.

[N-02] PRIVILEGED ROLES

Admins can change the `comp=blo` address using `_setCompAddress` and stop pending payouts using `_dropCompMarket`.

The allotted rewards of the users may not be paid out anymore due to admins changing the reward token (`comp`) address.

Privileged admin roles make the protocol less predictable for users leading to hesitance and lost opportunity costs.

Recommend only setting the `comp/blo` address if it has not been set already.

Recommend distributing the rewards up to now before cancelling rewards using `_dropCompMarket`.

ghoul-sol (Based Loans) commented:

> This is technically correct, however, having a context that admin role is only temporary and will be moved to governance in the near future, I don't consider this as an issue. Especially that `Comptroller` is using a proxy pattern so admin can always change the implementation at will. I consider this a non-critical issue.

cemozerr (Judge) commented:

> I'm rating this a non-critical issue, as the `Comptroller` using a proxy pattern would make this change redundant.

[N-03] UNISWAPANCHOREDVIEW'S PRICEUPDATED EVENT IS NEVER FIRED

`UniswapAnchoredView`'s `PriceUpdated` event is never fired.

Unused code can hint at programming or architectural errors.

Recommend using it or removing it.

cemozerr (Judge) commented:

> I'm rating this as non-critical as an unused event has no drawbacks.

[N-04] MULTIPLE ERROR ENUMS WITH OVERLAPPING VALUES

There are 3 error enums, which have overlapping values. This allows for mistakes with error codes and might make troubleshooting of deployed code more difficult.

There did not appear to be any such mistakes in the current code, but changes in the future might introduce mistakes.

ErrorReporter.sol:

```
contract ComptrollerErrorReporter {
    enum Error {
        NO_ERROR,
        UNAUTHORIZED,
        COMPTROLLER_MISMATCH,

contract TokenErrorReporter {
    enum Error {
        NO_ERROR,
        UNAUTHORIZED,
        BAD_INPUT,
```

CarefulMath.sol

```
contract CarefulMath {
    enum MathError {
        NO_ERROR,
        DIVISION_BY_ZERO,
```

Recommend inserting dummy values in the enums to make sure all equivalent numeric values are different.

Take care that the same enum values still have the same underlying value to prevent new mistakes.

You could for example do the following:

```
ComptrollerErrorReporter  NO_ERROR = 0
ComptrollerErrorReporter  UNAUTHORIZED = 1
ComptrollerErrorReporter  COMPTROLLER_MISMATCH = 2
TokenErrorReporter        NO_ERROR = 0
TokenErrorReporter        UNAUTHORIZED = 1
TokenErrorReporter        BAD_INPUT = 102
CarefulMath.sol           NO_ERROR = 0
CarefulMath.sol           DIVISION_BY_ZERO = 201
```

Note: In a few occasions there is a reliance on the fact that NO_ERROR = 0; see seperate issue.

Note this might break compatibility with Compound and/or other deployed code.

ghoul-sol (Based Loans) acknowledged:

> This is for sure confusing and should be refactored. However, it has very low priority so I'm going to add it to the backlog.

[N-05] NOW IS STILL USED

Most of the time `block.timestamp` is used, however in 1 location `now` is still used.

The global variable now is deprecated in solidity 7: https://docs.soliditylang.org/en/v0.7.0/070-breaking-changes.html#changes-to-the-syntax

Recommend replacing `now` with `block.timestamp`.

ghoul-sol (Based Loans) commented:

> Added to backlog for later refactoring, thanks!

[N-06] RELIANCE ON THE FACT THAT NO_ERROR = 0

In several occasions it's relied upon that the error value NO_ERROR is equivalent to 0.

No problems detected based on this yet, but however in most locations there is an explicit check for NO_ERROR and comparing with 0 allows for possible future mistakes (especially if the enums would change).

Recommend replacing 0 with the appropriate version of …NO_ERROR

ghoul-sol (Based Loans) acknowledged:

> Added to backlog.

[N-07] ALPHABETICAL ORDER NOT COMPLIED WITH (CONTRARY TO THE COMMENTS)

The enum `FailureInfo` in ErrorReporter.sol has a comment that the values are sorted in alphabetical order.

However they are not in alphabetical order.

Recommend sorting the enum values in alphabetical order or remove the comment.

ghoul-sol (Based Loans) acknowledged:

> Added to backlog, thanks!

[N-08] REQUIRENOERROR NOT USED IN A CONSISTENT WAY

Cether.sol has a function `requireNoError` to check for errors. This is used most of the time, however in one occasion it isn't used.

```
function getCashPrior() internal view returns (uint) {
    (MathError err, uint startingBalance) = subUInt(address(this).balance, msg.value);
    require(err == MathError.NO_ERROR);
    return startingBalance;
}
```

Recommend replacing *require(err == MathError.NO*ERROR);_

with:

requireNoError(err, "getCashPrior failed");

ghoul-sol (Based Loans) acknowledged:

> Technically, the code works but I agree that consistency should be kept. Added to backlog.

[N-09] UINT(-1)

In several occasions constructions like `uint(-1)` and `uint96(-1)` are used the reference the maximum values of uint and uint96.

This relies on the peculiarities of numbers.

Solidity also allows the following constructions:

- type(uint).max;
- type(uint96).max;

.\CToken.sol: startingAllowance = uint(-1); .\CToken.sol: if (startingAllowance != uint(-1)) { .\CToken.sol: if (repayAmount == uint(-1)) { .\CToken.sol: if (repayAmount == uint(-1)) { .\Governance\Blo.sol: if (rawAmount == uint(-1)) { .\Governance\Blo.sol: amount = uint96(-1); .\Governance\Blo.sol: if (spender != src && spenderAllowance != uint96(-1)) { .\UniswapOracle\UniswapConfig.sol: if (index != uint(-1)) { .\UniswapOracle\UniswapConfig.sol: if (index != uint(-1)) { .\UniswapOracle\UniswapConfig.sol: if (index != uint(-1)) {

Recommend replacing `uint(-1)` with `type(uint).max` and replacing `uint96(-1)` with `type(uint96).max`.

ghoul-sol (Based Loans) acknowledged:

> Added to backlog for later refactoring, thanks!

[N-10] MORE READABLE CONSTANTS

Some constant values are difficult to read in one time because they have at lot of 0's. Solidity allows _ to separate series of zeroes.

Recommend the following replacements:

- 1000000e18 with 1_000_000e18
- 4000000e18 with 4_000_000e18
- 100000000e18 with 100_000_000e18

Comments:

ghoul-sol (Based Loans) confirmed:

> Fixed as recommended

[N-11] FUNCTION GETUNDERLYINGPRICE COMPARES AGAINST "CETH"

Contract CompoundLens functions `cTokenMetadata` and `cTokenBalances` compare against "bETH" while contract `SimplePriceOracle` function `getUnderlyingPrice` compares against "cETH". It is not clear if this `SimplePriceOracle` will be used in production, probably only for testing, but still would be nice to unify it across all the contracts.

Recommend replacing "cETH" with "bETH" in SimplePriceOracle function `getUnderlyingPrice`.

ghoul-sol (Based Loans) acknowledged:

> This is not meant to be used on production, however, this contract is confusing and would not work if used so it was deleted. Thanks for pointing it out!

[N-12] USE 'INTERFACE' KEYWORD FOR INTERFACES

Interfaces are declared as contracts. For example, `ComptrollerInterface` name indicates that it should be an interface but it is declared as a contract. Solidity has a keyword "interface" that can be used here.

Recommend declaring interfaces with 'interface' keyword.

Comments:

ghoul-sol (Based Loans) acknowledged:

> Added to backlog, thanks!

[N-13] [INFO] FUNCTIONS ‘GETUNDERLYINGPRICEVIEW’ AND ‘PRICE’ ARE TOO SIMILAR

Not a bug, just FYI:

Function `getUnderlyingPriceView` is too similar to function `price`. It would be best to avoid code duplication by extracting common code and using it where necessary. Less code duplication makes it easier to maintain it and improves readability.

ghoul-sol (Based Loans) acknowledged:

> It’s added to our backlog for refactoring, thanks!

[N-14] REQUIRES A NON-ZERO ADDRESS CHECK WHEN DEPLOYING CERC20 TOKENS AND CETHER.

During the deployment of the contracts `CErc20` and `CErc20Immutable`, both input parameters `underlying_` and `ComptrollerInterface` lack a non-zero address check. In `CEther`, the `ComptrollerInterface` is not required to be non-zero either. If any of them were provided as `0` accidentally, there is no way to change the values, and the contract should be redeployed.

Recommend adding non-zero address checks in the constructor of the `CErc20`, `CErc20Immutable`, and `CEther` contracts.

ghoul-sol (Based Loans) commented:

> It’s definitely a good practice to require non-zero address, however, it’s not a threat. Severity should be 0.
>
> Added to backlog, thanks!

cemozerr (Judge) commented:

> Both the impact and the likelihood of this bug is low, so rating this as non-critical.

[N-15] MISSING EVENT VISIBILITY IN _SETCOMPADDRESS() FUNCTION

The `_setCompAddress()` function in the Comptroller contract does not emit an event when changing the comp address. While this does not impose any security risk, it does hinder a users ability to view any changes made to the comp address through the contract's lifetime.

It is recommended to emit an event indicating the old comp address, and the new comp address to be used when calling the `_setCompAddress()` function. An example of such an event is `event NewCompAddress(address oldCompAddress, address newCompAddress)`.

ghoul-sol (Based Loans) confirmed:

> Fixed as recommended.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code, but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.