Introducing Code4rena Blue: Dedicated defense. Competitive bounties. Independent judging.
Learn more →

Canto contest
Findings & Analysis Report

2022-10-18

Table of contents

- Overview
  - About C4
  - Wardens
- Summary
- Scope
- Severity Criteria
- High Risk Findings (14)
  - [H-01] Anyone can set the `baseRatePerYear` after the `updateFrequency` has passed
  - [H-02] Stealing Wrapped Manifest in WETH.sol
  - [H-03] `AccountantDelegate` : `sweepInterest` function will destroy the cnote in the contract.
  - [H-04] `lending-market/NoteInterest.sol` Wrong implementation of `getBorrowRate()`
  - [H-05] `zeroswap/UniswapV2Library.sol` Wrong init code hash in `UniswapV2Library.pairFor()` will break `UniswapV2Oracle` , `UniswapV2Router02` , `SushiRoll`

    - [H-06] Accountant can’t be initialized
    - [H-07] Anyone can create Proposal Unigov `Proposal-Store.sol`
    - [H-08] Transferring any amount of the underlying token to the CNote contract will make the contract functions unusable
    - [H-09] WETH.sol computes the wrong `totalSupply()`
    - [H-10] Comptroller uses the wrong address for the WETH contract
    - [H-11] `lending-market/Note.sol` Wrong implementation of access control
    - [H-12] In `ERC20`, `TotalSupply` is broken
    - [H-13] It’s not possible to execute governance proposals through the `GovernorBravoDelegate` contract
    - [H-14] `WETH.allowance()` returns wrong result
- Medium Risk Findings (12)
    - [M-01] Missing zero address check can set treasury to zero address
    - [M-02] Only the `state()` of the latest proposal can be checked
    - [M-03] Unable to check `state()` if `proposalId == 0`
    - [M-04] accountant address can be set to zero by anyone leading to loss of funds/tokens
    - [M-05] Incorrect amount taken
    - [M-06] Overprivileged admin can grant unlimited WETH
    - [M-07] CNote updates the accounts after sending the funds, allowing for reentrancy
    - [M-08] `zeroswap/UniswapV2Pair.sol` Token reserves per lp token can be manipulated due to lack of `MINIMUM_LIQUIDITY` when minting the first liquidity with `migrator`
    - [M-09] Incorrect condition always bound to fail
    - [M-10] Oracle may be attacked if an attacker can pump the tokens for the entire block
    - [M-11] In `Cnote.sol`, anyone can initially become both accountant and admin

  - [M-12] Note: When `_initialSupply ! = 0` , the `_mint_to_Accountant` function will fail
- Low Risk and Non-Critical Issues
  - L-01 assert statement should not be used
  - L-02 CloseFactor unbounded
  - L-03 Immutable addresses lack zero-address check
  - L-04 Receive function
  - L-05 Local variable shadowing
  - L-06 Avoid Using `.Transfer` to Transfer Native Tokens
  - N-01 Underflow desired but not possible
  - N-02 Comment Missing function parameter
  - N-03 Constants instead of magic numbers
  - N-04 Constructor visibility
  - N-05 Events indexing
  - N-06 Event should be emitted in setters
  - N-07 Function missing comments
  - N-08 Function order
  - N-09 Non-library files should use fixed compiler versions
  - N-10 Open TODOs
  - N-11 Public functions can be external
  - N-12 Require statements should have descriptive strings
  - N-13 Scientific notation
  - N-14 Styling
  - N-15 Typos
- Gas Optimizations
  - G-01 Initialising to Default Values
  - G-02 Emitting Storage Variables
  - G-03 Variables Can be Immutable/Constant

    - G-04 For Loop Optimisations
    - G-05 Minimising SLOAD’s
    - G-06 No Need to check == True in Conditional Statements
    - G-07 Custom Errors
    - G-08 Long Revert Strings
    - G-09 Uint > 0 Checks in Require Functions
    - G-10 && in Require Functions
- Disclosures

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit 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 audit contest outlined in this document, C4 conducted an analysis of the Canto smart contract system written in Solidity. The audit contest took place between June 14—June 21 2022.

Note: this audit contest originally ran under the name `New Blockchain`*.*

Wardens

63 Wardens contributed reports to the Canto contest:

1. WatchPug (jtp and ming)
2. cccz
3. hake
4. Ruhum
5. 0xf15ers (remora and twojoy)

6. Picodes
7. cryptphi
8. hansfriese
9. Chom
10. p4st13r4 (0x69e8 and 0xb4bb4)
11. Tutturu
12. gzeon
13. 0x52
14. codexploder
15. zzzitron
16. hyh
17. 0x1f8b
18. csanuragjain
19. joestakey
20. Soosh
21. TerrierLover
22. defsec
23. catchup
24. Dravee
25. _Adam
26. Lambda
27. 0xDjango
28. saian
29. 0xmint
30. oyc_109
31. 0xNazgul
32. robee
33. dipp
34. k

35. JMukesh
36. TomJ
37. Limbooo
38. Waze
39. 0xKitsune
40. Funen
41. sach1r0
42. simon135
43. fatherOfBlocks
44. 0x29A (0x4non and rotcivegaf)
45. c3phas
46. MadWookie
47. Bronicle
48. asutorufos
49. technicallyty
50. nxrblsrpr
51. ignacio
52. 0xkatana
53. JC
54. rfa
55. Tomio
56. ynnad
57. 0v3rf10w
58. ak1
59. Fitraldys

This contest was judged by Alex the Entreprenerd.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 26 unique vulnerabilities. Of these vulnerabilities, 14 received a risk rating in the category of HIGH severity and 12 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 45 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 39 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Canto contest repository, and is composed of 15 smart contracts written in the Solidity programming language and includes 2,379 lines of Solidity code. One Cosmos SDK blockchain is also included.

Severity Criteria

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

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

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 (14)

[H-01] Anyone can set the `baseRatePerYear` after the `updateFrequency` has passed

Submitted by 0xDjango, also found by 0x52, Chom, csanuragjain, JMukesh, k, oyc_109, Picodes, Soosh, and WatchPug

https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/NoteInterest.sol#L118-L129

The `updateBaseRate()` function is public and lacks access control, so anyone can set the critical variable `baseRatePerYear` once the block delta has surpassed the `updateFrequency` variable. This will have negative effects on the borrow and supply rates used anywhere else in the protocol.

The updateFrequency is explained to default to 24 hours per the comments, so this vulnerability will be available every day. Important to note, the admin can fix the `baseRatePerYear` by calling the admin-only `_setBaseRatePerYear()` function. However, calling this function does not set the `lastUpdateBlock` so users will still be able to change the rate back after the 24 hours waiting period from the previous change.

Proof of Concept

```
    function updateBaseRate(uint newBaseRatePerYear) public {
        // check the current block number
        uint blockNumber = block.number;
        uint deltaBlocks = blockNumber.sub(lastUpdateBlock);


        if (deltaBlocks > updateFrequency) {
            // pass in a base rate per year
            baseRatePerYear = newBaseRatePerYear;
            lastUpdateBlock = blockNumber;
            emit NewInterestParams(baseRatePerYear);
        }
```

```
            }
```

Recommended Mitigation Steps

I have trouble understanding the intention of this function. It appears that the rate should only be able to be set by the admin, so the `_setBaseRatePerYear()` function seems sufficient. Otherwise, add access control for only trusted parties.

tkkwon1998 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to probably an oversight, a core function that has impact in determining the yearly interest rate was left open for anyone to change once every 24 hrs.

> Because the impact is:

- Potential bricking of integrating contracts
- Economic exploits

> And anyone can perform it

> I believe that High Severity is appropriate.

> Mitigation requires either deleting the function or adding access control.

[H-02] Stealing Wrapped Manifest in WETH.sol

Submitted by Soosh, also found by 0x52, 0xDjango, cccz, saian, TerrierLover, WatchPug, and zzzitron

https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/WETH.sol#L85

Allows anyone to steal all wrapped manifest from the WETH.sol contract. Attacker can also withdraw to convert Wrapped Manifest to Manifest.

Issue in approve(address owner, address spender) external function. This allows an attacker to approve themselves to spend another user's tokens.

Attacker can then use transferFrom(address src, address dst, uint wad) function to send tokens to themself.

Proof of Concept

See warden's full report for further details.

Tools Used

VScode, hardhat

Recommended Mitigation Steps

I believe there is no need for this function. There is another approve(address guy, uint wad) function that uses msg.sender to approve allowance. There should be no need for someone to approve another user's allowance.

Remove the approve(address owner, address spender) function.

tkkwon1998 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, for whatever reason, an approve function which allows to pass the "approver" as parameter was present in the WETH contract.

> This allows anyone, to steal all WETH from any other holder.

> For that reason, High Severity is appropriate.

[H-03] `AccountantDelegate: sweepInterest` function will destroy the cnote in the contract.

Submitted by cccz, also found by WatchPug

When the user borrows note tokens, the AccountantDelegate contract provides note tokens and gets cnote tokens. Later, when the user repays the note tokens, the cnote tokens are destroyed and the note tokens are transferred to the AccountantDelegate contract. However, in the sweepInterest function of the AccountantDelegate contract, all cnote tokens in the contract will be transferred to address 0. This will prevent the user from repaying the note tokens, and the sweepInterest function will not calculate the interest correctly later.

Proof of Concept

https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/Accountant/AccountantDelegate.sol#L74-L92
https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/CToken.sol#L533

Recommended Mitigation Steps

```
        function sweepInterest() external override returns(uint) {

                    uint noteBalance = note.balanceOf(address(this))
                    uint CNoteBalance = cnote.balanceOf(address(this

                    Exp memory expRate = Exp({mantissa: cnote.exchar
                    uint cNoteConverted = mul_ScalarTruncate(expRate
                    uint noteDifferential = sub_(note.totalSupply(),

                    require(cNoteConverted >= noteDifferential, "Not

                    uint amtToSweep = sub_(cNoteConverted, noteDiffe

                    note.transfer(treasury, amtToSweep);

    -               cnote.transfer(address(0), CNoteBalance);

                    return 0;
        }
```

[tkkwon1998 (Canto) confirmed](#)

[Alex the Entreprenerd (judge) commented](#):

> The warden has shown how, due to a programmer mistake, interest bearing Note will be burned.

> It is unclear why this decision was made, and I believe the sponsor should look into `redeem` ing the `cNote` over destroying it.

> The sponsor confirmed, and because this finding shows unconditional loss of assets, I agree with High Severity.

[H-04] `lending-market/NoteInterest.sol` Wrong implementation of `getBorrowRate()`

Submitted by WatchPug, also found by 0x1f8b, Chom, and gzeon

https://github.com/Plex-Engineer/lending-market/blob/b93e2867a64b420ce6ce317f01c7834a7b6b17ca/contracts/NoteInterest.sol#L92-L101

```
function getBorrowRate(uint cash, uint borrows, uint reserves) p
    // Gets the Note/gUSDC TWAP in a given interval, as a mantis
    // uint twapMantissa = getUnderlyingPrice(note);
    uint rand = uint(keccak256(abi.encodePacked(msg.sender))) %
    uint ir = (100 - rand).mul(adjusterCoefficient).add(baseRate
    uint newRatePerYear = ir >= 0 ? ir : 0;
    // convert it to base rate per block
    uint newRatePerBlock = newRatePerYear.div(blocksPerYear);
    return newRatePerBlock;
}
```

The current implementation will return a random rate based on the caller’s address and `baseRatePerYear`.

This makes some lucky addresses pay much lower and some addresses pay much higher rates.

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to most likely a developer oversight, the unimplemented `getBorrowRate` returns a random value which can easily be gamed (and is not recommended for production).

> Because the contract is in scope, and the functionality is broken, I agree with High Severity.

[H-05] `zeroswap/UniswapV2Library.sol` Wrong init code hash in `UniswapV2Library.pairFor()` will break `UniswapV2Oracle`, `UniswapV2Router02`, `SushiRoll`

Submitted by WatchPug

https://github.com/Plex-Engineer/zeroswap/blob/03507a80322112f4f3c723fc68bed0f138702836/contracts/uniswapv2/libraries/UniswapV2Library.sol#L20-L28

```
    function pairFor(address factory, address tokenA, address tokenB
        (address token0, address token1) = sortTokens(tokenA, tokenB
        pair = address(uint(keccak256(abi.encodePacked(
                hex'ff',
                factory,
                keccak256(abi.encodePacked(token0, token1)),
                hex'e18a34eb0e04b04f7a0ac29a6e80748dca96319b42c54d67
            ))));
    }
```

The `init code hash` in `UniswapV2Library.pairFor()` should be updated since the code of `UniswapV2Pair` has been changed. Otherwise, the `pair` address calculated will be wrong, most likely non-existing address.

There are many other functions and other contracts across the codebase, including `UniswapV2Oracle`, `UniswapV2Router02`, and `SushiRoll`, that rely on the `UniswapV2Library.pairFor()` function for the address of the pair, with the `UniswapV2Library.pairFor()` returning a wrong and non-existing address, these functions and contracts will malfunction.

Recommended Mitigation Steps

Update the init code hash from `hex'e18a34eb0e04b04f7a0ac29a6e80748dca96319b42c54d679cb821dca90c6303'` to the value of `UniswapV2Factory.pairCodeHash()`.

tkkwon1998 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> Amazing catch, because the contract bytecode has been change, the init hash will be different.

> While the bug seems trivial, it's impact is a total bricking of all swapping functionality as the Library will cause all Periphery Contracts to call to the wrong addresses.

> Because of the impact, I agree with High Severity.

[H-06] Accountant can't be initialized

Submitted by Ruhum, also found by cccz

It's not possible to initialize the accountant because of a mistake in the function's require statement.

I rate it as MED since a key part of the protocol wouldn't be available until the contract is modified and redeployed.

Proof of Concept

The issue is the following `require()` statement: https://github.com/Plex-Engineer/lending-market/blob/main/contracts/Accountant/AccountantDelegate.sol#L29

There, the function checks whether the accountant has received the correct amount of tokens. But, it compares the accountant's balance with the `_initialSupply`. That value is always 0. So the require statement will always fail

When the Note contract is initialized, `_initialSupply` is set to 0:

- https://github.com/Plex-Engineer/lending-market/blob/main/deploy/canto/004_deploy_Note.ts#L14
- https://github.com/Plex-Engineer/lending-market/blob/main/contracts/Note.sol#L9
- https://github.com/Plex-Engineer/lending-market/blob/main/contracts/ERC20.sol#L32

After `_mint_to_Accountant()` mints `type(uint).max` tokens to the accountant: https://github.com/Plex-Engineer/lending-market/blob/main/contracts/Note.sol#L18
That increases the `totalSupply` but not the `_initialSupply`: https://github.com/Plex-Engineer/lending-market/blob/main/contracts/ERC20.sol#L242

The `_initialSupply` value is only modified by the ERC20 contract's constructor.

Recommended Mitigation Steps

Change the require statement to

```
    require(note.balanceOf(msg.sender) == note.totalSupply(), "Accou
```

nivasan1 (Canto) confirmed

Alex the Entreprenerd (judge) increased severity to High and commented:

> The warden has shown how, due to an incorrect assumption, `AccountantDelegate.initialize` cannot work, meaning part of the protocol will never work without fixing this issue.

> While the change should be fairly trivial, the impact is pretty high, for those reasons am going to raise severity to High.

[H-07] Anyone can create Proposal Unigov `Proposal-Store.sol`

Submitted by Soosh, also found by 0x1f8b, cccz, csanuragjain, hake, p4st13r4, Ruhum, TerrierLover, WatchPug, and zzzitron

https://github.com/Plex-Engineer/manifest/blob/688e9b4e7835854c22ef44b045d6d226b784b4b8/contracts/Proposal-Store.sol#L46
https://github.com/Plex-Engineer/lending-market/blob/b93e2867a64b420ce6ce317f01c7834a7b6b17ca/contracts/Governance/GovernorBravoDelegate.sol#L37

Proposal Store is used to store proposals that have already passed (https://code4rena.com/contests/2022-06-new-blockchain-contest#unigov-module-615-sloc) " Upon a proposal's passing, the proposalHandler either deploys the ProposalStore contract (if it is not already deployed) or appends the proposal into the ProposalStore's mapping (uint ⇒ Proposal)"

But anyone can add proposals to the contract directly via AddProposal() function.

Unigov proposals can be queued and executed by anyone in GovernorBravoDelegate contract
https://github.com/Plex-Engineer/lending-market/blob/b93e2867a64b420ce6ce317f01c7834a7b6b17ca/contracts/Governance/GovernorBravoDelegate.sol#L37

Proof of Concept

https://github.com/Plex-Engineer/manifest/blob/688e9b4e7835854c22ef44b045d6d226b784b4b8/contracts/Proposal-Store.sol#L46

Recommended Mitigation Steps

Authorization checks for AddProposal, only governance module should be able to update.

tkkwon1998 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to a lack of checks, anyone can create, queue, and execute a proposal without any particular checks.

> Because governance normally is limited via:

- Voting on a proposal
- Access control to limit transactions

> And the finding shows how this is completely ignored;

> I believe High Severity to be appropriate.

[H-08] Transferring any amount of the underlying token to the CNote contract will make the contract functions unusable

Submitted by Tutturu, also found by 0x52, hyh, p4st13r4, and WatchPug

The contract expects the balance of the underlying token to == 0 at all points when calling the contract functions by requiring getCashPrior() == 0, which checks token.balanceOf(address(this)) where token is the underlying asset.

An attacker can transfer any amount of the underlying asset directly to the contract and make all of the functions requiring getCashPrior() == 0 to revert.

Proof of Concept

CNote.sol#L43

CNote.sol#L114

CNote.sol#198

CNote.sol#310

1. Attacker gets any balance of Note (amount = 1 token)
2. Attacker transfers the token to CNote which uses Note as an underlying asset, by calling note.transfer(CNoteAddress, amount). The function is available since Note inherits from ERC20
3. Any calls to CNote functions now revert due to getCashPrior() not being equal to 0

Recommended Mitigation Steps

Instead of checking the underlying token balance via balanceOf(address(this)) the contract could hold an internal balance of the token, mitigating the impact of tokens being forcefully transferred to the contract.

tkkwon1998 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, via a simple transfer of 1 wei of token, the invariant of `getCashPrior() == 0` can be broken, bricking the functionality.

> Because of:

- the simplicity of the exploit
- The impact being inability to interact with the contract
- A protocol invariant is broken

> I agree with High Severity.

> Mitigation would require using delta balances and perhaps re-thinking the need for those intermediary checks.

[H-09] WETH.sol computes the wrong `totalSupply()`

Submitted by p4st13r4, also found by hansfriese, Ruhum, TerrierLover, WatchPug, and zzzitron

Affected code:

- https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/WETH.sol#L47

`WETH.sol` is almost copied from the infamous WETH contract that lives in mainnet. This contract is supposed to receive the native currency of the blockchain (for example ETH) and wrap it into a tokenized, ERC-20 form. This contract computes the `totalSupply()` using the balance of the contract itself stored in the `balanceOf` mapping, when instead it should be using the native `balance` function. This way, `totalSupply()` always returns zero as the `WETH` contract itself has no way of calling `deposit` to itself and increase its own balance

Proof of Concept

1. Alice transfers 100 ETH to `WETH.sol`
2. Alice calls `balanceOf()` for her address and it returns 100 WETH
3. Alice calls `totalSupply()`, expecting to see 100 WETH, but it returns 0

Tools Used

Editor

Recommended Mitigation Steps

```
    function totalSupply() public view returns (uint) {
        return address(this).balance
    }
```

[tkkwon1998 (Canto) confirmed]()

[Alex the Entreprenerd (judge) commented]():

> The warden has shown how, due to a programming mistake, the WETH totalSupply will be incorrect.

> Mitigation seems straightforward, however, because the vulnerability would have causes totalSupply to return 0, and shows a broken functionality for a core contract, I think High Severity to be appropriate

[H-10] Comptroller uses the wrong address for the WETH contract

Submitted by Ruhum, also found by 0xf15ers, cccz, hake, Soosh, and WatchPug

The Comptroller contract uses a hardcoded address for the WETH contract which is not the correct one. Because of that, it will be impossible to claim COMP rewards. That results in a loss of funds so I rate it as HIGH.

Proof of Concept

The Comptroller's `getWETHAddress()` function: https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/Comptroller.sol#L1469

It's a left-over from the original compound repo: https://github.com/compound-finance/compound-protocol/blob/master/contracts/Comptroller.sol#L1469

It's used by the `grantCompInternal()` function: https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/Comptroller.sol#L1377

That function is called by `claimComp()`: https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/Comptroller.sol#L1365

If there is a contract stored in that address and it doesn't adhere to the interface (doesn't have a `balanceOf()` and `transfer()` function), the transaction will revert. If there is no contract, the call will succeed without having any effect. In both cases, the user doesn't get their COMP rewards.

Recommended Mitigation Steps

The WETH contract's address should be parsed to the Comptroller through the constructor or another function instead of being hardcoded.

[tkkwon1998 (Canto) confirmed]

[Alex the Entreprenerd (judge) commented]:

> The warden has shown how the address for WETH / comp is hardcoded and the address is pointing to Mainnet's COMP.

> This misconfiguration will guarantee that any function calling `grantCompInternal` as well as `claimComp` will revert.

> Because the functionality is hampered, I agree with High Severity.

[H-11] `lending-market/Note.sol` Wrong implementation of access control

Submitted by WatchPug, also found by catchup, Lambda, p4st13r4, and Tutturu

[https://github.com/Plex-Engineer/lending-market/blob/b93e2867a64b420ce6ce317f01c7834a7b6b17ca/contracts/Note.sol#L13-L31]

```
    function _mint_to_Accountant(address accountantDelegator) exterr
        if (accountant == address(0)) {
            _setAccountantAddress(msg.sender);
        }
        require(msg.sender == accountant, "Note::_mint_to_Accountant
        _mint(msg.sender, type(uint).max);
    }
    function RetAccountant() public view returns(address) {
        return accountant;
    }

    function _setAccountantAddress(address accountant_) internal {
        if(accountant != address(0)) {
            require(msg.sender == admin, "Note::_setAccountantAddres
        }
        accountant = accountant_;
        admin = accountant;
    }
```

`_mint_to_Accountant()` calls `_setAccountantAddress()` when `accountant == address(0)`, which will always be the case when `_mint_to_Accountant()` is called for the first time.

And `_setAccountantAddress()` only checks if `msg.sender == admin` when `accountant != address(0)` which will always be `false`, therefore the access control is not working.

L17 will then check if `msg.sender == accountant`, now it will always be the case, because at L29, `accountant` was set to `msg.sender`.

tkkwon1998 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to a flaw in logic, via a front-run, anyone can become the `accountant` and mint all the totalSupply to themselves.

> While I'm not super confident on severity for the front-run as I'd argue the worst case is forcing a re-deploy, the warden has shown a lack of logic in the checks (`msg.sender == admin`) which breaks it's invariants.

> For that reason, I think High Severity to be appropriate.

[H-12] In `ERC20`, `TotalSupply` is broken

Submitted by Picodes, also found by cccz

https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/ERC20.sol#L33 https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/ERC20.sol#L95

For an obscure reason as it's not commented, `_totalSupply` is not initialized to 0, leading to an inaccurate total supply, which could easily break integrations, computations of market cap, etc.

Proof of Concept

If the constructor is called with `_initialSupply = 1000`, then `1000` tokens are minted. The total supply will be `2000`.

Recommended Mitigation Steps

Remove `_initialSupply`.

tkkwon1998 (Canto) disputed and commented:

> The explanation is not clear. We can't seem to reproduce this issue as we can't find a scenario where the `totalSupply` function returns an incorrect value.

Picodes (warden) commented:

> @tkkwon1998 to clarify:

> Deploy the ERC20 with `totalSupply_ = 1000`.

> Then `totalSupply()` returns 1000, which is incorrect.

> Then if someone mints 1000 tokens, there is 1000 tokens in the market but due to `_totalSupply += amount;`, totalSupply = 2000 which is still incorrect

Alex the Entreprenerd (judge) commented:

> I believe the submission could have benefitted by:

- A coded POC
- Recognizing a revert due to the finding

> However the finding is ultimately true in that, because `totalSupply` is a parameter passed in to the contract, and the ERC20 contract will not mint that amount, the `totalSupply` will end up not reflecting the total amounts of tokens minted.

> For this reason, I believe the finding to be valid and High Severity to be appropriate.

> I recommend the warden to err on the side of giving too much information to avoid getting their finding invalidated incorrectly.

Alex the Entreprenerd (judge) commented:

> After further thinking, I still believe the finding is of high severity as the ERC20 standard is also broken. I do believe the submission could have been better developed, however, I think High is in place here.

[H-13] It’s not possible to execute governance proposals through the `GovernorBravoDelegate` contract

Submitted by Ruhum, also found by 0xmint, cccz, csanuragjain, dipp, hake, and zzzitron

It’s not possible to execute a proposal through the GovernorBravoDelegate contract because the `executed` property of it is set to `true` when it’s queued up.

Since this means that the governance contract is unusable, it might result in locked-up funds if those were transferred to the contract before the issue comes up. Because of that I’d rate it as HIGH.

Proof of Concept

`executed` is set to `true`: https://github.com/Plex-Engineer/lending-market/blob/main/contracts/Governance/GovernorBravoDelegate.sol#L63

Here, the `execute()` function checks whether the proposal's state is `Queued`: https://github.com/Plex-Engineer/lending-market/blob/main/contracts/Governance/GovernorBravoDelegate.sol#L87

But, since the `execute` property is `true`, the `state()` function will return `Executed`: https://github.com/Plex-Engineer/lending-market/blob/main/contracts/Governance/GovernorBravoDelegate.sol#L117

In the original compound repo, `executed` is `false` when the proposal is queued up: https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/GovernorBravoDelegate.sol#L111

Recommended Mitigation Steps

Just delete the line where `executed` is set to `true`. Since the zero-value is `false` anyway, you'll save gas as well.

tkkwon1998 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to a coding decision, no transaction can be executed from the Governor Contract.

> Because the functionality is broken, I agree with High Severity.

[H-14] `WETH.allowance()` returns wrong result

Submitted by hansfriese, also found by 0xf15ers

https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/WETH.sol#L104

WETH.allowance() returns wrong result.
I can't find other contracts that use this function but WETH.sol is a base contract and it should be fixed properly.

Proof of Concept

In this function, the "return" keyword is missing and it will always output 0 in this case.

Tools Used

Solidity Visual Developer of VSCode

Recommended Mitigation Steps

L104 should be changed like below.

```
    return _allowance[owner][spender];
```

[nivasan1 (Canto) confirmed](#)

[Alex the Entreprenerd (judge) increased severity to High and commented](#):

> The warden has found a minor developer oversight, which will cause the view function `allowance` to always return 0.

> Breaking of a core contract such as WETH is a non-starter.

> Because I've already raised severity of #191 for similar reasons, I think High Severity is appropriate in this case.

Medium Risk Findings (12)

[M-01] Missing zero address check can set treasury to zero address

Submitted by cryptphi

AccountantDelegate.initialize() is missing a zero address check for `treasury_` parameter, which could maybe allow treasury to be mistakenly set to 0 address.

Proof of Concept

https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/Accountant/AccountantDelegate.sol#L20

Recommended Mitigation Steps

Add a require() check for zero address for the treasury parameter before changing the treasury address in the initialize function.

nivasan1 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> Because:

- The finding is technically correct
- The `treasury` variable is only set on the initializer
- An incorrect setting could cause loss of funds

> I'm going to mark the finding as valid and of Medium severity.

> In mentioning this report in the future, notice that the conditions that caused me to raise the severity weren't simply the lack of a check, but the actual risk of loss of funds, and the inability to easily fix.

[M-02] Only the `state()` of the latest proposal can be checked

Submitted by hake

https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/Governance/GovernorBravoDelegate.sol#L115 `state()` function cannot view the state from any proposal except for the latest one.

Proof of Concept

```
    require(proposalCount >= proposalId && proposalId > initialPropo
```

Currently `proposalCount` needs to be bigger or equal to `proposalId`. Assuming `proposalId` is incremented linearly in conjunction with `proposalCount`, this implies only the most recent `proposalId` will pass the `require()` check above. All other proposals will not be able to have their states checked via this function.

Recommended Mitigation Steps

Change above function to `proposalCount <= proposalId` (assuming `proposalId` is set linearly, which currently is not enforced by code).

nivasan1 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, due to a mistake in logic, only the `state` of the latest proposal can be read.

> Because the function `state` is used in `execute` we can conclude that only one proposal can be queue for execution at a time, drastically reducing the availability of the Governor.

> For this reason I believe medium severity is appropriate.

[M-03] Unable to check `state()` if `proposalId == 0`

Submitted by hake

https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/Governance/GovernorBravoDelegate.sol#L115

`state()` function cannot be called to view proposal state if `proposalId == 0`.

Proof of Concept

There is no check to prevent queueing a `proposalId` with a value of 0 via the `queue()` function.
However, in the `state()` function there is a check preventing using a `proposalId == 0`.
For clarity: `initialProposalId` must be zero according to `_initiate()`, therefore, `proposalId` cannot be 0 according to check below.

```
    function state(uint proposalId) public view returns (ProposalSta
        require(proposalCount >= proposalId && proposalId > initialP
```

Recommended Mitigation Steps

Implement check to preventing queueing a `proposalId == 0`.

nivasan1 (Canto) disputed and commented:

> The ProposalId cannot be 0 as the proposal IDs are fixed and will be set via the cosmos-sdk.

Alex the Entreprenerd (judge) commented:

> The warden has shown how, through a misconfiguration, a proposal could never be executable due to a revert in `state()`.

> While I believe the warden has already shown a remediation that would cover this scenario, I believe the Warden has shown a unique possible situation that can cause the system to stop working as intended.

> While the sponsor says the proposalId will never be 0, there is no way to avoid that at the Smart Contract level, meaning that any caller can set the proposal to 0.

> For these reasons, I think Medium Severity to be appropriate.

[M-04] accountant address can be set to zero by anyone leading to loss of funds/tokens

Submitted by cryptphi

https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/CNote.sol#L14-L21
https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/CNote.sol#L31
https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/CNote.sol#L96
https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/CNote.sol#L178
https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/CNote.sol#L258

In `CNote._setAccountantContract()` , the require() check only works when `address(_accountant) != address(0)` , leading to the ability to set `_accountant` state variable to the zero address, as well as setting admin to zero address.

The following below are impacts arising from above:

A. Users can gain underlying asset tokens for free by minting CToken in `mintFresh()` then calling `redeemFresh()`

Proof of Concept

1. Alice calls `_setAccountantContract()` with parameter input as 0.
2. The `_accountant` state variable is now 0.
3. Alice/or a contract calls `mintFresh()` with input address 0 and mintAmount 1000. (assuming function is external, reporting a separate issue on the mutability)
4. This passes the `if (minter == address(_accountant))` and proceeds to mint 1000 CTokens to address(0)
5. Alice then calls `redeemFresh()` with her address as the `redeemer` parameter, and redeemTokensIn as 1000.
6. Assume exchangeRate is 1, Alice would receive 1000 tokens in underlying asset.

B. Users could borrow CToken asset for free

A user can borrow CToken asset from the contract, then set `_accountant` to 0 after. With `_accountant` being set to 0 , the borrower , then call `repayBorrowFresh()` to have `_accountant` (address 0) to repay back the borrowed tokens assuming address(0) already has some tokens, and user's borrowed asset (all/part) are repaid.

Proof of Concept

1. Alice calls `borrowFresh()` to borrow 500 CTokens from contract.
2. Then Alice calls `_setAccountantContract()` with parameter input as 0.
3. The `_accountant` state variable is now 0.
4. With `_accountant` being set to 0, Alice calls `repayBorrowFresh()` having the payer be address 0, borrower being her address and 500 as repayAmount.
5. Assume address 0 already holds 1000 CTokens, Alice's debt will be fully repaid and she'll gain 500 CTokens for free.

C. Accounting contract could loses funds/tokens

When the `_accountant` is set to 0, CTokens/CNote will be sent to the zero address making the Accounting contract lose funds whenever `doTransferOut` is called.

Recommended Mitigation Steps

Instead of a `if (address(_accountant) != address(0))` statement, an additional require check to ensure `accountant_` parameter is not 0 address can be used in addition to the require check for caller is admin.

Change this

```
if (address(_accountant) != address(0)){
            require(msg.sender == admin, "CNote::_setAccountant(
        }
```

to this

```
require(msg.sender == admin, "CNote::_setAccountantContract:Only
require(accountant_ != address(0), "accoutant can't be zero addr
```

[tkkwon1998 (Canto) confirmed]

[Alex the Entreprenerd (judge) decreased severity to Medium and commented]:

> The warden has shown how, due to a misconfiguration, if the `accountant` is set to 0, users will be able to extra additional value (repay tokens for free).

> Because this is contingent on a misconfiguration, I believe Medium Severity to be more appropriate.

[M-05] Incorrect amount taken

Submitted by csanuragjain, also found by 0xf15ers and gzeon

https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/CNote.sol#L129

It was observed that in repayBorrowFresh function, User is asked to send repayAmount instead of repayAmountFinal. This can lead to loss of user funds as user might be paying extra

Proof of Concept

1. User is making a repayment which eventually calls repayBorrowFresh function
2. Assuming repayAmount == type(uint).max, so repayAmountFinal becomes accountBorrowsPrev
3. This means User should only transfer in accountBorrowsPrev instead of repayAmount but that is not true. Contract is transferring repayAmount instead of repayAmountFinal as seen at CNote.sol#L129

```
uint actualRepayAmount = doTransferIn(payer, repayAmount);
```

Recommended Mitigation Steps

Revise CNote.sol#L129 to below:

```
uint actualRepayAmount = doTransferIn(payer, repayAmountFinal);
```

tkkwon1998 (Canto) confirmed

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has showed how, due to an oversight, using `type(uint).max` to signify a complete repayment will actually attempt to transfer 2^256-1 units of token.

> While I think High severity would have been reasonable had the tokens gotten transferred, because what will actually happen is a revert, I think Medium Severity to be more appropriate.

> Remediation requires using `actualRepayAmount` or re-assigning the value of `repayAmount`

[M-06] Overprivileged admin can grant unlimited WETH

Submitted by hake

https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/Comptroller.sol#L1376

Admin can `_grantComp()` to any address using any amount and drain the contract.

Proof of Concept

If admin key gets compromised there is no timelock, no amount boundaries and no address limitations to prevent the assets to be drained immediately to the attacker's address.

Recommended Mitigation Steps

There is a few suggestions that could help mitigate this issue:
Implement timelock for `_grantComp()`
Implement hard coded recipient so funds cannot be arbitrarily sent to any address.
Implement a limit to the amount that can be granted.

Here is a reference to a past submission where this issue has been made by team Watchpug: https://github.com/code-423n4/2022-01-insure-findings/issues/271

nivasan1 (Canto) acknowledged and commented:

> We acknowledge that this is an issue, however we feel that changing the core functionality of compound would be too costly.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has shown how the Admin could sweep the reward token(in this case WETH) to any address, at any time, for an amount equal to all tokens available to the Comptroller.

> Because this is contingent on admin privilege, I think Medium Severity to be more appropriate.

[M-07] CNote updates the accounts after sending the funds, allowing for reentrancy

Submitted by hyh, also found by defsec

Having no reentrancy control and updating the records after external interactions allows for funds draining by reentrancy.

Setting the severity to medium as this is conditional to transfer flow control introduction on future upgrades, but the impact is up to the full loss of the available funds by unrestricted borrowing.

Proof of Concept

CNote runs doTransferOut before borrowing accounts are updated:

https://github.com/Plex-Engineer/lending-market/blob/2d423c7c3f62d65182d802deb99cc7bba4e057fd/contracts/CNote.sol#L70-L87

```
            /*
             * We invoke doTransferOut for the borrower and the borr
             *  Note: The cToken must handle variations between ERC-
             *  On success, the cToken borrowAmount less of cash.
             *  doTransferOut reverts if anything goes wrong, since
             */
            doTransferOut(borrower, borrowAmount);
            require(getCashPrior() == 0,"CNote::borrowFresh: Error i
        //Amount minted by Accountant is always flashed from account

        /* We write the previously calculated values into storage */
            accountBorrows[borrower].principal = accountBorrowsNew;
            accountBorrows[borrower].interestIndex = borrowIndex;
            totalBorrows = totalBorrowsNew;

            /* We emit a Borrow event */
            emit Borrow(borrower, borrowAmount, accountBorrowsNew, t
        }
```

Call sequence here is borrow() -> borrowInternal() -> borrowFresh() -> doTransferOut(), which transfers the token to an external recipient:

https://github.com/Plex-Engineer/lending-market/blob/2d423c7c3f62d65182d802deb99cc7bba4e057fd/contracts/CErc20.sol#L189-L200

```
/**
 * @dev Similar to EIP20 transfer, except it handles a False
 *      error code rather than reverting. If caller has not
 *      insufficient cash held in this contract. If caller h
 *      it is >= amount, this should not revert in normal co
 *
 *      Note: This wrapper safely handles non-standard ERC-2
 *            See here: https://medium.com/coinmonks/missing
 */
function doTransferOut(address payable to, uint amount) virt
    EIP20NonStandardInterface token = EIP20NonStandardInterf
    token.transfer(to, amount);
```

There an attacker can call exitMarket() that have no reentrancy control to remove the account of the debt:

https://github.com/Plex-Engineer/lending-market/blob/2d423c7c3f62d65182d802deb99cc7bba4e057fd/contracts/Comptroller.sol#L167-L174

https://github.com/Plex-Engineer/lending-market/blob/2d423c7c3f62d65182d802deb99cc7bba4e057fd/contracts/ComptrollerG7.sol#L157-L164

```
/**
 * @notice Removes asset from sender's account liquidity cal
 * @dev Sender must not have an outstanding borrow balance i
 *  or be providing necessary collateral for an outstanding
 * @param cTokenAddress The address of the asset to be remov
 * @return Whether or not the account successfully exited th
 */
function exitMarket(address cTokenAddress) override external
```

This attack was carried out several times:

https://certik.medium.com/fei-protocol-incident-analysis-8527440696cc

Recommended Mitigation Steps

Consider moving accounting update before funds were sent out, for example as it is done in CToken’s borrowFresh():

```
https:

        /*
         * We write the previously calculated values into st
         *  Note: Avoid token reentrancy attacks by writing 
        `*/
        accountBorrows[borrower].principal = accountBorrowsN
        accountBorrows[borrower].interestIndex = borrowIndex
        totalBorrows = totalBorrowsNew;

        /*
         * We invoke doTransferOut for the borrower and the 
         *  Note: The cToken must handle variations between 
         *  On success, the cToken borrowAmount less of cash
         *  doTransferOut reverts if anything goes wrong, si
         */
        doTransferOut(borrower, borrowAmount);
```

nivasan1 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> The warden has shown how, the in-scope codebase has historically been attacked due to a reentrancy attack.

> Because:

- The warden has provided POC and historical references
- The attack is contingent on a specific token that enables it

> I agree with Medium Severity.

[M-08] `zeroswap/UniswapV2Pair.sol` Token reserves per lp token can be manipulated due to lack of `MINIMUM_LIQUIDITY` when minting the first liquidity with `migrator`

Submitted by WatchPug

```
if (_totalSupply == 0) {
    address migrator = IUniswapV2Factory(factory).migrator();
    if (msg.sender == migrator) {
        liquidity = IMigrator(migrator).desiredLiquidity();
        require(liquidity > 0 && liquidity != uint256(-1), "Bad
    } else {
        require(migrator == address(0), "Must not have migrator'
        liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_
        _mint(address(0), MINIMUM_LIQUIDITY); // permanently loc
    }
} else {
    liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0,
}
```

https://github.com/Plex-Engineer/zeroswap/blob/03507a80322112f4f3c723fc68bed0f138702836/contracts/Migrator.sol#L28-L46

```
function migrate(IUniswapV2Pair orig) public returns (IUniswapV2
    require(msg.sender == chef, "not from master chef");
    require(block.number >= notBeforeBlock, "too early to migrat
    require(orig.factory() == oldFactory, "not from old factory'
    address token0 = orig.token0();
    address token1 = orig.token1();
    IUniswapV2Pair pair = IUniswapV2Pair(factory.getPair(token0,
    if (pair == IUniswapV2Pair(address(0))) {
        pair = IUniswapV2Pair(factory.createPair(token0, token1)
    }
    uint256 lp = orig.balanceOf(msg.sender);
    if (lp == 0) return pair;
    desiredLiquidity = lp;
    orig.transferFrom(msg.sender, address(orig), lp);
```

```
        orig.burn(address(pair));
        pair.mint(msg.sender);
        desiredLiquidity = uint256(-1);
        return pair;
    }
```

When minting LP tokens (`addLiquidity`), the amount of lp tokens you are getting is calculated based on `liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);` , if the `_totalSupply` is small enough, and `1 wei` of the lp token worth large amounts of `token0` and `token1` , the user who adds small amounts of liquidity will receive less amount of lp tokens due to precision loss.

A sophisticated attacker can artificially create that scenario by mint only `1 wei` of lp token and add `1e24` or even larger amounts of `token0` and `token1` by sending the tokens to the contract and then call `sync()` to update the reserves.

Then all the new depositors will lose up to `1e24` , let's say they deposited `1.99e24` , they will only receive `1 wei` of lp token, therefore, losing `0.99e24` of `token0` and `token1` .

This attack vector was mitigated the original version of UniswapV2Pair by introcuing the `MINIMUM_LIQUIDITY` minted and permanently lock in `address(0)` upon the first mint.

However, this can now be bypassed with the migrator, and this attacker vector is open again.

Recommended Mitigation Steps

Given the fact that zeroswap will be a DEX that does not need a feature to migrate liquidity from other DEXs, consider removing the migrator.

tkkwon1998 (Canto) acknowledged and commented:

> Issue acknowledged, we will not be migrating liquidity from zeroswap.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has shown how, due to legacy code, an LP pair can be permissionlessly minted and setup to cause loss to future depositors.

> Deployment of pairs is permissionless, however, the setup of Factory is Admin Dependent.

> While the Migrator file was out of scope, I believe the sponsor acknowledging, and the code being on the Pair file allows the finding to be valid.

> However, because it is ultimately contingent on setup, I think Medium Severity to be more appropriate.

[M-09] Incorrect condition always bound to fail

Submitted by codexploder

https://github.com/Plex-Engineer/lending-market/blob/755424c1f9ab3f9f0408443e6606f94e4f08a990/contracts/Governance/GovernorBravoDelegate.sol#L135

The state function check GovernorBravoDelegate.sol#L115 will always fail since proposalId cannot lie in between initialProposalId and proposalCount due to an initialization in `_initiate` function

Proof of Concept

1. The `_initiate` function sets initialProposalId = proposalCount;
2. Now lets say proposal count was 5 so initialProposalId and proposalCount are both set to 5
3. Now lets say state function is called on proposal id 2
4. The require condition checks proposalCount >= proposalId && proposalId > initialProposalId
5. This is equivalent to 5>=2 && 5>5, since 5>5 is not true this always fails even though proposal id 2 is correct

Recommended Mitigation Steps

Remove initialProposalId = proposalCount; in the `_initiate` function.

tkkwon1998 (Canto) disagreed with severity and commented:

> This is a bug, but will not lead to any attack or loss of funds. The initiate function will just fail, meaning the timelock admin cannot be set. This should be a 2 (Med Risk) issue.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has shown how, due to misconfiguration the Governor contract can be prevented from creating new proposals.

> Because this is contingent on setup, I think Medium Severity to be more appropriate.

[M-10] Oracle may be attacked if an attacker can pump the tokens for the entire block

Submitted by Chom

https://github.com/Plex-Engineer/stableswap/blob/489d010eb99a0885139b2d5ed5a2d826838cc5f9/contracts/BaseV1-core.sol#L190-L201
https://github.com/Plex-Engineer/stableswap/blob/489d010eb99a0885139b2d5ed5a2d826838cc5f9/contracts/BaseV1-core.sol#L154-L171

Attacker may use huge amount of their fund to pump the token in a liquidity pair for one entire block. The oracle will capture the manipulated price as current TWAP implementation may only cover 1 block if timed correctly. (First block on every periodSize = 1800 minutes)

This is possible and similar to Inverse finance April attack: https://www.coindesk.com/tech/2022/04/02/defi-lender-inverse-finance-exploited-for-156-million/

Proof of Concept

Assume currently is the first block after periodSize = 1800 minutes

```
function _update(uint balance0, uint balance1, uint _reserve
    uint blockTimestamp = block.timestamp;
    uint timeElapsed = blockTimestamp - blockTimestampLast;
    if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0)
        reserve0CumulativeLast += _reserve0 * timeElapsed;
        reserve1CumulativeLast += _reserve1 * timeElapsed;
    }

    Observation memory _point = lastObservation();
    timeElapsed = blockTimestamp - _point.timestamp; // comp
    if (timeElapsed > periodSize) {
        observations.push(Observation(blockTimestamp, reserv
    }
    reserve0 = balance0;
    reserve1 = balance1;
    blockTimestampLast = blockTimestamp;
    emit Sync(reserve0, reserve1);
}
```

timeElapsed > periodSize (Just greater as periodSize is just passed) -> add current cumulative reserve to observations list

Now, let pump the token. And use some technique that inverse finance hacker have used to hold that price for 1 block.

```
function current(address tokenIn, uint amountIn) external vi
    Observation memory _observation = lastObservation();
    (uint reserve0Cumulative, uint reserve1Cumulative,) = cu
    if (block.timestamp == _observation.timestamp) {
        _observation = observations[observations.length-2];
    }

    uint timeElapsed = block.timestamp - _observation.timest
    uint _reserve0 = (reserve0Cumulative - _observation.rese
    uint _reserve1 = (reserve1Cumulative - _observation.rese
    amountOut = _getAmountOut(amountIn, tokenIn, _reserve0,
}
```

1 Block passed

`reserve0Cumulative` or `reserve1Cumulative` may now be skyrocketed due to current token pumping.

timeElapsed = block.timestamp - `_observation.timestamp` is less than 20 seconds (= 1 block) since `_observation.timestamp` has just stamped in the previous block.

As timeElapsed is less than 20 seconds (= 1 block) this mean `_reserve0` and `_reserve1` are just a TWAP of less than 20 seconds (= 1 block) which is easily manipulated by Inverse finance pumping attack technique.

As `reserve0Cumulative` or `reserve1Cumulative` is skyrocketed in the 1 block timeframe that is being used in TWAP, `_reserve0` or `_reserve1` also skyrocketed.

As a conclusion, price oracle may be attacked if attacker can pump price for 1 block since TWAP just cover 1 block.

Recommended Mitigation Steps

You should calculate TWAP average of `_reserve0` and `_reserve1` in the `_update` function by using cumulative reserve difference from last update to now which has a duration of periodSize = 1800 minutes.

And when querying for current price you can just return `_reserve0` and `_reserve1`.

Refer to official uniswap v2 TWAP oracle example: https://github.com/Uniswap/v2-periphery/blob/master/contracts/examples/ExampleOracleSimple.sol

nivasan1 (Canto) acknowledged

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has shown how, the `current` pricing mechanism can be easily manipulated due to an excessively small observation window.

> Because the finding shows a property of the system, I believe it to be valid.

> However, the loss of funds is contingent on someone foolish enough to use that code for their pricing.

> Because of that, I believe Medium Severity to be more appropriate.

> To confirm: Do not use `current` for pricing an asset, you will get rekt.

[M-11] In `Cnote.sol`, anyone can initially become both accountant and admin

Submitted by p4st13r4, also found by 0x52 and Tutturu

Affected code:

- https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/CNote.sol#L14

The function `_setAccountantContract()` is supposed to be called after contract initialization, so that the `accountant` is immediately set. However, this function completely lacks any access control (it's just `public`) so an attacker can monitor the mempool and frontrun the transaction in order to become both `accountant` and `admin`

Tools Used

Editor

Recommended Mitigation Steps

The function should:

1. have a guard that regulates access control
2. not set the `admin` too, which is dangerous and out of scope

tkkwon1998 (Canto) confirmed

Alex the Entreprenerd (judge) commented:

> Frontrunnable Initializer without POC (is impact having to re-deploy?).

> Pretty confident will downgrade.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The warden has shown how, due to front-running, anyone can become the `accountant`. With the information I have, I think the worst case scenario is a re-deploy.

> Because the setter could have been written in a better way, but because the realistic consequence is a re-deploy, I think Medium Severity to be more appropriate.

[M-12] Note: When `_initialSupply ! = 0`, the `_mint_to_Accountant` function will fail

Submitted by cccz, also found by Picodes

In Note contract, if `_initialSupply ! = 0`, `_totalSupply` will overflow when the `_mint_to_Accountant` function executes `_mint(msg.sender, type(uint).max)`

```
        constructor(string memory name_, string memory symbol_, uint
            _name = name_;
            _symbol = symbol_;
                _initialSupply = totalSupply_;
                _totalSupply = totalSupply_;
        }
    ...
        function _mint(address account, uint256 amount) internal   {
            require(account != address(0), "ERC20: mint to the zero

            _beforeTokenTransfer(address(0), account, amount);

            _totalSupply += amount;
            _balances[account] += amount;
            emit Transfer(address(0), account, amount);
            _afterTokenTransfer(address(0), account, amount);
        }
```

Proof of Concept

https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/Note.sol#L13-L19 https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/ERC20.sol#L29-L34 https://github.com/Plex-Engineer/lending-market/blob/ab31a612be354e252d72faead63d86b844172761/contracts/ERC20.sol#L237-L247

Recommended Mitigation Steps

ERC20.sol

```
        constructor(string memory name_, string memory symbol_) publ
            _name = name_;
            _symbol = symbol_;
        }
```

note.sol

```
        constructor() ERC20("Note", "NOTE") {
            admin = msg.sender;
        }
```

nivasan1 (Canto) commented:

> Duplicate of Issue #53 (H-06)

Alex the Entreprenerd (judge) commented:

> In contrast to #53 -> Revert of `initialize`

> This finding shows how, based on `constructor` arguments, the function `_mint_to_accountant` can fail.

> Am not convinced on the impact as I believe in the worst case the Sponsor would just be forced to re-deploy

Alex the Entreprenerd (judge) commented:

> The warden has shown how, because the function `_mint_to_accountant` mints the maximum value representable, deploying Note with a non-zero `initialSupply` will cause a revert.

> Because this is contingent on a misconfiguration, I agree with Med Severity.

abhipingle (Canto) confirmed

Low Risk and Non-Critical Issues

For this contest, 45 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **joestakey** received the top score from the judge.

The following wardens also submitted reports: Dravee, robee, hake, oyc_109, 0xNazgul, 0xf15ers, zzzitron, Bronicle, 0x1f8b, TomJ, codexploder, hansfriese, Ruhum, csanuragjain, gzeon, hyh, Funen, TerrierLover, 0xDjango, sach1r0, Limbooo, 0x52, cccz, simon135, Picodes, asutorufos, catchup, _Adam, fatherOfBlocks, saian, Tutturu, 0x29A, 0xmint, MadWookie, technicallyty, nxrblsrpr, WatchPug, Waze, cryptphi, ignacio, JMukesh, c3phas, defsec, *and* k.

[L-01] assert statement should not be used

Properly functioning code should never reach a failing assert statement. If it happened, it would indicate the presence of a bug in the contract. A failing assert uses all the remaining gas, which can be financially painful for a user.

PROOF OF CONCEPT

Instances include:

lending-market/Comptroller.sol

```
l214 assert(assetIndex < len)
l360 assert(markets[cToken].accountMembership[borrower])
```

stableswap/BaseV1-periphery.sol

```
l82 assert(msg.sender == address(wcanto))
l227 assert(amountAOptimal <= amountADesired)
l273 assert(wcanto.transfer(pair, amountCANTO))
l419 assert(wcanto.transfer(pairFor(routes[0].from, routes[0].to
```

MITIGATION

Replace the assert statements with a require statement or a custom error

[L-02] CloseFactor unbounded

In `Comptroller.sol`, it is mentioned that `closeFactorMantissa` should be greater than `closeFactorMinMantissa` and less than `closeFactorMaxMantissa`. But in `_setCloseFactor`, these are not checked, meaning `closeFactorMantissa` can be set to a value outside the boundaries defined by the protocol.

PROOF OF CONCEPT

Instances include:

lending-market/Comptroller.sol

```
l81-l85
// closeFactorMantissa must be strictly greater than this value
uint internal constant closeFactorMinMantissa = 0.05e18; // 0.05

// closeFactorMantissa must not exceed this value
uint internal constant closeFactorMaxMantissa = 0.9e18; // 0.9
```

```
1850-1859
function _setCloseFactor(uint newCloseFactorMantissa) external r
    // Check caller is admin
    require(msg.sender == admin, "only admin can set close facto

    uint oldCloseFactorMantissa = closeFactorMantissa;
    closeFactorMantissa = newCloseFactorMantissa;
    emit NewCloseFactor(oldCloseFactorMantissa, closeFactorManti

    return uint(Error.NO_ERROR);
}
```

MITIGATION

Add checks in `_setCloseFactor` to ensure `closeFactorMantissa` is greater than `closeFactorMinMantissa` and less than `closeFactorMaxMantissa`.

[L-03] Immutable addresses lack zero-address check

Constructors should check the address written in an immutable address variable is not the zero address.

PROOF OF CONCEPT

Instances include:

stableswap/BaseV1-core.sol

```
l107 (token0, token1, stable) = (_token0, _token1, _stable)
```

stableswap/BaseV1-periphery.sol

```
l75factory = _factory;
pairCodeHash = IBaseV1Factory(_factory).pairCodeHash();
wcanto = IWCANTO(_wcanto);
```

MITIGATION

Add a zero address check for the immutable variables aforementioned.

[L-04] Receive function

`AccountantDelegate` has a `receive()` function, but does not have any withdrawal function. Any Manifest mistakenly sent to this contract would be locked.

PROOF OF CONCEPT

lending-market/AccountantDelegate.sol

```
receive() external override payable {}
```

MITIGATION

Add `require(0 == msg.value)` in `receive()` or remove the function altogether.

[L-05] Local variable shadowing

In `lending-market/NoteInterest.sol` , there is local variable shadowing: the constructor parameter has the same name as the storage variable `baseRatePerYear` . This will not lead to any error but can be confusing, especially in the constructor where `baseRatePerBlock` is computed using the constructor parameter `baseRatePerYear` .

PROOF OF CONCEPT

Instances include:

lending-market/NoteInterest.sol

```
constructor(uint baseRatePerYear) {
    baseRatePerBlock = baseRatePerYear.div(blocksPerYear)
```

MITIGATION

Add an underscore to the constructor parameter (`_baseRatePerYear`) to avoid shadowing.

[L-06] Avoid Using `.Transfer` to Transfer Native Tokens

In `WETH` and `TreasuryDelegate` , the `.transfer()` method is used to transfer Manifest.

The `transfer()` call requires that the recipient has a payable callback, only provides 2300 gas for its operation. This means the following cases can cause the transfer to fail:

- The contract does not have a payable callback
- The contract’s payable callback spends more than 2300 gas (which is only enough to emit something)
- The contract is called through a proxy which itself uses up the 2300 gas

Proof Of Concept

See this article.

The `.transfer` method is called in these places:

WETH.sol

```
131 payable(msg.sender).transfer(wad)
```

TreasuryDelegate.sol

```
152 to.transfer(amount)
```

Mitigation

Use `.call()` to send Manifest.

[N-01] Underflow desired but not possible

Underflow is desired in several price update functions of `stableswap/BaseV1Pair`, but as overflow/underflow checks are automatically performed since Solidity 0.8.0, the functions currently revert if there is underflow.

PROOF OF CONCEPT

Instances include:

stableswap/BaseV1-core.sol

```
l156 uint timeElapsed = blockTimestamp - blockTimestampLast; //
l183 uint timeElapsed = blockTimestamp - _blockTimestampLast
```

MITIGATION

Place these statements in an `unchecked` block to allow underflow

[N-02] Comment Missing function parameter

Some of the function comments are missing function parameters or returns

PROOF OF CONCEPT

Instances include:

lending-market/GovernorBravoDelegate.sol

```
l452 @param borrowerIndex
l526 @param seizeTokens
l677 @param accounts
l689 @param accounts
l826 @param newOracle
l1210 @param marketBorrowIndex
l1270 @param marketBorrowIndex
```

lending-market/CNote.sol

```
l31 @param borrower
```

lending-market/NoteInterest.sol

```
l92 @param cash
l92 @param borrows
l92 @param reserves
l109 @param cash
l109 @param borrows
l109 @param reserves
l109 @param reserveFactorMantissa
```

MITIGATION

Add a comment for these parameters.

[N-03] Constants instead of magic numbers

It is best practice to use constant variables rather than literal values to make the code easier to understand and maintain.

PROOF OF CONCEPT

Instances include:

lending-market/NoteInterest.sol

```
l95 100
l96 100
```

MITIGATION

Define constant variables for the literal values aforementioned.

[N-04] Constructor visibility

Visibility (public / external) is not needed for constructors anymore since Solidity 0.7.0, see here

PROOF OF CONCEPT

Instances include:

lending-market/AccountantDelegator.sol

```
l16 constructor(
                        address implementation_,
                        address admin_,
      address cnoteAddress_,
      address noteAddress_,
      address comptrollerAddress_,
      address treasury_) public
```

MITIGATION

Remove the `public` modifier from constructors.

[N-05] Events indexing

Events should use indexed fields

PROOF OF CONCEPT

Instances include:

lending-market/Comptroller.sol

```
l19 event MarketListed(CToken cToken)
l22 event MarketEntered(CToken cToken, address account)
l25 event MarketExited(CToken cToken, address account)
l28 event NewCloseFactor(uint oldCloseFactorMantissa, uint newCl
l31 event NewCollateralFactor(CToken cToken, uint oldCollateralF
l34 event NewLiquidationIncentive(uint oldLiquidationIncentiveMa
l37 event NewPriceOracle(PriceOracle oldPriceOracle, PriceOracle
l40 event NewPauseGuardian(address oldPauseGuardian, address new
l43 event ActionPaused(string action, bool pauseState)
l46 event ActionPaused(CToken cToken, string action, bool pauseS
l49 event CompBorrowSpeedUpdated(CToken indexed cToken, uint new
l52 event CompSupplySpeedUpdated(CToken indexed cToken, uint new
l55 event ContributorCompSpeedUpdated(address indexed contributo
l58 event DistributedSupplierComp(CToken indexed cToken, address
l61 event DistributedBorrowerComp(CToken indexed cToken, address
l64 event NewBorrowCap(CToken indexed cToken, uint newBorrowCap)
l67 event NewBorrowCapGuardian(address oldBorrowCapGuardian, add
l70 event CompGranted(address recipient, uint amount)
l73 event CompAccruedAdjusted(address indexed user, uint oldComp
l76 event CompReceivableUpdated(address indexed user, uint oldCo
```

lending-market/AccountantInterfaces.sol

```
l15 event AcctInit(address lendingMarketAddress)
l16 event AcctSupplied(uint amount, uint err)
l25 event NewImplementation(address oldImplementation, address n
```

lending-market/TreasuryInterfaces.sol

```
l17 event NewImplementation(address oldImplementation, address n
```

lending-market/CNote.sol

```
l10 event AccountantSet(address accountant, address accountantPr
```

lending-market/NoteInterest.sol

```
l17 event NewInterestParams(uint baserateperblock)
l61 event NewBaseRate(uint oldBaseRateMantissa, uint newBaseRate
l64 event NewAdjusterCoefficient(uint oldAdjusterCoefficient, ui
l67 event NewUpdateFrequency(uint oldUpdateFrequency, uint newUp
```

stableswap/BaseV1-core.sol

```
l88 event Mint(address indexed sender, uint amount0, uint amount
l89 event Burn(address indexed sender, uint amount0, uint amount
l90 event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
l98 event Sync(uint reserve0, uint reserve1);
l99 event Claim(address indexed sender, address indexed recipier
l101 event Transfer(address indexed from, address indexed to, ui
l102 event Approval(address indexed owner, address indexed spenc
l486 event PairCreated(address indexed token0, address indexed t
```

MITIGATION

Add indexed fields to these events so that they have the maximum number of indexed fields possible.

[N-06] Event should be emitted in setters

Setters should emit an event so that Dapps can detect important changes to storage

PROOF OF CONCEPT

Instances include:

lending-market/WETH.sol

```
l22 function deposit()
l28 function withdraw()
```

lending-market/GovernorBravoDelegate.sol

```
    l131 function _initiate()
```

stableswap/BaseV1-core.sol

```
    l497 function setPauser()
    l507 function setPause()
```

MITIGATION

Emit an event in all setters.

[N-07] Function missing comments

Some functions are missing Natspec comments.

PROOF OF CONCEPT

Instances include:

manifest/Proposal-Store.sol

```
    l46 function AddProposal
    l52 function QueryProp
```

lending-market/WETH.sol

All the functions are missing comments

lending-market/GovernorBravoDelegate.sol

```
    l77 function queueOrRevertInternal
    l180 function add256
    l186 function sub256
    l191 function getChainIdInternal()
```

lending-market/Comptroller.sol

```
l294 function redeemAllowedInternal
l180 function add256
l186 function sub256
l191 function getChainIdInternal()
l958 function _addMarketInternal()
l965 function _initializeMarket
l1050 function _setMintPaused
l1060 function _setBorrowPaused
l1070 function _setTransferPaused
l1079 function _setSeizePaused
l1088 function _become
l1094 function fixBadAccruals
l1144 function adminOrInitializing
l1461 function getBlockNumber
```

lending-market/CNote.sol

```
l14 function _setAccountantContract
l23 function getAccountant
```

stableswap/BaseV1-core.sol

All the functions are missing proper Natspec comments.

stableswap/BaseV1-periphery.sol

All the functions are missing proper Natspec comments.

MITIGATION

Add comments to these functions.

[N-08] Function order

Functions should be ordered following the [Soldiity conventions](): `receive()` function should be placed after the constructor and before every other function.

PROOF OF CONCEPT

Several contracts have `receive()` and `fallback()` at the end:

- lending-market/AccountantDelegate.sol
- lending-market/AccountantDelegator.sol
- lending-market/TreasuryDelegator.sol

MITIGATION

Place the `receive()` and `fallback()` functions after the constructor, before all the other functions.

[N-09] Non-library files should use fixed compiler versions

Contracts should be compiled using a fixed compiler version. Locking the pragma helps ensure that contracts do not accidentally get deployed using a different compiler version with which they have been tested the most.

PROOF OF CONCEPT

Instances include:

ZoneInteraction.sol

`WETH.sol`, `GovernorBravoDelegate.sol`, `Comptroller.sol`, `AccountantDelegate.sol`, `AccountantDelegator.sol`, `AccountantInterfaces.sol`, `TreasuryDelegate.sol`, `TreasuryDelegator.sol`, `TreasuryInterfaces.sol`, `CNote.sol` and `NoteInterest.sol` have floating pragmas.

MITIGATION

Used a fixed compiler version.

[N-10] Open TODOs

PROBLEM

There are open TODOs in the code. Code architecture, incentives, and error handling/reporting questions/issues should be resolved before deployment.

PROOF OF CONCEPT

Instances include:

lending-market/Comptroller.sol

```
l1232 // TODO: Don't distribute supplier COMP if the user is not
l1271 // TODO: Don't distribute supplier COMP if the user is not
```

MITIGATION

Remove the TODOs.

[N-11] Public functions can be external

It is good practice to mark functions as `external` instead of `public` if they are not called by the contract where they are defined.

PROOF OF CONCEPT

Instances include:

manifest/Proposal-Store.sol

```
l46 function AddProposal()
l52 function QueryProp()
```

lending-market/GovernorBravoDelegate.sol

```
l24 function initialize()
```

lending-market/Comptroller.sol

```
    l122 function enterMarkets()
    l677 function getAccountLiquidity()
    l703 function getHypotheticalAccountLiquidity()
    l826 function _setPriceOracle()
    l1033 function _setPauseGuardian()
    l1050 function _setMintPaused()
    l1060 function _setBorrowPaused()
    l1070 function _setTransferPaused()
    l1079 function _setSeizePaused()
    l1088 function _become()
    l1324 function claimComp(address holder)
    l1394 function _grantComp()
    l1407 function _setCompSpeeds()
    l1423 function _setContributorCompSpeed()
    l1444 function getAllMarkets()
```

lending-market/AccountantDelegate.sol

```
    l15 function initialize()
```

lending-market/AccountantDelegator.sol

```
    l109 delegateToViewImplementation()
```

lending-market/TreasuryDelegate.sol

```
    l15 function initialize()
```

lending-market/TreasuryDelegator.sol

```
    l84 delegateToViewImplementation()
```

lending-market/CNote.sol

```
l14 function _setAccountantContract
```

lending-market/NoteInterest.sol

```
l118 function updateBaseRate
```

MITIGATION

Declare these functions as `external` instead of `public`.

[N-12] Require statements should have descriptive strings

Some require statements are missing error strings, which makes it more difficult to debug when the function reverts.

PROOF OF CONCEPT

lending-market/WETH.sol

```
l69 require(_balanceOf[src] >= wad)
l72 require(_allowance[src][msg.sender] >= wad)
```

lending-market/GovernorBravoDelegate.sol

```
l53 require(proposals[unigovProposal.id].id == 0)
```

stableswap/BaseV1-core.sol

```
l125 require(_unlocked == 1)
l285 require(!BaseV1Factory(factory).isPaused());
l465 require(token.code.length > 0)
l468 require(success && (data.length == 0 || abi.decode(data, (b
l498 require(msg.sender == pauser)
```

```
l503 require(msg.sender == pendingPauser)
l508 require(msg.sender == pauser)
```

stableswap/BaseV1-periphery.sol

```
l210 require(amountADesired >= amountAMin);
l211 require(amountBDesired >= amountBMin)
l291 require(IBaseV1Pair(pair).transferFrom(msg.sender, pair, li
l456 require(token.code.length > 0)
l459 require(success && (data.length == 0 || abi.decode(data, (b
l463 require(token.code.length > 0, "token code length faialure'
l466 require(success && (data.length == 0 || abi.decode(data, (b
```

MITIGATION

Add error strings to all require statements.

[N-13] Scientific notation

For readability, it is best to use scientific notation (e.g `10e5`) rather than decimal literals(`100000`) or exponentiation(`10**5`).

PROOF OF CONCEPT

Instances include:

stableswap/BaseV1-periphery.sol

```
l67 uint internal constant MINIMUM_LIQUIDITY = 10**3
```

MITIGATION

Replace `10**3` with `10e3` .

[N-14] Styling

There should be space between operands in mathematical computations

PROOF OF CONCEPT

Instances include:

stableswap/BaseV1-periphery.sol

```
l134 routes.length+1
l139 amounts[i+1]
l366 routes[i+1].from, routes[i+1].to, routes[i+1].stable
```

MITIGATION

Add spaces, e.g

```
-routes.length+1
+routes.length + 1
```

[N-15] Typos

There are a few typos in the contracts.

PROOF OF CONCEPT

Instances include:

lending-market/NoteInterest.sol

```
l89 irrelevent
```

stableswap/BaseV1-periphery.sol

```
l463 faialure
```

MITIGATION

Correct the typos.

Gas Optimizations

For this contest, 39 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **_Adam** received the top score from the judge.

The following wardens also submitted reports: 0xNazgul, gzeon, 0xKitsune, saian, 0x1f8b, joestakey, Dravee, Limbooo, defsec, hansfriese, Waze, 0x29A, 0xf15ers, 0xkatana, c3phas, catchup, fatherOfBlocks, Funen, JC, oyc_109, rfa, robee, sach1r0, simon135, TerrierLover, Tomio, TomJ, ynnad, Ruhum, 0v3rf10w, 0xmint, ak1, Chom, Fitraldys, hake, k, MadWookie, *and* Picodes.

[G-01] Initialising to Default Values

When initialising a variable to its default variable, it is cheaper to leave blank. I ran a test in remix that initialises a single variable and got a saving of 2,246 gas.

```
contract Test {
    uint256 public variable = 0;     (69,312 gas)
    vs
    uint256 public variable;         (67,066 gas)
}
```

BaseV1-core.sol#L46 - can change to: uint public totalSupply;

[G-02] Emitting Storage Variables

You can save an SLOAD (~100 gas) by emiting local variables over storage variables when they have the same value.

BaseV1-core.sol#L170 - can emit balance0 & balance1 over reserve0 & reserve0 (save 2 SLOADS)
Comptroller.sol#L856 - can emit newCloseFactorMantissa instead of closeFactorMantissa
Comptroller.sol#L1045 - can emit newPauseGuardian instead of pauseGuardian
NoteInterest.sol#L127 - can emit newBaseRatePerYear instead of baseRatePerYear

NoteInterest.sol#L144 - can emit newBaseRateMantissa instead of baseRatePerYear
NoteInterest.sol#L144 - can emit newAdjusterCoefficient instead of adjusterCoefficient
NoteInterest.sol#L170 - can emit newUpdateFrequency instead of updateFrequency

[G-03] Variables Can be Immutable/Constant

The following variables are initialised either when created or in the constructor and then never modified and can be changed to immutable/constant to save gas.

Proposal-Store.sol#L35
BaseV1-core.sol#L39-L40
WETH.sol#L6-L8
NoteInterest.sol#L22

[G-04] For Loop Optimisations

When incrementing i in for loops there is no chance of overflow so unchecked can be used to save gas. I ran a simple test in remix and found deployment savings of 31,653 gas and on each function call saved ~141 gas per iteration.

```
contract Test {
        function loopTest() external {
                for (uint256 i; i < 1; ++i) {
                Deployment Cost: 125,637, Cost on function call:
                vs
                for (uint256 i; i < 1; ) {
                // for loop body
                unchecked { ++i }
                Deployment Cost: 93,984, Cost on function call:
                }
        }
}
```

In for loops pre increments can also be used to save a small amount of gas per iteraition.
I ran a test in remix using a for loop and found the deployment savings of 497 gas and ~5 gas per iteration.

```
contract Test {
        function loopTest() external {
                for (uint256 i; i < 1; i++) {
                (Deployment cost: 118,408, Cost on function call
                vs
                for (uint256 i; i < 1; ++i) {
                (Deployment cost: 117,911, Cost on function call
                }
        }
}
```

Looping over memory/storage variable lengths will cost ~3 gas/~100 gas per iteration, I recommend cacheing their value and looping over that instead.

Instances of for loops that can be optimised:
BaseV1-core.sol#L207
BaseV1-core.sol#L337
BaseV1-periphery.sol#L136
BaseV1-periphery.sol#L362
GovernorBravoDelegate.sol#L68
GovernorBravoDelegate.sol#L90
Comptroller.sol#L126
Comptroller.sol#L206
Comptroller.sol#L735
Comptroller.sol#L959
Comptroller.sol#L1005
Comptroller.sol#L1106
Comptroller.sol#L1347
Comptroller.sol#L1353
Comptroller.sol#L1359
Comptroller.sol#L1364
Comptroller.sol#L1413

[G-05] Minimising SLOAD’s

You can save 1 SLOAD (~97 gas) per use by cacheing a storage variable that is used more than once.

CNote.sol#L15-L18 - can cache `address(_ accountant)`

CNote.sol#L179-L180 - can cache `address(_ accountant)`

[G-06] No Need to check == True in Conditional Statements

You can save some gas (~1000 gas on deployment and ~10 gas on function call, based on remix test) by removing == True from the following locations.
Comptroller.sol#L149 Comptroller.sol#L1053 Comptroller.sol#L1063 Comptroller.sol#L1072 Comptroller.sol#L1081 Comptroller.sol#L1350 Comptroller.sol#L1357 Comptroller.sol#L1456

[G-07] Custom Errors

As your using a solidity version greater 0.8.4 you can replace revert strings with custom errors. This will save in deployment costs and runtime costs.
I ran a test in remix comparing a revert string vs custom errors and found that replacing a single revert string with a custom error saved 12,404 gas in deployment cost and 86 gas on each function call.

```
contract Test {
        uint256 a;
        function check() external {
                require(a != 0, "check failed");
        }
}   (Deployment cost: 114,703, Cost on Function call: 23,392)
vs
contract Test {
        uint256 a;
        error checkFailed();
        function check() external {
                if (a != 0) revert checkFailed();
        }
}   (Deployment cost: 102,299, Cost on Function call: 23,306)
```

I recommend replacing all revert strings with custom errors.

[G-08] Long Revert Strings

If opting not to update revert strings to custom errors, keeping revert strings <= 32 bytes in length will save gas.
I ran a test in remix and found the savings for a single short revert string vs long string to be 9,377 gas in deployment cost and 18 gas on function call.

```
contract Test {
        uint256 a;
        function check() external {
                require(a != 0, "short error message");
                (Deployment cost: 114,799, Cost on function call
                vs
                require(a != 0, "A longer Error Message over 32
        length");
                (Deployment cost: 124,176, Cost on function call
        }
}
```

I recommend shortenning the following revert strings to <= 32 bytes in length:

BaseV1-periphery.sol#L86
BaseV1-periphery.sol#L104-L105
BaseV1-periphery.sol#L223
BaseV1-periphery.sol#L228
BaseV1-periphery.sol#L295-L296
BaseV1-periphery.sol#L387
BaseV1-periphery.sol#L402
BaseV1-periphery.sol#L417
BaseV1-periphery.sol#L430
BaseV1-periphery.sol#L452
WETH.sol#L29
WETH.sol#L96-L97
GovernorBravoDelegate.sol#L25-L27
GovernorBravoDelegate.sol#L42-L47
GovernorBravoDelegate.sol#L78
GovernorBravoDelegate.sol#L87
GovernorBravoDelegate.sol#L115
GovernorBravoDelegate.sol#L132-L133
GovernorBravoDelegate.sol#L146
GovernorBravoDelegate.sol#L164

Comptroller.sol#L178

Comptroller.sol#L491

Comptroller.sol#L998

Comptroller.sol#L1016

Comptroller.sol#L1051-L1052

Comptroller.sol#L1061-L1062

Comptroller.sol#L1071

Comptroller.sol#L1080

Comptroller.sol#L1089

Comptroller.sol#L1095-L1096

Comptroller.sol#L1411

AccountantDelegator.sol#L43-L44

AccountantDelegator.sol#L124

TreasuryDelegator.sol#L31-L32

TreasuryDelegate.sol#L47

CNote.sol#L16

CNote.sol#L43

CNote.sol#L45

CNote.sol#L54

CNote.sol#L77

CNote.sol#L114

CNote.sol#L130

CNote.sol#L146

CNote.sol#L198

CNote.sol#L214

CNote.sol#L264

CNote.sol#L310

CNote.sol#L330

NoteInterest.sol#L141

NoteInterest.sol#L154

NoteInterest.sol#L167

[G-09] Uint > 0 Checks in Require Functions

If opting not to use custom errors when checking whether a uint is > 0 in a requrie functions you can save a small amount of gas by replacing with != 0. This is only true if optimiser is turned on and in a require statement. I ran a test in remix with optimisation set to 10,000 and found the savings for a single occurance is 632 in deployment cost and 6 gas on each function call.

```
    contract Test {
            uint256 a;
            function check() external {
                    require(a > 0);
                    (Deployment cost: 79,763, Cost on function call:
                    vs
                    require(a != 0);
                    (Deployment cost: 79,331, Cost on function call:
            }
    }
```

Instances where a uint is compared > 0:
BaseV1-core.sol#L253
BaseV1-core.sol#L272
BaseV1-core.sol#L286
BaseV1-core.sol#L303
BaseV1-core.sol#L465
BaseV1-periphery.sol#L104-L105
BaseV1-periphery.sol#L456
BaseV1-periphery.sol#L463

[G-10] && in Require Functions

If not opting to use custom errors and If optimising for running costs over deployment costs you can seperate && in require functions into 2 parts. I ran a basic test in remix and it cost an extra 234 gas to deploy but will save ~9 gas everytime the require function is called.

```
    contract Test {
            uint256 a = 0;
            uint256 b = 1;

            function test() external {
                    require(a == 0 && b > a)
                    (Deployment cost: 123,291, Cost on function call
                    vs
                    require(a == 0);
                    require(b > a);
                    (Deployment cost: 123,525, Cost on function call
            }
    }
```

Instances where require statements can be split into seperate statements:

BaseV1-core.sol#L272
BaseV1-core.sol#L288
BaseV1-core.sol#L294
BaseV1-core.sol#L431
BaseV1-core.sol#L468
BaseV1-periphery.sol#L105
BaseV1-periphery.sol#L459
BaseV1-periphery.sol#L466
GovernorBravoDelegate.sol#L42-L45
GovernorBravoDelegate.sol#L115
GovernorBravoDelegate.sol#L164
Comptroller.sol#L1003
Comptroller.sol#L1411

Alex the Entreprenerd (judge) commented:

> Over 10k between immutables and stuff

> **[G-01] Initialising to Default Values**
> Valid but I only count runtime gas vs deployment

> **[G-02] Emitting Storage Variables**
> 8 * 97 (3 gas for the MLOAD)

> 776

> **[G-03] Variables Can be Immutable/Constant**
> 7 * 2100
> 14700

> **[G-04] For Loop Optimisations**
> I think the benchmark is correct but may be unfairly skewed against the non-optimized (perhaps no optimizer)
> Will give 25 points per instance

> 17 * 25
> 425

> **[G-05] Minimising SLOAD’s**
> 94 each (6 gas for setup of cache)
> 188

> **[G-06] No Need to check == True in Conditional Statements**
> 6 gas per instance (3 for check, 3 for MLOAD of the hardcoded value)
> 8 * 6
> 48

> **[G-07] Custom Errors**
> Because you benchmarked it, I’ll keep that in mind in scoring

> **[G-08] Long Revert Strings**
> 6 per instance
> 51 * 6
> 301

> **[G-09] Uint > 0 Checks in Require Functions**
> 6 per instance
> 8 * 6
> 48

> **[G-10] && in Require Functions**
> 9 per instance (because you benchmarked, would have given 3 normally)
> 13 * 9
> 117

> **Total Gas Saved**
> **16603**

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.

Top