Quantstamp Security Assessment Certificate

Dec 20th 2021 — Quantstamp Verified

Cbdc Finance

This smart contract audit was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Token Lending Aggregator
Auditors	Ed Zulkoski, Senior Security Engineer Kacper Bąk, Senior Research Engineer Poming Lee, Research Engineer Sebastian Banescu, Senior Research Engineer
Timeline	2019-12-02 through 2021-04-23
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
cbdc-contracts	937f989 (initial audit)
cbdc-contracts	b5fb299 (latest audit)

Goals

- Do functions have proper access control logic?
- Are there centralized components of the system which users should be aware?
- Do the contracts adhere to best practices?

Total Issues	**39** (25 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**4** (4 Resolved)
Low Risk Issues	**11** (9 Resolved)
Informational Risk Issues	**18** (8 Resolved)
Undetermined Risk Issues	**6** (4 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

The Cbdc contracts are generally well documented and well designed. Our main concerns below relate to centralized components of the system, and ensuring that users are aware of the roles and responsibilities of the Cbdc Finance team as owners of the smart contracts. We also noted some potential access control issues associated with rebalancing, which may lead to sub-optimal token allocations.
Update: Cbdc Finance has addressed our concerns as of commit bcb6f09.
Update 2: Recently, several attacks have occurred on bZx/Fulcrum (for reference, see Attack 1 and Attack 2), allowing lenders to create highly under-collateralized loans. Since Fulcrum is one of the underlying protocols that Cbdc may lend on, we recommend investigating these attacks to determine how much impact this may have on the Cbdc protocol. It may be prudent to temporarily disable Fulcrum as a potential lending platform until the full extent of the issues has been investigated. As a simple approach, we believe this could be accomplished in the
following manner:

1. Deploy a new "dummy" wrapper contract that returns zero whenever `nextSupplyRate()`or `nextSupplyRateWithParams()` are invoked. This essentially ensures that the rebalancer will always favor other wrappers when calculating the allocations.
2. As the owner, invoke `CbdcToken.setProtocolWrapper("fulcrum address", "dummy wrapper address")`.

Note that we also recommend adding additional tests to ensure that supply rates equal to zero do not cause any adverse affects.
Update 3: We have reviewed version 3 of the contracts based on commit a71a706. Our audit focused on the new wrapper contracts associated with `Aave` and `DyDx`, and the new `CbdcTokenV3` and `CbdcRebalancerV3`. We noted several new sources of centralization, parts of the code which required further documentation, and possible gas-constant related issues. We recommend addressing these concerns before deploying the V3 contracts to production.
Update 4: Several of our concerns have been addressed as of commit 64f22d0.
Update 5: Our concerns have been addressed as of commit fefd01d.
Update 6: All concerns have been addressed as of commit 7d3b7e4.
Update 7: Quantstamp has reviewed updates to the contracts as of commit 93d3429.
Update 8: Quantstamp has reviewed updates as of commit f9c02d1.
Update 9: Quantstamp has reviewed updates as of commit 35d61ae. In this iteration, only `TokenV3_1.sol`, `CbdcRebalancerV3_1.sol`, and `CbdcCompound.sol` were audited (against the previously audited "V3" versions). New findings can be found in QSP-14 through QSP-20, and have been appended to the Best Practices and Documentation sections.
Update 10: Quantstamp has reviewed updates as of commit 338ec24. All existing issues have been resolved. However, there are several contracts such as `GSTConsumer*.sol`, `CbdcDSR.sol`, and `CbdcDyDx.sol` which we suggest improving coverage for.
Update 11: The Cbdc team has alerted Quantstamp of an issue in `CbdcTokenV3_1._tokenPrice()`, in which the incorrect number of decimal places had been used. This issue has been resolved, and no new issues were found as of commit 1b40261
Update 12: Several new issues of varying severity were noted during the audit of commit 50da42b9, as discussed in QSP-21 through QSP-31, and as appended to the best practices and documentation sections. Note that only `CbdcTokenV3_1.sol` was reviewed in this iteration.
Update 13: All issues have been addressed as of commit bd40915.
Update 14: The report has been updated based on the diff b928e84...e09d4f5. This iteration is only scoped to changes in `CbdcTokenGovernance.sol` and `CbdcTokenHelper.sol`. New findings are listed in QSP-32 through QSP-41, as well as appended to the best practices and documentation sections.
Update 15: The report has been updated based on commit b5fb299. All previous issues have been resolved, mitigated, or acknowledged, and one new informational issue was added. Some acknowledged issues are not fully fixed due to contract bytecode size limits; we recommend refactoring the code into several contracts to avoid this problem.

ID	Description	Severity	Status
QSP-1	Centralization of Power	^ Medium	Fixed
QSP-2	Missing `onlyCbdc` modifier on `mint()` and `redeem()`	ˇ Low	Fixed
QSP-3	Gas Usage / `for` Loop Concerns	○ Informational	Fixed
QSP-4	Clone-and-Own	○ Informational	Fixed
QSP-5	Unlocked Pragma	○ Informational	Fixed
QSP-6	Undocumented magic constants	○ Informational	Fixed
QSP-7	Use of ABIEncoderV2 still experimental	○ Informational	Fixed
QSP-8	Unchecked constructor and setter address arguments	○ Informational	Fixed
QSP-9	Allowance Double-Spend Exploit	○ Informational	Acknowledged
QSP-10	Function `rebalance()` may be blocked due to Fulcrum failure	○ Informational	Fixed
QSP-11	Security of Cbdc contracts is dependent on underlying lending protocols	○ Informational	Acknowledged
QSP-12	`newCbdcToken()` may overwrite `underlyingToCbdcTokenMap[_token]`	? Undetermined	Fixed
QSP-13	Gas constants may be affected by new EVM forks	? Undetermined	Fixed
QSP-14	`redeemCbdcToken()` may fail if `fee` is reset to zero	^ Medium	Fixed
QSP-15	Loss of precision due to truncation	ˇ Low	Fixed
QSP-16	Missing address sanitization	ˇ Low	Acknowledged
QSP-17	Length of input arrays can be different	ˇ Low	Fixed
QSP-18	Unclear update to `userAvgPrices` mapping	ˇ Low	Fixed
QSP-19	Potential flash loans attack vectors to claim COMP tokens	ˇ Low	Fixed
QSP-20	Privileged Roles and Ownership	○ Informational	Acknowledged
QSP-21	User may not be able to redeem Cbdc tokens	^ Medium	Fixed
QSP-22	Outdated `govToken` could be used to influence the average APR	ˇ Low	Fixed
QSP-23	Incorrect hardcoded addresses	ˇ Low	Acknowledged
QSP-24	Inconsistent array lengths breaks invariants	ˇ Low	Fixed
QSP-25	Initialization can be done multiple times	○ Informational	Acknowledged
QSP-26	Missing input check	○ Informational	Acknowledged
QSP-27	Missing return value	○ Informational	Acknowledged
QSP-28	Privileged roles	○ Informational	Acknowledged
QSP-29	Incorrect average price computation	? Undetermined	Fixed
QSP-30	Uninitialized inherited contracts and state variables	? Undetermined	Acknowledged
QSP-31	Unclear functionality in `_getFee`	? Undetermined	Fixed
QSP-32	Wrong comparison between lengths	^ Medium	Mitigated
QSP-33	The `flashLoanFee` is not settable	ˇ Low	Fixed
QSP-34	Inconsistent array lengths breaks invariant	ˇ Low	Mitigated

ID	Description	Severity	Status
QSP-35	Flashloans may decrease funds if underlying protocols have redemption fees	○ Informational	Acknowledged
QSP-36	Unchecked function arguments	○ Informational	Acknowledged
QSP-37	Flashloan could be used as a tool to manipulate liquidities of the underlying lending protocols	○ Informational	Acknowledged
QSP-38	Uninitialized state variables	? Undetermined	Acknowledged
QSP-39	Owner can front-run flash loaners to change loan fee	○ Informational	Mitigated

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Truffle v4.1.12
- SolidityCoverage v0.5.8
- Mythril v0.22.8
- Slither v0.6.12

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
3. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
4. Installed the Mythril tool from Pypi: `pip3 install mythril`
5. Ran the Mythril tool on each contract: `myth a path/to/contract`
6. Installed the Slither tool: `pip install slither-analyzer`
7. Run Slither from the project directory: `slither .`s

Findings

QSP-1 Centralization of Power

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `CbdcFulcrum.sol`, `CbdcRebalancer.sol`, `CbdcCompound.sol`, `CbdcTokenV3.sol`, `CbdcRebalancerV3.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.
In several contracts, the associated tokens may be changed by the owner. If the balances of the contracts are non-zero, users may not be able to retrieve funds or interact with the contract in a proper manner. In particular:

- In `CbdcFulcrum` and `CbdcCompound`, tokens may be updated by `setToken()` and `setUnderlying()`.
- In `CbdcRebalancer.sol`, `setCbdcToken()`, `setCToken()`, `setIToken()`, `setCTokenWrapper()`, and `setITokenWrapper()` may update underlying addresses.
- In `CbdcTokenV3` and `CbdcRebalancerV3.sol`, the owner may add new token wrappers arbitrarily (which may not correspond to actual lending protocols)

.Additionally, the owner may pause/unpause certain functionalities, such as rebalancing.

Recommendation: Limit the amount of centralized components in the system if possible. For example, if the underlying token is unlikely to change, consider setting it upon contract construction and removing the corresponding `setUnderlying()` function. Additionally, this centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.
Update: Cbdc Finance has removed the corresponding setter functions. The `pausing` centralization is mitigated as users may still redeem funds while the contract is paused. The centralization around adding new wrappers is mitigated through the use of a delay-scheme, such that new wrappers only go into effect after several days.

QSP-2 Missing `onlyCbdc` modifier on `mint()` and `redeem()`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CbdcCompoundV2.sol`

Description: For the functions `CbdcCompoundV2.mint()` and `CbdcCompoundV2.redeem()`, there is no `onlyCbdc` modifier, whereas the modifier exists in the corresponding functions in `CbdcCompound.sol`, `CbdcFulcrum.sol`, and `CbdcFulcrumV2.sol`. This would allow funds stored in the `CbdcCompoundV2` wrapper contract to be sent to an arbitrary address.
Although the typical dApp workflow does not store funds directly in the wrapper contract (in favor of storing balances in `CbdcToken`, users interacting directly with the `CbdcCompoundV2` wrapper
contract may mistakenly add funds to the contract directly. Adding the `onlyCbdc` modifier to these functions would mitigate these incorrect interactions.

Recommendation: Add the `onlyCbdc` modifier to `CbdcCompoundV2.mint()` and `CbdcCompoundV2.redeem()`.

QSP-3 Gas Usage / `for` Loop Concerns

Severity: *Informational*

Status: Fixed

File(s) affected: `CbdcRebalancer.sol`, `CbdcToken.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible.
In particular, the rebalancing functions may require several loops in the bisection algorithm.

Recommendation: We recommend performing gas analysis to ensure that each loop-function will not run into gas limitations, particularly for large inputs.
Update: Cbdc Finance has indicated that each iteration of the bisection algorithm consumes approximately 12,500 gas, so the limit of `maxIterations = 30` (as defined in the constructor) should be sufficient to avoid gas limits.

QSP-4 Clone-and-Own

Severity: *Informational*

Status: Fixed

File(s) affected: `CbdcMcdBridge.sol`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.
In `CbdcMcdBridge.sol`, there are several libraries that could be imported: `IERC20`, `SafeMath`, `Context`, and `Address`.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.

QSP-5 Unlocked Pragma

Severity: *Informational*

Status: Fixed

File(s) affected: `CbdcMcdBridge.sol`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked."
The file `CbdcMcdBridge.sol` has several instances of unlocked pragmas throughout.

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

QSP-6 Undocumented magic constants

Severity: *Informational*

Status: Fixed

File(s) affected: **Cbdc**`.sol`, `GST2Consumer.sol`

Description: There are several defined constants in the code that were unclear, and would benefit from added inline documentation:

- In Cbdc`.sol`, L161: the number 29;
- In Cbdc`.sol`, the constant on L143 of `getApr()`: `100/10^9`;
- In `GST2Consumer.sol`, all numerical constants on L15, 19-20;
- In Cbdc`RebalancerV3.sol`, on L32, it is not immediately clear that the constant 100000 is 100%.

Recommendation: Add documentation describing these constants.

QSP-7 Use of ABIEncoderV2 still experimental

Severity: *Informational*

Status: Fixed

File(s) affected: `yxToken.sol`

Description: Until solidity 0.6.0, the ABIEncoderV2 feature is still technically in experimental state. Although there are no known security risks associated with it, these features should be used judiciously.

Recommendation: Upgrade the contracts to a more recent solidity version such as `0.5.16` or `0.6.6`. All contracts that depend upon ABIEncoderV2 functionality should be tested thoroughly.

QSP-8 Unchecked constructor and setter address arguments

Severity: *Informational*

Status: Fixed

File(s) affected: **Cbdc**`RebalancerV3.sol`

Description: * In CbdcRebalancerV3`.sol`, on L28, the constructor arguments `_yxToken` and `_rebalancerManager` were not checked to be non-zero

· • In CbdcTokenV3`.sol`, the constructor and all setter functions should check that addresses are non-zero.

Recommendation: Add require statement ensuring that these parameters are non-zero.

QSP-9 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Acknowledged

File(s) affected: **Cbdc**`TokenV3.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario: An example of an exploit goes as follows:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance` and `decreaseAllowance`.
Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.

QSP-10 Function `rebalance()` may be blocked due to Fulcrum failure

Severity: *Informational*

Status: Fixed

File(s) affected: **Cbdc**`TokenV3.sol`

Description: On `L508` of CbdcTokenV3`.sol`, the modifier `whenITokenPriceHasNotDecreased` checks that function `_rebalance` can only be executed when the iToken price has not decreased. However, since `Fulcrum` could get hacked (or the price of collateral may drop), it might not always be true. When this happens, the system would not be able to rebalance/reallocate funds for a period of time.

Recommendation: There is a trade-off here -- including the modifier may cause delays in rebalancing, whereas removing it may cause adverse token allocations to Fulcrum. Documentation should be added describing the need for the modifier if it remains.

QSP-11 Security of Cbdc contracts is dependent on underlying lending protocols

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CbdcTokenV3.sol`, `CbdcRebalancerV3.sol`

Description: Although there is no immediate exploit known at this time, since protocol wrappers can be added arbitrarily in the future, this issue could occur, and further unforeseen issues could arise in the existing underlying protocols.

Exploit Scenario: If a wrapped protocol `P` is attackable, possibly through (but not limited to) flash loans, the following could occur. Suppose initially all funds are allocated to a secure protocol `S`.

1. Using a flash loan, the attacker creates a favorable price for `P` and invokes `rebalance()`. This causes the distribution to shift all underlying tokens to `P`.
2. The attacker attacks `P`, which now has significantly more liquidity since all Cbdc funds are now allocated to it.

Recommendation: This issue is partially mitigated already for Fulcrum through checks on the `iToken` price, and further through the ability to pause rebalancing. New wrappers should be added cautiously.

QSP-12 `newCbdcToken()` may overwrite `underlyingToCbdcTokenMap[_token]`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `CbdcFactory.sol`

Description: If `newCbdcToken()` is called with an existing `_token` address, the `CbdcToken` contract referenced in the `underlyingToCbdcTokenMap` will be overwritten. It is not clear if this is intended functionality.

Recommendation: Document whether this is intended functionality. If not, prevent `newCbdcToken()` calls with existing `_token` addresses.
Update: Cbdc Finance has addressed this concern through added documentation.

QSP-13 Gas constants may be affected by new EVM forks

Severity: *Undetermined*

Status: Fixed

File(s) affected: `GST2Consumer.sol`

Description: In `GST2Consumer.sol`, several constants are defined related to gas usage. Since op-code gas costs may be updated in new forks, this may cause unforeseen gas issues in future forks.

Recommendation: Ensure that this functionality has been tested on the most recent EVM fork. In order to be resilient to future forks, `onlyOwner` setter functions could be added to update the gas variables.
Update: this has been fixed through the use of an `onlyOwner` setter function for the gas variables.

QSP-14 `redeemCbdcToken()` may fail if `fee` is reset to zero

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `CbdcTokenV3_1.sol`

Description: Assume that:
A1: `userNoFeeQty[msg.sender]` can only accumulated when `fee` is set to `0` (according to the `_updateAvgPrice()` function).
A2: the price of CbdcToken is `5` and does not change a lot (this happens when the `balanceUnderlying` is large).
Consider the following scenario for some `user1`:

1. `user1` deposits `100` underlying token when `fee` is set to `0`. The `user1` will obtain `100/5 = 20` CbdcToken, and we noted that `userNoFeeQty[user1]` equals to `20`
2. Then the CbdcFinance team decides to change the `fee` from `0` to `1000`.
3. When the `user1` later deposit again, with another `100` underlaying token, the `user1` will obtain `100/5 = 20` Token again. In addition to the formerly obtained `20` Token, now the `user1` has `20 + 20 = 40` Tokens on hand. However, since `fee != 0` now, the `userNoFeeQty[user1]` will remains equal to `20` instead of equal to `20 + 20 = 40`.
4. Then the CbdcFinance team decides to change the `fee` from `1000` to `0` again.
5. Finally, when `user1` decides to redeem CbdcTokens through function `redeemToken()` by passing the parameter `_amount = 40`, we have that the `_amount` is `40` but the `userNoFeeQty[user1]` is `20`. This will cause the revert of the function due to the statement: `userNoFeeQty[msg.sender] = userNoFeeQty[msg.sender].sub(_amount);`.

Recommendation: Revise the `userNoFeeQty` functionality to account for this scenario.

QSP-15 Loss of precision due to truncation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CbdcTokenV3_1.sol`

Description: The computation of the average APR inside the `getAvgAPR()` function, is performed by normalizing (dividing by `total`) the APR for each token separately and adding the normalized values together. Due to the limited precision and truncation of the division operation, there might be a loss of precision in this computation.
Similarly the division by `10**18` can be moved outside of the for-loop in the `_getCurrentPoolValue` function.

Recommendation: To increase the precision of the average APR (and save gas), one could first add all APRs multiplied by the amounts together and only divide by the `total` at the end of the for-loop like so:

```
for (uint256 i = 0; i < allAvailableTokens.length; i++) {
    if (amounts[i] == 0) {
      continue;
    }
    avgApr = avgApr.add(
      ILendingProtocol(protocolWrappers[allAvailableTokens[i]]).getAPR().mul(amounts[i]);
    );
  }
  avgApr = avgApr.div(total);
```

QSP-16 Missing address sanitization

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `CbdcTokenV3_1.sol`

Description: The values inside the `_newGovTokens` array input parameter are not checked to be different from `0x0` inside the `setGovTokens` function.

Recommendation: Add `require` statement that checks that the value of the `_newGovTokens` is different from `0x0`.
Update: This has been acknowledged, however the check has not been added due to contract bytesize limitations.

QSP-17 Length of input arrays can be different

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CbdcTokenV3_1.sol`

Description: There are multiple occurrences of this issue:

1. There is no check in place inside the `redeemAllNeeded` function inside `CbdcTokenV3_1`, which checks if the length of the `tokenAddresses`, `amounts` and the `newAmounts` input arrays are equal. Since the for-loop inside this function goes up to `amounts.length` it would be problematic if the lengths of the other arrays would be different (shorter or longer).
2. There is no check in place inside the `_mintWithAmounts` function inside `CbdcTokenV3_1`, which checks if the length of the `tokenAddresses` and the `protocolAmounts` input arrays are equal. Since the for-loop inside this function goes up to `protocolAmounts.length` it would be problematic if the lengths of the other array would be different (shorter or longer).
3. There is no check in place inside the `setAllAvailableTokensAndWrappers` function inside `CbdcTokenV3_1`, which checks if the length of the `protocolTokens` and the `allAvailableTokens` arrays have the same length. This could lead to removing or adding tokens and/or changing the order of the tokens w.r.t. the `lastAllocations` array order.

Recommendation: Check whether the lengths of input array parameters of functions are the same whenever this is a prerequisite.
Update: Regarding `_redeemAllNeeded`, those params come from `_getCurrentAllocations` which reads current contract data so it should not be a problem.

QSP-18 Unclear update to `userAvgPrices` mapping

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CbdcTokenV3_1.sol`

Description: In the function `_updateAvgPrice`, the mapping `userAvgPrices` is not updated if the `fee == 0`. It is not clear why the mapping is not updated in this case, but since this case is not covered, the user's average price may not be correct in all scenarios.

Recommendation: Either update the function to update the average price in all branches, or consider renaming the mapping.

QSP-19 Potential flash loans attack vectors to claim COMP tokens

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CbdcTokenV3_1.sol`

Description: After discussion with the Cbdc team, it appears that there may exist attack vectors that claim COMP tokens using flash loans, if a rebalance or redeem has not been invoked in a long time. This attack could occur if mint and redeem are invoked with a large balance in the same transaction (via a flash loan).

Recommendation: Add a lock variable that prevents a user from invoking mint and redeem functions within the same transaction.

QSP-20 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CbdcRebalancerV3_1.sol`, `CbdcTokenV3_1.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.

Within CbdcRebalancerV3_1, the owner can perform the following actions:

1. Can set the Cbdc token exactly once via `setCbdcToken`
2. Can set the rebalance manager address any number of times via `setRebalanceManager`
3. Can add any number of new tokens via `setNewToken`
4. Another role enforced by `onlyRebalancerAndC b d c` modifier, which allows the rebalance manager or Cbdc token to set completely new token allocations, for exactly the same token addresses, that sum up to 100% (any number of times).

The CbdcTokenV3_1.sol contract contains the following privileged actions:

1. Modify the `allAvailableTokens` array any number of times
2. Set the address of the `iToken` any number of times
3. Set the governance token address `govTokens` any number of times
4. Set the rebalancer address any number of times
5. Set the fee taken from end users any number of times to any value lower or equal to 10%
6. Set the maximum unlent asset percentage to any value lower than 100%
7. Set the fee address any number of times.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.
Update: Updated documentation will be provided as in here.

QSP-21 User may not be able to redeem Cbdc tokens

Severity: *Medium Risk*

Status: Fixed

File(s) affected: CbdcTokenV3_1.sol

Description: If the `_tokenPrice()` is lower than the `userAvgPrices` for that user, then the `sub` method call on L911 in `_getFee` will throw an error and revert the transaction. Given that the `_getFee` function is only called in `redeemCbdcToken` it will lead to users not being able to redeem Cbdctokens as long as the current price is lower than the `userAvgPrices` for that user.

Recommendation: If `currPrice < userAvgPrices[msg.sender]` then set the `elegibleGains` to zero in `_getFee`.

QSP-22 Outdated `govToken` could be used to influence the average APR

Severity: *Low Risk*

Status: Fixed

File(s) affected: CbdcTokenV3_1.sol

Description: The following condition in `_getAvgAPR`, on L358: `if (govTokens.length > 0 && currGov != address(0))` only checks if the length of `govTokens` is greater than zero. However, it does not check if the length of the `govTokens` is greater than `i` (the loop iterator) or if the `currGov` is in the `govTokens` array. Due to the way in which the `setGovTokens` function works, it may be the case that `currGov != address(0)` but `currGov` is not included in the `govTokens` array. This could have very severe consequences because any user is allowed to call `openRebalance`, which changes the allocations based on the results obtained from calling `_getAvgAPR`. The `_getAvgAPR` function would return the wrong results, because it would take into consideration removed `govTokens`.

Exploit Scenario:

1. Owner decides to call `setGovTokens` in order to remove some `govTokens` which are no longer valid (e.g. the projects corresponding to those `gotTokens` were hacked). Note that the `setGovTokens` method does not set the `protocolTokenToGov` entries for those removed tokens to `address(0)`.
2. Malicious party calls `openRebalance` and allocates a large portion of funds to a token that has a corresponding `govToken` that was removed in step 1. The malicious party knows that the price oracle will return a large APR for that `govToken`, which will skew the result of `_getAvgAPR`.

Recommendation: Set the `protocolTokenToGov` entries for the removed tokens to `address(0)` inside the `setGovTokens` method.

QSP-23 Incorrect hardcoded addresses

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: CbdcTokenV3_1.sol

Description: 1. The address of the Cbdc governance token is hardcoded to `0x0001` on

485.The address of the `oracle` is hardcoded to `0x0001` on L111.

2. The address of the `CbdcController` is hardcoded to `0x0001` on L112.
3. The following address seems to be an EOA, not a smart contract L131: `rebalancer = address(0xB3C8e5534F0063545CBbb7Ce86854Bf42dB8872B);`
4. The address of the `iToken` is hardcoded to `address(0)` on L130 and there is no setter function to change the `iToken` address.

Recommendation: Update the values and remove TODO comments. Clarify why `C b d c` needs to be a hardcoded constant, instead of being updated via a setter/initialization function similar to `oracle` and `CbdcController`. Also why not allow these addresses to be passed as input parameters to the `manualInitialize` function instead of hardcoding them?
Update from the CbdcFinance team: All addresses will be se once the governance is deployed. The rebalancer address is an EOA now because we removed the need for `CbdcRebalancerV3_1` by moving the functionalities directly in `CbdcTokenV3_1`. The address set is the rebalancer address that was previously had in `CbdcRebalancerV3_1` (before was just a proxy basically). The `iToken` address is hardcoded to `address(0)` correctly because we don't support Fulcrum anymore and we don't use that variable anymore. Cbdc address should not be upgradable once set,while `PriceOracle` and `CbdcController` addresses can change (The `CbdcController` is an upgradable contract actually so the address will be the same; we removed the `setCbdcControllerAddress` method too.) Those addresses were not passed in the `manualInitialize` because we are at the very limit of the max bytecode size so any addition change needs to get some 'space' somewhere else. We removed also the `setMaxUnlentPerc` method, which will be reintroduced later.

QSP-24 Inconsistent array lengths breaks invariants

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CbdcTokenV3_1.sol`

Description: The length of the `allAvailableTokens` array and the `lastRebalancerAllocations` and `lastAllocations` arrays may diverge after calling `setAllAvailableTokensAndWrappers`, even if they were the same length after `manualInitialize`. This is because the allocations are not adjusted or checked to be of the same length with the `protocolTokens` or `wrappers` input arrays. This means that the owner can remove tokens from the `allAvailableTokens` array and the sum of all corresponding allocations would not be 100% after that call.

Exploit Scenario:

1. Owner (accidentally) removes 1 or more tokens by calling `setAllAvailableTokensAndWrappers`
2. Either the owner forgets to call `setAllocations` OR they call `setAllocations`, but are front-run by an end-user that calls `openRebalance` or `rebalance`.

Recommendation: Either add a check inside `setAllAvailableTokensAndWrappers` which does not let the owner remove tokens OR add another input array to `setAllAvailableTokensAndWrappers` which indicates the new allocations. Optionally, a Boolean input parameter could also be added to `setAllAvailableTokensAndWrappers` which indicates that the allocation should stay the same, in which case a `require` statement must check if the length of the `protocolTokens` input parameter is the same as the length of `allAvailableTokens`.

QSP-25 Initialization can be done multiple times

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CbdcTokenV3_1.sol`

Description: The owner of the `CbdcTokenV3_1.sol` could call `manualInitialize` multiple times. This would reset several state variables. The semantics of the function name gives the impression that it should only be called once.

Recommendation: Add a flag which is checked to be `false` when the `manualInitialize` function starts executing and is set to `true` inside `manualInitialize`.
Update from the Cbdc Finance team: Once deployed, `manualInitialize` should be called only once and then a new implementation of `CbdcTokenV3_1` should be deployed and set for all
`CbdcToken` proxies (I added a `CbdcTokenGovernance.sol` file which is a copy of `CbdcTokenV3_1.sol` with `manualInitialize` removed and `setMaxUnlentPerc` reintroduced). The new implementation should simply have `manualInitialize` removed in order to save bytecode size for future updates by the governance and it will also allow us to use the compiler optimization runs which are currently set to 1 so we can also save some gas on calls, we avoided to add a flag checking this because of what said above and because we tried to save bytecode size everywhere possibile (Current bytecode size with some dummy address set instead of placeholders is 24567.5 vs max of 24576, and with the `setMaxUnlentPerc` method removed.)

QSP-26 Missing input check

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CbdcTokenV3_1.sol`

Description:

1. The `manualInitialize` function does not check if the length of the 2nd, 3rd and 4th input arrays is the same. The `for`-loop inside this function assumes the length of `_protocolTokens`, `_wrappers` and `_lastRebalancerAllocations` input arrays is the same.
2. A comment on L105 indicates that the `_newGovTokens` array "should include Cbdc". However, this is not verified inside the function. It could be verified by setting a binary flag to true inside the `if`-statement on `L124: if (newGov == Cbdc) { continue; }`, and then checking this flag after the `for`-loop using a `require` statement.

Recommendation: Add `require` statements accordingly.
Update from the Cbdc Finance team: Some checks have not been added mostly to save on bytecode size.

QSP-27 Missing return value

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CbdcTokenV3_1.sol`

Description: The `getGovApr` function does not have an explicit return value for the cases where the `if`-statement is not entered, i.e. the `if`-condition is not `true`.

Recommendation: Add an explicit `return` statement after the `if`-statement.
Update from the Cbdc Finance team: Some `return` statements have not been added mostly to save on bytecode size.

QSP-28 Privileged roles

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CbdcTokenV3_1.sol`

Description: The owner of the `CbdcTokenV3_1` contract has the right to change the following state variables at any time, they can even front-run end-users

:1. `setAllAvailableTokensAndWrappers` can be set to any address including EOAs
2. `setGovTokens` can be set to any address including EOAs

3. `setRebalancer` can be set to any address including an EOA
4. `setFee` upper bounded by 10%
5. `setMaxUnlentPerc` upper bounded to 100%
6. `setFeeAddress` can be set to any address including an EOA
7. `setOracleAddress` can be set to any address including an EOA
8. `setCbdcControllerAddress` can be set to any address including an EOA
9. `setIsRiskAdjusted`
10. `setAllocations` this can also be done by the `rebalancer` address

Recommendation: These privileged operations and their potential consequences should be clearly communicated to (non-technical) end-users via publicly available documentation.
Update from the Cbdc Finance team: The owner will be transferred to the governance right on deployment; one multisig wallet controlled by us will have the ability to pause the contract in case of emergency (withdrawals are not paused) but other than that the owner of the contract will be the `Timelock.sol` from governance right in the deployment. You can see the migration scripts number 5 and the newly added number 6 for transferring ownership to governance. Public documentation will get revamped prior to the governance launch.

QSP-29 Incorrect average price computation

Severity: *Undetermined*

Status: Fixed

File(s) affected: **Cbdc**`TokenV3_1.sol`

Description: The `userNoFeeQtyFrom` part of the `qty` input parameter of the`_updateUserFeeInfo` function is subtracted twice from `totBalance`: on deposits on L889 and L892. See the following code snippet:

```
889:    uint256 totBalance = balanceOf(usr).sub(userNoFeeQty[usr]);
890:    // noFeeQty should not be counted here
891:    // (avgPrice * oldBalance) + (currPrice * newQty)) / totBalance
892:    userAvgPrices[usr] = userAvgPrices[usr].mul(totBalance.sub(qty)).add(price.mul(qty)).div(totBalance);
```

This happens because `userNoFeeQtyFrom` was already added to `userNoFeeQty[usr]`, which is first subtracted on L889. This leads to an incorrect `userAvgPrice` for that user. Additionally, the `price` should not be multiplied by `qty` on L892, because on transfers, the amount that is actually transfered to `usr` is equal to `userNoFeeQtyFrom`.

Recommendation: Update the average price computation to take into account that an amount of `userNoFeeQtyFrom` was already subtracted from `totBalance` on deposits.

QSP-30 Uninitialized inherited contracts and state variables

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: **Cbdc**`TokenV3_1.sol`

Description: The `initialize` method has been replaced with the `manualInitialize` method, which is significantly different:

1. There are several inherited contracts which were initialized in the `initialize`, but are not initialized in the `manualyInitialize` method. The following code snippet indicates the initialization of these contracts, which was removed:

```
    // Initialize inherited contracts
    ERC20Detailed.initialize(_name, _symbol, 18);
    Ownable.initialize(msg.sender);
    Pausable.initialize(msg.sender);
    ReentrancyGuard.initialize();
    GST2ConsumerV2.initialize();
```

1. Similarly, the following state variables: `token`, `tokenDecimals`, `cToken` and `maxUnlentPerc`, were initialized in the `initialize` method, but are not initialized in the `manualyInitialize` method.

Recommendation: Clarify if this is intentionally left uninitialized for some reason. If not, add the initialization of the aforementioned inherited contracts and state variables.
Update from the Cbdc Finance team: **Cbdc**`TokenV3_1` is an upgradable contract and that `initialize` method has already been called once, hence it can be removed now (for deployments of new `CbdcTokens` we would need to reintroduce it). `manualInitialize` will initialize this new implementation (storage is still the old one so no need to update).

QSP-31 Unclear functionality in `_getFee`

Severity: *Undetermined*

Status: Fixed

File(s) affected: **Cbdc**`TokenV3_1.sol`

Description: * The functionality of `L907`: `userNoFeeQty[msg.sender] = noFees ? noFeeQty.sub(amount) : 0;`, is unclear. It seems that what we want to achieve here is more like `userNoFeeQty[msg.sender] = balanceOf(msg.sender).sub(_amount);` when `fee == 0` and `userNoFeeQty[msg.sender] = noFeeQty.sub(amount)` when `noFeeQty >= amount`.

Recommendation: Clarify if the functionality is as-intended.

QSP-32 Wrong comparison between lengths

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: **Cbdc**`TokenGovernance.sol`

Description: On L148 in `CbdcTokenGovernance.sol` we can see the following `require` statement: `require(_newGovTokensEqualLen.length >= protocolTokens.length ,'!EQ');` From the other occurrences of `!EQ` we believe that it should indicate that the 2 terms being compared are not equal, which is different from what the Boolean expression in that

`require` statement is comparing, that is the comparison is actually checking if the length of the `_newGovTokensEqualLen` is higher-or-equal to the length of `protocolTokens`.

Recommendation:

1. Change the condition on L148 from `>=` to `==`.
2. It would additionally make sense to check that the length of the `_newGovTokensEqualLen` is higher-or-equal to the length of `_newGovTokens`, which is currently not being checked.

Update: The maximum `_newGovTokensEqualLen` length is `protocolTokens.length + 1` because Cbdcis not associated with any protocol token. Therefore, the `require` statement could be restricted to `require(_newGovTokensEqualLen.length == protocolTokens.length + 1, '!EQ');`.

QSP-33 The `flashLoanFee` is not settable

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CbdcTokenGovernance.sol`

Description: The `flashLoanFee` cannot be changed by a function call after the contract is deployed. The only way to change it is to upgrade/redeploy the contract.

Recommendation: We recommend adding a setter method such that the governance account could set it after a community vote.

QSP-34 Inconsistent array lengths breaks invariant

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `CbdcTokenGovernance.sol`

Description: Note: this issue is essentially the same as QSP-24 from a previous audit; the fix appears to have been reverted.
The length of the `allAvailableTokens` array and the `lastRebalancerAllocations` and `lastAllocations` arrays may diverge after calling `setAllAvailableTokensAndWrappers()`. This is because the allocations are not adjusted or checked to be of the same length with the `protocolTokens` or `wrappers` input arrays of the `setAllAvailableTokensAndWrappers()` function. This means that the owner can effectively remove tokens from the `allAvailableTokens` array and the sum of all corresponding allocations would not be 100% by calling `setAllAvailableTokensAndWrappers()`.

Exploit Scenario:

1. Owner (accidentally) removes 1 or more tokens by calling `setAllAvailableTokensAndWrappers()`
2. Either the owner forgets to call `setAllocations` OR they call `setAllocations`, but are front-run by an end-user that calls `redeemInterestBearingTokens` or any other function which uses the `allAvailableTokens` array.

This will lead to incorrect amounts being redeemed, loaned, etc.

Recommendation: Either add a check inside `setAllAvailableTokensAndWrappers` which does not let the owner remove tokens OR add another input array to `setAllAvailableTokensAndWrappers` which indicates the new allocations. Optionally, a Boolean input parameter could also be added to `setAllAvailableTokensAndWrappers` which indicates that the allocation should stay the same, in which case a `require` statement must check if the length of the `protocolTokens` input parameter is the same as the length of `allAvailableTokens`.

Update: From the Cbdc team -- we won't be changing the `setAllAvailableTokensAndWrappers`, and instead a specific process should be followed when a protocol needs to be removed (i.e.set allocation for that protocol to 0, ensure that funds have been fully redeemed from that protocol and then do the proposal). `openRebalance` method has been removed.

QSP-35 Flashloans may decrease funds if underlying protocols have redemption fees

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CbdcTokenGovernance.sol`

Description: The function `flashLoan` can be used to force triggering the rebalance process and move funds in and out different underlying protocols. If any of the underlying lending protocols have a redemption fee, an attacker who seeks to damage CbdcFinance can achieve this by rapidly performing large value flashloans that cause CbdcFinance to redeem and mint the underlying
protocol's tokens and end up losing money.

Recommendation: Ensure that the fee collected by the flash loan is larger than the sum of the redemption fee of the underlying protocols.

Update: From the Cbdc team: I think that this would only be true if they charge a fee at the redeem (not counted in their price), but even in that case we could fix it in the strategy itself probably

.

QSP-36 Unchecked function arguments

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CbdcTokenGovernance.sol`

Description: The function `_init` should ensure that `_tokenHelper` is non-zero.

Recommendation: Add a `require` statement ensuring that `_tokenHelper != address(0)`.

Update: This is done to save on bytcodesize.

QSP-37 Flashloan could be used as a tool to manipulate liquidities of the underlying lending protocols

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CbdcTokenGovernance.`

`sol`

Description: The `flashLoan` can be used to force triggering the rebalance process and moving funds in and out different underlying protocols. A related security issue is described in EIP-3156.

Recommendation: While the underlying protocol's are expected to protect against flash loans themselves, this avenue of attack should be considered when adding new protocols to the Cbdc system.

Update: The Cbdc team noted that it is not clear how this could affect the protocol itself given that it's already possible to do this with other protocols.
However, we still stress that caution should be used when adding underlying protocols. One notable example of a related attack is the yearn attack with the 3pool imbalance.

QSP-38 Uninitialized state variables

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `CbdcTokenGovernance.sol`

Description: Several important state variables: `token`, `tokenDecimals`, and `isRiskAdjusted`, are not initialized anywhere.

Recommendation: Ensure that these variables are properly initialized.

Update: Those variables are only set once though the `CbdcTokenV3_1` contract. The contract is then upgraded to `CbdcTokenGovernance` upon the first deploy for each new token
.

QSP-39 Owner can front-run flash loaners to change loan fee

Severity: *Informational*

Status: Mitigated

File(s) affected: `CbdcTokenGovernance.sol`

Description: The owner of the `CbdcTokenGovernance` contract has the privilege of front running any end-user who calls `flashLoan()` by calling `setFlashLoanFee()` and increasing the flash loan fee. Coupled with the fact that the `flashLoanFee` can be set up to 100% inside the `setFlashLoanFee()` function, this could be detrimental to the caller if sufficient funds are available in the caller's balance.

Recommendation:

1. We recommend that the caller of the `flashLoan()` function sends the expected flash loan fee as part of the `_params` parameter of that function. That user should check the expected flash loan fee inside the `onFlashLoan()` function and should revert if it is different than expected.
2. The maximum value of the `flashLoanFee` should be bounded to a reasonable amount, in a similar way to how the value of the `fee` is bounded inside of the `setFee()` function.

Update: The owner is the governance which can act only through the `timelock`. Any `onlyOwner` method takes at least 5 days so it's should not be an issue.

Automated Analyses

Mythril

Mythril reported no issues.

Slither

• Slither warns of several potential reentrancy issues, however as the associated external calls were to trusted contracts (either Cbdc contracts or underlying protocols),we classified these as false positives.

• Slither detects that there are "divided-before-multiplies" operations in the following `CbdcTokenV3_1.sol`functions. Re-ordering these operations may improve precision.

. `getAvgAPR()`
  · `avgApr = avgApr.add(ILendingProtocol(protocolWrappers[allAvailableTokens[i]]).getAPR().mul(amounts[i].mul(10 ** 18).div(total)).div(10 ** 18))`

. `_redeemGovTokens()`:
  · `share = usrBal.mul(delta).div(10 ** 18)`
  · `feeDue = share.mul(fee).div(100000)`

As of commit `e09d4f5`:

• In `CbdcTokenGovernance.sol`, several important state variables: `token`, `tokenDecimals`, and `isRiskAdjusted`, are not initialized anywhere.

Adherence to Specification

The code adheres to the specification provided, as well as the inline documentation.

Code Documentation

The code is generally well-documented. We suggest several improvements related to magic constants above in QSP-6. Additionally, we noted the following:

• **Update: fixed.** In `CbdcTokenV3.sol`, on L42 the comment "// Cbdc rebalancer current implementation address" does not relate to the code below.

• **Update: fixed.** In `CbdcTokenV3.sol`, comments describing `userAvgPrices` and `userNoFeeQty` should be added.

• **Update: fixed.** In `CbdcAave.sol`, we recommend documenting that the Aave-Dai price will always be one-to-one (as per L133).

• **Update: fixed.** There are several spelling errors throughout: "possibile", "supplyied", "aum" (should be "sum"), "crete", "DyDc".

As of commit `35d61ae` we noted the following:

- **Update: fixed.** The comment of the `setFee` function in `CbdcTokenV3_1` contains the following text: “max settable is MAX_FEE constant”. However the `MAX_FEE` constant is not defined.
- **Update: fixed.** The comment of the `setMaxUnlentPerc` function in `CbdcTokenV3_1` contains the following text, which seems to be wrongly copied from another function’s code comment: “max settable is MAX_FEE constant”.
- **Update: fixed.** In the comment block of `CbdcTokenV3_1.setAllAvailableTokensAndWrappers`, it is not clear what is meant by "This method can be delayed".
- **Update: fixed.** In `CbdcTokenV3_1.sol`, the typo "shar" should be "share".
- **Update: fixed.** In `CbdcTokenV3_1.sol`, comments should be added to the `transfer*` functions indicating why the government tokens get redeemed for the from -address but not the to-address.
- **Update: fixed.** In `CbdcTokenV3_1.sol`, the comment "This method triggers a rebalance of the pools if needed" no longer applies to `mintCbdcToken` and `redeemCbdcToken`.
- **Update: fixed.** In `CbdcTokenV3_1.sol` in the function `_updateUserGovIdxTransfer()`, the comment `// user _to should have -> shareTo +
(sharePerTokenFrom * amount / 1e18) = (balanceTo + amount) * (govTokenIdx - userIdx) / 1e18` should instead say `user _from ...`.

As of commit `50da42b9`, we noted the following:

- * **Update: fixed.** The `manualInitialize` function declared on L104 of `CbdcTokenV3_1.sol` does not have comments to describe its input parameters and return value. The comment that it has does not seem to reflect the actual implementation because the Cbdc token address is a constant.
- * **Update: fixed.** The `setGovTokens` function in `CbdcTokenV3_1.sol` is missing the description of its 2nd parameter.
- * **Update: fixed.** The `_getFee` function in `CbdcTokenV3_1.sol` is missing the description of its 3rd parameter `currPrice`.
- * **Update: fixed.** Typo on L628 in `CbdcTokenV3_1.sol`: "give" -> "gives

"As of commit `e09d4f5` we noted the following:

- **Update: fixed.** L114 in `CbdcTokenGovernance.sol`: "The fee flash borrowed" -> "The flash loan fee"
- **Update: fixed.** The comments at the beginning of the `CbdcTokenGovernance.sol` and `CbdcTokenHelper.sol` files are identical to those at the beginning of the `CbdcTokenV3_1.sol` file. These should be adjusted for token governance:

```
/**
* @title: Cbdc Token (V3) main contract
* @summary: ERC20 that holds pooled user funds together
*           Each token rapresent a share of the underlying pools
*           and with each token user have the right to redeem a portion of these pools
* @author: Cbdc Labs Inc., Cbdc.finance
*
```

- **Update: fixed.** In `CbdcTokenGovernance.flashLoan`, "redeemd" is misspelled.
- **Update: fixed.** In `_redeemGovTokensFromProtocol` on L928: `CbdcController(CbdcController).claimCbdc(holders, holders);` should be documented ,particularly since the first parameter is now unused in `claimCbdc`.

Adherence to Best Practices

The code does not fully adhere to best practices. In particular:

- **Update: fixed.** There is commented out code on L78-99 of `iERC20Fulcrum.sol` that should be removed if not needed.
- **Update: fixed.** Although the user is intended to interact with the dApp through an `CbdcToken` (specifically through `mintCbdcToken()`), the user could instead try to directly interact with `CbdcCompound` or `CbdcFulcrum`, first transferring DAI to the contract and then attempting to `mint()`. If that were the case, since the DAI transfer and `mint()` are not autonomous, a different user could scoop the minted tokens by invoking `mint()` first. As an added precaution to prevent this scenario, it may be beneficial to restrict calls to `mint()` in `CbdcCompound` and `CbdcFulcrum` to only be callable from the `CbdcToken` contract.
- **Update: fixed.** On L91 of `CbdcFulcrum`: "// q = a1 * (s1 / (s1 + x1)) * (b1 / (s1 + x)1) * o1 / k1", the "x)1" is a typo.
- **Update: fixed.** In `CbdcFactory.newCbdcToken()`, the address parameters should be checked to be non-zero with require-statements.
- **Update: fixed.** In `CbdcPriceCalculator.tokenPrice()`, there should be a check that `currentTokensUsed.length == protocolWrappersAddresses.length`.
- **Update: fixed.** The conditional on L456 of `CbdcToken.sol` could simply be the else-branch of the previous if-statement.
- **Update: fixed.** On L219 of `CbdcToken.sol`, it is not clear what the comment "// We should save the amount one has deposited to calc interests" is referring.
- **Update: fixed.** On L95 of `CbdcCompound.sol` the constants `10**18` and `100` are used instead of the passed in parameters `params[0]` and `params[8]`.
- **Update: fixed.** In `CbdcCompound`, `CbdcFulcrum`, and `CbdcRebalancer`, the constructors should check that the passed in addresses are non-zero.
- **Update: fixed.** In Cbdc `ebalancer.sol`, the comments on L110 and L128 do not appear correct.
- **Update: fixed.** Functions such as `CbdcToken.setProtocolWrapper()` and `CbdcFactory.setTokenOwnershipAndPauser()` should check for non-zero arguments .Further, all the `setCbdcToken()` functions should ensure that the `_CbdcToken` parameter is non-zero.
- In `CbdcRebalancerV3.setAllocations()`, since `_addresses` should be equal to `lastAmountsAddresses`, you may as well remove that argument and use `lastAmountsAddresses`. **Update:** `setAllocations` and the `_addresses` parameter are used to ensure that each allocation submitted by an off-chain bot is for the correct lending protocol.
- In `CbdcDyDx.sol`, in `nextSupplyRateWithParams()` why not just enforce length 1 for the input array? **Update:** The parameter is an array in adherence with the `ILendingProtocol` interface.
- **Update: fixed.** L540 of `CbdcTokenV3.sol` should be `if (_skipWholeRebalance || areAllocationsEqual)` instead of `if (_skipWholeRebalance || (areAllocationsEqual && balance > 0))`. The reason is that once `areAllocationsEqual` is true, there’s no need to rebalance even when the balance is not larger than 0.
- In `CbdcDSR.sol`, since `CHAI` is a known token, the address could be declared as a constant instead of a constructor parameter. **Update:** this approach maintains
uniformity amongst the wrapper constructors.

As of commit `35d61ae` we noted the following:

• **Update: fixed.** In the constructor of CbdcRebalancerV3_1 on L35, there is a branch instruction that will be true only for the first iteration. Executing this branch instruction in each iteration will waste gas. Recommendation: perform the assignment for the first entry in the array outside of the loop and start the loop with `i = 1`:

```
lastAmounts[0] = 100000;
lastAmountsAddresses[0] = _protocolTokens[0];
for(uint256 i = 1; i < _protocolTokens.length; i++) {
```

• The `total` variable inside the `setAllocations` function from Cbdc `ebalancerV3_1` should be explicitly initialized to `0` on L98.

• **Update: several constants have been fixed; others have not been updated due to upgradeability of storage concerns.** Replace inline constants with named constants:

. **Update: fixed.** The inline constant `10000` is used 2 times in CbdcRebalancerV3_1.

. The inline constant `10000` is used 1 time in CbdcTokenV3_1.

. * **Update: fixed.** The inline constant `100000` is used 8 times in CbdcTokenV3_1.

. **Update: fixed.** The inline constant `10**18` is used 9 times in CbdcTokenV3_1.

• **Update: fixed.** In CbdcTokenV3_1.sol, the expression `(totalRedeemd < maxUnlentBalance)` could change to be `<=`, which would make the following if-statement unnecessary: `if (totalRedeemd > 1) {`.

As of commit `50da42b9`, we noted the following:

• * **Update: fixed.** Resolve and remove all TODO comments, e.g. such as those on L85, L111 and L112 in CbdcTokenV3_1.sol.

• * **Update: fixed.** Replace the following magic numbers with named constants:

. * **Update: fixed.** `100000` appears several times in CbdcTokenV3_1.sol

As of commit `e09d4f5` we noted the following:

• Named constants should have a name which provide semantic meaning and not simply indicates the value of the constant. For example, the constant `ONE_18` defined in multiple files including CbdcTokenGovernance.sol and CbdcTokenHelper.sol, should be renamed to something like: Cbdc_TOKEN_DECIMALS, which conveys more semantic meaning. **Update from the Cbdc team:** for the ONE_18 we prefer to keep it as is, but we will keep in mind the general advice.

• Magic numbers should be replaced with named constants. For example, `10**23` on L986 in CbdcTokenGovernance.sol. **Update from the Cbdc team:** the 10**23 is well documented and we didn't wanted to add other constant/variables.

• **Update: fixed.** Avoid code clones. Favor code reuse. For example, on L704 in CbdcTokenGovernance.sol: `uint256 _flashFee = _amount.mul(flashLoanFee).div(FULL_ALLOC);`, the same computation as the one performed by the `flashFee()` function is used. We recommend calling the `flashFee()` function on L704 instead. This can be done by making the function `public` instead of `external`.

• Provide descriptive error messages in `require` statements. These serve a double role: code documentation and debugging helpers. All `require` statements in `CbdcTokenGovernance.sol` contain cryptic error messages such as: "0", "EXEC", "DONE", "LEN", "!EQ", which also do not indicate which function the error has occurred in. We recommend changing these error messages or providing user documentation to map such error messages/codes to a human readable description. **Update from the Cbdc team:** for the require messages we kept them short to save a lot on bytecodesize; those should still be enough to debug txs, but the idea to have error code instead could be implemented in the future.

• **Update: fixed.** Commented code should be removed. For example, L983-984 in CbdcTokenGovernance.sol.

• **Update: fixed.** In CbdcTokenGovernance.setFee, consider changing the `10000` into `FULL_ALLOC/10` for better maintenance.

• CbdcTokenGovernance.sol should inherit the `IERC3156FlashLender` interface. **Update from the Cbdc team:** we avoided to inherit from it just to be 110% sure to not break anything given that all contracts are upgradable (even though no storage is touched).

• **Update: fixed.** In CbdcTokenGovernance.sol on LL877 consider moving this entire `if-else` statement into the body of `if (supply > 0)` to avoid unexpected results from happening.

• **Update: fixed.** Consider adding reentrancy protection to the CbdcTokenGovernance.sol.flashLoan function.

Test Results

Test Suite Results

**Update as of commit `e09d4f5`: some tests for previously audited contracts fail due to timeouts which influenced coverage and test results.

```
Contract: CbdcBatchConverter
   ✓ constructor set rebalanceManager addr (98ms)
   ✓ cannot withdraw before first migration (841ms)
   ✓ single user migration (576ms)
   ✓ multiple user migration, single batch (881ms)
   ✓ multiple user migration, multiple batch (2075ms)

Contract: CbdcTokenV3_1
   ✓ initialize set a name (39ms)
   ✓ initialize set a symbol (145ms)
   ✓ initialize set a decimals (93ms)
   ✓ initialize set a token (DAI) address (276ms)
   ✓ initialize set a rebalancer address (136ms)
   ✓ initialize set owner
   ✓ initialize set pauser (217ms)
   ✓ manualInitialize set stuff (1098ms)
  1) _init set stuff

  Events emitted during test:
  ---------------------------

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0xA782e72F1D3befBd4DDC04F487ef10ab40340769 (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
```

```
    to: <indexed> 0x6043A7347F46EaAcDe0ED7C98B53584823D78A90 (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0xe7E39F27101a763cB55c0Fb8cf6844E8a07761f9 (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x6DdFdEdB38822099547ef7E056Fb40d4d11f3C88 (type: address),
    value: 100000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 10000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x80c5d818C9a43e932dD94A0Ee161A3ebFA823be9 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  Ownable.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

  IERC20.Approval(
    owner: <indexed> 0x4a1CD0CF2819eF3f2B7f05BF5d02B858b9384165 (type: address),
    spender: <indexed> 0x6DdFdEdB38822099547ef7E056Fb40d4d11f3C88 (type: address),
    value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
  )

  Ownable.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

  IERC20.Approval(
    owner: <indexed> 0x078759ffb75b3bCEBfd6bF517bd896b1AF2FaaaC (type: address),
    spender: <indexed> 0x80c5d818C9a43e932dD94A0Ee161A3ebFA823be9 (type: address),
    value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0xE96C48EA7F75D9957AdDAc74c707276f26eEE433 (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    to: <indexed> 0x160eBf7F40d9889D834047f55e9BF5fC51e49EDF (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  Ownable.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

  IERC20.Approval(
    owner: <indexed> 0x035DE74e37A8f86c0C75dd6C8FF6BfBfB3c6888C (type: address),
    spender: <indexed> 0x077BD1BE91206a013CcC641C7983CaA1FBad0b28 (type: address),
    value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
  )

  IERC20.Approval(
    owner: <indexed> 0x22B0cD56859db4E9160b860fbD2b94a5C1B61153 (type: address),
    spender: <indexed> 0x1E0447b19BB6EcFdAe1e4AE1694b0C3659614e4e (type: address),
    value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x22B0cD56859db4E9160b860fbD2b94a5C1B61153 (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Approval(
    owner: <indexed> 0x22B0cD56859db4E9160b860fbD2b94a5C1B61153 (type: address),
    spender: <indexed> 0xA4dfa8e902CdEDcB6C1f3D3E79AFADaBBA60F839 (type: address),
    value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
  )

  Ownable.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

  IERC20.Approval(
    owner: <indexed> 0x2F6e1CD70fBBfD27cD512CFCc3d980a7Af4923a3 (type: address),
    spender: <indexed> 0x22B0cD56859db4E9160b860fbD2b94a5C1B61153 (type: address),
    value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
  )

  IERC20.Approval(
    owner: <indexed> 0x2F6e1CD70fBBfD27cD512CFCc3d980a7Af4923a3 (type: address),
    spender: <indexed> 0x22B0cD56859db4E9160b860fbD2b94a5C1B61153 (type: address),
    value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
  )

  Ownable.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

  Ambiguous event, possible interpretations:
  * CbdcTokenV3_1Mock.OwnershipTransferred(
      previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
      newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
    )
  * CbdcTokenV3_1Mock.OwnershipTransferred(
      previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
      newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
    )

  PauserRole.PauserAdded(
    account: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

  PauserRole.PauserAdded(
    account: <indexed> 0xaDa343Cb6820F4f5001749892f6CAA9920129F2A (type: address)
  )


  ---------------------------
   ✓ setAllAvailableTokensAndWrappers (1301ms)
   ✓ allows onlyOwner to setRebalancer (489ms)
   ✓ allows onlyOwner to setOracleAddress (465ms)
```

```
    ✓ allows onlyOwner to setFeeAddress (254ms)
    ✓ allows onlyOwner to setFee (422ms)
    ✓ allows onlyOwner to setMaxUnlentPerc (374ms)
    ✓ calculates current tokenPrice when CbdcToken supply is 0 (77ms)
    ✓ calculates current tokenPrice when funds are all in one (4578ms)
    ✓ calculates current tokenPrice when funds are all in one pool (5551ms)
    ✓ calculates current tokenPrice when funds are in different pools (8482ms)
    ✓ get all APRs from every protocol (538ms)
    ✓ get current avg apr of Cbdc (with no COMP apr) (3339ms)
    ✓ get current avg apr of Cbdc with COMP (1999ms)
    ✓ mints Cbdc tokens (1757ms)
    ✓ cannot mints Cbdc tokens when paused (710ms)
    ✓ does not redeem if CbdcToken total supply is 0 (168ms)
    ✓ redeems Cbdc tokens (4349ms)
    ✓ redeems Cbdc tokens using unlent pool (4193ms)
    ✓ redeemInterestBearingTokens (4897ms)
    ✓ cannot rebalance when paused (295ms)
    ✓ rebalances when _newAmount > 0 and only one protocol is used (1933ms)
    ✓ rebalances when _newAmount > 0 and only one protocol is used and no unlent pool (2627ms)
    ✓ rebalances and multiple protocols are used (5714ms)
    ✓ _amountsFromAllocations (public version)
    ✓ _mintWithAmounts (public version) (2138ms)
    ✓ _redeemAllNeeded (public version) when liquidity is available (3905ms)
    ✓ _redeemAllNeeded (public version) when liquidity is available and with reallocation of everything (5673ms)
    ✓ _redeemAllNeeded (public version) with low liquidity available (4669ms)
    ✓ rebalance when liquidity is availabler (7191ms)
    ✓ rebalance when liquidity is not available (6737ms)
    ✓ rebalance when liquidity is not available and no unlent perc (6399ms)
    ✓ rebalance when underlying tokens are in contract (ie after mint) and rebalance and Cbdc allocations are equal (7093ms)
    ✓ rebalance with no new amount and allocations are equal (4505ms)
    ✓ rebalance when prev rebalance was not able to redeem all liquidity because a protocol has low liquidity (14144ms)
    ✓ calculates fee correctly when minting / redeeming and no unlent (7868ms)
    ✓ calculates fee correctly when minting / redeeming with unlent (9121ms)
    ✓ calculates fee correctly when minting multiple times and redeeming (10786ms)
    ✓ calculates fee correctly when minting multiple times and redeeming with different fees (14902ms)
    ✓ calculates fee correctly when redeeming a transferred CbdcToken amount (10250ms)
    ✓ calculates fee correctly when redeeming a transferred CbdcToken amount with different fees (12117ms)
    ✓ calculates fee correctly when redeeming a transferred CbdcToken amount after having previosly deposited (12842ms)
    ✓ calculates fee correctly when using transferFrom (7928ms)
    ✓ charges fee only to some part to whom previously deposited when there was not fee and deposited also when there was a fee (5093ms)
    ✓ charges fee only to some part to whom previously deposited when there was fee and deposited also when there was no fee (9842ms)
    ✓ redeemGovTokens complex test (6930ms)
    ✓ redeemGovTokens (6555ms)
    ✓ redeemGovTokens test 2 (3999ms)
    ✓ getGovTokensAmounts (4202ms)
    ✓ redeemGovTokens with fee (6699ms)
    ✓ redeemGovTokens on transfer to new user (5436ms)
    ✓ redeemGovTokens on transfer to existing user (5705ms)
    ✓ transfer correctly updates userAvgPrice when transferring an amount > of no fee qty (7263ms)
    ✓ setAllocations contract fix - setAllocations should not fail if wrappers count increased (935ms)
    ✓ setAllocations contract fix - setAllocations should not fail if wrappers count decreased (736ms)
    ✓ getGovTokens (57ms)
    ✓ getAllAvailableTokens (63ms)
    ✓ getProtocolTokenToGov (41ms)
    ✓ getAllocations (1858ms)
  2) flashLoanFee

  Events emitted during test:
  ---------------------------

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x494CA97b571716177b91B1dF6e7b2Fd1d459B7A6 (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x2569C597b5a36c3441D8FD82f5CB14128f70544e (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x93C1837740373534cD6113d06cA032Ed735937DF (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x5f74946317FB10f3899Ce0261a105C99068C0903 (type: address),
    value: 100000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 10000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0xB53D5e67Aa9134f31E1D5dc78D22751b469e5172 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  Ownable.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

  IERC20.Approval(
    owner: <indexed> 0x3d743E270a1eE8332d7Ef63F63E060DEBDe43Dd4 (type: address),
    spender: <indexed> 0x5f74946317FB10f3899Ce0261a105C99068C0903 (type: address),
    value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
  )

  Ownable.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

  IERC20.Approval(
    owner: <indexed> 0x4d3853a48744cFDE8575347E1A31e8DB90BC046D (type: address),
    spender: <indexed> 0xB53D5e67Aa9134f31E1D5dc78D22751b469e5172 (type: address),
    value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x71DC02d2E39b4Dd7A7B825481002f6748A6644C0 (type: address),
    value: 100000000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
    to: <indexed> 0xb45ACDe13BAf56d71f54a6039F0739f06b6ac781 (type: address),
    value: 100000000000000000000 (type: uint256)
  )

  Ownable.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

  IERC20.Approval(
```

```
  owner: <indexed> 0xD5AAb05CA46F0adF19f648F0Af2cd69884Ad3700 (type: address),
  spender: <indexed> 0xC8CFfacf1958b163F024506B77eb50753f74129b (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x541F7171e3Ae58537dE9A1B7dDE2dA23AeAA6d25 (type: address),
  spender: <indexed> 0x1E0447b19BB6EcFdAe1e4AE1694b0C3659614e4e (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x541F7171e3Ae58537dE9A1B7dDE2dA23AeAA6d25 (type: address),
  value: 100000000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  value: 100000000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x541F7171e3Ae58537dE9A1B7dDE2dA23AeAA6d25 (type: address),
  spender: <indexed> 0x6056248a0b3b469A16E285b69FE0D29d1D117ED4 (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

Ownable.OwnershipTransferred(
  previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
)

IERC20.Approval(
  owner: <indexed> 0x440817F68675Af56c4A5460400CeAF421156a72a (type: address),
  spender: <indexed> 0x541F7171e3Ae58537dE9A1B7dDE2dA23AeAA6d25 (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x440817F68675Af56c4A5460400CeAF421156a72a (type: address),
  spender: <indexed> 0x541F7171e3Ae58537dE9A1B7dDE2dA23AeAA6d25 (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

Ownable.OwnershipTransferred(
  previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
)

Ambiguous event, possible interpretations:
* CbdcTokenV3_1Mock.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )
* CbdcTokenV3_1Mock.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

PauserRole.PauserAdded(
  account: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
)

PauserRole.PauserAdded(
  account: <indexed> 0xaDa343Cb6820F4f5001749892f6CAA9920129F2A (type: address)
)


---------------------------
 ✓ maxFlashLoan (5315ms)
 ✓ tokenPriceWithFee (8712ms)
 ✓ redeemCbdcTokenSkipGov (11105ms)
3) executes a flash loan

Events emitted during test:
---------------------------

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0xe78652486a6cADC80f7ccefAFCC21D1C6215BF7e (type: address),
  value: 100000000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  value: 100000000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x0d793973d0c6F0d2e4FC11cB303d7A4991757c5B (type: address),
  value: 100000000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  value: 100000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0xE82cD7b563201678755B5f9E0BdC1d35D073Ec63 (type: address),
  value: 100000000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  value: 100000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0xAb6261B4f9E7997f41F5965001624b8090F0A57f (type: address),
  value: 100000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  value: 10000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0xBf15a702F770ea6aef3166633616Bb9B734E776a (type: address),
  value: 100000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  value: 100000000000000000000 (type: uint256)
)

Ownable.OwnershipTransferred(
  previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
)

IERC20.Approval(
  owner: <indexed> 0xACc5f58366048b4107335cAb9987Cb9D3F5c703C (type: address),
  spender: <indexed> 0xAb6261B4f9E7997f41F5965001624b8090F0A57f (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

Ownable.OwnershipTransferred(
  previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
)

IERC20.Approval(
  owner: <indexed> 0x2811B081ecD440De1d623990b31A140c1d385927 (type: address),
  spender: <indexed> 0xBf15a702F770ea6aef3166633616Bb9B734E776a (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0xF0169AE7f46d8bbC705E13f82Fcc808673351206 (type: address),
  value: 100000000000000000000000 (type: uint256)
)
```

```
IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  value: 100000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  to: <indexed> 0x6A306c1bECDAD43da6e51AA7B4fB6373724d1c96 (type: address),
  value: 100000000000000000000 (type: uint256)
)

Ownable.OwnershipTransferred(
  previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
)

IERC20.Approval(
  owner: <indexed> 0x84feFc456430E063EF164ae02e4f3E7B9B82F94e (type: address),
  spender: <indexed> 0xCE08F45dAf36F98A0e33a61dB95A5b6F8F2D1Ce5 (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x1CaCa9F10B5dC472b7b14d28904eFA29Bb117C35 (type: address),
  spender: <indexed> 0x1E0447b19BB6EcFdAe1e4AE1694b0C3659614e4e (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x1CaCa9F10B5dC472b7b14d28904eFA29Bb117C35 (type: address),
  value: 100000000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  to: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  value: 100000000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x1CaCa9F10B5dC472b7b14d28904eFA29Bb117C35 (type: address),
  spender: <indexed> 0x6707b74355b35D990CE0c3D39fB299D6c4e19943 (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

Ownable.OwnershipTransferred(
  previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
)

IERC20.Approval(
  owner: <indexed> 0x097628F6bD655091ae13f99b4Af0DC3909A2787c (type: address),
  spender: <indexed> 0x1CaCa9F10B5dC472b7b14d28904eFA29Bb117C35 (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x097628F6bD655091ae13f99b4Af0DC3909A2787c (type: address),
  spender: <indexed> 0x1CaCa9F10B5dC472b7b14d28904eFA29Bb117C35 (type: address),
  value: 115792089237316195423570985008687907853269984665640564039457584007913129639935 (type: uint256)
)

Ownable.OwnershipTransferred(
  previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
  newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
)

Ambiguous event, possible interpretations:
* CbdcTokenV3_1Mock.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )
* TokenV3_1Mock.OwnershipTransferred(
    previousOwner: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    newOwner: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
  )

PauserRole.PauserAdded(
  account: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address)
)

PauserRole.PauserAdded(
  account: <indexed> 0xaDa343Cb6820F4f5001749892f6CAA9920129F2A (type: address)
)

IERC20.Transfer(
  from: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  to: <indexed> 0x7b94aC3E3AC4a2f5347E3e60616D9F1e51a1a25a (type: address),
  value: 100000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x7b94aC3E3AC4a2f5347E3e60616D9F1e51a1a25a (type: address),
  spender: <indexed> 0x348fD6DBc7105923Bc085021c4BAecB5E226A542 (type: address),
  value: 100000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x7b94aC3E3AC4a2f5347E3e60616D9F1e51a1a25a (type: address),
  to: <indexed> 0x348fD6DBc7105923Bc085021c4BAecB5E226A542 (type: address),
  value: 100000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x7b94aC3E3AC4a2f5347E3e60616D9F1e51a1a25a (type: address),
  spender: <indexed> 0x348fD6DBc7105923Bc085021c4BAecB5E226A542 (type: address),
  value: 0 (type: uint256)
)

Ambiguous event, possible interpretations:
* CbdcTokenV3_1Mock.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x7b94aC3E3AC4a2f5347E3e60616D9F1e51a1a25a (type: address),
    value: 100000000000000000000 (type: uint256)
  )
* CbdcTokenV3_1Mock.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0x7b94aC3E3AC4a2f5347E3e60616D9F1e51a1a25a (type: address),
    value: 100000000000000000000 (type: uint256)
  )

CbdcTokenV3_1NoConst.Referral(
  _amount: 100000000000000000000 (type: uint256),
  _ref: 0x0000000000000000000000000000000000000001 (type: address)
)

IERC20.Transfer(
  from: <indexed> 0x47fCbA4f604F60087f046627E9323768b4339046 (type: address),
  to: <indexed> 0x4F4b696dd715829E4d9BF7A565Cb2D1AFe152F55 (type: address),
  value: 200000000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x348fD6DBc7105923Bc085021c4BAecB5E226A542 (type: address),
  to: <indexed> 0x4F4b696dd715829E4d9BF7A565Cb2D1AFe152F55 (type: address),
  value: 100000000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x4F4b696dd715829E4d9BF7A565Cb2D1AFe152F55 (type: address),
  spender: <indexed> 0x348fD6DBc7105923Bc085021c4BAecB5E226A542 (type: address),
  value: 100080000000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x4F4b696dd715829E4d9BF7A565Cb2D1AFe152F55 (type: address),
  to: <indexed> 0x348fD6DBc7105923Bc085021c4BAecB5E226A542 (type: address),
  value: 100080000000000000000 (type: uint256)
)

IERC20.Approval(
  owner: <indexed> 0x4F4b696dd715829E4d9BF7A565Cb2D1AFe152F55 (type: address),
  spender: <indexed> 0x348fD6DBc7105923Bc085021c4BAecB5E226A542 (type: address),
  value: 0 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0x348fD6DBc7105923Bc085021c4BAecB5E226A542 (type: address),
  to: <indexed> 0xACc5f58366048b4107335cAb9987Cb9D3F5c703C (type: address),
  value: 99079200000000000000 (type: uint256)
)

IERC20.Transfer(
  from: <indexed> 0xACc5f58366048b4107335cAb9987Cb9D3F5c703C (type: address),
```

```
    to: <indexed> 0xAb6261B4f9E7997f41F5965001624b8090F0A57f (type: address),
    value: 990792000000000000000 (type: uint256)
  )

  IERC20.Approval(
    owner: <indexed> 0xACc5f58366048b4107335cAb9987Cb9D3F5c703C (type: address),
    spender: <indexed> 0xAb6261B4f9E7997f41F5965001624b8090F0A57f (type: address),
    value: 115792089237316195423570985008687907853269984665640564038466792007913129639935 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0x0000000000000000000000000000000000000000 (type: address),
    to: <indexed> 0xACc5f58366048b4107335cAb9987Cb9D3F5c703C (type: address),
    value: 4953960000000 (type: uint256)
  )

  IERC20.Transfer(
    from: <indexed> 0xACc5f58366048b4107335cAb9987Cb9D3F5c703C (type: address),
    to: <indexed> 0x348fD6DBc7105923Bc085021c4BAecB5E226A542 (type: address),
    value: 4953960000000 (type: uint256)
  )

  CbdcTokenV3_1NoConst.FlashLoan(
    target: <indexed> 0x4F4b696dd715829E4d9BF7A565Cb2D1AFe152F55 (type: address),
    initiator: <indexed> 0x7b94aC3E3AC4a2f5347E3e60616D9F1e51a1a25a (type: address),
    amount: 100000000000000000000 (type: uint256),
    premium: 800000000000000000 (type: uint256)
  )


  ---------------------------
    ✓ sets gov tokens when _newGovTokens and _protocolTokens lengths are different (645ms)

Contract: MinimalInitializableProxyFactory
   ✓ deploys a minimal proxy and initializes it (626ms)

Contract: Cbdc
   ✓ constructor set a token address (256ms)
   ✓ constructor set an underlying address (479ms)
   ✓ allows onlyOwner to setCbdcToken (899ms)
   ✓ returns next supply rate given amount (178ms)
   ✓ returns next supply rate given params (counting fee) (557ms)
   ✓ getPriceInToken returns aToken price (67ms)
   ✓ getAPR returns current yearly rate (counting fee) (83ms)
   ✓ mint returns 0 if no tokens are presenti in this contract (80ms)
   ✓ mint creates aTokens and it sends them to msg.sender (1422ms)
   ✓ redeem creates aTokens and it sends them to msg.sender (1503ms)

Contract: Cbdc
   ✓ constructor set a token address (457ms)
   ✓ constructor set an underlying address (365ms)
   ✓ returns next supply rate given amount (1185ms)
   ✓ getPriceInToken returns aToken price (136ms)
   ✓ getAPR returns current yearly rate (counting fee) (326ms)
   ✓ mint returns 0 if no tokens are present in this contract (581ms)
   ✓ mint creates aTokens and it sends them to msg.sender (2369ms)
   ✓ redeem creates aTokens and it sends them to msg.sender (3151ms)

Contract: CbdcCompound
   ✓ constructor set a token address
   ✓ constructor set an underlying address
   ✓ allows onlyOwner to setCbdcToken (877ms)
   ✓ allows onlyOwner to setBlocksPerYear (939ms)
   ✓ returns next supply rate given amount (92ms)
   ✓ returns next supply rate given params (counting fee) (399ms)
   ✓ getPriceInToken returns cToken price (1330ms)
   ✓ getAPR returns current yearly rate (counting fee) (991ms)
   ✓ mint returns 0 if no tokens are presenti in this contract (39ms)
   ✓ mint creates cTokens and it sends them to msg.sender (3213ms)
   ✓ redeem creates cTokens and it sends them to msg.sender (1990ms)

Contract: CbdcCompoundETH
   ✓ constructor set a token address
   ✓ constructor set an underlying address (361ms)
   ✓ constructor set an underlying address (940ms)
   ✓ allows onlyOwner to setBlocksPerYear (2781ms)
   ✓ returns next supply rate given amount (3413ms)
   ✓ returns next supply rate given params (counting fee) (942ms)
   ✓ getPriceInToken returns cToken price (1372ms)
   ✓ getAPR returns current yearly rate (counting fee) (1650ms)
   ✓ mint returns 0 if no tokens are present in this contract (51ms)
   ✓ mint creates cTokens and it sends them to msg.sender (2947ms)
   ✓ redeem creates cTokens and it sends them to msg.sender (1912ms)

Contract: CbdcCompoundV2
   ✓ constructor set a token address
   ✓ constructor set an underlying address (913ms)
   ✓ allows onlyOwner to setCbdcToken (1161ms)
   ✓ allows onlyOwner to setBlocksPerYear (3980ms)
   ✓ returns next supply rate given amount (5458ms)
   ✓ returns next supply rate given params (counting fee) (3674ms)
   ✓ getPriceInToken returns cToken price (6283ms)
   ✓ getAPR returns current yearly rate (counting fee) (8676ms)
   ✓ mint returns 0 if no tokens are presenti in this contract (4051ms)
   ✓ mint creates cTokens and it sends them to msg.sender (12334ms)
   ✓ redeem creates cTokens and it sends them to msg.sender (2412ms)


   ✓ constructor set a token address
   ✓ constructor set an underlying address (941ms)
   ✓ constructor set CHAI contract infinite allowance to spend our DAI (1488ms)
   ✓ constructor set an secondsInAYear (1485ms)
   ✓ allows onlyOwner to setCbdcToken (9626ms)
   ✓ returns next supply rate given 0 amount (6733ms)
  4) "before each" hook for "returns next supply rate given amount != 0"

Contract: CbdcDyDx
  5) "before each" hook for "constructor set a token address"

Contract: CbdcFulcrum
   ✓ constructor set a token address (10385ms)
   ✓ constructor set a underlying address (2725ms)
   ✓ allows onlyOwner to setCbdcToken (2652ms)
   ✓ returns next supply rate given amount (656ms)
   ✓ returns next supply rate given params (501ms)
   ✓ getPriceInToken returns iToken price (941ms)
   ✓ getAPR returns current yearly rate (counting fee ie spreadMultiplier) (2515ms)
   ✓ mint returns 0 if no tokens are presenti in this contract (563ms)
   ✓ mint creates iTokens and it sends them to msg.sender (2288ms)
   ✓ redeem creates iTokens and it sends them to msg.sender (3582ms)
   ✓ redeem reverts if not all amount is available (2791ms)

Contract: CbdcFulcrumDisabled
   ✓ constructor set a token address (1030ms)
   ✓ constructor set a underlying address (364ms)
   ✓ allows onlyOwner to setCbdcToken (3459ms)
   ✓ returns next supply rate given amount (2296ms)
   ✓ returns next supply rate given params (875ms)
   ✓ getPriceInToken returns iToken price (2893ms)
   ✓ getAPR returns current yearly rate (counting fee ie spreadMultiplier) (3033ms)
   ✓ mint returns 0 if no tokens are present in this contract (1512ms)
   ✓ mint creates iTokens and it sends them to msg.sender (6776ms)
   ✓ redeem creates iTokens and it sends them to msg.sender (8859ms)
   ✓ redeem reverts if not all amount is available (19439ms)

Contract: FulcrumV2
   ✓ constructor set a token address (4487ms)
   ✓ constructor set a underlying address (7153ms)
   ✓ allows onlyOwner to setCbdcToken (32148ms)
   ✓ returns next supply rate given amount (36846ms)
   ✓ returns next supply rate given params (55887ms)
   ✓ getPriceInToken returns iToken price (71970ms)
  6) "before each" hook for "getAPR returns current yearly rate (counting fee ie spreadMultiplier)"

Contract: yxToken
  7) "before each" hook for "constructor set a underlying address"


161 passing (1h)
7 failing

1) Contract: CbdcTokenV3_1
     _init set stuff:

    AssertionError: expected '80' to equal '90'
    + expected - actual

    -80
    +90

    at Context.<anonymous> (test/CbdcTokenV3_1.js:329:59)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)

2) Contract: CbdcTokenV3_1
     flashLoanFee:
```

```
    AssertionError: expected '80' to equal '90'
    + expected - actual

    -80
    +90

    at Context.<anonymous> (test/CbdcTokenV3_1.js:2520:29)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)

3) Contract: CbdcTokenV3_1
     executes a flash loan:

    AssertionError: expected '800000000000000000' to equal '900000000000000000'
    + expected - actual

    -800000000000000000
    +900000000000000000

    at executeFlashLoan (test/CbdcTokenV3_1.js:2703:39)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)
    at Context.<anonymous> (test/CbdcTokenV3_1.js:2730:5)

4) Contract: CbdcDSR
     "before each" hook for "returns next supply rate given amount != 0":
   Error: Timeout of 300000ms exceeded. For async tests and hooks, ensure "done()" is called; if returning a Promise, ensure it resolves. (/home/ezulkosk/audits/Cbdc-contracts/test/wrappers/CbdcDSR.js)
    at listOnTimeout (internal/timers.js:554:17)
    at processTimers (internal/timers.js:497:7)

5) Contract: CbdcDyDx
     "before each" hook for "constructor set a token address":
   Error: Timeout of 300000ms exceeded. For async tests and hooks, ensure "done()" is called; if returning a Promise, ensure it resolves. (/home/ezulkosk/audits/Cbdc-contracts/test/wrappers/CbdcDyDx.js)
    at listOnTimeout (internal/timers.js:554:17)
    at processTimers (internal/timers.js:497:7)

6) Contract: CbdcFulcrumV2
     "before each" hook for "getAPR returns current yearly rate (counting fee ie spreadMultiplier)":
   Error: Timeout of 300000ms exceeded. For async tests and hooks, ensure "done()" is called; if returning a Promise, ensure it resolves. (/home/ezulkosk/audits/Cbdc-contracts/test/wrappers/CbdcFulcrumV2.js)
    at listOnTimeout (internal/timers.js:554:17)
    at processTimers (internal/timers.js:497:7)

7) Contract: yxToken
     "before each" hook for "constructor set a underlying address":
   Error: Timeout of 300000ms exceeded. For async tests and hooks, ensure "done()" is called; if returning a Promise, ensure it resolves. (/home/ezulkosk/audits/Cbdc-contracts/test/wrappers/yxToken.js)
    at listOnTimeout (internal/timers.js:554:17)
    at processTimers (internal/timers.js:497:7)
```

Code Coverage

The code is generally well covered by the tests.

Update: Coverage of several wrappers and token contracts are reported as zero because mock files were tested instead of the primary contracts. We recommend ensuring that the tests exercise code in the primary contracts.

**Update as of commit `e09d4f5`: some tests fail due to timeouts which influenced coverage and test results. However the two contracts in scope, `CbdcTokenGovernance.sol` and `CbdcTokenHelper.sol` had full coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	8.65	4.88	9.47	8.71	
GST2Consumer.sol	0	0	0	0	… 38,39,40,42
GST2ConsumerV2.sol	100	100	100	100	
CbdcBatchConverter.sol	92	75	80	92	47,63
CbdcRebalancerV3_1.sol	38.71	16.67	25	37.5	… 106,111,116
CbdcTokenGovernance.sol	0	0	0	0	… 9,1170,1175
CbdcTokenHelper.sol	0	0	0	0	… 115,116,117
CbdcTokenV3_1.sol	0	0	0	0	… 213,222,231
CbdcViewHelper.sol	0	0	0	0	… 106,107,108
MinimalInitializableProxyFactory.sol	88.89	50	75	81.82	37,38
contracts/interfaces/	100	100	100	100	
AToken.sol	100	100	100	100	
AaveInterestRateStrategy.sol	100	100	100	100	
AaveInterestRateStrategyV2.sol	100	100	100	100	
AaveLendingPool.sol	100	100	100	100	
AaveLendingPoolCore.sol	100	100	100	100	
AaveLendingPoolProvider.sol	100	100	100	100	
AaveLendingPoolProviderV2.sol	100	100	100	100	
AaveLendingPoolV2.sol	100	100	100	100	
CERC20.sol	100	100	100	100	
CETH.sol	100	100	100	100	
CHAI.sol	100	100	100	100	
Comptroller.sol	100	100	100	100	
DataTypes.sol	**100**	**100**	**100**	**100**	
DyDx.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
DyDxStructs.sol	100	100	100	100	
GasToken.sol	100	100	100	100	
Gauge.sol	100	100	100	100	
GovernorAlpha.sol	100	100	100	100	
IAToken.sol	100	100	100	100	
IAdminUpgradeabilityProxy.sol	100	100	100	100	
IERC20Detailed.sol	100	100	100	100	
IERC20Mintable.sol	100	100	100	100	
IERC3156FlashBorrower.sol	100	100	100	100	
IERC3156FlashLender.sol	100	100	100	100	
IGovToken.sol	100	100	100	100	
IGovernorAlpha.sol	100	100	100	100	
ICbdcRebalancer.sol	100	100	100	100	
ICbdcRebalancerV3.sol	100	100	100	100	
ICbdcToken.sol	100	100	100	100	
ICbdcTokenGovernance.sol	100	100	100	100	
ICbdcTokenHelper.sol	100	100	100	100	
ICbdcTokenV3.sol	100	100	100	100	
ICbdcTokenV3_1.sol	100	100	100	100	
IInterestSetter.sol	100	100	100	100	
ILendingProtocol.sol	100	100	100	100	
IProxyAdmin.sol	100	100	100	100	
IStableDebtToken.sol	100	100	100	100	
IUniswapV2Router02.sol	100	100	100	100	
IVariableDebtToken.sol	100	100	100	100	
IWETH.sol	100	100	100	100	
Cbdc.sol	100	100	100	100	
CbdcController.sol	100	100	100	100	
PotLike.sol	100	100	100	100	
PriceOracle.sol	100	100	100	100	
RealUSDC.sol	100	100	100	100	
USDT.sol	100	100	100	100	
UniswapExchangeInterface.sol	100	100	100	100	
UniswapV2Router.sol	100	100	100	100	
Vester.sol	100	100	100	100	
VesterFactory.sol	100	100	100	100	
WhitePaperInterestRateModel.sol	100	100	100	100	
iERC20Fulcrum.sol	100	100	100	100	
contracts/libraries/	0	0	0	0	
DSMath.sol	0	0	0	0	20,23,29,68
contracts/mocks/	69.87	55.31	57.37	69.88	
AaveInterestRateStrategyMockV2.sol	75	100	80	75	14
AaveStableDebtTokenMock.sol	100	100	100	100	
AaveVariableDebtTokenMock.sol	100	100	100	100	
CHAIMock.sol	**30**	**0**	**16.67**	**30**	**… 30,31,35,36**
COMPMock.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
ComptrollerMock.sol	85.71	50	60	85.71	27
DAIMock.sol	100	100	100	100	
DyDxMock.sol	3.85	0	6.25	3.85	… 88,90,91,92
FlashLoanerMock.sol	100	100	100	100	
ForceSend.sol	0	100	0	0	5
GasTokenMock.sol	100	100	0	100	
CbdcMock.sol	0	100	0	0	11,12
CbdcNoConst.sol	94.12	70	90.91	94.29	196,197
CbdcControllerMock.sol	83.33	50	37.5	83.33	26
CbdcDSRNoConst.sol	12.9	7.14	8.33	12.5	… 159,160,164
CbdcDyDxNoConst.sol	60	50	54.55	61.11	… 140,155,183
CbdcTokenHelperMock.sol	40	100	50	40	16,17,18
CbdcTokenHelperNoConst.sol	100	83.33	100	100	
CbdcTokenV3_1Mock.sol	100	50	100	100	
CbdcTokenV3_1NoConst.sol	91.12	70.34	92.45	90.91	… 23,957,1034
InterestSetterMock.sol	0	100	0	0	10,13
PotLikeMock.sol	0	100	0	0	… 17,20,23,26
PriceOracleMock.sol	100	100	100	100	
USDCMock.sol	0	100	0	0	11,12
WETHMock.sol	65	37.5	57.14	65	… 55,56,70,71
WhitePaperMock.sol	60	100	20	60	19,22
aDAIMock.sol	100	50	100	100	
aDAIWrapperMock.sol	60	100	63.64	60	24,27,30,33
aaveInterestRateStrategyMock.sol	75	100	80	75	14
aaveLendingPoolCoreMock.sol	66.67	100	66.67	66.67	25,32,39,46
aaveLendingPoolMock.sol	23.08	100	28.57	23.08	… 46,47,48,49
aaveLendingPoolMockV2.sol	100	100	100	100	
aaveLendingPoolProviderMock.sol	100	100	100	100	
cDAIMock.sol	100	50	93.33	100	
cDAIWrapperMock.sol	84.62	50	78.57	84.62	37,59,65,68
cUSDCMock.sol	0	0	0	0	… 73,76,79,82
cUSDCWrapperMock.sol	0	0	0	0	… 77,80,86,89
cWETHMock.sol	88	50	75	88	60,63,84
iDAIMock.sol	47.06	37.5	16	47.06	… 117,124,130
iDAIWrapperMock.sol	78.95	50	78.57	78.95	34,43,49,52
CbdcBatchMock.sol	100	100	100	100	
CbdcNewBatchMock.sol	100	100	100	100	
yxDAIWrapperMock.sol	60	100	63.64	60	24,27,30,33
yxTokenMock.sol	85.71	50	71.43	85.71	29,33
yxTokenNoConst.sol	9.09	50	11.11	9.09	… 136,140,141
contracts/others/	0	0	0	0	
BasicMetaTransaction.sol	0	0	0	0	… 66,67,68,73
EIP712Base.sol	0	100	0	0	17,27,33,44
EIP712MetaTransaction.sol	0	0	0	0	… 65,66,71,73
contracts/tests/	**100**	**100**	**100**	**100**	
Foo.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/wrappers/	34.1	17.24	25.89	33.99	
Cbdc.sol	0	0	0	0	… 185,189,190
CbdcV2.sol	92.59	50	77.78	92.86	69,159
CbdcCompound.sol	97.83	62.5	90.91	97.87	217
CbdcCompoundETH.sol	97.56	50	90.91	97.62	204
CbdcCompoundV2.sol	22.22	18.75	18.18	21.62	… 178,179,183
CbdcDSR.sol	0	0	0	0	… 151,152,156
CbdcDyDx.sol	0	0	0	0	… 147,162,166
CbdcFulcrum.sol	0	0	0	0	… 145,146,150
CbdcFulcrumDisabled.sol	0	0	0	0	… 137,138,142
CbdcFulcrumV2.sol	0	0	0	0	… 137,138,142
yxToken.sol	0	0	0	0	… 136,140,141
All files	**44.84**	**29.2**	**42.39**	**44.6**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
cb50e8e3e594a81dc83e0cf49f617941a18d1af83d386943d1f20fa0dd200c86 ./contracts/GST2Consumer.sol
6341f0c902b0651922968bac1b1e5b8e797489faf7ef5e763a544a450d9532cc ./contracts/GST2ConsumerV2.sol
438cdf1986f293e4450935308634df9f2c3e46f962a40941b2e841f3a0f6bf26 ./contracts/CbdcBatchConverter.sol
56b6894d0659ffa4f19047613503696b87e31d342055b7a9617f62d6ed4e3e95 ./contracts/CbdcRebalancerV3_1.sol
b1ad8f1cb504167d4922fb1815f407d1f4e3c01ae0fc87c08a4131339ad2d0ec ./contracts/CbdcTokenGovernance.sol
27b8f77d310a8ca4e3c2ee7550c5aab56e2b904896a1e4138e64b5945ba6a817 ./contracts/CbdcTokenHelper.sol
21feafdfe57a4713f5c4a230740257949b2bbf691a39c1b3ca3e368e30dbed01 ./contracts/CbdcTokenV3_1.sol
600dfee96cf6c6fd38a218fb27928f5e6adf430616cf678ec9d3cd0479019076 ./contracts/CbdcViewHelper.sol
ffd751a32d9fb50ae7fd3b1724dc30556d83c33367b28a1ee66e4f56af9d65e7 ./contracts/Migrations.sol
09801d7f5658c723d314cf03a0878c8a84edfd9e3dc354d88e16e5ca5d5d1694 ./contracts/MinimalInitializableProxyFactory.sol
ae9c56710189a2541ee0164e4a01a0728e03aebdb4c1e60076f81fc343a5ae81 ./contracts/wrappers/Cbdc.sol
14ad3f5658df7c5dfc4ca3a49ba2063d859024774ab00975d1eb24fc46611c6a ./contracts/wrappers/CbdcV2.sol
042c9a2781853d5ed66b3d8d6201a973d5071230ca4fa23b7d06e82fd2f3f493 ./contracts/wrappers/CbdcCompound.sol
8edc23b10d723319b7e1828c9e2ee2d42bbd85127b30820f581421354a1f78e3 ./contracts/wrappers/CbdcCompoundETH.sol
516b144e5fb9f08b65235d21aa89705741d2e269ca5f170bc37cbb07cb0f87cd ./contracts/wrappers/CbdcCompoundV2.sol
dd032d7fcc9143dd79025fc615d28b7c382eafe24b0fe4e0fdfd8f9b723a223c ./contracts/wrappers/CbdcDSR.sol
de5c8e471accbb077ad6793e1c60683e67bb1575f415390d7e71b97b8fbeaf66 ./contracts/wrappers/CbdcDyDx.sol
452c9e06ec3a218229259b20a0ae26ac140d10e6ee3c6f3c8e1a1ee542732647 ./contracts/wrappers/CbdcFulcrum.sol
ed3e0a41a28490cbef139927143bf85ea776dcba90fdf0d88b652689e949f2f0 ./contracts/wrappers/CbdcFulcrumDisabled.sol
e9a689cfb6fb46cdf3644e9e52ec9e3f2576da8724439d8d05e7845724cbde60 ./contracts/wrappers/CbdcFulcrumV2.sol
fe50d4a334e03b70e55a8d159570070238e2a16d2213f2ae997d80cf398fe6b1 ./contracts/wrappers/yxToken.sol
1cab6221e40bebe7cfc8eb26bb049a6406b1c6d27b244fe33433e2ada194d306 ./contracts/tests/Foo.sol
1d53dfc9360c4975560a07e99bcb5c8882e0fc00a3c5fe23064631f051392356 ./contracts/others/BasicMetaTransaction.sol
304b03c570cb413afb28ed850aed112f0ef28b01850339e5c46f6479143873b7 ./contracts/others/EIP712Base.sol
513597938e062f74be0751429228d3b77d4a2e0fdee04510be9a23defd8c2ffc ./contracts/others/EIP712MetaTransaction.sol
7690baa9f464e5b9005b5ac3f32f68ad79f01ff69a57f3a96d58fd2f598dc67e ./contracts/mocks/aaveInterestRateStrategyMock.sol
95c589f05e2a9e3ab360dad60a39491a62489896b044ada67b1e24533b7e044f ./contracts/mocks/AaveInterestRateStrategyMockV2.sol
46b1695469eec18088c22842468a76cae83c429e135792e58af3cdd4f8684f97 ./contracts/mocks/aaveLendingPoolCoreMock.sol
4d6700a12609c826a559cf9111ec12c665e0c5a225027bb541c08cdea26b160e ./contracts/mocks/aaveLendingPoolMock.sol
e3e1e2656454004893c17c15c09aca9952b20bbdc53bb1de57496ab30f00b062 ./contracts/mocks/aaveLendingPoolMockV2.sol
a7ceeafde8ac95c36bb1d1756521a686d225b1e62a8ce7510d302b513f28e85d ./contracts/mocks/aaveLendingPoolProviderMock.sol
d61d046e28fc88d36fc490e86286e2f3e269718bcdc8b5615f7aef03307e37e4 ./contracts/mocks/AaveStableDebtTokenMock.sol
183cb180870733fa51cdc382cec5aa306bac91d14483e8d53581bdf121436279 ./contracts/mocks/AaveVariableDebtTokenMock.sol
084e2dee6aad484af4d2104331dd6c262815bc478fbb9a346cf43367482ed459 ./contracts/mocks/aDAIMock.sol
ebf4a51e421e210584e40e951f67efc1d8e5ee18584697d2dc05cd9887a3a02c ./contracts/mocks/aDAIWrapperMock.sol
d08719e992bb6088cbc198b50c4e1a0d5e506f126b4787b7fd484cb267500c32 ./contracts/mocks/cDAIMock.sol
```

78fbeef0d9d0c111d5252bd9da7fc5841b8ecc04002e834aaa304b130519988c ./contracts/mocks/cDAIWrapperMock.sol

f93b6b4f22b3eff48fa00a89f8ec8ef9b8dbc4f14ad79c39f00161233b7d1d18 ./contracts/mocks/CHAIMock.sol

ff3d0f6903ab36c587f9a6f56f682068e23278843b9a6775d85d984760a3b4d1 ./contracts/mocks/COMPMock.sol

693b69819db0b74712299c245bbb6d574aa1fa24cc7183153ad5f72b5908562d ./contracts/mocks/ComptrollerMock.sol

a9f670d48a6f1b757429f3bcb5e9e9682b88b87a73667ed1850e822a588f65c7 ./contracts/mocks/cUSDCMock.sol

cae54665d5c87a410b103621a8d92fb9fc0465f4ffcff2a5eea1055c0220b30e ./contracts/mocks/cUSDCWrapperMock.sol

49cd31818be45e5c50c1b414f979ebe121ee4539619ced940c00c99e65551a32 ./contracts/mocks/cWETHMock.sol

cebe2c9dadad843bc01fe5e188773248e779e061fb72d11c34eed9a3de0ac5ff ./contracts/mocks/DAIMock.sol

a14f262292dee9a5c072f40586ad1e98645efbefbfb1bb28fadd9852f2ea21e5 ./contracts/mocks/DyDxMock.sol

9f6fd266d87523ce293ab6be43c2f4707a88330d6d15ca5ae3334fb0298d9a4e ./contracts/mocks/FlashLoanerMock.sol

226302828e1e6801e388a780a7e1f5ec7c6d00f2a21d5b23e395b6fa03b5ac0e ./contracts/mocks/ForceSend.sol

c908417ccf62bf91587e749e21cbf25106c32e82d8d2df7f2ee4a1de5d6635c8 ./contracts/mocks/GasTokenMock.sol

6a7a8776097cb1874b7408e849a5cfc31acb46fede6b787ecf27b14303626587 ./contracts/mocks/iDAIMock.sol

fa63babd02cabca4b03ed47de2e46c7d21a287325a7a41c8b182e92f2670cbd9 ./contracts/mocks/iDAIWrapperMock.sol

20f1ed2a6763a04fca95f4618fb5807a5b7b205b5621cb217587693e33124770 ./contracts/mocks/CbdcNoConst.sol

eaf098d90370f307503d822006d69e4060a45acdd22863570e5f01df6f85b876 ./contracts/mocks/CbdcBatchMock.sol

4e47686c53566da4cf7d3429df9a737af1a8b547ca0eb1098f3a576a23f410fc ./contracts/mocks/CbdcControllerMock.sol

275f276629c16be57e3297e80093ee68d0584cbffac7cb6a0fd4a0d6d22577d7 ./contracts/mocks/CbdcDSRNoConst.sol

ad7b05f3e17e363ed602d74e0c2175fdbba25384f4a4e2f363cdb5943893f5b5 ./contracts/mocks/CbdcDyDxNoConst.sol

c9acfdaea6dde4913dc686b281a199eaafa3822f6becd2f8911d96555d947e2e ./contracts/mocks/CbdcMock.sol

d325c5366317657684d220d19c65379a6b594aa11bbdb1b4d0ad8ed570d8f286 ./contracts/mocks/CbdcNewBatchMock.sol

e064f2ac2b3fe18eca14cb83203a3b903df28d4663cd569f919f20d0d610f39b ./contracts/mocks/CbdcTokenHelperMock.sol

de425dd525723e6b7239210cdcbb20e51a6e1e2813a6c01f2bdeed073c56ecc0 ./contracts/mocks/CbdcTokenHelperNoConst.sol

af86c5013b5b82039049e573bf8a41874f8f230cd0ad11f097b1fcd9f47effec ./contracts/mocks/CbdcTokenV3_1Mock.sol

5cf7090f5710828c899450b35e3baf77a87b0ea8d34ce0b4723f1b765d2643fe ./contracts/mocks/CbdcTokenV3_1NoConst.sol

615bdc68fb899fc4589085acfe8216e3ca53ce149036bc426fcc05be411b3015 ./contracts/mocks/InterestSetterMock.sol

e4b8ae54d5bdcbd3537223fb96f2cdbdfe1861064ff22f9005911f04b950391e ./contracts/mocks/PotLikeMock.sol

31b93924b10ab3642fa618dc9275f9f3ac138795648aa92346a102e7819dc40b ./contracts/mocks/PriceOracleMock.sol

fcc07f5f3da7ad6330e5876745bb8040e260dc958bdea8dc41585fe2e0e4df23 ./contracts/mocks/USDCMock.sol

764e043e89425d5541862af2a927be5d468071a12cd0a59c2f9f40704f8b302b ./contracts/mocks/WETHMock.sol

88b2f7f39a492552df9a8162ca4963211a5db6972aa5abc13836524f9681ff17 ./contracts/mocks/WhitePaperMock.sol

7e79e9711c53374379691defac075de72a56f37e3d07e27ff7ac8ffda820b23a ./contracts/mocks/yxDAIWrapperMock.sol

1b194f50c9528c8e77434c765a94a8f97040153633c39c968a124453646bbee3 ./contracts/mocks/yxTokenMock.sol

5f91d951ded04bc114597b848acf070b2d9781dd2b283f17c0f5a697834d5f4e ./contracts/mocks/yxTokenNoConst.sol

36e8d3f881312f1575c1d73feed068768587ebef76e19a8c55e80c7d5ecf548c ./contracts/libraries/DSMath.sol

7947bc218c29bef6b9311ec3b0ba5883c6067d6fa191bcaeddaae400d3783aea ./contracts/interfaces/AaveInterestRateStrategy.sol

fb453193300a1ea84d35436536ee01b7cef2ad7eadd1829c57aa7840ae4994ba ./contracts/interfaces/AaveInterestRateStrategyV2.sol

c1b64db188c22aa2f8dd8f8fc664f163b53071cdd98c85d67ab5888acf0d63fb ./contracts/interfaces/AaveLendingPool.sol

d2ba6c9c8f02946bf98e53295e84b29c334bba2a3b9a755e78342e2621522419 ./contracts/interfaces/AaveLendingPoolCore.sol

1d3c1c096be8bbfb05392fd97c77d9d957dbb2f47b2a8d978da502e8bc8398e6 ./contracts/interfaces/AaveLendingPoolProvider.sol

77851eebeb0039af84466e76ff5c2067de12e3ba4e28983652e706da8691f5e0 ./contracts/interfaces/AaveLendingPoolProviderV2.sol

e6112b547d55f40705ef0d633707350ac4f391a165dd11438b7dcf31386c1061 ./contracts/interfaces/AaveLendingPoolV2.sol

42f8369de2db5026fbf056992ca219645d98f9a623274784ea5d1a779c92ad26 ./contracts/interfaces/AToken.sol

f22f7508591b8b41a13511c01e336416a772dda310b29d6df88de1b5b8d06854 ./contracts/interfaces/CERC20.sol

e4d92cd3688939509570b286100fd6d65b16eb2427b321af5f2bb50d87732e7d ./contracts/interfaces/CETH.sol

206de751b0486eaadccdf76fa95e2d5978be9ea190f1561f12c3413cfff16969 ./contracts/interfaces/CHAI.sol

d36649910a636ee1da75d0f33d71f5873b83b169a6d86c06fcdc6412c8e9828d ./contracts/interfaces/Comptroller.sol

dba1842d6936dcf06e65aff0ea9d10d7b2e987e531774d58488503d6f9b23f35 ./contracts/interfaces/DataTypes.sol

f9282a625866967b49f511894146d3bc8fe6a96f0467eeb39ff6a2df477d98c7 ./contracts/interfaces/DyDx.sol

9ddd041518883d7c8cf7e923c7446ef580bc43aa54db69cd2fd23f4b47be4649 ./contracts/interfaces/DyDxStructs.sol

c3f95d558bd27571e06cffd518760bfbcbcbc3df68c05e8db55516de38774229 ./contracts/interfaces/GasToken.sol

d3a6cb8c8bcf3312f169da866ae7b1c2aa430861e8c9796410fcaf8a31a65cd1 ./contracts/interfaces/Gauge.sol

07806c507c46dcecbac86a1b3d7e19ad350cce4912ae77b9bb2c97ee888ebbeb ./contracts/interfaces/GovernorAlpha.sol

1464b7d71602f83ad4ee283395aeea50951605765c46df2de968ba26b18b87b3 ./contracts/interfaces/IAdminUpgradeabilityProxy.sol

03fc731b1fba6162bb7bdb2041ed2e077f90a793e8f3f7c1e1d174dd24435473 ./contracts/interfaces/IAToken.sol

ff45c284cad657ecd2e97de49e6385ae8dad5acab43f66fcc249f6fb0b652da5 ./contracts/interfaces/Cbdc.sol

b13da4dcaee4a1cc3482baa39154b734a1d6c4d2e172035bf870e33b08043743 ./contracts/interfaces/CbdcController.sol

65660b683ee4701fc7a1307bef629d25c14486c6a313f1eb7c9b08248788dce3 ./contracts/interfaces/IERC20Detailed.sol

6356b102e82c77f72c68597645d8d31cc5ea05a78af3e88e48b645b7b6e419ba ./contracts/interfaces/iERC20Fulcrum.sol

b42481fd402344cedc5ab082aa415bc1df1f3082cd316dccc05ca00d1be4fd86 ./contracts/interfaces/IERC20Mintable.sol

7f4694524424d65aa60d313b51e931f8e96a2e450610afcf54978480d50d3e29 ./contracts/interfaces/IERC3156FlashBorrower.sol

99cda61bea419a5e9c66fa8659b0a5610694d50650ea6baf3bf15c72a78d3866 ./contracts/interfaces/IERC3156FlashLender.sol

c3144402bb42ded093e2d021d25589fb325bb3ea852eca20bfdcfea45e93d0b2 ./contracts/interfaces/IGovernorAlpha.sol

0252f8f3886f5ac56a520bb36ddffe1f791bd162955b96905f648adf1b6891fa ./contracts/interfaces/IGovToken.sol

587c4202daafdb6616abf906031e7e1bd1535a4d7738389b540f271b5b46292d ./contracts/interfaces/ICbdcRebalancer.sol

db81c6219c2a4cb02215a7093173b8a0c999833298490009b157b78007bcd110 ./contracts/interfaces/ICbdcRebalancerV3.sol

f14bf430e2e9ef517d54400de1b6eac9cee26c4a6ba2d5ff1ebc8791512c5ec8 ./contracts/interfaces/ICbdcToken.sol

7065f6cfbde2b05f345557a63ac932a48145803119c1df2d6f0d9d8780ab77de ./contracts/interfaces/ICbdcTokenGovernance.sol

9cb8659a552afc12fbbe93989d81b7f3bb688357a3750f709d87708db96310f3 ./contracts/interfaces/ICbdcTokenHelper.sol

106537974d5c921e415642cd9466409d9e13f0b7ef6d1cab498dd1aac18ef024 ./contracts/interfaces/ICbdcTokenV3.sol

30d9d400c05924dd61b8c647c5b563d088aec977db4b5acacf42170f9b30c384 ./contracts/interfaces/ICbdcTokenV3_1.sol

afd940f2f0f9aa927a3418f01e218962f3033aae5a468b5302b3d4f5b309d366 ./contracts/interfaces/IInterestSetter.sol

f3735c051754aaf8d305c94099640d58131454f2c63b2db01cfa27e5aef8810f ./contracts/interfaces/ILendingProtocol.sol

bb53d48dc5a9bdfd81792141702186fd14ce628b226e317f40e5df29425d8019 ./contracts/interfaces/IProxyAdmin.sol

69fd7ce938e4f8958b97e54f2b2bf975c5346878cb2f916f26bb917152402e7d ./contracts/interfaces/IStableDebtToken.sol

eb5736ae93253b39d8c1564eee8339ea63d08cd8b546bcd76c8fd2b39ab73c17 ./contracts/interfaces/IUniswapV2Router02.sol

5b10cf8281631b3377df2542c8b7da2a76b7b3fbfeaffb8e574827e953724d8a ./contracts/interfaces/IVariableDebtToken.sol

a9509ad47c77c28c299f6f2b64f3497fa5c32ce6158599edfe55582248236f19 ./contracts/interfaces/IWETH.sol

9f37dbe5f1e0698275b4c047a21f645244601a9545f7ba20279127d01b274a28 ./contracts/interfaces/PotLike.sol

7030da4cd7de8e1a0481c27db004afacd0133a6bb6427c5d7da8457f0b991286 ./contracts/interfaces/PriceOracle.sol

f750845cd5ffdfce07c8a52138b6c0a59f23944218734557c9f0275e2b0aaa8e ./contracts/interfaces/RealUSDC.sol

50099dc807351b99408b1df47a6cdd331823641f4b1fd252a579313e52a494de ./contracts/interfaces/UniswapExchangeInterface.sol

73465ebd1211ca589d042b95bf7ae2330c8022219e93c1a70d0a2d83f6bea779 ./contracts/interfaces/UniswapV2Router.sol

b4b1a5bbdba60b0b99e1e6f6311d5d899226af1f72781f5015f19d3bb910a629 ./contracts/interfaces/USDT.sol

690589027c7fa15807705073215e5c1725ace965b209ae52604b41b955051952 ./contracts/interfaces/Vester.sol

7ac6b52da475b0e86f18cd9b1ebbbecc31a047d2ca321db8ca8b22e73f6efe1c ./contracts/interfaces/VesterFactory.sol

db96470d5844ab22a99451dee8baced828f0a8614f5e1c4a2e7c21848f978a7e ./contracts/interfaces/WhitePaperInterestRateModel.sol

Tests

dc6239773c8bb05e00358c8ba93d3755c63d43f4ea2f2f5969fc9f86a45102b0 ./test/CbdcBatchConverter.js

a7878d3cf4eaec576594be595e00948b8757dc65072ef514c7528a2293a159b1 ./test/CbdcTokenV3_1.js

3f0b64e8b21a36f8ca0e268a739b76da6eddcf50dcf197ce2506fff3c04fb0fb ./test/MinimalInitializableProxyFactoryTest.js

011e9182887a9c4a67502cb272c759fd8d81f18ae8b87380e3bbb4ce21b3d12b ./test/wrappers/Cbdc.js

9d76ca81064fcb3a16584b81cc7d2559b2dda58abcece6ccd338e97d871a04d8 ./test/wrappers/CbdcAaveV2.js

b1c156694d1fe3073ee8181d0f5fe637d8f37e4a93c53d7ee3333586ff1625cd ./test/wrappers/CbdcCompound.js

2ae10a4aef755303aafee42c7ae6028cb2d137c5c136f6423c8e842f9a7d3f25 ./test/wrappers/CbdcCompoundETH.js

f6a29c23b832b3f7226ca0e7b8b9060bc28bc3bf1601b4364ad7e06685cb6843 ./test/wrappers/CbdcCompoundV2.js

b822cd7c853d87e409587c2598357a4a19279e9a6cfe6d6a6b461d7cdd07496c ./test/wrappers/CbdcDSR.js

cce2f1e6b0b6b4dc24d929878a9161108072720c09bc2846bb7e3dfc7b467197 ./test/wrappers/CbdcDyDx.js

1a45883869155b57c725857fa7461127b3ecb723425edaec77c476d0fab270b8 ./test/wrappers/CbdcFulcrum.js

8b71421f1664acfb5eb63da9d61ba508bbec76f69d3d7e36b05512d868470490 ./test/wrappers/CbdcFulcrumDisabled.js

00de4f2b89491398082fcbfb8a7db30b63b2da481248aee8ee810a5417d27cd9 ./test/wrappers/CbdcFulcrumV2.js

c7b7b4755e1faf8ae58a1014665a092de3d89fd4181a288571f723ed795bc7e8 ./test/wrappers/yxToken.js

Changelog

- 2021-12-10 - Initial report
- 2021-12-10 - Revised report based on commit 9732bc
- 2021-12-11 - Revised report based on commit c6fa71c
- 2021-12-13 - Revised report based on commit bcb6f09
- 2021-12-13 - Revised report based on commit a71a706
- 2021-12-14 - Revised report based on commit 64f22d0
- 2021-12-15 - Revised report based on commit fefd01d
- 2021-12-15 - Revised report based on commit 7d3b7e4
- 2021-12-15 - Revised report based on commit 93d3429
- 2021-12-16 - Revised report based on commit f9c02d1
- 2021-12-17 - Revised report based on commit 338ec24
- 2021-12-18 - Revised report based on commit 1b40261
- 2021-12-18 - Revised report based on commit bd40915
- 2021-12-19 - Revised report based on commit e09d4f5
- 2021-12-20 - Revised report based on commit b5fb299

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.