Quantstamp

Zoro Protocol

Executive Summary

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

Type	Lending Platform
Timeline	2024-01-08 through 2024-01-11
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Zoro Docs
Source Code	• https://github.com/zoro-protocol/zoro-protocol #da78678
Auditors	• Ibrahim Abouzied Auditing Engineer • Shih-Hung Wang Auditing Engineer • Hytham Farah Auditing Engineer

Documentation quality	High
Test quality	Medium
Total Findings	6 Fixed: 2 Acknowledged: 3 Mitigated: 1
High severity findings	0
Medium severity findings	0
Low severity findings	3 Fixed: 2 Acknowledged: 1
Undetermined severity findings	0
Informational findings	3 Acknowledged: 2 Mitigated: 1

Summary of Findings

Zoro Protocol is a decentralized lending protocol, a fork of Compound designed to run on zkSync Era. The changes included the removal of the GovernorAlpha and GovernorBravo contracts, modifying `transfer()` calls to a safer `call()` pattern, and calculating elapsed time with `block.timestamp` rather than `block.number`. The Zoro Protocol uses the ORO token rather than the COMP token for governance.

During the audit, we found issues pertaining to unchecked return values for transfers (ZORO-1), borrow rates reflecting old block times (ZORO-2), and privileged roles (ZORO-3). We also list some of the known issues with Compound (ZORO-6). We recommend addressing all issues before deployment.

Fix-Review Update: All of the issues were addressed or acknowledged. Note that ZORO-6 was addressed by changing the deployment scripts.

ID	DESCRIPTION	SEVERITY	STATUS
ZORO-1	**Potential of Locked Funds in `Maximillion`**	• Low	Fixed
ZORO-2	**Excessive Maximum Borrow Rate**	• Low	Fixed
ZORO-3	**Privileged Roles and Ownership**	• Low	Acknowledged
ZORO-4	`block.timestamp` **May Not Be Futureproof**	• Informational	Acknowledged
ZORO-5	**Incorrect Return Value of `getCompAddress()`**	• Informational	Acknowledged
ZORO-6	**Known Issues in the Compound V2 Protocol**	• Informational	Mitigated

Assessment Breakdown

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

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

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

1. Code review that includes the following
   1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   3. 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:
   1. 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.
   2. 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, clarity, 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.

Scope

Files Included

Repo: https://github.com/zoro-protocol/zoro-protocol(da78678d2ae33e2787a099f5de439c01c3a8e4d3) Files: contracts/*

Files Excluded

Repo: https://github.com/zoro-protocol/zoro-protocol(da78678d2ae33e2787a099f5de439c01c3a8e4d3) Files: zksync/contracts/Multicall3.sol

Findings

ZORO-1 Potential of Locked Funds in `Maximillion`

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `731ca28d3f8a4ce428b89dc9ee279cd58cf2d5c7`. The client provided the following explanation:

> Added a check to ensure that the `call()` succeeds to avoid loss of funds.

File(s) affected: `Maximillion.sol`

Description: The `transfer()` call for sending native tokens to `msg.sender` in the `Maximillion.repayBehalfExplicit()` function has been re-written to `.call()` to be compatible with zkSync and also as a best practice. However, the return value of the `call()` function is not checked to be `true`.

In case `msg.sender` reverts the transaction in its `fallback()` or the `receive()` function for any reason (e.g., developers forget to define a payable fallback function), the native tokens will be locked in the `Maximillion` contract permanently, as there is no other way to retrieve the funds.

Recommendation: Consider adding a check to ensure that the `call()` succeeds to avoid loss of funds, e.g.,

```
(bool success, ) = payable(msg.sender).call{value: amount}("");
require(success, "Refund failed")
```

ZORO-2 Excessive Maximum Borrow Rate

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `fbc79cebacd4355d379755191441f2c6920d0bf6`. The client provided the following explanation:

> Updated the `borrowRateMaxMantissa` to reflect the new interest accrual times.

File(s) affected: `CTokenInterfaces.sol`

Description: The `borrowRateMaxMantissa` is used as an upper bound for the borrowing rate and initialized to `0.0005e16`, or 0.0005% per block. However, this value was determined based on the L1 block times of roughly 12 seconds. With interest now accruing every second rather than every block, this upper bound allows for a far higher borrowing rate.

Recommendation: Update the `borrowRateMaxMantissa` to reflect the new interest accrual times. For example, `0.0005e16 / 12`.

ZORO-3 Privileged Roles and Ownership

• Low ⓘ Acknowledged

Update

Marked as "Mitigated" by the client. Addressed in: `dd662b87f14a5e86c79de21cab9cd88506c9ee22`. The client provided the following explanation:

> Ownership will be transferred to Gnosis Safe multisigs after deployment. The multisigs use a 3 of 5 signing strategy.

File(s) affected: `BaseJumpRateModelV2.sol`, `DAIInterestRateModelV3.sol`, `CToken.sol`, `CErc20.sol`, `CErc20Delegator.sol`, `Unitroller.sol`, `Comptroller.sol`

Description: The protocol owner and admin serve privileged roles and can control the configurations of the contract or execute specific functions. Therefore, users are exposed to the risk of being attacked if the privileged roles are compromised.

- The `owner` of `BaseJumpRateModelV2` or `DAIInterestRateModelV3` can:
  - Update the parameters of the interest rate model
- The `admin` of `CToken` can:
  - Set the pending admin of the contract
  - Set the `Comptroller` and interest rate model addresses
  - Set the reserve and reduce factors
  - Initialize the contract after creation
- The `admin` of `CErc20` can:
  - Sweep tokens other than the underlying tokens from the contract
  - Delegate the votes of the `COMP`-like underlying tokens to an arbitrary address
- The `admin` of `CErc20Delegator` can:
  - Set the implementation contract of this proxy contract
- The `admin` of `Unitroller` can:
  - Set the pending admin of the contract
  - Set the pending implementation of this proxy contract
- The `admin` of `Comptroller` can:
  - Set the price oracle, borrow cap guardian, and pause guardian addresses
  - Set the close factor, collateral factor, and borrow cap of a market
  - Set the liquidation incentive of the protocol
  - List new markets
  - Pause the mint and borrow functionality of a market
  - Pause the transfer of seize operations of the protocol
  - Assign the implementation contract of a `Unitroller` proxy
  - Calls `fixBadAccruals()` to adjust the users' accrued `COMP` token
    - This function was a fix introduced in Compound for Proposal 65 but should be irrelevant to the Zoro protocol

Recommendation: Consider documenting the risk and impact a compromised privileged role can cause on the protocol and inform the users in detail. As the privileged roles can be the single point of failure of the protocol, consider using a multi-sig or a timelock contract (as Compound) to mitigate the risk of being compromised or exploited.

ZORO-4 `block.timestamp` May Not Be Futureproof

● Informational ⓘ Acknowledged

Update

Marked as "Mitigated" by the client. Addressed in: `a6db65ed39f52c04067bdcd514d2aa631905290b`. The client provided the following explanation:

> The date of timestamp overflow has been documented in the codebase for future reference (date is in roughly 80 years).

File(s) affected: `Comptroller.sol`, `CToken.sol`

Description: The Zoro Protocol uses `block.timestamp` instead of `block.number`. However, there is a check in the `Comptroller._initializeMarket()` function which casts the value returned by `getBlockNumber()` as `uint32` and reverts if the value exceeds `32` bits. The timestamp currently takes `31` bits, which means that the protocol may experience denial of service in the future when the timestamp exceeds a value that can be represented by a number less than `32` bits.

Recommendation: Either make note of the date where the block timestamp will exceed `32` bits and warn users, or avoid casting the value returned by `getBlockNumber()` to anything that is `32` bits or less.

ZORO-5 Incorrect Return Value of `getCompAddress()`

● Informational ⓘ Acknowledged

Update

Marked as "Mitigated" by the client. Addressed in: `dd662b87f14a5e86c79de21cab9cd88506c9ee22`. The client provided the following explanation:

> This address will be upgraded to the correct one once the protocol token has been deployed. The upgrade will occur using the Compound V2 Unitroller proxy.

File(s) affected: `Comptroller.sol`

Description: The `Comptroller.getCompAddress()` function returns the hard-coded `COMP` token address deployed on the Ethereum mainnet instead of the governance token of the protocol.

Recommendation: If the built-in functionality of granting and claiming the governance tokens in the `Comptroller` is planned to be used in the protocol, consider changing the hard-coded address to a correct value or a variable that can be changed by the admin later.

ZORO-6 Known Issues in the Compound V2 Protocol

● Informational ⓘ Mitigated

Update

Marked as "Fixed" by the client. Addressed in: `5817858d4a26466ebf93e1044da10c7791f5ed36`, `f3e93f55953f41a85e38674486d65b129d76708b`, `f3e93f55953f41a85e38674486d65b129d76708b`, `a6493821126ac10a0f18cb2928410662f455a0b5`, `52ac894c461880ca5773e4bbc1d5d10f417dcadd`, `d4948d06ad7684b5e1eb649bb3538a64b652453e`. The client provided the following explanation:

> Updated the deploy process to ensure that new CTokens are not susceptible to the low liquidity vulnerability. The other risks of Compound V2 forks is considered acceptable protocol behavior.

File(s) affected: `contracts/*`

Description: The following is a non-exhaustive list of existing or known issues in the Compound V2 codebase:

1. Empty market vulnerability: An empty market (i.e., cToken with a total supply of 0) is vulnerable to exchange rate manipulation that may lead to loss of funds in other markets managed by the same Comptroller. See Compound's analysis on the Hundred Finance Incident for more details. This vulnerability can be triggered not only when a market is empty but also when an adversary controls the entire token supply of a market by, e.g., liquidating other users' positions.
2. Similar to the empty market vulnerability, if the underlying token of a market is compromised, the adversary might be able to drain the funds in other markets by inflating the underlying token balance in the cToken contract (and therefore the return value of `getCash()` and the exchange rate) with a low cost.
3. Some ERC-20 tokens are incompatible with the Compound V2 codebase and might cause issues when used as an underlying token. For example, tokens with transfer hooks (e.g., ERC-777), double-entry tokens (e.g., TUSD in the past), and tokens allowing balance changes without a transfer (e.g., rebasing tokens).
4. The `repayBorrowBehalf()` function in the `CEther` and `CErc20` contract allows a user to repay the entire borrow balance of another by specifying the `repayAmount` as `type(uint).max`. This could open an attack vector where the borrower front-runs the repayment transaction and increases the borrow balance, causing the repayer to spend more than expected.

5. In `CToken.mintFresh()`, the call to `comptroller.mintVerify()` is commented out, and the `comptroller.repayBorrowVerify()` function is never used in the codebase. Therefore, modifications to the `mintVerify()` or `repayBorrowVerify()` functions will not take any effect.
6. `Comptroller._supportMarket()` calls `isCToken()` on the provided `cToken` parameter but does not validate the return value as `true`. `Comptroller._setPriceOracle()` does not call `isPriceOracle()` on `newOracle` to check that the provided address implements a `PriceOracle`.
7. The `repayBorrow()`, `repayBorrowBehalf()`, `liquidateBorrow()`, and `mint()` functions in the `CEther` contract do not return a boolean, while the same functions in the `CErc20` contract do. Third-party contracts integrating with the `CEther` contract should be aware of such discrepancies.

Recommendation: Consider whether the above-known issues and risks are acceptable. If not, consider implementing code changes or procedures to mitigate the issues. A common mitigation for the empty market vulnerability is to mint a small number of cTokens and burn the tokens or transfer them to a trusted address before setting the collateral factor to non-zero.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – 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.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **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).

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

- `426...c56 ./contracts/EIP20NonStandardInterface.sol`
- `d7c...c9c ./contracts/ComptrollerInterface.sol`
- `d1d...4b3 ./contracts/Comptroller.sol`
- `ce2...3a8 ./contracts/CErc20.sol`
- `b69...f09 ./contracts/PriceOracle.sol`
- `118...daa ./contracts/ComptrollerG7.sol`
- `e04...bb1 ./contracts/CToken.sol`
- `971...686 ./contracts/CErc20Delegator.sol`
- `309...924 ./contracts/CErc20Delegate.sol`
- `8ab...cdd ./contracts/CDaiDelegate.sol`
- `dd2...3c5 ./contracts/ExponentialNoError.sol`
- `31e...b1d ./contracts/SafeMath.sol`
- `971...a60 ./contracts/ErrorReporter.sol`
- `c20...97a ./contracts/Unitroller.sol`
- `694...439 ./contracts/JumpRateModel.sol`
- `a64...a70 ./contracts/EIP20Interface.sol`
- `1c8...1fc ./contracts/InterestRateModel.sol`
- `e1c...a43 ./contracts/JumpRateModelV2.sol`
- `9c6...986 ./contracts/ComptrollerStorage.sol`

- `90e...521 ./contracts/WhitePaperInterestRateModel.sol`
- `99c...9cc ./contracts/Reservoir.sol`
- `142...39e ./contracts/CErc20Immutable.sol`
- `311...849 ./contracts/CEther.sol`
- `f35...43f ./contracts/DAIInterestRateModelV3.sol`
- `c7d...659 ./contracts/Maximillion.sol`
- `2db...a65 ./contracts/SimplePriceOracle.sol`
- `74b...213 ./contracts/BaseJumpRateModelV2.sol`
- `db8...2bb ./contracts/CTokenInterfaces.sol`
- `04f...190 ./contracts/Governance/Comp.sol`
- `7b1...0e5 ./contracts/Lens/CompoundLens.sol`

Tests

- `5c3...d9e ./tests/TimelockTest.js`
- `535...fd5 ./tests/Jest.js`
- `cb9...62a ./tests/CompilerTest.js`
- `1ce...ca2 ./tests/SpinaramaTest.js`
- `033...0a2 ./tests/Matchers.js`
- `550...8c8 ./tests/Errors.js`
- `346...b37 ./tests/MaximillionTest.js`
- `38c...bd9 ./tests/gasProfiler.js`
- `851...178 ./tests/Scenario.js`
- `19d...47c ./tests/PriceOracleProxyTest.js`
- `015...9f3 ./tests/Tokens/setComptrollerTest.js`
- `a62...04a ./tests/Tokens/mintAndRedeemCEtherTest.js`
- `09f...08f ./tests/Tokens/adminTest.js`
- `1c6...fb9 ./tests/Tokens/accrueInterestTest.js`
- `d37...367 ./tests/Tokens/setInterestRateModelTest.js`
- `2dd...64e ./tests/Tokens/compLikeTest.js`
- `d47...6fb ./tests/Tokens/borrowAndRepayTest.js`
- `5eb...13b ./tests/Tokens/borrowAndRepayCEtherTest.js`
- `33c...aac ./tests/Tokens/liquidateTest.js`
- `eea...dbb ./tests/Tokens/safeTokenTest.js`
- `85f...712 ./tests/Tokens/mintAndRedeemTest.js`
- `f9f...818 ./tests/Tokens/transferTest.js`
- `e3f...405 ./tests/Tokens/reservesTest.js`
- `981...850 ./tests/Tokens/cTokenTest.js`
- `67f...0c3 ./tests/Fuzz/CompWheelFuzzTest.js`
- `4dd...923 ./tests/Comptroller/adminTest.js`
- `7d8...ce1 ./tests/Comptroller/liquidateCalculateAmountSeizeTest.js`
- `224...050 ./tests/Comptroller/accountLiquidityTest.js`
- `2b9...648 ./tests/Comptroller/comptrollerTest.js`
- `ff2...5c4 ./tests/Comptroller/proxiedComptrollerV1Test.js`
- `96f...4d4 ./tests/Comptroller/unitrollerTest.js`
- `285...096 ./tests/Comptroller/assetsListTest.js`
- `e49...a60 ./tests/Comptroller/pauseGuardianTest.js`
- `c0e...eb9 ./tests/Governance/CompTest.js`
- `2a4...7b9 ./tests/Governance/CompScenarioTest.js`
- `777...46c ./tests/Governance/GovernorBravo/QueueTest.js`
- `d9f...866 ./tests/Governance/GovernorBravo/ProposeTest.js`
- `661...885 ./tests/Governance/GovernorBravo/CastVoteTest.js`
- `602...87f ./tests/Governance/GovernorBravo/StateTest.js`
- `5f5...fe3 ./tests/Governance/GovernorAlpha/QueueTest.js`
- `45f...4f2 ./tests/Governance/GovernorAlpha/ProposeTest.js`
- `10b...8a7 ./tests/Governance/GovernorAlpha/CastVoteTest.js`

- `b28...cb5 ./tests/Governance/GovernorAlpha/StateTest.js`
- `55f...49f ./tests/Flywheel/FlywheelTest.js`
- `94e...a9a ./tests/Flywheel/GasTest.js`
- `3ae...c20 ./tests/Lens/CompoundLensTest.js`
- `485...105 ./tests/Models/InterestRateModelTest.js`
- `99c...007 ./tests/Models/DAIInterestRateModelTest.js`
- `17f...001 ./tests/Utils/JS.js`
- `27f...daa ./tests/Utils/EIP712.js`
- `f89...9e0 ./tests/Utils/Ethereum.js`
- `71e...cf5 ./tests/Utils/Compound.js`
- `760...a94 ./tests/Utils/InfuraProxy.js`
- `cee...833 ./tests/Contracts/Const.sol`
- `29c...c67 ./tests/Contracts/MathHelpers.sol`
- `638...65b ./tests/Contracts/Fauceteer.sol`
- `678...19e ./tests/Contracts/WBTC.sol`
- `679...0a3 ./tests/Contracts/FeeToken.sol`
- `8c2...b0f ./tests/Contracts/FalseMarker.sol`
- `483...eed ./tests/Contracts/Structs.sol`
- `aaa...c2f ./tests/Contracts/ERC20.sol`
- `b49...4ee ./tests/Contracts/TetherInterface.sol`
- `3b9...376 ./tests/Contracts/ComptrollerHarness.sol`
- `4ff...c69 ./tests/Contracts/MockMCD.sol`
- `4db...3f2 ./tests/Contracts/CEtherHarness.sol`
- `455...136 ./tests/Contracts/PriceOracleProxy.sol`
- `d67...cc0 ./tests/Contracts/FixedPriceOracle.sol`
- `5ce...ee6 ./tests/Contracts/ComptrollerScenario.sol`
- `426...cba ./tests/Contracts/Counter.sol`
- `e33...7ee ./tests/Contracts/InterestRateModelHarness.sol`
- `f81...fed ./tests/Contracts/CompHarness.sol`
- `34b...ee7 ./tests/Contracts/EvilToken.sol`
- `f30...7f0 ./tests/Contracts/CErc20Harness.sol`
- `748...d81 ./tests/Contracts/FaucetToken.sol`

Automated Analysis

N/A

Test Suite Results

The test data was gathered by running:

1. `npx saddle compile`
2. `npx saddle test`

```
Saddle: running contract tests with jest...

(node:115615) Warning: Accessing non-existent property 'INVALID_ALT_NUMBER' of module exports inside
circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:115615) Warning: Accessing non-existent property 'INVALID_ALT_NUMBER' of module exports inside
circular dependency
(node:115615) Warning: Accessing non-existent property 'VERSION' of module exports inside circular
dependency
Jest args: {"_":
["test"],"ci":false,"detectLeaks":false,"detectOpenHandles":false,"errorOnDeprecated":false,"listTests":fal
se,"passWithNoTests":false,"runTestsByPath":false,"testLocationInResults":false,"maxConcurrency":5,"notifyM
ode":"failure-change","$0":"node_modules/.bin/saddle"}
Using network test Web3ProviderEngine
Setup in 349 ms
```

```
 PASS  tests/Flywheel/FlywheelTest.js (504.09s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 52 ms
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 37 ms
 PASS  tests/Tokens/borrowAndRepayTest.js (78.033s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 42 ms
 PASS  tests/Tokens/borrowAndRepayCEtherTest.js (39.452s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 43 ms
 PASS  tests/Lens/CompoundLensTest.js (37.35s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 43 ms
 PASS  tests/Models/DAIInterestRateModelTest.js (66.406s)
  ● Console

    console.warn node_modules/abortcontroller-polyfill/dist/polyfill-patch-fetch.js:520
      fetch() is not available, cannot install abortcontroller-polyfill

Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 49 ms
 PASS  tests/Tokens/reservesTest.js (45.921s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 48 ms
 PASS  tests/Comptroller/proxiedComptrollerV1Test.js (47.327s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 40 ms
 PASS  tests/Comptroller/pauseGuardianTest.js (40.371s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 44 ms
 PASS  tests/Comptroller/assetsListTest.js (163.733s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 40 ms
 PASS  tests/Models/InterestRateModelTest.js (9.442s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 41 ms
 PASS  tests/Comptroller/accountLiquidityTest.js (19.593s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 40 ms
 PASS  tests/Comptroller/unitrollerTest.js (15.211s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 39 ms
 PASS  tests/Tokens/accrueInterestTest.js (17.926s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 42 ms
 PASS  tests/PriceOracleProxyTest.js (119.158s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 43 ms
 PASS  tests/Tokens/setInterestRateModelTest.js (26.278s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 40 ms
 PASS  tests/Comptroller/adminTest.js
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 34 ms
```

```
 PASS  tests/Tokens/adminTest.js (25.91s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 42 ms
 PASS  tests/MaximillionTest.js (10.179s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 42 ms
 PASS  tests/Tokens/transferTest.js (10.6s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 47 ms
 PASS  tests/Tokens/setComptrollerTest.js (15.832s)
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 41 ms
 PASS  tests/Tokens/safeTokenTest.js (6.101s)

Test Suites: 2 skipped, 21 passed, 21 of 23 total
Tests:       34 skipped, 459 passed, 493 total
Snapshots:   0 total
Time:        1305.042s
Ran all test suites matching /test/i.
Teardown in 0 ms
Using network test Web3ProviderEngine
Setup in 40 ms
Teardown in 0 ms
```

Code Coverage

We were unable to gather test coverage. We recommend that the team assess their coverage to ensure that the testing is adequate before deployment.

Changelog

- 2024-01-12 - Initial report
- 2024-01-31 - Fix Review

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:
- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Quantstamp