Quantstamp Security Assessment Certificate

June 8th 2021 — Quantstamp Verified

Cozy Finance

This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	Lending Protocol
Auditors	Fayçal Lalidji, Security Auditor Joseph Xu, Technical R&D Advisor Sebastian Banescu, Senior Research Engineer
Timeline	2021-04-01 through 2021-05-06
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md Online Documentation
Documentation Quality	Medium
Test Quality	High

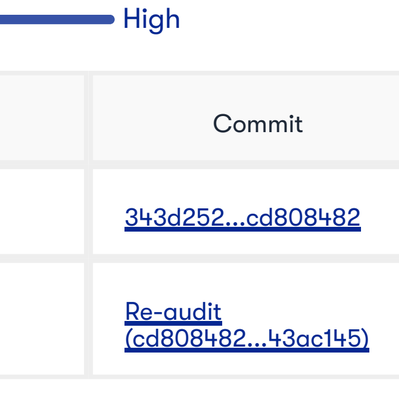
Source Code

Repository	Commit
cozy-protocol	343d252...cd808482
cozy-protocol	Re-audit (cd808482...43ac145)

Goals

- Assess the security and accuracy of the modification applied to the original Compound fork.
- Code adherence to the documentation and specification.
- Find issues that could allow attackers to drain user assets.
- Find issues that could lead to fund losses or freeze.

Total Issues	20 (8 Resolved)
High Risk Issues	1 (1 Resolved)
Medium Risk Issues	4 (4 Resolved)
Low Risk Issues	6 (1 Resolved)
Informational Risk Issues	8 (1 Resolved)
Undetermined Risk Issues	1 (1 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

Initial Audit:
Through reviewing the code, we found **20 potential issues** of various levels of severity. We recommend addressing the findings prior to deploying the smart contracts to the main network. We only audited a diff of Cozy protocol contracts using the following hashes `343d252...cd808482`. Every protection markets in the protocol will involve an external "trigger" contract deployed possibly by a third party. Quantstamp has not audited any trigger contracts. Therefore, Quantstamp cannot make statements on the actual user safety from interacting with the external trigger contract.
There are potential usages outside of the intended ones that may put certain user funds at risk. This is mainly due to the permission-less nature of the protocol that allows arbitrary protection market and trigger contract deployment. Having said that, to the extent that usages are limited to the original ones intended by the developers, this protocol should function as intended.
Re-audit:
All highlighted issues have been either fixed or acknowledged by the Cozy team in the re-audited diff (cd808482...43ac145). For all acknowledged issues we added the responses received from the Cozy team in the "Update" section below the "Recommendation". Please be aware that all newly introduced features (e.g. GovernorBravo) and related source code is considered out of the audit scope, the re-audit only focused on the fixes resulting from the issues descriptions.

ID	Description	Severity	Status
QSP-1	Front Running Attack	High	Fixed
QSP-2	Borrow Index Not Updated Correctly	Medium	Fixed
QSP-3	Stale Oracle Price	Medium	Fixed
QSP-4	Incorrect Max Inflation Value	Medium	Fixed
QSP-5	Problematic Violation of Checks Effects Interactions Pattern In `CToken.checkAndToggleTrigger()`	Medium	Fixed
QSP-6	Incorrect Quorum Value and Proposal Threshold	Low	Acknowledged
QSP-7	Incorrect Return Value	Low	Fixed
QSP-8	Loose Input Validation	Low	Acknowledged
QSP-9	No Way to Enforce How Funds Borrowed From the Protection Market Are Used	Low	Acknowledged
QSP-10	The Ability to Deploy Arbitrary Protection Contracts Makes Certain Families of Protection Markets Easy to Arbitrage	Low	Acknowledged
QSP-11	Protection Providers Can Be Allowed to Borrow	Low	Acknowledged
QSP-12	Malicious or Bad Trigger	Informational	Acknowledged
QSP-13	Allowance Double-Spend Exploit	Informational	Acknowledged
QSP-14	Unlocked Pragma	Informational	Fixed
QSP-15	Clone-and-Own	Informational	Acknowledged
QSP-16	Spam/Defacement	Informational	Acknowledged
QSP-17	Hack Reimbursed to Protection Seekers Twice	Informational	Acknowledged
QSP-18	Correlated Triggers	Informational	Acknowledged
QSP-19	Generated Interest Forgiveness	Informational	Acknowledged
QSP-20	Protection Markets With the Same Underlying Asset All Use the Same Interest Rate Model	Undetermined	Fixed

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:

- Slither v0.7.1

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Front Running Attack

Severity: *High Risk*

Status: Fixed

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

Description: A block is an ordered collection of transactions from all around the network. It's possible for the ordering of these transactions to manipulate the end result of a block.

- A miner attacker can take advantage of this by generating and moving transactions in a way that benefits themselves.
- Bot can be implemented to propagate a transaction with a higher gas price to manipulate the transaction orders in a block.

Exploit Scenario:

1. A `CToken.checkAndToggleTrigger()` transaction is propagated in the network.
2. A bot detects the transaction and pre-computes the outcome.
3. If the trigger is toggled, the attacker propagate a transaction to front-run the initial transaction with a higher gas price. The attacker deposits fund into a normal Cozy money market and borrows fund from the protection market to be triggered.
4. The initial transaction is executed last, allowing the attacker to withdraw his deposited collateral and keep the borrowed fund.

Recommendation: Front running attacks can be hard to work around. A possible solution will be track users borrow timestamps and trigger toggle time and forget the debt based on the time

difference or on both time difference and governance proposal.

Update: Fixed in https://github.com/Cozy-Finance/cozy-protocol/pull/43.

QSP-2 Borrow Index Not Updated Correctly

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Comptroller.sol`

Description: In `Comptroller.repayBorrowAllowed`, `updateCozyIndex(cToken, borrowIndex, false)` the Boolean input (3rd parameter) should be set to `true` to indicate that the borrow index should be updated.

Recommendation: Change the `updateCozyIndex` input, `updateBorrowMarket` to `true`.

Update: Fixed in https://github.com/Cozy-Finance/cozy-protocol/pull/34.

QSP-3 Stale Oracle Price

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `ChainlinkReporter.sol`, `EthPassThroughAggregator.sol`

Description: `EthPassThroughAggregator.getUnderlyingPrice` and `ChainlinkReporter.getUnderlyingPrice` call `AggregatorV3Interface.latestRoundData` without checking `startedAt` and `updatedAt` values, meaning that the lastest round returned can be obsolete since the Chainlink aggregators rely on external nodes to be updated once a request is submitted by one of the sponsors.

Recommendation: The `updatedAt` value returned by Chainlink aggregator must be validated. In case the price is stale, we recommend implementing a fallback oracle (in-house or a decentralized oracle like Uniswap).

Update: Fixed in https://github.com/Cozy-Finance/cozy-protocol/pull/44.

QSP-4 Incorrect Max Inflation Value

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Cozy.sol`

Description: The actual `maxInflationRatePercentMantissa` is set to `5e18 / (365.25*3600*24)`. When setting the `inflationRateMantissa`in `_setInflationRate`, its value cannot be higher than `totalSupply * maxInflationRatePercentMantissa / 10**18` which is equal to `1.584404391 × 10**20`. Supposing that the `inflationRateMantissa` is set to its maximum, after a year of inflation the total minted value will be equal to 5 billion tokens which is 500% of the initial total supply.

Recommendation: Reformulate the max inflation rate formulation to match the wished value of 5% of the `totalSupply`.

Update: Fixed in https://github.com/Cozy-Finance/cozy-protocol/pull/35.

QSP-5 Problematic Violation of Checks Effects Interactions Pattern In `CToken.checkAndToggleTrigger()`

Severity: *Medium Risk*

Status: Fixed

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

Description: In `CToken.checkAndToggleTrigger()`, `TriggerInterface(trigger).checkAndToggleTrigger()` is called before setting the contracts states (`CToken`, `Comproller`) to their final values. The `trigger` contract is an external non audited contract that should not be trusted unless verified, setting important state variables after calling the `trigger` contract can lead to possible reentrancy exploits.

Exploit Scenario: As an example a malicious trigger can execute some logic before returning `TriggerInterface(trigger).checkAndToggleTrigger()`. One possibility, is that the trigger profits from a protection market by borrowing funds before that the `CToken` and `Comproller` contracts state changes.

Recommendation: We recommend to the change the system state (set `isTriggered` to `true` and call `comproller._zeroOutCozySpeed`)before executing the external call. If the value returned by `TriggerInterface(trigger).checkAndToggleTrigger()` is `false` then restore the contracts to their initial state.

Update: Fixed in https://github.com/Cozy-Finance/cozy-protocol/pull/48.

QSP-6 Incorrect Quorum Value and Proposal Threshold

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `Governance/Cozy.sol`, `Governance/GovernorAlpha.sol`

Description: The governance process does not account for the fact that the COZY token is inflationary (though with a 4 year delay since deployment) in terms of the quorum and the proposal threshold. This is in contrast to the COMP token, which has a fixed supply.

Recommendation: `GovernorAlpha::quorumVotes()`, `GovernorAlpha::proposalThreshold()` should refer to COZY token's `totalSupply`. In addition, users need to be aware of the tokenomics of the COZY token, especially given the fact that inflation has been added.

Update: The system prohibits inflation for a minimum of 4 years, after which it may be implemented only via Governance vote. Because Governance must implement and manage inflation, it is reasonable to assume Governance is also responsible for managing quorum constants with respect to inflation.

QSP-7 Incorrect Return Value

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CToken.sol`

Description: The `CToken.approve()` function ignores the return value of `_approve(msg.sender, spender, amount)`; on `L206` and always returns `true`. In the current implementation, this is low risk, because `_approve()` can only return `true`. However, if the implementation of `_approve()` would at some point change such that it may also return false, then this issue would impose a significant risk.

Recommendation: `approve()` should return the same value returned by `_approve()`.

Update: Fixed in https://github.com/Cozy-Finance/cozy-protocol/pull/36.

QSP-8 Loose Input Validation

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `contracts/CErc20Immutable.sol`, `contracts/CToken.sol`, `contracts/CErc20Delegator.sol`, `contracts/CErc20.sol`, `contracts/CEther.sol`

Description: The following instances were encountered:

1. In `Comptroller.deployProtectionMarket`, the trigger check was updated from `trigger.isTriggered() to trigger.checkAndToggleTrigger()`, as the latter is the method used by the market.
2. Adding checks for the zero address is done more for historical reasons from when Solidity would not verify data length and would instead default to the zero address. As a result, it's not easy to accidentally pass in the zero address, so these checks are not necessary.
3. Same as 2.
4. These are passed in by the deployer at deploy time, so we the Cozy team are not worried about bad inputs.
5. These inputs do not need to be validated in `ProtectionMarketFactory.deployProtectionMarket`, because that method can only be called by the Comptroller, and the Comptroller validates those inputs.
6. A check has been added to `ChainlinkReporter` to verify that aggregator addresses return a valid price.

QSP-9 No Way to Enforce How Funds Borrowed From the Protection Market Are Used

Severity: *Low Risk*

Status: Acknowledged

Description: The protocol allows for multiple protection pools for each unique pair of trigger and asset. However, once those funds are borrowed from the protection pool, there is no way to enforce how those funds are used. It could be the case that the borrowed funds are used in a way such that they are not lost when the corresponding trigger is set to `true`. Therefore, at that point the borrower would have his debt set to 0 and could withdraw their deposit from the Cozy money market. This incentivizes protection seekers to borrow from the protection market with the trigger which is most likely to be set to `true` in the future. This strategy has the potential to prevent users who actually deposit their funds in riskier DeFi protocols to buy coverage (i.e. borrow from the corresponding protection market), because that market will probably have insufficient funds since every DeFi user will want to borrow from that pool.

Recommendation: This requires a redesign of the protocol to ask users affected by the hack to provide proof that their funds were affected, e.g. providing aDAI or yDAI as proof that funds were deposited in Aave and yearn respectively.

Update: The description above is considered part of the protocol design by Cozy team.

QSP-10 The Ability to Deploy Arbitrary Protection Contracts Makes Certain Families of Protection Markets Easy to Arbitrage

Severity: *Low Risk*

Status: Acknowledged

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

Description: The ability to deploy arbitrary protection contracts makes it extremely easy to arbitrage on the protection market contribution + borrow side. This makes it highly disadvantageous for users to be a depositor in a protection market.

Exploit Scenario: - Suppose that ETH price is at $2500 right now. An arbitrageur deploys two DAI protection markets, one with trigger when ETH < $2500 in 365 days and another with trigger when ETH >= $2500 in 365 days. These protection markets look innocuous enough to naive users, and attract deposits.

- The arbitrageur has some deposit in the non-protection market ETH. The arbitrageur uses these ETH as the collateral to borrow 10k from both DAI protection markets.
- After a year, one and only one of the DAI protection markets will trigger, allowing the arbitrageur to have one of the debt forgiven. The arbitrageur pays back the untriggered DAI protection market. The arbitrageur's PNL is +10k DAI (forgiven debt) - interest from the untriggered protection market + yield from putting 20k DAI to work for a year + 2*COZY subsidy.
- Depositors in either protection market face 50% chance of having their debt cancelled after a year in exchange for a modest interest income compared to other DeFi yields (note that the current Compound DAI interest rate model is 34% yield even at 100% utilization so the expected value from being a protection provider is negative).
- Note that this arbitrage did not require any malicious or faulty trigger contract.

Recommendation: Maybe this is a feature, not a bug. However, "smart money" allocators will eventually realize that the game is heavily tilted against depositing to the protection market. The

protocol economics needs to be re-worked so that the market is attractive enough to both deploy and borrow capital.

Update: The description above is considered part of the protocol design by Cozy team.

QSP-11 Protection Providers Can Be Allowed to Borrow

Severity: *Low Risk*

Status: Acknowledged

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

Description: If users are allowed to borrow using protection market deposits as collateral, the protocol can accumulate significant downside price risk on the underlying assets used in the protection market (users can easily deploy a protection market that acts as a synthetic long put option that trades against the protocol). This raises the insolvency risk of the protocol because it effectively acts as an option market-making bot that only sells volatility. Furthermore, liquidators may never be incentivized to liquidate the outstanding borrow even if it is heavily under-collateralized.

Exploit Scenario: - Assume that governance now allows protection markets at a collateral factor of 75%.

- Alice creates a protection market with ETH as the underlying with a trigger on ETH != $2500. Alice deposits 10 ETH @ $2500 into the protection market, uses the ETH as the collateral, and borrows 18750 USDC from the non-protection USDC market at the collateralization limit of 75%.
- The market moves a little and the protection market is triggered. The USDC debt is not forgiven, so it might get liquidated if ETH price moves against it. However, the ETH protection market is now locked (no new mint or borrow).
- Scenario 1: ETH price drops hard to $1750.
  - Alice's ETH in the protection market cannot be redeemed but she can market buy 10 ETH with 17500 USDC now. So Alice now gets her 10 ETH back and keeps 1250 USDC.
  - A liquidator must supply 17500 USDC plus interest to recover Alice's 10 ETH collateral (receives 10 ETH worth of Cozy-cETH tokens which are then redeemed for the underlying collateral). 10 ETH is the most that the liquidator can receive since there is not enough collateral to pay for the liquidation incentive.
- Scenario 2: If ETH price pumps hard to $3250.
  - Alice can pay back the USDC debt plus interest and her Cozy-cETH token to redeem the original 10 ETH.
- So the overall accounting is:
  - Scenario 1 - ETH drops:
    - Alice gets 1250 USDC.
    - Liquidator gets at most 10 ETH in exchange for 17500 USDC plus interest on 18750 USDC. No users would be willing to liquidate because it's cheaper to market buy 10 ETH.
    - If a liquidator does somehow step in, the protocol loses 1250 USDC but gets interest on 18750 USDC. The interest rate on the USDC debt must be quite high to balance the potential 1250 USDC loss, which makes it even less likely for a liquidator to appear.
  - Scenario 2 - ETH rises
    - Alice pays interest on 18750 USDC.
    - Protocol gets interest on 18750 USDC.
- Payout-wise, the protocol is doing the equivalent of selling a put option on ETH to Alice. While this is the intended functionality of a protection contract, the protocol cannot naturally take the opposing short ETH trade to hedge the risk.
- To offset the risk, the protocol would have to hope that another user would create and deposit into a similar USDC protection market to borrow ETH in the near future. An automated-market making bot would be able to adjust bids and offers to hedge these risks. The protocol might be able to achieve similar effect by adjusting the interest rate models of each market, but these must be done very frequently and calibrated to this specific case. It is highly unlikely that the current Compound interest rate model is suited to this purpose.

Recommendation: Do not allow borrows from the protection market unless enough research is conducted on the economic risk and market-making strategies.

Update: Collateral factor for Protection Markets will be kept at zero to remove this risk.

QSP-12 Malicious or Bad Trigger

Severity: *Informational*

Status: Acknowledged

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

Description: Anyone can be a trigger designer and deploy a protection market. There is no guarantee that the functions `isTriggered()` and `checkAndToggleTrigger()` will work as intended for any arbitrary trigger contract. Also there will probably be malicious trigger designers who will try to trick liquidity providers and steal their funds.
Examples:

- Consider a trigger contract that will always return `false` (or always return `true`) on `isTriggered()` and `checkAndToggleTrigger()`.
- `checkAndToggleTrigger()` always reverts, `checkAndToggleTrigger()` gives false positive, or `checkAndToggleTrigger()` gives false negative.
- Deposits into protection markets are highly susceptible to flashloan or Sybil attacks if the protection market contributor deploys a malicious trigger contract.

The main consequence is that `CToken::checkAndToggleTrigger()` may not properly toggle `isTriggered`, set `totalBorrow = 0`, or zero out COZY distribution speed for the affected protection markets. Essentially the trigger contract ends up serving as an untrusted oracle. Unless a trigger is audited by a reputable 3rd party it should not be trusted. Any method of the trigger could return false information.

Exploit Scenario: - There are two markets: USDC non-protection market and a malicious DAI protection market created by Mallory (e.g., the trigger contract returns true after a certain block number).

- Alice deposits 1000 DAI into the DAI protection market.
- Mallory flashloans to the USDC market, uses the USDC as the collateral to borrow from the malicious DAI protection market, triggers the trigger contract to get the DAI debt forgiven, withdraws the USDC collateral, swaps some of the DAI into USDC, and pays back the flashloan.
- Alice's deposit to the DAI protection market is now drained by Mallory.
- Note that similar attacks are possible without flashloan as long as the attacker has enough liquidity to post borrow collateral in the first place, is willing to repeat many times, or uses multiple addresses. This means that restricting same-block transactions or imposing limits on a single address might not be enough.

Recommendation: We recommend to clearly document and actively communicate to the end-users that may not understand the fact that Cozy triggers were not audited. Therefore, triggers cannot be trusted by default.

Also one or multiple of these measures can be considered:

- Protection market contributors must post deposits for each protection market created (locked up for a certain time or contributes to a “protection” fund for solvency).
- Protection market contributor whitelisting or a reputation system. However, reputation systems are far less desirable as they won’t fix the undercollateralized vault issue.
- Develop a verification process for trigger contracts.

Update: The description above is considered part of the protocol design by Cozy team.

QSP-13 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CToken.sol`

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

Exploit Scenario:

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.

Update: DAI, Compound and other popular ERC20s have the same issue and there have been few to no major real-world exploits of this issue

QSP-14 Unlocked Pragma

Severity: *Informational*

Status: Fixed

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.5.*`. 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".

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.

Update: Fixed in https://github.com/Cozy-Finance/cozy-protocol/pull/37

QSP-15 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `AggregatorV3Interface.sol`, `Strings.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.

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

Update: Care was taken in ensuring the copied files were unchanged except for adjusting the pragma.

QSP-16 Spam/Defacement

Severity: *Informational*

Status: Acknowledged

Description: A malicious entity who wants to deface the GUI/protocol might spam the protocol by creating many markets and triggers.

Update: Curation of markets is the responsibility of the app layer, outside of the protocol layer.

QSP-17 Hack Reimbursed to Protection Seekers Twice

Severity: *Informational*

Status: Acknowledged

Description: There have been hacks in the past where a snapshot was taken just before the hack and after migrating to a new token/contract, the DeFi platform reimbursed the affected end-users. In this case the end-users who also have funds borrowed from the corresponding protection market in Cozy would get double their funds back: once from Cozy and once from the hacked protocol who is reimbursing affected end-users. At the same time the protection providers would lose their funds.

Recommendation: We recommend to document this behaviour and actively informing the end users about all the nuances related with the usage of the protection markets, both as a protection

provider and as a protection seeker.

Update: The description above is considered part of the protocol design by Cozy team.

QSP-18 Correlated Triggers

Severity: *Informational*

Status: Acknowledged

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

Description: The protocol may accumulate fat-tail risk due to too many protection markets getting debt simultaneously cancelled through identical or correlated triggers.

- An attacker can deploy multiple instances of the same trigger contract. This can easily bypass the deployment check in `Comptroller::L779-782`.
- Even if there are no malicious trigger contracts, the protocol might inadvertently build up a large number of protection markets with identical (can be superficially different - e.g., UniswapV2 liquidity pool contract gets hacked vs Sushiswap liquidity pool contract gets hacked) or correlated triggers due to user demand.

Recommendation: Consider one or multiple of these measures:

- Protection market contributors must post deposits for each protection market created (locked up for a certain time or contributes to a "protection" fund for solvency).
- Protection market contributor whitelisting or a reputation system. However, reputation systems are far less desirable as they won't fix the undercollateralized vault issue.
- Develop a verification process for trigger contracts. In addition, conduct periodic reviews of protection markets to assess concentrated risk.

Update: The description above is considered part of the protocol design by Cozy team.

QSP-19 Generated Interest Forgiveness

Severity: *Informational*

Status: Acknowledged

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

Description: Forgiving the borrowed amount and the generated interest rate of the borrower seems unfair for the protection providers. The borrower should have to pay the interest rate even if the trigger is set to true. A possible behaviour from the protection seeker can be waiting for an indefinite period of time (assuming they are not liquidated) for the contract to be eventually triggered eventually, then their whole debt is forgotten.

Recommendation: We recommend to re-evaluate the debt forgiveness model because the current sequence of cash flow and debt forgiveness model are highly favorable to protection seekers (and conversely highly unfavorable to protection providers).

Update: If protection providers want to lock-in interest earned, they can withdraw from the protocol.

QSP-20 Protection Markets With the Same Underlying Asset All Use the Same Interest Rate Model

Severity: *Undetermined*

Status: Fixed

File(s) affected: `ProtectionMarketFactory.sol`

Description: Protection markets with the same underlying asset all use the same interest rate model, which is presumably the ones specifically optimized from the Compound money-market protocol as opposed to a protection market.

- The borrowing limit and interest rate should also be determined as a function of the trigger as opposed to using the same interest rate model for the underlying asset. This is because the protection contract should be pricing the risk of triggers as opposed to balancing the supply/demand for the underlying asset as is the case for Compound.
- The optimal interest rate model of each protection market is likely to differ substantially from Compound's own money-market interest rate model.
- As of now, the protection markets can end up acting as a binary option market-maker bot with severe mispricing, which opens it up to a whole host of arbitrage risks.

Recommendation: A deeper look at the economic model is required to ensure proper pricing of risk. The protocol seems to expect that a large number of arbitrageur will drive the market to an equilibrium in the long run. However, it is unclear whether the interest rate model parameters or the functional forms will be properly calibrated for each specific use-case to ensure eventual convergence to market equilibrium.

Update: Fixed in https://github.com/Cozy-Finance/cozy-protocol/pull/40.

Automated Analyses

Slither

Slither was run on the following selected files:

```
slither contracts/Comptroller.sol
slither contracts/CToken.sol
slither contracts/ChainlinkReporter.sol
slither contracts/EthPassThroughAggregator.sol
slither contracts/FixedPriceAggregator.sol
slither contracts/Governance/SimpleCozySpeedAugmentor.sol
slither contracts/ProtectionMarketFactory.sol
```

Almost all Slither findings were considered either false positive or out of scope. A single issue was considered and added to the audit findings.

Adherence to Specification

• The documentation just says: Cozy is a fork of Compound. However, the documentation does not specify which commit/version of Compound was forked. Compound has been evolving since the Cozy fork was performed. The last common commits of the 2 protocols are from December 2020. Since then Compound has merged several pull requests which included multiple features (e.g. Governor Bravo and sweepToken functionality), which are not present in Cozy. We recommend to indicate which version/commit of Compound was forked and indicate that the two protocols diverged from that commit onward.

Code Documentation

• Updating the contributors rewards one by one through `Comptroller.updateContributorRewards` is not a viable solution since `proposalMaxOperations` are set to 10 transactions, even if some sort of contract is implemented to call multiple update reward for multiple contributors in a single transaction, it can still reach high gas consumption (even block gas limit since the proposal will most likely contain other transactions). Also, relying on contributors to update their reward is challenging since if the the contributor distribution speed is reduced they will not be incentivized to call the update function by themselves. We recommend to update the documentation according the highlighted concerns.

Adherence to Best Practices

• **[FIXED]** Avoid cyclic dependencies. A cyclic dependency was introduced by importing the `CToken.sol` file in `ComptrollerInterface.sol`, because the former had already had an import for the latter. This import was added because of the input parameter of type `CToken` of the `_zeroOutCozySpeeds()` function. We recommend changing the type of this parameter to address and removing the import to avoid the cyclic dependency.

• **[ACKNOWLEDGED]** A strange behavior related to `uint256 public totalSupplyWhenTriggered` is documented on `CTokenInterfaces.sol` L128-132, we recommend to open an issue in ethereum/solidity repository to investigate this issue further.

-Instead of using magic numbers like 1584404390 in `Cozy.sol`, just write the mathematical formula for it in the code, the compiler will simplify it at compile time and it won't cost any gas or increase the contract byte-code size.

Test Results

Test Suite Results

Required command line to run the tests:

```
yarn
yarn test:local
yarn test:fork
```

```
 PASS  tests/Flywheel/FlywheelTest.js (12.759s)
 PASS  tests/Governance/CozyTest.js (19.306s)
 PASS  tests/TimelockTest.js (20.004s)
 PASS  tests/Triggers/integrationTest.js (22.149s)
 PASS  tests/SpinaramaTest.js (27.121s)
 PASS  tests/Triggers/triggerFunctionalityTest.js (59.591s)
 PASS  tests/Lens/CozyFinanceLensTest.js (55.069s)
 PASS  tests/Tokens/borrowAndRepayCEtherTest.js (82.555s)
 PASS  tests/Tokens/deployProtectionMarketTest.js (69.345s)
 PASS  tests/Governance/GovernorAlpha/CastVoteTest.js (6.516s)
 PASS  tests/Tokens/reservesTest.js (76.602s)
 PASS  tests/Tokens/mintAndRedeemTest.js (103.995s)
 PASS  tests/Comptroller/comptrollerTest.js (47.397s)
 PASS  tests/Tokens/accrueInterestTest.js (34.725s)
 PASS  tests/Tokens/liquidateTest.js (99.24s)
 PASS  tests/Governance/GovernorAlpha/ProposeTest.js
 PASS  tests/Tokens/borrowAndRepayTest.js (119.824s)
 PASS  tests/Tokens/cTokenTest.js (121.273s)
 PASS  tests/Governance/GovernorAlpha/StateTest.js (8.153s)
 PASS  tests/Models/InterestRateModelTest.js (17.42s)
 PASS  tests/Comptroller/accountLiquidityTest.js (27.303s)
 PASS  tests/Tokens/signatureTest.js (8.725s)
 PASS  tests/Comptroller/adminTest.js (6.807s)
 PASS  tests/Tokens/mintAndRedeemCEtherTest.js (23.59s)
 PASS  tests/Comptroller/pauseGuardianTest.js (53.92s)
 PASS  tests/Governance/CozyAdminTest.js (6.952s)
 PASS  tests/Comptroller/proxiedComptrollerV1Test.js (70.512s)
 PASS  tests/Models/DAIInterestRateModelTest.js (131.592s)
 PASS  tests/Governance/GovernorAlpha/QueueTest.js (6.637s)
 PASS  tests/Tokens/setInterestRateModelTest.js (35.503s)
 PASS  tests/CompilerTest.js
 PASS  tests/Comptroller/cozyTest.js
 PASS  tests/Tokens/safeTokenTest.js (10.581s)
 PASS  tests/Tokens/transferTest.js (12.366s)
 PASS  tests/Factory/protectionMarketFactoryTest.js (53.267s)
 PASS  tests/MaximillionTest.js (15.377s)
 PASS  tests/Comptroller/unitrollerTest.js
 PASS  tests/Comptroller/reserveGuardianTest.js (11.003s)
 PASS  tests/Tokens/cozyLikeTest.js (8.508s)
 PASS  tests/Tokens/adminTest.js (34.386s)
 PASS  tests/Governance/CozyScenarioTest.js (14.721s)
 PASS  tests/Flywheel/SubsidyTest.js (176.475s)
 PASS  tests/Comptroller/liquidateCalculateAmountSeizeTest.js (52.832s)
 PASS  tests/Scenarios/Governor/UpgradeScenTest.js (72.13s)
 PASS  tests/Scenarios/Governor/GuardianScenTest.js (86.168s)
 PASS  tests/Scenarios/Flywheel/ReservoirScenTest.js (90.14s)
 PASS  tests/Scenarios/HypotheticalAccountLiquidityScenTest.js (128.472s)
 PASS  tests/PriceOracleProxyTest.js (203.956s)
 PASS  tests/Comptroller/assetsListTest.js (247.45s)
 PASS  tests/Scenarios/Governor/ExecuteScenTest.js (173.407s)
 PASS  tests/Scenarios/Flywheel/CozySpeedScenTest.js (196.921s)
 PASS  tests/Scenarios/Governor/ProposeScenTest.js (206.705s)
 PASS  tests/Scenarios/Governor/DefeatScenTest.js (103.529s)
 PASS  tests/Scenarios/Governor/QueueScenTest.js (166.511s)
 PASS  tests/Scenarios/Governor/CancelScenTest.js (150.608s)
 PASS  tests/Scenarios/Governor/VoteScenTest.js (99.094s)
 PASS  tests/Scenarios/PriceOracleProxyScenTest.js (152.076s)
 PASS  tests/Scenarios/RedeemUnderlyingWBTCScenTest.js (441.213s)
 PASS  tests/Scenarios/RedeemUnderlyingEthScenTest.js (286.8s)
 PASS  tests/Scenarios/BreakLiquidateScenTest.js (84.795s)
 PASS  tests/Scenarios/Flywheel/GrantsScenTest.js (239.534s)
 PASS  tests/Scenarios/Flywheel/FlywheelScenTest.js (470.781s)
 PASS  tests/Scenarios/InKindLiquidationScenTest.js (486.394s)
 PASS  tests/Scenarios/ReduceReservesScenTest.js (249.415s)
 PASS  tests/Scenarios/TokenTransferScenTest.js (218.27s)
 PASS  tests/Scenarios/RedeemUnderlyingScenTest.js (377.839s)
 PASS  tests/Scenarios/ExchangeRateScenTest.js (130.102s)
 PASS  tests/Scenarios/BorrowBalanceScenTest.js (182.167s)
 PASS  tests/Scenarios/UnitrollerScenTest.js (42.875s)
 PASS  tests/Scenarios/EnterExitMarketsScenTest.js (290.098s)
 PASS  tests/Scenarios/RepayBorrowWBTCScenTest.js (403.558s)
 PASS  tests/Scenarios/CTokenAdminScenTest.js (110.88s)
 PASS  tests/Scenarios/Cozy/CozyScenTest.js (245.321s)
 PASS  tests/Scenarios/RepayBorrowEthScenTest.js (517.81s)
 PASS  tests/Scenarios/ReEntryScenTest.js (42.229s)
 PASS  tests/Scenarios/TetherScenTest.js
 PASS  tests/Scenarios/AddReservesScenTest.js (252.707s)
 PASS  tests/Scenarios/RedeemEthScenTest.js (188.053s)
 PASS  tests/Scenarios/BorrowEthScenTest.js (130.535s)
 PASS  tests/Scenarios/BorrowWBTCScenTest.js (153.144s)
 PASS  tests/Scenarios/MCDaiScenTest.js (5.083s)
 PASS  tests/Scenarios/RepayBorrowScenTest.js (361.942s)
 PASS  tests/Scenarios/SeizeScenTest.js (127.044s)
 PASS  tests/Scenarios/MintWBTCScenTest.js (210.978s)
 PASS  tests/Scenarios/MintEthScenTest.js (173.192s)
 PASS  tests/Scenarios/TimelockScenTest.js (263.769s)
 PASS  tests/Scenarios/BorrowScenTest.js (178.574s)
 PASS  tests/Scenarios/MintScenTest.js (211.144s)
 PASS  tests/Scenarios/RedeemWBTCScenTest.js (425.901s)
```

```
 PASS  tests/Scenarios/FeeScenTest.js (135.265s)
 PASS  tests/Scenarios/BorrowCapScenTest.js (225.764s)
 PASS  tests/Scenarios/RedeemScenTest.js (282.404s)

Test Suites: 2 skipped, 92 passed, 92 of 94 total
Tests:       105 skipped, 15 todo, 1000 passed, 1120 total
Snapshots:   0 total
Time:        1439.86s
Ran all test suites matching /test/i.
Teardown in 0 ms
Done in 1468.91s.

yarn test:fork
yarn run v1.22.5
$ export FORK=true && ./script/test -- tests/Oracle/*.js && export FORK=false
Skipping Scenario Rebuild (set rebuild=true to force)
Skipping Contract Rebuild (set rebuild=true to force)
Skipping Scenario Stub Rebuild (set rebuild=true to force)
Saddle: running contract tests with jest...

 PASS  tests/Oracle/EthPassThroughAggregatorTest.js (67.163s)
 PASS  tests/Oracle/ChainlinkReporterTest.js (214.132s)

Test Suites: 2 passed, 2 total
Tests:       30 passed, 30 total
Snapshots:   0 total
Time:        217.313s
Ran all test suites matching /test|tests\/Oracle\/ChainlinkReporterTest.js|tests\/Oracle\/EthPassThroughAggregatorTest.js/i.
Teardown in 0 ms
Done in 224.22s.
```

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

```
0dcdf9869ee8734cb914dc747e9e021878fd24ce5a01e9735ccfb3e15d739c86 ./contracts/ComptrollerG3.sol
1ee5569172243e0cb2936c40072d5f94a0acf4a9d58e7afc2805929aee17241f ./contracts/ComptrollerG6.sol
7a50083a1204facbc096a1432246b110fb4684dd5ab2498348d08eccf15d72fe ./contracts/SimplePriceOracle.sol
464060725d9e8676d399750468cc0b5888a6592727e77698d5b8d6e1e66d8648 ./contracts/Maximillion.sol
4a62dda9e3d7541fb4f2f797f02e7a6ad825b9c27df862ccaec97e5bdbf76c45 ./contracts/SafeMath.sol
7309d9dd25fcb643b05c0e8bf635cbde50006e054a1febded14ebea883df6557 ./contracts/CErc20.sol
028e8c04a2c36402a9cab4447c07a96182061ac4488de7451c45bb4e801ee9c9 ./contracts/Unitroller.sol
60ba3b5e8c17872931f33b7ef2696f01580d8a1469c30c5a1770f54ae5f038b5 ./contracts/CEther.sol
f32b804567fba1bde75fe8f2d1b56bd963583131ae492adf1d246aaffd94d0f2 ./contracts/Exponential.sol
df32a75a11e0b30ed5c280fb47529087ce786cbf8752c9032966eb6ea7cb7f2e ./contracts/InterestRateModel.sol
1b7f3d265aadeb270b545c17b07a566f9459ca813fad010503ad46168a161f15 ./contracts/CErc20Delegator.sol
5313ddd22e7a550892fc81ba01cb9a0910367cb51f354a1ffeb39020d2e7179c ./contracts/CToken.sol
2dea3eb913bde1201880cdf860187433dbcfb3a1d2da743b1055125b86ba6e6e ./contracts/CCozyLikeDelegate.sol
169bffcdf2b1524ccfdc605053c6842f5e15dab678f2f205a1f1d5cc8b8f9a84 ./contracts/ProtectionMarketFactory.sol
487085d25f715edcdb7e056d207e5f566b45a769969c23d180a4642a00deba6f ./contracts/JumpRateModel.sol
ba1347838ef7389b00412bf6edec36ac77b852297ce6c83cee54cf4095d155b0 ./contracts/ChainlinkReporter.sol
f635b44360fa7b9c88173e97a3dee4ec7334a074efb155f4dbee90c841070985 ./contracts/EIP20NonStandardInterface.sol
b4abec6c6fbfa2d694a7802b3a81e9b003cd6e26589f50cc4d0f233c7562d249 ./contracts/ComptrollerStorage.sol
a6fcb9cbd4df0ec987ad7b8433ea66766a862cc422cb8e726c017b751484c02b ./contracts/ComptrollerG1.sol
853993f5404a660a8ed184d6a1bc60b28318443f521eec6bcf2fc17fce7136bd ./contracts/AggregatorV3Interface.sol
6c7e70282d946b9e1f5665c1ee8732ce036706248b8dcde96986df4956b18344 ./contracts/CTokenInterfaces.sol
6d7439071cc4f63cf654f9f79a956c386d1470f5dc64a589d97b1dc24225ffd2 ./contracts/ComptrollerG4.sol
87b8236e40dd0bbb46b008e58e91024e59e8d1c8e47a543a5a3713a674486109 ./contracts/TriggerInterface.sol
81e9004c6588c974ad4ff44a09d632a96f31a1e6944ebe26845f163740ce4e49 ./contracts/CErc20Delegate.sol
4c317ad6f986f13ce8b85f51b5e9533b423a05534a1aa4ff1aa2db100dadba60 ./contracts/PriceOracle.sol
d7f9900c77e819119ef3b478b76eafe05b255f7d11ee3026ee71b8a9b5b985ab ./contracts/EthPassThroughAggregator.sol
6df3c432f603a0d2487abe0f2b64a24bb9edde6a16da76eb7f925034f68da3f7 ./contracts/LegacyInterestRateModel.sol
48443aee9451696f46a20599886828de7baa7aaa5de3d859eeddb7220cff38f8 ./contracts/DAIInterestRateModelV3.sol
46b8cd1659813c7d61c715f886224d97d86e1ef78b6e46ef0d0067c0220bec8a ./contracts/ExponentialNoError.sol
58ad312819004ce5a3a1e6364b4bd46e0e8c6cd165e839d4f0330f2862053d50 ./contracts/CarefulMath.sol
61f0714df8f7a426bc35e8465700eb547ccd4ceefa0d8baa9d78f5b2b7df6221 ./contracts/Timelock.sol
9871e1cbdf42ad39b050daabd72156fcd0e347cb6018ab90d417c5fdb3888dfc ./contracts/WhitePaperInterestRateModel.sol
6c029ec634523401b16b39483c9197baf4142e24327956ac6c2ecab048a3428f ./contracts/ErrorReporter.sol
cb198def3d351678363c0aeb578acf93434b10e6e11341681d258c29fa0baa66 ./contracts/BaseJumpRateModelV2.sol
63168b334dfe639b577db0ae95bc4931fa19948f2908b60340858644575d6a57 ./contracts/Reservoir.sol
52be04c8a1f7ff15eb50ced60c4b6f80275dd17fac16a08bd721c4cbe4b5c4bb ./contracts/Comptroller.sol
f86cad70cf5bc9956330bd3a6aa2da3e3b54aba0ac06823fb11620363a9c58b3 ./contracts/FixedPriceAggregator.sol
98bf601fab9eb9df0cc9a8d98eab4efa0a1efca3696775a1fc42a8428d2137ef ./contracts/EIP20Interface.sol
076162ceef9d6fc4085670569ae2e52278cdd8d4d7c16698764c054034c91af3 ./contracts/ComptrollerG5.sol
74cb41701a022da4a860c98f6df6c056aa048a6fa9cca14fee5cfb0711a297fe ./contracts/Strings.sol
9bcf6c0d29d1f46c0027e0bcf0c6fe167c30cf52a2b0e0487cf8cda1b3c3dd65 ./contracts/CErc20Immutable.sol
f62747b00b6dc5896232b02218fc7c11de53401ab4cab6b2b3206608903d1d6c ./contracts/ComptrollerG2.sol
1cbafdbb3511aa8bf27df9ae98d8f569b188a17e2064375d6cc07685cb16c2a4 ./contracts/JumpRateModelV2.sol
89c8e66051a2c152786b4da1ef2f30bc33034e9e54c778ff8649ea847bc81610 ./contracts/LegacyJumpRateModelV2.sol
```

4565ecbdc678773da709c4bd2d82870b3ce16a26f626508bc4b193fb1256642f ./contracts/ComptrollerInterface.sol

1d5ccf59e55a950a1a1d7272d260598a59caf627e11a92cc1df484fb620d008a ./contracts/Lens/CozyFinanceLens.sol

959369803ab66ce310e74758b1e8a2e25c46566c8b6a3c99903a923cebec0eec ./contracts/Governance/CozySpeedAugmentorInterface.sol

466e21131f341b4575c1081af6f14a99b2f91b6fe456a0caeec080b84a7b364f ./contracts/Governance/SimpleCozySpeedAugmentor.sol

603f2c21bcf2b98fc1f2f31a0da4d85d3690a9b05777eb5916631c5764f15ace ./contracts/Governance/Cozy.sol

fc0d95c04db4af75a16ecd5b66306f9c85b924063dda14940eb69ed191a7d3be ./contracts/Governance/GovernorAlpha.sol

Tests

2c6a9ed651f3d6d8fa2a7238522f70322da2746f0f825c0053fadd24103c392d ./tests/PriceOracleProxyTest.js

abff7cc7dd45170de5329d9d83b9614ed346d296af5da2cd3a5e6698f0a1cdaa ./tests/gasProfiler.js

2564f58c1c3ec17dad1a45d98873d7a70cf3a345f1896357e7ad95a700a2522a ./tests/TimelockTest.js

5bf89ac9f57bd8bc8a5347f0eecf0ab2aae2a8b9414819a527890a99b395d6c6 ./tests/Scenario.js

84f8dabf71cee55e57d33fffe4bee7a04a036d261de14f3edf27c4263307748b ./tests/Errors.js

0e72038a2c7da5d39c01fc39b7d6a3ad48fa5a8868061a8a185bf04541e0d94b ./tests/Jest.js

613c37b04494d53220a9ca7177d65d9d1299b67f04da34e9512a4f3a2fe5ca67 ./tests/CompilerTest.js

44da73acf2352ad032a9f138d9bf5c498313e7750299bb8ca96099d3bd54ef92 ./tests/MaximillionTest.js

716deea1882286fef564cf87a27543f5b080c6d170a5249ed8ace9d73066530c ./tests/Matchers.js

4f16038d441c2f10684f93e38f2e80c75b394cacbaf3fdc5c87d002577f6ca8d ./tests/SpinaramaTest.js

c919c8648a6e8192074c08cd84d3d65121498cb602e064da5e348eda4fd3c0b8 ./tests/Lens/CozyFinanceLensTest.js

47b3f9ec1cc2fc02eb78ea9063380c4b1703981090ac919a5cbef6e4b05f7567 ./tests/Contracts/MockMCD.sol

8790cc0a9b4bba342cd5b3b419c04d68534d16775c18380f757ba94237ee428e ./tests/Contracts/FalseMarker.sol

ed8f57cb3fdf64df86ffc84d9dd8e4993a7b5e4fdbc7e423554b1188b360fb3d ./tests/Contracts/TetherInterface.sol

a32221ae03086b3fcaf419658b54b39e9859792368bf37282d6b72de05a5eab2 ./tests/Contracts/FeeToken.sol

91ab74ab073431f7af8529dcf05491be1fef5a77dc41371ba7c43034c28d12b8 ./tests/Contracts/CErc20Harness.sol

6a5a61a66ca97df13ec8fa5c8e46c7e931247b846ec385cae3cf10c4693617b8 ./tests/Contracts/CozyHarness.sol

5eec453b27f7ea3e1ef41f0a7b3df911757f86c178ba6d2fa787be47f033c8e7 ./tests/Contracts/PriceOracleProxy.sol

1d1301f8fcd7cdb53a19c86ba8fdee62d2151d70520db584845292bc95fce5d5 ./tests/Contracts/ComptrollerScenarioG1.sol

11ea776c2c22c1ac080d1553eb662ae670f6b85d2b3eb421f420da49975b2361 ./tests/Contracts/MathHelpers.sol

454b0ccb35c602b65b0cf4c654b3815dd7262f07ed2190fb05d39dd22b09de97 ./tests/Contracts/TimelockHarness.sol

320202fc02053cd9d87b60484f477933a346fd5853b07a6baafd0792850f1256 ./tests/Contracts/Fauceteer.sol

34c08bbd27eec0529192041a186dbc3141da67d2db3c53e0e8dcb8af7169ca55 ./tests/Contracts/MockTrigger.sol

3ab1cd5e890251664a03165924ace36b2196231175c1fd6707d26378b68b7039 ./tests/Contracts/EvilToken.sol

8d8ca90dba2265f275f83f0230cf83a47279d1d1e9cd9afdf8fce4b33daec3bd ./tests/Contracts/ERC20.sol

6f7d9dd6f9b47d634e374d97fe7adf8f57f4dac66f19d07f49477aca3a64e50c ./tests/Contracts/FixedPriceOracle.sol

bb90b5ffa06632d1d4d0308044d2e07b3d949ae3da40affc3b7a05121cb7f446 ./tests/Contracts/Counter.sol

2db8c0eea02d6cd64ecc9385138ac6d6b41ba376de3354f82ddc1e77b1188b41 ./tests/Contracts/ComptrollerScenarioG6.sol

31814663bf356d4224f53c316478a638fe514aacc89e40eb9e983553e2b1c3a3 ./tests/Contracts/ComptrollerScenarioG4.sol

3d3ce38f8c728429801c5e73d01327da217620a356af0c48dbdd1419f2bcb32d ./tests/Contracts/ComptrollerScenarioG2.sol

1a82e5c83f57b99109dd0c1c720a975d0bb32226b2fc3268102a0a26f5a14152 ./tests/Contracts/ComptrollerHarness.sol

14cfc9074073debd0b339ff388f3a56277b821cbb2f6372b782b86410a2257c5 ./tests/Contracts/WBTC.sol

4fd61d6ac68ad3a899239c8a85ecd5e486585ef919271a595f76db476f5de808 ./tests/Contracts/ComptrollerScenarioG3.sol

54885e49fb03f660fd6dddd17e7a55cb7934ced2bf47a5667c12142cc208d6f8 ./tests/Contracts/GovernorAlphaHarness.sol

db32ebec3a8b7bb4da2d3b15ea216853cb02a8677a9cb854942096edab9705e2 ./tests/Contracts/CEtherHarness.sol

b1361c2a5b5df7b72bac922df2eacddf7f8c4c70c2b0099bc037abdce3d91d7b ./tests/Contracts/FaucetToken.sol

89b9e212c4b93d6e69a49e4a559e43d16897b863a2682c40d14e698fd55e0c82 ./tests/Contracts/InterestRateModelHarness.sol

1cbc304fc1eeb33a0408e6839440dff49a2913644a1d9009de3517b04cf726c4 ./tests/Contracts/Const.sol

e4338d879863a46e41e18f00f6caf81f22a671bb566d659006037d06cae1c98e ./tests/Contracts/Structs.sol

a74ce3ad819fb13330e46b64cd3276c1dcee1345e689017f9c53f4d8873742db ./tests/Contracts/ComptrollerScenario.sol

2beb4952b38fa1c0af1bd2f8bfd02ab33ee6a631c266973791eb15eab721ac8a ./tests/Contracts/ComptrollerScenarioG5.sol

fce03f5eed41883a9de6762adc8e24fc8e4122ca715f90997be4a3013d82a50d ./tests/Utils/Cozy.js

fed8cc489cb782ad51712b1d1158c6ec97f1dbde804208575981ad0abefcaae7 ./tests/Utils/InfuraProxy.js

8c5dc04fbc652293fa786f34207cccd1b5d19b952559bf27a499778e3f39ac6d ./tests/Utils/Ethereum.js

6501f6a93932ae2ca6aa00cd43aea49b36d807c5d4177817775cac7c3abc511b ./tests/Utils/JS.js

2935e4ded3e48133514f3099827cae1f31d3d313949a2f085e09adeca2606249 ./tests/Utils/EIP712.js

a2a133a94b3100ed627ba1f4750701311a48244cdb584e79ab9ff06b1fd84b01 ./tests/Governance/CozyScenarioTest.js

f58ad83e24af07016908c2029371a234750d453f6942bc2f99d27ae12c13973d ./tests/Governance/CozyAdminTest.js

d1213be08ecff51fcbbdda731f25b1d14690238aaa44347b128c988a65b24157 ./tests/Governance/CozyTest.js

a8a323738d130771c5e7cf672f33c976805aaf1a3510ee07fbfcb1d607f9f86c ./tests/Governance/GovernorAlpha/StateTest.js

5b8e9bc1e1563f5e1f10b092d23c69a491be46c77b52e15feb042c9b1b8c9a37 ./tests/Governance/GovernorAlpha/QueueTest.js

fd8c255d0ac9e713f7b3d51de61fefeed84cf587366a3a7698caf4a996b299ea ./tests/Governance/GovernorAlpha/ProposeTest.js

31ac728dbd77c14470f66742a77a27fc93932d63e852fc7d84f928c9b3ceedc8 ./tests/Governance/GovernorAlpha/CastVoteTest.js

8e8b23d890c2c95bbc6adec14363a19f9d82dd3fa989a8ce3641e90b5fcb4b62 ./tests/Scenarios/RepayBorrowScenTest.js

9ba1859b1e2341272c60a134855b585b9044d3b98d60e4cbbad571fe7423effc ./tests/Scenarios/CTokenAdminScenTest.js

506be5485394cb2c9bbc6f6bb6cc45b234a6c352172577706b27d1a7de4f4c9f ./tests/Scenarios/RedeemUnderlyingScenTest.js

ecfbedea3ca6e97266b4e76555ec6f7705628055998a3bc7f7051039292a067a ./tests/Scenarios/RedeemUnderlyingWBTCScenTest.js

7e6e76b14ed1fcf84ea6ac065be86fe0392cd2ac56851b5dc13ba9d7e6a37334 ./tests/Scenarios/BorrowScenTest.js

e3523f04ddfd19a14a44f74f32dd77305e06414af2e0ba1749b00c258b00ea87 ./tests/Scenarios/ExchangeRateScenTest.js

4c716c17c8d6d607621dd117900898731e9380df408ec22a1c141bcd7ec4965e ./tests/Scenarios/FeeScenTest.js

48966575141a703b0b5ffae7883627768eb63fbf15deedff9446fb3be607b0ee ./tests/Scenarios/RepayBorrowWBTCScenTest.js

16b28c43b7e03d0940111656945db3b1053c2753a623333ebfd85e81dfba4b1c ./tests/Scenarios/HypotheticalAccountLiquidityScenTest.js

b37e241c41fe97f45361a7d135afb2c699fccb565ecd2abf9d32ef57b50c0562 ./tests/Scenarios/BreakLiquidateScenTest.js

be689993bebc216c4cac9781ae286bf810aa34c793d8d743c53945c787d3ebd9 ./tests/Scenarios/EnterExitMarketsScenTest.js

e08db9fbdfd99a4b7704073b2cc64dcc7a18371ff0ec37723decdc7df5cefd90 ./tests/Scenarios/RedeemUnderlyingEthScenTest.js

a05ea0319b7966741c6a4944680ff5b7586132c5bca1b649685a9d1f0a97dcf9 ./tests/Scenarios/RepayBorrowEthScenTest.js

fbebcc9776712f53927fda86b2f86093e6b749f4602e31630dfb04462d30cd3c ./tests/Scenarios/BorrowEthScenTest.js

b3e59040b0087633e9f66dc4259d1d4fd5a04e4cfb76bb877713f8c830e9c690 ./tests/Scenarios/MintEthScenTest.js

9462f13e5d02224092386a00d92d261bb805079c1131fe2d1ca159d87a03d30a ./tests/Scenarios/BorrowBalanceScenTest.js

e37a817659914f87330a3347a534a4b42aa98ee8307f8f4e4ead02f3f4c0c639 ./tests/Scenarios/RedeemScenTest.js

3f8068cd66e6d3dd9e483cabc896690dacc3050446d97c85bcba37ad4524d9a5 ./tests/Scenarios/AddReservesScenTest.js

76bdb38fdec13324d65e2e22d5a51cc11971e92d29f26f3671143151e6788955 ./tests/Scenarios/TetherScenTest.js

c7889c9279fe003850a17fcb8a14f16357af221b522d8163decd38908e70ef68 ./tests/Scenarios/MintScenTest.js

13f66b96a6e1ef1f0150a609c9a841fd01ce62493f6dfda92a6af821a218b6d8 ./tests/Scenarios/MCDaiScenTest.js

4bab260de71fdf7f22d7419ee041e68ecfe68c245e0bfe17af9b5df9394f8dbc ./tests/Scenarios/UnitrollerScenTest.js

5e1c8ebd93d8065bd53b7ff1867dcb2a8dc430b6faa9d5dad949a0b7d7831aad ./tests/Scenarios/InKindLiquidationScenTest.js

93a699f3cb8cf2978e5ad148d25443f355a3f119bdf84d4f7a4fcbefa0629c4a ./tests/Scenarios/ReduceReservesScenTest.js

2f903f59c90057cfe955b933ae3fb7b17f097e8ca28d2efb3e8e7cc56e1403eb ./tests/Scenarios/RedeemWBTCScenTest.js

01ca493f015cc003b578b60a7df83a8c7c576dbff3b0efbb91bf1ea67ad153ec ./tests/Scenarios/TimelockScenTest.js

c3261939c88aa2a210d91c18118f6f06d38212ca3e8cb0125c79538bc601989d ./tests/Scenarios/BorrowWBTCScenTest.js

18bd40435c9385aae3b5018bdb65da6265eff8b26d16d8e9a03ffa26049efff9 ./tests/Scenarios/ReEntryScenTest.js

d505cbc2d5d96010232526ce9f8c44f32e8c0f8cd732ef8a8da11b4c1c5a676e ./tests/Scenarios/MintWBTCScenTest.js

c294549c150c8f3fe0ce7f9708d4e12860c5725fe20948e712d8e8651f540e6b ./tests/Scenarios/RedeemEthScenTest.js

4a3529fcea2305838a08275b4ceeb4861fea396e9a5cb4acb651d96c0c3de729 ./tests/Scenarios/TokenTransferScenTest.js

2eb4bcabc0cbd1af93d91ff1157b2183cfb9bd881e8e977bccf1575b5443e799 ./tests/Scenarios/SeizeScenTest.js

cfce4030a370f632f1d9df7d2d44e4dc0af05ec641bd223ec906b24b0c09bb07 ./tests/Scenarios/PriceOracleProxyScenTest.js

ad7f7b28e17a9d715b0ef8d811c7bc7fca4aa9e23aa0d2f706abc1cbab70f8f4 ./tests/Scenarios/BorrowCapScenTest.js

a8d77f870a989264aaa2c6361d0cd46ea93497dc886d851d7c068a087674aee2 ./tests/Scenarios/Governor/VoteScenTest.js

dcff6540ca7ad2d404d6f0820f1f699c5e2a721883a2115a094067768d327068 ./tests/Scenarios/Governor/QueueScenTest.js

3ed48d345ed89b6f02c81990f3ba912ea71500d177d7920ef95d11363e868869 ./tests/Scenarios/Governor/DefeatScenTest.js

00b7d5ad7266361d1de01459f809b178c1f683a2714fed986fdbbdda9675d185 ./tests/Scenarios/Governor/CancelScenTest.js

aa4f9419cfa64c2781b88e3a8a86f15243e7d1ffd3d10ceba24f09a158856ffa ./tests/Scenarios/Governor/ProposeScenTest.js

d258fb116bb44586f517e6703f1be7e244d5f566eb76882c2cebdecfc9608b7c ./tests/Scenarios/Governor/ExecuteScenTest.js

98e20441a2e53f58fdcdf95d3bd60f708ad96597dec7e140d0fbceebd0d3e03c ./tests/Scenarios/Governor/GuardianScenTest.js

4eeafe9f7d5b95fe0737438464ec96a1ee1337408e44457f57307ea973f64a77 ./tests/Scenarios/Governor/UpgradeScenTest.js

37452323c84d2dceabd552bc6c2f9695e32be8b94448f44c64fee260acbcee9a ./tests/Scenarios/Cozy/CozyScenTest.js

ca74d555dd74dde421e36b0449a917ff8eaede6da90c02e6b0c0c009d737c383 ./tests/Scenarios/Flywheel/GrantsScenTest.js

0dd36bafff7cf8d9400c7917bb87dcc2839c172bf49faad41a1746ca6286bbf0 ./tests/Scenarios/Flywheel/FlywheelScenTest.js

2365ad209d707a0000b88aed6d3154b23b0db326f4d0d86d703e480ac4064382 ./tests/Scenarios/Flywheel/CozySpeedScenTest.js

734e67402eafdb096dc1a32e670a2e9306fc22a47ccea4d1cbd7669f5d7b28ca ./tests/Scenarios/Flywheel/ReservoirScenTest.js

a3080630f7942cd219f151ac39d4f84bcb78fc5f2bd8526da276eb853e5e28f8 ./tests/Triggers/integrationTest.js

40b99ddee8b237c945999bdafd49b84fd2b1b9d29de813f78279c40482c816a8 ./tests/Triggers/triggerFunctionalityTest.js

150265fa73d686900551a585872205d7cd6343af6e31fc9456619d413d0d3726 ./tests/Factory/protectionMarketFactoryTest.js

55106dfeb8838e170eec4c71567f51cbec4d61c03ae95e0965715ffe4c167dae ./tests/Flywheel/FlywheelTest.js

bfab788e1a12ce6491f536ab62aaf91ddd76f96d9bebbdc934b91f7c5ba7c470 ./tests/Flywheel/GasTest.js

237026fe3650b01db1bf8d3cf4669cf16ec8851515f52a25704b056c98559c23 ./tests/Flywheel/SubsidyTest.js

c6ddafb32ba828cf5231bd4f3a6736bcaadaf1db3a379408152c9e32e275d08f ./tests/Oracle/ChainlinkReporterTest.js

edfc7bc6003032669b4b32679390900b778e8794cbf7beb5ac67718cc14cac76 ./tests/Oracle/EthPassThroughAggregatorTest.js

bf28ac31424807b6f833e735e372797440ff228bd64477aef491a427d9e96173 ./tests/Fuzz/CompWheelFuzzTest.js

f691ab402b932dc09121229de00e3cd73e7f590bff90eb8339360a819c77399f ./tests/Tokens/adminTest.js

630663389894c01e0aee2966830118b0d53701643a3be18d0f88c8639a20139f ./tests/Tokens/cozyLikeTest.js

e7fd3d740eebded4f0ac1b0b1b440ff1634f50d702421a2d799a3fdb445de4ae ./tests/Tokens/mintAndRedeemTest.js

977aaa8998cbfcd25c6c5580b6d23741a4961de3a3823914ed0e14cbba03ce3c ./tests/Tokens/setInterestRateModelTest.js

17f74a77bc24d4e09d7e455dabced844a1daa6df8cdd25a159da60a7392687c9 ./tests/Tokens/borrowAndRepayTest.js

dbb2bbfc5c862639f62ca3015cc15c1a78449a73cee415ea8c4612a27722d032 ./tests/Tokens/accrueInterestTest.js

da7ef640001878236bf278f73e56280491cd3e0fbd7878e5ed67ce1d8161410d ./tests/Tokens/mintAndRedeemCEtherTest.js

8b0f43d6252007273e84db4c0f2f1c56163661a47d5d71e61db394548936bbb6 ./tests/Tokens/borrowAndRepayCEtherTest.js

3813ec5f3c5116aad4cb71f5b5f57ffa89fe86d25bcf582cbccc00c3c355e0c7 ./tests/Tokens/safeTokenTest.js

d8a4966f68fe392d998c1a22932ab25ad673a8b02ff783c71f9b57655e1fc1ba ./tests/Tokens/transferTest.js

3ee155f5e9046e0091f60ae737f53be5673808e08854e330615fece2785aec1f ./tests/Tokens/deployProtectionMarketTest.js

51d1207d59dbef517391e4c27b19bb1af3b1236a65cbb275703c4384242becf6 ./tests/Tokens/reservesTest.js

c9c9276c329482043df642586a2b233cba9f352568c0aee6d15d3c78230cf92b ./tests/Tokens/cTokenTest.js

01fce6e4fcf4fd521770bb46e9ddd54aac94885ab111b427979e39722122db0b ./tests/Tokens/liquidateTest.js

9d620d9b2c91212e1e4f66e94471e5c5781214c1adef233bacba5b8ba6b8da19 ./tests/Tokens/signatureTest.js

6565bc8483f9ff276893e2abf1890b246bcdbb7e6b2e1553dff7054b8093132d ./tests/Models/InterestRateModelTest.js

3328562450ab6869b9a63bb155fb75c78ed6554a5a52c7765e09e009544754d7 ./tests/Models/DAIInterestRateModelTest.js

b4124701c1e4a370a18cbf81dc9b836cde775c94f3f1b9480852cde25e9d074a ./tests/Comptroller/adminTest.js

255a68d37964d2faa258e78fb13474c40b723d82074c632e9ff640f02166d7f7 ./tests/Comptroller/accountLiquidityTest.js

5ff8d4df758790b5b9530e39051a0900152df547132c8541e891a989f30cad64 ./tests/Comptroller/comptrollerTest.js

e4f27491957ab0914645b9f6bdf33e5a0ae777fcfcec272b16edaf6fd13779d8 ./tests/Comptroller/proxiedComptrollerV1Test.js

bdd4c104b9fc8f490b70d2a7de6a02324db11c8f2b7e039ed1d2fd2cdec68d89 ./tests/Comptroller/unitrollerTest.js

6e888e6807648f5a5f70e3326f8fe201174d47a6de3149675c56487c32ea5230 ./tests/Comptroller/liquidateCalculateAmountSeizeTest.js

53a0bb2c77ae4590ee5fa59a01acfd85dc12ee04e29732c15da46fc8149a7cc0 ./tests/Comptroller/reserveGuardianTest.js

eb70b71897b5db3bf246c31a790ffd161a224dcd768a1e8a0c2018cf2e53f0eb ./tests/Comptroller/cozyTest.js

5e73692696e47cdb4289e95c7297a07159899c2d1fc367cc4f788538795e3874 ./tests/Comptroller/assetsListTest.js

a07b963e803eefc49824f82cead332f0289da2ce833f4ce23a190bda2ac7c998 ./tests/Comptroller/pauseGuardianTest.js

Changelog

- 2021-04-23 - Initial report
- 2021-05-06 - Re-audit
- 2021-05-24 - Report update

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.