Quantstamp Security Assessment Certificate

January 22nd 2021 — Quantstamp Verified

88mph (v2)

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

Executive Summary

Type	DeFi platform
Auditors	Jan Gorzny, Blockchain Researcher Joseph Xu, Technical R&D Advisor Jose Ignacio Orlicki, Senior Engineer
Timeline	2020-12-08 through 2021-01-15
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Smart Contract Reference
Documentation Quality	Low
Test Quality	Low
Source Code	

Repository	Commit
88mph-contracts	2fc696b

Total Issues	13	(4 Resolved)
High Risk Issues	0	(0 Resolved)
Medium Risk Issues	3	(2 Resolved)
Low Risk Issues	6	(1 Resolved)
Informational Risk Issues	1	(0 Resolved)
Undetermined Risk Issues	3	(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

Quantstamp has performed an audit on the Solidity contracts in the 88mph system. The project has been updated since the last audit, and the new features and code changes introduced additional concerns. Quantstamp found 14 issues, and we recommend fixing all of them before the system is used by the general public. Some issues arise from lack of documentation and privileged roles (which may be unavoidable); these issues and the general understanding of the system would be avoided or improved through better documentation. Although there are more tests than in the previous audit, we were unable to generate code coverage for the tests. As in the case of the previous audit, only the smart contracts were audited (and in particular, the web-interface was not audited).
Note that the files in the `fractionals` and `zaps` directory were not in scope for this audit (which was based off of commit `467bfcd`).

ID	Description	Severity	Status
QSP-1	Potentially Incorrect Minting	Medium	Acknowledged
QSP-2	Functions May Become Disabled	Medium	Fixed
QSP-3	Privileged Minting	Medium	Fixed
QSP-4	Possible Market Mismatch	Low	Fixed
QSP-5	Privileged Roles and Ownership	Low	Acknowledged
QSP-6	Copycat NFTs can be minted	Low	Acknowledged
QSP-7	Gas Usage / `for` Loop Concerns	Low	Acknowledged
QSP-8	Unchecked Parameters	Low	Acknowledged
QSP-9	Privileged Roles and Ownership	Low	Acknowledged
QSP-10	Use `external` declaration for functions not used in other functions	Informational	Acknowledged
QSP-11	Possible Surplus Funds	Undetermined	Acknowledged
QSP-12	Linear Interest Model May Underestimate Deposit APY	Undetermined	Acknowledged
QSP-13	Underlying Rate Manipulation	Undetermined	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:

- Slither v0.6.6

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 Potentially Incorrect Minting

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `MPHIssuanceModel01.sol`

Description: The documentation indicates minting of MPH tokens to the dev fund on new deposits, but the issuance model also mints MPH tokens to the dev fund on funding. In addition, the dev fund would accrue MPH tokens rather quickly because it receives 10% of the full MPH tokens minted on deposit, even though the depositor is supposed to pay back 90% of these tokens on withdrawal. The full distribution of the token as a result of a single deposit held to maturity (without funding) is 9% to the depositor, 9% to the devs, and 82% to the governance treasury.

Recommendation: Correct the implementation or add additional clarifications regarding the developer fund in the Tokenomics section of the documentation.
Update: The developer confirms that this is the intended tokenomics at 90% payback percentage (but the payback percentage is at 30% as of this report).

QSP-2 Functions May Become Disabled

Severity: Medium Risk

Status: Fixed

File(s) affected: `MPHMinter.sol`

Description: `setMPHTokenOwner()` and `setMPHTokenOwnerToZero()` will not allow further MPH token minting in the `mintDepositorReward()` and `mintFunderReward()` functions within the MPHMinter contract, which in turn can disable `_deposit()`, `_withdraw()`, and `_payInterestToFunder()` functions in `DInterest.sol`.

Recommendation: Check the owner of MPH token and return 0 on `mintDepositorReward()` and `mintFunderReward()` if the owner is not the MPHMinter contract.
Update: the recommendation was implemented.

QSP-3 Privileged Minting

Severity: Medium Risk

Status: Fixed

File(s) affected: `MPHMinter.sol`

Description: Privileged roles: `setMPHTokenOwner()` can be called at any time with the new owner minting as many MPH Tokens as desired.

Recommendation: Consider a timelock for ownership transfer due to the highly privileged nature of the `MPHToken` contract.
Update: The developers have stated that "the ownership of the MPHMinter contract is given to a 48-hour timelock contract.".

QSP-4 Possible Market Mismatch

Severity: Low Risk

Status: Fixed

File(s) affected: `DInterest.sol`

Description: `setInterestOracle()` does not check for the same money market as the pool. Ideally it should also check consistency of data between the old and new oracle before updating to avoid significant changes in the interest rate parameters.

Recommendation: Check that the market is the same one expected by the pool.
Update: the check has been implemented.

QSP-5 Privileged Roles and Ownership

Severity: Low Risk

Status: Acknowledged

File(s) affected: `MPHIssuanceModel01.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.
In particular, `devRewardMultiplier` can be changed with no check on "reasonable" values.

Recommendation: Add a `require()` statement to function `setDevRewardMultiplier()` with an appropriate upper bound on the `devRewardMultiplier` (The doc indicates 10%).

QSP-6 Copycat NFTs can be minted

Severity: Low Risk

Status: Acknowledged

File(s) affected: `DInterest.sol`

Description: Even though complete forks cannot be avoided, an attacker can create rogue pools using rogue `DInterest` contracts with a popular stablecoin. The attacker can make a big deposit, recover the deposit with a fake `IMoneyMarket` implementation, and get a fake NFT that can be sold in a secondary market like OpenSea. RCN is an example of financial NFTs being sold in open markets (check history of RCN trades https://nonfungible.com/market/history/ripiocreditnetwork). Open NFT markets can be deceived to accept fake ones if they accept all NFTs or their filters are bypassed by the attacker (https://opensea.io/get-listed).

Recommendation: Because for 88mph NFTs the `owner` must be set to this `DInterest` contract, currently each pool has a separate NFT implementation. It is recommended to have only one NFT implementation for the whole project (or one NFT implementation for deposits and one for funding tickets), so third parties can check that these are official deposits or funding tickets. As the number of different pools is unlimited and will grow over time, it otherwise becomes more difficult for a buyer to check that one is indeed buying an official deposit from 88mph in an open market.

QSP-7 Gas Usage / `for` Loop Concerns

Severity: Low Risk

Status: Acknowledged

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.

Recommendation: Ensure that these `for` loops are not expected to exceed gas limits.

QSP-8 Unchecked Parameters

Severity: Low Risk

Status: Acknowledged

File(s) affected: `OneSplitDumper.sol`, `Rewards.sol`, `Vesting.sol`, `MPHMinter.sol`, `MPHIssuanceModel01.sol`, `EMAOracle.sol`

Description: Several functions and constructors do not check if addresses are non-zero, which may cause headaches during deployment or unintended consequences when such an address is not intended as an argument.

- `OneSplitDumper.sol`: `getDumpParams`, `dump`.
- `Rewards.sol`: constructor.
- `Vesting.sol`: constructor.
- `MPHMinter.sol`: constructor, `takeBackDepositorReward`.
- `MPHIssuanceModel01.sol`: `setPoolDepositorRewardMintMultiplier`.
- `EMAOracle.sol`: constructor.

Recommendation: Check that the functions revert on the zero address or confirm that this behaviour is intended.

QSP-9 Privileged Roles and Ownership

Severity: Low Risk

Status: Acknowledged

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

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.

QSP-10 Use `external` declaration for functions not used in other functions

Severity: Informational

Status: Acknowledged

File(s) affected: `EMAOracle.sol`, `MPHToken.sol`

Description: Functions only called externally by other contracts or users can be only declared as `external`, gas is saved and attackers are given less internal functions and control in case of vulnerabilities. This way they cannot be called from other `internal` or `public` functions.

- `/contracts/models/interest-oracle/EMAOracle.sol`: `updateAndQuery()` and `query()`
- `/contracts/rewards/MPHToken.sol`: `init()` and `ownerMint()`

Recommendation: Make the appropriate functions `external`.

QSP-11 Possible Surplus Funds

Severity: Undetermined

Status: Acknowledged

File(s) affected: `DInterest.sol`

Description: The system can have a lot of surplus but the surplus coins are stuck inside the protocol (deposited in the respective money markets) and can only be used to fund possible deficits in the future

Recommendation: Ensure this is not problematic or provide some way to extract such funds.

QSP-12 Linear Interest Model May Underestimate Deposit APY

Severity: Undetermined

Status: Acknowledged

File(s) affected: `LinearInterestModel.sol`

Description: A linear interest rate model is good for small rates around 1 or 2%, but for bigger rates around 5, 10 or 20% this model will underestimate the rates against the user. The original exponential rate is doing `(1+dailyRate)^365 - 1` to get the rate, and the linearized version does `dailyRate*365`. The linear model uses seconds but the money market rate `moneyMarketInterestRatePerSecond` is computed using daily rates, so we can use daily rates. For daily samples and annualizing, approx. numbers:

- 1% annual: Exponential gives 1% with daily rate of 0.0000273, linear gives 0.99%, an underestimation of 1% in rates.
- 5% annual (similar to USDC Aave on 88mph today December 16th): Exponential gives 5% with a daily rate of 0.0001337, linear gives 4.88%, an underestimation of 0.12% (a fraction of 2.4%).
- 10% annual: Exponential gives 10% with a daily rate of 0.0002612, linear gives 9.53%, an underestimation of 0.47% (a fraction of 4.66%).
- 20% annual: Exponential gives 20% with a daily rate of 0.0004997, linear gives a rate of 18.24%, an underestimation of 1.76% (a fraction of 6.2%).

Recommendation: If you see large APY over 10% very often, consider migrating to an exponential model using widely-used arithmetic libraries.

QSP-13 Underlying Rate Manipulation

Severity: *Undetermined*

Status: Mitigated

File(s) affected: `DInterest.sol`

Description: In broad strokes, the attack is as follows:

- Make a large deposit in the underlying pool
- Buy the corresponding floating rate bond
- Make a large withdrawal in the underlying pool
- Liquidate the floating rate bond indirectly when the depositors withdraw
- Slowly repeat the process, extracting value from the system

This may have the effect of causing the underlying floating rate to spike, in which case the fixed rate depositors bear the loss of these actions.

Exploit Scenario: More concretely, in the code, an attacker would need to do the following, slowly repeating the attack:
Step 1: Make a large deposit in the underlying pool (e.g., yUSD, ycrvSBTC) - this will decrease the APY of the underlying pool
Step 2: Buy up the available floating rate bond NFTs
Step 3: Make a large withdrawal from the corresponding asset's underlying pool - this will increase the APY of the underlying pool
Step 4: Liquidate the floating-rate bond indirectly and automatically, when the depositor withdraws. It is indirect in the sense that the depositor triggers this action. This results in the system having to payout funds from the depositors to the attacker (more than the expected amount, due to the shifts in the underlying pools' APY).

Recommendation: Ensure that this is not possible, or make this issue clear to end users.
Update: This attack is mitigated by the design of the contracts. In particular, `DInterest.sol` use of the income index before withdrawal to disregard the impact of 88mph's withdrawal on the money market income index itself.

Automated Analyses

Slither

Slither found many best practice issues, for which examples have been provided in the `Adherence to Best Practices` section. Other results, like reentrancy, were determined to be false positives.

Adherence to Best Practices

1. `EMAOracle.sol`: For EMA, `UPDATE_MULTIPLIER` is uniquely determined between the interval (0, 1] so it can be specified directly using an argument in the constructor. The calculation on line 44 using `_smoothingFactor` and `_averageWindowIntervals` seems redundant. If these two variables are needed for informational reasons, then it may be worth keeping these as variables or emitting an event in the constructor.
2. `Vesting.sol`: Typo in function `withdrawVested()` - it should be `withdrawableAmount` instead of `withdrawnAmount`.
3. Not all variables are in `mixedCase`, e.g. `MaxDepositPeriod` of `DInterest.sol`.
4. Various functions, like `query` of `EMAOracle.sol`, could be made external.
5. Multiplication should not be performed after division. This happens at least twice: `EMAOracle.updateAndQuery()` lines 63 and 66, and `Rewards.notifyRewardAmount()` lines 212 and 215. **Update:** this is no longer relevant.
6. Using a contract registry, which is a more expensive solution but it pays out depending on how many contracts you deploy. Examples:
   https://github.com/celo-org/celo monorepo/blob/master/packages/protocol/contracts/common/UsingRegistry.sol
   https://github.com/OpiumProtocol/opium-contracts/blob/master/contracts/Lib/UsingRegistry.sol

Test Results

Test Suite Results

```
Contract: Aave
  normal operations
     ✓ deposit() (1612ms)
     ✓ withdraw() (5410ms)
     ✓ earlyWithdraw() (4068ms)
     ✓ fundAll() (9146ms)
     ✓ fundMultiple() (11067ms)
     ✓ totalInterestOwedToFunders() (2264ms)
  MPH tokenomics
     ✓ should mint correct MPH depositor reward (1618ms)
     ✓ should take back correct MPH depositor reward (2390ms)
     ✓ should mint correct MPH funder reward (9481ms)
     ✓ should not increase depositor MPH balance if early withdraw (2179ms)
     ✓ should not increase attacker balances if deposit => buy bonds => immediately early withdraw (3674ms)
     ✓ should not increase attacker balances if deposit => buy bonds => wait a while => early withdraw (3090ms)
     ✓ MPHMinter should not accept calls from random account (61ms)
     ✓ owner can update MPH owner (40ms)

Contract: Compound
  normal operations
     ✓ deposit() (1551ms)
     ✓ withdraw() (4752ms)
     ✓ earlyWithdraw() (3610ms)
```

```
      ✓ fundAll() (9010ms)
      ✓ fundMultiple() (11784ms)
      ✓ totalInterestOwedToFunders() (1911ms)
      ✓ claimRewards() (158ms)
    MPH tokenomics
      ✓ should mint correct MPH depositor reward (1275ms)
      ✓ should take back correct MPH depositor reward (2143ms)
      ✓ should mint correct MPH funder reward (7978ms)
      ✓ should not increase depositor MPH balance if early withdraw (2015ms)
      ✓ should not increase attacker balances if deposit => buy bonds => immediately early withdraw (3084ms)
      ✓ should not increase attacker balances if deposit => buy bonds => wait a while => early withdraw (3051ms)
      ✓ MPHMinter should not accept calls from random account (62ms)
      ✓ owner can update MPH owner

  Contract: FractionalDeposit
    ✓ create fractional deposit (830ms)
    ✓ withdraw deposit after maturation (2250ms)
    redeem with direct withdrawal
      ✓ redeem share (1049ms)
      ✓ transfer NFT to creator (127ms)
    redeem without direct withdrawal
      ✓ redeem share (1973ms)

  Contract: Harvest
    normal operations
      ✓ deposit() (1455ms)
      ✓ withdraw() (5601ms)
      ✓ earlyWithdraw() (4339ms)
      ✓ fundAll() (10656ms)
      ✓ fundMultiple() (15359ms)
      ✓ totalInterestOwedToFunders() (2865ms)
      ✓ claimRewards() (2034ms)
    MPH tokenomics
      ✓ should mint correct MPH depositor reward (1681ms)
      ✓ should take back correct MPH depositor reward (2842ms)
      ✓ should mint correct MPH funder reward (11247ms)
      ✓ should not increase depositor MPH balance if early withdraw (2612ms)
      ✓ should not increase attacker balances if deposit => buy bonds => immediately early withdraw (4414ms)
      ✓ should not increase attacker balances if deposit => buy bonds => wait a while => early withdraw (4095ms)
      ✓ MPHMinter should not accept calls from random account (57ms)
      ✓ owner can update MPH owner

  Contract: Vesting
    ✓ should vest full amount after vesting period (311ms)
    ✓ should vest linear partial amount during vesting period (246ms)
    ✓ should vest full amount after vesting period only once (519ms)
    ✓ should vest linear partial amount during vesting period only once (599ms)
    ✓ should fail tx when withdrawing from non-existent vest object (182ms)

  Contract: YVault
    normal operations
      ✓ deposit() (1481ms)
      ✓ withdraw() (4719ms)
      ✓ earlyWithdraw() (3849ms)
      ✓ fundAll() (9964ms)
      ✓ fundMultiple() (12346ms)
      ✓ totalInterestOwedToFunders() (2173ms)
    MPH tokenomics
      ✓ should mint correct MPH depositor reward (1426ms)
      ✓ should take back correct MPH depositor reward (2451ms)
      ✓ should mint correct MPH funder reward (10481ms)
      ✓ should not increase depositor MPH balance if early withdraw (2210ms)
      ✓ should not increase attacker balances if deposit => buy bonds => immediately early withdraw (3541ms)
      ✓ should not increase attacker balances if deposit => buy bonds => wait a while => early withdraw (3695ms)
      ✓ MPHMinter should not accept calls from random account (95ms)
      ✓ owner can update MPH owner

  Contract: ZeroCouponBond
    ✓ create zero coupon bond (219ms)
    ✓ mint zero coupon bond (1669ms)
    ✓ redeem stablecoin from zero coupon bond (3685ms)


  71 passing (6m)
```

Code Coverage

Code coverage was not correctly computed. In particular, although the test suite runs on its own, tests fail when running soldity-coverage.

Update: This data is from the original report commit, as coverage failed using the new commit, even though the tests passed on their own.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	41.8	20.65	28.21	41.8	
DInterest.sol	41.8	20.65	28.21	41.8	… 818,821,822
contracts/libs/	100	100	100	100	
DecMath.sol	100	100	100	100	
contracts/models/fee/	40	0	66.67	40	
IFeeModel.sol	100	100	100	100	
PercentageFeeModel.sol	40	0	66.67	40	27,28,29
contracts/models/interest-**oracle/**	95.24	100	66.67	95.24	
EMAOracle.sol	95.24	100	66.67	95.24	73
IInterestOracle.sol	100	100	100	100	
contracts/models/interest/	100	100	100	100	
IInterestModel.sol	100	100	100	100	
LinearInterestModel.sol	100	100	100	100	
contracts/models/issuance/	75	37.5	80	75	
IMPHIssuanceModel.sol	100	100	100	100	
MPHIssuanceModel01.sol	75	37.5	80	75	… 248,252,253
contracts/moneymarkets/aave/	100	50	71.43	100	
AaveMarket.sol	100	50	71.43	100	
contracts/moneymarkets/compound/	0	0	0	0	
CompoundERC20Market.sol	0	0	0	0	… 109,110,111
contracts/moneymarkets/harvest/	0	0	0	0	
HarvestMarket.sol	0	0	0	0	… 111,112,113
contracts/moneymarkets/harvest/imports/	100	100	100	100	
HarvestStaking.sol	100	100	100	100	
HarvestVault.sol	100	100	100	100	
contracts/moneymarkets/yvault/	0	0	0	0	
YVaultMarket.sol	0	0	0	0	… 72,73,74,78
contracts/moneymarkets/yvault/imports/	100	100	100	100	
Vault.sol	100	100	100	100	
contracts/rewards/	56.63	28.13	52.94	57.14	
IRewards.sol	100	100	100	100	
MPHMinter.sol	46.77	23.08	41.67	47.62	… 247,248,249
Vesting.sol	85.71	50	80	85.71	54,72,93
contracts/rewards/dumpers/	0	0	0	0	
Dumper.sol	100	100	0	100	
OneSplitDumper.sol	0	0	0	0	… 63,70,71,72
contracts/rewards/dumpers/imports/	100	100	100	100	
Curve.sol	100	100	100	100	
OneSplitAudit.sol	100	100	100	100	
yERC20.sol	100	100	100	100	
contracts/rewards/dumpers/withdrawers/	0	100	0	0	
CurveLPWithdrawer.sol	0	100	0	0	… 60,61,62,63
YearnWithdrawer.sol	0	100	0	0	9,10,11
All files	**41.77**	**21.51**	**35.65**	**41.9**	

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

```
dc39b5d9ae0c2d664f7436090ef8b88d72b5251fbd439045f5147ee4a0d0193c ./contracts/DInterest.sol
669ab8b9c8720a6bcb6160227e10e6b78ff68c82f41b77a3788fa5425b85e626 ./contracts/NFT.sol
0c381877868b3c537bf4cf3bdd6794aca176ef1fe38ec380b7eb03957dc99ad6 ./contracts/rewards/IRewards.sol
f9b9c91c1190bec1c1a93bbc679c817fe818893c47860edb9dda3a5c7528f662 ./contracts/rewards/MPHMinter.sol
b9ee096ae465551d4e8f3a7ad56f4f062ff9c63f6de1dbe6e915e0467fea6dcf ./contracts/rewards/MPHToken.sol
e414d942803f4267dd2db53edc8d69c5be79ad5ea8022d96bfb52f1f62ca7679 ./contracts/rewards/Rewards.sol
6356d09271d5ec4cdbfe9fd685af82e06368085354041018fd689e067b1d5246 ./contracts/rewards/Vesting.sol
b31b6ec411997b43ccb17772ce1e628be43b99bfc1be034c2d4afe70c10b3ffc ./contracts/rewards/dumpers/Dumper.sol
80e3a885057c51f16380bd81743721280c8cfaf817bbad2bb643bad3605c7a9a ./contracts/rewards/dumpers/OneSplitDumper.sol
22f64575c80f6ed4061a60171f557a3222770524a317762907d7e0044531f314 ./contracts/rewards/dumpers/withdrawers/CurveLPWithdrawer.sol
12bd139068527c87fa527e9f4615c6f3847821da4c792c95ca5c9762214a1554 ./contracts/rewards/dumpers/withdrawers/YearnWithdrawer.sol
d0b15c8ac19054bc441d24f5742594faee204b20a54dd3025901a2585ffce279 ./contracts/rewards/dumpers/imports/Curve.sol
c6baa0c38e2c4523434e7e7dcb22ac79dad718c0cc2bec445a1cf34786976a10 ./contracts/rewards/dumpers/imports/OneSplitAudit.sol
93e92a5917ef1dd1367488fe55a33736c2020183838fb70c2e5662c8bf370b9a ./contracts/rewards/dumpers/imports/yERC20.sol
610d0a7cad6066c91062e1f7286bc1ea3e02eefb903fa6afadbd54b4bd5372b7 ./contracts/moneymarkets/IMoneyMarket.sol
1fcaf247d7cdca11d119cbfa0761744ab229648e7656a1d1f12f6f7fab98d669 ./contracts/moneymarkets/yvault/YVaultMarket.sol
71c5740a78d87eb15171940510b8c261827ad25fbfdbcba323e2c487d773cf5b ./contracts/moneymarkets/yvault/imports/Vault.sol
e63f8706310c25555aecddee72d6bb5a2f2a6dee0435c7069885c51d01667a07 ./contracts/moneymarkets/harvest/HarvestMarket.sol
68803e6ba5301f7dfb28d6d138538109c0b1eae8a4a056615fc3997803dfe645 ./contracts/moneymarkets/harvest/imports/HarvestStaking.sol
cd7a02321f3b05c48ad87e2c4cdda7879217fac472f477e18556b7e34458cc50 ./contracts/moneymarkets/harvest/imports/HarvestVault.sol
d012b10abde89aded37e52f5b0c5549601e6b77d3098acfb64f0bd580ea151b4 ./contracts/moneymarkets/compound/CompoundERC20Market.sol
d51373871e9a063f4f9bb66e1c3c72c65dde82042260aaa5889dbe29369c5e70 ./contracts/moneymarkets/compound/imports/ICERC20.sol
65d17e0687d1ade74ffa025a43b10cca6137a853842ccc3af867b483f14c5693 ./contracts/moneymarkets/compound/imports/IComptroller.sol
e0cb1902c713d111de6086cde7c94a0755d25b30fab195c63e88e035dfdb722d ./contracts/moneymarkets/aave/AaveMarket.sol
fbd3c310ef1e4e4fd6420aff3b990819db29db696809e78b54674b700c21aa64 ./contracts/moneymarkets/aave/imports/ILendingPool.sol
613c38d7b934fbd806d3d30f65df65577f68e0a14863af04c78ae87db287ecb4 ./contracts/moneymarkets/aave/imports/ILendingPoolAddressesProvider.sol
df859cd1086a13d0643fe2b611d9c040177c90eebd3f6cef2792e301e029908d ./contracts/models/issuance/IMPHIssuanceModel.sol
77b7acfd0d612fd097b749b92112b55f673dd50b0898153f28835653c7d45937 ./contracts/models/issuance/MPHIssuanceModel01.sol
a26616b3509439be1be2a074eb2bd108cb621b1eb98371ef5b06bb92dae54db7 ./contracts/models/interest-oracle/EMAOracle.sol
2901fe9fa003bad45f0ef4a0e8832015ec38d9b0c39ef465810e412883cc0ed1 ./contracts/models/interest-oracle/IInterestOracle.sol
632b7946f3a40c104d9fe014895b14f21f9dfb6cec65f8ef2e3cb2423f0501d4 ./contracts/models/interest/IInterestModel.sol
5e2dc453d2c16fe8747b5948a65206e53e6b1da45610747d0a770024c4943d10 ./contracts/models/interest/LinearInterestModel.sol
a862bafb8f4aa88723e15ceb7e50e5f328121238638d3055d943a730fee6a454 ./contracts/models/fee/IFeeModel.sol
db50a1c87d4daa7c9c2a00f7ddac89ae744bb99b7ddfe132737dd1a241fd6ab1 ./contracts/models/fee/PercentageFeeModel.sol
0dcbd311898fcdbcf519788ed4b9ccbb531c6befada98211dae975ce242da5e2 ./contracts/mocks/ATokenMock.sol
87bca01caabf5a06b35ac83df4465c7eac309ce346cb8733b15ecda58220c417 ./contracts/mocks/CERC20Mock.sol
d232de4161fb9cb41b8614e9dfc27188b5bb6fac36459cb159aa24a8b374444d ./contracts/mocks/ComptrollerMock.sol
b710dc03c894bb8b4e2a95f37074f720381153e674e57683632b537bfaba2ec2 ./contracts/mocks/ERC20Mock.sol
c628498ec2d68935c754d21cd7a61ae6d459169ebea40939d7c3fc1cb432ccd7 ./contracts/mocks/HarvestStakingMock.sol
9de76f0e1f206a968ec57b0bf48a82af43ca01443404660dbc1eb069647b9500 ./contracts/mocks/LendingPoolAddressesProviderMock.sol
93b2b2bf9d2128041ae71f850140a38ee001b24d47960ae895521b305d209836 ./contracts/mocks/LendingPoolMock.sol
03c74d31219e264dea03518729c628fa2cae5c00ac35f10d30eca3e699631618 ./contracts/mocks/VaultMock.sol
0df2248ddfdb1f3e20a430684a8b69ca8fd991dbfcb89f562d276d4e48240de2 ./contracts/libs/CloneFactory.sol
576b9393b8f9ad7833cd5393f7845c952ec95f9e01f7bf2ef8c763da1c3131c3 ./contracts/libs/DecMath.sol
```

Tests

```
d60402fad136e422f6beea2096dbd5b8f83576922b5f18765834963b975f0c9a ./test/aave.js
0d6364531837be5ad71f6c0ca32cca4bd55295c357024876b4ce68a89b099587 ./test/compound.js
0ce760533aaf9e47a3605c79895d8ed4855aaa674f0e70f1dd5ddaae0abb2ca4 ./test/fractional-deposit.js
82116f53153a285b273dfd50779b5718c0f5b9afb4beb46b628c3c9e91bb7288 ./test/harvest.js
6a484c35dec70f02b126bb706d46c98c5181a8ad992aab9bb5e33b8ffa1ee3ee ./test/vesting.js
294ea65b5ce001ae7b2bdd184e38a91e30f86b66b6a97501ab2ae2771ba7a829 ./test/yvault.js
f14bdf4b20e1fe6dc388cea4b96cb7b547075b18b85ec10851cb23b26f07dd2a ./test/zero-coupon-bond.js
```

Changelog

- 2020-12-17 - Initial report [467bfcd]
- 2021-01-12 - Revised report [2fc696b]

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.