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
Specification: Smart Contract Reference
Documentation Quality: Low
Test Quality: Low
Source Code: Repository Commit

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>88mph-contracts</td>
<td>2fc696b</td>
</tr>
</tbody>
</table>

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.
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 9 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).

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

**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 (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, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

**Toolset**

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

**Setup**

**Tool Setup:**

- *Check v0.6.5*
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 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 DInterest 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-tips).

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, MHPIssueModel01.sol, EMADecrease.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: `getDepositParams`, `dump`
- Rewards.sol: `constructor`
- Vesting.sol: `constructor`
- MPHMinter.sol: `constructor`, `takeBackDepositorReward`
- MHPIssueModel01.sol: `setPoolDepositorRewardMintMultiplier`
- EMADecrease.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:** EMADecrease.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 control in case of vulnerabilities. This way they cannot be called from other internal or public functions.

- `/contracts/models/interest-oracle/EMADecrease.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 $(1+\text{dailyRate})^{365} - 1$ to get the rate, and the linearized version does $\text{dailyRate} \times 365$. The linear model uses seconds but the money market rate $\text{moneyMarketInterestRatePerSecond}$ 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

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:

1. Make a large deposit in the underlying pool (e.g., yUSD, ycrvSBTC) - this will decrease the APY of the underlying pool
2. Buy up the available floating rate bond NFTs
3. Make a large withdrawal from the corresponding asset’s underlying pool - this will increase the APY of the underlying pool
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 smoothRateFactor and averageWindowInterval 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 withdrawAmount.

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 lock correct MPH depositor reward (209ms)
  ✓ should have lock correct MPH depositor reward (209ms)
  ✓ should not increase depositor MPH balance of early withdraw (217ms)
  ✓ should not increase attacker balances if deposit => buy bonds => immediately early withdraw (797ms)
  ✓ should not increase attacker balances if deposit => buy bonds => wait a while => early withdraw (898ms)
  ✓ MPHOracle should not accept too late requests (120ms)

  Owner should not receive calls from random account (5ms)

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)

Contract: FractionalDeposit
✓ create fractional deposit (830ms)
✓ withdraw deposit after maturation (2250ms)
✓ redeem with direct withdrawal
✓ redeem share (1049ms)
✓ transfer NFT to creator (127ms)
✓ redeem share (225ms)

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)

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)

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 solidity-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.

<table>
<thead>
<tr>
<th>File</th>
<th>% Stmts</th>
<th>% Branch</th>
<th>% Funcs</th>
<th>% Lines</th>
<th>Uncovered Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>contracts/</td>
<td>41.77</td>
<td>21.51</td>
<td>35.65</td>
<td>41.9</td>
<td></td>
</tr>
<tr>
<td>DInterest.sol</td>
<td>41.8</td>
<td>28.65</td>
<td>28.21</td>
<td>41.8</td>
<td></td>
</tr>
<tr>
<td>contracts/libs/</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>DecMath.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>contracts/models/fee/</td>
<td>40</td>
<td>0</td>
<td>66.67</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>IFreeModel.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>PercentageFeeModel.sol</td>
<td>40</td>
<td>0</td>
<td>66.67</td>
<td>48</td>
<td>27,28,29</td>
</tr>
<tr>
<td>contracts/models/interest-oracle/</td>
<td>95.24</td>
<td>100</td>
<td>66.67</td>
<td>95.24</td>
<td>73</td>
</tr>
<tr>
<td>EMAOracle.sol</td>
<td>95.24</td>
<td>100</td>
<td>66.67</td>
<td>95.24</td>
<td></td>
</tr>
<tr>
<td>IInterestOracle.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>contracts/models/interest/</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>IIInterestModel.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>LinearInterestModel.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>contracts/models/issuance/</td>
<td>75</td>
<td>37.5</td>
<td>80</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>MPHIssuanceModel.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>MPHIssuanceModel01.sol</td>
<td>75</td>
<td>37.5</td>
<td>80</td>
<td>75</td>
<td>248,252,255</td>
</tr>
<tr>
<td>contracts/moneymarkets/aave/</td>
<td>100</td>
<td>50</td>
<td>71.43</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>AaveMarket.sol</td>
<td>100</td>
<td>50</td>
<td>71.43</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>contracts/moneymarkets/compound/</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CompoundERC20Market.sol</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>109,110,111</td>
</tr>
<tr>
<td>contracts/moneymarkets/harvest/</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>HarvestMarket.sol</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>111,112,113</td>
</tr>
<tr>
<td>contracts/moneymarkets/harvest/imports/</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>HarvestStaking.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>HarvestVault.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>contracts/moneymarkets/yvault/</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>YVaultMarket.sol</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>72,73,74,78</td>
</tr>
<tr>
<td>contracts/moneymarkets/yvault/imports/</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Vault.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>contracts/rewards/</td>
<td>56.63</td>
<td>28.13</td>
<td>52.94</td>
<td>57.14</td>
<td></td>
</tr>
<tr>
<td>IRewards.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>MPHMinter.sol</td>
<td>46.77</td>
<td>23.08</td>
<td>41.67</td>
<td>47.62</td>
<td>247,248,249</td>
</tr>
<tr>
<td>Vesting.sol</td>
<td>85.71</td>
<td>50</td>
<td>80</td>
<td>85.71</td>
<td>54,72,93</td>
</tr>
<tr>
<td>contracts/rewards/dumpers/</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Dumper.sol</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>OneSplitDumper.sol</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>63,70,71,72</td>
</tr>
<tr>
<td>contracts/rewards/dumpers/imports/</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Curve.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>OneSplitAudit.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>yERC20.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>contracts/rewards/dumpers/withdrawers/</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CurveLPWithdrawer.sol</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>60,61,62,63</td>
</tr>
<tr>
<td>YearnWithdrawer.sol</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>9,10,11</td>
</tr>
</tbody>
</table>
Contracts
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.

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

Contracts

/contracts/moneymarkets/compound/imports/ICERC20.sol
/contracts/moneymarkets/compound/CompoundERC20Market.sol
/contracts/moneymarkets/harvest/imports/HarvestVault.sol
/contracts/moneymarkets/harvest/imports/HarvestStaking.sol
/contracts/moneymarkets/aave/AaveMarket.sol
/contracts/moneymarkets/yvault/YVaultMarket.sol
/contracts/rewards/dumpers/OneSplitAudit.sol
/contracts/rewards/dumpers/Curve.sol
/contracts/rewards/dumpers/withdrawers/CurveLPWithdrawer.sol
/contracts/rewards/dumpers/OneSplitDumper.sol
/contracts/rewards/dumpers/Dumper.sol
/contracts/rewards/Vesting.sol
/contracts/rewards/MPHToken.sol
/contracts/rewards/MPHMinter.sol
/contracts/NFT.sol
/contracts/DInterest.sol
/contracts/moneymarkets/IMoneyMarket.sol
Changelog

• 2020-12-17 - Initial report [467bfc6]

• 2021-01-12 - Revised report [2f696b]
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.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.