Quantstamp Security Assessment Certificate

August 21st 2020 — Quantstamp Verified

DeFi Dollar

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

Executive Summary

Type	Stablecoin backed by Curve tokens
Auditors	Jan Gorzny, Blockchain Researcher Poming Lee, Research Engineer Ed Zulkoski, Senior Security Engineer
Timeline	2020-07-28 through 2020-08-20
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Blog Post
Documentation Quality	Low
Test Quality	Low

Source Code

Repository	Commit
defidollar-core	b6a77bd

Goals

- Review the staking mechanism
- Review the rewards mechanism
- Review the proxy pattern implemented by the code

Total Issues	7 (1 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	2 (0 Resolved)
Informational Risk Issues	5 (1 Resolved)
Undetermined Risk Issues	0 (0 Resolved)

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

Summary of Findings

The project is mostly well-written, though tests and comments could both be improved. This audit discovered several issues, all of which are informational or low severity. In many cases, these issues may not be avoidable due to system requirements, but users should be aware of them. Please note that only the Solidity files in the repository were reviewed.

ID	Description	Severity	Status
QSP-1	Omitted SafeMath	Low	Acknowledged
QSP-2	Gas Usage / `for` Loop Concerns	Low	Acknowledged
QSP-3	Unclear Integer Sizes	Informational	Acknowledged
QSP-4	Unchecked Input Parameters	Informational	Acknowledged
QSP-5	Possible Duplicate Coins	Informational	Fixed
QSP-6	Privileged Roles and Ownership	Informational	Acknowledged
QSP-7	Clone-and-Own	Informational	Acknowledged

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 Omitted SafeMath

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `Oracle.sol`

Description: The `SafeMath` library prevents overflows and underflows, and should be used everywhere to prevent this behaviour.

Recommendation: Use the `SafeMath` library in `Oracle.sol`.

QSP-2 Gas Usage / `for` Loop Concerns

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `Core.sol`, `CurveSusdPeak.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible.
The functions `mint`, `mintWithCurvePoolTokens`, `redeem`, `portfolio`, and `replenish_approvals` in `CurveSusdPeak.sol` and the functions `redeem` and `_updateFeed` in `Core.sol` have `for`-loops.

QSP-3 Unclear Integer Sizes

Severity: *Informational*

Status: Acknowledged

Description: Many files declare integers as `uint` instead of `uint256`. This makes the size of these variables unclear, and obfuscates the maximum range of values stored as integers.

Recommendation: Change `uint` declarations to `uint256` (or other sizes) where appropriate.

QSP-4 Unchecked Input Parameters

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Core.sol`, `CurveSusdPeak.sol`, `StakeLPToken.sol`

Description: Many input parameters are assumed to be non-trivial, but these conditions are not checked. For example, addresses of target contracts should not be `0x0`. These checks are omitted from the various `initialize` functions in the system.

Recommendation: Require that the input addresses for initialisation functions are non-zero.
Update: this was fixed in `Core.sol`.

QSP-5 Possible Duplicate Coins

Severity: *Informational*

Status: Fixed

File(s) affected: `Core.sol`

Description: The function `whitelistPeak()` does not check that the items in `_systemCoins` are unique. This could tamper with the pricing logic associated with a peak if the same coin is double-counted. This uniqueness check could be linearly achieved if `_systemCoins` is assumed sorted.

Recommendation: Ensure that the items in `_systemCoins` are unique.

QSP-6 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Core.sol`, `UpgradableProxy.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. However, 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.
In this case, the owner can set any `redeemFee` in the constructor and the owner can add peaks and system coins arbitrarily; the latter privileges may enable the owner to mint an arbitrary amount of DUSD tokens.
Since the system follows an upgradable contract pattern, the admin may also be able to make arbitrary calls via `execute` in `UpgradableProxy.sol`

Recommendation: There is no recommendation at this time; however, an requirement that the `redeemFee` is at least 10,000 may be desirable, as the variable is used in L161: `dusdAmount = usdToDusd(usdDelta).mul(redeemFee).div(10000);` where values less than 10,000 may not be desirable.

QSP-7 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Ownable.sol`, `Initializable.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. Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.

Automated Analyses

Slither

Slither reported that several functions are prone to reentrancy, although all were false positives. Other issues reported by Slither are presented in the best practices section.

Adherence to Specification

No technical specification was provided, so we cannot comment on the code's adherence. A description is provided in the linked blog post.

Code Documentation

While the `README.md` file contains compilation and testing instructions, it lacks a description of the system entirely. Furthermore, many non-trivial functions lack comment blocks in the code.

Adherence to Best Practices

- `CurveSusdPeak.replenish_approvals()` ignores the return values of `curveToken.approve` and `IERC20(...).approve(...)`; these calls should be in a `require` statement. Similar cases are irrelevant in mock files, as those are for testing.
- `Oracle.getPriceFeed` uses a numerical value with too many digits; in this case, an exponentiation would improve readability. **Update:** fixed.
- `StakeLPToken.sol` uses `1e18` in various places: consider naming this value to enable easier readability and code maintainability.
- The constants `MAX` and `PRECISION` in `Core.sol` are not used; they can be removed. **Update:** fixed.
- Various functions, including `CurveSusdPeak.portfolio`, `IPeak.portfolio()`, `CurveSusdPeak.intialize`, `UpgradableProxy.updateAndCall`, `StakeLPToken.initialize`, `Oracle.getPriceFeed()`, `Core.initialize`, and `Core.lastPeriodIncome()`, could be made `external`.
- There are spelling mistakes in `Core.sol`: L288 "Intialize" and L227 "accross". **Update:** fixed.
- In `Core.sol`, on L154 of `redeem()`, there is the optimization `if (delta[i] == 0) continue;`. The same check could occur in `mint()`. **Update:** fixed/removed.
- In `Core.sol`, the function `whitelistTokens()`, should ensure that the array lengths are equal. **Update:** fixed/removed.
- In `Core.sol`, the if-statement on L232, `if (peak.state != PeakState.Extinct)`, should happen before the loop. **Update:** fixed/removed.
- A function to undo the action of `replenish_approvals` in `CurveSusdPeak.sol` may be desirable.
- The code may be clearer if `rewardPerToken` was named `deltaRewardPerToken` in `StakeLPToken.sol`.
- In `Core.sol`, using `PeakState.Extinct` implies that the peak may have once existed; perhaps renaming `Extinct` to `NonExistent` or something similar may remove this implication.
- Some mock files do not use `SafeMath`; it's good code hygiene to use the library everywhere.
- While most `external`/`public` functions are in `mixedCase` and most `internal`/`private` functions are in `snake_case`, `replenish_approvals` in `CurveSusdPeak.sol` does not follow this trend. **Update:** fixed.

Test Results

Test Suite Results

The tests consist largely of a few scenarios. Some output from the tests has been removed for this report.

```
Contract: Deficit flow (staked funds cover deficit)
   ✓ bob mints ~110 dusd (678ms)
   ✓ bob transfers 10 to alice (40ms)
   ✓ alice stakes 10 (209ms)
   ✓ 10 are withdrawable for alice
   ✓ alice withdraws 2 (203ms)
   ✓ 8 are withdrawable for alice
   ✓ drop price to create deficit of ~4 (461ms)
   ✓ ~3 are withdrawable for alice
   ✓ reverts if alice withdraws 4 (166ms)
   ✓ alice exits (289ms)

Contract: Deficit flow (staked funds don't cover deficit)
   ✓ charlie mints 40 dusd (471ms)
   ✓ bob mints 120 dusd (707ms)
   ✓ bob transfers 10 each to alice and charlie (85ms)
   ✓ alice stakes 10 (185ms)
   ✓ 10 are withdrawable for alice
   ✓ drop coin price to 0 (487ms)
   ✓ 0 are withdrawable for alice
   ✓ reverts if alice attempts to withdraw 1 wei (154ms)
   ✓ dusd is devalued while redeeming
   ✓ all dusd can be redeemed (race to exit was avoided) (591ms)
   ✓ all except staked dusd was redeemed (76ms)
```

```
    ✓ minting and redeeming devalued dusd is unprofitable (1134ms)

  Contract: Core
    ✓ mint (74ms)
    ✓ redeem (80ms)
    ✓ set peak as dormant (40ms)
    ✓ redeem is possible from dormant peak (82ms)
    ✓ mint fails for dormant peak (54ms)
    ✓ set peak as extinct (43ms)
    ✓ mint fails for dormant peak (53ms)
    ✓ redeem fails from extinct peak (49ms)
    ✓ adding a duplicate token fails (78ms)
    Only Owner
      ✓ whitelistTokens fails (61ms)
      ✓ setPeakStatus fails (54ms)

  Contract: core-client-lib: CurveSusdPeak
    ✓ curveSusd.add_liquidity (421ms)
    ✓ peak.mintWithScrv (2276ms)
    ✓ curveSusdPeak.mint (7877ms)
    ✓ peak.redeem: Alice redeems 1/2 her dusd (3592ms)
    ✓ peak.redeemInSingleCoin(3): Alice redeems 1/2 her leftover dusd (1812ms)
    ✓ peak.redeemInScrv (2808ms)
    ✓ curveSusd.remove_liquidity (144ms)

  Contract: core-client-lib: StakeLPToken
    ✓ alice mints 1M (CurveSusdPeak) (1406ms)
    ✓ alice stakes=500k (1867ms)
    ✓ CurveSusdPeak accrues income=4 (162ms)
    ✓ apy (2169ms)

  Contract: core-client-lib: StakeLPToken
    ✓ alice mints 40 (CurveSusdPeak) (1309ms)
    ✓ bob mints 20 (CurveSusdPeak) (1656ms)
    ✓ alice stakes=4 (2254ms)
    ✓ CurveSusdPeak accrues income=4 (135ms)
    ✓ alice gets reward (2559ms)
    ✓ CurveSusdPeak accrues income=8 (289ms)
    ✓ bob redeems=4 (3715ms)
    ✓ should not affect lastPeriodIncome (180ms)
    ✓ bob stakes=2 (2257ms)
    ✓ CurveSusdPeak accrues income=6 (136ms)
    ✓ bob exits (2705ms)
    ✓ CurveSusdPeak accrues income=3 (129ms)
    ✓ alice withdraws stake (2375ms)
    ✓ alice exits (356ms)

  Contract: StakeLPToken
    ✓ alice mints 40 (CurveSusdPeak) (689ms)
    ✓ bob mints 20 (CurveSusdPeak) (789ms)
    ✓ alice stakes=4 (533ms)
    ✓ CurveSusdPeak accrues income=4 (145ms)
    ✓ alice gets reward (360ms)
    ✓ CurveSusdPeak accrues income=8 (306ms)
    ✓ bob redeems=4 (577ms)
    ✓ should not affect lastPeriodIncome (188ms)
    ✓ bob stakes=2 (224ms)
    ✓ period income was transferred to stakeLPToken for rewards
    ✓ CurveSusdPeak accrues income=6 (141ms)
    ✓ bob exits (359ms)
    ✓ CurveSusdPeak accrues income=3 (150ms)
    ✓ 10% admin fee was introduced (98ms)
    ✓ alice withdraws stake (181ms)
    ✓ admin withdraws fee (69ms)
    ✓ alice exits (338ms)

  Contract: StakeLPToken
    ✓ alice mints 40 (CurveSusdPeak) (719ms)
    ✓ CurveSusdPeak accrues income=4 (139ms)
    ✓ alice stakes=4 (493ms)
    ✓ updateProtocolIncome was triggered correctly
    ✓ CurveSusdPeak accrues income=8 (127ms)
    ✓ lastPeriodIncome (121ms)
    ✓ drop coin prices to $0.5 (480ms)

  Contract: CurveSusdPeak
    ✓ curveSusd.add_liquidity (302ms)
    ✓ peak.mintWithScrv (225ms)
    ✓ curveSusdPeak.mint (540ms)
    ✓ peak.redeem: Alice redeems 1/2 her dusd (256ms)
    ✓ peak.redeemInSingleCoin(3): Alice redeems 1/2 her leftover dusd (474ms)
    ✓ peak.redeemInScrv (181ms)
    ✓ curveSusd.remove_liquidity (127ms)

  Contract: CurveSusdPeak
bob got 400 yCRV
    ✓ curveSusd.add_liquidity: bob seeded initial liquidity of 400 in curve (529ms)
bob got 40 DUSD
    ✓ curveSusdPeak.mintWithScrv: bob minted dusd with 1/10 of their Scrv (214ms)
    ✓ alice took a flash loan for 1000 dai and 600 usdc (83ms)
alice got 961.612394626043743547 yCRV
    ✓ curveSusd.add_liquidity: alice dumped 1000 dai in curve (290ms)
alice got 605.602485702808449937 DUSD
    ✓ curveSusdPeak.mint(600 usdc): alice mints dusd with usdc (490ms)
    ✓ curveDeposit.remove_liquidity_one_coin(dai): alice (532ms)
    ✓ curveSusdPeak.redeemInSingleCoin(usdc): alice (452ms)
    ✓ alice did not make a profit


  97 passing (1m)
```

Code Coverage

Code coverage is fairly high, though the overall branching level is low.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
base/	96.36	75	100	94.74	
Core.sol	96.19	75.86	100	94.44	… 217,239,360
CoreProxy.sol	100	100	100	100	
DUSD.sol	100	50	100	100	
common/	90	66.67	85.71	92.86	
Initializable.sol	100	50	100	100	
Ownable.sol	87.5	75	83.33	90.91	36
common/mocks/	50	100	66.67	50	
Reserve.sol	50	100	66.67	50	25,26
common/proxy/	86.67	50	87.5	89.47	
IERCProxy.sol	100	100	100	100	
Proxy.sol	0	100	50	50	35
UpgradableProxy.sol	92.86	50	100	94.12	46
interfaces/	100	100	100	100	
ICore.sol	100	100	100	100	
IDUSD.sol	100	100	100	100	
IOracle.sol	100	100	100	100	
IPeak.sol	100	100	100	100	
IStakeLPToken.sol	100	100	100	100	
peaks/curve/	92.63	57.69	91.3	92.63	
CurveSusdPeak.sol	92.63	57.69	90.91	92.63	… 179,184,185
CurveSusdPeakProxy.sol	100	100	100	100	
ICurve.sol	100	100	100	100	
IGauge.sol	100	100	100	100	
peaks/curve/mocks/	100	100	75	100	
Gauge.sol	100	100	80	100	
Mintr.sol	100	100	0	100	
MockSusdToken.sol	100	100	100	100	
stream/	88.89	100	66.67	88.89	
Oracle.sol	88.89	100	66.67	88.89	41
stream/mocks/	100	100	100	100	
MockAggregator.sol	100	100	100	100	
valley/	100	87.5	100	100	
StakeLPToken.sol	100	87.5	100	100	
StakeLPTokenProxy.sol	100	100	100	100	
All files	**94.5**	**71.77**	**91.67**	**94.12**	

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

```
e574e18a19cd60d87ff43d4d6fcf35f166ba3677fd8918c8e3cd5d710b6b423f ./contracts/Migrations.sol
8cce662b687bc2a28f79a2ad6fef292d6a344c30f860d428f6b7cdcb9a7767f2 ./contracts/valley/StakeLPToken.sol
ac16d7daa9ec76350cbcf915f8e1bf26eecaf60e0f2bbb01ec472e64b880e9d9 ./contracts/valley/StakeLPTokenProxy.sol
8ee38a3e13abbd202f7c2e2ebc0c2531828c557f04eed6763ddd181dd1780bef ./contracts/stream/Oracle.sol
193644d57d58ea4b48b0b45bdbb0987c799e705f6aa1bbc9da26884239dd7e99 ./contracts/stream/mocks/MockAggregator.sol
b821492fb4d8415d308530c35859c0a585e5f2a16944bcce644f5a1347605ecf ./contracts/peaks/curve/CurveSusdPeak.sol
33a0b950f7774783bd36bdacd6014cd791c475b89d221ffdb84cc4a17782b740 ./contracts/peaks/curve/CurveSusdPeakProxy.sol
6dd626323ced0f9b2810d6419138d5c26b250d67d9d5ee974718a486507030c3 ./contracts/peaks/curve/ICurve.sol
5318ac74fd0d57b2f8e1194da7fc02f8b6b835aecc9f96853ced85701c1dca5e ./contracts/peaks/curve/IGauge.sol
448f6aa855a5ca9639f588e0c32f49efb53b4e00f4f8292d21a07bbe0e737066 ./contracts/peaks/curve/mocks/Gauge.sol
d0e5209d5520ff48d0417347effb2ec52b46c8e8d9fe0575dffd4063479118cd ./contracts/peaks/curve/mocks/Mintr.sol
69363d68b170c2b1cafe85f9b326c429c70e705c33308db14c0c5d5c0b2e4a81 ./contracts/peaks/curve/mocks/MockSusdToken.sol
86bcbe16e3ebd7c9474981e4fd82447e2e469c869b3ba765aadf80f41c3e75d0 ./contracts/interfaces/ICore.sol
d01b18a1eeea41c01982491639d6dd63993cbd3d367f2ad1804e2f8575c57495 ./contracts/interfaces/IDUSD.sol
6f8335b77badd2a637223b3c8aea908207b9a2603e17726fe5ae8f13fbfe28c2 ./contracts/interfaces/IOracle.sol
cef75cfa097c0ceacdb07a665fe421197bc27a4006b527b27013aa977b39ba3f ./contracts/interfaces/IPeak.sol
b8e5224696494caf82b3ed85853d611eec310ae9521331b456c958324b475f5a ./contracts/interfaces/IStakeLPToken.sol
8224d630aa9b7b87bfe47029cdefdb9aff8aaf371efa808551230517fc85a357 ./contracts/common/Initializable.sol
79a952c6d99a88fb54392c03f4f91415f52aa80a49de93c442b5fd4dfc1c4072 ./contracts/common/Ownable.sol
00aee5d6d2b68330a937b97d4127ebac79a55f6073eed5ffe383de935456c5b7 ./contracts/common/proxy/IERCProxy.sol
aefc6bab2478153c637dea9ebdb4353d143af1bb9aa1d85b97fbb66ff80e11f3 ./contracts/common/proxy/Proxy.sol
aeed9a2484a92a59f205cd3bbc96b58375c86d99d9a0fdcb4a1c06144a41ed34 ./contracts/common/proxy/UpgradableProxy.sol
09f7c442321152a7e22a7a5ab0315319dee1a2d5aea4740069c8b54aaa0a56e1 ./contracts/common/mocks/Reserve.sol
2ba6624b505fadd9122c983f1479a65abda7054807ea8e2a111459ec8483372d ./contracts/base/Core.sol
bf0623b761f9eb34141b2761a31e110e514ba6a63b814cfa8635f7fcbb2a92f2 ./contracts/base/CoreProxy.sol
2eba64482e5a2aa15e00e4219960faf57e7f0904d37885ac99a694d3a424e9e6 ./contracts/base/DUSD.sol
```

Tests

```
aa156a3c04d967830953c8e4acb6cd03a4713f92e0ca8efa80ba4a7756e798d8 ./test/Deficit.test.js
819609fac4bb07c2f24609eda99c69754de082e2669029013abace53232ac60b ./test/utils.js
0a929850f68b6f54a390711d2501c6524f59cb7231f629e29e13ae906665a24c ./test/valley/StakeLPToken.test.js
9135d397fcf1264dd9b85f194df12194b66fab92780ec456de2d94de24814f18 ./test/valley/StakeLPToken2.test.js
2ece40f2d1face3c18bf4681ffa2e759f132af3901ae4d131ff7ac76704eb8ec ./test/peaks/curve/CurveSusdPeak.test.js
d3fb8fc714f61c06a8fc0d7c9ba8c90666128358385534a91b25e8495f6249f4 ./test/peaks/curve/CurveSusdPeak2.test.test.js
3c6cb26b65778a72abd84d157957fb1aeabf198d523e1c2ea52b264e42cd282c ./test/core-client-lib/CurveSusdPeak.test.js
350e1bbbbbbf2b158a304bde66eb6d989fd7d53b7152d1a473af82f1a74d33b9 ./test/core-client-lib/StakeAPY.test.js
e9c5b7a3e0969c89fc8ad7cb42447125ca5d1bfbeebc34f4594ebf1526b0e579 ./test/core-client-lib/StakeLPToken.test.js
0a38d57cb5972d9bd58ac5fd19efc82cc175d1c13774239eb3d672b5b5b3b8dd ./test/base/Core.test.js
```

Changelog

- 2020-08-10 - Initial report [`5e21b06`]
- 2020-08-20 - Revised report [`b6a77bd`]

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.