Quantstamp Security Assessment Certificate

January 28th 2021 — Quantstamp Verified

Mimo DeFi - Parallel Protocol

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

Executive Summary

Type	Decentralized Stablecoin
Auditors	Kacper Bąk, Senior Research Engineer Kevin Feng, Blockchain Researcher Jan Gorzny, Blockchain Researcher
Timeline	2020-11-02 through 2020-11-19
EVM	Muir Glacier
Languages	Solidity, Typescript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Whitepaper
Documentation Quality	High
Test Quality	High

Source Code

Repository	Commit
titan	680fa1a

Goals

- Can funds get locked in the contract?
- Can user's collateral get liquidated prematurely?

Total Issues	**8** (4 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**6** (3 Resolved)
Informational Risk Issues	**1** (1 Resolved)
Undetermined Risk Issues	**1** (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

We have found a few low-severity issues and two issues of undetermined severity. Overall, the code is well-written, well-documented, and well-tested. We recommend addressing all the issues to ensure code correctness with respect to the intent and to ensure it follows best practices.

ID	Description	Severity	Status
QSP-1	Privileged Roles and Ownership	Low	Acknowledged
QSP-2	Dependency on external contracts	Low	Acknowledged
QSP-3	Liquidation may be impossible is there is not funds in the insurance fund	Low	Acknowledged
QSP-4	Various calls to ERC20 functions are not in `require` statements	Low	Fixed
QSP-5	Lack of input validation	Low	Fixed
QSP-6	Potential division by zero	Low	Fixed
QSP-7	Unlocked Pragma	Informational	Fixed
QSP-8	Potential sensitivity to price movements	Undetermined	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 v0.6.12

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 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `EURX.sol`, `USDX.sol`, `FeeDistributor.sol`, `AddressProvider.sol`, `ConfigProvider.sol`, `PriceFeed.sol`, `PriceFeedEUR.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. Specifically, the contract `EURX` and `USDX` have a special minter role. The contracts `FeeDistributor`, `AddressProvider`, `ConfigProvider`, `PriceFeed`, `PriceFeedEUR` feature a manager role. Notably, in `ConfigProvider.sol` manager can set arbitrary debt limit, collateral ratio, borrow rate, etc.

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.
Update: the team informed us that they plan on decentralizing the governance before the full launch.

QSP-2 Dependency on external contracts

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `PriceFeed.sol`, `PriceFeedEUR.sol`, `VaultsCore.sol`

Description: The contracts `PriceFeed`, `PriceFeedEUR`, and `VaultsCore` depend on external contracts. Furthermore, `PriceFeed`, `PriceFeedEUR` depend on `decimals()` to determine price accuracy. It is important to note that according to EIP-20, this method/field is optional. `VaultsCore` depends on the `transfer()` functionality of external tokens.

Recommendation: We recommend vetting the underlying tokens carefully to ensure they are compatible with TenX contracts and that they behave as expected.

QSP-3 Liquidation may be impossible is there is not funds in the insurance fund

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `VaultsCore.sol`

Description: Due to the condition in L293 liquidation may be impossible if there is not enough funds in the insurance fund in a corner case when value of collateral drops below debt value.

Recommendation: The team has acknowledged this corner case scenario and informed us that they are planning on creating further fallback (e.g., automatic selling of governance token) once governance is in place. We do not have any further recommendations at this time.

QSP-4 Various calls to ERC20 functions are not in `require` statements

Severity: *Low Risk*

Status: Fixed

File(s) affected: `VaultsCore.sol`

Description: Calls to ERC20 functions on external assets are not wrapped in `require()` statements.

Recommendation: Since both functions return boolean values, we recommend wrapping them in `require()` statements.

QSP-5 Lack of input validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AddressProvider.sol`, `LiquidiationManager.sol`, `ConfigProvider.sol`, `PriceFeed.sol`, `PriceFeedEUR.sol`, `RatesManager.sol`, `VaultsCore.sol`, `VaultsDataProvider.sol`, `FeeDistributor.sol`, `EURX.sol`, `USDX.sol`

Description: Functions do not check if arguments of type address are non-zero in the following:

- `AddressProvider.sol`, functions: `setAccessController()`, `setConfigProvider()`, `setVaultsCore()`, `setStableX()`, `setRatesManager()`, `setPriceFeed()`, `setVaultsDataProvider()`, `setFeeDistributor()`.
- `ConfigProvider.setCollateralConfig()`.
- `LiquidiationManager.constructor()`.
- `PriceFeed.setAssetOracle()`.
- `PriceFeedEUR.sol`, functions: `setAssetOracle()` and `setEurOracle()`.
- `RatesManager.constructor()`.
- `VaultsCore.sol`, functions: `constructor()`, `upgrade()`, `initializeRates()`, `refreshCollateral()`, and `deposit()`.
- `VaultsDataProvider.sol`, functions: `constructor()` and `createVault()`.
- `FeeDistributor.constructor()`.
- `EURX.constructor()`.
- `USDX.constructor()`.

Recommendation: We recommend adding the relevant checks.

QSP-6 Potential division by zero

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LiquidationManager.sol`

Description: There is a chance that `ConfigProvider.minCollateralRatio` is set to 0. In case that happens, the division in `LiquidationManager.sol#63` would fail due to division by 0.

Recommendation: Verify if this is the intended behavior. If so add a require/if statement to check if `minRatio` in L62 is 0.

QSP-7 Unlocked Pragma

Severity: *Informational*

Status: Fixed

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.6.*`. 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.

QSP-8 Potential sensitivity to price movements

Severity: *Undetermined*

Status: Acknowledged

Description: As with many DeFi project which depend on external oracles, if an attacker can manipulate the market (and, consequently, the price as reported by oracles), they may be able to liquidate other participants' vaults when the collateral value drops below a liquidation threshold.

Recommendation: We recommend informing users about a variety of potential DeFi risks and attacks.

Automated Analyses

Slither

Slither reported the following:

- reentrancy in `FeeDistributor.changePayees()` and `ConfigProvider.setCollateralConfig()`. We consider it a false positive because both are protested using `onlyManager` modifier and the manager is assumed to be a trusted entity.
- ignored return values in `VaultsCore.sol#51`, `VaultsCore.sol#56`, `VaultsCore.sol#133`, `VaultsCore.sol#163`, and `VaultsCore.sol#309`. We recommend checking the return values.

Adherence to Specification

1. The flowchart for function `borrow` is not in the same order as the implementation. The health check occurs after updating total balance, total income and total debt.

Code Documentation

The code is well documented.

Adherence to Best Practices

1. In `RatesManager.sol`, the modifier `onlyVaultsCore` is unused. **Update:** fixed.
2. In `FeeDistributor.sol#15` and `IConfigProvider.sol#50`, TODO items. **Update:** fixed.

Test Results

Test Suite Results

The project features a comprehensive test suite.

```
Contract: AddressProvider
   ✓ should initialize address provider with correct controller & empty addresses for everything else (180ms)
   ✓ manager should be able to update addresses (536ms)
   ✓ non-manager should NOT be able to update addresses (734ms)
  - every module should have the addressProvider readable & setable

Contract: ConfigProvider
   ✓ should initialize config provider with correct addressProvider & correct default values (47ms)
   ✓ manager should be able to add a collateral config (171ms)
   ✓ manager should be able to update existing collateral config (1583ms)
   ✓ manager should be able to set a liquidation bonus (54ms)
   ✓ NON-manager should NOT be able to set a liquidation bonus (39ms)

Contract: FeeDistributor
   ✓ should initialize fee distributor as minter
   ✓ should initialize with total shares
   ✓ should initialize with payees and shares (83ms)
   ✓ should initialize cumulative and current rate to 1
   ✓ available income should be 0 after initialize (50ms)
   ✓ should be able to accrue fees (533ms)
   ✓ should be able to release accrued fees to payees (905ms)
   ✓ should allow updating payees (122ms)
   ✓ release without any payees configured should fail (631ms)
  - changePayees should distribute existing income
  - updating payees should work correctly

Contract: Liquidation Manager
   ✓ should calculate the correct health factor & isHealthy (249ms)
   ✓ calculate a correct liquidation bonus
   ✓ calculate a correct applyLiquidationDiscount
   ✓ should throw error for isHealthy when collateralization ratio is not defined for collateral

Contract: PriceFeed
   ✓ Access controller initializes correctly (42ms)
   ✓ Feed initializes correctly (51ms)
   ✓ Aggregator initializes correctly
   ✓ Non-feed managers cannot add oracles (43ms)
   ✓ Feed manager can add oracles (55ms)
   ✓ Price feed returns data
   ✓ should convert stablecoin to collateral (44ms)
   ✓ should convert collateral to stablecoin (51ms)
```

```
    ✓ should prevent overflow errors when the aggregater returns a negative answer (45ms)

Contract: PriceFeedEUR
    ✓ Access controller initializes correctly (42ms)
    ✓ Feed initializes correctly (53ms)
    ✓ Aggregator initializes correctly
    ✓ Non-feed managers cannot add oracles (49ms)
    ✓ Feed manager can add oracles (104ms)
    ✓ Price feed returns data (51ms)
    ✓ should convert stablecoin to collateral (78ms)
    ✓ should convert collateral to stablecoin (84ms)
    ✓ should prevent overflow errors when the aggregater returns a negative answer (76ms)
    ✓ should prevent overflow errors when the EUR aggregater returns a negative answer (78ms)

Contract: RatesManager calculations
    ✓ test helpder function cumulativeRateHelper should calculate correctly
    ✓ debt should equal baseDebt at rate 1
  - lastRefresh should be set correctly
  - Cumulative Rate should be correctly calculated for an interval as small as 1 second
  - Base Debt should be correctly calculated and rounded up to make sure totaldebt is never below withdrawn amount
  - Incremental cumulativeRate updates should be the same as infrequent updates
  - Should not allow for buffer overflow on interest rates
  - Should not allow for buffer overflow on debt calculations
  - Cumulative Rate update function should work with 0 time has passed
  - Updating the borrow rate should correctly calculate previous rate increase

Contract: USDX
    ✓ USDX initializes correctly (99ms)
    ✓ deployer is minter
    ✓ deployer can mint tokens (72ms)
    ✓ non-deployer address shall not be able to mint (47ms)
    ✓ deployer can burn tokens (110ms)
    ✓ non-deployer address shall not be able to burn tokens (101ms)
    ✓ should be able to update the allowance with approve without having to go to 0 (194ms)

Contract: EURX
    ✓ USDX initializes correctly (112ms)
    ✓ deployer is minter (41ms)
    ✓ deployer can mint tokens (79ms)
    ✓ non-deployer address shall not be able to mint (52ms)
    ✓ deployer can burn tokens (139ms)
    ✓ non-deployer address shall not be able to burn tokens (108ms)
    ✓ should be able to update the allowance with approve without having to go to 0 (200ms)

Contract: VaultsCore config & access control
    ✓ should be able to deploy vaults core (250ms)
  - manager should be able to update the rates Module
  - non-manager should not be able to update the rates Module
  - non-manager should not be able to update interest rates

  Contract: VaultsCore Debt Limits
    ✓ should be able to borrow up to the collateral debt limit, but not beyond (396ms)
    ✓ should be able to update the collateral debt limit (299ms)

  Contract: VaultsCore Debt
    ✓ total debt outstanding should be correctly calculated without interest (268ms)
    ✓ total debt outstanding should be correctly calculated with interest applied (345ms)
  - total debt outstanding should be correctly calculated for multiple vaults

  Contract: VaultsCore GAS costs
gasUsed deposit: 230557
    ✓ GAS for depositing collateral should not exceed 240k GAS (132ms)
gasUsed withdraw: 55065
    ✓ GAS for withdrawing collateral from no debt vault should not exceed 180k GAS (176ms)
gasUsed withdraw: 226953
    ✓ GAS for withdrawing collateral from vault with debt should not exceed 280k GAS (397ms)
gasUsed borrow 335135
    ✓ GAS for borrowing should not exceed 380k GAS (328ms)
gasUsed repay: 205645
    ✓ GAS for repayment should not exceed 260k GAS (414ms)

  Contract: VaultsCore income
    ✓ should calculate income correctly for a single vault (477ms)
    ✓ should add origination fee to income when borrowing (351ms)
    ✓ withdraw should update income and cumulative rate (712ms)
  - should calculate income correctly for multiple vaults
  - should calculate income correctly with changing interest rates
  - should calculate income correctly when repaying
  - should calculate income correctly when liquidating
  - should calculate income correctly when borrowing more

  Contract: VaultsCore liquidation
    ✓ should not allow to liquidate when healthy (463ms)
    ✓ should allow liquidation when healthfactor below 1 (638ms)
    ✓ Should NOT be able to liquidate vaults without debt (309ms)
    ✓ should allow liquidation with insurance fund (722ms)
    ✓ liquidation that requires insurance should fail if insurance fund is low/empty (708ms)
  - allow liquidation after setMinCollateralizationRatio has reduced the ratio

  Contract: VaultsCore rates multi collateral
    ✓ interest rate for WETH and WBTC should be properly initialized
    ✓ Cumulative Rates for each collateral should change as time passes (142ms)
    ✓ should be possible to remove collateral type (150ms)
  - should be able to configure different margin limits for each collateral
  - margin limits should be enforced correctly for different collateral types
  - total debt for 2 vaults with different colalteral should be calculated correctly

  Contract: VaultsCore rates
    ✓ should initialize cumulative and current rate to 1
    ✓ should initialize the last rate update timestamp to deployment timestamp
    ✓ should be able to set a new rate (84ms)
    ✓ Cumulative Rate should change as time passes (155ms)
    ✓ setBorrowRate should emit CumulativeRateUpdated event if time has passed (151ms)
    ✓ setBorrowRate should update income (500ms)

Contract: VaultsCore EUR vaults
    ✓ vault owner cannot withdraw and let the health factor go below 1 (582ms)
    ✓ vault owner cannot borrow and let the health factor go below 1 (648ms)

Contract: VaultsCore vaults
    ✓ user can open a vault via a deposit (122ms)
    ✓ depositing without sufficient allowance should fail (138ms)
    ✓ depositing with sufficient allowance should add to vault balance (153ms)
    ✓ non-owners of a vault cannot withdraw (167ms)
    ✓ vault owner cannot withdraw more than vault balance (185ms)
    ✓ vault owner can withdraw part of his vault collateral (193ms)
    ✓ vault owner can borrow (343ms)
    ✓ vault owner can borrow with an origination fee (371ms)
    ✓ should calculate origination fee correctly after time has passed (518ms)
  - should calculate income correctly with orignation fee
    ✓ vault owner can repay partially (454ms)
    ✓ vault owner can repay fully (438ms)
    ✓ vault owner can repay without specifying amount (448ms)
    ✓ vault owner cannot withdraw and let the health factor go below 1 (732ms)
    ✓ vault owner cannot borrow and let the health factor go below 1 (628ms)
    ✓ upgrading to new vaultscore should approve tokentranfers for both stablex and collateral types (92ms)
  - should enforce minimum margin limit during borrowing
  - should enforce minimum margin limit during collateral withdrawal
  - should properly account for over-repayment
  - should not allow to open vault with not authorized collateral
  - should allow to withdrawl all collateral with withdrawAll
  - Should not allow to borrow below the minimum to allow for efficient liquidiation (gas costs)
  - should throw proper error when accessing non-existant vault


100 passing (2m)
32 pending
```

Code Coverage

The project has a good coverage. We recommend improving the coverage in `RatesManager.sol`.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
access/	100	100	100	100	
AccessController.sol	100	100	100	100	
chainlink/	100	100	100	100	
AggregatorV3Interface.sol	100	100	100	100	
core/	94.72	81.94	95.35	94.53	
AddressProvider.sol	100	100	100	100	
ConfigProvider.sol	81.58	75	100	83.33	… 63,64,65,66
LiquidiationManager.sol	100	100	100	100	
PriceFeed.sol	100	100	100	100	
PriceFeedEUR.sol	95.83	87.5	100	96	54
RatesManager.sol	77.78	50	66.67	66.67	24,25,34
VaultsCore.sol	98.04	85.29	94.44	98.1	174,175
VaultsDataProvider.sol	97.14	62.5	93.33	97.22	124
fees/	96.55	55.56	100	96.88	
FeeDistributor.sol	96.55	55.56	100	96.88	104
interfaces/	100	100	100	100	
IAccessController.sol	100	100	100	100	
IAddressProvider.sol	100	100	100	100	
IConfigProvider.sol	100	100	100	100	
IFeeDistributor.sol	100	100	100	100	
ILiquidationManager.sol	100	100	100	100	
IPriceFeed.sol	100	100	100	100	
IRatesManager.sol	100	100	100	100	
ISTABLEX.sol	100	100	100	100	
IVaultsCore.sol	100	100	100	100	
IVaultsDataProvider.sol	100	100	100	100	
libraries/	78.26	100	66.67	78.26	
MathPow.sol	100	100	100	100	
WadRayMath.sol	72.22	100	63.64	72.22	33,37,61,63,67
mocks/	59.38	100	64.29	59.38	
MockChainlinkAggregator.sol	60	100	66.67	60	… 60,61,62,63
MockChainlinkAggregatorEUR.sol	53.33	100	50	53.33	… 60,61,62,63
MockWBTC.sol	100	100	100	100	
MockWETH.sol	100	100	100	100	
token/	100	100	100	100	
EURX.sol	100	100	100	100	
USDX.sol	100	100	100	100	
All files	**90.62**	**78.57**	**89.84**	**90.73**	

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

```
b4b245d1a1d8c961457f7cc8e6b6d2d4b8345e76f8692870e7d28e56934019d7 ./contracts/Migrations.sol
151f9bf761e8cf87b727c3b6bd3349199aeeb85feb43754a4ba490556c54f6a2 ./contracts/token/EURX.sol
e70fc294ccccc862b6ede5297d145348993a3b3acc91d3671c1c3112f732bb42 ./contracts/token/USDX.sol
7d090fa097a655dc50831b7385cc6e394b00a96d07fa6dccfd8a5ac2972dfdba ./contracts/mocks/MockBuggyERC20.sol
45dffb475856d68956f6c1e1f85cb77d0d47be621dd39e00961962d06cfdebe7 ./contracts/mocks/MockChainlinkAggregator.sol
f086725775085f861a0d45e0140c1834e1f85dc0b1b7b7e3fe1cd5bc95ec3620 ./contracts/mocks/MockChainlinkAggregatorEUR.sol
f75fd63cfd49a6069f7110d80b8e0990d136cc0cccc247875826e3182aa1cb11 ./contracts/mocks/MockWBTC.sol
0a3ff072e0e7e1cfc707fcdb10ed612b9e5387927abfc8989aeb97c5da0ee28f ./contracts/mocks/MockWETH.sol
92b70f40e163d5df7685fa8afc508d9b4dad62c2dd945e63fb7a5084ef3389f7 ./contracts/libraries/MathPow.sol
de8f20bd936811cfca6c970831e9cd0420119b51202efa2f06e60d3679f45af2 ./contracts/libraries/WadRayMath.sol
70f13d333ebff96bdd8f22d0474c182d080382684b0134f88d2ee886a6d6dcfe ./contracts/interfaces/IAccessController.sol
ae76da8aeba2dc45c2e752703a1ed9f159258a1a518e3cfd0085063b7e1d0bc3 ./contracts/interfaces/IAddressProvider.sol
5a53d27c4c4dffce62dc066a0e6a5d465a33bf22fed8778b68827708658682fe ./contracts/interfaces/IConfigProvider.sol
f42b2905d2a38c5d4ddc83d8513c3ad368e7c6ec6660b2ba00352b3ad7142e2f ./contracts/interfaces/IFeeDistributor.sol
5603cbc23d7bb05f4d3754c38a35b2807114ad7b3ff922466f5a48e828476c4c ./contracts/interfaces/ILiquidationManager.sol
2ec9ae5e2ea4cc5a9e75ad7596b939a5558b2fb6fbb55f6564ffc63adcfd0ac9 ./contracts/interfaces/IPriceFeed.sol
c2946f15dc72541b7ccee03ed1204dcf185a7d3e6d8bfd567da4c69b8da61995 ./contracts/interfaces/IRatesManager.sol
709a7e275e15f5c05c1d33e1809e3adac933d462e7699c82c62904eed81bec8f ./contracts/interfaces/ISTABLEX.sol
951c0ac432a3e4e69fb17340ec2bd1d1acc5c6b4da391e78617be848cfce0793 ./contracts/interfaces/IVaultsCore.sol
19a5b1c054fd7f9ccc624fc221634c072cfc575f2e9b84951eee298938707b96 ./contracts/interfaces/IVaultsDataProvider.sol
a2512a519b984c13b9c489eb9a6e88a9284c9c7b684316b11b855e808db3c69d ./contracts/fees/FeeDistributor.sol
5f33c718b8a1ed87404dbc07513efeb98f1bf1540c4343740e5b4bf03099d62e ./contracts/core/AddressProvider.sol
9870f9054a5ff1d3711a2bf6549831c2ac4dfac3e4a571872933cf0720ac186d ./contracts/core/ConfigProvider.sol
0f6e6458c5c63224fb71fb43b0c0c0c6fcfd68439cfc3cfd2566ae646645778c ./contracts/core/LiquidiationManager.sol
498687a192b2ec611305f9c15cfa3cc42403ea4d6ceb69e7986c11724b6c015c ./contracts/core/PriceFeed.sol
1c1a011b4cec7db833b958838ab4c1d7d8380640957dad90e7bdedbc21a2768f ./contracts/core/PriceFeedEUR.sol
81212dd6631ecd140e00b6a9d06486177d87dde9db33e502cfef3338638431e6 ./contracts/core/RatesManager.sol
53e4b34a3fc5c5c1ac0bbb9b1d46233ade47122df4c40299f832d813d27c908e ./contracts/core/VaultsCore.sol
02e4308513b66f4eb018558f8f326dafc31b3dc900e7b38e56c2d18a9930844e ./contracts/core/VaultsDataProvider.sol
575a1dbbac4bc877218ccf223ac66977ccbb24aa5f4dacf6a64ebb03849a95f8 ./contracts/chainlink/AggregatorV3Interface.sol
46595eb1a9b41259036fb514c76e38c4c75d6513319e55df2f830698e3183b58 ./contracts/access/AccessController.sol
```

Tests

```
86a69bdfe1b1526f9d396bac193203a2977d25251177b9d1d8a6824594e3ecba ./test/AccessControl.behaviour.ts
3fabd74f74cb0bdd574a99221a84499f28d6d23aac5b34d176867ee2639c30ac ./test/AddressProvider.test.ts
68d6795e6e17df6cf2166f166f99c05a05da69d8cf1d9768f11c78e0e7c0bd76 ./test/ConfigProvider.test.ts
b44291b9017f526078906a033f166d47085cede4c23a7c932f8f4c1790e5b8c4 ./test/FeeDistributor.test.ts
58a3f4c4e71c69c83170276de6f691ee0584195d147fa3f22ddb6b981d82ec72 ./test/LiquidationManager.ts
ef784f3057826a974111992902931e41d0794ef7eff7a062e4c22418a2a030a1 ./test/PriceFeed.test.ts
d076eadfc8c7324562f1d4f8dc62b1b80c4ad0e2af7a3eda833186af74204017 ./test/PriceFeedEUR.test.ts
399c6a3e2d3e7835cbac35d8cd2090a6896187c83ca1415abf50561d8e267961 ./test/RatesManager.test.ts
a2c5c76d3fdefd3e20497b78ba806a88255ded6cde7dd9ee62a766d5d2003091 ./test/STABLEX.test.ts
bf6a16e91e0b9ed5b79ea08b942961df65c451fca83d8dde3b2b9ee1500972f5 ./test/VaultsCore.config.test.ts
590569a759d65a323b778e00a6d3000644afe3e2aac3896d0b90e32a661fbe38 ./test/VaultsCore.debt-limits.test.ts
f29099d914201ca24dc305959dd6ed40f7dca90ae7e8f19a7cc71fd5c1bdc173 ./test/VaultsCore.debt.test.ts
85de6a6b7a92caec245fd79da1208c9e2c89e084a40abc9597eed75eaca7df4d ./test/VaultsCore.gas.test.ts
38f3b24448c1ff51a9b60aeaee8c1c097b0f586c593a2a296dff85a21990e6fc ./test/VaultsCore.income.test.ts
5e940f6cf2bb4c9a9749c8fe1b4e6180fe3e8504a1482b99cafd90cab89374e0 ./test/VaultsCore.liquidation.test.ts
e3924938058aa433963150d1d0f604c95a7e300174511b54a31efd769bf8012d ./test/VaultsCore.rates.multicollateral.test.ts
f4051f65f026a1d661e7f1a5b41e6cdf7ab5f5dfedffb91dcacd4a9ca442c8b8 ./test/VaultsCore.rates.test.ts
56c72c5a97d39fa85c5090610b7007d6ac37a214c10de8e243c21e7700a839c3 ./test/VaultsCore.vaults.eur.test.ts
6fa14ca63ba47f3f6d72bc49b3ee33ffbeab99a903a7e9912eecbf7db699c96a ./test/VaultsCore.vaults.test.ts
62f9449665503ab44bb925e16f4a1c8fc93f045d03625d576f87c5b57b18e253 ./test/utils/helpers.ts
45695e8c77b4cf6f3eea15d36b673b4a9b06a7436ad49a2b1008d232675a96fc ./test/utils/truffle-events.js
```

Changelog

- 2020-11-11 - Initial report
- 2020-11-23 - Revised report based on commit `680fa1a`

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.