Quantstamp Security Assessment Certificate

November 18th 2020 — Quantstamp Verified

PEAKDEFI

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

Executive Summary

Type	decentralized asset management fund
Auditors	Fayçal Lalidji, Security Auditor Kacper Bąk, Senior Research Engineer Shunsuke Tokoshima, Software Engineer
Timeline	2020-10-05 through 2020-10-19
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	User Documentation
Documentation Quality	Medium
Test Quality	Low

Source Code

Repository	Commit
PeakDeFi-token	70ba3df
peakdefi-contracts	097ec7d

Total Issues	**18** (7 Resolved)
High Risk Issues	**1** (1 Resolved)
Medium Risk Issues	**2** (2 Resolved)
Low Risk Issues	**9** (3 Resolved)
Informational Risk Issues	**6** (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

Through reviewing the code, we found **20 potential issues** of various levels of severity: one high, five medium, eight low, and six informational severity. We recommend addressing the findings before using them in production.

In addition, we provided ideas for further code and documentation improvements.

ID	Description	Severity	Status
QSP-1	`PeakDefiFund` Migration	High	Fixed
QSP-2	Peak Minting Cap	Medium	Fixed
QSP-3	Signature Execution	Medium	Fixed
QSP-4	Decimals Setter	Low	Acknowledged
QSP-5	Missing Input Validation	Low	Acknowledged
QSP-6	Unlocked Dependency Version	Low	Fixed
QSP-7	Gas Estimation	Low	Acknowledged
QSP-8	`PeakStaking` and Specification Discrepancy	Low	Fixed
QSP-9	Uniswap Oracle Stale Price	Low	Acknowledged
QSP-10	Manager Registration (contracts)	Low	Acknowledged
QSP-11	Off Chain Message Signature	Low	Acknowledged
QSP-12	PeakDefiFund initialization	Low	Fixed
QSP-13	Unlocked Pragma	Informational	Acknowledged
QSP-14	Allowance Double-Spend Exploit	Informational	Acknowledged
QSP-15	Clone-and-Own	Informational	Acknowledged
QSP-16	Uniswap Oracle Period	Informational	Acknowledged
QSP-17	Compound Token List Initialization	Informational	Fixed
QSP-18	Privileged Roles and Ownership	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 v0.6.13

Steps taken to run the tools:

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

Findings

QSP-1 `PeakDefiFund` Migration

Severity: *High Risk*

Status: Fixed

File(s) affected: `PeakDeFiFund.sol`

Description: When migrating the `PeakDeFiFund` to a new version, `migrateOwnedContractsToNextVersion` function does not transfer the ownership of `peakReferralToken` which will break token deposit and withdrawal, since the new contract won't own the `peakReferralToken`.

Recommendation: Call `peakReferralToken.transferOwnership()` inside `PeakDeFiFund.migrateOwnedContractsToNextVersion()`. `

QSP-2 Peak Minting Cap

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `PeakStaking.sol`

Description: Peak token cap is fixed to five billions, `_interestRateFactor` function is supposed to return zero rate after 5 billions cap. However, the function will throw instead since `SafeMath.sub` is used. As a consequence `PeakStaking` contract will be inaccessible after that the max cap is reached, disallowing new managers to join.

Recommendation: Add a condition that checks if `_mintedPeakTokens` rate is higher than the max cap and return zero for such case.

QSP-3 Signature Execution

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `PeakDefiLogic.sol`

Description: External signature won't work since `msg.sender` won't be equal to `address(this)`.
As an example, if user A calls `PeakDefiFund.createInvestmentWithSignature` that does a delegate call to `PeakdefiLogic.createInvestmentWithSignature` the `msg.sender` value will still be equal to user A address, once all ECDSA verifications pass with the signature of user B, `createInvestmentV2` is called without using the keyword `this.createIn...` meaning that the `msg.sender` value will still be equal to the initial sender (user A), The requirements that checks if `msg.sender == _sender || msg.sender == address(this)` will throw since both condition are wrong.
Affected function:

- `PeakDefiLogic.createInvestmentWithSignature()`.
- `PeakDefiLogic.sellInvestmentWithSignature()`.
- `PeakDefiLogic.createCompoundOrderWithSignature()`.
- `PeakDefiLogic.sellCompoundOrderWithSignature()`.
- `PeakDefiLogic.repayCompoundOrderWithSignature()`.

Recommendation: Set visibility of `createInvestmentV2`, `sellInvestmentAssetV2` and `repayCompoundOrder` to external and use `this` keyword when calling those functions from the functions listed in the description.

QSP-4 Decimals Setter

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `PEAKDEFI_V2.sol`

Description: `changeTokenData` must not allow to change the decimals value, many third party platform rely on the token decimals value to do sensitive calculation, changing the decimals value will lead to high severity issues.

Recommendation: Remove the decimals input from `changeTokenData` function.

QSP-5 Missing Input Validation

Severity: *Low Risk*

Status: Acknowledged

Description: Functions should always validate input parameters before using their values. One common mistake that could occur due to missing input validation is that funds are sent to the `0x0` address.
Parameters input in the functions listed below are not validated:

- `CompoundOrderFactory.constructor()`.

- `PeakReward.constructor()`.
- `PeakStaking.constructor()` .
- `PeakStaking.init()`.
- `PeakDeFiFund.initParams()`.
- `PeakDeFiFactory.constructor()`.
- `PeakDeFiV1.initialize()`.
- `PeakSwap.constructor()`.

Recommendation: Add input parameter validation which checks that input parameters of type `address` are different from zero, in all functions where this is needed.

QSP-6 Unlocked Dependency Version

Severity: *Low Risk*

Status: Fixed

File(s) affected: `peakdefi-contracts/eth/package.json`, `PeakDefi-Token/package.json`

Description: The version of `@openzeppelin/contracts`, `@uniswap/v2-periphery`, and `@uniswap/v2-core` are not locked in `peakdefi-contracts` and the version of `@openzeppelin/contracts` is not locked in `peakDeFi-token`. This entails some risk of unexpected behaviours due to updates of contracts in these packages.

Recommendation: Use the package manager lock mechanism to freeze the version for each dependency.

QSP-7 Gas Estimation

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `PeakDeFiLogic3.sol`

Description: Multiple functions in `PeakDeFiLogic3` loop through `cycleNumber`. Depending on the current cycle value the gas consumption can be excessively high with a low risk to reach block gas limit.

Recommendation: It is recommended to run thorough gas consumption simulation for such cases.

QSP-8 `PeakStaking` and Specification Discrepancy

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PeakStaking.sol`

Description: As per the specification `PeakStaking._interestRateFactor` does not seem to implement "Bigger Bonus Daily".

Recommendation: If it is an intended design the specification should be updated, otherwise `PeakStaking._interestRateFactor()` must be re-implemented correctly.

QSP-9 Uniswap Oracle Stale Price

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `UniswapOracle.sol`

Description: When getting the price average for a token using `UniswapOracle.consult` the price might be stale, meaning that if the oracle stops being updated, the contract calling `UniswapOracle.consult` won't be notified.

Recommendation: Consider adding an extra boolean return value that inform the caller contract if the price is stale.

QSP-10 Manager Registration (contracts)

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `PeakDeFiLogic2.sol`

Description: If a contract is used to register as a manager using `registerWithETH` or `registerWithToken`, the contract might end by receiving USDC back if the sent balance to register is higher than the maximum allowed. This can be an issue if the contract does not handle USDC correctly.

Recommendation: Send back the remaining fund following the initial used currency.

QSP-11 Off Chain Message Signature

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `PeakDefiFund.sol`

Description: When users interact with their installed wallet to sign an unstructured message they only see an unreadable bytestring, therefore no useful indication is displayed to allow avoiding phishing attempts or possible replay attacks (when interacting with different EVM capable blockchains).
Affected functions:

- `PeakDefiFund.createInvestmentWithSignature ()`
- `PeakDefiFund.sellInvestmentWithSignature()`
- `PeakDefiFund.createCompoundOrderWithSignature()`
- `PeakDefiFund.sellCompoundOrderWithSignature()`
- `PeakDefiFund.repayCompoundOrderWithSignature()`

Recommendation: We strongly recommend to use EIP-712 instead of the adopted unstructured signature , This will allow user more readability when signing using Metamask and other wallets. Another important advantage is if the `chainId` is set a mandatory field, official wallets will block signature request from networks that uses different `chainID`in the purpose of a reply attack.

QSP-12 PeakDefiFund initialization

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PeakdefiFactory.sol`

Description: `PeakdefiFactory createFund` and all initialisation functions do not implement any authorisation mechanism, depending on how often `PeakDefiFund`s are deployed, an attacker can front run a transaction to set initialisation parameters to its own.

Recommendation: If all functions are not guaranteed to be executed in a single transaction for gas consumption matters, a mechanism should be implemented to allow only the fund creator to execute the remaining initialisation functions.

QSP-13 Unlocked Pragma

Severity: *Informational*

Status: Acknowledged

File(s) affected: `PeakDefi-token/contracts/*`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.*.*`. 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-14 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Acknowledged

File(s) affected: `PEAKDEFI_V1.sol`, `PEAKDEFI_V2.sol`, `PeakToken.sol`

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

Exploit Scenario: An example of an exploit goes as follows:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance` and `decreaseAllowance`.
Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.

QSP-15 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CloneFactory.sol`, `MiniMeToken.sol`, `ERC20.sol`, `ReentrancyGuard.sol`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

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

QSP-16 Uniswap Oracle Period

Severity: *Informational*

Status: Acknowledged

File(s) affected: `UniswapOracle.sol`

Description: `UniswapOracle.update` is allowed to be called only once every 24 hrs. Using uniswap as oracle represents some risks since if the cumulative price window is too short the `UniswapOracle.consult` return value can be subject to manipulation. However, higher values of `period` can lead the prices to not be reflected in time and negatively impact on the system.

Recommendation: The impact of the selected `period` must be analyzed carefully.

QSP-17 Compound Token List Initialization

Severity: Informational

Status: Fixed

File(s) affected: `PeakDefifund.sol`

Description: `initTokenListings` does not check if the listed compound tokens are set in the compound order factory. Adding a token that is not listed may lead to unexpected behaviour.

Recommendation: add a the following requirement before setting `isCompoundToken` to true:

```
require(factory.tokenIsListed(_token));
```

QSP-18 Privileged Roles and Ownership

Severity: Informational

Status: Acknowledged

File(s) affected: `PeakDefiLogic2.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. In`PeakDefiLogic2.peakAdminRegisterManager` owner can add manager without any initial donation, Please note that the function is not accessible through `PeakDefiFund`. However, it can be deployed in future upgrades .

Recommendation: Quantstamp recommend to make centralization of power clear to the users, especially depending on the level of privilege the contract allows to the owner.

Automated Analyses

Slither

The analysis was completed successfully. No issues were detected.

Adherence to Specification

- In `ERC20._initialize()` L48-49: It reads “initializes {decimals} with a default value of 18.” However, its value is set to 8 in `PEAKDEFI_V1._initialize()` L12 . The code and the specification should be aligned.
- `PeakReward.canRankUp` contains a condition in L318 that is not stipulated in the specifications. We recommend adding the requirement to the specification.

Code Documentation

- In `Utils#L143` we recommend adding a comment explaining the origin of hard-coded address.
- In `Utils#L35` It is recommended to clarify that `ETH_TOKEN_ADDRESS` is the address that `KyberNetwork` uses to represent ether.
- It is recommended to paraphrase Career Value(CV) in `PeakReward` contract with `DSV` aligning with the user documentation.
- Unexpected indents are observed in `PeakSwap` over several lines (e.g. L7, 19, 31). It is recommended to use a linter (e.g. solhint).
- `PeakStaking._interestRateFactor` naming should be aligned with "EarlyFactor" documentation naming.

Adherence to Best Practices

- `ShortCEtherOrder` L8-9 can be simplified to `uint256 tokenPrice = __tokenToUSDC(CETH_ADDR, PRECISION);`
- In `ShortCEtherOrder#L66` , `LongCEtherOrder#L68`, `LongCERC20Order#L74`, `ShortCERC20Order#L71` and `PeakDeFiFund#L117,#L121` `add()` is not needed, use a normal addition operation instead.
- Use normal addition operation instead of `add()` in `PeakDeFiLogic3` L236, L358, L449 and L524.
- `PEAKDEFI_V1.mint()` and `PEAKDEFI_V1.burn()` are external functions whereas they are public in `IPeakDeFi`.
- Inconsistent use of `uint` and `uint256` in `Utils` contract.
- Most `require` statements do not return an error message.
- `LongCERC20Order` reimplement `__tokenToUSDC` function that is already inherited from `CompoundOrder`, please note that both functions are similar.

Test Results

Test Suite Results

```
###PeakDefi-Token:

Compiling your contracts...

> Everything is up to date, there is nothing to compile.



  Contract: PeakDefi Token
    Check initial values and init the token contract
       ✓ Checks the initial values of the contract (58ms)
       ✓ Init contact values (96ms)
       ✓ Checks the values after init (48ms)
    Mint tokens and changed allowance before upgrading
       ✓ Mint tokens (74ms)
       ✓ Approve tokens (68ms)
    Deploy and upgrade token contract to V2
       ✓ Deploy and upgrade V2 (107ms)
       ✓ Check the contract initial values after upgrade (51ms)
       ✓ Check the balance and allowances after upgrade
    Use the new method of V2 token
       ✓ Called the new method (100ms)


  9 passing (911ms)

### peakdefi-contracts

Compiling...

Compiled 58 contracts successfully


  Contract: simulation
     ✓ deposit_usdc (177ms)
     ✓ deposit_token (222ms)
     ✓ deposit_ether (119ms)
     ✓ withdraw_usdc (110ms)
     ✓ withdraw_token (148ms)
     ✓ withdraw_ether (128ms)
     ✓ register_accounts (592ms)
     ✓ phase_0_to_1
     ✓ can't_burn_deadman (46ms)
     ✓ create_investment (239ms)
     ✓ sell_investment (316ms)
     ✓ create_compound_orders (478ms)
     ✓ sell_compound_orders (490ms)
     ✓ next_cycle (82ms)
     ✓ redeem_commission (118ms)
     ✓ redeem_commission_in_shares (186ms)
     ✓ burn_deadmen (300ms)

  Contract: price_changes
     ✓ prep_work (225ms)
     ✓ raise_asset_price (1162ms)
     ✓ lower_asset_price (1176ms)
     ✓ lower_asset_price_to_0 (258ms)

  Contract: param_setters
     ✓ prep_work
     ✓ decrease_only_proportion_setters (54ms)
     ✓ address_setters

  Contract: peak_staking
     ✓ stake() (150ms)
     ✓ withdraw() (71ms)

  Contract: peak_reward
     ✓ prep_work
     ✓ refer()
     ✓ canRefer()
     ✓ payCommission() (285ms)
     ✓ rankUp() (167ms)


  31 passing (14s)
```

Code Coverage

PeakDefi-Token:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	44.83	18.18	58.33	44.83	
ERC20.sol	47.5	25	58.82	47.5	… 249,250,251
IPeakDeFi.sol	100	100	100	100	
PEAKDEFI_V1.sol	85.71	50	66.67	85.71	27
PEAKDEFI_V2.sol	100	100	100	100	
PeakSwap.sol	0	0	0	0	… 39,40,43,45
TokenProxy.sol	100	100	100	100	
All files	**44.83**	**18.18**	**58.33**	**44.83**	

peakdefi-contracts

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	66.03	42.51	64.67	66.13	
PeakDeFiFactory.sol	100	100	100	100	
PeakDeFiFund.sol	47.62	33.02	45.76	47.37	… 5,1086,1098
PeakDeFiLogic.sol	72.44	43.88	76	72.44	… 691,775,806
PeakDeFiLogic2.sol	78.1	43.59	85.71	78.42	… 572,573,576
PeakDeFiLogic3.sol	67.18	48.61	55.56	66.67	… 551,554,558
PeakDeFiProxy.sol	28.57	0	50	28.57	16,17,19,20,21
PeakDeFiProxyInterface.sol	100	100	100	100	
PeakDeFiStorage.sol	66.67	60	80	66.67	… 414,423,438
Utils.sol	64.29	47.22	88.89	66.67	… 193,194,195
contracts/derivatives/	**68.93**	**42.7**	**77.08**	**69.05**	
CompoundOrder.sol	66.15	46.15	81.82	66.15	… 145,147,160

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
CompoundOrderFactory.sol	87.1	75	80	87.1	71,91,92,93
LongCERC20Order.sol	87.34	52.5	100	87.65	… 104,124,139
LongCEtherOrder.sol	85.92	50	100	86.3	… 4,95,98,130
ShortCERC20Order.sol	87.67	52.63	100	88	… ,97,100,128
ShortCEtherOrder.sol	0	0	0	0	… 143,147,148
contracts/interfaces/	100	100	100	100	
CERC20.sol	100	100	100	100	
CEther.sol	100	100	100	100	
Comptroller.sol	100	100	100	100	
IMiniMeToken.sol	100	100	100	100	
KyberNetwork.sol	100	100	100	100	
OneInchExchange.sol	100	100	100	100	
PriceOracle.sol	100	100	100	100	
contracts/lib/	83.33	50	75	77.78	
CloneFactory.sol	50	100	50	50	44,45
ReentrancyGuard.sol	100	50	100	100	
contracts/peak/	100	100	0	100	
IUniswapOracle.sol	100	100	100	100	
PeakToken.sol	100	100	0	100	
contracts/peak/reward/	86.26	62.5	91.67	88.98	
PeakReward.sol	86.26	62.5	91.67	88.98	… 314,315,316
contracts/peak/staking/	92.59	60	100	92.59	
PeakStaking.sol	92.59	60	100	92.59	186,193,197,214
contracts/tokens/minime/	40.52	20	45.45	41.59	
MiniMeToken.sol	40.52	20	45.45	41.59	… 503,504,505
TokenController.sol	100	100	100	100	
All files	**67.67**	**42.43**	**67.62**	**67.95**	

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.

Tests

```
8b6120f154b7e03e424fd2075f22dd3977fd12cd28a993be9abf7be4f22a09fb ./PeakToken.test.js
53fac6a89b4d17846709f94b216562e6b152d1e113e06e9afe44cbdac36b5102 ./ERC20.sol
09f16dc82725dbefe5b5beacd77eba55d0ab37051eb7a57dd7e98a78e2687c77 ./PEAKDEFI_V2.sol
f48627d56bdd1d9a8bd64fc963279ecd1f0912c1a18ccae22b35be7e9bec9a50 ./TokenProxy.sol
373bcde0f7485697401fa229703c5d7aa3f9f918e2552547fc23033a5c9288eb ./PEAKDEFI_V1.sol
a9b56ae59c287031ab773b0baaa1c2cff8cf43def1540aa4c5f4869f96fe0377 ./IPeakDeFi.sol
8a6b38936c738a0e612391ee231f39352cc8878f4a5b41c05f0895fb662b3fd6 ./Migrations.sol
c0e1d9382897935b060bcef092bb0d9e1bd0524274aa2f688fabf242482412dd ./PeakSwap.sol
9555bb5ca5432eca2393744c4f3f7e8e644ba1a1659b6575bb0600b33848c28c ./PeakDeFiLogic3.sol
b053f5e77a326b759fd88236dd169198f3d84aa47a2b898ad91fe9d05ad46095 ./PeakDeFiFund.sol
e52db9629e4210f7cbbe0d2d43e0f3c554b5633bfb8ac479ed23987e78582055 ./PeakDeFiProxyInterface.sol
242880f9a5a5d4ac5e9ee330d15a7efa89b59e0a61f46d24f359f55ee4acee08 ./PeakDeFiFactory.sol
009817c775f239836dab82d3b25c1457bab78278d8adc12a2f0b9b325f687472 ./PeakDeFiLogic2.sol
6e60cae897f72a053fdeb8d74b5e639b7bac4abd6b817303d7b9e85b7fbbabdb ./PeakDeFiLogic.sol
6759223d4df79b79761c4c63dd12b7d06c8abf21a33eb6de77af661e3fc45be2 ./Utils.sol
ab0b1d6382d40a35d463f3ca50713d30c37497df1ee52fc90ba859b22d0adaa5 ./PeakDeFiProxy.sol
f4d16a5ed3901fd648b4ff30d4dae238b002a5da67307c394a59c7a172d5f2a7 ./PeakDeFiStorage.sol
0df2248ddfdb1f3e20a430684a8b69ca8fd991dbfcb89f562d276d4e48240de2 ./CloneFactory.sol
f12b927f372895fc929f05a6987f6333ce2f69ba338708b22659906f665f026a ./ReentrancyGuard.sol
f9005510b0ac058ecdbadea4a03a4521614af2e653819f338c56a03af32100a9 ./LongCEtherOrder.sol
abd1a23a74f201638cf74556b1f387b7ee7a0ec90675e272064c63c3d4b25f62 ./LongCERC20Order.sol
253e2af079102a42f107830e15c32e4feccf263315cc1d963d84ff08c82090aa ./ShortCEtherOrder.sol
26594ff84a94997ae0c70a041587b8b8f0b0e1c30d74aa74bfa0f72514cddd42 ./CompoundOrder.sol
fd07c73f6635064e77ca215b0763239903fd49f0b2ecf5af9270f11a77b1a94d ./ShortCERC20Order.sol
a97a3791e73b41441453b7c15bc01b7cbceb45ca7aec9290349c167677174452 ./CompoundOrderFactory.sol
60a26938757ddaf73334321d9a0d743c5ed74c2fb5b0321ebd89290ba80dc2b4 ./TokenController.sol
553ea4788cfe5515a6a34a394c353c90cb4db0254debdb1ea6cf8557367a394d ./MiniMeToken.sol
3a11d8824606fb147181a55e2103bd782dcc0956ff1bbe6a5638cb9b7498a411 ./IUniswapOracle.sol
bd9dbe32d76b249b518dd6c75161080320dbc40ddbb00f0529e99bec0db9d36d ./PeakToken.sol
fdcf7d8c7ceb21005fc1595f3ae4d85987390a29aac1e989f5cf33e48652a02a ./PeakReward.sol
b13032651d8623656bbefb08ad1a641adc156b612727c4aa8f3cb13ed38f7d7f ./PeakStaking.sol
8d61db5420e0f670b0b4300de150b9e28e432b256f6a74a34588ed80eece53f4 ./OneInchExchange.sol
f579222ca383c83ef9ba875c188a9e9639c002dc1b8cd737b0acdeeadc16e120 ./KyberNetwork.sol
09989107490f3bdba2dfebd1d9904cc196c673e40d37313595372f78dc4430d8 ./CEther.sol
0d4426ee24543e36a401dc33b0095c1494b4b1585c02ec1325b4339fc7a8826b ./PriceOracle.sol
03ae67b49de947720503bd7c1a03fab0c3a6f66bf58ffb98c0d3d83ea7199b55 ./IMiniMeToken.sol
386d74cd13d6efef4cd2a3e77c3d2702204559aed4500dbd5d348e6441ec28e4 ./Comptroller.sol
d2cff8c96551d2d585fb87efbbab8f65063852fb162ed2056be393a9b5f8c4de ./CERC20.sol
d7d80471e0624f184ee4f6fefcb17d391b0761ee85878cd3b55d875385a8c069 ./UniswapOracle.sol
ea20e9283be59b5c598fdfd30bea2a29a96e24d55de108b3bd501819cb97c7ee ./test/truffle-fixture.js
6e6316d5bd53f76e6cba631119cfb531e471de131392e17f9278b1136c4f57f4 ./test/Test.js
```

Changelog

- 2020-10-19 - Initial report
- 2020-10-20 - Report update
- 2020-10-21 - Tests and coverage update
- 2020-10-30 - Updating final fix

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.