Quantstamp Security Assessment Certificate

April 21st 2021 — Quantstamp Verified

Cover Protocol

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

Executive Summary

Type	P2P Coverage
Auditors	Kacper Bąk, Senior Research Engineer Poming Lee, Research Engineer Sebastian Banescu, Senior Research Engineer
Timeline	2021-03-09 through 2021-04-19
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Computer-Aided Verification, Manual Review
Specification	Cover Protocol Documentation
Documentation Quality	High
Test Quality	Medium

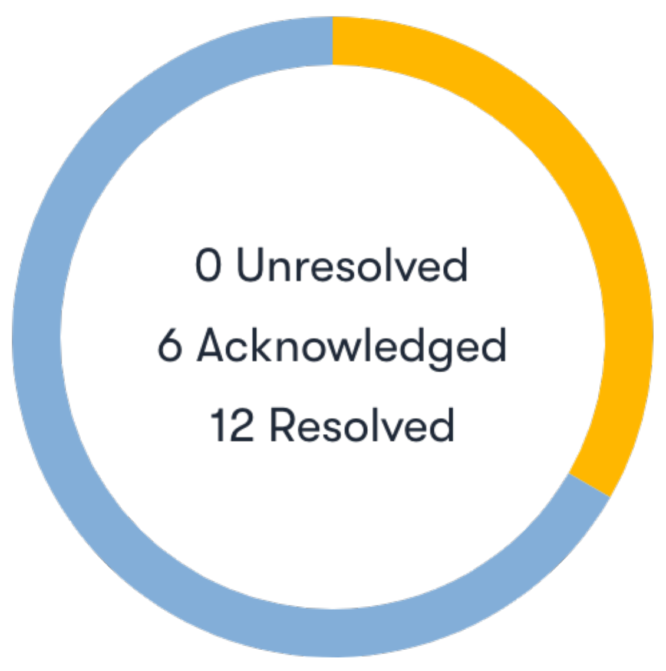
Source Code

Repository	Commit
cover-core-v2	ace9ba3

Goals

- Is the implementation vulnerable to DoS attacks?
- Does the implementation deviate from the specification?
- Are calculations implemented correctly?

Total Issues	18	(12 Resolved)
High Risk Issues	0	(0 Resolved)
Medium Risk Issues	0	(0 Resolved)
Low Risk Issues	8	(6 Resolved)
Informational Risk Issues	9	(5 Resolved)
Undetermined Risk Issues	1	(1 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

We have found a number of issues in the reviewed protocol. Although most of the issues are of low and informational severity, we recommending addressing each and every one. Overall, the system is quite complex, and, therefore, we highly recommend performing functional tests with multiple users, pools, and tokens. Perhaps, complex simulations could provide further confidence in the protocol.
Update: the team addressed all the issues as of commit `abc34e6`.

ID	Description	Severity	Status
QSP-1	Gas usage due to O(n^2) complexity in `addCVCForPools()`	Low	Acknowledged
QSP-2	`_removeCVCForPool()` and `deleteRisk()` would not delete an element that doesn't exist	Low	Fixed
QSP-3	Privileged Roles and Ownership	Low	Mitigated
QSP-4	Missing checks if address is non-zero	Low	Fixed
QSP-5	Unconstrained redeem delay	Low	Fixed
QSP-6	Gas concerns / Incorrect cover deployment	Low	Fixed
QSP-7	Off-by-one error	Low	Fixed
QSP-8	Loose input validation	Low	Acknowledged
QSP-9	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-10	Unlocked Pragma	Informational	Mitigated
QSP-11	Unnecessary nonce checks	Informational	Acknowledged
QSP-12	Rebasing/deflationary tokens	Informational	Acknowledged
QSP-13	Unclear return when adding risks	Informational	Fixed
QSP-14	Actual expiry timestamp may differ from expiry string	Informational	Acknowledged
QSP-15	Inconsistency between documented and implementation of token names	Informational	Fixed
QSP-16	3-day period might be too short	Informational	Acknowledged
QSP-17	Incentives	Informational	Mitigated
QSP-18	Unclear factory pausing mechanism	? Undetermined	Fixed

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 v0.7.0

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 Gas usage due to O(n^2) complexity in `addCVCForPools()`

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `ClaimConfig.sol`

Description: The function `_addCVCForPool()` is called whenever a CVC group is to be added. The function checks if the group hasn't been added yet. The check requires two for-loops, one nested within the other, which results in quadratic complexity. Depending on the number of groups, the check may be gas-intensive.

Recommendation: We recommend using linked lists (e.g., see LinkedListLib) to reduce the complexity to O(n).

Update: The team informed us that the square complexity should not be an issue in the short-term.

QSP-2 `_removeCVCForPool()` and `deleteRisk()` would not delete an element that doesn't exist

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ClaimConfig.sol`, `CoverPool.sol`

Description: Although the functions are callable by the owner, they assume that the given argument will be deleted and so they allocate shorter arrays. Consequently, a transaction would fail if one attempts to delete an element that doesn't exist.

Recommendation: We recommend fixing the code so that it doesn't fail. Furthermore, instead of copying elements, it may be cheaper to move the last element in place of the one that gets deleted.

Update: The issue has been resolved in commit 09c51df.

QSP-3 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Mitigated

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.
There are some actions that could have important consequences for end-users:

1. The owner of the `CoverPoolFactory` can call `CoverPool.deleteRisk()` at any point in time (even after a hack corresponding to that risk), which would effectively block claims (i.e. having covers paid out).
2. The owner of the `CoverPoolFactory` can set any token as collateral.

There are several roles having different privileges in the system, but which are not clearly explained in the specification. Here are some examples of such roles:

- `claimManager` defined in the `CoverPoolFactory` contract is the only user allowed to call `CoverPool.enactClaim()` and is also allowed to call `CoverPool.setNoclaimRedeemDelay()`.
- `responder` defined in the `CoverPoolFactory` contract is the only user in addition to the owner who is allowed to pause the factory and block some of the functions inside the `CoverPool` and `Cover` contracts.
- `treasury` which is indicated to "receive fees collected". However, this address is allowed to be any address including an EOA, which increases the risk of losing funds.

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.
Furthermore:

1. Calling `CoverPool.deleteRisk()` should revert if there is an active claim for that risk and should only be permitted again once the claim is resolved.
2. The owner should ensure that rebasing and/or deflationary tokens are never used as collateral. A list of such tokens should be added using a script and set to status `Disabled`.
3. All roles should be documented together with their purpose on a dedicated page.
4. The `treasury` address should be a multi-sig wallet with key deletion/recovery features.

QSP-4 Missing checks if address is non-zero

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CoverPoolFactory.sol`, `CoverPool.sol`

Description: Functions `createCoverPool()` and `initialize()` don't check if `_collateral` is non-zero.

Recommendation: Add relevant checks.

QSP-5 Unconstrained redeem delay

Severity: Low Risk

Status: Fixed

File(s) affected: `CoverPool.sol`, `CoverPoolFactory.sol`

Description: Line 90 in `CoverPool.sol` indicates that: “Claim manager can set it 10 days when claim filed”, which refers to the value of `noclaimRedeemDelay`. It is not clear if this means 10 days before or after the first claim is filed. However, there is no upper bound of the actual value which can be assigned to this `noclaimRedeemDelay` state variable. The only enforcement is that it should be higher-or-equal to the `factory.defaultRedeemDelay()`, which itself can be set to any value by the owner of the factory.

Recommendation: We recommend adding an upper bound.

QSP-6 Gas concerns / Incorrect cover deployment

Severity: Low Risk

Status: Fixed

File(s) affected: `CoverPool.sol`

Description: The `CoverPool.deployCover()` function comment indicates that this function: “Will only deploy or complete existing deployment if necessary.” However, the implementation deviates from this specification because after (trying to) deploy a cover it will also check if it is complete and attempt to complete the existing deployment if not, i.e.:

```
318:    if (!ICover(addr).deployComplete()) {
319:      ICover(addr).deploy();
320:    }
```

If this `if`-statement is ever entered it will probably lead to an out-of-gas error or to simply not deploying any additional covTokens for any additional risks in the `riskList`, because the `ICover(addr).deploy()` function was already called in the `ICover(addr).initialize()` function in the previous `if`-statement.

Recommendation: We recommend changing the `if`-statement on L318 to an `else if`-statement connected to the previous `if`-statement.

QSP-7 Off-by-one error

Severity: Low Risk

Status: Fixed

File(s) affected: `Cover.sol`

Description: The upper bound of the `for`-loop inside the `Cover._convert()` function is off-by-one. This bound should be `i < futureCovTokensCopy.length - 1` and not `i < futureCovTokensCopy.length`, because if the value of `i` equals `futureCovTokensCopy.length - 1` (during the last iteration of this loop), then the `futureCovTokensCopy` array will be indexed out of bounds on L317: `ICoverERC20 futureCovToken = futureCovTokensCopy[i + 1];`. This would cause the transaction to revert with an ambiguous error message.

Recommendation: Fix the upper bound of the loop to `uint256 upperBound = futureCovTokensCopy.length - 1`.

QSP-8 Loose input validation

Severity: Low Risk

Status: Acknowledged

File(s) affected: `CoverPoolFactory.sol`

Description: The `CoverPoolFactory.constructor()` checks if the addresses of `_coverPoolImpl`, `_coverImpl` and `_coverERC20Impl` are contracts. The same applies to the 3 setter functions: `setCoverPoolImpl()`, `setCoverImpl()` and `setCoverERC20Impl()`. However, this check uses `extcodehash` which is considered unsafe in some circumstances. Moreover, this check does not actually verify if those contracts comply with the expected interface. Therefore, even if the order of the contracts would be given incorrectly in the `constructor()` the checks would not fail.

Recommendation: Use EIP-165 to check if the aforementioned contracts comply with their respective interfaces, namely: `ICoverPool`, `ICover`, and `ICoverERC20`.

QSP-9 Allowance Double-Spend Exploit

Severity: Informational

Status: Mitigated

File(s) affected: `ERC20/ERC20.sol`, `ERC20/ERC20Permit.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.
A similar problem occurs in `ERC20/ERC20Permit.sol` in functions `permit()` and `transferFrom()`

Exploit Scenario:

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-10 Unlocked Pragma

Severity: *Informational*

Status: Mitigated

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. 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.

Update: The issue is fixed since the compiler version is locked using hardhat. It is important to note that the version 0.8 of Solidity is vulnerable to Keccak caching bug and ABIDecodeTwoDimensionalArrayMemory bug. The `keccak256` function is used in assembly code inside `Clones.sol`, which is called by the `CoverPoolFactory`. Also `abi.decode()` is used with a param from `memory` on L70 in `SafeERC20.sol`.

QSP-11 Unnecessary nonce checks

Severity: *Informational*

Status: Acknowledged

File(s) affected: `ClaimManagement.sol`, `CoverPool.sol`

Description: In `ClaimManagement.sol` in lines 89 and L120, `_nonce` must equal `_getCoverPoolNonce(_coverPool)`. The check is not strictly necessary and `_nonce` could be removed as an argument.
Similar remark applies to checks for `_coverPoolNonce` in `CoverPool.enactClaim()`.

Recommendation: We recommend removing the argument and the check.

QSP-12 Rebasing/deflationary tokens

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CoverPool.sol`

Description: The protocol supports tokens with fixed decimals and inflationary tokens. It is incompatible, however, with rebasing and deflationary tokens if the rebasing operation is indirectly invoked during the transfer.

Recommendation: We recommend informing users and not using such tokens in the protocol.

QSP-13 Unclear return when adding risks

Severity: *Informational*

Status: Fixed

File(s) affected: `CoverPool.sol`

Description: When the `CoverPool.addRisk()` function returns it is not clear to the caller if this function has successfully finished adding the risk to all active covers or if it stopped to avoid an out of gas error. This requires the caller to remember to check the value of the `addingRiskWIP` state variable to clarify this question. This leaves some room for human error in case the caller forgets to do this check.

Recommendation: To reduce the potential for human error we recommend adding a Boolean return value to the `addRisk()` function, which returns `false` when the risk could not be added to all active covers due to insufficient gas, and `true` otherwise.

QSP-14 Actual expiry timestamp may differ from expiry string

Severity: *Informational*

Status: Acknowledged

File(s) affected: `CoverPool.sol`

Description: There does not seem to be any check between the value of `_expiry` and that of `_expiryString` in the various functions inside the `CoverPool` contract which receive this value from a user, e.g.:

1. The `initialize()` function.
2. The `setExpiry()` function.

However, these two values are always assumed to be in sync. If the values of `_expiry` and that of `_expiryString` were not in sync it could mean that the date indicated in the token name would be earlier or later than the actual expiry date, which would crate confusion and could lead to reputation damage.

Recommendation: Checking whether the values are in sync inside of smart contracts is a costly option. It may be useful to do it occasionally, however. A less expensive option to mitigate this issue is to compare the two values in the user interface which is typically used to enter these values. It is important to note, however, that it is not enforceable that only user interfaces that check these values are used.

QSP-15 Inconsistency between documented and implementation of token names

Severity: *Informational*

Status: Fixed

File(s) affected: `CoverPool.sol`

Description: The documentation indicates that token names will be of the following form:

> The format for ticker symbols for the covTokens is: `{ Direction, C | NC }_{ Risk Name | Future }_{ Cover Pool }_{ Nonce }_{ Collateral }_{ Expiry }`. Example of covTokens for a Yearn Cover Pool with two risks (3Crv and yCrv). In this example, 4 covTokens will be created. One ncToken, one future cToken, and 2 cTokens (one for each Risk).
>
> - C_FUTO_Yearn_0_DAI_12_31_20
> - C_3Crv_Yearn_0_DAI_12_31_20
> - C_yCRV_Yearn_0_DAI_12_31_20
> - NC_Yearn_0_DAI_12_31_20

However, in the implementation we have noticed the expiry string appears in 2 different formats:

1. The `MONTH_DATE_YEAR` format as indicated in the NatSpec comment of the `CoverPoolFactory.createCoverPool()` function. This is partially aligned with the documentation, however, the problem with this comment is that it does not give a clear indication how many characters should be used for each of the 3 date items. Use `MM_DD_YY` to be more specific.
2. The `YYMMDD` format as indicated in the comment of the `CoverPool._getCoverName()` function, that is L332: `3POOL_0_DAI_210131`.

Recommendation: Align the date formats of the documentation and the comments in the implementation.

QSP-16 3-day period might be too short

Severity: Informational

Status: Acknowledged

Description: The Claim Guidelines indicate that:

- There will be a default 3 day grace period after expiry to make a claim on an incident that occurred before the expiry of a specific coverage.
- When no claim is accepted there will be a default 10 day delay for redemption of NOCLAIM tokens.
- When a claim is accepted there will be a default 3 day delay for CLAIM tokens (and NOCLAIM tokens if partial payout).
- Up to 3 days of voting for the community on snapshot.page and up to 3 days of voting for the CVC - multisig.
- [...] and the coverage purchaser submits a claim within 3 days of the incident.

Notice that a 3 day period is given for several steps including submitting a claim and investigating a claim. It is not clear why this strict 3 day period is required, but we do envision situations when such a short period could be problematic, e.g. consider that the coverage purchaser goes on a trip (e.g. Burning Man) for a week.

Recommendation: Extend the 3-day period for filing claims to at least 10 days or some period which would permit more flexibility.

Update: 3 days restrict is on filing a claim. As long as there is one person (the team always do) files a claim within the 3 days after expiry (rare occasions), everyone on vacation will also be able to redeem if the claim were to be accepted. Other 3 day restrictions are just waiting time, does not really affect redeeming eligibility.

QSP-17 Incentives

Severity: Informational

Status: Mitigated

File(s) affected: `ClaimManagement.sol`

Description: It may happen that not all members in the CVC agree on the outcome of a claim done in `ClaimManagement.decideClaim()`. Any one member could submit a transaction, or even front-run another member, to decide a claim in any way they want. This would be unlikely to happen if the incentives offered by Cover are higher than the incentives that the CVC member would get to decide a certain way. If there's a 50 MM USD hack and someone who was affected can bribe any CVC member to decide if the claims should be accepted, then the bribe could be quite substantial. It is unclear if Cover provides enough incentives to CVC members to withstand the bribe threat.

Recommendation: One possible fix would be to require a majority vote from all CVC members in order to decide a claim. The voting process would have to be time-bounded over a certain period when votes are accepted.

Update: CVCs are in a multi-sig and requires a majority of them agree on the validity and the payout percentages.

QSP-18 Unclear factory pausing mechanism

Severity: Undetermined

Status: Fixed

File(s) affected: `CoverPoolFactory.sol`

Description: The `CoverPoolFactory.setPaused()` function allows users to set the paused state variable to `true` or `false`. However, this feature is used in a different way than the common `Pausable` contract developed by OpenZeppelin. Instead, the `paused` state variable is used by the `CoverPool` and `Cover` contracts to block the following function calls when the factory paused variable is set to `true`:

1. `CoverPool.addCover()`
2. `Cover.redeem()`
3. `Cover.convert()`
4. `Cover.redeemClaim()`.

However, it is unclear from the specification if only the four functions mentioned above should be blocked when the factory is paused or other functions as well. There are several other functions that use the factory contract inside the `CoverPool` and `Cover` contracts, namely:

- `CoverPool.addRisk()`
- `CoverPool.setNoclaimRedeemDelay()`

- `CoverPool.enactClaim()`
- `CoverPool.deployCover()`
- `Cover.deploy()`
- `Cover.collectFees()`
- `Cover._createCovToken()` which is a private function called by `Cover.addRisk()` and `Cover.deploy()`.

Recommendation: Clarify which functions should be blocked when `paused` is set to `true`.

Automated Analyses

Slither

Slither does not support Solidity 0.8.x yet.

Code Documentation

1. There are several typos and grammar mistakes in the code comments.
2. Some functions are properly commented with NatSpec comments describing purpose, input params and return values, while other functions only have a brief description of the function. We recommend using NatSpec and describing purpose, input params and return values for all functions.

Adherence to Best Practices

1. In `utils/StringHelper.sol`, there is a TODO item. We recommend resolving it. **Update:** fixed.
2. Magic numbers should be replace with constants having descriptive names, e.g. 1e18 is used several times in the `Cover` contract. **Update:** fixed.
3. Events should have indexed `address` parameters, e.g.:
    - `FutureTokenConverted` on L15 in `ICover.sol`. **Update:** fixed.
    - `CoverCreated` on L10 in `ICoverPool.sol`. **Update:** fixed.
    - `CollateralUpdated` on L17 in `ICoverPool.sol`. **Update:** fixed.
    - `CoverPoolCreated` on L10 in `ICoverPoolFactory.sol`. **Update:** fixed.
    - `AddressUpdated` on L12 in `ICoverPoolFactory.sol`. **Update:** fixed.
4. Commented code should be removed, e.g., L14 in `ICoverPoolFactory.sol` and in `utils/StringHelper.sol`. **Update:** fixed.
5. `CoverPool.sol`: function `_getCoverName()`: consider checking if `_collateralSymbol` is not empty. **Update:** fixed.

Test Results

Test Suite Results

Test cases appear to be relatively simple. We strongly recommend adding complex functional tests. Furthermore, add a test which due to large gas amounts performs multiple calls to `CoverPool.deployCover()` in order to successfully complete the deployment of a cover. We also recommend adding more tests that deal with non-standard decimals.

```
CoverPoolFactory
   ✓ Should deploy with correct state variable values (96ms)
   ✓ Should emit CoverPoolCreation event (254ms)
   ✓ Should update vars by authorized only (381ms)
   ✓ Should add 2 new coverPools by owner (370ms)
   ✓ Should compute the same coverPool addresses (163ms)
   ✓ Should NOT add new coverPool by userA

CoverERC20
   ✓ Should deploy with correct name, symbol, decimals, totalSupply, balanceOf, permit nonce and domainSeparator (49ms)
   ✓ Should mint tokens to userA and userB
   ✓ Should update totalSupply after mint
   ✓ Should NOT mint tokens signed by non-owner
   ✓ Should allow transfer from userA to userB (44ms)
   ✓ Should approve
   ✓ Should transferFrom within allowance (38ms)
   ✓ Should NOT transferFrom when amount > allowance
   ✓ Should NOT burn by non-owner
   ✓ Should burn userB balance by owner
  permit
     ✓ accepts walletAddress signature
     ✓ rejects reused signature
     ✓ rejects other signature
     ✓ rejects expired permit

CoverPool
   ✓ Should initialize correct state variables (50ms)
   ✓ Should update state variables by the correct authority (104ms)
   ✓ Should NOT update state variables by the wrong authority
   ✓ Should add and delete risk for open pool (297ms)
   ✓ Should ONLY delete, NOT add risk for close pool (225ms)
   ✓ Should delete risk from pool correctly (55ms)
   ✓ Should add cover for userA and emit event (45ms)
   ✓ Should match cover with computed cover address (75ms)
   ✓ Should add cover for userB on existing contract (259ms)
   ✓ Should create new cover for userB on existing contract when accepted claim (253ms)
   ✓ Should emit event and enactClaim if called by claimManager (58ms)
   ✓ Should NOT enactClaim if coverPool nonce not match (83ms)
   ✓ Should NOT enactClaim if have non active risk
   ✓ Should NOT enactClaim if called by non-claimManager
   ✓ Should NOT add cover for userA for expired timestamp

Cover
   ✓ Should initialize correct state variables (154ms)
   ✓ Should addCover flashMint by contract (152ms)
   ✓ Should deploy Cover in two txs with CoverPool (259ms)
   ✓ Should match computed covToken addresses (44ms)
```

```
    ✓ Should add risk, convert, mint, and redeem with new active tokens only (516ms)
    ✓ Should delete risk, mint, and redeem with active tokens only (236ms)
    ✓ Should mint, and redeem correctly for non 1 deposit ratio (341ms)
    ✓ Should redeem collateral without accepted claim (95ms)
    ✓ Should redeem collateral with accepted claim with all tokens (129ms)
    ✓ Should redeem collateral after cover expired with all tokens (120ms)
    ✓ Should NOT redeem if dont have all tokens (51ms)
    ✓ Should redeem after expire and after wait period ends (73ms)
    ✓ Should NOT redeem after expire if does not hold noclaim covToken (49ms)
    ✓ Should NOT redeemClaim before accepted claim
    ✓ Should NOT redeemClaim after enact claim before redeemDelay ends (79ms)
    ✓ Should allow redeem partial claim and noclaim after enact 40% claim after defaultRedeemDelay ends (296ms)
    ✓ Should allow redeem ONLY after enact and noclaimRedeemDelay if incident after expiry (184ms)
    ✓ Should NOT redeemClaim after enact if does not have claim token

  ClaimConfig
    ✓ Should deploy ClaimManagement correctly (105ms)
    ✓ Should only set if authorized (101ms)
    ✓ Should set fees and currency (50ms)

  ClaimManagement
    ✓ Should deploy ClaimManagement correctly
    ✓ Should set vars correctly
    ✓ Should add cvc to deployed cover pool
    ✓ Should return false when no claims exist
    ✓ Should file a claim for incident correctly (83ms)
    ✓ Should return true for pending claim on coverPool
    ✓ Should return false for pending claim on non-existent pool
    ✓ Should cost 80 dai to file next claim (56ms)
    ✓ Should file a forced claim (70ms)
    ✓ Should NOT validate if condition is wrong (79ms)
    ✓ Should invalidate claim once (56ms)
    ✓ Should validate claim (39ms)
    ✓ Should file and validate more claims for further testing (97ms)
    ✓ Should NOT decideClaim if condition is wrong (118ms)
    ✓ Should accept claim (70ms)
    ✓ Should file new claims under nonce = 1 (39ms)
    ✓ Should NOT decideClaim throw if payoutNumerator != 0 when denying claim
    ✓ Should deny claim (57ms)
    ✓ Should return false for pending claim
    ✓ Should file 2 new claims (65ms)
    ✓ Should revert if try to validate claim with payoutNumerator > 0 after window passed
    ✓ Should deny claim if window passed (125ms)


  78 passing (26s)
```

Code Coverage

We recommend improving branch coverage of the main contracts.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	95.7	63.49	95.51	97.12	
ClaimConfig.sol	84.44	50	78.57	86.36	… 4,45,80,102
ClaimManagement.sol	98.8	75	92.31	98.8	174
Cover.sol	93.9	70	100	96.23	… 212,213,249
CoverERC20.sol	100	100	100	100	
CoverPool.sol	97.64	61.9	100	99.22	311
CoverPoolFactory.sol	100	52.94	100	100	
contracts/ERC20/	88.24	50	84.62	88.24	
ERC20.sol	88.24	50	84.62	88.24	82,83,87,88
contracts/interfaces/	100	100	100	100	
IClaimConfig.sol	100	100	100	100	
IClaimManagement.sol	100	100	100	100	
ICovTokenProxy.sol	100	100	100	100	
ICover.sol	100	100	100	100	
ICoverERC20.sol	100	100	100	100	
ICoverPool.sol	100	100	100	100	
ICoverPoolCallee.sol	100	100	100	100	
ICoverPoolFactory.sol	100	100	100	100	
IOwnable.sol	100	100	100	100	
contracts/testing/	100	50	100	100	
FlashCover.sol	100	50	100	100	
All files	**95.27**	**62.59**	**94.23**	**96.58**	

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

```
c511d15954c7c747135ce534c7543be4538c9f554270eee4573aa9a0dbb4dd34 ./contracts/CoverPool.sol

a071ebf1202f16496ad9a8827bdac3e5d658b35be3354dbaf13b6baf39054db9 ./contracts/ClaimManagement.sol

e83b966e92c9c78c48a514b4d149c641ea8eecab94f2afaa11b44d68fde02a3a ./contracts/CoverPoolFactory.sol

6e3f309b2dd8399bb2ed63df2cbf0d0a7d2db65039836468464d0eea67662246 ./contracts/Cover.sol

d0ebf1ea1239cff9f8da7a0c299d4b94318b434cc66863a36fdd09bb554e1041 ./contracts/ClaimConfig.sol

2f61d6adb9d656ef9dea5545d32e5955912fb018528011043553dc86da729333 ./contracts/CoverERC20.sol

8392617a3a5ecf55f0c0fcf8bdb7aca7bef433648804d8f85d691b7f72daa7ac ./contracts/interfaces/IOwnable.sol

877e4d97b548fa0b1a3da82da0081bc402a9eca534523aa56454053a17417d5b ./contracts/interfaces/ICoverERC20.sol

093e132b62ff8807f11e7c1b434417993213779aa03f08bf7d6b221a91b578db ./contracts/interfaces/ICoverPool.sol

e990e828d5881062e62928ce7b432996deee3af2df2e7cddb14cb538899b3530 ./contracts/interfaces/IClaimManagement.sol

27d7d5e7c2ed54977cf652c496653a24089987c86ae7db62fb07fc97a9a42933 ./contracts/interfaces/ICoverPoolCallee.sol

613db7753bc6a20b5e338accabe2e589d87a84e5f3aea491f9954418664d5d2e ./contracts/interfaces/IClaimConfig.sol

44b6d105fbe8fbf6f3e00465666964c4db9f3c8760c0bb33fcf7eddbb6900677 ./contracts/interfaces/ICover.sol

289b49477c0a1fcabbe4bc9406eea077e72abbe8bdddf8da4d10702c4848881c ./contracts/interfaces/ICovTokenProxy.sol

2d9094e04bb41a08be1bcdb910d59600229b604bf57d0c7b4a234f0548f3698d ./contracts/interfaces/ICoverPoolFactory.sol

beaa7957fb911221c25fb241c91223791c6cb010fbab60a2f6b952a3f61ce11c ./contracts/testing/FlashCover.sol

dd5602700b2e3e6da4e4d051db25b908c829cf7ef3d72f2225348087ede219fb ./contracts/utils/Initializable.sol

4a221930283da9be869eb6726890213322308f49ff9b325d918892cbac5c6701 ./contracts/utils/ReentrancyGuard.sol

fff4771d5cf625daf302e6a604c8ea3d3ccb038bd6797f9d2bf62033bfeb6649 ./contracts/utils/Ownable.sol

b6b3b74521aaf418f85136363bcde7e8a105844db03ebdab51bd238c6924b585 ./contracts/utils/Address.sol

4732978e7246021d84b2d07b967d6d1e10010d4898233d7378fcd970f2416c64 ./contracts/utils/Create2.sol

ed62fa60822fd36684e2a278e5d1fe8a0e5dc9b7d5b311fb9636ff470a3e80d0 ./contracts/utils/StringHelper.sol

647a1c145409f43a298b494424304c7fc4807f19238c0a20011d7b4518863506 ./contracts/ERC20/IERC20.sol

16c8db6010e5edbb18b1b0ac77b8ea7a7f5ea3cc89ecda7904a5fff9c058af36 ./contracts/ERC20/ERC20.sol

047eb50b7a42ce5fe534f279481eabd54631442c630c53006bc7825322c6ed49 ./contracts/ERC20/IERC20Permit.sol

ade2f02a5b4e3938778747b2317bbd8dc6a61aabb6fe8c6fcacb23c99852e712 ./contracts/ERC20/ERC20Permit.sol

914298c9034dea9e45a798360231977c9b037846ff774595125bde6e814afeef ./contracts/ERC20/SafeERC20.sol

181c2f4d60a58d95ba0b72541e319f2d03c95cc262ff119cb6aa6b22ca653a9d ./contracts/ERC20/ECDSA.sol

96a7f58c6a418953a47d9a641f02284251156e213a97492e4ee26b82f257ba91 ./contracts/ERC20/EIP712.sol

b19f1d039af0456f3908ecd76af7afd1f123ca2620238eadaf7c03d7c511c463 ./contracts/proxy/BaseUpgradeabilityProxy.sol

27f3f182315c4f5d315b2a5c4424e1d4959d4314bed11dd4fdf259db916caaa1 ./contracts/proxy/Clones.sol

4fae882d7abf8297e0f336f83ec08f47b25563acbfd3b1cd9548c449a436ec4e ./contracts/proxy/BaseAdminUpgradeabilityProxy.sol

d43c6173a7f70b6de3d92d8630ba09c20c26348d749383c3618a913ecce23fe8 ./contracts/proxy/Proxy.sol

d7a3f97c9edb0dc5681bd09c31e7a1321ab84191488a93135ff081436d63c7e2 ./contracts/proxy/InitializableAdminUpgradeabilityProxy.sol
```

Tests

```
4ac8e42a0e3967c897d5ec486ceaf6354434dccf3703cd83e038a5bec91a34b2 ./test/3-CoverPool-test.js

19860f2e7e40a0e30d268bfa5126bbcf4e81c5943d88018fea689512f9a433c7 ./test/testHelper.js

0dc7f623b0cf9bf0d9ae3ab896b95ba6656529aaa48ee8f9f3ae65ec37b2dee1 ./test/ClaimManagement-test.js

fbb5410fed95d7b35a74713a6a0ee69d748f3b2142ef0137640d918d9d4d2dae ./test/1-CoverPoolFactory-test.js

6b4fdc533a953d42ed61f96a70b9b959f0a90426dfd1f42d31cbe971ba72f62b ./test/eip712.js

0ecec691651bb781a0cf4b92dbd8ee06fafd1982cd0af8dcab907a1b9da3851a ./test/2-CoverERC20-test.js

a760716bcb1f4773d20feb0d5ef764d45c861a9f4b5f584898e318b5b7d06067 ./test/4-Cover-test.js

a0ab4a64da571cf9d53f0f007c15af88166a507120ab261f2ce9c92afd302621 ./test/ClaimConfig-test.js
```

Changelog

- 2021-04-09 - Initial report
- 2021-04-19 - Reaudit based on commit abc34e6

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.