Quantstamp Security Assessment Certificate

May 5th 2021 — Quantstamp Verified

# KeeperDAO

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

## Executive Summary

| | |
|---|---|
| Type | DeFi protocol |
| Auditors | Kacper Bąk, Senior Research Engineer<br>Mohsen Ahmadvand, Senior Research Engineer |
| Timeline | 2021-04-02 through 2021-05-04 |
| EVM | Muir Glacier |
| Languages | Solidity |
| Methods | Architecture Review, Unit Testing, Computer-Aided Verification, Manual Review |
| Specification | https://github.com/keeperdao/kcompound/wiki |
| Documentation Quality | High |
| Test Quality | Medium |

Source Code

| Repository | Commit |
|---|---|
| kcompound | 77871e3 |

Goals

- Can funds get locked up in the contract?
- Can fund migration fail?

| | |
|---|---|
| Total Issues | **7** (3 Resolved) |
| High Risk Issues | 0 (0 Resolved) |
| Medium Risk Issues | 0 (0 Resolved) |
| Low Risk Issues | **2** (2 Resolved) |
| Informational Risk Issues | **4** (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 reviewed the code, documentation, and test suite and found a few low-severity and informational issues. Overall, we consider the code to be well-written and with sufficient documentation. The test suite is in a good shape, although we recommend improving the branch coverage. We also provide suggestions for improvements to follow the best practices.
**Update:** the team fixed QSP-1 and QSP-2 in commit `383cdb7` and `69f7905`. Meanwhile, we also reviewed the following commits that fix other issues: `6ae6ca2`, `bed384c`, and `8ab8b6d`.

| ID | Description | Severity | Status |
|---|---|---|---|
| QSP-1 | Missing checks if addresses are non-zero | Low | Fixed |
| QSP-2 | Usage of the deprecated function `safeApprove()` | Low | Fixed |
| QSP-3 | Privileged Roles and Ownership | Informational | Fixed |
| QSP-4 | Gas Usage / `for` Loop Concerns | Informational | Acknowledged |
| QSP-5 | Optimizer Keccak Caching Bug | Informational | Acknowledged |
| QSP-6 | Potentially error-prone logic due to external indices | Informational | Acknowledged |
| QSP-7 | Incorrect/incomplete specification | 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 v0.7.1

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 Missing checks if addresses are non-zero

Severity: *Low Risk*

**Status:** Fixed

**File(s) affected:** `CompoundVars.sol`, `KCompound.sol`, `CompoundMigrator.sol`, `KCompoundPosition.sol`, `KComptroller.sol`

**Description:** There are no checks if addresses are non-zero in the following:

- `CompoundVars.constructor()`,
- `KCompound.constructor()`,
- `CompoundMigrator.constructor()`,
- `KCompoundPosition.constructor()`,
- `KComptroller.constructor()`.

**Recommendation:** Add relevant checks.

**Update:** Fixed in commit `383cdb7`. The team informed us that `CompoundVars.constructor()` does not check for the non-zero address argument intentionally since some contracts may be deployed after `CompoundVars`.

## QSP-2 Usage of the deprecated function `safeApprove()`

Severity: *Low Risk*

**Status:** Fixed

**File(s) affected:** `KCompoundPosition.sol`, `JITU.sol`, `CompoundMigrator.sol`

**Description:** The use of function `safeApprove()` is deprecated. Please find more details here.

**Recommendation:** Consider using `safeDecreaseAllowance()`/`safeIncreaseAllowance()` instead.

**Update:** Fixed in commit `69f7905`.

## QSP-3 Privileged Roles and Ownership

Severity: *Informational*

**Status:** Fixed

**File(s) affected:** `CompoundVars.sol`

**Description:** Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.

**Recommendation:** This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

**Update:** The team informed us that the contract will be governed by a DAO.

## QSP-4 Gas Usage / `for` Loop Concerns

Severity: *Informational*

**Status:** Acknowledged

**File(s) affected:** `CompoundMigrator.sol`

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

**Recommendation:** Perform an analysis of how many tokens may be supported in for-loops in `CompoundMigrator`.

## QSP-5 Optimizer Keccak Caching Bug

Severity: *Informational*

**Status:** Acknowledged

**File(s) affected:** `Comptroller.sol`

**Description:** Whereas it is unlikely to impact your code, there is a known optimizer keccak caching bug in Solidity versions older than 0.8.3.

**Recommendation:** Consider migrating to Solidity version 0.8.3 or newer.

## QSP-6 Potentially error-prone logic due to external indices

Severity: *Informational*

**Status:** Acknowledged

**File(s) affected:** `CompoundMigrator.sol`

**Description:** The function `migrateCallback()` iterates over `vars.cTokens()` and calls `migrateCTokenLoan(i, _account);` in each iteration, where `i` corresponds to the index of iterating `cTokens`. In the `migrateCTokenLoan()` function the `i`-th index of `var.cERC20s` is retrieved, while the original indices are populated based upon `var.cTokens()`. This can lead to problems if `vars.cTokens()` and `var.cERC20s` do not match exactly.

**Recommendation:** Consider using maps instead.

## QSP-7 Incorrect/incomplete specification

### Severity: *Undetermined*

**Status:** Acknowledged

**File(s) affected:** `KCompoundPosition.sol`

**Description:** According to the lines 102-103, 128-129:

> This will not consider the assets provided by JITU to underwrite the current compound position

However, the functions `redeemAllowed()` and `borrowAllowed()` call `getAccountLiquidity()`, which calls `bufferCollateralValueInUSD()`, so the buffer is actually considered in the shortfall calculations.

**Recommendation:** We recommend aligning the specification with the implementation.

**Update:** The team informed us they'll fix the comment. The team also clarified that “the buffer is not considered” means that, when someone looks at the position through Compound directly, they see the position as `Collateral=UserCollateral+BufferProvidedByJITU` and `Loan=UserLoan`; when they call `borrowAllowed()` or `redeemAllowed()` we only consider `Collateral=UserCollateral` and `Loan=UserLoan`.

# Automated Analyses

### Slither

Slither reported the following:

- ether transfers to arbitrary user in `CompoundMigrator.sol` (lines 37, 58, 101, 280, 294)
- strict equalities in `Exponential.sol` (lines 37, 439) and in `KComptroller.sol` (lines 45, 62, 94, 154, 158, 174, 225, 262, 306, 313, 343, 365).

We classified all the findings as false positives.

# Code Documentation

Overall the code is well-documented, however,

1. The wiki should be updated to reflect new contract names, i.e., `KCompoundWallet` -> `KPosition`, `KWalletFactory` -> `KFactory`;
2. `KCompoundPosition.preempt()` is missing natspec for `_liquidator`;
3. Document the significance of `0xEeeeeEeeeEeEeeEeEeEeeeEEEeeeeEeeeeeeeEEeE` in `KCompoundPosition.sol#198`,
4. We recommend documenting how the COMP interest are accrued and distributed among users.
5. In `KComptroller.sol#247`, consider adding a more detailed comment explaining the buffer check.

# Adherence to Best Practices

1. In `CompoundVars.sol`, the function `addCERC20()` is defined as a `public`. Consider using `external` instead.
2. In `KComptroller.sol`, the function `requireNoError()` is defined as public. Consider using `internal` instead.

# Test Results

### Test Suite Results

All tests passed. Please note that the contract `SimplePriceeOracle` is used for testing purposes only and therefore it is excluded from the audit.

```
  KCompoundPosition
     ✓ Should be able to mint a compound position (176ms)
     ✓ Should return the correct address as beneficiary (41ms)
     ✓ Should fail to deposit not registered token (43ms)
     ✓ Should fail to deposit zero ETH (44ms)
Balance before deposit:  0 0
Balance after deposit:  99999940931109 498992
     ✓ Should be able to deposit ETH (100000000000000) (144ms)
     ✓ Should fail to deposit zero DAI
Balance before deposit:  0 0
Balance after deposit:  999999999999871613195 4686197440071
     ✓ Should be able to deposit DAI (100000000000000000000) (167ms)
     ✓ Should fail to borrow not registered token
     ✓ Should fail to borrow zero amount of DAI
     ✓ Should fail to borrow zero amount of ETH
Balance before borrow:  0
Balance after borrow:  100000000000000
     ✓ Should be able to borrow DAI (100000000000000) (299ms)
Balance before borrow:  0
Balance after borrow:  100000000000000
     ✓ Should be able to borrow ETH (100000000000000) (288ms)
     ✓ Should fail to repay not registered token
     ✓ Should fail to repay zero amount of DAI
     ✓ Should fail to repay zero amount of ETH
Balance before repayment:  100000000000000
Balance after repayment:  0
     ✓ Should be able to repay DAI (86ms)
Balance before repayment:  100000000000000
Balance after repayment:  0
     ✓ Should be able to repay ETH (71ms)
     ✓ Should fail to withdraw not registered token
     ✓ Should fail to withdraw zero amount of ETH (41ms)
     ✓ Should fail to withdraw zero amount of DAI (48ms)
Balance before withdrawal:  99999941802445 498992
Balance after withdrawal:  49999970925426 249496
     ✓ Should be able to withdraw ETH (50000000000000) (352ms)
Balance before withdrawal:  1000000326490973339108 4686197440071
Balance after withdrawal:  500000346896814228188 2343099532849
     ✓ Should be able to withdraw DAI (500000000000000000000) (339ms)
     ✓ Can claim COMP tokens
     ✓ Can add new compound tokens (95ms)
Balance before migrate:  24949614
```

```
Balance after migrate:  0
     ✓ Should be able to migrate a position (1277ms)
Balance before migrate:  23430971422187
Balance after migrate:  0
     ✓ Should be able to migrate a position (1162ms)
     ✓ ALICE can transfer her position to BOB

  CompoundVars
     ✓ Should not be able to update JITU with address 0
     ✓ Should not be able to update KCompound with address 0
     ✓ Should not be able to update KComptroller with address 0
     ✓ Should not be able to update Comptroller with address 0
     ✓ Can update comptroller
     ✓ Should not be able to add erc20 with address 0
     ✓ Should not be able to add erc20 that already exists
     ✓ Should not be able to get cTokenWrapper with address 0

  KCompound
     ✓ Should be able to mint a compound position (54ms)
     ✓ Should fail to update CompoundVars with 0 address
     ✓ Can update CompoundVars address
     ✓ Should fail to update KFactory with 0 address
     ✓ Can update KFactory address

  KComptroller
1000000000000000000 cdai =>  998955000000000000  USD
1000000000000000000 ceth =  2227285000000000000000  USD
1000000000000000000 cwbtc =  622187400000000000000000000000000  USD
     ✓ Should be able to calculate the value of the given underlying token amount in USD
1000000000000000000 cdai =>  1598773538297918509507975 16  USD
1000000000000000000 ceth =>  3347674506280829949283954968 89  USD
1000000000000000000 cwbtc =>  817108095771122024354100000000  USD
     ✓ Should be able to calculate the collateral value of the given cToken amount (57ms)
     ✓ Should not revert when the error code is 0
     ✓ Should revert if the error code is not zero

  KCompound - jitu
Balance before deposit:  0
Balance after deposit:  1999999999944314666
     ✓ Should be able to deposit eth (130ms)
Balance before borrow:  100000000000000000000000
Balance after borrow:  103300000000000000000000
     ✓ Should be able to borrow dai (325ms)
Loan before liquidation:  330000000000000000000
Simulating a price fall from  22272850000000000000  to 20000000000000000000
Loan after liquidation:  1650000338235427206261
     ✓ Should be able to preempt a position that is underwater (404ms)
Balance before deposit:  0
Balance after deposit:  1000000000000000000
     ✓ Should be able to deposit WETH tokens (40ms)
Balance before deposit:  0
Balance after deposit:  200000000000000000000
     ✓ Should be able to deposit DAI tokens
Loan before liquidation:  167000000000000000000
Simulating a price fall from  22272850000000000000  to 20000000000000000000
Loan after liquidation:  835000244525717665614
     ✓ Should be able to preempt by providing buffer (Collateral: ETH) (1106ms)
Loan before liquidation:  1000000000000000000
Simulating a price fall from  22272850000000000000  to 23000000000000000000
Unhealth raised from  99  to  102
Unhealth falls from  102  to  87
Loan after liquidation:  500000071556004251
     ✓ Should be able to preempt by providing buffer (Collateral: DAI) (2063ms)
Balance before withdraw:  1000000000000000000
Balance after withdraw:  2000000000742790131
     ✓ Should be able to withdraw (38ms)
     ✓ Should fail to call JITU with blacklisted keeper
     ✓ Should check max buffer value (66ms)

  Underwriter Story
[underwriter.deposit] Ether balance:  9990300509240000000000  ->  9990297859280000000000
[underwriter.deposit] kEther balance:  0  ->  2000000000000000
[underwriter.deposit] Dai balance:  1074850000000000000000000  ->  1074830000000000000000000
[underwriter.deposit] kDai balance:  0  ->  2000000000000000000
     ✓ Underwriter can deposit tokens to JITU contract and will receive kTokens (518ms)
[underwriter.withdraw] Ether balance:  9998819996288000000000  ->  9998820996288000000000
[underwriter.withdraw] kEther balance:  2000000000000000  ->  1000000000000000
[underwriter.withdraw] Dai balance:  9450000500000000000000  ->  9450100500000000000000
[underwriter.withdraw] kDai balance:  2000000000000000000  ->  1000000000000000000
     ✓ Underwriter can withdraw tokens from JITU contract by providing kTokens (can specify the recipient who will get the borrowed tokens) (74ms)
     ✓ Underwriter can blacklist or whitelist keepers
     ✓ Underwriter should never loose funds if the position is liquidated by a third-party liquidator (514ms)


  58 passing (19s)
```

## Code Coverage

The code features decent coverage.

| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
|---|---|---|---|---|---|
| **contracts/** | 99.64 | 78.36 | 100 | 100 | |
| CompoundMigrator.sol | 100 | 81.25 | 100 | 100 | |
| CompoundVars.sol | 100 | 95 | 100 | 100 | |
| Interfaces.sol | 100 | 100 | 100 | 100 | |
| JITU.sol | 100 | 100 | 100 | 100 | |
| KCompound.sol | 100 | 100 | 100 | 100 | |
| KCompoundPosition.sol | 100 | 84.21 | 100 | 100 | |
| KComptroller.sol | 99.1 | 64.81 | 100 | 100 | |
| KFactory.sol | 100 | 100 | 100 | 100 | |
| Tokens.sol | 100 | 100 | 100 | 100 | |
| **All files** | **99.64** | **78.36** | **100** | **100** | |
| **Initial audit stats** | 91.26 | 74.38 | 100 | 91.72 | |

# 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

```
4deede42577b47d76b56bf8746e97ee93652fa334d9d217ec09073cb863ecf24 ./contracts/Tokens.sol
36097198c801b0febdae7edc7f8f4bef353014d17abb183cab88032bef0de2d1 ./contracts/KComptroller.sol
7b5189b237ee2b971e354c46224868298772632617e4ec988da6508090553425 ./contracts/KCompoundPosition.sol
81a08443fae1336a88262c3d868489938696c57d33d232575a20f61f6642c254 ./contracts/JITU.sol
475475f82baa54d58b46fe38064e163f42ad2d7047d30c52eee9028ec69c2a5c ./contracts/CompoundVars.sol
d467186d70ac161731bcb163f7b814ec8e83cea579a25bcdf3677773faf85299 ./contracts/KCompound.sol
14ccc7855df2c2e1f21042ae1a9c3eb073d200eef3285843a181623077992f8f ./contracts/Interfaces.sol
69961878482bc877577e841bf73d17f2acd49b31ad9ed5be7cc73f29d751094c ./contracts/CompoundMigrator.sol
f39ca1b637a4ad0d45b4c73f5e33c4c7d2355d14d3ca93770802ffaeeb46225e ./contracts/compound/Comptroller.sol
da66a59c718691e789bf652cac3903899de2bcd983b9487c93257d01af6415cb ./contracts/compound/Exponential.sol
34872c11e1bad45a44cb2da706615477f44bf020f7903ef104a12e434dfbe7f0 ./contracts/compound/CToken.sol
b2c68c0fed85c8b42900007008f062934b7d3204d99cddc3ee454963c06bc9a3 ./contracts/compound/ComptrollerErrorReporter.sol
```

### Tests

```
b022c4bf489c1bffa9d23163bd213f1dcf4e7fdca74170992ed7fbbb16a8e7af ./test/helper.ts
0ad46af019cab5d9ae815aa551fa835b59f82da08803c1012521fd0e2e143ff0 ./test/user-stories/underwriter.ts
e5c26f72b3d2136b711e1f5ea82b2f0156db3c6977e3f04d17a541ce92e3bf37 ./test/user-stories/keeper.ts
fbd9bb0110a825fd643f708434f93ddb197e061b5d5c9b3be43693ae5c8f7c9d ./test/user-stories/user.ts
92cba4d0fa4c6cfef3b0f37a270914d2d09de0d98c151ee5df0972c64cf65cf3 ./test/unit-tests/compound-vars.ts
1db28bb22b28a39c97020e104004bdaf3090ee0fa38d5204de2fea65bf6b27ae ./test/unit-tests/underwriter.ts
9322c7c10d26a26b7f2ef83496eb90683220538f4e09868f4f7549025427ff2a ./test/unit-tests/kcompound.ts
62647d56eb9375779b3df85f53da09cb3c775c37b0ee0dd354582f6fa6418c15 ./test/unit-tests/compound-position.ts
5aa74fd6d023fbfa6155dd0f54ea566003eef63f79a1fd55ba41033a7356d9b1 ./test/unit-tests/kcomptroller.ts
```

# Changelog

- 2021-04-21 - Initial report
- 2021-05-04 - Revised report based on commit `383cdb7` and `69f7905`.

## 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.