Quantstamp Security Assessment Certificate

April 6th 2021 — Quantstamp Verified

Covalent Query Token

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

Executive Summary

Type	ERC20
Auditors	Leonardo Passos, Senior Research Engineer Poming Lee, Research Engineer Ed Zulkoski, Senior Security Engineer
Timeline	2020-10-05 through 2020-10-22
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Vesting Google Doc Spreadsheet
Documentation Quality	Low
Test Quality	Medium

Source Code

Repository	Commit
covalent-query-token	a4e4b73
None	4cd2968
None	500ac62

Total Issues	**8** (3 Resolved)
High Risk Issues	**3** (2 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	0 (0 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 found three high severity issues in the audited code base, specifically: two issues related to undocumented privileged roles on the token vesting code that could potentially affect token holders, and one issue related to a mismatch between the implementation and the given specification. All other issues (5), are either undetermined (1) or informational (4). Last, but not least, we found the test branch coverage to be not ideal (currently at around 73%) - it should be as close as possible to 100%. Altogether, we recommend prompt attention from developers to address all the issues herein reported prior to deployment of the audited contracts.

ID	Description	Severity	Status
QSP-1	Privileged roles: Vesting can be unilaterally revoked	High	Mitigated
QSP-2	Privileged roles: Vesting balance can be defunded	High	Mitigated
QSP-3	Vesting values do not match the "Vesting Contract" spreadsheet spec	High	Acknowledged
QSP-4	External and public functions are not fully annotated using the Natspec format	Informational	Unresolved
QSP-5	Clone-and-Own	Informational	Acknowledged
QSP-6	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-7	Use of early `ERC20Permit` draft	Informational	Acknowledged
QSP-8	Privileged roles: Tokens may not be rescued upon ownership renouncement	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 .`
3. Filter out false-positives (manual step)

Findings

QSP-1 Privileged roles: Vesting can be unilaterally revoked

Severity: *High Risk*

Status: Mitigated

File(s) affected: `contracts/CovalentQueryTokenVesting.sol`

Description: The same way the contract owner can grant vesting to an address, it can unilaterally revoke it by calling `removeVesting`. However, there is no documentation stating under what conditions should a vesting be removed.

Recommendation: Provide user facing documentation stating under what conditions `removeVesting` is called. Alternatively, rely on a multisig wallet to perform the operation.

> *Update*: It was confirmed that a multisig wallet will be used for handling owner-only operations.

QSP-2 Privileged roles: Vesting balance can be defunded

Severity: *High Risk*

Status: Mitigated

File(s) affected: `contracts/CovalentQueryTokenVesting.sol`

Description: According to the deployment scripts, the `CovalentQueryTokenVesting` address is given 836458333 tokens, which corresponds to the vesting supply. However, the owner of `CovalentQueryTokenVesting` could transfer that fund to any address he so chooses. Then, any user given a vesting will not be able to release his tokens after the corresponding release time.

Recommendation: Use a multisig wallet to mitigate the potential centralization of a single person having access to all the vesting funds. Any operation required or needed by the owner should then meet the quorum requirements of the multisig wallet.

> *Update*: It was confirmed that a multisig wallet will be used for handling owner-only operations.

QSP-3 Vesting values do not match the "Vesting Contract" spreadsheet spec

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `contracts/CovalentQueryTokenVesting.sol`

Description: The vesting schedule as set in the contract does not match the Google Doc Spreadsheeet. A spec not matching the code can lead to a deployed contract that does not match the expected behavior. For example, the value after 90 days is 63541667 in the spreadsheet, but 54166667 in the code.

Recommendation: We recommend reviewing the spreadsheet spec against the implemented vesting contract, making sure both are in synch.

> *Update*: According to developers, the vesting is still subject to change and updates can occur in the code, which may not be necessarily reflected across the board (e.g., spreadsheets).

QSP-4 External and public functions are not fully annotated using the Natspec format

Severity: *Informational*

Status: Unresolved

File(s) affected: `contracts/*`

Description: External and public functions are not fully annotated using the Natspec format; lacking detailed documentation could impair users of the purpose of each function and its usage.

Recommendation: Fully annotate public and external functions using NatSpec.

QSP-5 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/ERC20Permit/*`

Description: The `ERC20Permit` implementation is a clone from Openzeppelin's draft implementation. Cloning requires one to monitor the cloned code for potential issues, as well as manually merging fixes and patches. If not managed properly, it could lead to outdated and vulnerable code being pushed to production.

Recommendation: At the very least, document that the code was copied from Openzeppelin, as well as indicating the commit hash adding/changing the `ERC20Permit` implementation in Openzeppelin's repository.

QSP-6 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Mitigated

File(s) affected: `contracts/CovalentQueryToken.sol`

Description: As with any other ERC20 contract, `CQT` is vulnerable to the allowance double-spend exploit.

Exploit Scenario:

Recommendation: The issue can be mitigated through the use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance` and

`decreaseAllowance` (already the case for the `CQT` token contract).

QSP-7 Use of early `ERC20Permit` draft

Severity: *Informational*

Status: Acknowledged

File(s) affected: `contracts/ERC20Permit/*`

Description: The `ERC20Permit` implementation is a clone from Openzeppelin's draft code. It should be used with caution, as that code, as it stands, does not have a comprehensive test suite. According to its author, the given implementation is not yet mature:

> *This adds support for the ERC20 Permit extension, tentatively called ERC2612 and based on the discussion in ethereum/EIPs#2612. The EIP is still in draft status: we'll need to monitor it for any changes, and will likely not want to publish this code until it is more mature / finalized*

Recommendation: As of this audit, we did not find any vulnerability in `contracts/ERC20Permit/*.sol`. Nonetheless, a comprehensive test suite (currently lacking) could evidence issues that could be lingering around and/or that could be missed by a manual audit. Hence, we suggest monitoring the cloned code for potential bugs and/or issues, having an upgrade plan in case the code is found to be prone to any attack and/or issues.

QSP-8 Privileged roles: Tokens may not be rescued upon ownership renouncement

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `contracts/CovalentQueryToken.sol`

Description: The `rescueToken` function seems to exist as a means to aid users to recover their tokens in case they inadvertently transfer tokens to the `CovalentQueryToken` contract . As acknowledged by the comments in the code, this function can be disabled by destroying ownership, i.e., when the owner invokes the `renounceOwnership` function. Hence, its disabling is permanent. However, it is unclear under what circumstances this function should be disabled.

Recommendation: Properly document under what conditions disabling `rescueToken` is deemed safe. Additionally, make sure that if such disabling requirements do not hold, the function will continue enabled.

> *Update*: Disabling will be controlled by a multisig wallet that (hopefully) should be given control to the community. It is up to them when to disable `rescueToken`.

Automated Analyses

Slither

• CovalentQueryToken.constructor(string,string,uint256).name (CovalentQueryToken.sol#13) shadows:
  . ERC20.name() (openzeppelin-solidity/contracts/token/ERC20/ERC20.sol#66-68) (function)

• CovalentQueryToken.constructor(string,string,uint256).symbol (CovalentQueryToken.sol#13) shadows:
  . ERC20.symbol() (openzeppelin-solidity/contracts/token/ERC20/ERC20.sol#74-76) (function)

• CovalentQueryToken.constructor(string,string,uint256).totalSupply (CovalentQueryToken.sol#13) shadows:
  . ERC20.totalSupply() (openzeppelin-solidity/contracts/token/ERC20/ERC20.sol#98-100) (function)
  . IERC20.totalSupply() (openzeppelin-solidity/contracts/token/ERC20/IERC20.sol#12) (function)

• ERC20.constructor(string,string).name (openzeppelin-solidity/contracts/token/ERC20/ERC20.sol#57) shadows:
  . ERC20.name() (openzeppelin-solidity/contracts/token/ERC20/ERC20.sol#66-68) (function)

• ERC20.constructor(string,string).symbol (openzeppelin-solidity/contracts/token/ERC20/ERC20.sol#57) shadows:
  . ERC20.symbol() (openzeppelin-solidity/contracts/token/ERC20/ERC20.sol#74-76) (function) token() should be declared external:
  . CovalentQueryTokenVesting.token() (CovalentQueryTokenVesting.sol#105-107)

• beneficiary(uint256) should be declared external:
  . CovalentQueryTokenVesting.beneficiary(uint256) (CovalentQueryTokenVesting.sol#114-116)

• releaseTime(uint256) should be declared external:
  . CovalentQueryTokenVesting.releaseTime(uint256) (CovalentQueryTokenVesting.sol#123-125)

• vestingAmount(uint256) should be declared external:
  . CovalentQueryTokenVesting.vestingAmount(uint256) (CovalentQueryTokenVesting.sol#132-134)

• removeVesting(uint256) should be declared external:
  . CovalentQueryTokenVesting.removeVesting(uint256) (CovalentQueryTokenVesting.sol#140-147)

• release(uint256) should be declared external:
  . CovalentQueryTokenVesting.release(uint256) (CovalentQueryTokenVesting.sol#175-187)

• retrieveExcessTokens(uint256) should be declared external:
  . CovalentQueryTokenVesting.retrieveExcessTokens(uint256) (CovalentQueryTokenVesting.sol#193-196)

• setCompleted(uint256) should be declared external ermit(address,address,uint256,uint256,uint8,bytes32,bytes32) should be declared external:
  . ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (ERC20Permit/ERC20Permit.sol#45-72)

• nonces(address) should be declared external:
  . ERC20Permit.nonces(address) (ERC20Permit/ERC20Permit.sol#77-79)

Adherence to Best Practices

• The `rescueTokens` function exists to recover tokens that users may inadvertently transfer to the `CovalentQueryToken` contract (such transfer cause users to

• effectively lose their tokens). At first, the purpose of `rescueTokens` appears unclear, as the documentation is rather short. We suggest having improved documentation stating the functions purpose. Example: "This function allows rescuing funds that users may inadvertently transfer to the CovalentQueryToken contract". **Not fixed**

• In `contracts/CovalentQueryTokenVesting.sol`, the constructor does not check if the given contract address is a contract or not. Add such a check (e.g., using the `Address` library). **Not fixed**

• In `contracts/CovalentQueryTokenVesting.sol` (L42) the message "CovalentQueryToken token address is not valid" covers more cases that the check being done: `address(_token) != address(0x0)`. Hence, we suggest changing it to “CovalentQueryToken token address is not different from 0x0”. **Fixed**

• Some functions have parameters that shadow state variables. We recommend renaming such parameters. See Slither output. **Not fixed**

• Following Slither's recommendation, some functions should be declared as `external` instead of `public` to save gas. See Slither output. **Not fixed**

• In the `README.md` file, document which node version one should be using, as well as updated instructions on how to run the test suite. **Not fixed**

Test Results

Test Suite Results

Overall, the test suite has been found to cover many test case scenarios, although not necessarily covering all possible code paths (see coverage info).

```
Contract: Token
  Setup: totalsupply, permit
     ✓ returns the total amount of tokens (77ms)
     ✓ returns correct permit hash
  transfer
     ✓ should successfully transfer 1 wei (81ms)
     ✓ should successfully transfer full balance (72ms)
     ✓ should fail to transfer amount exceeding balance (71ms)
  Rescue funds
     ✓ Should be able to get accidentally sent tokens back (356ms)
  permit function
     ✓ permit (184ms)
     ✓ permit: wrong data (178ms)

Contract: Token
  Token Vesting
     ✓ should test token vesting for userX (437ms)
     ✓ should test addVesting data (120ms)
     ✓ Removing a vesting entry with the owner account (204ms)
     ✓ Removing a vesting entry with a non-owner account (73ms)
     ✓ Trying to remove a non-existent vesting entry (58ms)
     ✓ Trying to remove an already released vesting entry (155ms)
     ✓ Trying to remove an already removed vesting entry (115ms)
     ✓ Trying to add a vesting entry from a non-owner account (52ms)
     ✓ should test token vesting for amount greater then balance of vesting contract (208ms)
     ✓ Trying to release the tokens associated with existing vesting entry (86ms)
     ✓ should test token vesting for amount exactly equal to the balance of vesting contract (4174ms)


19 passing (28s)
```

Code Coverage

Branch coverage is not extensive; around 27% of code branches are left untested, which could hide potential issues and or bugs. We suggest increasing that coverage to as close as 100%.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	73.08	100	100	
CovalentQueryToken.sol	100	50	100	100	
CovalentQueryTokenVesting.sol	100	77.27	100	100	
contracts/ERC20Permit/	100	75	100	100	
ERC20Permit.sol	100	75	100	100	
IERC2612Permit.sol	100	100	100	100	
All files	**100**	**73.33**	**100**	**100**	

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

```
65c2fcf6bfd749f781709745d974d5a36a5cc920f496750c22befc8481da6701 ./contracts/CovalentQueryToken.sol
826324acff11e98482f8c42ae6bdc18dd3a592d402eabc0ae2018fa36cc9a78d ./contracts/Migrations.sol
f038e2daa6e29a8c2a03ddeb3674dacfa5a5e2e068b1d69cb38758c44ad77d64 ./contracts/CovalentQueryTokenVesting.sol
ec61823e14f5b2796866dcac4bd13ade1d858583ecbc5e91480786d36b79cfb1 ./contracts/ERC20Permit/ERC20Permit.sol
7fbb3fa5739ba273766081abcb03ff6f3fa15da76ee013436e8d33ad861b6986 ./contracts/ERC20Permit/IERC2612Permit.sol
```

Tests

```
17883bb1c939e025db28cf2b70d80fdfbdb66caeba23fc5fcd703ecb10735733 ./test/CovalentQueryToken.test.js
3ffb3eb1307af2923a3f5ddca720b117282347f7de5d5cc22bc6b363b9f437c5 ./test/CovalentQueryTokenVesting.test.js
80a9ab4ec5adc9da3817ecd91eb8ef764070ea8198ceb0da182bb9674998ff4c ./test/utils/permitUtils.js
```

Changelog

- 2020-10-09 - Initial report
- 2020-10-19 - Re-audit
- 2020-10-22 - Re-audit (2)

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.