Quantstamp Security Assessment Certificate

June 18th 2021 — Quantstamp Verified

Qredo Token

This smart contract audit was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Token contract
Auditors	Kacper Bąk, Senior Research Engineer Jose Ignacio Orlicki, Senior Engineer
Timeline	2021-05-26 through 2021-06-08
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Specification
Documentation Quality	High
Test Quality	Medium

Source Code

Repository	Commit
qredo-token	67ce8d3

Goals

- Is the implementation prone to front-running?
- Are the computations implemented correctly?

Total Issues	4 (2 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	2 (1 Resolved)
Informational Risk Issues	2 (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

Overall the code is well written and follows best practices. We recommend, however, addressing all of the issues before deploying the contract.

ID	Description	Severity	Status
QSP-1	Privileged Roles and Ownership	Low	Acknowledged
QSP-2	Missing Input Validation	Low	Fixed
QSP-3	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-4	Clone-and-Own	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 v0.7.1

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 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `QredoToken.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. Specifically, some accounts are authorized to mint tokens (up to a certain global limit)

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.

QSP-2 Missing Input Validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `QredoToken.sol`

Description: The constructor does not check if values of `name_` and `symbol_` are non-empty.

Recommendation: We recommend adding the relevant checks.

QSP-3 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Mitigated

File(s) affected: `QredoToken.sol`

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

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-4 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

File(s) affected: `QredoToken.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.
Specifically, the `Mintable` and `Burnable` functionalities can be reused from OZ 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.

Automated Analyses

Slither

Slither did not report any issues.

Adherence to Specification

We found the following discrepancies between specification and implementation:

1. According to the specification functions may return `false` value and it is important to the check this (`Callers MUST handle false from returns (bool success). Callers MUST NOT assume that false is never returned!`). The functions, however, never return `false` and in case of a problem they just fail and revert.
2. According to the specification name and symbol of the token are fixed to `QREDO` and `QRDO`, respectively. The implementation, however, allows one to set any name and symbol. The same applies to decimals.

Code Documentation

Document that the event `Approval()` is also emitted by functions `increaseAllowance()` and `decreaseAllowance()`.

Adherence to Best Practices

Overall, the code adheres to best practices, however a magic number is used in line 28. Probably it signifies that 10% of the supply goes to the deployer. We recommend using a named constant instead.

Test Results

Test Suite Results

All tests executed successfully.

```
Contract: Token tests
  0.QredoToken:name
     ✓ 0.should get proper name (46097 gas)
  1.QredoToken:symbol
     ✓ 0.should get proper symbol
  2.QredoToken:decimals
     ✓ 0.should get proper decimals
  3.QredoToken:totalSupply
     ✓ 0.should get proper totalSupply
  4.QredoToken:circulatingSupply
     ✓ 0.should get proper circulatingSupply
  5.QredoToken:balanceOf
     ✓ 0.should get proper balanceOf
  6.QredoToken:transferTokens
     ✓ 0.should transfer tokens to owner (36312 gas)
  7.QredoToken:transferOwnership
     ✓ 0.should transfetOwnership (30945 gas)

Contract: Token tests
  0.QredoToken:name
     ✓ 0.should get proper name (46097 gas)
  1.QredoToken:symbol
     ✓ 0.should get proper symbol
  2.QredoToken:decimals
     ✓ 0.should get proper decimals
  3.QredoToken:totalSupply
     ✓ 0.should get proper totalSupply
  4.QredoToken:circulatingSupply
     ✓ 0.should get proper circulatingSupply
  5.QredoToken:balanceOf
     ✓ 0.should get proper balanceOf
  6.QredoToken:mint
     ✓ 0.should successfully mint tokens (38681 gas)
     ✓ 1.should not successfully mint tokens to zero address (22892 gas)
     ✓ 2.should not successfully mint tokens over totalSupply (24938 gas)
  7.QredoToken:transfer
     ✓ 0.should successfully transfer tokens (51300 gas)
     ✓ 1.should not transfer tokens successfully to 0 address (21933 gas)
     ✓ 2.should not transfer tokens successfully without enough balance (23100 gas)
  8.QredoToken:approve
     ✓ 0.should approve tokens successfully (44197 gas)
     ✓ 1.should not approve tokens successfully to 0 address (21957 gas)
  9.QredoToken:increaseAllowance
     ✓ 0.should increase allowance tokens successfully (30356 gas)
     ✓ 1.should not increase allowance successfully to 0 address (21860 gas)
  10.QredoToken:decreaseAllowance
     ✓ 0.should decrease allowance tokens successfully (30575 gas)
     ✓ 1.should not decrease allowance successfully to 0 address (21926 gas)
     ✓ 2.should not decrease allowance successfully below zero (23217 gas)
  11.QredoToken:transferFrom
     ✓ 0.should successfully transferFrom tokens (30184 gas)
     ✓ 1.should not transferFrom tokens successfully to 0 address (83338 gas)
     ✓ 2.should not transferFrom tokens successfully without enough allowance (23559 gas)
     ✓ 3.should not transferFrom tokens successfully without enough balance (84601 gas)
  12.QredoToken:burn
     ✓ 0.should successfully burn tokens (17945 gas)
     ✓ 1.should not burn tokens successfully without enough balance (22772 gas)
  13.QredoToken
     ✓ 0.should check test passed successfully
```

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

`4fd6092bdfa8b42f19d535c5ac69c4323b0b894717c699e58d5552eeabd04cd4 ./contracts/Migrations.sol`

`5a8a36964620187c6c25271c83cc44c21a10decfed3b4adffbaaa773cfd5b1e7 ./contracts/QredoToken.sol`

`7abe305fb810b3b1181934361219ab1b85ca09bcd54067da38db50a21a75295c ./contracts/utils/Authorizable.sol`

`0cd2042adb78c962b12a91cebd4c91c32c0c44196ab713e0a465f0fdcb1570d2 ./contracts/utils/Context.sol`

`2b2daa20a2253ba514c04439a2a7a619d50d2b009ff8e98488236fe5ee56a6a5 ./contracts/ownable/Ownable.sol`

Tests

`b4f863e43141c6ab3c2611438f6ab48e6ebf40b75df15ddae4a3e2d33d2ef444 ./test/qredo_tests.js`

`6051972aa8fc38c4f524fe8e467861b6582ecc4f7953edd2b4fd9f8b76556e20 ./test/qredo_deployment.js`

`dd99f87961fb0cad5b30ed859398af6ae58cea1cfd286942904613be85f48224 ./test/utils/utils.js`

Changelog

- 2021-05-28 - Initial report
- 2021-06-08 - Revised report based on commit `704c81a`.

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.