Quantstamp Contract Security Certificate

October 15th 2019 — Quantstamp Verified

Coin Zoom

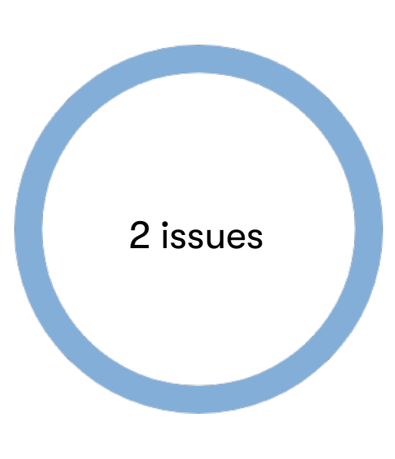
This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Token contract
Auditors	Martin Derka, Senior Research Engineer Nadir Akhtar, Software Auditing Intern Yohei Oka, Forward Deployed Engineer
Timeline	2018-11-26 through 2018-12-04
Languages	Solidity
Methods Specification	README
Source Code	

Repository	Commit
ZoomToken	a339bf8

Total Issues	**2** (0 Resolved)
High Risk Issues	0
Medium Risk Issues	0
Low Risk Issues	0
Informational Risk Issues	2 (0 Resolved)
Undetermined Risk Issues	0



Overall Assessment

The token is a standard ERC20 token. The implementation is clean and minimalistic with the use of OpenZeppelin libraries.

Severity Categories	
High	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	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	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.

Goals

Changelog

- 2018-12-04 - Initial report

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the Coin Zoom 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 below notes outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Truffle v4.1.12
- Ganache v1.1.0
- Oyente v1.2.5
- Mythril v0.2.7
- MAIAN commit sha: ab387e1
- Securify

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
4. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
5. Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
6. Installed the Mythril tool from Pypi: `pip3 install mythril`
7. Ran the Mythril tool on each contract: `myth -x path/to/contract`
8. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
9. Installed the Oyente tool from Docker: `docker pull luongnguyen/oyente`
10. Migrated files into Oyente (root directory): `docker run -v $(pwd):/tmp - it luongnguyen/oyente`
11. Ran the Oyente tool on each contract: `cd /oyente/oyente && python oyente.py /tmp/path/to/contract`
12. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
13. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`

Assessment

Findings

Allowance Double-Spend Exploit

Severity: *Informational*

Contract(s) affected: `CZToken.sol`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens. An example of an exploit goes as follows:

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

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as increaseAllowance and decreaseAllowance.

Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on approve() / transferFrom() should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.

Centralization of Power

Severity: *Informational*

Contract(s) affected: `CZToken.sol`

Description: Smart contract allows accounts with the role of a `Pauser` to suspend transfers of the token. This is problematic especially when private keys of such accounts are compromised. Such a centralization of power should be made clear to the users.

Recommendation: The Quantstamp team recommends that the pausing features are well communicated to the users and that the Coin Zoom team takes all reasonable precautions for protecting the private keys of pausers.

Test Results

Test Suite Results

The minimalistic implementation does not require many tests. The test suite well covers basic invariants and is adequate.

```
Contract: CZToken
     ✓ has a name (68ms)
     ✓ has a symbol (65ms)
     ✓ has 18 decimals (80ms)
    balanceOf
      when the requested account has no tokens
        ✓ returns zero (92ms)
    INITIAL_SUPPLY
      Initial supply of tokens
        ✓ is correct
    when tokens are burned
BigNumber { s: 1, e: 26, c: [ 9999999900000 ] }
      ✓ Burn test (285ms)
    transfer
      when the recipient is not the zero address
        when the sender has enough balance
          ✓ transfers the requested amount (168ms)
          ✓ emits a transfer event (130ms)
      when the recipient is the zero address
        ✓ reverts (91ms)
    approve
      when the spender is not the zero address
        when the sender has enough balance
          ✓ emits an approval event (83ms)
          when there was no approved amount before
            ✓ approves the requested amount (210ms)
          when the spender had an approved amount
            ✓ approves the requested amount and replaces the previous one (177ms)

  12 passing (5s)
```

Code Coverage

The code is well covered with tests.

```
----------------|----------|----------|----------|----------|----------------|
File            |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------|----------|----------|----------|----------|----------------|
 contracts/     |      100 |      100 |      100 |      100 |                |
  CZToken.sol   |      100 |      100 |      100 |      100 |                |
----------------|----------|----------|----------|----------|----------------|
All files       |      100 |      100 |      100 |      100 |                |
----------------|----------|----------|----------|----------|----------------|
```

Automated Analyses

Oyente

Oyente reported no issues.

Mythril

Mythril reported no issues.

MAIAN

MAIAN reported no issues.

Securify

Securify reported no issues.

Adherence to Specification

The code adheres to the requirements that were communicated to Quantstamp by the Coin Zoom team.

Code Documentation

The code is minimalistic and does not require extensive documentation.

Adherence to Best Practices

The code adheres to best practices.

Appendix

File Signatures

The following are the SHA-256 hashes of the audited contracts and/or test files. A smart contract or file with a different SHA-256 hash has been modified, intentionally or otherwise, after the audit. 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 audit.

Contracts

```
265d9ca19bd7dfa6e3b5db121a6fc84036cab856233f14fa5b99a4b1a0de3503
./contracts/CZToken.sol

8fb77dcc7448c8904bf92036ff4269d509c2804e1cb8d4ecc463b51de6b3665f
./contracts/Migrations.sol
```

Tests

```
f4acc8352117080aaf073ed6d5e50460a8ef5e06e08dbb90025f6be6d8e4858e
./test/c_z_token.js
```

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure smart contracts at scale using computer-aided reasoning tools, with a mission to help boost adoption of this exponentially growing technology.

Quantstamp's team boasts decades of combined experience in formal verification, static analysis, and software verification. Collectively, our individuals have over 500 Google scholar citations and numerous published papers. In its mission to proliferate development and adoption of blockchain applications, Quantstamp is also developing a new protocol for smart contract verification to help smart contract developers and projects worldwide to perform cost-effective smart contract security audits.

To date, Quantstamp has helped to secure hundreds of millions of dollars of transaction value in smart contracts and has assisted dozens of blockchain projects globally with its white glove security auditing 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.

Finally, Quantstamp's dedication to research and development in the form of collaborations with leading academic institutions such as National University of Singapore and MIT (Massachusetts Institute of Technology) reflects Quantstamp's commitment to enable world-class smart contract innovation.