Quantstamp Contract Security Certificate

April 15th 2019 — Quantstamp Verified

Hub Token

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

Executive Summary

Type	Token Audit
Auditors	Martin Derka, Senior Research Engineer Kacper Bąk, Senior Research Engineer Sebastian Banescu, Senior Research Engineer
Timeline	2019-01-22 through 2019-02-04
EVM	Byzantium
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	Repository: hub-ethereum-token; Commit: 14268d2

Total Issues	3 (2 Resolved)
High Risk Issues	0
Medium Risk Issues	0
Low Risk Issues	0
Informational Risk Issues	3 (2 Resolved)
Undetermined Risk Issues	0

3 issues

Overall Assessment

The implementation of the Hub Token relies on external libraries that are cloned to the project. This makes the implementation simple. However, cloning the libraries contradicts best practices for the smart contract development. As a result of this, the test coverage is inadequate. The tests suite also contains several constants that need to be manually initialized which makes execution of test coverage tools out of the box impossible. Overall, the implementation of the token can be deemed secure, but the project code quality and setup can be improved.

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

- 2019-01-23 - Initial report
- 2019-02-04 - Revised report based on commit `1ebc22a`

Quantstamp Audit Breakdown

This report focused on evaluating security of smart contracts, as requested by the Hub Token ERC20 team. Only the token smart contracts (along with their dependencies) were audited.

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*

Status: Fixed

Contract(s) affected: `HubToken.sol`, `BurnableToken.sol`, `StandardToken.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 `increaseApproval` and `decreaseApproval`.

The Quantstamp team recommends that the existence of `increaseApproval` and `decreaseApproval` be clearly communicated to the users. The Hub Token contract is still vulnerable to the double-spend exploit if these methods are not used.

Unlocked Pragma

Severity: *Informational*

Status: Fixed

Contract(s) affected: `HubToken.sol`, `BurnableToken.sol`, `StandardToken.sol`, `ERC20.sol`, `Pausable.sol`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. 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.

Clone-and-Own

Severity: *Informational*

Contract(s) affected: `HubToken.sol`, `BurnableToken.sol`, `StandardToken.sol`, `ERC20.sol`, `Pausable.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.

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. The Quantstamp team recommends that the OpenZeppelin libraries be used as managed dependencies instead of cloning contracts.

Test Results

Test Suite Results

```
  Contract: HubToken
     ✓ should transfer (131ms)
async () => {
            await instance.transfer(TEST_TARGET_WALLET_ADDRESS, 10);
        } failed as expected
async ()=> {
            await instance.transferFrom(HUB_TOKEN_OWNER_WALLET_ADDRESS, TEST_TARGET_WALLET_ADDRESS, 10, { from: PURSER_WALLET_ADDRESS })
        } failed as expected
async ()=> {
            await instance.burn(10);
        } failed as expected
async ()=> {
            await instance.burnFrom(HUB_TOKEN_OWNER_WALLET_ADDRESS, 10, { from: PURSER_WALLET_ADDRESS } );
        } failed as expected
     ✓ should pause transfers and burns (368ms)
     ✓ should approve allowance (113ms)
     ✓ should transferFrom (256ms)
     ✓ should burnFrom (196ms)
     ✓ should increase allowance (110ms)
     ✓ should decrease allowance (119ms)

  7 passing (1s)
```

Code Coverage

The code features low code coverage due to the cloned code.

```
------------------------|----------|----------|----------|----------|----------------|
File                    |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
------------------------|----------|----------|----------|----------|----------------|
 contracts/             |     7.06 |     4.55 |     7.41 |     6.93 |                |
    MultiSigWallet.sol    |     7.06 |     4.55 |     7.41 |     6.93 |... 389,390,391 |
 contracts/lifecycle/   |      100 |       75 |      100 |      100 |                |
    Pausable.sol          |      100 |       75 |      100 |      100 |                |
 contracts/math/        |     37.5 |    16.67 |       40 |     37.5 |                |
    SafeMath.sol          |     37.5 |    16.67 |       40 |     37.5 |... 32,35,63,64 |
 contracts/ownership/   |       25 |       25 |       40 |    33.33 |                |
    Ownable.sol           |       25 |       25 |       40 |    33.33 |... 52,60,61,62 |
 contracts/token/ERC20/ |     91.3 |       50 |    78.95 |       90 |                |
    BurnableToken.sol     |    33.33 |      100 |    33.33 |       50 |          19,28 |
    ERC20.sol             |      100 |      100 |      100 |      100 |                |
    HubToken.sol          |      100 |      100 |       80 |       80 |             27 |
    StandardToken.sol     |    95.12 |       50 |    90.91 |    95.12 |         27,153 |
------------------------|----------|----------|----------|----------|----------------|
All files               |    38.51 |    21.43 |    41.67 |     37.5 |                |
------------------------|----------|----------|----------|----------|----------------|
```

Automated Analyses

Oyente

Oyente did not detect any issues.

Mythril

Mythril detected potential integer overflow in method `add()` in the SafeMath library. This is a false positive.

MAIAN

Maian did not detect any issues.

Securify

Securify reported missing input validations in contracts `Ownable.sol` and `Pausable.sol`. The Quantstamp team verified that all the reports are benign or false positives.

Adherence to Specification

No specification was provided for this audit. The implemented token is a standard ERC20 token with pausability feature.

Code Documentation

The code is appropriately documented.

Adherence to Best Practices

With the exception of the aforementioned issues, the code adheres to best practices for Solidity.

Appendix

File Signatures

The following are the SHA-256 hashes of the audited contracts and 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

```
55a5b0e785e49777aac9361ab8fb22050b957fd816749519c5c1d7c3ec7dc98a
./contracts/token/ERC20/StandardToken.sol

18dde0e106decb6f4f11452a6de60cba1806e8b486c47b6ff1ebe5ee02d4df5c
./contracts/token/ERC20/ERC20.sol

ea492bb295be274e4e1c86f4538b3efcde6aa0e2993c61ecf308e9fd34504f24
./contracts/token/ERC20/HubToken.sol

b9cf9686555738975f7ed256c8617a3b8535b64dcb4c7b50c1779bb88fdbc15e
./contracts/token/ERC20/BurnableToken.sol

3279980b962a1374a710b936a067a5262ad2c3ea3778e2380989f0e39928baf4
./contracts/MultiSigWallet.sol

6037cc1d2a8adff9501b652df2689b95f4435d9cd745a030f8a93527872ac2a9
./contracts/lifecycle/Pausable.sol

45c062a2e7039e75c47b128c592a587e554b85aed97d52ffd9e5382219c3fe1b
./contracts/Migrations.sol

ed979bdd492b210048e0539357e413c0eaff9df06e588658aeab0266eaa538b1
./contracts/math/SafeMath.sol

7a088b95d0b62f96edbb1d9a4a9161a507a1a543510bb2bb043dbc002002970f
./contracts/ownership/Ownable.sol
```

Tests

```
d39f16c48dde8e24cfa4741a5f2f6b9e258988abddeecc2f5f2e25f95e211c37
./test/test.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.