Quantstamp Contract Security Certificate

October 15th 2019 — Quantstamp Verified

PlusCoin Token

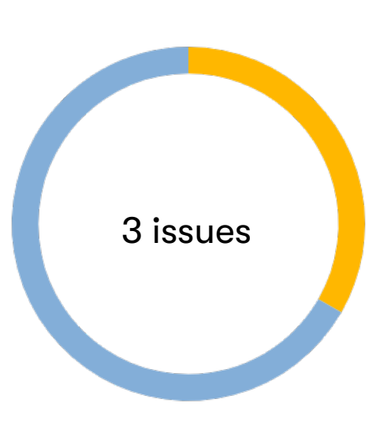
This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Token Contract Audit
Auditors	Yohei Oka, Forward Deployed Engineer John Bender, Senior Research Engineer Michael Teixeira, Software Developer Martin Derka, Senior Research Engineer
Timeline	2018-09-28 through 2018-10-02
Languages	Solidity, Javascript
Methods Specification	PlusCoin Whitepaper
Source Code	Repository: NPLC — Commit: 12fbe86

Repository	Commit
NPLC	12fbe86

Total Issues	3	(0 Resolved)
High Risk Issues	0	
Medium Risk Issues	0	
Low Risk Issues	1	(0 Resolved)
Informational Risk Issues	2	(0 Resolved)
Undetermined Risk Issues	0	



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

- 2018-10-09 - First Version

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the PlusCoin Token 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

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. Installed the Oyente tool from Docker: `docker pull luongnguyen/oyente`
9. Migrated files into Oyente (root directory): `docker run -v $(pwd):/tmp - it luongnguyen/oyente`
10. Ran the Oyente tool on each contract: `cd /oyente/oyente && python oyente.py /tmp/path/to/contract`
11. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
12. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`

Assessment

Findings

Centralization of Power

Severity: *Low*

Description: The token contract features the following centralization characteristic:

- The owner can enable and disable transfer-related functionalities at will through the use of the `isNotFrozen` and `isNotFrozenFrom` modifier.

While not necessarily a vulnerability, if the contract owner's private key is compromised or the owner becomes malicious, then token holders may be unable to transfer their tokens. This issue is magnified in that it can be performed by all super admins, not only the contract owner. In the spirit of decentralization, we recommend taking away these functionalities.

The PlusCoin team has communicated to Quantstamp that the ability to freeze the token and specific accounts is implemented for AML purposes. This should also be communicated clearly to token holders upfront.

Allowance Double-Spend Exploit

Severity: *Informational*

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.

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.

Unused Event

Severity: *Informational*

Description: The event `Burn` (`PlusCoin.sol` L12) doesn't appear to be used anywhere and should be removed.

Test Results

Test Suite Results

```
Contract: Administratable
  addSuperAdmin
    ✓ contract owner can add a super Admin (45260 gas)
    ✓ a super admin can not add another super admin (23292 gas)
    ✓ a non super admin cannot add another super admin (23292 gas)
    ✓ cannot add a super admin that is already a super admin (68994 gas)
  removeSuperAdmin
    ✓ contract owner can remove a super Admin (60407 gas)
    ✓ a super admin can not remove another super admin (184329 gas)
    ✓ a non super admin cannot add another super admin (23292 gas)
    ✓ cannot add a super admin that is already a super admin (68994 gas)

Contract: Freezable
  freezeAccount
    ✓ contract owner can freeze token (66246 gas)
    ✓ non owner can not freeze token (22379 gas)
    ✓ contract owner can add a frozen account (45912 gas)
    ✓ a non super admin can not freeze another account (23952 gas)
    ✓ cannot add a frozen account that is already a frozen account (69994 gas)

Contract: PlusCoin
  constructor
    ✓ validate token minting
  transfer
    ✓ simple transfer case should succeed and change balance (52961 gas)
    ✓ transferFrom case should succeed and change balance (144945 gas)
    ✓ transfer fails when token is frozen (68331 gas)
    ✓ transfer fails if sender is frozen (70829 gas)
    ✓ transfer fails if destination is frozen (71163 gas)
    ✓ transfer fails if destination is not a valid address (22256 gas)
    ✓ when the spender is the frozen account (70061 gas)
    ✓ when the spender is the frozen token (92521 gas)
    ✓ when the spender is the frozen account(message sender) (95744 gas)
    ✓ increase allowance (78722 gas)
    ✓ decrease allowance (78986 gas)
    ✓ change owner and add super admin (115350 gas)
    ✓ try to drain contract from admin address (27765 gas)
```

Gas					Block limit: 6721975 gas	
Methods		5 gwei/gas			257610.97 krw/eth	
Contract	Method	Min	Max	Avg	# calls	krw (avg)
Administratable	addSuperAdmin	-	-	45260	6	58.30
Administratable	removeSuperAdmin	-	-	15147	1	19.51
Freezable	freezeAccount	-	-	45912	2	59.14
Freezable	freezeToken	-	-	43917	1	56.57
PlusCoin	addSuperAdmin	-	-	45590	1	58.72
PlusCoin	approve	46381	46445	46402	3	59.77
PlusCoin	decreaseAllowance	-	-	32605	1	42.00
PlusCoin	freezeAccount	46220	46550	46385	4	59.75
PlusCoin	freezeToken	44093	44423	44258	2	57.01
PlusCoin	increaseAllowance	-	-	32341	1	41.66
PlusCoin	removeSuperAdmin	-	-	15301	1	19.71
PlusCoin	transfer	-	-	52961	2	68.22
PlusCoin	transferFrom	-	-	45539	1	58.66
PlusCoin	transferOwnership	-	-	30837	1	39.72
Deployments					% of limit	
Administratable		-	-	615857	9.2 %	793.26
Freezable		-	-	930766	13.8 %	1198.88
PlusCoin		3120608	3120672	3120619	46.4 %	4019.53

```
27 passing (11s)
```

Code Coverage

The PlusCoin repository has very high test coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
Administratable.sol	100	100	100	100	
Freezable.sol	100	100	100	100	
PlusCoin.sol	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

Automated Analyses

Oyente

Symbolic execution (the Oyente tool) did not detect any vulnerabilities of types Parity Multisig Bug 2, Transaction-Ordering Dependence (TOD), Callstack Depth Attack, Timestamp Dependency, Re-Entrancy Vulnerability, and Integer Over/Underflow.

Mythril

The Mythril tool detects defects such as Integer underflow, Unprotected functions, Missing check on call return value, Re-entrancy, Multiple sends in a single transaction, External call to untrusted contract, `delegatecall` or `callcode` to untrusted contract, Timestamp dependence, Use of `tx.origin`, Predictable RNG, Transaction order dependence, Use of `assert()` instead of `require()`, Use of deprecated functions, Detect tautologies. The tool found a potential integer underflow issue in the `add()` method of `SafeMath.sol`, but Quantstamp determined that it was a false positive. Mythril did not find any other issues.

MAIAN

The MAIAIN tool detects three kinds of bad contracts, Greedy, Prodigal and Suicidal. It found no defects.

Adherence to Specification

With minimal written specification we were unable to judge to what degree the code conforms to the specification. Private conversation with the PlusCoin team convinced us that the contract code implements the desired functionality within the context of its intended usage.

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

```
105236f1f4649fdfddbe7aa4857cc8466c2406e6ea30b668836e0f8f88f2cd27
./contracts/Administratable.sol

2cf21015089404778ecb5a4ad22a3bfb7dac64709ec8444ab2d6bd5c4d3e7ab7
./contracts/Freezable.sol

700c0904cfbc20dba65f774f54a476803a624372cc95d926b3eba9b4d0f0312e
./contracts/Migrations.sol

7a60bbce9324841a5534206bf2c0fe998c211e71fedb161c16889c51058275fe
./contracts/PlusCoin.sol
```

Tests

```
42b4e37d142c1232023e30efaa659f6785019704fce61022a29fcd782cccbded
./test/Administratable.test.js

5dc8653aa162669f1f8651ac25effd79c96b06814a10297cdf0cc6dcb32dba0e
./test/Freezable.test.js

6f222918c590e3f2ced22b752649aebfe7819c412b02c0db2954077ef3cb14c3
./test/PlusCoin.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.