Quantstamp Security Assessment Certificate

September 28th 2020 — Quantstamp Verified

PowerTrade

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

Executive Summary

Type	Audit
Auditors	Kevin Feng, Blockchain Researcher Joseph Xu, Technical R&D Advisor Luís Fernando Schultz Xavier da Silveira, Security Consultant
Timeline	2020-09-16 through 2020-09-23
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	PowerTrade Fuel Token Offical Token Paper PowerTrade Fuel (PTF) Google Doc
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
fuel-dao	200947a
fuel-dao	70c6424

Total Issues	**10** (0 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**8** (0 Resolved)
Informational Risk Issues	**2** (0 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

The scope of this audit is the diff between FuelToken contracts and their base contracts, and the Vesting smart contracts. During this audit, Quantstamp have identified 8 low-severity, and 2 informational issues. Additionally, Quantstamp has also recommended 2 suggestions regarding the implementation of the ERC20 specification, 6 suggestions regarding documentation, and 5 suggestions regarding best practices. Overall, the code is well-structured, well-documented and decently tested. However, there are several security considerations that need to be addressed before the live deployment. We recommend addressing these issues before going into production.

ID	Description	Severity	Status
QSP-1	Use of `uint32` for Block Number and Checkpoints	Low	Unresolved
QSP-2	Clone-and-Own	Low	Unresolved
QSP-3	Updating Beneficiary Reduces Its Rate of Receiving Tokens	Low	Unresolved
QSP-4	Time Unit Changed in `Timelock.sol` When Compared to Compound Contract	Low	Unresolved
QSP-5	Lack of Timelock When Changing Admin Addresses	Low	Unresolved
QSP-6	Updating Beneficiary before Vesting Cliff Will Revert	Low	Unresolved
QSP-7	Privileged Roles and Ownership	Low	Unresolved
QSP-8	Missing Input Validation	Low	Unresolved
QSP-9	Unlocked Pragma	Informational	Unresolved
QSP-10	Non-zero Balances Shown for Voided Beneficiaries	Informational	Unresolved

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 v0.6.6
- Mythril v0.2.7

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth analyze <path/to/contract>`

Findings

QSP-1 Use of `uint32` for Block Number and Checkpoints

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `FuelToken.sol`

Description: In `FuelToken.sol`, the use of `uint32` for `BlockNumber` and `numCheckpoints` could conceivably result in the token's lockdown in the future as these numbers approach their max limits.

Recommendation: It is recommended to change `uint32` to `uint128` for accommodating a larger max value as these variables increment in the future.

QSP-2 Clone-and-Own

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `CloneFactory.sol`, `IERC20.sol`, `ReentrancyGuard.sol`, `SafeMath.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.
Based on the scope of this audit, the following files can instead be imported from libraries:

- `CloneFactory.sol`: Optionality
- `IERC20.sol`: OpenZeppelin
- `ReentrancyGuard.sol`: OpenZeppelin
- `SafeMath.sol`: OpenZeppelin

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.

QSP-3 Updating Beneficiary Reduces Its Rate of Receiving Tokens

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `VestingContract.sol`

Description: In `VestingContract.sol`, the rate of vesting tokens that the beneficary receive when calling `_drawDown()` is calculated based on the `amount` parameter in the beneficiary's schedule. When updating the beneficiary with the function `updateScheduleBeneficiary()`, the `amount` property for the new beneficiary's schedule is decreased. This means that down the line, the new beneficiary would receive the tokens at a slower rate than the original beneficiary.

Recommendation: Check if this is the intended behavior as it is not certain from the specification. If not, change `L154` to `amount : schedule.amount` and add `totalDrawn[_newBeneficiary] = totalDrawn[_currentBeneficiary]`after `L156`. This suggestion is the minimal needed to keep the same rate of vesting for the new beneficiary. Additional modifications may be needed to return accurate vesting information to the new beneficiary depending on the specification.

QSP-4 Time Unit Changed in `Timelock.sol` When Compared to Compound Contract

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Timelock.sol`

Description: The contract `Timelock.sol` has been modified to handle `delay` and `gracePeriod` as `uint` seconds while the original Compound contract explicitly enforces limits to these parameters in time units of days. This change may introduce the risk of having a unusually short delay and grace period through manual error when calling `setDelay()` or `setGracePeriod()`.

Recommendation: It is recommended to explicitly enforce time parameters in a similar manner to the original contract, possibly with changes to `hours` instead of `days` if desired.

QSP-5 Lack of Timelock When Changing Admin Addresses

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `Timelock.sol`, `Governor.sol`

Description: The contracts `Timelock.sol` and `Governor.sol` have been modified to allow updates to `pendingAdmin` without being subjected to the timelock by the `admin`. This also means that the `admin` role in `Timelock.sol` can be assigned to a new address without the timelock.

Recommendation: The lack of timelock feature on `pendingAdmin` update and additional `admin` privilege when compared to the original Compound contracts should be disclosed in user-facing documentations.

QSP-6 Updating Beneficiary before Vesting Cliff Will Revert

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `VestingContract.sol`

Description: In the contract `VestingContract.sol`, beneficiaries currently cannot be updated through `updateScheduleBeneficiary()` before the vesting cliff as the function's call to `_drawDown()` will revert.

Recommendation: Check to see if this is the intended behavior since it is unclear from the specification. If so, it is recommended to add a `require` statement to check for this behavior in `updateScheduleBeneficiary()` directly instead of having it revert at `_drawdown()`.

QSP-7 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `FuelToken.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. However, 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. In `FuelToken.sol`, the minter role has the power to arbitrary mint tokens for themselves or for potential malicious users.

Recommendation: This privileged role should be clearly documented in user-facing documentations.

QSP-8 Missing Input Validation

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `FuelToken.sol`, `VestingContract.sol`

Description: The following inputs/return values, if unvalidated, may create undesirable effects for the smart contracts:

- `FuelToken.sol` may permenantly lose its minter if the current minter accidently sets the new minter as the zero address with `changeMinter()`
- In `VestingContract.sol`, beneficiaries may be updated to address `0x0` through `updateScheduleBeneficiary()`
- In `VestingContract.sol`, beneficiary schedules may also be overridden through `updateScheduleBeneficiary()`if the address of the `_newBeneficiary` already exists in `vestingSchedule`

Recommendation: Make the following changes to the respective issues in question:

- If the developer does not plan to permanently burn the ability to mint tokens, `changeMinter()` in `FuelToken.sol` should check that `account` is non-zero
- `updateScheduleBeneficiary()` in `VestingContract.sol` should check that `_newBeneficiary` is non-zero
- `updateScheduleBeneficiary()` in `VestingContract.sol` should also check if `vestingSchedule[_newBeneficiary].amount == 0`

QSP-9 Unlocked Pragma

Severity: *Informational*

Status: Unresolved

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.5.*`. 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.

QSP-10 Non-zero Balances Shown for Voided Beneficiaries

Severity: *Informational*

Status: Unresolved

File(s) affected: `VestingContract.sol`

Description: In `VestingContract.sol`, a voided beneficiary can still see non-zero values when calling `tokenBalance()` and `remainingBalance()`. Similary, information about the vesting schedule of a voided beneficiary can still be seen through `vestingScheduleForBeneficiary()`.

Recommendation: Add a check to see if the beneficiary has been voided. If so, return 0 for both `tokenBalance()`and `remainingBalance()`. Similarly, return an extra value for `vestingScheduleForBeneficiary()` indicating whether or not the beneficiary's schedule is voided.

Automated Analyses

Slither

After filtering out the false positives and previously mentioned issues Slither did not detect any issues.

Mythril

After filtering out the false positives and previously mentioned issues Mythril did not detect any issues.

Adherence to Specification

The smart contracts in scope adheres to the token paper and Google document that is provided for the context of this audit. However, regarding the ERC20 standards in the implementation of these contracts, Quantstamp has the following observations:

1. In `transferFrom()` of `FuelToken.sol`: ERC20 should not allow for an `Approval` event on `transferFrom()`
2. In `transferFrom()` of `FuelToken.sol`: ERC20 requires allowances to be considered even if the owner of the tokens is the same address as the spender.

Code Documentation

Quantstamp have several observations and suggestions on improving documentation:

1. The function `tokenBalance()` of `VestingContractWithoutDelegation.sol` is documented to return the beneficiary's vested token balance but instead returns the total escrowed amount in the contract
2. The function `delegate()` in `FuelToken.sol` has a return value but there is no `@return` documented in developer documentation
3. The function `delegateBySig()` in `FuelToken.sol` has a return value but there is no `@return` documented in developer documentation
4. `VestingContractWithoutDelegation.sol`, `VestingDepositAccount.sol` has many references to `wei` in their documentation even though the currency is not in ETH
5. In `VestingDepositAccount.sol`, the function `init` is documented to be "only controller" but in actuality, it can be called by anyone
6. In `L121` to `L138` of `VestingContractWithoutDelegation.sol`, the documentation is inconsistent; beneficiary is a parameter and not `msg.sender`.

Adherence to Best Practices

Quantstamp have observations and suggestions on improving best practices:

1. In `L33` of `Timelock.sol`, the error message for the `gracePeriod_` check in the constructor is incorrect
2. In `L308` of `Timelock.sol`, the error message should say "amount underflows" instead of "amount overflows"
3. The constructor for `FuelToken.sol` should have `require` statements to check that `account` and `minter` are non-zero addresses
4. The function `init` of `VestingDepositAccount.sol` should have `require` statements to check that `_controller` and `_beneficiary` are non-zero addresses
5. In the function `createVestingSchedules()` of `VestingContractWithoutDelegation.sol`, the declaration of `result` is unnecessary as the function will always return `true` on completion

Test Results

Test Suite Results

All tests have passed.

```
Contract: VestingContract
   ✓ should return token and baseDepositAccount address (517ms)
   ✓ reverts when trying to create the contract with zero address token (160ms)
  reverts
    createVestingSchedule() reverts when
       ✓ Not the owner (97ms)
       ✓ specifying a zero address beneficiary (75ms)
       ✓ specifying a zero vesting amount (52ms)
       ✓ trying to overwrite an inflight schedule (191ms)
       ✓ trying to update schedule beneficiary when not owner (253ms)
    drawDown() reverts when
       ✓ no schedule is in flight (65ms)
       ✓ a beneficiary has no remaining balance (823ms)
       ✓ the allowance for a particular second has been exceeded (two calls in the same block scenario) (475ms)
  single schedule - incomplete draw down
     ✓ beneficiary 1 balance should equal vested tokens when called direct
     ✓ vesting contract balance should equal vested tokens when proxied through (139ms)
     ✓ should be able to get my balance
     ✓ correctly emits ScheduleCreated log
     ✓ vestingScheduleForBeneficiary()
     ✓ lastDrawDown()
     ✓ validateAvailableDrawDownAmount() (84ms)
    single drawn down
       ✓ should emit DrawDown events
       ✓ should move tokens to beneficiary
       ✓ should reduce validateAvailableDrawDownAmount() (40ms)
       ✓ should update vestingScheduleForBeneficiary() (99ms)
    completes drawn down in several attempts
      after 1 day
         ✓ some tokens issued (198ms)
        after 5 day - half tokens issues
           ✓ more tokens issued (249ms)
          after 11 days - over schedule - all remaining tokens issues
             ✓ all tokens issued (99ms)
  single schedule - future start date
     ✓ lastDrawnAt()
     ✓ vestingScheduleForBeneficiary()
     ✓ remainingBalance()
     ✓ validateAvailableDrawDownAmount() is zero as not started yet
  single schedule - starts now - full draw after end date
     ✓ should draw down full amount in one call (203ms)
  single schedule - future start - completes on time - attempts to withdraw after completed
    After all time has passed and all tokens claimed
       ✓ mo more tokens left to claim (48ms)
       ✓ draw down rates show correct values (68ms)
       ✓ no further tokens can be drawn down (66ms)
  single schedule - update beneficary
```

```
    after 6 days you can update the beneficiary
      ✓ should have draw down to original beneficiary and transferred (503ms)
  Schedules with cliff durations
    ✓ Should return 0 for amount available during the cliff period
    should allow full draw after end date
      ✓ should draw down full amount in one call (167ms)
Contract: VestingContractWithoutDelegation
  ✓ should return token address
  reverts
    contract creation reverts when
      ✓ trying to create the contract with zero address token (65ms)
      ✓ trying to create the contract where the end date is before the start (71ms)
    createVestingSchedule() reverts when
      ✓ Not the owner
      ✓ specifying a zero address beneficiary
      ✓ specifying a zero vesting amount
      ✓ trying to overwrite an inflight schedule (53ms)
    drawDown() reverts when
      ✓ no schedule is in flight (38ms)
      ✓ a beneficiary has no remaining balance (202ms)
      ✓ the allowance for a particular second has been exceeded (two calls in the same block scenario) (208ms)
  single schedule - incomplete draw down
    ✓ vesting contract balance should equal vested tokens
    ✓ should be able to get my balance
    ✓ correctly emits ScheduleCreated log
    ✓ vestingScheduleForBeneficiary()
    ✓ lastDrawDown()
    ✓ validateAvailableDrawDownAmount()
    single drawn down
      ✓ should emit DrawDown events
      ✓ should move tokens to beneficiary
      ✓ should reduce validateAvailableDrawDownAmount()
      ✓ should update vestingScheduleForBeneficiary()
    completes drawn down in several attempts
      after 1 day
        ✓ some tokens issued (103ms)
       after 5 day - half tokens issues
          ✓ more tokens issued (109ms)
         after 11 days - over schedule - all remaining tokens issues
            ✓ all tokens issued (54ms)
  single schedule - future start date
    ✓ lastDrawnAt()
    ✓ vestingScheduleForBeneficiary()
    ✓ remainingBalance()
    ✓ validateAvailableDrawDownAmount() is zero as not started yet
  single schedule - starts now - full draw after end date
    ✓ should draw down full amount in one call (128ms)
  single schedule - future start - completes on time - attempts to withdraw after completed
    After all time has passed and all tokens claimed
      ✓ mo more tokens left to claim (52ms)
      ✓ draw down rates show correct values (41ms)
      ✓ no further tokens can be drawn down (66ms)
  setting up multiple schedules in a single transaction
    ✓ reverts when not owner (53ms)
    ✓ reverts when the arrays are empty
    ✓ reverts when the arrays are of differing lengths
    ✓ sets up multiple schedules successfully (101ms)
  Schedules with cliff durations
    ✓ Should return 0 for amount available during the cliff period
    should allow full draw after end date
      ✓ should draw down full amount in one call (104ms)
  VestingContract
    ✓ returns zero for empty vesting schedule
Contract: VestingDepositAccount
  ✓ should return token address, controller, and beneficary (59ms)
  reverts
    ✓ when init called twice (101ms)
    ✓ when transferToBeneficiary not called by controller (54ms)
    ✓ when switchBeneficiary not called by controller (38ms)
77 passing (42s)
```

Code Coverage

Overall, the test suite has decent coverage across the files in scope. However, it is not perfect. Most notably, the coverage results for `VestingContract.sol` can be improved by adding test cases for `L255` and `L263`. Likewise, coverage for `VestingContractWithoutDelegation.sol` can be improved by adding test cases for `L207`. Moreover, since `Timelock.sol` seems to have been modified when compared to the original contract from Compound, we strongly advise adding tests to increase the coverage score for it.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	48.39	36.64	52.17	49	
CloneFactory.sol	100	100	100	100	
FuelToken.sol	55.36	34.78	60	54.87	… 367,368,369
Governor.sol	0	0	0	0	… 320,321,322
IERC20.sol	100	100	100	100	
ReentrancyGuard.sol	100	50	100	100	
SafeMath.sol	59.26	33.33	60	59.26	… 168,183,184
Timelock.sol	0	0	0	0	… 115,117,122
VestingContract.sol	96.43	79.41	90.91	96.43	255,263
VestingContractWithoutDelegation.sol	97.96	90.63	91.67	97.96	207
VestingDepositAccount.sol	100	100	100	100	
contracts/mock/	87.5	75	100	87.5	
VestingContractWithFixedTime.sol	75	50	100	75	30
VestingContractWithoutDelegationFixedTime.sol	100	100	100	100	
All files	**49.15**	**37.29**	**55.1**	**49.76**	

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

```
f847d6934da197243b051ecbe8a994014a78539bbdaa28621bfafd7a18925454 ./contracts/CloneFactory.sol
9f2f286aa855a2357745c747b868dc19c35f70286761f4fef57fb7ff993e3f43 ./contracts/FuelToken.sol
ccfb98a84f139f2bcc4ec89ed9de163c787482748b516e13500cb582b323aaf0 ./contracts/Governor.sol
f9dd3bef4149ff4d7ea1ba1667341aef422b23bb9b6415e1c65a89944a91769b ./contracts/IERC20.sol
a19e492ee19ed4289476b6a192572692a3911a176567bd6c90a40d686966ffed ./contracts/ReentrancyGuard.sol
7a44af6e18146a12c4e6e59acd4499ebefbd127182356f1e01fe84ca93339043 ./contracts/SafeMath.sol
3e84c7a6251d127eefc3186596d867fbb7581039c64614fbbdb23bf872cb6901 ./contracts/Timelock.sol
635c38313747d05c95bdc29f78b780745aa65dbbc2ee709530994f85d9bc6311 ./contracts/VestingContract.sol
abbcea3c090f72e9360a24b3001a39edb2daea32e90a48b6d938a612f20790fd ./contracts/VestingContractWithoutDelegation.sol
b27cfe449ab59508c279e8e22fbc5ac0e497bc138504454d35b7576b7410599a ./contracts/VestingDepositAccount.sol
```

Changelog

- 2020-09-23 - Initial report

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.