Quantstamp Security Assessment Certificate

October 6th 2021 — Quantstamp Verified

Cryptex

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

Executive Summary

Type	Staking Function
Auditors	Kacper Bąk, Senior Research Engineer Fayçal Lalidji, Security Auditor Cristiano Silva, Research Engineer
Timeline	2021-09-16 through 2021-10-06
EVM	London
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
governance-staking	76961ae

Total Issues	**6** (4 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**1** (0 Resolved)
Low Risk Issues	**3** (2 Resolved)
Informational Risk Issues	**2** (2 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

During the review we have found a few issues. Notably, one is of medium severity. Furthermore, although the code contains inline comments, it is missing a proper documentation stating functional and non-functional requirements.
Update: the team has addressed all of the issues as of commit `6343d69`.

ID	Description	Severity	Status
QSP-1	Incorrect requirement in `notifyRewardAmount()`	Medium	Acknowledged
QSP-2	No checks if addresses are non-zero	Low	Fixed
QSP-3	Multiple invocations of `stake()`	Low	Acknowledged
QSP-4	Missing validation of the number of decimals for ERC20 tokens	Low	Fixed
QSP-5	Gas optimization	Informational	Fixed
QSP-6	Withdrawal logic requires users to save all their delegators addresses	Informational	Mitigated

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 v0.8.0

Steps taken to run the tools:

Installed the Slither tool: `pip install slither-analyzer` Run Slither from the project directory: `slither .`

Findings

QSP-1 Incorrect requirement in `notifyRewardAmount()`

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `DelegatorFactory.sol`

Description: The requirement in `notifyRewardAmount()` can be invalid if part of the users do not withdraw their rewards, meaning that the entity in charge of sending the new reward tokens might send less than expected and the requirement will still execute without throwing.

Recommendation: Use `ERC20.transferFrom()` instead of checking the contract token balance.

Update: the team informed us that `notifyRewardAmount()` can only be called by the owner which will be the core team multi-signature wallet or the DAO; not calling `transferFrom()` allows the contract to receive funds from different addresses and give the team a way to prevent `rewardsTokens` being sent by users by mistake to be burned.

QSP-2 No checks if addresses are non-zero

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Delegator.sol`, `DelegatorFactory.sol`

Description: The functions `Delegator.constructor()` and `DelegatorFactory.constructor()` do not check if arguments of type address are non-zero.

Recommendation: Add relevant checks.

QSP-3 Multiple invocations of `stake()`

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `DelegatorFactory.sol`

Description: If `stake()` is called multiple times for the same `delegator_`, user may be unable to withdraw previous stakes before the last `block.timestamp + waitTime`.

Recommendation: Unless this is intentional, timestamp each stake operation individually so that withdrawals of previous stakes are possible. Otherwise, inform users and document this behavior.

Update: the team informed us that besides acknowledging the issue they provided a mitigation strategy by updating the documentation and the frontend to give a warning to the users before staking happens.

QSP-4 Missing validation of the number of decimals for ERC20 tokens

Severity: *Low Risk*

Status: Fixed

File(s) affected: `DelegatorFactory.sol`

Description: The contract assumes tokens with 18 decimals. Based on the provided code, it is unclear whether this assumption holds.

Recommendation: Ensure that the used tokens have 18 decimals; otherwise adjust the code accordingly.

QSP-5 Gas optimization

Severity: *Informational*

Status: Fixed

File(s) affected: `DelegatorFactory.sol`

Description: `updateReward()` calls `rewardPerToken()` twice, in the function body and when executing `earned()`, we recommend to use the state variable `rewardPerTokenStored` inside `earned()` instead of calling the `rewardPerToken()`.

Recommendation: If a public view function is needed to get the latest reward value off-chain, another function can be added which will optimize the gas consumption.

QSP-6 Withdrawal logic requires users to save all their delegators addresses

Severity: *Informational*

Status: Mitigated

File(s) affected: `DelegatorFactory.sol`

Description: Every time a user stakes, they are free to change the delegator contract address, meaning that when withdrawing a user might be forced to do multiple withdrawals if a single delegator does not contain a sufficient balance.

Recommendation: Inform users of this possibility, and, perhaps, recommend sticking to a single delegator.

Update: The team informed us that they created a graph using The Graph to save all the information about the address and amounts to display on the front end. They will also recommend in the Frontend and community channels to stakers to stick to a single Delegator.

Automated Analyses

Slither

Slither reported the following:

1. Ignored return values in
    1. `src/Delegator.sol#59`,
    2. `src/DelegatorFactory.sol#198`,
    3. `src/DelegatorFactory.sol#241`.
    We recommend adding relevant checks. **Update:** fixed.

2. Multiplication performed before division in `DelegatorFactory.notifyRewardAmount()`, however, it is a false positive.
3. Reentrancy in `DelegatorFactory.createDelegator()` due to L214, however, it is a false positive.
4. Reentrancy in `DelegatorFactory.notifyRewardAmount()` due to the call in L164 and then variable updates in L170 and 171. We recommend rearranging the statements.

Code Documentation

Although there are inline comments, there is no documentation that would describe functional and non-functional requirements.

Adherence to Best Practices

1. `Delegator` and `DelegatorFactory` inherit from `DSTest` but this code is never used. Remove this dependency. **Update:** fixed.
2. `SafeMath` is not needed in Solidity 8. **Update:** fixed.
3. The modifier `DelegatorFactory.updateReward()` modifies state. Typically modifiers are used for checks. **Update:** fixed.

Test Results

Test Suite Results

All tests passed.

```
Running 10 tests for src/tests/Delegator.t.sol:DelegatorTest
+++ OK, passed 100 tests.
[PASS] testFail_stake_notOwner(uint256) (runs: 100)
+++ OK, passed 100 tests.
[PASS] testFail_removeStake_notEnoughBalance(uint256) (runs: 100)
[PASS] testFail_invalidTokenDecimals() (gas: 114069)
[PASS] test_removeStake() (gas: 141817)
[PASS] test_parameters() (gas: 6071)
[PASS] testFail_invalidToken() (gas: 57376)
+++ OK, passed 100 tests.
[PASS] test_stake(address,uint256) (runs: 100)
[PASS] testFail_invalidDelegatee() (gas: 57272)
+++ OK, passed 100 tests.
[PASS] testFail_removeStake_notOwner(uint256) (runs: 100)
+++ OK, passed 100 tests.
[PASS] test_removeStakeFuzz(address,uint256) (runs: 100)

Running 34 tests for src/tests/DelegatorFactory.t.sol:DelegatorFactoryTest
+++ OK, passed 100 tests.
[PASS] test_updateWaitTimeFuzz(uint256) (runs: 100)
[PASS] testFail_invalidTimelock() (gas: 108141)
[PASS] test_earnRewards() (gas: 1098655)
[PASS] test_delegate() (gas: 807164)
+++ OK, passed 100 tests.
[PASS] testFail_setRewardsDuration_rewardsNotComplete(uint256) (runs: 100)
[PASS] testFail_invalidRemoveDelegator() (gas: 75495)
[PASS] test_setRewardsDuration() (gas: 4803)
[PASS] testFail_invalidCreateDelegator() (gas: 1232)
[PASS] testFail_invalidRemoveAmount() (gas: 587719)
[PASS] test_notifyRewards() (gas: 92699)
[PASS] test_parameters() (gas: 6018)
[PASS] test_unDelegate() (gas: 820958)
[PASS] testFail_invalidStakingTokenDecimals() (gas: 164896)
+++ OK, passed 100 tests.
[PASS] test_notifyRewardsFuzz(uint256) (runs: 100)
+++ OK, passed 100 tests.
[PASS] test_delegateFuzz(address,uint256) (runs: 100)
+++ OK, passed 100 tests.
[PASS] test_createDelegator(address) (runs: 100)
+++ OK, passed 100 tests.
[PASS] testFail_invalidUnDelegateAmount(address,uint256) (runs: 100)
+++ OK, passed 100 tests.
[PASS] test_unDelegateSpecific(address,uint256,uint256) (runs: 100)
[PASS] testFail_notifyRewards_notOwner() (gas: 2028)
[PASS] testFail_setRewardsDuration_notOwner() (gas: 2072)
+++ OK, passed 100 tests.
[PASS] testFail_updateWaitTimeNotAdmin(uint256) (runs: 100)
[PASS] test_updateWaitTime() (gas: 4638)
+++ OK, passed 100 tests.
[PASS] test_unDelegateFuzz(address,uint256) (runs: 100)
[PASS] testFail_invalidAmount() (gas: 587224)
[PASS] testFail_invalidStakingToken() (gas: 108096)
+++ OK, passed 100 tests.
[PASS] test_multipleDelegators(uint256,uint256) (runs: 100)
[PASS] testFail_invalidRewardTokenDecimals() (gas: 165684)
[PASS] testFail_invalidDelegator() (gas: 133161)
+++ OK, passed 100 tests.
[PASS] test_setRewardsDurationFuzz(uint256) (runs: 100)
[PASS] testFail_notifyRewards_rewardToHigh() (gas: 52019)
[PASS] testFail_invalidRewardToken() (gas: 108109)
+++ OK, passed 100 tests.
[PASS] testFail_unDelegateNoWait(address,uint256) (runs: 100)
+++ OK, passed 100 tests.
[PASS] testFail_createDelegator(address) (runs: 100)
+++ OK, passed 100 tests.
[PASS] test_moveDelegation(uint256,uint256) (runs: 100)

✓  Done in 16.26s.
```

Code Coverage

According to `dapp test --coverage` coverage is missing in the following:

- `Delegator.removeStake()`, **Update:** fixed.
- `DelegatorFactory.notifyRewardAmount()`, **Update:** fixed.
- `DelegatorFactory.setRewardsDuration()`, **Update:** fixed.
- `DelegatorFactory.withdraw()`, **Update:** fixed.
- `DelegatorFactory.updateWaitTime()`, **Update:** fixed
- `DelegatorFactory.getRewardForDuration()`.

Since the codebase is rather small, we recommend covering all statements.

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

```
04afcfbd19ecb41a12b27ba32c9f364788672ebde6ac5530ad156a0aca4510e8 ./IGovernanceToken.sol
db3ab3ddd126d70da61a71ed26f208d0b36dcd5fadec88b40cf299702181389d ./Delegator.sol
ce06bac7ee0e6c48f986785a07623a7751c955be2402500e47a1b1ebb28a61f2 ./DelegatorFactory.sol
```

Tests

```
6389049058381d33469e70132905e607e2ba2d3cc355c576da7ece403168f180 ./tests/hevm.sol
2d4b3bbbcc61211c4203d451298de2fa06f5c230bf93e3dfa67bb95c23d84014 ./tests/Delegator.t.sol
551b9514896898808b877c9b53db8d6d8ce16cb0e55ee4fbbaef573d67364d43 ./tests/DelegatorFactory.t.sol
```

Changelog

- 2021-09-17 - Initial report
- 2021-10-06 - Reaudit based on commit `6343d69`

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.