Quantstamp Security Assessment Certificate

December 10th 2020 — Quantstamp Verified

StakeHound

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

Executive Summary

Type	Token contract
Auditors	Fayçal Lalidji, Security Auditor Kevin Feng, Blockchain Researcher Luís Fernando Schultz Xavier da Silveira, Security Consultant
Timeline	2020-09-30 through 2020-10-02
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	litepaper
Documentation Quality	Medium
Test Quality	Medium
Source Code	

Repository	Commit
stakehound-core	0f1d6e4

Goals

- Can an attacker steal users' funds?
- Is there any rounding or truncation errors?

Total Issues	**8** (4 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**3** (1 Resolved)
Informational Risk Issues	**5** (3 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

StakeHound is token contract and a staking algorithm and as any ERC20 token, it is vulnerable to allowance double-spend exploit. The staking reward mechanism contain some medium flaws that can be addressed.

ID	Description	Severity	Status
QSP-1	Token Distribution	Low	Acknowledged
QSP-2	Gas Consumption	Low	Acknowledged
QSP-3	Execute Transactions	Low	Fixed
QSP-4	Unlocked Pragma	Informational	Fixed
QSP-5	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-6	DownstreamCaller Update	Informational	Acknowledged
QSP-7	Privileged Roles and Ownership	Informational	Acknowledged
QSP-8	Token Burning	Informational	Fixed

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 .s`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`

Findings

QSP-1 Token Distribution

Severity: Low Risk

Status: Acknowledged

File(s) affected: `StakedToken`

Description: The requirement that restricts the `supplyController` from minting more than `_maxSupply` is implicit (`SafeMath` function will throw inside `StakedToken.mint`). However, `distributeTokens` alter `_sharesPerToken` value, therefore cancelling the initial state `_sharesPerToken = MAX_UINT256.div(maxSupply_)`, this will allow the `supplyController` to mint more tokens than `_maxSupply` or in the opposite case (contracted supply) restrict the total supply from reaching `_maxSupply`.

Recommendation: Consider removing the supply contraction mechanism and adding a requirement in `mint` and `distributeTokens` functions to check if the `_totalSupply + amount` or `_totalSupply + supplyChange_` is lower than `_maxSupply`.

QSP-2 Gas Consumption

Severity: Low Risk

Status: Acknowledged

File(s) affected: `DownstreamCaller, StakedToken`

Description: Depending on `DownstreamCaller.transactions` array length, the gas consumed during a call to `executeTransactions` can be excessively high potentially throwing the transaction for out of gas or block gas limit. Since `executeTransactions` is used by `StakedToken.distributeTokens` function, a bad management of the transactions array can lead to a temporary denial of service for the token distribution logic.

Recommendation: Even if the elements in `transactions` array can be selectively deleted or disabled, Quantstamp recommend to run a gas consumption simulation before adding transactions to the `DownstreamCaller` contract.

QSP-3 Execute Transactions

Severity: Low Risk

Status: Fixed

File(s) affected: `DownstreamCaller`

Description: `DownstreamCaller.executeTransactions` is a public function. Depending on the listed transactions, allowing it to be called by a non-owner or by any other address than `StakedToken` can be a risk. No specifications were provided to correctly estimate the impact of this issue.

Recommendation: Only allow `DownstreamCaller.executeTransactions` to be called by `StakedToken` contract address.

QSP-4 Unlocked Pragma

Severity: Informational

Status: Fixed

File(s) affected: `DownstreamCaller, StakedToken`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.6.*`. 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."

Exploit Scenario: 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-5 Allowance Double-Spend Exploit

Severity: Informational

Status: Mitigated

File(s) affected: `StakedToken`

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`.

Recommendation: 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.

QSP-6 DownstreamCaller Update

Severity: *Informational*

Status: Acknowledged

File(s) affected: `StakedToken`

Description: `DownstreamCaller` is deployed when `StakedToken` is initialized. However, the contract can be modified by the owner using `setDownstreamCaller`, this does not guarantee that the already listed transactions will be migrated to the new contract.

Recommendation: Depending on the importance of the listed transactions, the migration process can be implemented automatically to avoid any possible issue.

QSP-7 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

File(s) affected: `StakedToken`

Description: - `transfer` and `transferFrom` can be paused by the owner.

- Users can be denied access by the owner to `transfer`, `transferFrom`, `approve`, `increaseAllowance`, `decreaseAllowance` and `mint`, the blacklisting is selective and can be applied to any address.
- `mint`function allows the Supply controller can change the supply of token arbitrarily without using `distributeTokens`

Recommendation: The privileged roles need to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

QSP-8 Token Burning

Severity: *Informational*

Status: Fixed

File(s) affected: `StakedToken`

Description: Only the supply controller is allowed to burn tokens through `StakedToken.burn`, however, `msg.sender` is used to set the `account` from where the tokens are burned. We cannot determine if this is an error or part of the design, the code documentation does not specify the addresses allowed to use the described functionality.

Recommendation: We recommend to clearly specify the intended behavior or modify the function implementation to meet the specification.

Automated Analyses

Slither

- StakedToken.initialize(string,string,uint8,uint256,uint256) performs a multiplication on the result of a division, this issue is classified as false positive since it is an intended behavior.

Mythril

Mythril reported several issues, however, after the manual review all issues were classified as false positive.

Adherence to Specification

- The developer code documentation of 'distributeTokens' is incorrect as the function can both increase or decrease the supply of tokens if `positive`parameter is falsed.

Adherence to Best Practices

- Implement input validation in `DownstreamCaller.addTransaction`, `destination` should be different than `address(0x0)` and `data` length should be higher than zero.
- To manage an added transaction in `DownstreamCaller` contract the transaction index is used. However, `addTransaction` does not return the index or emit an event that allows to read the transaction index. Either use the index as a return value or implement an event to keep track of the `{Transaction, index}` pair.
- `StakedToken.approve`, `StakedToken.increaseAllowance` and `StakedToken.decreaseAllowance` functions allow the `spender` address to be `address(0x0)`. In this case the allocation can not be spent since it is allowed to `address(0x0)`. However, the functions should throw with a correct error message to inform the user about the input error.
- `account` input in `StakedToken.mint` function is not checked to be different than `address(0x0)`.
- `supplyController_` input in `StakedToken.setSupplyController` is not checked to be different than `address(0x0)`.

Test Results

Test Suite Results

```
  StakedToken
    Initialization
       ✓ should be set up properly (226ms)
       ✓ should reject ETH transfers
    Upgrades
       ✓ should be upgradeable (878ms)
    setSupplyController
       ✓ should update supply controller (263ms)
       ✓ should not be callable by others (38ms)
    setName
       ✓ should update name (254ms)
       ✓ should not be callable by others (40ms)
    setSymbol
       ✓ should update symbol (233ms)
       ✓ should not be callable by others (45ms)
    Transfers
       ✓ should transfer tokens (183ms)
       ✓ should fail to transfer too many tokens (84ms)
    Minting
       ✓ should mint new tokens (130ms)
       ✓ should not be callable by others (39ms)
    Burning
       ✓ should burn tokens (111ms)
       ✓ should fail to burn more than in account
       ✓ should not be callable by others (47ms)
    Reward distribution
       ✓ should distribute rewards (235ms)
       ✓ should contract the supply (231ms)
Increased supply by 1 to 1000000000000000000001, actually increased by 1
Doubling supply 0
Increased supply by 1 to 2000000000000000000003, actually increased by 1
Doubling supply 1
Increased supply by 1 to 4000000000000000000007, actually increased by 1
Doubling supply 2
Increased supply by 1 to 8000000000000000000015, actually increased by 1
Doubling supply 3
Increased supply by 1 to 16000000000000000000031, actually increased by 1
Doubling supply 4
Increased supply by 1 to 32000000000000000000063, actually increased by 1
Doubling supply 5
Increased supply by 1 to 64000000000000000000127, actually increased by 1
Doubling supply 6
Increased supply by 1 to 128000000000000000000255, actually increased by 1
Doubling supply 7
Increased supply by 1 to 256000000000000000000511, actually increased by 1
Doubling supply 8
Increased supply by 1 to 512000000000000000001023, actually increased by 1
Doubling supply 9
Increased supply by 1 to 1024000000000000000002047, actually increased by 1
Doubling supply 10
Increased supply by 1 to 2048000000000000000004095, actually increased by 1
Doubling supply 11
Increased supply by 1 to 4096000000000000000008191, actually increased by 1
Doubling supply 12
Increased supply by 1 to 8192000000000000000016383, actually increased by 1
Doubling supply 13
Increased supply by 1 to 16384000000000000000032767, actually increased by 1
Doubling supply 14
Increased supply by 1 to 32768000000000000000065535, actually increased by 1
Doubling supply 15
Increased supply by 1 to 65536000000000000000131071, actually increased by 1
Doubling supply 16
Increased supply by 1 to 131072000000000000000262143, actually increased by 1
Doubling supply 17
Increased supply by 1 to 262144000000000000000524287, actually increased by 1
Doubling supply 18
Increased supply by 1 to 524288000000000000001048575, actually increased by 1
Doubling supply 19
       ✓ should maintain supply precision for 20 doublings (4211ms)
       ✓ should not be callable by others
    Allowances
       ✓ should transfer if allowance is big enough (241ms)
       ✓ should fail to transfer if the allowance is too small (127ms)
    External calls
       ✓ should register a downstream contract and call it on distribution (239ms)
       ✓ should remove a downstream transaction (318ms)
       ✓ should disable a downstream transaction (309ms)
       ✓ should change the downstream caller contract (592ms)
       ✓ should not be callable by others (38ms)
    Pausable
       ✓ should fail token transfers when paused (85ms)
       ✓ should fail to transferFrom when paused (206ms)
       ✓ should unpause (361ms)
       ✓ should not be callable by others (62ms)
    Blacklisting
       ✓ should fail token transfers when sender is blacklisted (77ms)
       ✓ should fail token transfers when recipient is blacklisted (80ms)
       ✓ should fail to transferFrom when sender is blacklisted (197ms)
       ✓ should fail to transferFrom when recipient is blacklisted (196ms)
       ✓ should fail to set allowance when sender is blacklisted (91ms)
       ✓ should fail to increase allowance when sender is blacklisted (92ms)
       ✓ should fail to decrease allowance when sender is blacklisted (87ms)
       ✓ should fail to set allowance when spender is blacklisted (85ms)
       ✓ should fail to increase allowance when spender is blacklisted (91ms)
       ✓ should fail to decrease allowance when spender is blacklisted (85ms)
       ✓ should disable blacklist (442ms)
       ✓ should not be callable by others


  43 passing (27s)
```

Code Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts/	86.61	72.73	91.89	86.96
DownstreamCaller.sol	82.35	60	100	83.33
StakedToken.sol	87.37	76.47	90.32	87.63
All files	**86.61**	**72.73**	**91.89**	**86.96**

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

```
c773287965ad4e612efdc08b6cbe17973aa23b81ed3255557a71dfa4a12d64b2 ./contracts/DownstreamCaller.sol
8ff22272f3f9466ed336e827ae69829bed8cea0093921db8a21a71caaa5d80f0 ./contracts/StakedToken.sol
```

Tests

```
909107da61056e680cf842c916dcf9e89d2a8d229c7938cc33c7131de1c2814c ./test/StakedToken.behavior.ts
7a015ec4eef320407aa5bfc2326d26ac8ccf7802b818fb43e1d7dbdb6ceaf322 ./test/StakedToken.ts
```

Changelog

- 2020-10-02 - Initial report
- 2020-10-07 - re-audit and report update

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.