Quantstamp Security Assessment Certificate

April 13th 2021 — Quantstamp Verified

# Voyager Token

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

## Executive Summary

| | |
|---|---|
| Type | Token, Staking and Swapping |
| Auditors | Sebastian Banescu, Senior Research Engineer<br>Jan Gorzny, Blockchain Researcher<br>Leonardo Passos, Senior Research Engineer |
| Timeline | 2021-02-10 through 2021-04-05 |
| EVM | Muir Glacier |
| Languages | Solidity |
| Methods | Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review |
| Specification | Voyager Token, Swap & Staking |
| Documentation Quality | Medium |
| Test Quality | Medium |

Source Code

| Repository | Commit |
|---|---|
| swap-for-stake/src/blockchain | 8073cff |

| | |
|---|---|
| Total Issues | **10** (8 Resolved) |
| High Risk Issues | **2** (2 Resolved) |
| Medium Risk Issues | **2** (2 Resolved) |
| Low Risk Issues | **2** (2 Resolved) |
| Informational Risk Issues | **3** (1 Resolved) |
| Undetermined Risk Issues | **1** (1 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

Quantstamp has performed a security audit of the Republic Voyager Token, Swap and Staking contracts and has identified 10 issues ranging across all severity levels. Additionally we have identified 6 best practice issues, 3 documentation issues and a low branch coverage when executing the test suite. We recommend addressing all of these issues before deploying the code on a production environment.

**Update:** The reaudit has been performed and the report has been updated w.r.t. the code under commit hash `c8c2ad1`.

**Update:** The reaudit has been performed and the report has been updated w.r.t. the code under commit hash `8073cff`.

| ID | Description | Severity | Status |
|---|---|---|---|
| QSP-1 | Undocumented Staking And Accrued Interest Logic | High | Fixed |
| QSP-2 | Unchecked Return Value | High | Fixed |
| QSP-3 | Missing Input Validation | Medium | Fixed |
| QSP-4 | Integer Overflow / Underflow | Medium | Fixed |
| QSP-5 | Increased Loss Of Precision | Low | Mitigated |
| QSP-6 | `init` Accepts Zero Minters | Low | Fixed |
| QSP-7 | Unlocked Pragma | Informational | Fixed |
| QSP-8 | Allowance Double-Spend Exploit | Informational | Acknowledged |
| QSP-9 | Gas Usage / `for` Loop Concerns | Informational | Acknowledged |
| QSP-10 | Accruing Interest Starting Point Could Be In The Past | Undetermined | 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 v0.7.0

Steps taken to run the tools:

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

# Findings

## QSP-1 Undocumented Staking And Accrued Interest Logic

Severity: *High Risk*

**Status:** Fixed

**File(s) affected:** `Staking.sol`

**Description:** The logic in `_stake`, `_unstake`, `_accrueInterest` and `_calculateAccruedInterest` is not properly documented. The specification does not mention anything related to accrued interest; hence, there is no reference to match against the implemented code. Moreover, the mathematical relations for computing the total amount of shares when staking and unstaking is not documented. As a consequence, one cannot tell if the implementation is indeed correct and fulfils the intended behavior. This may have a significant impact on end-users.

**Recommendation:** Enhance the specification markdown file with a descriptive explanation of staking, unstaking and interest accrual.

**Update:** This issue was fixed by adding a description of the 4 functions in the `README.md` file.

## QSP-2 Unchecked Return Value

Severity: *High Risk*

**Status:** Fixed

**File(s) affected:** `TokenSwap.sol`

**Related Issue(s):** SWC-104

**Description:** According to the ERC20 standard the `transfer` and `transferFrom` functions return `true` if the transfer was successful and `false` otherwise. These functions do not need to revert if the transfer is not successful. However, `TokenSwap.swap(uint256)` ignores return value by `fromToken.transferFrom(msg.sender,burnAddress,fromAmount)` on L54. Since `fromToken` is an arbitrary IERC20 (which was not in the scope of this audit), one cannot assume anything about how that token may fail.

**Recommendation:** Require the return value of `transferFrom` to be `true`.

**Update:** This issue has been fixed by replacing `transferFrom` with `safeTransferFrom`, which reverts in case of failure to transfer funds.

## QSP-3 Missing Input Validation

Severity: *Medium Risk*

**Status:** Fixed

**File(s) affected:** `StakeToken.sol`, `Staking.sol`, `TokenSwap.sol`

**Description:** The following functions are missing input parameter validations:

1. `StakeToken.init` does not check if its input parameters of type `address` are different from `address(0)`.
2. `StakeToken.mint` does not check if its input parameter of type `address` is different from `address(0)`. It also does not check if the value of `amount` is higher than `0`.
3. `Staking.constructor` does not check if its input parameter of type `address` is different from `address(0)`. It also does not check if the values of its `uint256` parameters are higher than `0`. Also the timestamp parameter is not checked and could be in the past.
4. `Staking.stake` does not check if its `amount` is higher than `0`.
5. `Staking.stakeFor` does not check if its `user` is different than `address(0)` and does not check if `amount` is higher than `0`.
6. `Staking.setInterestRate` does not check if its `newInterestRate` is higher than `0`.
7. `TokenSwap.constructor` does not check if its input parameters of type `address` are different from `address(0)`. It also does not check if the values of its `uint256` parameters are higher than `0`.
8. `TokenSwap.swap` does not check if the value of `fromAmount` is higher than `0`.

**Recommendation:** Add input validation for each of the input parameters and functions mentioned above. Unless input addresses are known to be correct (e.g., controlled by deployment scripts), we suggest verifying that:

1. they are different than `address(0)`;
2. they are indeed contracts (when applicable).

**Update:** The status of each sub-point above is given below along with a clarification comment, where necessary

1. Fixed
2. Fixed
3. Fixed, because the `address` parameter is being checked. Acknowledged that the timestamp can be in the past by at most 1 day.
4. Fixed by adding those checks in the `_stake` function
5. Fixed by adding those checks in the `_stake` function
6. Acknowledged as intended behavior. `Staking.setInterestRate` accepting `0` is intended behavior, allowing the owner to disable interest accrual entirely, should that be required.
7. Fixed
8. Fixed. `TokenSwap.swap` does not check if `fromAmount > 0`.

## QSP-4 Integer Overflow / Underflow

**Severity:** *Medium Risk*

**Status:** Fixed

**File(s) affected:** `StakeToken.sol`

**Description:** Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute.

The addition on L64 of `StakingToken` may overflow since there is no hard cap on the total token supply:

```
return totalSupply() + _stakingContract.totalUnmintedInterest();
```

**Recommendation:** Rely on `SafeMath` arithmetic operators, which throw upon overflow/underflow, instead of using Solidity native ones, which do not revert.

## QSP-5 Increased Loss Of Precision

**Severity:** *Low Risk*

**Status:** Mitigated

**File(s) affected:** `Staking.sol`

**Description:** The integer division in Solidity leads to a loss in precision due to the truncation of decimal digits. Therefore division should be always performed as the last arithmetic operation in any computation where this is possible.

1. `Staking._calculateAccruedInterest()` performs a multiplication on the result of a division:
   - `newBalance = newBalance.mul((uint256(1000000).add(_interestRateDaily)) ** daysRemain).div(1000000 ** daysRemain)` on L228
   - `newBalance = newBalance.mul((uint256(1000000).add(_interestRateDaily)) ** 10).div(1000000 ** 10)` on L231
2. `Staking._calculateAccruedInterest()` performs a multiplication on the result of a division:
   - `daysElapsed = (block.timestamp.sub(_lastInterestAccruedTimestamp)).div(86400)` on L222
   - `timestamp = _lastInterestAccruedTimestamp.add(daysElapsed.mul(86400))` on L237.

**Recommendation:** Move the division after the multiplication.

**Update from dev team:** The issue is related to deliberate design decisions, and, as a result, remains in the code. Steps have been taken to demonstrate the extent of potential effects:

1. In case 1, a unit test has been added to demonstrate that the loss in tokens does not exceed $10^{-5}$ tokens per year per position, which is negligible.
2. In case 2, loss of precision is deliberate, as this is a rounding operation - the function increments `_lastInterestAccruedTimestamp` only by full days.

## QSP-6 `init` Accepts Zero Minters

**Severity:** *Low Risk*

**Status:** Fixed

**File(s) affected:** `StakeToken.sol`

**Description:** Presently, the `init` function in the `StakeToken` contract does not require the `minterAddresses` array to have a length greater than zero. If it has a length of zero, `StakeToken` will be deemed initialized, but will prevent minting from ever occurring, which defeats the purpose of the contract.

**Recommendation:** Add a `require` statement to the `init` function s.t. it checks that the `minterAddresses` array has `length > 0`.

## QSP-7 Unlocked Pragma

**Severity:** *Informational*

**Status:** Fixed

**File(s) affected:** `All`

**Description:** Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.7.*`. 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-8 Allowance Double-Spend Exploit

**Severity:** *Informational*

**Status:** Acknowledged

**File(s) affected:** `StakeToken.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 `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.

**Update from dev team:** `StakeToken` inherits from OpenZeppelin ERC-20, which implements `increaseAllowance` and `decreaseAllowance` - these functions can be used by third-party developers interacting with the token.

### QSP-9 Gas Usage / `for` Loop Concerns

Severity: *Informational*

**Status:** Acknowledged

**File(s) affected:** `StakingToken.sol`

**Related Issue(s):** SWC-128

**Description:** Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely.

The `StakeToken.init` function contains a loop which is bounded by user provided input and hence it could run into an out of gas error if the length of the `minterAddresses` array was long enough.

**Recommendation:** Require the length of the `minterAddresses` array to be less than a reasonable number of minters, e.g. `5`. In case a high number of minters is needed perform a gas analysis to make sure that this function does not run out of gas.

**Update from dev team:** The function in question is called only once on initial contract deployment - the `StakeToken` is initialized with only three minter addresses - two `TokenSwap` contracts and the `Staking` contract. Therefore, an "out of gas" exception is unlikely.

### QSP-10 Accruing Interest Starting Point Could Be In The Past

Severity: *Undetermined*

**Status:** Mitigated

**File(s) affected:** `Staking.sol`

**Description:** In the `Staking` contract constructor, `_lastInterestAccruedTimestamp` could be set to be in the past, i.e., when `startingTimestamp < block.timestamp`. This seems incorrect, and could lead to unknown side effects.

**Recommendation:** Add a `require` statement to the `Staking` contract constructor verifying that `startingTimestamp` is great than `block.timestamp`.

**Update:** The timestamp to start staking cannot be set to earlier than 1 day behind the deployment date.

## Automated Analyses

### Slither

The Slither analyzer has detected 74 issues, the majority of which were filtered out as false positives. The remaining issues were integrated in the findings listed above.

## Code Documentation

1. **[Fixed]** The `SPEC.md` file states that "*the staking token implements both the ERC-20 interface and the EIP-900 staking contract interface.*". This is not accurate, as the `StakeToken` contract only conforms to the `ERC20` interface.

2. **[Fixed]** The specification markdown file does not explicitly make a connection between `StakeToken` and `VGR`. Make such a connection explicit in the documentation and in the `StakeToken.sol` file as a means to aid understanding.

3. **[Fixed]** In `TokenSwap.sol`, L29 and 30 state that:

```
_swapFromQuantity: numerator in the swapping fraction (e.g. 100 OldTokens for 30 NewTokens)
_swapToQuantity: denominator in the swapping fraction, respectively
```

   However, as per calculation in the `swap` function (`TokenSwap.sol`, L55), `_swapFromQuantity` is the denominator, whereas `_swapToQuantity` is the numerator.

## Adherence to Best Practices

1. **[Fixed]** There is no need for the `StakeToken` contract to be a direct subtype of `IERC20`, as that is already the case through the inheritance of `ERC20`. Hence, change its declaration to `contract StakeToken is ERC20, Ownable {`

2. **[Fixed]** Some parts of the code could use better indenting. We recommend using an automated indenting tool to make indentation consistent across the code.

3. **[Fixed]** Mock files, which fulfil a testing concern, are mixed with production code that ought to be deployed to mainnet. This is not ideal. Rather, we recommend isolating test-related code into its own folder (e.g., `test/resources/contracts`).

4. **[Fixed]** `Staking._calculateAccruedInterest()` uses literals with too many digits on L228: `newBalance = newBalance.mul((uint256(1000000).add(_interestRateDaily)) ** daysRemain).div(1000000 ** daysRemain)`. Same applies for L231. These should be replaced with `10**6`.

5. **[Fixed]** Avoid using magic numbers and replace them with named constants, whose names should provide a semantic meaning, not simply a phrase indicating the constant's value. There are several magic numbers like `10` and `1000000` in `Staking.sol`. These should be replaced.

6. **[Fixed]** The formula for `_newShares` (`Staking.sol`) can be simplified by replacing the existing code on L167-169 with:

```
uint256 newShares = _totalShares == 0 ? amount : _totalShares.mul(amount).div(_totalStaked);

_totalStaked = _totalStaked.add(amount);
```

## Test Results

### Test Suite Results

We confirm that all 22 test cases are passing.

**Update:** The team has added 9 tests and all 31 tests are passing.

**Update:** The team has added 21 tests and all 52 tests are passing.

```
Network Info
============
> HardhatEVM: v2.0.5
> network:    hardhat


  StakingConstructor
     ✓ should not allow to set a zero token address
     ✓ should not allow to launch with interest rate > max interest rate
     ✓ should not allow to launch with max interest set to 0
     ✓ should not allow to launch with interest set to 0
     ✓ should not allow to set a date more than 1 day in the past

  Staking
     ✓ should not allow setting more than the max interest rate
     ✓ should have expected token test setup
     ✓ should allow staking and unstaking tokens (210ms)
     ✓ should revert on zero staked amount
     ✓ should revert when staking for zero address
     ✓ should revert on insufficient balance
     ✓ should revert on insufficient allowance
     ✓ should revert on incorrect transferFrom (212ms)
     ✓ should revert on incorrect transfer (224ms)
     ✓ should not allow to unstake 0 (40ms)
     ✓ should not allow to unstake more than the user balance (47ms)
     ✓ should keep track of total staked from multiple stakers (173ms)
     ✓ should define the staking token
     ✓ should not support history
     ✓ should emit a staked and unstaked events
     ✓ should allow the owner to set the interest rate
     ✓ should not allow non-owners to set the interest rate
     ✓ should not allow the interest rate to exceed the allowed maximum
     ✓ should accrue interest over time (112ms)
     ✓ should be able to accrue interest manually (85ms)
     ✓ should not accrue interest in less than a day (70ms)
     ✓ should accrue interest correctly over a long period of time (124ms)
     ✓ should accrue interest correctly after interest rate change (126ms)
     ✓ should accrue interest correctly for multiple stakers in a complex scenario (309ms)
     ✓ staking contract should display unminted interest correctly (145ms)
     ✓ token contract should display real + virtual tokens correctly (181ms)
     ✓ precision loss from staking accrual should not exceed 0.00001 tokens a year (167ms)
     ✓ 30 years of unaccrued interest still fits into a block (5056ms)

  Token
     ✓ deploys a token with the expected details (75ms)
     ✓ new tokens are minted correctly and only by a designated minter (55ms)
     ✓ should be initializable only once
     ✓ should not allow to set a zero address staking contract
     ✓ should not allow zero addresses in the minter list
     ✓ should not allow to initialize with no minters
     ✓ should not allow tokens to mint to zero address
     ✓ should not allow tokens to mint to zero tokens

  TokenSwapConstructor
     ✓ should not allow to create a swap contract with zero from amount
     ✓ should not allow to create a swap contract with zero to amount
     ✓ should not allow to create a swap contract with zero old token address
     ✓ should not allow to create a swap contract with zero new token address

  TokenSwap
     ✓ should deploy with the correct owner
     ✓ should have expected token test setup
     ✓ should deploy swap token with constructor configuration variables
     ✓ should return the right ratio of tokens and burn the swapped tokens (84ms)
     ✓ should not allow to swap zero tokens
     ✓ should emit a swap event
     ✓ can be paused and unpaused (admin only), no swaps during pause (47ms)


  52 passing (20s)
```

## Code Coverage

The branch coverage is low. We recommend increasing the branch coverage up to 100%, in order to increase confidence in the fact that the majority of the functionality in the code is properly covered by tests. Note that an increase in coverage has to be accompanied by an increase in assertion statements that check all the desired effects but also the side effects of function calls.

**Update:** The test suite has been improved and the branch coverage is now higher than 77%. We recommend further increasing this up to 100%.

**Update:** The test suite has been improved and the branch coverage is now 100%.

| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
|---|---|---|---|---|---|
| **contracts/** | 100 | 100 | 100 | 100 | |
| BurnAddress.sol | 100 | 100 | 100 | 100 | |
| StakeToken.sol | 100 | 100 | 100 | 100 | |
| Staking.sol | 100 | 100 | 100 | 100 | |
| TokenSwap.sol | 100 | 100 | 100 | 100 | |
| **contracts/interfaces/** | 100 | 100 | 100 | 100 | |
| IStakingContract.sol | 100 | 100 | 100 | 100 | |
| **contracts/mocks/** | 88.89 | 100 | 88.89 | 88.89 | |
| StakeTokenMock.sol | 100 | 100 | 100 | 100 | |
| StakingMock.sol | 0 | 100 | 0 | 0 | 8 |
| WeirdTokenMock.sol | 100 | 100 | 100 | 100 | |
| **All files** | **99.12** | **100** | **97.44** | **99.13** | |

## 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

```
6c0ffd1da3e273998bfb0eefc6dfe247c953eeeabeb4d33c991446687c37fb4a ./contracts/StakeToken.sol
bd434d1f4b5ac5a60760dfd6b4dffec81a7f6caf43a3ff47f8e4b71847e2b5db ./contracts/TokenSwap.sol
2bfd5a27ea36b43b0732598da5cd4d10916fa98a89bbc56284070e87127e3784 ./contracts/BurnAddress.sol
ce76cb1ccd874ef7a2a3dea5554041e914a9d6dbf876ae1cf4b2eb5efe201764 ./contracts/Staking.sol
6dc1096dc1e7fb7893ca6976d5928f0b132927fd18190cc4c3bc4e11f1f54868 ./contracts/interfaces/IStakingContract.sol
c8e712afc0ad570c799daf89a995248a491b1a455eb9d6f51992d518f6439dd7 ./contracts/mocks/StakingMock.sol
3143c493ee309a6d3dbea7e19cd3db0f12a911adc86dd2bd5a64222c8f3c1df9 ./contracts/mocks/WeirdTokenMock.sol
84438e91cac8d1d131019767d80ff6043cca73c0e60a7f4144094653c82db917 ./contracts/mocks/StakeTokenMock.sol
```

#### Tests

```
1ad5c85a29de55cddea46998d401c228047f0caff26b34f2d23b4fd0008a5a16 ./test/token-deploy.test.js
82954bbf8958eaa376990d788ba637a3b746f4a7c525b3ff4fd620c0a15c0a43 ./test/token-swap.test.js
d953090530d941e66cd8852f8b4f948c6c289760f5d38d1ec13b9c897ed4c81b ./test/staking.test.js
```

## Changelog

- 2021-02-12 - Initial report according to commit hash `f80d689`
- 2021-03-10 - Updated report according to commit hash `c8c2ad1`
- 2021-04-05 - Updated report according to commit hash `8073cff`

## 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.


### Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

### Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

### Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

### Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.