Quantstamp Security Assessment Certificate

August 26th 2021 — Quantstamp Verified

SuperRare Token

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

Executive Summary

Type	ERC-20 Token and its Airdrop
Auditors	Mohsen Ahmadvand, Senior Research Engineer Poming Lee, Research Engineer Joseph Xu, Technical R&D Advisor
Timeline	2021-05-19 through 2021-06-15
EVM	Berlin
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Low
Test Quality	Low

Source Code

Repository	Commit
rarest-token (initial audit)	8c5abd3
rarest-token (re-audit)	50bafc8

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

No High or Medium severity issues were detected in our audit. There is one Low severity issue pertaining to input parameter validations. Three informational issues and two undetermined threats were identified. The SuperRare contract enables the contract owner to mint an unlimited amount of tokens to arbitrary addresses. This resembles a centralisation of power and therefore it has to be explicitly communicated with the platform users. Furthermore, the documentation and test coverage need to be improved.

ID	Description	Severity	Status
QSP-1	Missing Checks If Important Parameters Are Non-Zero	Low	Fixed
QSP-2	Privileged Roles and Ownership	Informational	Acknowledged
QSP-3	Unlocked Pragma	Informational	Fixed
QSP-4	Allowance Double-Spend Exploit	Informational	Acknowledged
QSP-5	Tokens Can Potentially Get Locked in the Airdrop Contract	Undetermined	Fixed
QSP-6	Potentially Zero-Addressed Retrieved Contracts	Undetermined	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

Steps taken to run the tools:

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

Findings

QSP-1 Missing Checks If Important Parameters Are Non-Zero

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/claim/SuperRareTokenMerkleDrop.sol`, `contracts/erc20/SuperRareToken.sol`

Description: `contracts/claim/SuperRareTokenMerkleDrop.sol`: `constructor` does not check if `superRareToken` and `merkleRoot` are non-zero.
`contracts/erc20/SuperRareToken.sol`: `init` does not check if `_owner` is non-zero.

Recommendation: Add relevant checks.

QSP-2 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

File(s) affected: `SuperRareToken.sol`

Description: Smart contracts will often have some variables to designate the person(s) 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.

Exploit Scenario: The SuperRare token deployer address can mint additional tokens to arbitrary addresses without any restriction and can pause all transfers. This easily leads to censorship.

Recommendation: We recommend explicitly mentioning the following information in the user-facing documentation:
There is no cap on the amount of tokens that can be minted. Contract admins can also update the transfer rules at any moment in time as many times as they want.
One possible mitigation strategy on the minting aspect is to allow minting only to a designated time-lock or vesting contract or to include the inflation/distribution mechanism explicitly in the token.

Update: The response from the SuperRare team:

> We're aware of the optics of allowing admins to mint tokens at will and control if people can transfer them, this will be reflected in our Terms of Service of our platform.

QSP-3 Unlocked Pragma

Severity: *Informational*

Status: Fixed

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. 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".

```
- Version used: ['>=0.4.24<0.8.0', '>=0.6.0<0.8.0', '>=0.6.2<0.8.0', '^0.7.3']
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/presets/ERC20PresetMinterPauserUpgradeable.sol#3)
- >=0.4.24<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/Initializable.sol#4)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20BurnableUpgradeable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20PausableUpgradeable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#3)
- >=0.6.2<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/EnumerableSetUpgradeable.sol#3)
- >=0.6.0<0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/PausableUpgradeable.sol#3)
- ^0.7.3 (contracts/InitializableV2.sol#3)
- ^0.7.3 (contracts/Migrations.sol#3)
- ^0.7.3 (contracts/claim/SuperRareTokenMerkleDrop.sol#3)
- ^0.7.3 (contracts/erc20/SuperRareToken.sol#3)
- ^0.7.3 (contracts/registry/Registry.sol#3)
```

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. Hardhat locks the project to a specific Solidity version but this issue still applies as is very common for projects to re-use contracts from other projects.

QSP-4 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Acknowledged

File(s) affected: `SuperRareToken.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: The SuperRare team responded with "we will be sure to use the increase/decrease approval on our end and advise others to do the same."

QSP-5 Tokens Can Potentially Get Locked in the Airdrop Contract

Severity: *Undetermined*

Status: Fixed

File(s) affected: `SuperRareTokenMerkleDrop.sol`

Description: The `_merkleRoot` value in the `SuperRareTokenMerkleDrop.sol` contract must be guaranteed to be correct before tokens are transferred, otherwise Tokens may get locked in the contract.

Recommendation: Consider including an emergency function for the contract owner to either update the `_merkleRoot` or to recover locked tokens.

Update: With the fix the `Owner` can update the Merkle root. On a side note, extra indents seem to be introduced to the contract, which needs to be linted.

QSP-6 Potentially Zero-Addressed Retrieved Contracts

Severity: *Undetermined*

Status: Fixed

File(s) affected: `Registry.sol`

Description: The `getContractVersionCount` function also counts in the entries whose contract has been removed through function `removeContract`. This leads to address `0x0` being counted as one version. Moreover, the `getContract` function can potentially return a zero address when `_version` is pointing to an element in the `addressStorageHistory` map that is set to zero by the `removeContract` function.

Recommendation: We recommend to adhere to the fail early principle. Revise the logic to ensure that zero addresses can not be returned as a legitimate contract address.

Update: The issue was fixed by entirely removing the Registry.sol contract.

Automated Analyses

Slither

In total 119 issues were detected. We analysed those issues and triaged false positives. Except for the first issue (faulty modifier), we believe the rest of short-listed issues are not security relevant.

Faulty modifier

```
Modifier Migrations.restricted() (contracts/Migrations.sol#13-17) does not always execute _; or revertReference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-modifier
```

Dead code

```
InitializableV2._isInitialized() (contracts/InitializableV2.sol#39-41) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

Variable naming

```
Parameter Migrations.setCompleted(uint256)._completed (contracts/Migrations.sol#19) is not in mixedCase
Parameter Migrations.upgrade(address)._newAddress (contracts/Migrations.sol#23) is not in mixedCase
Variable SuperRareTokenMerkleDrop._owner (contracts/claim/SuperRareTokenMerkleDrop.sol#9) is not in mixedCase
Variable SuperRareTokenMerkleDrop._merkleRoot (contracts/claim/SuperRareTokenMerkleDrop.sol#10) is not in mixedCase
Variable SuperRareTokenMerkleDrop._superRareToken (contracts/claim/SuperRareTokenMerkleDrop.sol#11) is not in mixedCase
Variable SuperRareTokenMerkleDrop._claimed (contracts/claim/SuperRareTokenMerkleDrop.sol#12) is not in mixedCase
Parameter SuperRareToken.init(address)._owner (contracts/erc20/SuperRareToken.sol#34) is not in mixedCase
Variable SuperRareToken.DOMAIN_SEPARATOR (contracts/erc20/SuperRareToken.sol#29) is not in mixedCase
Parameter Registry.addContract(bytes32,address)._name (contracts/registry/Registry.sol#53) is not in mixedCase
Parameter Registry.addContract(bytes32,address)._address (contracts/registry/Registry.sol#53) is not in mixedCase
Parameter Registry.removeContract(bytes32)._name (contracts/registry/Registry.sol#74) is not in mixedCase
Parameter Registry.upgradeContract(bytes32,address)._name (contracts/registry/Registry.sol#93) is not in mixedCase
Parameter Registry.upgradeContract(bytes32,address)._newAddress (contracts/registry/Registry.sol#93) is not in mixedCase
Parameter Registry.getContract(bytes32)._name (contracts/registry/Registry.sol#118) is not in mixedCase
Parameter Registry.getContract(bytes32,uint256)._name (contracts/registry/Registry.sol#125) is not in mixedCase
Parameter Registry.getContract(bytes32,uint256)._version (contracts/registry/Registry.sol#125) is not in mixedCase
Parameter Registry.getContractVersionCount(bytes32)._name (contracts/registry/Registry.sol#143) is not in mixedCase
Parameter Registry.setAddress(bytes32,address)._key (contracts/registry/Registry.sol#155) is not in mixedCase
Parameter Registry.setAddress(bytes32,address)._value (contracts/registry/Registry.sol#155) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

Gas optimization opportunities

```
setCompleted(uint256) should be declared external:
    - Migrations.setCompleted(uint256) (contracts/Migrations.sol#19-21)
upgrade(address) should be declared external:
    - Migrations.upgrade(address) (contracts/Migrations.sol#23-26)
claim(uint256,bytes32[]) should be declared external:
    - SuperRareTokenMerkleDrop.claim(uint256,bytes32[]) (contracts/claim/SuperRareTokenMerkleDrop.sol#25-33)
init(address) should be declared external:
    - SuperRareToken.init(address) (contracts/erc20/SuperRareToken.sol#34-65)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (18 contracts with 75 detectors), 119 result(s) found
```

Adherence to Specification

`SuperRareToken.sol` comment on L21 has incorrect calculations. 1 million tokens at 18 decimals is equal to 10^24 base units. Ensure that the comments in the code correspond to the actual implementation.

Code Documentation

There is no documentation. It is strongly advised to document assumptions, usage, and future plans.

Adherence to Best Practices

The included tests indicate a low coverage (about ~37%). It is strongly recommended to aim for a 100% test coverage.

Test Results

Test Suite Results

SuperRareTokenMerkleDrop
😓 Update Merkle Root
😓 Attempt to Update Merkle Root from Non-Owner Address
😓 Deploy - fail - 0 token Address
😓 Deploy - fail - 0 Merkle Root

SuperRareToken
😓 Token init - fail
😓 Token Properties Setup Correctly (41ms)
😓 Token Roles Setup Correctly (60ms)
😓 Token Minting Functionality (92ms)
😓 Token Pausing Functionality (102ms)

9 passing (1s)

Code Coverage

The implemented tests achieve an overall statement and branch coverage of 36.84 and 36.36, respectively. We strongly advice to develop more tests reaching a 100% coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	12.5	0	14.29	11.11	
InitializableV2.sol	33.33	0	33.33	33.33	33,40
Migrations.sol	0	0	0	0	… 15,20,24,25
contracts/claim/	36.84	50	33.33	36.84	
SuperRareTokenMerkleDrop.sol	36.84	50	33.33	36.84	… 47,48,51,55
contracts/erc20/	54.55	33.33	50	58.33	
SuperRareToken.sol	54.55	33.33	50	58.33	79,80,87,88,92
All files	36.84	36.36	26.67	37.5	

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

```
51373f620e0355729083e78076380f42fadee79b39dcb714d5da4cb81d0d95e7 ./contracts/Migrations.sol
f0e808bd8dea48f176f4fa977f3e418387372e3256b42f7aa5067cd597e280f6 ./contracts/InitializableV2.sol
7aea17f95f57e909eb319f14bbaf67eeab3bc429af0d719050cc5c3f178e5191 ./contracts/claim/SuperRareTokenMerkleDrop.sol
c0d270bd6e00cc925eaf9a68fc3e833bf94bb57c71a4270c19e705a91eba88e7 ./contracts/erc20/SuperRareToken.sol
```

Tests

```
173f97220be7515b14b6dbc2e6009d948c860cbb36b992f97ac1b19a5b70618e ./test/claim/SuperRareTokenMerkleDrop.test.js
949be2edeec22886cded83c80bd948a51349b3a4d874c1bfaf282e6fb97a98b4 ./test/erc20/SuperRareToken.test.js
```

Changelog

- 2021-05-24 - Initial report
- 2021-06-15 - Fixes audit (re-audit)

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.