Quantstamp Security Assessment Certificate

March 22nd 2022 — Quantstamp Verified

Athens

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

Executive Summary

Type	Governance Token
Auditors	Cristiano Silva, Research Engineer Jan Gorzny, Blockchain Researcher Marius Guggenmos, Senior Research Engineer
Timeline	2021-11-22 through 2021-11-29
EVM	London
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Documentation Quality	Medium
Test Quality	High

Source Code

Repository	Commit
athens-token	7ed63b
None	3d10dc

Goals

- Match code against specification
- Find potential exploits
- Find logical bugs

Total Issues	4 (1 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	2 (1 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 code is well presented, and it is a fork from the Merkle Tree implementation from Uniswap. The `README.md` file states that these audited modules receive only the `SGOV` token, a proprietary token, and decimals according to the standard (18 decimals). Because of this, we will not raise issues related to: (a) acceptance of tokens having arbitrary number of decimals; (b) duplicated token names and symbols allowed. A total of four issues were reported. After the fixes, 1 issue was resoved, 1 issues was acknowledged, and 2 issues were unresolved.

ID	Description	Severity	Status
QSP-1	Ownership can be renounced	⌄ Low	Fixed
QSP-2	Missing validation of the token address	⌄ Low	Acknowledged
QSP-3	Different versions of solidity used	○ Informational	Unresolved
QSP-4	Gas Optimization: use the solidity 0.8 new data type named `Errors` in `revert` operations	○ 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

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 Ownership can be renounced

Severity: *Low Risk*

Status: Fixed

File(s) affected: `contracts/*`

Description: The contract `Ownable.sol` has the function `renounceOwnership()`. Although it can only be called by the `owner`, such a function leaves the contract without an owner, which surely compromises any ability to manage the contract. Below we present the code of this dangerous function.

```
    function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(address(0));
    }
```

Recommendation: Overwrite this function to make it revert in order to eliminate the risk of leaving the contract without an owner and thus making unclaimed funds unrecoverable.

Update: The developers team replied that "RenounceOwnership function has been overridden to revert the transaction when called".

QSP-2 Missing validation of the token address

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `contracts/MerkleDistributor.sol`

Description: The `constructor(...)` isn't hard-wired for the governance token. This may increase reusability, but also increases the risk that deployment must be repeated.

Recommendation: Ensure that the constructor is able to receive only the proprietary token of the application (SGOV). If the token address is known, hard-code it.

Update: The development team replied that "StormX will take extra care when deploying the Merkle Distributor to use the correct Governance token address during the deployment. We want to make the repository more universal if someone wants to fork and reuse it".

QSP-3 Different versions of solidity used

Severity: *Informational*

Status: Unresolved

File(s) affected: `contracts/*`

Description: The code use versions 0.8.4 and 0.8.0.

Recommendation: Stick to version 0.8.4.

Update: The development team replied that: "OpenZeppelin contracts specify Solidity version 0.8.0 or above (^0.8.0) and based on our research, that's intended. To use a fixed version of Solidity that matches our contract, we configured Hardhat to use 0.8.4 in the configuration file (hardhat.config.ts)".

QSP-4 Gas Optimization: use the solidity 0.8 new data type named `Errors` in `revert` operations

Severity: *Informational*

Status: Unresolved

File(s) affected: `contracts/*`

Description: The cost of the `revert` operation is proportional to the string length passed as parameter. Solidity 0.8 introduced another way of reporting errors, which incurs in less gas usage. Basically, we define an error object, and such object is passed as parameter to the `revert` command. Notice that such error objects can be defined outside of the contract, allowing the developer to create one single file defining all the possible errors. Errors also accept parameters. More information is available at: https://docs.soliditylang.org/en/v0.8.9/contracts.html#errors-and-the-revert-statement

Recommendation: In order to save gas, adopt the error scheme proposed in Solidity 0.8.x.

Update: The development team replied that "the error data type is still not widely used, especially in the current version of OpenZeppelin. We decided to keep the same errors convention as OpenZeppelin uses, that is: require(condition, “ContractName: error message”)".

Automated Analyses

Slither

No outstanding issue was left after filtering the issues reported by Slither in the scope of this audit.

Adherence to Specification

The code adheres to the requirements. The specification is short, and it could be improved for the general public.

Code Documentation

The code presents a medium level of documentation. However, this is basically a fork from the Uniswap Merkle Tree implementation.

Adherence to Best Practices

The code does not take advantage of the new features of solidity 0.8.x (use of the object Error in revert operations).

Test Results

Test Suite Results

The tests report adequate results.

```
yarn run v1.22.17
$ hardhat test
No need to generate any newer typings.


  Governance
    ERC20
       ✓ reverts if initialize() called when owner is zero address (150ms)
       ✓ has correct name
       ✓ has correct symbol
       ✓ has correct number of decimals
       ✓ has correct total supply
    Staking
       ✓ reads locked wallet balance successfully and emits TokenLocked event (46ms)
       ✓ reads unlocked balance successfully and emits TokenUnlocked event (51ms)
    Transfers
       ✓ reverts when transfering to the contract
       ✓ sends transfer successfully (80ms)
       ✓ reverts a transfer if not enough unlocked token (45ms)
       ✓ reverts transferFrom if not enough unlocked token (78ms)
       ✓ uses transferFrom successfully (71ms)
       ✓ reverts lock if not enough unlocked token (48ms)
       ✓ locks successfully (89ms)
       ✓ reverts unlock if not enough locked token (48ms)
       ✓ unlocks tokens successfully (123ms)
       ✓ reverts if input lengths do not match in transfers
       ✓ reverts if any transfer fails
       ✓ uses transfers successfully (47ms)
    Upgradability
       ✓ reverts if initialize() called more than once

  MerkleDistributor
    Token
       ✓ reverts when token is zero address
       ✓ returns the token address (94ms)
    Owner
       ✓ returns the owner (90ms)
       ✓ reverts when calling renounceOwnership (88ms)
    MerkleRoot
       ✓ reverts when merkle root is zero (64ms)
       ✓ returns the merkle root (94ms)
    LockTime
       ✓ reverts lock time is in the past (53ms)
       ✓ returns the lock time (83ms)
    Claim
       ✓ reverts when proof is empty (99ms)
       ✓ reverts when index is invalid (93ms)
       ✓ reverts when trying to claim after lockTime (101ms)
      two account tree
         ✓ successful claim
         ✓ transfers the token
         ✓ must have enough to transfer (103ms)
         ✓ sets isClaimed
         ✓ cannot allow two claims
         ✓ cannot claim more than once: 0 and then 1 (38ms)
         ✓ cannot claim more than once: 1 and then 0 (38ms)
         ✓ cannot claim for address other than proof
         ✓ cannot claim more than proof
         ✓ gas
      larger tree
         ✓ claim index 4
         ✓ claim index 9
         ✓ gas
         ✓ gas second down about 15k
      realistic size tree
         ✓ proof verification works
         ✓ gas
         ✓ gas deeper node
         ✓ gas average random distribution (479ms)
         ✓ gas average first 25 (516ms)
         ✓ no double claims in random distribution
    parseBalanceMap
       ✓ check the proofs is as expected
       ✓ all claims work exactly once (72ms)
    Recover
       ✓ only the owner can call recover
       ✓ reverts when trying to recover before lockTime
       ✓ reverts when trying to recover due to not enough balance (100ms)
       ✓ recovers after lockTime (113ms)


  57 passing (15s)
```

Code Coverage

Code coverage is adequate.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
Governance.sol	100	100	100	100	
MerkleDistributor.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
IMerkleDistributor.sol	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

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

```
e2a1135fd2c07a7d554be363839463bbb1d8c9fbaf71f5a61cbce6594682de43 ./contracts/MerkleDistributor.sol

92355cd588cddba2b588ec22b4470f798f762390b7f9cf2ad1f011412f7b3f3e ./contracts/interfaces/IMerkleDistributor.sol
```

Tests

```
4243e7c93489bc8b7ab86bb3f6f1578c44777edd1766668d5fe02e8ae86e9a7a ./test/Governance.spec.ts

23767a2ac6b5bb8b6649d237997cee5bb33a4dbfa9a173327aaa1d173a5f7423 ./test/MerkleDistributor.spec.ts

937f47428b2e5ee918f4145652511a67b50555e808d49e46c8b8874bd90acbd3 ./test/types.ts

952c156af2d0935f924eaeb34adbbe12b1b684775c51c9435822f75bd79a14a5 ./test/shared.ts
```

Changelog

- 2021-11-23 - Initial report
- 2021-11-29 - Final 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.