Quantstamp Security Assessment Certificate

May 4th 2022 — Quantstamp Verified

Mythical Games 5

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

Executive Summary

Type	ERC20
Auditors	Rabib Islam, Research Engineer Sung-Shine Lee, Senior Research Engineer Fatemeh Heidari, Security Auditor
Timeline	2022-04-11 through 2022-05-04
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
mythical (initial audit)	17bd0fc
mythical (re-audit)	cb2a42f

Total Issues	**4** (3 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	0 (0 Resolved)
Informational Risk Issues	**3** (2 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).
Fixed	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

Initial Audit
In addition to the usual issues found in ERC20s and contracts with privileged roles, we found an issue which, if exploited, would perpetually render users unable to transfer tokens. We are unsure as to whether this is a feature of the contract, as the developers expressed a desire for the possibility of all privileged roles being revoked, and so we raise it as an issue of undetermined severity.
Reaudit Update
All reported issues have been either fixed or acknowledged. In particular, the issue discussed above was fixed.

ID	Description	Severity	Status
QSP-1	Unlocked Pragma	○ Informational	Fixed
QSP-2	Privileged Roles and Ownership	○ Informational	Fixed
QSP-3	Allowance Double-Spend Exploit	○ Informational	Acknowledged
QSP-4	Contract Can Be Left Stuck in Paused State	? 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 v0.8.2

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 Unlocked Pragma

Severity: *Informational*

Status: Fixed

Description: The file `myth.sol` specifies in the header a version number in the format `pragma solidity ^0.8.12`. 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.

Update: The file is now locked to version 0.8.12 of Solidity.

QSP-2 Privileged Roles and Ownership

Severity: *Informational*

Status: Fixed

Description: The smart contract grants powers to an admin address. This address is given the roles `DEFAULT_ADMIN_ROLE` and `PAUSER_ROLE`. Accounts with `PAUSER_ROLE` have the power to pause the contract, preventing users from transferring their tokens. Accounts with `DEFAULT_ADMIN_ROLE` have the power to grant and revoke the above roles.

Recommendation: 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.

Update: The Pausable and AccessControl extensions were removed.

QSP-3 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Acknowledged

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: Project team: "We acknowledge this is an issue with ERC-20 standard and are not going to take any action on it."

QSP-4 Contract Can Be Left Stuck in Paused State

Severity: *Undetermined*

Status: Fixed

Description: `AccessControl.sol` has functions `revokeRole` and `renounceRole`. As it stands, `myth.sol` may be left in a state from which it may be impossible to pause or unpause the contract. If left paused, users of the contract would no longer be able to make token transfers.

Exploit Scenario:

1. An account with `PAUSER_ROLE` pauses the contract.
2. An account with `DEFAULT_ADMIN_ROLE` decides to revoke `PAUSER_ROLE` and `DEFUALT_ADMIN_ROLE` from all other accounts.
3. The above account then renounces its own `DEFAULT_ADMIN_ROLE`.
4. From this point onwards, no account can have `DEFAULT_ADMIN_ROLE` or `PAUSER_ROLE`, and no token transfers can be made.

Recommendation: Determine whether it should be possible to leave the contract in a paused state without recourse. If so, notify users that this is possible. If not, ensure that the contract cannot be left without an account with `PAUSER_ROLE` or `DEFAULT_ADMIN_ROLE` while paused.

Update: The Pausable and AccessControl extensions were removed.

Automated Analyses

Slither

Slither did not return any significant result.

```
Different versions of Solidity is used:
        - Version used: ['0.8.12', '^0.8.0']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - 0.8.12 (contracts/myth.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version0.8.12 (contracts/myth.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.12 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#87-89)
totalSupply() should be declared external:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#94-96)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#101-103)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#113-117)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#136-140)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#158-167)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#181-185)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#201-210)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
. analyzed (5 contracts with 77 detectors), 17 result(s) found
```

Adherence to Best Practices

BP 1.

When changing privileged roles (such as admin and pauser), it is best to do that in two steps:

1. The present privileged role suggests a new address for the change.
2. In a separate transaction, the newly suggested address claims the privileged role.

This two-step update enables for the correction of accidental proposals rather than leaving the system functioning with no/malicious privileged role.
Update: Fixed. The Pausable and AccessControl extensions were removed.

Test Results

Test Suite Results

npx hardhat test

```
Myth token tests
  initial values
     ✓ Should mint 1 Billion tokens to initialTokenHolder when created
     ✓ Should grant the initial admin role
     ✓ Should grant the initial pauser role
  Pauser roles
     ✓ Should be able to grant a new pauser role from the admin role
     ✓ Should NOT be able to grant a new pauser role from a non admin role (40ms)
     ✓ Should be able to renounce a pauser role
     ✓ Should NOT be able to renounce another addresses pauser role
  Pausable
     ✓ Should be able to pause the contract if it has pauser role
     ✓ Should be able to unpause the contract if it has pauser role
     ✓ Should not be able to pause the contract if it does not have the pauser role
     ✓ Should not be able to unpause the contract if it does not have the pauser role
     ✓ Should not be able to transfer tokens when the contract is paused
     ✓ Should be able to transfer tokens when the contract is not paused


13 passing (728ms)
```

Code Coverage

Quantstamp usually recommends developers to increase the branch coverage to 90%and above before a project goes live, in order to avoid hidden functional bugs that might not be easy to spot during the development phase. For branch code coverage, the file currently targeted by the audit achieves a lower score that can be improved further.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	50	100	100	
myth.sol	100	50	100	100	
All files	**100**	**50**	**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

```
1562c475ce87234e0e8b37957622af9aa779ae31278f6404f477eefe37f6213b ./contracts/myth.sol
```

Tests

```
0842a55d4290b5f5d9027ce2166bb590ad108da7b26db194342f460c7d4af65c ./test/myth-test.js
```

Changelog

- 2022-04-13 - Initial report (17bd0fc)
- 2022-05-04 - Re-audit update (cb2a42f)

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.