Quantstamp Security Assessment Certificate

October 24th 2020 — Quantstamp Verified

Axie Infinity

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

Executive Summary

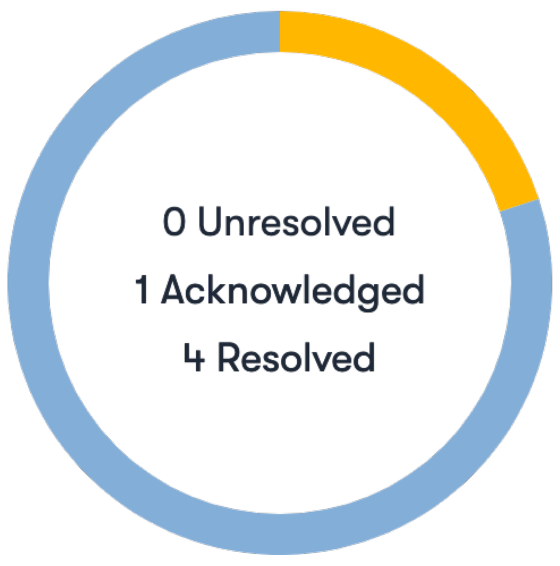
Type	Token Contract
Auditors	Fayçal Lalidji, Security Auditor Joseph Xu, Technical R&D Advisor Luís Fernando Schultz Xavier da Silveira, Security Consultant
Timeline	2020-09-28 through 2020-09-30
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Low
Test Quality	Low
Source Code	

Repository	Commit
axs-smart-contracts	5df14c0

Goals

- Is there any centralization of power?
- Does the code conform to ERC20?
- Can an attacker steal users' funds?

Total Issues	5 (4 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	0 (0 Resolved)
Informational Risk Issues	5 (4 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 implementation of the AXS token does not rely on external reference implementation, this makes the implementation simple. However, cloning the libraries contradicts best practices for the smart contract development. As any ERC20 token, it is vulnerable to allowance double-spend exploit.

ID	Description	Severity	Status
QSP-1	Possible Transfer to Contract Address	Informational	Fixed
QSP-2	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-3	Unlocked Pragma	Informational	Fixed
QSP-4	Clone-and-Own	Informational	Acknowledged
QSP-5	Input Validation	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 .`
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 Possible Transfer to Contract Address

Severity: Informational

Status: Fixed

File(s) affected: `ERC20`

Description: It is rarely desirable for tokens to be sent to the contract itself. However, these mistakes are often made due to human errors. Hence, it's often a good idea to prevent these mistakes from happening within the smart contract itself.

Recommendation: Add a requirement that prevents the destination address to be equal to `address(this)`.

QSP-2 Allowance Double-Spend Exploit

Severity: Informational

Status: Mitigated

File(s) affected: `ERC20`

Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario: 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.

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.

QSP-3 Unlocked Pragma

Severity: Informational

Status: Fixed

File(s) affected: `Several Contracts`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.5.*`. 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-4 Clone-and-Own

Severity: Informational

Status: Acknowledged

File(s) affected: `ERC20, SafeMath`

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.

QSP-5 Input Validation

Severity: Informational

Status: Fixed

File(s) affected: `ERC20`

Description: Due to human errors, function inputs are prone to mistakes. Edge cases should be checked carefully, for example in `transferFrom` and `approve` functions `_from` and `_spender` parameters are not checked to be different than zero address. Even if the input validation is not mandatory, throwing a transaction with a correct revert message helps the users to get a correct feedback.

Recommendation: Add all the necessary requirements with the correct revert messages.

Automated Analyses

Slither

The analysis was completed successfully. No issues were detected.

Mythril

The analysis was completed successfully. No issues were detected.

Code Documentation

The code does not contain documentation. Quantstamp strongly recommends adding comments to describe the implemented logic.

Adherence to Best Practices

- Add messages to `require` statements to indicate why the function call failed in `SafeMath` and `ERC20`.
- When the `allowance` is updated in `ERC20.transferFrom` the `Approval` event is not emitted. This is not required but may be used by Dapps to track allowances.
- In `ERC20` functions `approve`, `transfer` and `transferFrom`, `bool _success` is defined as the return value but it is not set in the functions implementation, instead `return true` is directly used. A similarly practice can be found in `SafeMath` functions `sub`, `div` and `mod`. It is not mandatory to specify a variable name when a function requires a return value, instead just use the return type.

Test Results

Test Suite Results

```
Contract: AXS contract
  ✓ transfer (205ms)


1 passing (2s)
```

Code Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
AXS.sol	100	100	100	100	
contracts/math/	58.33	33.33	60	58.33	
SafeMath.sol	58.33	33.33	60	58.33	17,27,28,33,34
contracts/token/erc20/	47.06	25	50	47.06	
ERC20.sol	35.71	25	33.33	35.71	… 31,32,33,34
ERC20Detailed.sol	100	100	100	100	
IERC20.sol	100	100	100	100	
IERC20Detailed.sol	100	100	100	100	
All files	**56.25**	**31.25**	**60**	**56.25**	

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

```
207aa399088c896e7c0a2c8ba659a6d2240fa2427a91dd702ea8870f2aa86360 ./AXS.sol
a2ae752d26af9c63e6a4c23af318d7e14425b274393a57eb02e67cdb2370ce4c ./token/erc20/IERC20Detailed.sol
6bcf321dce20d1097024e332967bbed8fbe72f882a2e4d6ebd1835601dc9743e ./token/erc20/ERC20Detailed.sol
d63f075de2289a54827a6ae3f52e4421b651114757b5bb7fc45a6e47e6abe74c ./token/erc20/IERC20.sol
0beed269b9ccceb73db5b9fda18761081569dbd5da71db5b221f3e179b110d5a ./token/erc20/ERC20.sol
```

Tests

```
b52fed990b45e64f01b2e7b08dc6a000d1405eb5c2456db23c33cf4e66a00af6 ./test/TokenVesting_test.ts
```

Changelog

- 2020-09-28 - Initial report
- 2020-10-13 - 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.