Quantstamp Contract Security Certificate

May 4th 2020 — Quantstamp Verified

POA Mania

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Lottery Platform Audit
Auditors	Martin Derka, Senior Research Engineer Ed Zulkoski, Senior Security Engineer Kacper Bąk, Senior Research Engineer
Timeline	2020-04-20 through 2020-05-04
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	POA Mania Documentation POA Randomness
Source Code	

Repository	Commit
poamania-contracts	780d5f7

Goals	• To assess the security of the lottery, including but not limited to the safety of user deposits, selection of winners, distribution of winnings, and possible denial of service
Changelog	• 2020-04-29 - Initial report • 2020-05-04 - Updated re-audit report based on 5d6c91d
Overall Assessment	The implementation of the lottery is mostly clean. It features on-chain libraries for selecting winners from the set of participants. The winnings are to be paid out from the operator's reserves, not from the participants' deposits. Quantstamp discovered a scenario when the owner can cause a lock of the platform by setting an incorrect value of the round duration. The test coverage of the implementation is sufficient. The platform could generally use additional input validations as listed in the report, but could be operated as-is with some caution.

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

Summary of Findings

ID	Description	Severity	Status
QSP-1	Temporary Lock of Funds	^ Medium	Resolved
QSP-2	Unlocked Pragma	○ Informational	Resolved
QSP-3	Missing Input Validation	○ Informational	Acknowledged
QSP-4	Centralization of Power	○ Informational	Acknowledged
QSP-5	Cloned Dependency	? Undetermined	Acknowledged
QSP-6	Biased Randomness	? Undetermined	Acknowledged

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:

- Truffle
- Ganache
- SolidityCoverage
- Mythril
- Truffle-Flattener
- Slither

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
4. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
5. Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
6. Installed the Mythril tool from Pypi: `pip3 install mythril`
7. Ran the Mythril tool on each contract: `myth -x path/to/contract`
8. Installed the Slither tool: `pip install slither-analyzer`
9. Run Slither from the project directory `slither .`

Assessment

Findings

QSP-1 Temporary Lock of Funds

Severity: *Medium Risk*

Status: Resolved

File(s) affected: `PoaMania.sol`

Description: If the owner sets the `roundDuration` to an extremely small number, such that on L422, the expression returns a number less than `startedAt`, the funds would get locked until the owner increases `roundDuration`. This vulnerability is related to the centralization of power and missing validations, but has significant consequences and thus is pointed out individually.

Recommendation: Quantstamp recommends adding checks to `_setRoundDuration()` to prevent setting small values.
Update: The issue was resolved. The status in this report was updated accordingly.

QSP-2 Unlocked Pragma

Severity: *Informational*

Status: Resolved

File(s) affected: `PoaMania.sol`, `IPOSDAORandom.sol`, `Sacrifice.sol`

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." 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.

Recommendation: Quantstamp recommends locking pragma at a specific version of solidity.
Update: The issue was resolved. The status in this report was updated accordingly.

QSP-3 Missing Input Validation

Severity: *Informational*

Status: Acknowledged

File(s) affected: `PoaMania.sol`

Description: The code does not validate the input for `minDeposit` and `maxDeposit`. Both the values are repeatedly settable, and the following scenarios may occur:

- `minDeposit` can be larger than `maxDeposit`
- `minDeposit` can increase above the value of the smallest user deposit
- `maxDeposit` can be 0
- `minDeposit` can be 0 (which may be intended), or become prohibitively large

With respect to prizes, all the values are repeatedly settable. The smart contracts do not provide any guarantees for the prize values to match the documentation. The prize values are at a discretion of the POA Mania team. While this may be intended, the following cases are possible to set due to missing validations:

- the probability of winning the jackpot can be 0%
- the jackpot share can be 0%
- the third prize can be higher that the first or the second prize
- the first or second prize can be 0, provided that their sum is positive

Recommendation: Quantstamp recommends adding validation of inputs.
Update: The POA team improved validation of prize sizes and round duration. The team kept setting `minDeposit`, `maxDeposit`, `jackpotShare`, `jackpotChance` and 2nd-3rd prizes to 0 as is. It does not lock users' funds; it is just a game parameter setting. Users can withdraw their money if they do not like changes in game parameters. Quantstamp agrees with the reasoning.

QSP-4 Centralization of Power

Severity: *Informational*

Status: Acknowledged

File(s) affected: `PoaMania.sol`

Description: Smart contracts will often have `owner` variables to designate the person 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.
In the context of POA Mania, the centralization is exhibited by repeatedly settable values that influence deposits, prize values, and the probability of winning a jackpot. Admin does not have any abilities to withdraw participants' funds or change the contract's code.

Recommendation: Quantstamp recommends informing the community about the privileged access.
Update: The POA team acknowledges the ownership role in README.

QSP-5 Cloned Dependency

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `PoaMania.sol`

Description: POA Mania cloned the SortitionSumTree from Kleros and used it as a dependency. The team added functions `total()` and `numberOfNodes()` that are not part of the original Kleros implementation, and the fork does not appear to have tests for them. The functions (linked in the commit below) and this dependency are outside of scope of the current audit.
https://github.com/poanetwork/kleros/commit/4fcc5dff554a3372814b4b8b3784c9b4f4dc0dbe

Recommendation: Quantstamp recommends informing the community about cloning and modifying this dependency
Update: The POA team acknowledges the dependency in README.

QSP-6 Biased Randomness

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `PoaMania.sol`

Description: The randomness in the smart contracts is driven by on-chain data and hash values, using a variation of the commit-and-reveal scheme. Thus, no conclusions can be drawn about the distribution of the values. This is further amplified by using a modulo (%) of the random seeds (so-called "modulo bias"). For the mechanics of the lottery, this mechanism appears sufficient, or at the very least difficult to exploit on regular basis. Yet, the community should be aware of the drawbacks.

Recommendation: Quantstamp recommends that the POA team informs the community about all the issues related to the randomness bias and/or predictability of the on-chain values.
Update: The POA team refers to the randomness description from README.

Automated Analyses

Mythril

Quantstamp assessed all issues reported by Mythril as false positives.

Slither

At the high severity level, Slither reported dangerous send of Ether to an unknown contract. Quantstamp assessed the report as benign. At the medium severity, the following two uninitialized variables, as well as an ignored return value, should be addressed to follow the best practices.

```
INFO:Detectors:
-- jackpotWinner in PoaMania._reward() (PoaMania.sol#256) is a local variable never initialiazed
winnersCurrentDeposits in PoaMania._reward() (PoaMania.sol#234) is a local variable never initialiazed
-- winners in PoaMania._reward() (PoaMania.sol#244) is a local variable never initialiazed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

INFO:Detectors:
-- PoaMania.withdraw(uint256) (PoaMania.sol#202) ignores return value by external calls
"drawManager.withdraw(msg.sender,_amount)" (PoaMania.sol#204)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

Adherence to Specification

A specification is linked from the code repository. It provides clear intentions for the implementation. However, it promises a certain distribution of prizes and the jackpot winning probability, which is not guaranteed in the smart contracts.

Code Documentation

The code is well and clearly documented.

Adherence to Best Practices

The code adheres to the best practices.

Test Results

Test Suite Results

All the provided tests pass and appear to cover the relevant edge cases.

```
  Contract: PoaMania
    initialize
       ✓ should be set up correctly (518ms)
       ✓ fails if any of parameters is incorrect (3935ms)
    deposit
       ✓ should deposit (342ms)
       ✓ fails if zero value (429ms)
       ✓ fails if less than min deposit (292ms)
       ✓ fails if greater than min deposit (327ms)
       ✓ fails if locked (484ms)
    withdraw
       ✓ should withdraw all (572ms)
       ✓ should withdraw specified amount (445ms)
       ✓ fails if zero value (345ms)
       ✓ fails if less than min deposit (228ms)
       ✓ fails if greater than user deposit (181ms)
       ✓ fails if locked (636ms)
       ✓ should withdraw if simple send fails (538ms)
    nextRound
       ✓ should reward and start next round (2097ms)
       ✓ fails if the round is not over yet (80ms)
       ✓ fails if random number was not updated (971ms)
       ✓ should reward if only 1 participant (1139ms)
       ✓ should complete 10 rounds (12733ms)
       ✓ should draw the jackpot (3361ms)
       ✓ should confirm participants chances (381153ms)
    setRoundDuration
       ✓ should set (175ms)
       ✓ fails if not an owner (66ms)
       ✓ fails if wrong value (281ms)
    setFee
       ✓ should set (116ms)
       ✓ fails if not an owner (59ms)
       ✓ fails if wrong value (82ms)
    setFeeReceiver
       ✓ should set (124ms)
       ✓ fails if not an owner (62ms)
       ✓ fails if wrong value (136ms)
    setJackpotShare
       ✓ should set (87ms)
       ✓ fails if not an owner (88ms)
       ✓ fails if wrong value (84ms)
    setJackpotChance
       ✓ should set (150ms)
       ✓ fails if not an owner (60ms)
       ✓ fails if wrong value (55ms)
    setExecutorShare
       ✓ should set (135ms)
       ✓ fails if not an owner (65ms)
       ✓ fails if wrong value (104ms)
    setPrizeSizes
       ✓ should set (193ms)
       ✓ fails if not an owner (198ms)
       ✓ fails if wrong value (476ms)
    setBlockTime
       ✓ should set (76ms)
       ✓ fails if not an owner (59ms)
       ✓ fails if wrong value (101ms)
    setMinDeposit
       ✓ should set (125ms)
       ✓ fails if not an owner (64ms)
       ✓ fails if greater than max deposit (211ms)
    setMaxDeposit
       ✓ should set (144ms)
       ✓ fails if not an owner (50ms)
       ✓ fails if less than min deposit (249ms)


  51 passing (8m)
```

Code Coverage

The test coverage of the codebase is good. The uncovered branches and statement have a minimal impact on the quality of the test suite.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.14	93.33	96.3	98.15	
DrawManager.sol	100	100	100	100	
IPOSDAORandom.sol	100	100	100	100	
PoaMania.sol	98.33	97.83	95	98.35	447,471
Random.sol	95	75	100	95	57
Sacrifice.sol	100	100	100	100	
contracts/mocks/	100	100	100	100	
RandomMock.sol	100	100	100	100	
ReceiverMock.sol	100	100	100	100	
All files	**98.26**	**93.55**	**96.83**	**98.27**	

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

```
01c571355f45ab0db20ded6eb4da8c90fefd19f48688fd1a7678b9fa33a091e6 ./contracts/Sacrifice.sol

822864724eb6b87bf9fd937b85f80d76f6561fee73dfcd96248f9e3a15c9dda4 ./contracts/Random.sol

254ec4dae124551ef7545f14bb160ccb6e8f16194e30558e116747dcbe9a1a61 ./contracts/DrawManager.sol

7a617b2d16e8cf6370b85ff4107968a3014afbaaf0cea177551714277fdfbd71 ./contracts/PoaMania.sol

d0ed140fffbd1a31a7e42b58ece2b6ce33d7ab55319d6d369abc428ed8f0df0a ./contracts/IPOSDAORandom.sol

a6bf2f4a3fd220fcd4d48fae147c138f9481d4668953db1467da4f1838cd414c ./contracts/mocks/RandomMock.sol

a1e5f0af930978a77ea28f29523830c1523af527aaf7825c3c89e48bc6ac1f20 ./contracts/mocks/ReceiverMock.sol
```

Tests

```
f460e6f40fcc83f8cac4727bcf6eb4add36ed0caec6a624f99290cc23a350fc7 ./test/PoaMania.test.js
```

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure smart contracts at scale using computer-aided reasoning tools, with a mission to help boost adoption of this exponentially growing technology.

Quantstamp's team boasts decades of combined experience in formal verification, static analysis, and software verification. Collectively, our individuals have over 500 Google scholar citations and numerous published papers. In its mission to proliferate development and adoption of blockchain applications, Quantstamp is also developing a new protocol for smart contract verification to help smart contract developers and projects worldwide to perform cost-effective smart contract security audits.

To date, Quantstamp has helped to secure hundreds of millions of dollars of transaction value in smart contracts and has assisted dozens of blockchain projects globally with its white glove security auditing 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.

Finally, Quantstamp's dedication to research and development in the form of collaborations with leading academic institutions such as National University of Singapore and MIT (Massachusetts Institute of Technology) reflects Quantstamp's commitment to enable world-class smart contract innovation.