POA Network Contract Audit

December, 2017

Alexander Wade

Contents

1. Overview

This document serves as the official audit report of a set of network validator and consensus contracts, created and published by PoA Network. PoA Network seeks to create the first Ethereum-based public network with PoA consensus, where validation of transactions is managed by trusted (and publicly identified) validators. Consensus is aided by a governance-based penalty system, where the current list of validators is managed by an on-chain smart contract managed through voting. Each validator must have a public notary license and be located in the United States, ensuring that their identities (stored on-chain) can be validated by any user of the network. Additionally, validators will be independent and not affiliated with each other in any way. Through these measures, the PoA network will secure a fair consensus.

The PoA network itself supports on-chain governance as well as several “Proof-of-Identity” Oracles. PoA Network believes that this system will help close the gap between public and private networks, creating a model for open networks using PoA consensus.

The project audited is the backbone for this on-chain governance system: a series of contracts that will be deployed to the main Ethereum network, and will contain validator identification data, as well as various voting mechanisms to allow a majority of validators to secure the network. The contracts are highly modular as well and feature many voting-run upgrade mechanisms, allowing most of the contracts in the project to be changed to accommodate different situations as the network grows.

These features are incredibly powerful, allowing the network to grow with its userbase, as well as change to accommodate new features. The voting mechanisms act as a failsafe as well – if an issue is discovered, validators can vote to use new contracts to fix the issue.

2. Introduction

2.1 Authenticity

The audited contracts are in the OraclesOrg poa-network-consensus repository: https://github.com/oraclesorg/poa-network-consensus-contracts. The version used for this audit is commit **31209cd154d35f2040ff2b894e8b98b801b416de**. The main contract for this audit is located in PoaNetworkConsensus.sol. The main contract for managing validator keys is located in KeysManager.sol. The main contract for managing other contract addresses is ProxyStorage.sol.

2.2 Scope

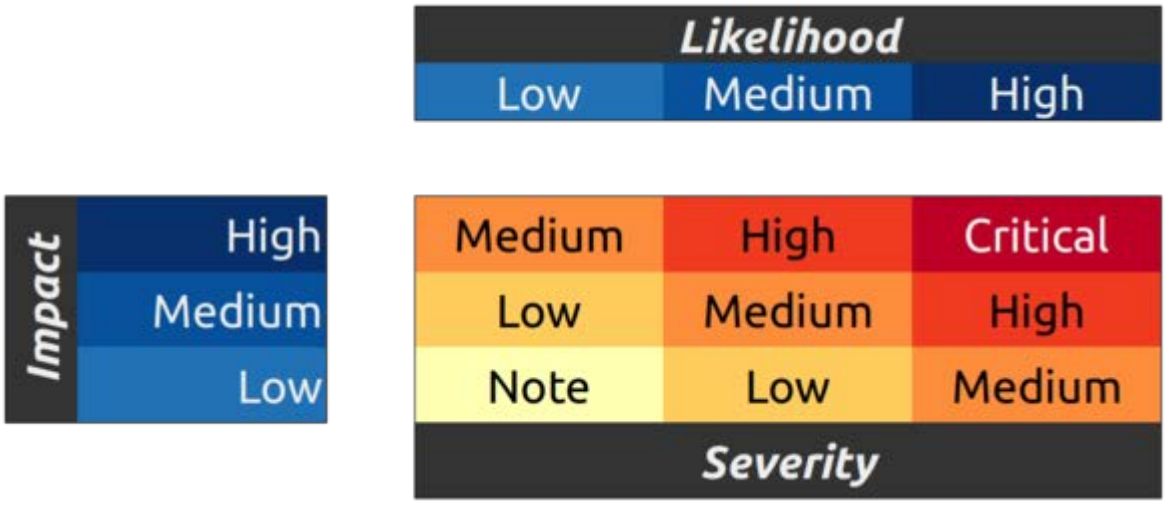
This audit covered all of the solidity files located in the contracts folder of the poa-network-consensus repository: https://github.com/oraclesorg/poa-network-consensus-contracts/tree/master/contracts. This audit does not cover files located in any other folder.

2.3 Methodology

This audit focuses heavily on not only inspecting the smart contracts for vulnerabilities and potential for losses in funds, but also on working closely with the PoA Network team to scrutinize the contracts for execution of intent. The end goal of this audit is to help the team not only secure their contracts, but also to ensure their vision for the project is best represented by the project they put forward. As a result, additional concerns such as efficiency and design are included in this report as well.

2.4 Terminology

This audit categorizes vulnerabilities using the OWASP risk rating method based on impact and likelihood. Each vulnerability is given impact and vulnerability scores, which are used to give a more accurate estimation of a vulnerability's overall severity. An additional factor in severity is the relative ease with which a vulnerability is fixed: an issue which requires extreme refactoring will be weighted higher than one with the same severity which is a quick fix.



2.5 Disclaimer

This document reflects the understanding of security flaws and vulnerabilities as they are known to the Authio team, and as they relate to the reviewed project. This document makes no statements on the viability of the project, or the safety of its contracts. This audit is not intended to represent investment advice and should not be taken as such.

3. Findings

3.1 General

The PoA Network team has undertaken a difficult project with their unique on-chain governance system. The resulting smart contracts were complicated and featured several cross-contract interactions, creating a lot of potential for issues and vulnerabilities. However, the PoA Network team was quick to respond to any concerns raised, and fixes for any issues found were quickly implemented, tested, and approved.

The quality of the contracts audited is very high, featuring incredible amounts of modularity and portability, ensuring that new contracts added to the project will have very standardized interfaces with which to interact. The codebase itself it clean, and well-organized, although it lacked detailed commenting throughout.

3.2 Contract Explanation

3.2.1 Function – Validator Management

PoaNetworkConsensus.sol: This contract houses the logic for adding and removing validators in the network. The KeysManager contract calls functions here to add and remove validators, and the PoA Network system address uses this contract to finalize changes made to active validators.

3.2.2 Function – Validator Multi-Key Management

KeysManager.sol: This contract allows new validators to initialize their mining, voting, and payout keys. It also interacts with voting contracts to add, remove, and swap out keys both in KeysManager and PoaNetworkConsensus.

3.2.3 Function – Validator Identity Management

ValidatorMetadata.sol: This contract keeps track of the public identities of validators, storing various pieces of information including name, public identity, and location. It also allows validators to request changes to their metadata, which is done through voting in-contract.

3.2.4 Function – Contract Upgrades

ProxyStorage.sol: This contract keeps a record of which contracts associated with the POA network are at which addresses. Validators can submit and vote on ballots in VotingToChangeProxyAddress to change these addresses.

3.2.5: Function – Voting

BallotsStorage.sol: This contract serves as a storage containing the number of votes required to pass various types of ballots. It is written to be highly modular: if new types of ballots are created, the BallotsStorage contract can accommodate them.

VotingToChangeKeys.sol: This contract allows validators to create ballots to request key changes, additions, or removal.

VotingToChangeMinThreshold.sol: This contract allows validators to create ballots to change the number of votes required to pass key-change ballots.

VotingToChangeProxyAddress.sol: This contract allows validators to create ballots to change the addresses of other contracts stored in BallotsStorage.

3.3 Critical Severity

Validator Ballot Creation Blocking: This issue allows a validator to completely block any ballot creation. Because the maximum number of active ballots is capped at 101, a validator can repeatedly create ballots until that number is reached, at which point no more ballots can be created. This issue becomes more serious with the fact that the validator can create ballots which are not finalizable by setting start and end dates far into the future. The end result is a complete block on ballot creation until the ballots can be finalized and removed. Because removing a validator requires a ballot to be passed, a validator can safeguard against their removal by making it impossible to create a ballot petitioning for their removal.

Because this vulnerability completely breaks the project's intended upgradability, and because the vulnerability is simple to exploit (albeit tedious), this vulnerability has high impact and likelihood. As a result, it has been marked as critical severity.

The vulnerability can be avoided either by removing a cap on the number of active ballots, or by setting an active ballot limit per validator. The POA Network team opted to set an active ballot limit per validator that depended on the number of current active validators, as well as raising the maximum active ballot cap to 200. This vulnerability was **fixed in commit ab48864**.

Validator Overriding New Ballots: This vulnerability allows a validator to override changes made in a passed ballot by finalizing a ballot that was already passed and finalized. A validator can call the finalize function on an already-passed ballot and because of a faulty check to see if the ballot is already finalized, the ballot is able to make it to the finalizeBallot function, pass all checks, and enact the update the ballot was initially supposed to make.

For example, if Ballot A changes contract address X to contract address Y, and Ballot B changes contract address Y to contract address Z, anyone can call finalize on Ballot A again, and change contract address Z back to Y. This can happen indefinitely, as any ballot can be re-finalized. This easy-to-execute issue goes around the entire premise of voting and can cause widespread unwanted side effects and unpredictable behavior. As a result, this issue was marked as critical severity.

This vulnerability is fixable by correctly checking whether a ballot is already finalized before allowing the finalize function to be called. This issue was **fixed in commit 7bce83d.**

3.4 High Severity

No high severity issues were found.

3.5 Medium Severity

Multiple Voting Through Circular Mining History Manipulation: A validator can change their mining key to another mining key by creating a ballot in VotingToChangeKeys.sol. Each time a mining key is swapped, the new key is mapped to the old key in KeysManager in the miningKeyHistory mapping. The idea behind this tracking is to avoid validators switching their mining keys to vote twice on a ballot. However, through a series of mining key ballots, a validator can create a circular mining history that cuts an old mining key out of the history, allowing them to vote twice on a ballot. The manipulation is as follows:

1. A validator with mining key A places a vote on Ballot X, then requests a mining key change. The mining key change is accepted, and the validator now has mining key B. Attempting to vote on Ballot X again, the validator is stopped because A is in B's history, and A already voted on Ballot X:

History(B): B => A => 0x0

2. The same validator now requests a mining key change, to key C. The change is accepted. Attempting to vote on Ballot X again does not work because A is in the history of C:

History(C): C => B => A => 0x0

3. The validator requests a mining key change again, back to key B. The new mining key history is circular, and does not include A:

History(B): B => C => B => C => B => …

The validator is now able to use mining key B to vote on Ballot X for the second time.

This attack requires 3 consecutive mining key change requests to be voted on and accepted, which is a large number, but is definitely possible. While the

impact of this attack is high, allowing a validator a second vote on a potentially important ballot, the likelihood that a validator will successfully change their key 3 times before Ballot X is finalized is low. As a result, this issue was given medium severity.

We suggested that validators not be allowed to switch their mining keys to previously-held keys. The POA Network team agreed, and **fixed the issue in commit 6e3fc0a.**

Unsafe Ballot Deactivation and Removal: This issue was due to unsafe deletion methods when removing a finished ballot. When a ballot was deleted, the length of the activeBallots array was decreased by one, and the ballot to be deleted was set to default values by calling delete on its current position in the activeBallots array. Decreasing the length of the array by one, however, simply removes the last ballot in the activeBallots array – which was never guaranteed to be the ballot intended to be removed. As a result, the most recently created ballot would be removed every time an old ballot was finalized.

While this did not have high impact, it happened very frequently, and so was denoted medium severity. We recommended swapping the last ballot in the activeBallots array to the position of the ballot marked for removal, and then decrementing the length. The POA Network team agreed, and the issue was **fixed in commit 60c1fcf.**

3.6 Low Severity

Single Validator Metadata Change Request Confirmation: This issue allows a validator to confirm a metadata change request for another validator regardless of the number of signatures required. The function to confirm a change did not check to see if the sender had already voted on the change request, and so

validators would be able to send multiple calls to the change confirmation function to confirm metadata change without input from other validators.

This issue was simply executed, but carried very little consequences, as metadata is not inherently needed to run the network, and the offending parties would be noticed and removed though a separate vote. As such, the issue was given low severity, and **fixed with a simple check in commit 8389e69.**

CreateMetadata Bypass: In ValidatorMetadata, the expected behavior is that a validator will set their metadata by calling createMetadata (which can only be done once), and by calling changeRequest to submit a vote to change metadata. However, changeRequest does not check to see if a validator has called createMetadata, and will allow for the submission of a change request regardless. If the change request is voted in, the validator is then able to call createMetadata at any point in time to get a 'free' metadata change without needing to pass a vote first.

This issue has few negative consequences, none of which are necessary for the function of the POA network. As such, the issue was marked as low severity. The POA Network team opted not to fix this issue, as metadata changes have little to no impact on the security of the network, and bad actors can be removed through votes.

3.7 Notes & Recommendations

Many locations in the code should use the 'memory' keyword, because state is not modified:

The areas that can use 'memory' are denoted as such in the commented code, supplied at the bottom of this document.

Ensure code fails quickly: At multiple points in the code, require statements are located after several operations have already been performed. Placing require statements early in functions ensures that code will fail as quickly as possible. Exact locations are marked as such in the commented code, supplied at the bottom of this document.

Unnecessary time checking: In each voting contract, creating a ballot requires startTime and endTime to be greater than 0. However, the next check makes this superfluous, as startTime and endTime are both checked again block.timestamp, and each other.

Unnecessary checking of affectedKeyType: The affectedKeyType of a key add, remove, or swap ballot will only ever be one value. Checking for each type using multiple if statements wastes gas. Consider using an if-else structure instead.

Unnecessary complication: for loop: Placing a for loop in the PoaNetworkConsensus constructor here is unnecessary, because there will only ever be one item in the array to loop over.

Unused variable: votingKey: This modifier meant to use _votingKey, but instead retrieves a mining key for msg.sender. In the context of the code, the effect is the same, but could be confusing. Additionally, this creates a compiler warning.

4. Documents & Resources

4.1 Line-By-Line Comments

Line-by-line commenting can be found in the EthereumAuthio github repository:

https://github.com/EthereumAuthio/Audits/tree/master/POANetwork

4.2 Project Code

The current POA Network Consensus contracts can be found in their public github repository: **https://github.com/oraclesorg/poa-network-consensus-contract**

5. Conclusion

The Authio team would like to commend the POA Network team on a well-implemented idea. The project underwent extensive testnet testing prior to the audit, and was reviewed by two independent third-party auditing bodies. The Authio team recommends that for the next project, the POA Network team leaves enough time before deployment to place a public bug bounty. Bug bounties are by no means a replacement for an audit, as participants are incentivized to rush through the contracts to find issues, but they are an excellent final sanity check: more eyes are always better. For upcoming projects, it would be wise to include auditors in the consulting, ideation, and building process for contracts. When security becomes part of the whiteboarding process, issues can be caught before they become problems down the line.

The POA Network has undertaken a difficult project with their unique idea. Their code was complicated and had many contracts interacting to create a lot of potential for vulnerability. Generally, a restructuring of the code would have prevented some of these issues from cropping up in the first place. However, after extensive review, and after working closely with the POA Network team, all issues of note have been fixed to the satisfaction of the Authio team.