PUBLIC

Code Assessment

of the POSDAO Smart Contracts

June 25, 2021

Produced for

by

Contents

1 Executive Summary

Dear Sir or Madam,

First and foremost we would like to thank POA Network for giving us the opportunity to assess the current state of their POSDAO system. This document outlines the findings, limitations, and methodology of our assessment.

Throughout the project, we greatly appreciated working with your team, whom are both collaborative and available to discuss the issues at hand. It was clear from our interactions that your team is committed to quality, and this shows in the smart contract code which favors security above efficiency. We were impressed with the well structured documentation your team provided, and suggest in the future to invest further in the corner case specifications as this would help to clarify the expected behavior of the system. Our assessment uncovered a few issues, but no "critical" severity security flaws which would prevent the launch of the smart contracts. We found one "high" severity security flaw: 6.1 - EIP-170 Mix Up / Unlimited Contract Size.

We hope that this assessment provides more insight into the current implementation and provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	1
• Risk Accepted	1
Medium-Severity Findings	7
• Code Corrected	1
• Specification Changed	2
• Risk Accepted	2
• Acknowledged	2
Low-Severity Findings	11
• Code Corrected	7

• Risk Accepted	1
• Acknowledged	3

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the POSDAO repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	30 April 2021	34552ccdbfc148ae48a7a6de44e1b435ff21f5f8	Initial Version
2	10 June 2021	9de3a94be98d9553c6ac72916221b42fc56daec2	After Intermediate Report

The smart contracts allow for a custom configuration of the network. All production smart contracts in the repository and the commits listed above were part of the audit scope, however, the main focus was on the configuration for the xDAI POSDAO AuRa implementation.

2.1.1 Excluded from scope

For the solidity smart contracts, the compiler version `0.5.10` was chosen. The outdated compiler version has been explicitly used by POA Network so the compiler version to remain consistent across the project and is not subject of the audit.

We checked that the random aura smart contract implements the algorithm. However, the guarantees provided by the alogirthm and in particular the randomness provided by the algorithm is out of scope. The technical implementation of the `RandomAura` smart contract was in scope, the RANDAO methodology however is out of scope.

The `Governance` contract has been added to the repository after the intermediate report and is not part of this assessement.

3 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Introduction

The smart contracts reviewed implement the configurable logic for the operation of a POSDAO network. The actual configuration implemented corresponds to the settings for the xDAI POSDAO AuRa network. These smart contracts are used by the client software (currently OpenEthereum or Nethermind) to determine how to run the proof of stake network. Amongst others, this includes the logic to determine the set of active validators and the block rewards. The client software is configured accordingly through the genesis configuration of the chain and the core smart contracts expose standardized functions which the

client queries. A staking contract deployed on chain allows participants to stake (either the native coin of the chain or tokens, depending on configuration) and to participate in the consensus.

Further utility contracts are:

- Registry: Matches addresses with human readable keys
- TxPriority: Individual smart contracts at each node, allow each node to configure the priority of transactions to be included.

Most of the smart contracts function independently, some have a loose connection. Following, all deployed smart contracts are presented in detail:

3.1 Registry

DNS-Like registry associates human readable information with addresses.

Anyone may reserve any free name. Reserving a name costs a fixed fee. The owner of an entry can set it's data, drop or transfer the ownership to another address. To set an address for an entry (called the reverse) the owner first proposes the address using `proposeReserve()` before the address confirms it by calling `confirmReserve()`. As contracts may be unable to do software `confirmReserveAs()` allows the owner of the contract (not the owner of the entry) to manually confirm an entry.

The contract's owner can set the fee & collect the Ether collected.

3.2 TxPriority

This is a non-consensus feature. Every Validator node may run their own `TxPriority` contract in order to determine the order of transactions to be included at their node. A priority may be set based on the `to` and `data` field of a transaction. Additionally, a minimum gas price for destination / function signature combination may be set.

There is a whitelist for `sender` addresses, transactions originating from such an address have highest priority to be included irregardless of other settings.

3.3 TxPermissioning

Controls the use of zero gas price in service transactions alongside the `Certifier` contract. It defines a function `allowedTxTypes`. Two version exists:

TxPermissioningV3: The allowed Tx types depend on: `_sender`, `_to`, `_value`, `_gasPrice` and `_data`. TxPermissioningV4: In preparation for EIP-1559, the allowed tx types additionally may depend on `_maxFeePerGas`, `_maxInclusionFeePerGas` and `_gasLimit` instead of the gas price.

3.4 Certifier

This upgradable contract allows validators to use a zero gas price for their service transactions.

The contract owner can set addresses to be whitelisted. The function `certified()` can be used to query the status of an address. The function returns true if either the address has been explicitly whitelisted by the owner or `validatorSetContract.isReportValidatorValid(_who, true)` returns `true`.

3.5 ValidatorSetAuRa

Stores the current validator set and contains the logic for choosing new validators at the beginning of each staking epoch.

3.6 StakingAuRa

Contains staking logic including pool management withdrawing moving stakes.

3.7 BlockRewardAuRa

Generates and distributes rewards (block rewards and bridge rewards) to the pools and splits it for delegators and validators.

3.8 RandomAuRa

Allows validators to commit and reveal random numbers generated off-chain. Aggregates and these numbers in a RANDAO manner.

3.9 Configurations

Albeit the reviewed smart contracts where configured for the operation of the xDai POSDAO AuRa implementation, the smart contracts allow the configuration to be heavily customized.

Staking can be done in either Tokens or the native coin, depending on the configuration in contract `StakingAuRa`. Block reward can be paid in either Tokens or the native coin, depending on the configuration in contract `BlockRewardAura`

The repository contains implementation code for these different options.

The chain may be started from genesis or forking of an already existing AuRa chain.

3.10 Roles

Most functions are permissioned and require the caller to bear a specific role. Due to the complexity and upgradeability of the projects, the admin role need to be carefully distinguished as their are multiple contracts that can have an admin. The admin is supposed to be a multi signature contract with accordingly well chosen signers.

The `ERC677BridgeTokenRewardable` contract has an owner which is the account deploying the contract. This account can mint directly and indirectly by setting a malicious. Hence, the owner needs to be fully trusted. The owner needs to either the `TokenMinter` contract or another address that only temporarily has the owner power. This is because some of the functions are not callable from the `TokenMinter` contract if it is not the owner but simultaneously some of the functions cannot be called but the `TokenMinter` and require another address to which the admin power is transferred temporarily.

The `TokenMinter` contract has a owner that is defined when the contract is deployed. `claimTokens`, `setBridgeContract` and `transferTokenOwnership` fail if the contract is not also the owner of the token contract. Hence, `transferTokenOwnership` renders the `TokenMinter` functions useless unless ownership is transferred back to the minter contract. The owner can also define minting addresses that are allowed to call `mint` which is called in the token contract and also only works as long as the `TokenMinter` contract is the owner of the token contract.

`TxPriority` has an owner which is set when the contract is deployed and no proxies.

All other contracts listed with a * are the implementation contract for a proxy contract. The proxy has an admin which, by default, is the deployer. The functions with an added * are functions of the proxy contract.

Contract	Role	Power to call
ERC677BridgeTokenRewardable	owner	• transferOwnership • claimTokens • addBridge • removeBridge • setBlockRewardContract • setStakingContract • mint (indirectly) • stake (indirectly) • mint (directly)
	BlockRewardContract	• mintReward
	StakingAuRaContract	• stake
TokenMinter	owner	• addMinter • removeMinter • claimTokens • setBlockRewardContract • setBridgeContract • transferOwnership • transferTokenOwnership
	BlockRewardContract	• mintReward
	minter	• mint
Certifier*	admin	• certify • revoke • changeAdmin* • upgradeTo* • upgradeToAndCall*

RandomAura*	admin	• setPunishForUnreveal • changeAdmin* • upgradeTo* • upgradeToAndCall*
	miningAddress	• commitHash • revealNumber • revealSecret
	ValidatorSetContract	• clearCommit
	BlockRewardContract	• onFinishCollectRound
TxPriority	owner	• transferOwnership • setPriority • removePriority • setSendersWhitelist • setMinGasPrice • removeMinGasPrice
ValidatorSetAuRa*	owner	• clearUnremovableValidator • changeAdmin* • upgradeTo* • upgradeToAndCall*
	blockRewardContract	• newValidatorSet
	randomContract	• removeMaliciousValidators
	stakingContract	• addPool
	System	• finalizeChange
	unremovableStakingAddr	• clearUnremovableValidator
	stakingAddress	• changeMiningAddress • changeStakingAddress • changeMetadata

BlockRewardAuRaTokens*	owner	• setErcToNativeBridgesAllowed • setTokenMinterContract • setErcToErcBridgesAllowed • setNativeToErcBridgesAllowed • changeAdmin* • upgradeTo* • upgradeToAndCall*
	ErcToNativeBridge	• addBridgeNativeFeeReceivers • addBridgeNativeRewardReceivers • addExtraReceiver
	System	• reward
	ValidatorSetContract	• clearBlocksCreated
	stakingContract	• transferReward
	XToErcBridge	• addBridgeTokenFeeReceivers • addBridgeTokenRewardReceivers
BlockRewardAuRaCoins*	owner	• setErcToNativeBridgesAllowed • changeAdmin* • upgradeTo* • upgradeToAndCall*
	ErcToNativeBridge	• addBridgeNativeFeeReceivers • addBridgeNativeRewardReceivers • addExtraReceiver
	System	• reward
	ValidatorSetContract	• clearBlocksCreated
	stakingContract	• transferReward

StakingAuRaTokens*	owner	• initialValidatorStake • setCandidateMinStake • setDelegatorMinStake • setErc677TokenContract • changeAdmin* • upgradeTo* • upgradeToAndCall*
	ValidatorSetContract	• clearUnremovableValidator • incrementStakingEpoch • removePool • removePools • setStakingEpochStartBlock
	stakingAddress	• removeMyPool
	erc677TokenContract	• onTokenTransfer
StakingAuRaCoins*	owner	• initialValidatorStake • setCandidateMinStake • setDelegatorMinStake • changeAdmin* • upgradeTo* • upgradeToAndCall*
	ValidatorSetContract	• clearUnremovableValidator • incrementStakingEpoch • removePool • removePools • setStakingEpochStartBlock
	stakingAddress	• removeMyPool
	erc677TokenContract	• onTokenTransfer
TxPermission*	owner	• addAllowedSender • removeAllowedSender • setDeployerInputLengthLimit • setSenderMinGasPrice • changeAdmin* • upgradeTo* • upgradeToAndCall*

Registry has an own owner implementation no Proxy and no special owner permissions.

3.11 Assumptions

- All software (not including the reviewed smart contracts) and hardware on the machine works as intended, acts non-malicious and is secure
- The proxy owner role of the smart contracts is fully trusted
- Correct deployment of the smart contracts
- The randomness created by the node is sufficient

3.12 Address Setup

Each validator controls two addresses with the correponding private keys. These are:

- Mining keys
- Staking keys

4 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third party infrastructure which adds further inherent risks as we rely on the correct execution of the included third party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severities. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

6 Findings

In this section, we describe any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of the findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1

- EIP-170 Mix Up / Unlimited Contract Size Risk Accepted

Medium-Severity Findings	4

- Changing Mining and Staking Addresses While Banned Risk Accepted
- Incoherent Event ChangedMiningAddress Emitted Acknowledged
- Limitations of the TxPermissions Contract Risk Accepted
- Role Switch Needed Acknowledged

Low-Severity Findings	4

- Gas Inefficiency During Removal From Array Acknowledged
- Inconsistent Use of Safemath Risk Accepted
- Potentially Compromised Key Needed to Change Key Acknowledged
- Superfluous Call of _finalizeNewValidators Acknowledged

6.1 EIP-170 Mix Up / Unlimited Contract Size

Correctness High Version 1 Risk Accepted

EIP-170 has been introduced into the Ethereum mainnet with the Spurious Dragon hardfork in order to limit the maximum codesize of a contract.

The short specification of the EIP reads:

> ... if contract creation initialization returns data with length of more than 0x6000 (2**14 + 2**13) bytes, contract creation fails with an out of gas error.

The data returned by the contract creation initialization is the code of the newly deployed smart contract that will be stored as the code of the smart contract. This is valid regardless wether the contract has been deployed directly from a transaction or a during code execution of a `CREATE` / `CREATE2` opcode. For more details please refer to chapter 7 of the Ethereum Yellowpaper.

The `TxPermissionBased` contract in the POSDAO system attempts to enforce a `_deployerInputLengthLimit`. There is an annotated function for the owner to set this variable:

```
/// @dev Sets the limit of `input` transaction field length in bytes
/// for contract deployment transaction made by the specified deployer.
/// @param _deployer The address of a contract deployer.
```

```
/// @param _limit The maximum number of bytes in `input` field of deployment transaction.
/// Set it to zero to reset to default 24Kb limit defined by EIP 170.
```

And inside the `_allowedTxTypes` function which is annotated with:

```
/// @dev Defines the allowed transaction types which may be initiated by the specified sender with
/// the specified gas price and data. Used by node's engine each time a transaction is about to be
/// included into a block.
```

there is:

```
if (_to == address(0) && _data.length > deployerInputLengthLimit(_sender)) {
    // Don't let to deploy too big contracts
    return (NONE, false);
}
```

There is a mixup here: What the `TxPermission` contract actually limits with this parameter is the lenght of the `data` field of the transaction, not the limit of a contract's code size. This has nothing to do with EIP-170. Hence if the limit is only "enforced" by the `TxPermission` contract and there is no further limit set in the chain specification anyone may deploy a contract of arbitrary size, limited only by the gas limit. EIP-170 is not activated in the `template/spec.json` chain sepcification file available in the repository.

Note that the Ethereum mainnet has no excplicit limit on the data field of a transaction (called *input* in the function description in POSDAO). This is only limited by the gas limit of a block.

Ethereum Yellowpaper: https://ethereum.github.io/yellowpaper/paper.pdf
EIP-170 Specification: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-170.md

---

Risk Accepted:

POA Network accepts this risk and states: `Some popular projects on xDai require the ability to deploy contracts with size greater than 24 Kb. The limit on transaction size is intended as an easy protection against script kiddies.`

6.2 Changing Mining and Staking Addresses While Banned

Design | Medium | Version 1 | Risk Accepted

`ValidatorSetAuRa` allows to change the mining and staking address while a pool is banned. This updates the state, including:

```
idByStakingAddress[oldStakingAddress] = 0;
idByStakingAddress[_newStakingAddress] = poolId;
```

or

```
idByMiningAddress[_oldMiningAddress] = 0;
idByMiningAddress[_newMiningAddress] = _poolId;
```

The available specification does not cover this scenario and it remains unclear if this should be possible or not.

In case of a change of the mining address while a pool is banned, the return value of following functions may be unexpected for the caller:

```
/// @dev Returns the block number when the ban will be lifted for the specified mining address.
/// @param _miningAddress The mining address of the pool.
function bannedUntil(address _miningAddress) public view returns(uint256) {
    return _bannedUntil[idByMiningAddress[_miningAddress]];
}
```

`bannedUntil()` will return 0 if the mining address of the banned pool has been changed even though the pool is banned.

```
function isValidatorBanned(address _miningAddress) public view returns(bool) {
    uint256 bn = bannedUntil(_miningAddress);
    if (bn == 0) {
        // Avoid returning `true` for the genesis block
        return false;
    }
    return _getCurrentBlockNumber() <= bn;
}
```

This holds similarly for this function which notably is querried by `BlockRewardAuRaBase.reward()`.

Within the system one such address can only be used once for an unique purpose, e.g. an address that has been a mining or staking address once cannot be reused anymore.

This is tracked by following mappings:

```
mapping(address => uint256) public hasEverBeenMiningAddress;
mapping(address => bool) public hasEverBeenStakingAddress;
```

The information to which pool the mining address once belonged to is availabe in this mapping.

Risk accepted:

POA Network states this is expected behavior in order to allow pools to change their staking or mining address if they are compromised during the ban period.

6.3 Incoherent Event `ChangedMiningAddress` Emitted

Design | Medium | Version 1 | Acknowledged

To change a mining address, `changeMiningAddress` is called from the participants staking address. If the participant is a current validator, the change is not done immediately. This emits the `InitiateChange`. Additionally, the function will always emit the `ChangedMiningAddress` event. Given the name of the event and that it is also emitted when the mining address is changed immediately because the participant is not part of the current validator set, this seems incoherent. As the event name suggests, the event should be emitted only when the mining address is changed or maybe renamed.

Acknowledged:

POA Network is aware that the `ChangeMiningAddress` event only corresponds to the immediate change of the mining address when a pool is not a validator. Unfortunately no events can be emitted at

the moment of the real change for the delayed case inside the system's `finalizeChange` function as events cannot be emitted during execution of this system operation.

6.4 Limitations of the `TxPermissions` Contract

Design | Medium | Version 1 | Risk Accepted

The `_allowedTxTypes` function of the `TxPermissions` contract is applied to all transactions to be include into a block. However this means all checks are only done on external transactions created from externally owned accounts, internal transactions (calls within transactions) are not subject to these checks.

Some of these checks including e.g.

```
if (validatorSetContract.isValidator(_to)) {
    // Validator's mining address can't receive any coins
    return (NONE, false);
}
```

can be circumvented by internal transaction. Internal transactions are calls from within bytecode execution, e.g. during execution of a smart contract.

Risk Accepted:

POA Network is aware that the rules defined by the `TxPermissions` contracts are only applied to transactions of EOAs.

6.5 Role Switch Needed

Design | Medium | Version 1 | Acknowledged

The `TokenMinter` contract calls permissioned token contract functions. These are `mint`, `setBridgeContract`, `transferOwnership`. To successfully call these functions, the `TokenMinter` contract needs to be the owner of the `ERC677MultiBridgeToken` contract.

Regarding the `setBridgeContract` we have opened a separate issue because this call will always fail. But the `ERC677MultiBridgeToken` contract also implements other functions that are permissioned to be called only by the owner. Given the `TokenMinter` contract is the owner these functions could not be called. These functions are: `addBridge`, `removeBridge`, `setBlockRewardContract`, `setStakingcontract`.

To call this functions, the ownership needs to be transferred from the minter contract to an other contract and then back. This seems undesirable.

Acknowledged:

POA Network explains that the TokenMinter contract is used as an intermediate owner contract for the PermittableToken contract wich represents the STAKE token. To clarify this, comments where added to the TokenMinter contract.

6.6 Gas Inefficiency During Removal From Array

Design Low Version 1 Acknowledged

The staking contract keeps track of the pools using multiple arrays. When an entry has to be removed, this is done as in the following example:

```
uint256 indexToDelete = poolToBeRemovedIndex[_poolId];
    if (_poolsToBeRemoved.length > indexToDelete && _poolsToBeRemoved[indexToDelete] == _poolId) {
        uint256 lastPool = _poolsToBeRemoved[_poolsToBeRemoved.length - 1];
        _poolsToBeRemoved[indexToDelete] = lastPool;
        poolToBeRemovedIndex[lastPool] = indexToDelete;
        poolToBeRemovedIndex[_poolId] = 0;
        _poolsToBeRemoved.length--;
    }
```

In case that the removed entry was already last in the list two SSTORE and one SLOAD operation could have been skipped.

Acknowledged:

Client states that this operation is quiet rare and, hence, will not change the implementation.

6.7 Inconsistent Use of Safemath

Design Low Version 1 Risk Accepted

The code has multiple calculations including multiplications and divisions without safemath. Even though we could not find a place where we think calculation would over or underflow, the consistent use of safemath would ensure this.

Risk accepted:

Safe math was not used intentionally in critical functions to not cause reverts and risk a network break down. Hence, POA network accepted the risk.

6.8 Potentially Compromised Key Needed to Change Key

Design Low Version 1 Acknowledged

To change a potentially compromised staking key, the staking key is needed. Even though, the mining key is not used for tasks like key changes, in this case it might make sense from a security perspective. One reason to change a key is that it might be corrupted. In this case, it might be safer to use an other already existing key to change it.

Acknowledged:

POA network wants to keep the strong separation regarding the key usage.

6.9 Superfluous Call of `_finalizeNewValidators`

Design | Low | Version 1 | Acknowledged

`changeMiningAddress` sets `_finalizeValidators.list` to the unedited `_pendingValidators`. In `finalizeChange` this causes the else if condition to be true and triggers `_finalizeNewValidators`. `_finalizeNewValidators` first removes all validators and then adds the same. This seems unnecessary. Additionally, the comment suggest another use case for the else if.

Acknowledged:

POA network acknowledged the issue but decided to leave the code unchanged.

7 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	3

- Failing Function Call Specification Changed
- No Canonical Definition of Calldata for onTokenTransfer Specification Changed
- claimOrderedWithdraw Not Always Successful Code Corrected

Low-Severity Findings	7

- Incorrect Comment in finalizeChange Code Corrected
- Incorrect Description Code Corrected
- Make onTokenTransfer() External Code Corrected
- Multiplication After Division Code Corrected
- No Indexed Fields for ReportedMalicious Code Corrected
- Unchecked Return Value of Transfer Code Corrected
- certify Missing Sanity Check Code Corrected

7.1 Failing Function Call

Design Medium Version 1 Specification Changed

The `TokenMinter` contract implements the function `setBridgeContract` which should call `tokenContract.setBridgeContract`. The `setBridgeContract` function does not exists in the `ERC677MultiBridgeToken` contract. Hence, the function call would fail and the interface definition at the beginning is incorrect.

Specification changed:

POA Network explains that the TokenMinter contract is used as an intermediate owner contract for the PermittableToken contract wich represents the STAKE token. To clarify this, comments where added to the TokenMinter contract.

7.2 No Canonical Definition of Calldata for `onTokenTransfer`

Correctness Medium Version 1 Specification Changed

The function `onTokenTransfer` uses inline assembly to read the receiver and calldata from the calldata arguments. The assembly strongly relies on some assumptions about the argument encoding of the Solidity. One of them is that there are no "garbage bits" between the byte offset of the `bytes calldata _data` variable and the length field of the `bytes calldata _data` argument. This assumption will hold true in most cases, but is not guaranteed to hold. This assumption can be eliminated letting the compiler copy the `_data` into the memory and dealing with it there. Full expectations about the expected information in the `_data` argument must be properly documented, to avoid the misinterpretation of the interface.

```
function onTokenTransfer(
    address _from,
    uint256 _value,
    bytes calldata _data
) external returns (bool) {
```

A similar situation can be found in the `TxPermissions` contract.

Specification Changed:

The code has been commented as follows:

```
// It is assumed that the `_data` field contains the `length` field in its first 32 bytes.
// There are data bytes right after the `length` field (without "garbage bits" between them).
```

7.3 `claimOrderedWithdraw` Not Always Successful

Design | Medium | Version 1 | Code Corrected

After using the `StakingAuRa.orderWithdraw()` function the validator can complete the withdrawal starting from the next epoch using `claimOrderedWithdraw()`.

To prevent abuse, this function queries `_isWithdrawAllowed` once more in order to determined if the validator may have been banned in the meantime. However `_isWithdrawAllowed` also includes a check whether staking or withdrawals are currently allowed using `areStakeAndWithdrawAllowed()`.

Normally such actions are not allowed near the end of a staking epoch in order to not interfere with the validator selection process. Note that claiming a previously ordered withdrawal has no influence on this and hence shouldn't be subject to this restriction. If a party happens to claim their withdrawal at the end of an epoch their withdrawal fails without apparent reason.

Code corrected:

The `_isWithdrawAllowed` function has been refactored and parts of it's functionality has been moved into a new `_isPoolBanned()` function. This function is now querried in `claimOrderedWithdraw()` which resolves problem with the blocked withdrawals at the end of an epoch as described above.

7.4 Incorrect Comment in `finalizeChange`

Correctness | Low | Version 1 | Code Corrected

The comment in the else if branch suggest, it is only been executed in case of malicious validator reporting.

But the code is also executed in case of mining address changes.

Code corrected:

The code comments were corrected and elaborated.

7.5 Incorrect Description

Correctness Low Version 1 Code Corrected

In `StakingAuRaBase` the function description of `_stake(address, address, uint256)` is

```
// @dev The internal function used by the `_stake` and `moveStake` functions.
```

But function is also used in `initialValidatorStake`, `_addPool`.

Code corrected:

The code comments were corrected and elaborated.

7.6 Make `onTokenTransfer()` External

Design Low Version 1 Code Corrected

Function `StakingAuraTokens.onTokenTransfer()` has visibility `public`. This means the function can be called externally and internally from within the contract.

Inside this function the `calldata` is read. This is the data passed alongside the call to the contract and remains unchanged if another function within the contract executes another contract as on a bytecode level this is only a `JUMP`. Function `onTokenTransfer` is currently only called from externally and not internally from within the `StakingAuraTokens` contract. Hence, the calldata consists of the function arguments as expected. Due to the dependency on calldata the functions visibility may be `external` instead of `public` to avoid the function being called from within the contract accidentally during future code changes.

Code corrected:

The function visibility as well as the reads from memory were changed accordingly.

7.7 Multiplication After Division

Design Low Version 1 Code Corrected

In `ValidatorSetAuRa.reportMaliciousCallable()` a multiplication is performed after a division:

```
averageReportsNumber = (reportsTotalNumber - reportsNumber) / (validatorsNumber - 1)
[...]
reportsNumber > validatorsNumber * 50 && reportsNumber > averageReportsNumber * 10
```

Due to possible precision loss, this should be avoided.

Code corrected:

The multiplication is now done before division.

7.8 No Indexed Fields for `ReportedMalicious`

Design | Low | Version 1 | Code Corrected

The `ReportedMalicious` event has multiple fields that might be worth indexing. No field is indexed. POA Network might re-evaluate if this is desired.

Code corrected:

The event has now indexed fields.

7.9 Unchecked Return Value of Transfer

Design | Low | Version 1 | Code Corrected

In `BlockRewardAuRaTokens.transferReward()` and `StakingAuRaTokens._sendWithdrawnStakeAmount()` the boolean return value of the call to `erc677TokenContract.transfer()` is ignored.

While most ERC-20 Tokens (ERC-677 implements the ERC-20 Standard) and the ERC677 token implementation available in the repository revert upon failure, the standard does not require this and returning `false` instead of reverting is valid according to the standard. As the POSDAO system is highly customizable the situation may arises where a token contract not reverting on failure is used.

Similarly the return value of the call to `tokenContract.mint()` inside `TokenMinter.mintReward()` is also ignored.

Code corrected:

The transfer functions are wrapped into a require. The mint function remained as it is since it is called by the `BlockReward.reward` function which is critically sensible for reverting according to POA network.

7.10 `certify` Missing Sanity Check

Design | Low | Version 1 | Code Corrected

The Certifier implements `certify`. The function allows certifying the same address multiple times. The `Confirmed` event would be emitted misleadingly multiple times.

Code corrected:

An appropriate sanity check was added.

8 Notes

We leverage this section to highlight potential pitfalls which are fairly common when working Distributed Ledger Technologies. As such technologies are still rather novel not all developers might yet be aware of these pitfalls. Hence, the mentioned topics serve to clarify or support the report, but do not require a modification inside the project. Instead, they should raise awareness in order to improve the overall understanding for users and developers.

8.1 Avoiding Function Identifier Clashes

Note Version 1

The current proxy scheme is vulnerable to duplicated 4-byte function identifiers which could result in a function identifer clash. POA Network prevents this by using an off-chain script to check for clashes. There are also on-chain solutions like the upgradable transparent proxy solution by OpenZeppelin which might be worth considering.

8.2 ERC677 Standard Is a DRAFT

Note Version 1

The ERC677 standard is based on a eip having draft status since it's creation in 2017. Such standards are subject to changes before the eip's status is finalized.

8.3 Most of the RedBlackTree Library Functions Unused

Note Version 1

Following function of the RedBlockTree Library are unused and hence dead code.

```
BokkyPooBahsRedBlackTreeLibrary.first()
BokkyPooBahsRedBlackTreeLibrary.getEmpty()
BokkyPooBahsRedBlackTreeLibrary.getNode()
BokkyPooBahsRedBlackTreeLibrary.isEmpty(uint256)
BokkyPooBahsRedBlackTreeLibrary.next()
BokkyPooBahsRedBlackTreeLibrary.treeMinimum()
```

Note as the functions are not used within the POSDAO system these were not reviewed as part of this audit.

8.4 Pragma Experimental ABIEncoderV2

Note Version 1

Contract `TxPriority` uses `pragma experimental ABIEncoderV2`. In the compiler version choosen the new ABI encoder is still considered to be experimental.

8.5 UTF-8 Charset

Note Version 1

The validator and the staking contract allow names to be set for pools. The charset for string is UTF-8. UTF-8 has some similar looking characters, which for a human reader some of these letters are indistinguishable.

This allows so called visual spoofing of names. Users and front-end developper should excercise extra caution.

8.6 Unreliable Event Emission When Mining Address Is Changed

Note Version 1

When reporting a malicious validator, the mining address is used and the following event emitted.

The `_maliciousMiningAddress` might change between multiple reportings. Hence, the mining address is no reliable information to process from across multiple events.

8.7 Unused Code `_removeMaliciousValidator`

Note Version 1

The `ValidatorSetAuRa` contract implements the function `_removeMaliciousValidator`. This function is not called at all. The only function it appears is in `_removeMaliciousValidators`. But it is commented out there. Furthermore, the function `_removeMaliciousValidators` does not do anything except for setting `lastChangeBlock`. Hence, also `removeMaliciousValidators` is basically only setting this variable.

This also affects parts of `reportMalicious`. We were verbally informed that client is aware of this and this will be fixed for the final review. Else, this would turn into an issue.