CERTIK

KONET mainnet

Security Assessment

CertiK Assessed on Jan 13th, 2025

CertiK Assessed on Jan 13th, 2025

KONET mainnet

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
POSDAO	OpenEthereum	Formal Verification, Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 01/13/2025	N/A

CODEBASE

https://github.com/kon-mainnet/posdao-contracts

View All in Codebase Page

COMMITS

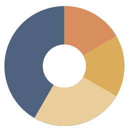
- 0315e8ee854cb02d03f4c18965584a74f30796f7
- ff5efc589e70d5b1755537e2a0d3afd3508482f8
- 5d29eef28407c97c9c8a5b92fd701b4b2d26f643

View All in Codebase Page

Highlighted Centralization Risks

ⓘ Contract upgradeability ⓘ Privileged role can mint tokens ⓘ Fees are unbounded

Vulnerability Summary



12	9	0	0	3	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥	0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧	2	Major	2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨	2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	3	Minor	3 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦	5	Informational	4 Resolved, 1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | KONET MAINNET

Disclaimer

CODEBASE | KONET MAINNET

Repository

https://github.com/kon-mainnet/posdao-contracts

Commit

- 0315e8ee854cb02d03f4c18965584a74f30796f7
- ff5efc589e70d5b1755537e2a0d3afd3508482f8
- 5d29eef28407c97c9c8a5b92fd701b4b2d26f643
- 1f402761720694ad92c4b6f33b9ea57c6742fb6d

AUDIT SCOPE | KONET MAINNET

96 files audited ● 7 files with Acknowledged findings ● 9 files with Resolved findings ● 80 files without findings

ID	Repo	File	SHA256 Checksum
● BAR	kon-mainnet/posdao-contracts	contracts/base/BlockRewardAuRaBase.sol	ded59605d8361fa57f945425146f854e98db56b9c3beb381b0f04b23f41b4f22
● SAB	kon-mainnet/posdao-contracts	contracts/base/StakingAuRaBase.sol	2e51d8b0ca28273b10ebf853958c20458a78ba93d8b8e2a1d10064dc920da666
● TXE	kon-mainnet/posdao-contracts	contracts/base/TxPermissionBase.sol	3d1c75d9c70ce06a2e4a202d1ebf59b16b21c8a158120998be116ef5864c4fe2
● CET	kon-mainnet/posdao-contracts	contracts/Certifier.sol	27170325990c9aa1f64eed2bb5612b3e854e55feebd84d76ce205b747cc88019
● GOE	kon-mainnet/posdao-contracts	contracts/Governance.sol	a7b60e2f3d0dd0afa9f4b59b55f71deb6b8992401bb4f405466fa15e1c54c9da
● INI	kon-mainnet/posdao-contracts	contracts/InitializerAuRa.sol	73a8b411814d6fba179512dd1e35064262ff1c0270cfe6495ff6b0aa54f0714d
● RAN	kon-mainnet/posdao-contracts	contracts/RandomAuRa.sol	ae94ef2380259f0c066aaa11d3b11c39fc5acf412e5f7259ff87f6beb7ed468a
● BAT	kon-mainnet/posdao-contracts	contracts/base/BlockRewardAuRaTokens.sol	786e5a69d63e5ff3aed8255cf96deb80814951aced6181a70df03c7cb2d50be6
● SRC	kon-mainnet/posdao-contracts	contracts/base/StakingAuRaCoins.sol	7889e852e335224a4ff8a43f693cbbdb3f8706a6b2a59666cc372b9a846fb9ee

ID	Repo	File	SHA256 Checksum
● SRT	kon-mainnet/posdao-contracts	contracts/base/StakingAuRaTokens.sol	dbffb42149c41828a1b0209dc2922216b507d68424dc2616679a8fc52f5263f7
● ADM	kon-mainnet/posdao-contracts	contracts/upgradeability/AdminUpgradeabilityProxy.sol	f0aefc13447440d8805d97b2446e6b26e3bc0216f6c2cc812fcde84d6a6ce63e
● PRX	kon-mainnet/posdao-contracts	contracts/upgradeability/Proxy.sol	3d72095667402bb873f5e657505160ae4d20a25bdecff3f025083c95fb1c81a1
● ERB	kon-mainnet/posdao-contracts	contracts/ERC677BridgeTokenRewardable.sol	846e71f06324db1581d66018648d182330fa201b5ad5eca8dba742c7f225fdde
● MIR	kon-mainnet/posdao-contracts	contracts/Migrations.sol	009565c035f8b841612dd99f5073b578c04a7cba792b2d4de2809751a6cf1771
● REI	kon-mainnet/posdao-contracts	contracts/Registry.sol	7e5df38054de82be25fabb86089fb0493a4283d7df7f432dfb26b10d5adfecce
● TPT	kon-mainnet/posdao-contracts	contracts/TxPriority.sol	cc5f4037c8264d433ff3e734a131a3292e98aae78f1235d791a75f802f8738cc
○ BRU	kon-mainnet/posdao-contracts	contracts/base/BanReasons.sol	025bcfbd6769471065bd250ab0742140c826d12e9409b61bb8d5f4faf296bbe1
○ BRC	kon-mainnet/posdao-contracts	contracts/base/BlockRewardAuRaCoins.sol	4f76402994ca3bf221aabf9ad807c880ce29bbcafa25d9ce0034ddcea01b2418
○ TXR	kon-mainnet/posdao-contracts	contracts/base/TxPermissionV3.sol	d054680eb848c682d06cf6a288c80f2467ded23c584e13e4c852982d4ae549f6
○ TXM	kon-mainnet/posdao-contracts	contracts/base/TxPermissionV4.sol	abbf143902225573342cbe4a59d16307f595e72241a0ed317de1c823aeaa16bc

ID	Repo	File	SHA256 Checksum
● ADR	kon-mainnet/posdao-contracts	contracts/libs/Address.sol	d839610c9aba9e14646163932cc0c1924ae84abee7816c08d70e6bedbbb187ce
● BPR	kon-mainnet/posdao-contracts	contracts/libs/BokkyPooBahsRedBlackTreeLibrary.sol	0db44cd6d8695237d31ffa112c20c7c022e58ecb58b1423c9590fc7043af531e
● SMU	kon-mainnet/posdao-contracts	contracts/libs/SafeMath.sol	374c8fd43329210c914c3daa7dc37fdc0df0a6430274286af05b7189cabe75ac
● BAP	kon-mainnet/posdao-contracts	contracts/upgradeability/BaseAdminUpgradeabilityProxy.sol	b22d12e70ea84efc9e03895a6d7471b2db22968e05bfae3b942b7c6399d75929
● BAG	kon-mainnet/posdao-contracts	contracts/upgradeability/BaseUpgradeabilityProxy.sol	4431d79bb059dfdbb224fd7ca1735415c1ffaf0d2c9e9f1c53044dfd9cf1581f
● UAU	kon-mainnet/posdao-contracts	contracts/upgradeability/UpgradeabilityAdmin.sol	9d5c661b3866e0219e91640d449cfb9b7e4e2f64c2e4ec04e39317dc36a43dc4
● UPU	kon-mainnet/posdao-contracts	contracts/upgradeability/UpgradeabilityProxy.sol	0d3625b3297e043296dd55d6abc8e1ea5c50933f6f9649d3e86546f1ecc9de5d
● UOU	kon-mainnet/posdao-contracts	contracts/upgradeability/UpgradeableOwned.sol	4a3c67b3c485e4b92e34bfdae5b225347b774b7996c4a6d86b5c861c0f1e57d8
● BLO	kon-mainnet/posdao-contracts	contracts/BlockRewardAuRa.sol	5de2db5c51451a8745958a14a325a5cb7cb0bef49747ea2a4873abd74e717bc3
● STA	kon-mainnet/posdao-contracts	contracts/StakingAuRa.sol	5595337851a42c319d0195d6273b250392d46e4620bac971243cf4ffff3a3b03
● TMU	kon-mainnet/posdao-contracts	contracts/TokenMinter.sol	ab41337212b9c2ef77bdd47786a4ac201dc53457b47028d52624987c8efb3025

ID	Repo	File	SHA256 Checksum
TPH	kon-mainnet/posdao-contracts	contracts/TxPermission.sol	46599157e5c5dc0eb97c22f7c146db25fd9ffa251d8ec143f1d44bb3f7a00773
VSR	kon-mainnet/posdao-contracts	contracts/ValidatorSetAuRa.sol	9db72699ef9cf97627cc0403b682c57e4b2ccf95f72ba0f6e16cdaea417010fb
IBR	kon-mainnet/posdao-contracts	contracts/interfaces/IBlockRewardAuRa.sol	0f48078ab8745425d134483ac56adad84fe0a1912f094d34441ebe075848de34
IBA	kon-mainnet/posdao-contracts	contracts/interfaces/IBlockRewardAuRaCoins.sol	c9bd8eadbfb62737eb84eae47179fb6ea76818fd11d5077a7771fbdb9118a048
IBT	kon-mainnet/posdao-contracts	contracts/interfaces/IBlockRewardAuRaTokens.sol	bed345b82c4e9333f15d2cd040467723dd10d142834ee3bc31f1f91f4f50d688
ICB	kon-mainnet/posdao-contracts	contracts/interfaces/ICertifier.sol	a580428db800f1bb77bff3059e0e5a0bfe821cd57efca4d2d07f56e06904e48b
IER	kon-mainnet/posdao-contracts	contracts/interfaces/IERC677.sol	52ac95191bb4b842edd39a62944a033e37631c34bfb87e6cc436b58c1711345a
IGB	kon-mainnet/posdao-contracts	contracts/interfaces/IGovernance.sol	5fa2f8959349df8de37219e56409df457d88c93bfdb231296a61e25baec2e16c
IMR	kon-mainnet/posdao-contracts	contracts/interfaces/IMetadataRegistry.sol	23cee29978d4623276e66477314ba112361b77eb2fd768dcb4a42e89a02e5747
IOR	kon-mainnet/posdao-contracts	contracts/interfaces/IOwnerRegistry.sol	48a31257bc61353ebb037f58c0f2d333cdc88c0d093f217c1fd4d8a918bf866f
IRA	kon-mainnet/posdao-contracts	contracts/interfaces/IRandomAuRa.sol	95849f74cc6e6720e2fd6630b653ec3b7c1fdb8478ee31e83b0ae7730fa03d2a

ID	Repo	File	SHA256 Checksum
IRR	kon-mainnet/posdao-contracts	contracts/interfaces/IReverseRegistry.sol	39434601fb6eeaa92a76bf2f30bf615c139e727c1ba2ae0a1759b30db00be6e6
ISA	kon-mainnet/posdao-contracts	contracts/interfaces/IStakingAuRa.sol	85c2c958d24de79b331b42e953996c655e42fcc4e74fef4b3d42e41ea106aa4d
ISR	kon-mainnet/posdao-contracts	contracts/interfaces/IStakingAuRaTokens.sol	0579173964320627 4c6335164a591fcd576446f55e683e843c73ad82f14f41e4
ITM	kon-mainnet/posdao-contracts	contracts/interfaces/ITokenMinter.sol	cd8784c2c05453058d956d55b71d70651d6f5e3d4fbde2b7c9cfdf3e8744f8b3
ITP	kon-mainnet/posdao-contracts	contracts/interfaces/ITxPermission.sol	ca71772bb8976e308e1dbc414284e65be738f1ee2590192a54417253ba626d2c
IVS	kon-mainnet/posdao-contracts	contracts/interfaces/IValidatorSetAuRa.sol	348e0262cc5a4eb9ccecf61b2c5ad8699857149a8015b0b096e742e5b2868ec9
BRH	kon-mainnet/posdao-contracts	contracts/base/BanReasons.sol	025bcfbd6769471065bd250ab0742140c826d12e9409b61bb8d5f4faf296bbe1
BAB	kon-mainnet/posdao-contracts	contracts/base/BlockRewardAuRaBase.sol	3fde6a339dec6a358f6c0a3d6f8639fbeb8e98b0820340eb1210a218aebad9ce
RAC	kon-mainnet/posdao-contracts	contracts/base/BlockRewardAuRaCoins.sol	4f76402994ca3bf221aabf9ad807c880ce29bbcafa25d9ce0034ddcea01b2418
RAT	kon-mainnet/posdao-contracts	contracts/base/BlockRewardAuRaTokens.sol	d8eff2de3702393f15b9301264ee6e5f027d28ae9d64deba51a6fe236ff94f4f
SRB	kon-mainnet/posdao-contracts	contracts/base/StakingAuRaBase.sol	756dfe9863e9c4bdd47289b35137f70beef0ac56b4d7da7c4b98811d5a7819c8

ID	Repo	File	SHA256 Checksum
● ARC	kon-mainnet/posdao-contracts	contracts/base/StakingAuRaCoins.sol	7889e852e335224a4ff8a43f693cbbdb3f8706a6b2a59666cc372b9a846fb9ee
● ART	kon-mainnet/posdao-contracts	contracts/base/StakingAuRaTokens.sol	33ccf49cbf43eaee66eb4a746f1884829d6a887cb8b3a4a6c06a0ac69c11fcfd
● TXI	kon-mainnet/posdao-contracts	contracts/base/TxPermissionBase.sol	3e767cf08e136b78500a808dfa29ddf14336d906cbec62556f49d46ee859680a
● TXS	kon-mainnet/posdao-contracts	contracts/base/TxPermissionV3.sol	d054680eb848c682d06cf6a288c80f2467ded23c584e13e4c852982d4ae549f6
● TXO	kon-mainnet/posdao-contracts	contracts/base/TxPermissionV4.sol	abbf143902225573342cbe4a59d16307f595e72241a0ed317de1c823aeaa16bc
● IBL	kon-mainnet/posdao-contracts	contracts/interfaces/IBlockRewardAuRa.sol	0f48078ab8745425d134483ac56adad84fe0a1912f094d34441ebe075848de34
● IBC	kon-mainnet/posdao-contracts	contracts/interfaces/IBlockRewardAuRaCoins.sol	c9bd8eadbfb62737eb84eae47179fb6ea76818fd11d5077a7771fbdb9118a048
● IRT	kon-mainnet/posdao-contracts	contracts/interfaces/IBlockRewardAuRaTokens.sol	bed345b82c4e9333f15d2cd040467723dd10d142834ee3bc31f1f91f4f50d688
● ICU	kon-mainnet/posdao-contracts	contracts/interfaces/ICertifier.sol	a580428db800f1bb77bff3059e0e5a0bfe821cd57efca4d2d07f56e06904e48b
● IEC	kon-mainnet/posdao-contracts	contracts/interfaces/IERC677.sol	52ac95191bb4b842edd39a62944a033e37631c34bfb87e6cc436b58c1711345a
● IGU	kon-mainnet/posdao-contracts	contracts/interfaces/IGovernance.sol	5fa2f8959349df8de37219e56409df457d88c93bfdb231296a61e25baec2e16c

ID	Repo	File	SHA256 Checksum
IME	kon-mainnet/posdao-contracts	contracts/interfaces/IMetadataRegistry.sol	23cee29978d4623276e66477314ba112361b77eb2fd768dcb4a42e89a02e5747
IOW	kon-mainnet/posdao-contracts	contracts/interfaces/IOwnerRegistry.sol	48a31257bc61353ebb037f58c0f2d333cdc88c0d093f217c1fd4d8a918bf866f
IRN	kon-mainnet/posdao-contracts	contracts/interfaces/IRandomAuRa.sol	95849f74cc6e6720e2fd6630b653ec3b7c1fdb8478ee31e83b0ae7730fa03d2a
IRE	kon-mainnet/posdao-contracts	contracts/interfaces/IReverseRegistry.sol	39434601fb6eeaa92a76bf2f30bf615c139e727c1ba2ae0a1759b30db00be6e6
IST	kon-mainnet/posdao-contracts	contracts/interfaces/IStakingAuRa.sol	85c2c958d24de79b331b42e953996c655e42fcc4e74fef4b3d42e41ea106aa4d
IAT	kon-mainnet/posdao-contracts	contracts/interfaces/IStakingAuRaTokens.sol	05791739643206274c6335164a591fcd576446f55e683e843c73ad82f14f41e4
ITO	kon-mainnet/posdao-contracts	contracts/interfaces/ITokenMinter.sol	cd8784c2c05453058d956d55b71d70651d6f5e3d4fbde2b7c9cfdf3e8744f8b3
ITX	kon-mainnet/posdao-contracts	contracts/interfaces/ITxPermission.sol	ca71772bb8976e308e1dbc414284e65be738f1ee2590192a54417253ba626d2c
IVA	kon-mainnet/posdao-contracts	contracts/interfaces/IValidatorSetAuRa.sol	348e0262cc5a4eb9ccecf61b2c5ad8699857149a8015b0b096e742e5b2868ec9
ADE	kon-mainnet/posdao-contracts	contracts/libs/Address.sol	d839610c9aba9e14646163932cc0c1924ae84abee7816c08d70e6bedbbb187ce
BPT	kon-mainnet/posdao-contracts	contracts/libs/BokkyPooBahsRedBlackTreeLibrary.sol	0db44cd6d8695237d31ffa112c20c7c022e58ecb58b1423c9590fc7043af531e

ID	Repo	File	SHA256 Checksum
SMH	kon-mainnet/posdao-contracts	contracts/libs/SafeMath.sol	374c8fd43329210c914c3daa7dc37fdc0df0a6430274286af05b7189cabe75ac
ADI	kon-mainnet/posdao-contracts	contracts/upgradeability/AdminUpgradeabilityProxy.sol	f41d3a87864a6aa0428f49f2a944f5ddf72db4081f66057368e69acd4933da7e
BAA	kon-mainnet/posdao-contracts	contracts/upgradeability/BaseAdminUpgradeabilityProxy.sol	b22d12e70ea84efc9e03895a6d7471b2db22968e05bfae3b942b7c6399d75929
BAD	kon-mainnet/posdao-contracts	contracts/upgradeability/BaseUpgradeabilityProxy.sol	4431d79bb059dfdbb224fd7ca1735415c1ffaf0d2c9e9f1c53044dfd9cf1581f
PRY	kon-mainnet/posdao-contracts	contracts/upgradeability/Proxy.sol	3d72095667402bb873f5e657505160ae4d20a25bdecff3f025083c95fb1c81a1
UAH	kon-mainnet/posdao-contracts	contracts/upgradeability/UpgradeabilityAdmin.sol	9d5c661b3866e0219e91640d449cfb9b7e4e2f64c2e4ec04e39317dc36a43dc4
UPH	kon-mainnet/posdao-contracts	contracts/upgradeability/UpgradeabilityProxy.sol	0d3625b3297e043296dd55d6abc8e1ea5c50933f6f9649d3e86546f1ecc9de5d
UOH	kon-mainnet/posdao-contracts	contracts/upgradeability/UpgradeableOwned.sol	4a3c67b3c485e4b92e34bfdae5b225347b774b7996c4a6d86b5c861c0f1e57d8
BLC	kon-mainnet/posdao-contracts	contracts/BlockRewardAuRa.sol	5de2db5c51451a8745958a14a325a5cb7cb0bef49747ea2a4873abd74e717bc3
CEI	kon-mainnet/posdao-contracts	contracts/Certifier.sol	27170325990c9aa1f64eed2bb5612b3e854e55feebd84d76ce205b747cc88019
ERT	kon-mainnet/posdao-contracts	contracts/ERC677BridgeTokenRewardable.sol	5601e9428cbe8fbcf05cdee61c754a193dcd700c5f788395cfcb44c48785a1c0

ID	Repo	File	SHA256 Checksum
● GOR	kon-mainnet/posdao-contracts	contracts/Governance.sol	6acbd6e963eb877ff4d77383502012322be0f7a972dfdbb05f6202aef22edfcc
● INA	kon-mainnet/posdao-contracts	contracts/InitializerAuRa.sol	73a8b411814d6fba179512dd1e35064262ff1c0270cfe6495ff6b0aa54f0714d
● MIA	kon-mainnet/posdao-contracts	contracts/Migrations.sol	945476ab90acc73693e4d2135c424676aea1c932c06fde43a8c5b1869bf71e22
● RAO	kon-mainnet/posdao-contracts	contracts/RandomAuRa.sol	6b94cec41e91e12357a62341ea2a5cdc7550d55addcd8c090aa3481de633f8a8
● RES	kon-mainnet/posdao-contracts	contracts/Registry.sol	8a7341acc44dfff2d95e6b6a8bfe7daed14437969dc54e130fccd502aef1de7d
● STI	kon-mainnet/posdao-contracts	contracts/StakingAuRa.sol	5595337851a42c319d0195d6273b250392d46e4620bac971243cf4ffff3a3b03
● TMH	kon-mainnet/posdao-contracts	contracts/TokenMinter.sol	ab41337212b9c2ef77bdd47786a4ac201dc53457b47028d52624987c8efb3025
● TPG	kon-mainnet/posdao-contracts	contracts/TxPermission.sol	46599157e5c5dc0eb97c22f7c146db25fd9ffa251d8ec143f1d44bb3f7a00773
● TPE	kon-mainnet/posdao-contracts	contracts/TxPriority.sol	92bc85f092dc48078f2cce0bfc8498080c71b7f0067dc3d4339abf4ff603f264
● VAR	kon-mainnet/posdao-contracts	contracts/ValidatorSetAuRa.sol	9db72699ef9cf97627cc0403b682c57e4b2ccf95f72ba0f6e16cdaea417010fb

APPROACH & METHODS | KONET MAINNET

This report has been prepared for KONET to discover issues and vulnerabilities in the source code of the KONET mainnet project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Formal Verification, Manual Review, and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | KONET MAINNET

Overview

KONET mainnet is a fork of the POSDAO smart contract suite, designed for deployment on the OpenEthereum network. The system is implemented as a set of Solidity smart contracts that operate using a general-purpose BFT consensus protocol, such as AuthorityRound (AuRa) with a leader and probabilistic finality, or Honeybadger BFT (HBBFT) with instant finality. The algorithm incentivizes actors to behave in the best interests of the network by providing a Sybil control mechanism for reporting and managing malicious validators, distributing block rewards, and maintaining the validator set.

Validator Set Module

`ValidatorSetAuRa` serves as the backbone of the smart contract system, integrating with all other modules to manage and preserve the network's validator set. Here are its main functions and interactions with other modules:

- Adds new candidate validators:
    - Retrieves staking information from the Staking module.
    - Utilizes randomness from the Randomness module for selecting new validator sets.
    - Triggered by the Block-Reward module to update validator and delegator statuses.
- Enables validators to update their mining and staking addresses.
- Deals with malicious validators:
    - Removes and bans validators through the Governance contract.

Staking Module

The Staking module facilitates staking and rewards distribution using ERC677 tokens and native tokens within a proof-of-stake (PoS) blockchain network. It offers users the ability to register as validators by staking tokens into their own staking pools or to participate as delegators by staking tokens into existing validator pools. For these staked validators/delegators, the module enables claiming rewards and withdrawing their staked tokens. The involved contracts are:

- `StakingAuRaBase`
- `StakingAuRaCoins`
- `StakingAuRaTokens`
- `ERC677BridgeTokenRewardable`

Block Reward Module

The Block Reward module executes the essential logic for generating and distributing rewards according to users' staking data. It collaborates with other contracts like "ValidatorSetAuRa" and "StakingAuRa" to manage the reward distribution process and maintain the statuses of validators and delegators. The involved contracts are:

- `BlockRewardAuRaBase`
- `BlockRewardAuRaCoins`
- `BlockRewardAuRaTokens`
- `ERC677BridgeTokenRewardable`
- `TokenMinter`

Transaction Optimization Module

This module defines allowed transaction types for a given sender based on various criteria, limits contract deployment transaction sizes, sets a minimum gas price for specific senders, and manages priorities for specific transaction destinations. It serves to regulate and restrict transaction usage within the network, enabling validators to set zero gas prices and safeguarding the network against potential misuse while maintaining system integrity. The involved contracts are:

- `TxPermissionBase`
- `TxPermissionV3`
- `TxPermissionV4`
- `Certifier`
- `TxPriority`
- `Registry`

Governance Module

`Governance` provides a mechanism for validators to remove other validators from the validator set, either by voting to remove them or by voting to remove and ban them. This can be useful if a validator is not performing their duties properly or is acting maliciously.

Randomness Module

`RandomAuRa` contract provides a transparent and verifiable source of randomness to the consensus protocol. The random seed generated by this contract can help ensure the fairness and unpredictability of the validator selection process.

External Dependencies

In **KONET mainnet**, the module inherits or uses a few of the depending injection contracts or addresses to fulfill the need of its business logic. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Addresses

The following addresses interact at some point with specified contracts, making them an external dependency. All of following values are initialized either at deploy time or by specific functions in smart contracts.

BlockRewardAuRaBase:

- `_prevBlockRewardContract` , `validatorSetContract` , `stakingContract` , `_stakingContract` .

BlockRewardAuRaTokens:

- `tokenMinterContract` , `stakingContract` , `erc677TokenContract` , `tokenContract` , `minterContract` .

StakingAuRaBase:

- `validatorSetContract` , `governanceContract` .

StakingAuRaCoins:

- `validatorSetContract` , `blockRewardContract` , `_to` .

StakingAuRaTokens:

- `erc677TokenContract` , `validatorSetContract` , `blockRewardContract` .

TxPermissionBase:

- `certifierContract` , `validatorSetContract` .

BaseAdminUpgradeabilityProxy:

- `newImplementation` .

UpgradeabilityProxy:

- `_logic` .

Certifier:

- `validatorSetContract` .

Governance:

- `validatorSetContract` , `stakingContract` .

RandomAuRa:

- `validatorSetContract` , `stakingContract` .

TokenMinter:

- `tokenContract` .

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Privileged Functions

In the **KONET mainnet** project, the privileged roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the following finding: `Centralization Risks`.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

FINDINGS | KONET MAINNET

12	0	2	2	3	5
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for KONET mainnet. Through this audit, we have uncovered 12 issues ranging from different severity levels. Utilizing the techniques of Formal Verification, Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CON-01	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	● **Acknowledged**
GLOBAL-01	**Centralization Risks**	**Centralization**	**Major**	● **Acknowledged**
CON-02	Lack Of Storage Gap In Upgradeable Contract	Logical Issue	Medium	● Resolved
GOE-01	Incorrect Usage Of Equality Symbol `==`	Logical Issue	Medium	● Resolved
CON-03	Pull-Over-Push Pattern	Logical Issue	Minor	● Resolved
CON-05	Usage Of `transfer()` For Sending Ether	Volatile Code	Minor	● Resolved
GOE-02	Finalizing A Vote	Design Issue	Minor	● Resolved
CON-04	Missing Emit Events	Volatile Code	Informational	● Resolved
GOE-03	Ballot Results	Design Issue	Informational	● Resolved
INI-01	Deploying The Forked Project On Archived Platform	Volatile Code	Informational	● Acknowledged
REI-01	No Upper Limit In `setFee` Function	Logical Issue	Informational	● Resolved

ID	Title	Category	Severity	Status
UPG-01	Unsafe Proxy Pattern	Logical Issue	Informational	● Resolved

CON-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	● **Major**	**contracts/Certifier.sol (posdao-contracts): 10; contracts/Governance.sol (posdao-contracts): 14; contracts/RandomAuRa.sol (posdao-contracts): 12; contracts/base/BlockRewardAuRaBase.sol (posdao-contracts): 20; contracts/base/StakingAuRaBase.sol (posdao-contracts): 13; contracts/base/TxPermissionBase.sol (posdao-contracts): 14**	● **Acknowledged**

Description

Based on the project organization and logic, these contracts listed below serve as implementation contracts, paired with proxy contracts for contract upgrades. The `admin` role of the proxy contract holds the authority to update the implementation contract behind it. Any compromise to the `admin` account may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

List of contracts:

- `Certifier`
- `Governance`
- `RandomAuRa`
- `BlockRewardAuRaBase`
- `StakingAuRaBase`
- `TxPermissionBase`
- `Certifier`
- `Governance`
- `RandomAuRa`
- `BlockRewardAuRaBase`
- `StakingAuRaBase`
- `TxPermissionBase`

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[KONET Team, June 21, 2024]: The team acknowledged the finding and decided not to change the current codebase. We plan on using a timelock and multisig.

[CertiK, June 24, 2024]: CertiK strongly encourages the project team periodically revisit the private key security management of all centralized roles and addresses.

[KONET Team, January 05, 2025]: The project team has carefully evaluated the risks associated with the admin authority of the proxy contracts and has decided to proceed with the renunciation of the admin role. This decision reflects our commitment to security, decentralization, and community trust. The commit details can be found in the following log: https://github.com/kon-mainnet/posdao-contracts/commit/a322f3aa5fd99458ead9d985dd571a18b200fc00

[CertiK, January 05, 2025]: The team has introduced a `renounceAdmin` function in the latest commit (a322f3aa5fd99458ead9d985dd571a18b200fc00). This function allows the owner to set the `_owner` address to the zero address when necessary.

GLOBAL-01 | CENTRALIZATION RISKS

Category	Severity	Location	Status
Centralization	● Major		● Acknowledged

Description

In `BlockRewardAuRaBase` contract,

- the role `_ercToNativeBridgeAllowed` has authority over the following functions:
  - `addBridgeNativeRewardReceivers` : Called by the `erc-to-native` bridge contract when a portion of the bridge fee/reward should be minted and distributed to participants (validators and their delegators) in native coins.
  - `addExtraReceiver` : Called by the `erc-to-native` bridge contract when the bridge needs to mint a specified amount of native coins for a specified address using the `reward` function.
- the role `validatorSetContract` has authority over the following functions:
  - `clearBlocksCreated` : Clears the values in the `blocksCreated` mapping for the current staking epoch and a new validator set.
- the role `_admin` has authority over the following functions:
  - `initialize` : Initializes the contract at network startup.
- the role `0xffffFFFfFFfffffffffffffFfFFFfffFFFfFFFfE` has authority over the following functions:
  - `reward` : Called by the validator's node when producing and closing a block.
- the role `owner` has authority over the following functions:
  - `setErcToNativeBridgesAllowed` : Sets the array of `erc-to-native` bridge addresses which are allowed to call some of the functions with the `onlyErcToNativeBridge` modifier. This setter can only be called by the `owner` .

In `BlockRewardAuRaCoins` contract,

- the role `validatorSetContract.stakingContract()` has authority over the following functions:
  - `transferReward` : Called by the `StakingAuRa.claimReward` function to transfer native coins rom the balance of the `BlockRewardAuRa` contract to the specified address as a reward.

In `BlockRewardAuRaTokens` contract,

- the role `_ercToErcBridgeAllowed` and `_nativeToErcBridgeAllowed` have authority over the following functions:
    - `addBridgeTokenRewardReceivers` : Called by the `erc-to-erc` or `native-to-erc` bridge contract when a portion of the bridge fee/reward should be minted and distributed to participants in staking tokens.
- the role `owner` has authority over the following functions:
    - `setErcToErcBridgesAllowed` : Sets the array of `erc-to-erc` bridge addresses which are allowed to call some of the functions with the `onlyXToErcBridge` modifier.
    - `setNativeToErcBridgesAllowed` : Sets the array of `native-to-erc` bridge addresses which are allowed to call some of the functions with the `onlyXToErcBridge` modifier.
    - `setTokenMinterContract` : Sets the address of the contract which will mint staking tokens.
- the role `validatorSetContract.stakingContract()` has authority over the following functions:
    - `transferReward` : Called by the `StakingAuRa.claimReward` function to transfer tokens and native coins from the balance of the `BlockRewardAuRa` contract to the specified address as a reward.

In `StakingAuRaBase` contract,

- the role `owner` has authority over the following functions:
    - `setCandidateMinStake` : Sets (updates) the limit of the minimum candidate stake (CANDIDATE_MIN_STAKE).
    - `setDelegatorMinStake` : Sets (updates) the limit of the minimum delegator stake (DELEGATOR_MIN_STAKE).
    - `initialValidatorStake` : Makes initial validator stakes.
    - `withdrawPortis` : Temporary function to withdraw subsidized stake of Portis pool.
- the role `validatorSetContract` has authority over the following functions:
    - `incrementStakingEpoch` : Increments the serial number of the current staking epoch.
    - `removePool` : Removes a specified pool from the `pools` array.
    - `removePools` : Removes pools which are in the `_poolsToBeRemoved` internal array from the `pools` array.
    - `setStakingEpochStartBlock` : Sets the number of the first block in the upcoming staking epoch.
    - `addUnremovableValidator` : Temporary function to add an unremovable validator.
    - `clearUnremovableValidator` : Adds the `unremovable validator` to either the `poolsToBeElected` or the `poolsToBeRemoved` array.
- the role `admin` has authority over the following functions:

    - `initialize` : Initializes the network parameters.

In `StakingAuRaTokens` contract,

- the role `owner` has authority over the following functions:
    - `setErc677TokenContract` : Sets the address of the ERC677 staking token contract.
- the role `erc677TokenContract` has authority over the following functions:
    - `onTokenTransfer` : Stakes the sent tokens to the specified pool by the specified staker.

In `TxPermissionBase` contract,

- the role `owner` has authority over the following functions:
    - `addAllowedSender` : Adds the address for which transactions of any type must be allowed.
    - `removeAllowedSender` : Removes the specified address from the array of addresses allowed to initiate transactions of any type.
    - `setDeployerInputLengthLimit` : Sets the limit of `input` transaction field length in bytes for contract deployment transaction made by the specified deployer.
    - `setSenderMinGasPrice` : Sets the min gas price allowed for a specified sender.
- the role `admin` has authority over the following functions:
    - `initialize` : Initializes the network parameters.

In `BaseAdminUpgradeabilityProxy` contract, the role `admin` has authority over the following function:

- `changeAdmin` : Changes the admin of the proxy.
- `upgradeTo` : Upgrades the backing implementation of the proxy.
- `upgradeToAndCall` : Upgrades the backing implementation of the proxy and call a function on the new implementation.

In `Certifier` contract,

- the role `owner` has authority over the following functions:
    - `certify` : Allows the specified addresses to use a zero gas price for their transactions.
    - `revoke` : Denies the specified addresses using a zero gas price for their transactions.
- the role `admin` has authority over the following functions:

    - `initialize` : Initializes the contract at network startup.

In `Ownable` contract,

- the role `owner` has authority over the following functions:
    - `transferOwnership` : Transfers ownership to a specified address.
    - `renounceOwnership` : Allows the current owner to renounce onwership.
- the role `pendingOwner` has authority over the following function:
    - `claimOwnership` : Allows the newOwner to transfer control of the contract to a newOwner.

In `MintableToken` contract, the role `owner` has authority over the following function:

- `mint` : Mints new tokens to a specified address.

In `ERC677BridgeToken` contract, the role `owner` has authority over the following function:

- `claimTokens` : Allows the owner to claim tokens sent to the contract.

In `ERC677MultiBridgeToken` contract, the role `owner` has authority over the following functions:

- `addBridge` : Adds a new bridge contract to the list of allowed bridges.
- `removeBridge` : Removes an existing bridge contract from the list of allowed bridges.

In `ERC677BridgeTokenRewardable` contract,

- the role `owner` has authority over the following functions:
    - `setBlockRewardContract` : Sets the address of the block reward contract.
    - `setStakingContract` : Sets the address of the staking contract.
- the role `blockRewardContract` has authority over the following function:
    - `mintReward` : Mints new tokens as a reward.
- the role `stakingContract` has authority over the following function:
    - `stake` : Transfers tokens from a staker to the staking contract.

In `Governance` contract,

- the role `stakingAddress` has authority over the following functions:
  - `create` : Creates a new ballot for removing a validator from the validator set.
  - `vote` : Gives a vote for the specified ballot.
- the role `ballot creator` has authority over the following function:
  - `cancel` : Cancels the specified ballot before its expiration.
- the role `admin` has authority over the following functions:
  - `initialize` : Initializes the contract at network startup.

In `Migrations` contract, the role `owner` has authority over the following function:

- `setCompleted` : Sets the `last_completed_migration` status.
- `upgrade` : Upgrades the new migration address.

In `RandomAuRa` contract,

- the role `miningAddress` has authority over the following functions:
  - `commitHash` : Called by the validator's node to store a hash and a cipher of the validator's number on each collection round. The validator's node must use its mining address to call this function.
  - `revealNumber` : Called by the validator's node to XOR its number with the current random seed. The validator's node must use its mining address to call this function.
  - `revealSecret` : The same as the `revealNumber` function (see its description).
- the role `blockRewardContract` has authority over the following function:
  - `onFinishCollectRound` : Checks whether the current validators at the end of each collection round revealed their numbers, and removes malicious validators if needed.
- the role `owner` has authority over the following function:
  - `setPunishForUnreveal` : Changes the `punishForUnreveal` boolean flag.
- the role `validatorSetContract` has authority over the following function:
  - `clearCommit` : Clears commit and cipher for the given validator's pool if the pool hasn't yet revealed their number.
- the role `admin` has authority over the following function:
  - `initialize` : Initializes the contract at network startup.

In `Registry` contract,

- the role `owner` has authority over the following functions:
    - `setOwner` : Sets a new owner of the contract.
    - `setFee` : Sets the fee amount for reserving a name.
    - `drain` : Transfers the contract balance to the owner.
    - `confirmReverseAs` : Confirms the reverse registration of a name for a specified address.
- the `entries[_name].owner` has authority over the following functions:
    - `transfer` : Transfers the ownership of a name to another address.
    - `drop` : Drops the ownership of a name and deletes the associated reverse registration.
    - `setData` : Sets a key-value pair of data associated with a `_name` .
    - `setAddress` : Sets an address value associated with a key for a `_name` .
    - `setUint` : Sets a uint value associated with a key for a `_name` .
    - `proposeReverse` : Proposes a reverse registration for a `_name` , specifying the address to be associated with the name.

In `TokenMinter` contract,

- the role `owner` has authority over the following functions:
    - `addMinter` : Adds a new minter address to the list of allowed minters.
    - `removeMinter` : Removes an existing minter address from the list of allowed minters.
    - `claimTokens` : Calls the `claimTokens` function of the token contract, allowing the owner to claim any ERC20 tokens sent to the token contract
    - `setBlockRewardContract` : Sets the address of the block reward contract.
    - `setBridgeContract` : Calls the `setBridgeContract` function of the token contract, allowing the owner to set the address of the bridge contract.
    - `transferOwnership` : Transfers the ownership of the `TokenMinter` contract to a new owner.
    - `transferTokenOwnership` : Calls the `transferOwnership` function of the token contract, allowing the owner to transfer the ownership of the token contract.
- the role `minter` has authority over the following function:
    - `mint` : Calls the `mint` function of the token contract to mint new tokens to a specified address.
- the role `blockRewardContract` has authority over the following function:
    - `mintReward` : Mints new tokens to the block reward contract.

In `TxPriority` contract, the role `owner` has authority over the following functions:

- `setPriority` : Sets transaction destination priority (weight).
- `removePriority` : Removes a destination from the priority list.
- `setSendersWhitelist` : Sets sender whitelist, an array of `from` addresses which have a top priority: if a whitelisted address sends a transaction, this transaction should be mined before transactions defined by the `setPriority` function.
- `setMinGasPrice` : Sets an exclusive min gas price for the specified transaction destination.
- `removeMinGasPrice` : Removes an exclusive min gas price for the specified transaction destination.
- `transferOwnership` : Transfers ownership of the contract to the `pendingOwner` .
- `renounceOwnership` : Allows the current owner to renounce onwership.

The role `pendingOwner` has authority over the following function:

- `claimOwnership` : Allows the new owner to transfer control of the contract to a new owner.

In `ValidatorSetAuRa` contract,

- the role `admin` has authority over the following functions:
  - `addUnremovableValidator` : Adds a validator to the list of unremovable validators.
  - `addUnremovableValidators` : Adds specified validators to the list of unremovable validators.
  - `clearUnremovableValidator` : Makes the non-removable validator removable.
  - `initUnremovableValidators` : Temporary function to initialize a new set of unremovable validators.
  - `initialize` : Initializes the network parameters.
- the role `stakingAddress` has authority over the following function:
  - `addPool` : Binds a mining address to the specified staking address and vice versa, generates a unique ID for the newly created pool, binds it to the mining/staking addresses, and returns it as a result.
- the role `SYSTEM_ADDRESS` has authority over the following function:
  - `finalizeChange` : Called by the system when an initiated validator set change reaches finality and is activated.
- the role `blockRewardContract` has authority over the following functions:
  - `newValidatorSet` : Implements the logic which forms a new validator set.
- the role `randomContract` has authority over the following function:
  - `removeMaliciousValidators` : Removes malicious validators.
- the role `Governance` has authority over the following function:

    - `removeValidator` : Removes a validator from the validator set and bans its pool.

- the `pool owner` has authority over the following functions:
    - `changeMetadata` : Changes pool's metadata (such as name and short description).
    - `changeMiningAddress` : Makes a request to change validator's mining address or changes the mining address of a candidate pool immediately.
    - `changeStakingAddress` : Changes the staking address of a pool.

In `AdminUpgradeabilityProxy` contract, the role `admin` has authority over the following function:

- `renounceAdmin` : Renounces the admin rights.

Any compromise to the privileged accounts may allow the hacker to take advantage of this authority, altering critical system settings, transferring funds to the hacker's account, updating migration addresses, ultimately damaging the entire ecosystem.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[KONET Team, June 21, 2024]: The team acknowledged the finding and decided not to change the current codebase. We plan on using a timelock and multisig.

[CertiK, June 24, 2024]: CertiK strongly encourages the project team periodically revisit the private key security management of all centralized roles and addresses.

[KONET Team, January 05, 2025]: The project team has carefully evaluated the risks associated with the admin authority of the proxy contracts and has decided to proceed with the renunciation of the admin role. This decision reflects our commitment to security, decentralization, and community trust. The commit details can be found in the following log: https://github.com/kon-mainnet/posdao-contracts/commit/a322f3aa5fd99458ead9d985dd571a18b200fc00

[CertiK, January 05, 2025]: The team has introduced a `renounceAdmin` function in the latest commit (a322f3aa5fd99458ead9d985dd571a18b200fc00). This function allows the owner to set the `_owner` address to the zero address when necessary.

CON-02 | LACK OF STORAGE GAP IN UPGRADEABLE CONTRACT

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/Governance.sol (posdao-contracts): 14; contracts/RandomAuRa.sol (posdao-contracts): 12; contracts/base/TxPermissionBase.sol (posdao-contracts): 14	● Resolved

Description

There is no storage gap preserved in the logic contract. Any logic contract that acts as a base contract that needs to be inherited by other upgradeable child should have a reasonable size of storage gap preserved for the new state variable introduced by the future upgrades.

Recommendation

We recommend having a storage gap of a reasonable size preserved in the logic contract in case that new state variables are introduced in future upgrades. For more information, please refer to: https://docs.openzeppelin.com/contracts/3.x/upgradeable#storage_gaps.

Alleviation

[KONET Team, July 09, 2024]: The team heeded the advice and resolved the issue in commits 848c539d37a4b3d4e6a97d6e23ac5b6e92bcee21 and 837bdac69e0e50db9710c73c5872b8031dd0a9c3 by putting storage gaps in the above mentioned contracts.

GOE-01 | INCORRECT USAGE OF EQUALITY SYMBOL `==`

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/Governance.sol (posdao-contracts): 383	● Resolved

Description

The equality symbol `==` is incorrectly used to set a variable to a value.

Recommendation

We recommend changing the equality symbol `==` to a single equals symbol `=` .

Alleviation

[KONET Team, June 21, 2024]: The team heeded the advice and resolved the issue in commit ff5efc589e70d5b1755537e2a0d3afd3508482f8 by using `=` instead of `==` .

CON-03 | PULL-OVER-PUSH PATTERN

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/ERC677BridgeTokenRewardable.sol (posdao-contracts): 335~336; contracts/TxPriority.sol (posdao-contracts): 54~59	● Resolved

Description

In the `Ownable` and `TxPriority` contracts, when the `transferOwnership()` function changes the `owner` , it replaces the previous `owner` with the new one without ensuring that the new `owner` can perform transactions on-chain. Consequently, if the newly assigned `owner` is invalid, there is no way to revert back to the original owner.

Recommendation

We advise refactoring the linked codes as below:

```
address public pendingOwner;

function renounceOwnership() public onlyOwner {
    _owner = address(0);
    pendingOwner = address(0);
    emit OwnershipTransferred(_owner, address(0));
}

function transferOwnership(address newOwner) public onlyOwner {
    require(address(0) != newOwner, "pendingOwner set to the zero address.");
    pendingOwner = newOwner;
}

function claimOwnership() public {
    require(msg.sender == pendingOwner, "caller != pending owner");

    _owner = pendingOwner;
    pendingOwner = address(0);
    emit OwnershipTransferred(_owner, pendingOwner);
}
```

Alleviation

[KONET Team, July 09, 2024]: The team partially resolved this issue in commit c13dbd239ef4bd6e42d6eef4569f49d3d3a33c7b by revising the related function in the `ERC677BridgeTokenRewardable` contract. This issue still exists in the `TxPriority` contract.

[KONET Team, January 05, 2025]: The ownership management functions have been enhanced and applied to the TxPriority contract to address potential vulnerabilities. The commit details can be found in the following log.
https://github.com/poanetwork/posdao-contracts/commit/9dd5f40afb9368f8c06ab2e2799871a559f93a05

CON-05 | USAGE OF `transfer()` FOR SENDING ETHER

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/ERC677BridgeTokenRewardable.sol (posdao-contracts): 484; contracts/Registry.sol (posdao-contracts): 237, 237; contracts/base/BlockRewardAuRaBase.sol (posdao-contracts): 1026; contracts/base/StakingAuRaCoins.sol (posdao-contracts): 180	● Resolved

Description

In the `Registry` contract, the `drain` function enables the contract owner to transfer the entire balance of the contract to its own account. After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring ether as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300` . This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation

We advise that the linked `.transfer()` and `.send()` calls are substituted with the utilization of the `sendValue()` function from the `Address.sol` implementation of OpenZeppelin either by directly importing the library or copying the linked code.

Alleviation

[KONET Team, July 09, 2024]: The team resolved this issue in commit 6715acc88976228b3460c1469305fca76e909e3f by adding the following function in their contract:

```
function sendValue(address payable recipient, uint256 amount) internal {

        require(address(this).balance >= amount, "Address: insufficient balance");



        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value

        (bool success, ) = recipient.call.value(amount)("");

        require(success, "Address: unable to send value, recipient may have
reverted");

    }
```

GOE-02 | FINALIZING A VOTE

Category	Severity	Location	Status
Design Issue	● Minor	contracts/Governance.sol (posdao-contracts): 297~299	● Resolved

Description

A vote can be finalized in the current staking epoch if the number of votes is at least the number of validators.

```
        } else if (
            IStakingAuRa(validatorSetContract.stakingContract()).stakingEpoch()
 == ballotStakingEpoch[_ballotId]
        ) {
            uint256 keepVotesCount = ballotVotesKeep[_ballotId];
            uint256 removeVotesCount = ballotVotesRemove[_ballotId];
            uint256 banVotesCount = ballotVotesBan[_ballotId];
            uint256 validatorsLength = validatorSetContract.getValidatorsIds().
length;

            if (keepVotesCount.add(removeVotesCount).add(banVotesCount) >=
 validatorsLength) {
                return true;
```

However, the validator that a ballot is for is unable to vote in that ballot.

```
    function vote(uint256 _ballotId, uint256 _choice) public {
        require(ballotCreator[_ballotId] != 0);
        uint256 senderPoolId = validatorSetContract.idByStakingAddress(msg.
sender);
        require(validatorSetContract.isValidatorById(senderPoolId));
        require(senderPoolId != ballotPoolId[_ballotId]);
```

This means that if a current validator is under a ballot, the only way for a vote to be finalized is after it has expired.

Recommendation

If this is not the intended design, a different threshold should be used for finalizing votes before expiration, such as at least 2/3 of all current validators have voted.

Alleviation

[KONET Team, June 21, 2024]: The team acknowledged the finding and decided not to change the current codebase.

[KONET Team, January 05, 2025]:The issue has been addressed following the recommendations provided in the audit report. The ballot finalization criteria have been updated to include a new threshold, allowing a ballot to be finalized if at least 2/3 of the validators have voted. This modification ensures efficient voting processes while maintaining strong consensus among validators. The changes have been committed and implemented successfully.

https://github.com/kon-mainnet/posdao-contracts/commit/532614c629d2584c97b99aee0655ec60a18d91e6

CON-04 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Volatile Code	● Informational	contracts/ERC677BridgeTokenRewardable.sol (posdao-contracts): 867~868, 872~873; contracts/Migrations.sol (posdao-contracts): 16~17, 20~21; contracts/RandomAuRa.sol (posdao-contracts): 149~150; contracts/base/BlockRewardAuRaBase.sol (posdao-contracts): 338; contracts/base/BlockRewardAuRaTokens.sol (posdao-contracts): 86, 103~104, 123~124; contracts/base/StakingAuRaBase.sol (posdao-contracts): 456~457, 651, 658~659; contracts/base/StakingAuRaTokens.sol (posdao-contracts): 205~206	● Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Function List:

- `setErcToNativeBridgesAllowed` --
- `setErcToErcBridgesAllowed` --
- `setNativeToErcBridgesAllowed` --
- `setTokenMinterContract` --
- `setStakingEpochStartBlock` --
- `setCandidateMinStake` --
- `setDelegatorMinStake` --
- `setErc677TokenContract`
- `setBlockRewardContract` --
- `setStakingContract` --
- `setCompleted` --
- `upgrade` --
- `setPunishForUnreveal` --

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[KONET Team, July 09, 2024]: The team resolved this issue in commit 6715acc88976228b3460c1469305fca76e909e3f by emitting events for these functions.

GOE-03 | BALLOT RESULTS

Category	Severity	Location	Status
Design Issue	● Informational	contracts/Governance.sol (posdao-contracts): 417~423	● Resolved

Description

Currently, the result of a ballot is removal or ban if the number of removal, respectively ban, votes exceeds the other two voting options.

```
417         if (removeVotesCount > banVotesCount) {
418             if (removeVotesCount > keepVotesCount) {
419                 result = BALLOT_RESULT_REMOVE;
420             }
421         } else {
422             if (banVotesCount > removeVotesCount && banVotesCount >
keepVotesCount) {
423                 result = BALLOT_RESULT_BAN;
```

For example, if there is a situation where ban votes and remove votes are tied, but both are far larger than keep votes, then the keep result is chosen.

Recommendation

If this is unintended, it is recommended to not have the lowest voting category be executed.

Alleviation

[KONET Team, January 05, 2025]: The identified issue has been resolved following the recommendation to prevent the lowest voting category from being executed unintentionally.

Changes Made:

1. Updated Tie-Breaking Logic:

In the _calcBallotResult function, additional checks were added to ensure that the lowest voting category, such as keep, is not selected in cases where remove and ban votes are significantly higher but tied.

The decision now defaults to a higher-priority option, such as ban or remove, ensuring fairness and logical consistency.

2. Threshold-Based Fallback:

Keep will only be selected when the total number of votes fails to meet the predefined threshold, adhering to the recommendation.

These changes address the reported issue while maintaining the integrity of the voting process. The updated logic ensures a fair and predictable determination of ballot results, as described in commit

https://github.com/kon-mainnet/posdao-contracts/commit/1f402761720694ad92c4b6f33b9ea57c6742fb6d

INI-01 | DEPLOYING THE FORKED PROJECT ON ARCHIVED PLATFORM

Category	Severity	Location	Status
Volatile Code	● Informational	contracts/InitializerAuRa.sol (posdao-contracts): 16~17	● Acknowledged

Description

This project is a fork of the POSDAO smart contract suites , which were written in Solidity version 0.5.10. It's important to note that the original POSDAO contracts have not being updated or maintained for at least three years.

Although the OpenEthereum client supports POSDAO features, the repository was archived by its owner on May 24, 2022. When deploying Solidity contracts version 0.5.10 on the archived OpenEthereum platform, several potential security concerns arise:

1. **Archived Platform**: OpenEthereum is archived and lacks ongoing support and updates, potentially exposing contracts to security risks due to unaddressed bugs or vulnerabilities. Limited community activity may lack timely assistance or feedback on security issues.
2. **Gas Price Considerations**: OpenEthereum permits whitelisting of accounts for zero gas price transactions. Within this suite of smart contracts, validators are designed and permitted to have a zero-value gas price. Prior to deployment, thorough testing is essential to ensure the correct execution of the smart contracts' functionalities on OpenEthereum.
3. **Outdated Solidity Version**: Solidity version 0.5.10 is relatively old and lacks the latest features, improvements, and security enhancements introduced in newer versions. This may limit contract capabilities and security.

Recommendation

It's important to carefully consider the risks and trade-offs before deploying contracts on an archived platform like OpenEthereum and to have contingency plans in place to handle any potential issues that may arise.

Alleviation

[KONET Team, July 09, 2024]: We will change client Openethereum to Nethermind which supports POSDAO contracts.

Our team admits that KONET Mainnet is maintained more than 3 years and legacy protocols and programs exists.

Archived Platform : migrate to Nethermin which is compatible openethereum and POSDAO. POSDAO contract will be upgraded to solidity 0.8.x

Gas Price Considerations : After adopting new gas price model witch use KONE and erc-1559 we will use POSDAO with gas consumption. originally validators are designed and permitted to have a zero-value gas price but our version need gas.

Outdated Solidity Version : POSDAO contract will be upgraded to solidity 0.8.x

REI-01 | NO UPPER LIMIT IN `setFee` FUNCTION

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/Registry.sol (posdao-contracts): 226	● Resolved

Description

In the `Registry` contract, the `setFee` function enables the owner to designate the `fee` variable, which serves as the service fee for users accessing the `reserve` function. However, the `fee` setting lacks an upper limit. This means that it's possisble to set the total fee rate to an arbitrary amount.

Recommendation

We recommend adding reasonable boundaries for the fee.

Alleviation

[KONET Team, July 09, 2024]: The team resolved this issue in commit 6715acc88976228b3460c1469305fca76e909e3f by adding the following check:

```
        require(_amount <= 10000 ether, "should not be exceed more than 10,000
KONET");
```

UPG-01 | UNSAFE PROXY PATTERN

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/upgradeability/AdminUpgradeabilityProxy.sol (posdao-contracts): 11~12; contracts/upgradeability/Proxy.sol (posdao-contracts): 65~66	● Resolved

Description

The `Proxy` contract lacks a definition for the `_willFallback()` function, and proxy contracts inheriting from `Proxy` fail to override this function. Consequently, there is no access control for the fall back function of `AdminUpgradeabilityProxy` contract, allowing the proxy admin unrestricted interaction with the implementation contract:

```
function _willFallback() internal {
}
```

The absence of access control means that both the `AdminUpgradeabilityProxy` contract and the `_logic` contract may share the same admin. While convenient, this setup poses a risk of function signature collisions, potentially rendering functions on the implementation contract inaccessible to the admin. A more secure approach to the proxy pattern typically involves assigning one admin for proxy upgrades and another for calling access-controlled functions on the implementation contract.

Recommendation

To mitigate this risk, ensure that no functions in the implementation contract share the same signature as those in the proxy contract. If this is not possible within the current codebase, consider the following:

```
  /**
   * @dev Only fall back when the sender is not the admin.
   */
  function _willFallback() internal {
    require(msg.sender != _admin(), "Cannot call fallback function from the proxy admin");
    super._willFallback();
  }
```

Alleviation

[KONET Team, July 09, 2024]: The team resolved this issue in commit 5d29eef28407c97c9c8a5b92fd701b4b2d26f643 by adding the following function in their contract:

```
function _willFallback() internal {
    require(msg.sender != _admin(), "Cannot call fallback function from the proxy
admin");
    super._willFallback();
    }
```

APPENDIX | KONET MAINNET

Finding Categories

Categories	Description
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.