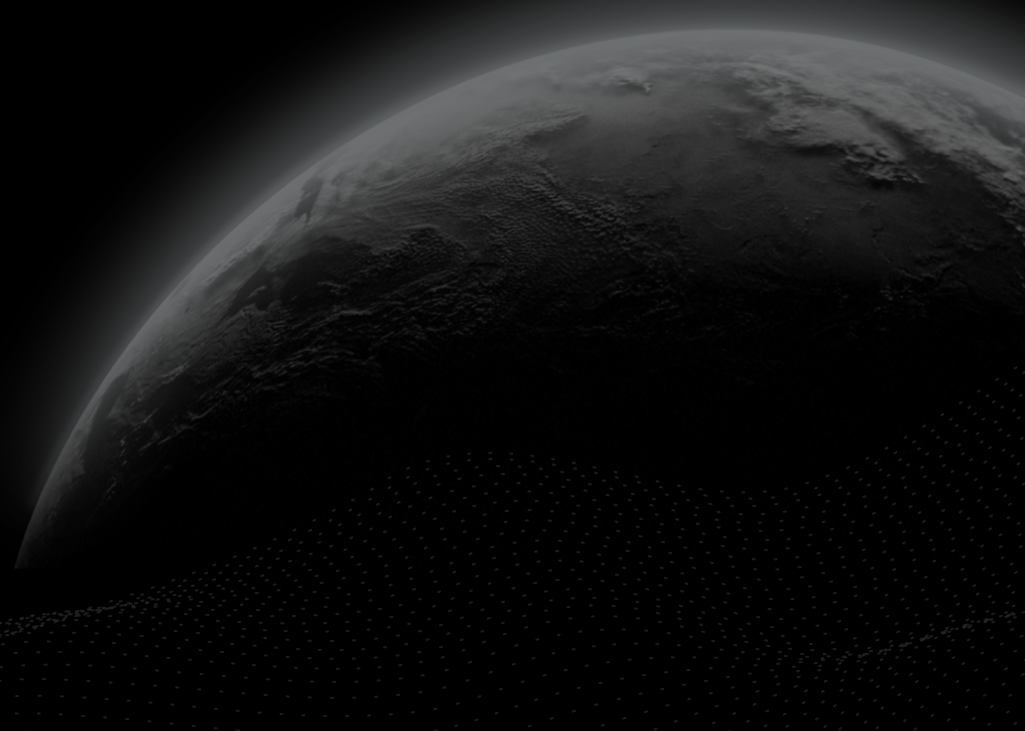
CERTIK

Security Assessment

Ronin DPoS Contracts

CertiK Verified on Mar 30th, 2023



CertiK Verified on Mar 30th, 2023

Ronin DPoS Contracts

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
GameFi	Ronin	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 03/30/2023	N/A

CODEBASE

https://github.com/axieinfinity/ronin-dpos-contracts/

...View All

COMMITS

- 1d3f5e3c1de471edd6e8b4ea15167130f40e3d90
- 450241f8e4fa2be33c9f14ca6dca57f12af0e15a
- 66903dbcdfb64964abe16994b4b2e7d5d9057ded

...View All

Vulnerability Summary

14	11	0	0	3	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

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.
1	Major	1 Resolved	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.
4	Minor	4 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.
7	Informational	4 Resolved, 3 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 | RONIN DPOS CONTRACTS

CODEBASE | RONIN DPOS CONTRACTS

Repository

https://github.com/axieinfinity/ronin-dpos-contracts/

Commit

- 1d3f5e3c1de471edd6e8b4ea15167130f40e3d90
- 450241f8e4fa2be33c9f14ca6dca57f12af0e15a
- 66903dbcdfb64964abe16994b4b2e7d5d9057ded

AUDIT SCOPE | RONIN DPOS CONTRACTS

116 files audited ● 3 files with Acknowledged findings ● 9 files with Resolved findings ● 104 files without findings

ID	File	SHA256 Checksum
MAA	contracts/mainchain/MainchainGatewayV2.sol	3ef6b44db6f8b6be9f800baaa418e252f081a86202a7bcf706ca2468ee95a55e
CSI	contracts/ronin/staking/CandidateStaking.sol	5e5205ac23a69f07f5f797440d2e866db3e3e891e315906d103168d287c02c6a
ROG	contracts/ronin/RoninGatewayV2.sol	bb7242884766e14a0af68e3d6b6e3a2f28226cab4b078756d96435db3460ed80
BOP	contracts/extensions/isolated-governance/bridge-operator-governance/BOsGovernanceProposal.sol	f28114a9bd80154eb9719c35a7ede1a0ef2692f8f37ee6b6b66d187c8706cf69
CGU	contracts/extensions/sequential-governance/CoreGovernance.sol	67388300ec0a814f2eed95d131eaf1a9b4cdf15aa65b4c8a8e0ae1c872428602
ROR	contracts/multi-chains/RoninTrustedOrganization.sol	0f0fb5408c85c59905f2a6e6e5816dfe9436710651d42d213cbb6fb14b416378
CST	contracts/ronin/slash-indicator/CreditScore.sol	319de9dd538cffc728931dc6e2b194264ab8b5837b6b3f5b904d41543a3ce69e
SLD	contracts/ronin/slash-indicator/SlashDoubleSign.sol	f73238eb26a05ac996fd1cef5b0201a0c80a98130ebe73d0ebcc6686f0699b8a
SUU	contracts/ronin/slash-indicator/SlashUnavailability.sol	57150c76e6efbfa5089a78973d4ff40d7954c0ecc6708966020a34339de20a31
DSU	contracts/ronin/staking/DelegatorStaking.sol	25cc2eb6e53129043ae4cbb250e5d78c8682337f5b8867b94dda83d25e29d870
CEU	contracts/ronin/validator/CoinbaseExecution.sol	302d23e0dbc32bed81200bb1d5cf6ca51f40837d4489738963c73823378b3794
ROO	contracts/ronin/RoninGovernanceAdmin.sol	6c4d3186f26893665af85049c05e9758e9f5860b350d5c77f69261f8212452e3
HAS	contracts/extensions/collections/HasBridgeContract.sol	c37c3558f0a0d5222c1ecfb331ace6804323526fbee4757d3661f884bbdbd3c2
HTC	contracts/extensions/collections/HasBridgeTrackingContract.sol	3c06023e75e4872122f2a7c1148f89e1c57724e4ae088a2204d4f573408ffb84

ID	File	SHA256 Checksum
HAM	contracts/extensions/collections/HasMaintenanceContract.sol	af69bf4bad3842b31ac9a588ee190b71f0b0c76145b93d2ed8994ce66b2c04ae
HAP	contracts/extensions/collections/HasProxyAdmin.sol	24aef138712d0d2f8d18da3ac5fd2b873d10ce60eb845ffdcb0768fb7b452580
HRA	contracts/extensions/collections/HasRoninGovernanceAdminContract.sol	1374a14ceaa37995b1fa989530a3cbc507299e583951a8c3404d232a1d0e9d20
HRO	contracts/extensions/collections/HasRoninTrustedOrganizationContract.sol	209b982c945157b632c779f171199e0d44096cf5131eb0e4b9bedc62da1fce48
HIC	contracts/extensions/collections/HasSlashIndicatorContract.sol	7ae0efd54c7729b994c4f3fbefda12feab4d7d7b9d23f0416cff7805284cbfd4
HAT	contracts/extensions/collections/HasStakingContract.sol	a13a19fdbbc7edfc18baca7b8e1599682fff7299761cb979ddb53da529416d77
SVC	contracts/extensions/collections/HasStakingVestingContract.sol	7ef73a03d53bf83b6c856ee54cb345219bb4b09ccff34364d2fbe9d52287836b
HAV	contracts/extensions/collections/HasValidatorContract.sol	a3f3f315984bb9d48b767fb8e44c6212fa72ad51ecb0c01ccb062a57eb87f99a
PCH	contracts/extensions/consumers/PercentageConsumer.sol	8fcdff8e48c8919da5ae9455b104b8ab1b26857c9f4f23239cd455ee875854f2
FOW	contracts/extensions/forwarder/Forwarder.sol	ee774fa6269dea2cd157a4a60ff00d2f68943fb9943cd643669bd753a3b245ec
BGR	contracts/extensions/isolated-governance/bridge-operator-governance/BOsGovernanceRelay.sol	eba8de12ff0fa9ada12e76823709e97f3f83a201a89fb4bc25ff6b08a9b5607e
IGU	contracts/extensions/isolated-governance/IsolatedGovernance.sol	98ded14354f2b3096973b8770d75d765e56cb1be6f1b1be8ed84ac72d371d3d3
GPT	contracts/extensions/sequential-governance/GovernanceProposal.sol	21f29753dbd276a9ce551d249cc6e1894e71e32cdf19f6d8eeb24398eed6bb4f
GRU	contracts/extensions/sequential-governance/GovernanceRelay.sol	e8efb1ed6ec5e4435b7dce6fd43cca9b50bccc3ad34baa0b7a6a83b6fccd29c1
GVU	contracts/extensions/GatewayV2.sol	d7ee4e26682d0f3567920c25c5c3a2bf4d0f2ff47b8335a3bd7411c87dbb6bd1

ID	File	SHA256 Checksum
GAU	contracts/extensions/GovernanceAdmin.sol	0ae4fb0e8ce9b53d04f7baff93e89b717613a4e2fc59b12fc1545e483571debf
MWU	contracts/extensions/MinimumWithdrawal.sol	9202954d5720d1c60529d7ace13e03b2fae85bbd889cd5c1e7833ff96b542a3a
ROH	contracts/extensions/RONTransferHelper.sol	628949ee5db8a7204dd403991fe7965ab7e34480879ff0eb8b1b862e46fb7b5f
TUV	contracts/extensions/TransparentUpgradeableProxyV2.sol	c0a99344bad819e90cd406994305e1bd8a317abad8929d0db25fa9205d4c11cf
WLU	contracts/extensions/WithdrawalLimitation.sol	461882adc2b000b39b01232195ba651ffac023fdd1af58c134b6657bdab1a8c8
IBC	contracts/interfaces/collections/IHasBridgeContract.sol	d67ce4511e4f56d1baaa8cbb2fe707123660c781ea5455ebec0ce72c84f3cba8
ITC	contracts/interfaces/collections/IHasBridgeTrackingContract.sol	6e2bbeee2bb8017d24901a40753c5655327d6e8c79675482345ea573ddd23f1a
IHA	contracts/interfaces/collections/IHasContract.sol	447ed3c6dd62cd1212e69536041d41e93277df9045e37994deab80a237c6112d
IMC	contracts/interfaces/collections/IHasMaintenanceContract.sol	7e34f130b4bfd91483e8e357ead737cd0a99cf00a8dd60d28bf9c5c1d6ec5692
IHG	contracts/interfaces/collections/IHasRoninGovernanceAdminContract.sol	41cc0e79e822fb5c1d7c73c4cfd8250338e80261ff3bfb24ac6e5b50f3d6959d
ITO	contracts/interfaces/collections/IHasRoninTrustedOrganizationContract.sol	623e9462083a27a0aa419ed783a4f9c1db5adfecefe7c15b72ecaa9c939bd99d
IHI	contracts/interfaces/collections/IHasSlashIndicatorContract.sol	fdc013247a2c7ae892cc500d770c83cd458d00ff5da9a24b3d9fa4016357a2c7
ISC	contracts/interfaces/collections/IHasStakingContract.sol	a461101cd1409a2447198570cc62e989d039512bc5ce5d4c3825909e47329189
IHK	contracts/interfaces/collections/IHasStakingVestingContract.sol	49c81dfd36d9189124adbec32dd79a7ab2fbd70e69b10797b9aea8bb28451428
IHL	contracts/interfaces/collections/IHasValidatorContract.sol	12e2e9908507456618424a4f9b7f990cfb4bce41a3d2e519f57eea47fb994c7f

ID	File	SHA256 Checksum
CHA	contracts/interfaces/consumers/ChainTypeConsumer.sol	cd3480f51cb2e431ce59f9b75a8c570780c3ecf167bf01bd4bf27f2d6424fe7d
MAP	contracts/interfaces/consumers/MappedTokenConsumer.sol	6bfc25eb193416e1dcdd81457690a2cce58a222b7f3847759b054bf0854a8e94
PER	contracts/interfaces/consumers/PeriodWrapperConsumer.sol	cc0bbd5df9805828fe2bade53c970a929187745f10c340139e3ee0c6ad4a9ad3
SCU	contracts/interfaces/consumers/SignatureConsumer.sol	f9f8a78e55b9de1c5627e5be695e004c7bc29a3e387358e5a25d430550791052
VOT	contracts/interfaces/consumers/VoteStatusConsumer.sol	a638606fd88078d3bb58da5f2086ae514d293b6d68b7a3d599d8582041d8780f
WEI	contracts/interfaces/consumers/WeightedAddressConsumer.sol	e6a0f5c53db2d7a2da81ff00a1ae7e74e39542ce36f92cdafd8978b55bca0015
IBE	contracts/interfaces/slash-indicator/IBaseSlash.sol	4b9fe9be49a6decef2fea992c159bdd961d93b54ed0fa5313e3cd133ce50f2b5
ICR	contracts/interfaces/slash-indicator/ICreditScore.sol	7aec5a9092bd61e1dc1ffb15a57671f1a7a43da2a21fb387f2b225ce0e3bc09a
IBO	contracts/interfaces/slash-indicator/ISlashBridgeOperator.sol	2d9704afeb480b0b17d2ab45905ac8fe3ccd1739cdac70b99540b52ec6b56cf9
ISA	contracts/interfaces/slash-indicator/ISlashBridgeVoting.sol	c85c78e8ab1bd95cf9fc987b6d4cdd1a463f81f8c12d6c37cdb0e46a8750bd95
ISS	contracts/interfaces/slash-indicator/ISlashDoubleSign.sol	232615a9055f28b734adacc9281f8ee02c602c6d1c815ec0ac5dfb2b3a8aa0b6
ISN	contracts/interfaces/slash-indicator/ISlashIndicator.sol	f0ed57caebf615bcb5cdeb9fedb2bfe8df3922fc7c4a677f757502574ebcc8e2
ISY	contracts/interfaces/slash-indicator/ISlashUnavailability.sol	ba3009d06cd95f73e3ed028a54c85113018f0371094c9d439aec00d0f0fa48e2
IBK	contracts/interfaces/staking/IBaseStaking.sol	0f32e3cdda85f7352d608952cf9172988932eceb7200e51d519c2788c83406d6
ICN	contracts/interfaces/staking/ICandidateStaking.sol	b19d68f731699c633892c492db6bdb4d024ceec9d4f91dc001c41eeb3bdda392

ID	File	SHA256 Checksum
IDE	contracts/interfaces/staking/IDelegatorStaking.sol	1c554e36645712492bdb4943ba96339ca8c4edf35070f0307b9d84758891b44d
IRE	contracts/interfaces/staking/IRewardPool.sol	d7697ba2d37e2d3d3b8baee13b517f19ae75c06a584e79cf054ca973806c560e
ISG	contracts/interfaces/staking/IStaking.sol	267c20f73306320453e15c9b0e4cfc9c734baa294d849bfb4ae938893a7d4b35
ICF	contracts/interfaces/validator/info-fragments/ICommonInfo.sol	af5480dbd2f9a70b0f1e3b26591cbb61d89aee83039633e3b319a4bcda6edb53
IJA	contracts/interfaces/validator/info-fragments/IJailingInfo.sol	fbb94c4252b273f8a4ec7304fd92976ea4538233850822c6ab937b47b28ff5ca
ITM	contracts/interfaces/validator/info-fragments/ITimingInfo.sol	33e7424251f7434cdfeedef7049baaf70532805bc7df6aeccb0be87fd88d5bb0
IVA	contracts/interfaces/validator/info-fragments/IValidatorInfo.sol	51a7299d3f9b11c332d53abcc902a80ac06ee4f77ef3b8699c6d4782711cd88b
ICD	contracts/interfaces/validator/ICandidateManager.sol	db34fe1adc8c5dd4f3ca9746c4592f3beb0f5ed0bce2079739547be481afcaaa
ICO	contracts/interfaces/validator/ICoinbaseExecution.sol	193f7856e8dd785cc5b8d7e6884cfa67fb97c9818e927f9a0a7ed91751460d02
IEE	contracts/interfaces/validator/IEmergencyExit.sol	9d6f2745132f95df3a9f2a3e0853f68067ae1f8753eaa6451e3f2ae7e11ef21b
IRS	contracts/interfaces/validator/IRoninValidatorSet.sol	a53ef8ef2107b231c8359256e71772fbc44d808846590901e18adb9d818e7e68
ISX	contracts/interfaces/validator/ISlashingExecution.sol	e10beb4c46978b2b494a516286bdc8e53e5daf361576ef65346a86b29e4ddfe5
IBU	contracts/interfaces/IBridge.sol	b64fcf72842ebfcf6207c23c2fd0622953e053715b0a8747a07e8ce24d20b4da
IBR	contracts/interfaces/IBridgeTracking.sol	22fea89e0c031f6a3611342d46e5f24668aadfa27a172a67ff9d0176af37f996
IEM	contracts/interfaces/IERC20Mintable.sol	4795937cb211a75c6c525b06508e7f57d73e7bbc24d6b4e36cb3d26b2c19aea5
IRC	contracts/interfaces/IERC721Mintable.sol	a93c33101084deef5fca264a4dff73f05cce8ca33519648d2128596b62946214

ID	File	SHA256 Checksum
IMV	contracts/interfaces/IMainchainGatewayV2.sol	95cad7f21180621b0a1a7b40d8ba64c232303dd87ae88ec839c3301427bdafa7
IMU	contracts/interfaces/IMaintenance.sol	99c4de034df72dd8c5f4bfec542dba50329d56d716fa36d3cc1e2140ff2d144d
IQU	contracts/interfaces/IQuorum.sol	5e12f2f1134550dfe70bc1f2503ff11fb9181c6b874f29bc262393e01c5daa12
IGV	contracts/interfaces/IRoninGatewayV2.sol	c995cf52fb48ece798bf456d568b8ff75f8ad00366d59c49a5ff9bd513a68dce
IGA	contracts/interfaces/IRoninGovernanceAdmin.sol	d4d16a973e9651dcc2832c0f3a0efeed3067b9764174c2fe6dbe0dd653139dbb
IRO	contracts/interfaces/IRoninTrustedOrganization.sol	1c16884f149b8f135ee2d0bf2a1ee91fda236b206e1345e76a8957fac69943f5
IST	contracts/interfaces/IStakingVesting.sol	abf3c4577855301d11c77f9b0b9eeaf7733dee127149d9765f95a41741121e52
IWT	contracts/interfaces/IWETH.sol	688a73efabe2972c17647f4daba15e1e55d59aa9a5d267cf7c1f2aca26dddfda
ADD	contracts/libraries/AddressArrayUtils.sol	017df09a2ed4f948df75f8cf186d1f67b67ce0248edcf7c4b8190d37ef6422df
BAO	contracts/libraries/Ballot.sol	ebaac64bd83794d8051c5e3067c04320a24055e14dd5454a17dba7cb117ad23b
BRD	contracts/libraries/BridgeOperatorsBallot.sol	cfe83bba024c8da96c22cdd73a52979ab83e983777d0b8e89b99385e59a75ff0
EEB	contracts/libraries/EmergencyExitBallot.sol	1ce876de19627afcd26b9feb59b2e1a07cc533ac202d46025935a582cfa246c7
EFU	contracts/libraries/EnumFlags.sol	9362dea4679b4cbd5432321576596fbce9a8b3914088b02fc6d3da70f4f4499c
GPH	contracts/libraries/GlobalProposal.sol	ddcfeb1b84f0c85c3e4ef99a7d22130751370462f27c452de4bc73e51791c018
MAH	contracts/libraries/Math.sol	76f4e16dca3d869646724cee16835aac6b1ff092451fa450889b9fac9734ad98
PRP	contracts/libraries/Proposal.sol	3c0e994ce0418fab258fdf8ab75f33a74819d4ce5a7ce48f4b5df0c3542317a5
TOE	contracts/libraries/Token.sol	acfe38ceee8be89581c8689b51d0e879daedff91534697825f4071a22322b2f1

ID	File	SHA256 Checksum
TRN	contracts/libraries/Transfer.sol	86ba568b7e2d0c28b57e319423db1291fa5409a0408e755978f78fd6ebdceb53
MAG	contracts/mainchain/MainchainGovernanceAdmin.sol	4c833c73e05a428dfb72d610f5da6e35f2f07e5fde3e06e89acf31488b5945ff
PCU	contracts/precompile-usages/PCUPickValidatorSet.sol	579d6e69c8bc09054c5a4af6ddf56890ef0e1253398de08893ba96a337d374a6
PCS	contracts/precompile-usages/PCUSortValidators.sol	f62c809c91f551668746a4733a39db6c5fa6f18ffdd9bf3ca54f8c47d4963a5e
PCV	contracts/precompile-usages/PCUValidateDoubleSign.sol	933b0fd4bbe82bb5c4fe3f4fb605ab70c87e2b4ebca87e7f9f435212fe5b492b
PUB	contracts/precompile-usages/PrecompiledUsage.sol	5c1345da8a30a90045db5836d3bfbf0d85865ebab907a9e3ecdf9c7d92e2f5c8
SLB	contracts/ronin/slash-indicator/SlashBridgeOperator.sol	90c3439fe64d3ccddaaeb1f4afce7c6aa023b9d5c1f1df41866140e6007b7018
SLR	contracts/ronin/slash-indicator/SlashBridgeVoting.sol	885ce172503383939c093b7f181170c383a030414fb6b6c72f5ad1b376bb6e22
SIU	contracts/ronin/slash-indicator/SlashIndicator.sol	162e5c7c6c23eec5828a9abc270f5dd86d86ff9e371590983786cf57afc049c6
BSU	contracts/ronin/staking/BaseStaking.sol	6487ef7cf74b0b05d0257d0892f21c50a3a92e77f83b0e1ac612b774d21e2ea6
RCU	contracts/ronin/staking/RewardCalculation.sol	9e34358a40fdb756904eb643b0c01b798e8a014e92e43ff20743e650da3eb968
STS	contracts/ronin/staking/Staking.sol	5f918705b3652214c601d114c74df163d989062a5e73a9006e71843ab02b9ad7
CSG	contracts/ronin/validator/storage-fragments/CommonStorage.sol	bdaaa0a68958b0ba074f8f875978bc1e7595dc68fea134a10cfcc17ba4bcf655
JSU	contracts/ronin/validator/storage-fragments/JailingStorage.sol	0d54da71b1a35661c20ca8f675848e70e1468a67714ca01883ecd39ad7acacac
TSU	contracts/ronin/validator/storage-fragments/TimingStorage.sol	60b9d0aa8843f53a3469ee031d11994af48114f8a8a9d12ba2ce9066c77ed7f4
VAT	contracts/ronin/validator/storage-fragments/ValidatorInfoStorage.sol	f8ae57c608f76d1cdad3467ea972dafc5b60f4e28e58dff52fe0715650cfa162

ID	File	SHA256 Checksum
CMU	contracts/ronin/validator/CandidateManager.sol	0422484b10fe6f868813e68d021abfab6b37c1e84ac69c51333cabe6d86784c9
EEU	contracts/ronin/validator/EmergencyExit.sol	3d79b59d6e862470197ddeb816bcba60007e6cedecdcc17177a80f1c1d5b2f28
ROV	contracts/ronin/validator/RoninValidatorSet.sol	532150628f3e0619971f4e658f49c374e04231053f32b727879ceff92d3abe69
SEH	contracts/ronin/validator/SlashingExecution.sol	1f1e4a8ebc5b56635e8b97ccc46cb2a7a2595148b071f5e2125b04142a0f3870
BTU	contracts/ronin/BridgeTracking.sol	1a975f5db1d848bd82a6efbc0efe9d557e0712899d37aa257682f4e3d5f693d2
MAE	contracts/ronin/Maintenance.sol	8884f3280ac08276ebd9575a3345f6c4de74a453d932582c100987230ed2f65c
SVU	contracts/ronin/StakingVesting.sol	7092cd13a6495f081c456db96df3da930807197386fd4a50331f1c10b5d7c08f
VFB	contracts/ronin/VaultForwarder.sol	4dfc9eb5562088f362b499595e9d32f13c24e70e11f6f25e767b901f1a87b2cc

APPROACH & METHODS | RONIN DPOS CONTRACTS

This report has been prepared for Sky Mavis to discover issues and vulnerabilities in the source code of the Ronin DPoS Contracts project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing 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.

OVERVIEW | RONIN DPOS CONTRACTS

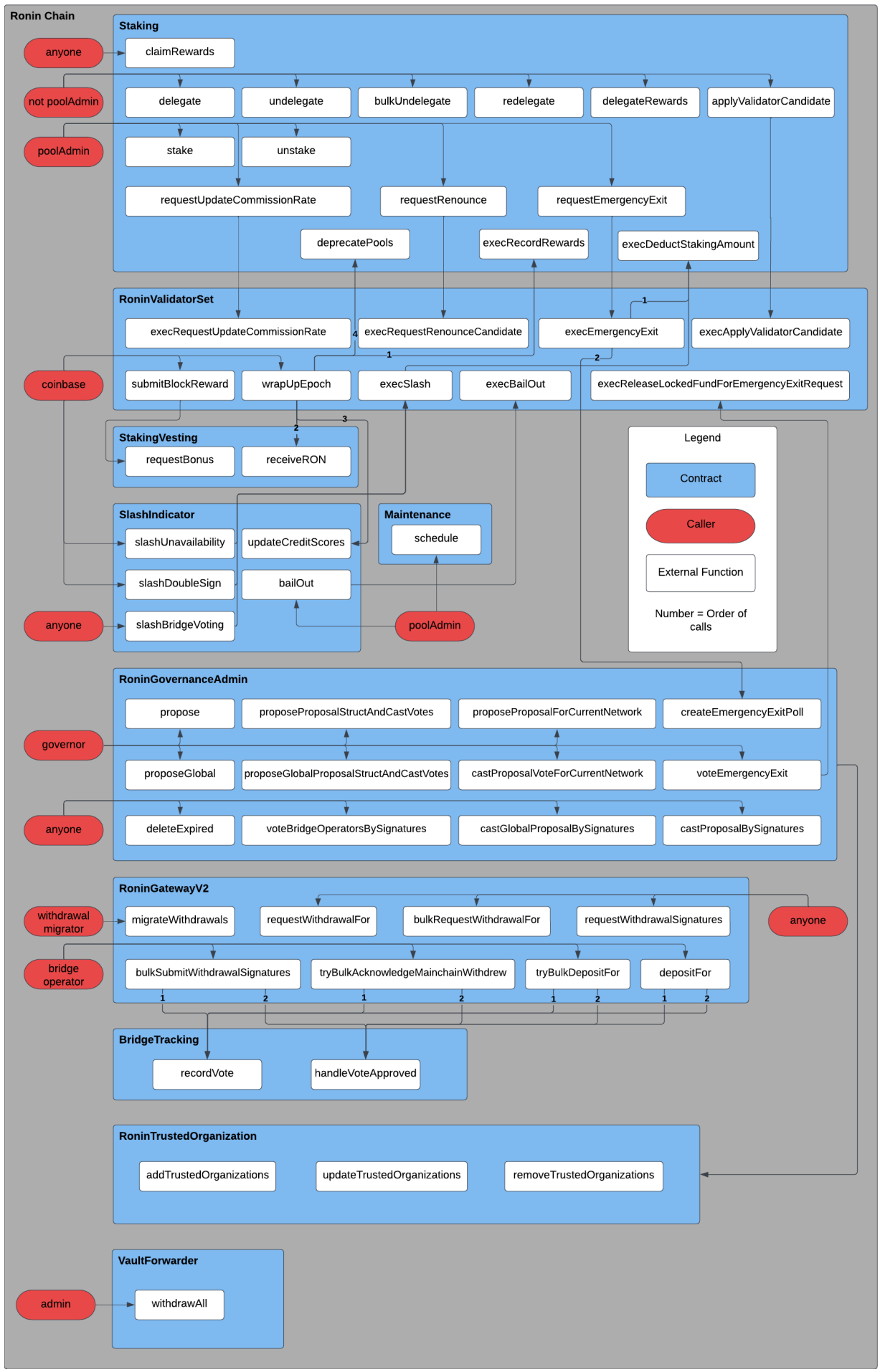
Ronin DPoS Contracts is a collection of smart contracts that power the Ronin Delegated Proof of Stake (DPoS) network.

The project includes the following components on Ronin and Mainchains separately:

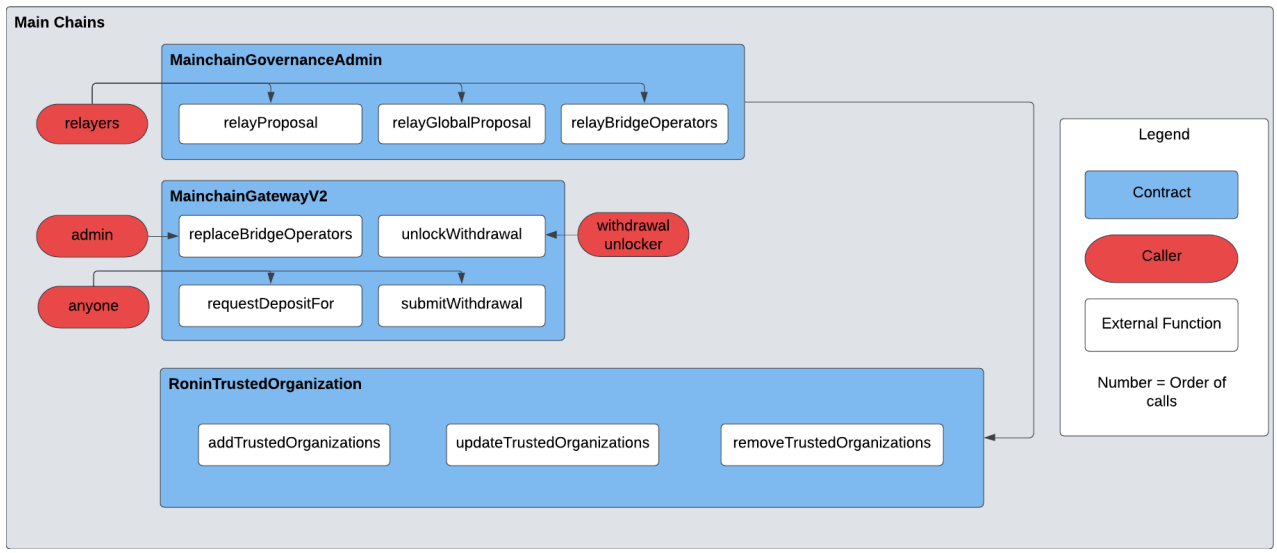
- On Ronin chain:
  - The governance contract: `RoninGovernanceAdmin`
  - The bridge operation contract: `RoninGatewayV2`
  - The trusted organization contract: `RoninTrustedOrganization`
  - DPoS contracts: `SlashIndicator`, `Staking`, `RoninValidatorSet`, `BridgeTracking`, `StakingVesting`, `Maintenance`, `VaultForwarder`, etc.
- On Mainchains:
  - The governance contract: `MainchainGovernanceAdmin`
  - The bridge contract: `MainchainGatewayV2`
  - The trusted organization contract: `RoninTrustedOrganization`

Workflow Graph

Ronin Chain


Ronin Chain
Staking
anyone
claimRewards
not poolAdmin
delegate
undelegate
bulkUndelegate
redelegate
delegateRewards
applyValidatorCandidate
poolAdmin
stake
unstake
requestUpdateCommissionRate
requestRenounce
requestEmergencyExit
deprecatePools
execRecordRewards
execDeductStakingAmount
RoninValidatorSet
execRequestUpdateCommissionRate
execRequestRenounceCandidate
execEmergencyExit
execApplyValidatorCandidate
coinbase
submitBlockReward
wrapUpEpoch
execSlash
execBailOut
execReleaseLockedFundForEmergencyExitRequest
StakingVesting
requestBonus
receiveRON
Legend
Contract
Caller
External Function
Number = Order of calls
SlashIndicator
slashUnavailability
updateCreditScores
slashDoubleSign
bailOut
anyone
slashBridgeVoting
Maintenance
schedule
poolAdmin
RoninGovernanceAdmin
propose
proposeProposalStructAndCastVotes
proposeProposalForCurrentNetwork
createEmergencyExitPoll
governor
proposeGlobal
proposeGlobalProposalStructAndCastVotes
castProposalVoteForCurrentNetwork
voteEmergencyExit
anyone
deleteExpired
voteBridgeOperatorsBySignatures
castGlobalProposalBySignatures
castProposalBySignatures
RoninGatewayV2
withdrawal migrator
migrateWithdrawals
requestWithdrawalFor
bulkRequestWithdrawalFor
requestWithdrawalSignatures
anyone
bridge operator
bulkSubmitWithdrawalSignatures
tryBulkAcknowledgeMainchainWithdrew
tryBulkDepositFor
depositFor
BridgeTracking
recordVote
handleVoteApproved
RoninTrustedOrganization
addTrustedOrganizations
updateTrustedOrganizations
removeTrustedOrganizations
VaultForwarder
admin
withdrawAll

Mainchains



External Dependencies

The project relies on pre-compiled contracts for the following functionalities:

- sorting validators: `address(0x66)`
- validating double sign evidence: `address(0x67)`
- picking validator set: `address(0x68)`

It is worth noting that the set of validators can be partitioned into two groups: trusted organizations (whitelisted candidates who will always be selected) and standard validators (the candidates with the highest staking total).

Also, the project relies on relayer services for passing proposals from the Ronin chain to Mainchains and bridge operators to facilitate asset transfers between the Ronin chain and Mainchains.

The project uses OpenZeppelin library 4.6.0 for contract format, functionality as well as security and verification purposes. The following contracts & libraries are referenced in the current project:

- "access/AccessControlEnumerable.sol"
- "proxy/transparent/TransparentUpgradeableProxy.sol", "proxy/utils/Initializable.sol"
- "security/Pausable.sol", "security/ReentrancyGuard.sol",
- "token/ERC721/IERC721.sol", "token/ERC20/IERC20.sol"
- "utils/Strings.sol", "utils/cryptography/ECDSA.sol", "utils/StorageSlot.sol"

The above dependencies are not within the current audit scope and serve as a black box. Modules/contracts within the module are assumed to be valid and non-vulnerable actors in this audit and implement proper logic to collaborate with the current project and other modules.

DECENTRALIZATION EFFORTS | RONIN DPOS CONTRACTS

Description

To ensure proper project setup, access control, and upgradability, the Ronin protocol adopts multiple roles, including:

- Proxy Admin (`onlyAdmin`)
- MODERATOR_ROLE
- RELAYER_ROLE
- WITHDRAWAL_MIGRATOR
- DEFAULT_ADMIN_ROLE
- Coinbase (`onlyCoinbase`)
- Governor (`onlyGovernor`)

In the contract `MainchainGovernanceAdmin` , the `RELAYER_ROLE` has the authority over the following functions:

- `relayProposal()` : Relay a proposal.
- `relayGlobalProposal()` : Relay a global proposal.
- `relayBridgeOperators()` : Relay the bridge operators.

In the contract `CoinbaseExecution` , the Coinbase (`onlyCoinbase`) has the authority over the following functions:

- `submitBlockReward()` : Submit the block reward.
- `wrapUpEpoch()` : Wrapup an epoch.

In the contract `Forwarder` , the `MODERATOR_ROLE` has the authority to invoke the low-level calls to a given target address.

In the contract `RoninGovernanceAdmin` , the Governor (`onlyGovernor`) has the authority over the following functions:

- `propose()` : Make a proposal.
- `proposeProposalStructAndCastVotes()` : Make a proposal and cast vote.
- `proposeProposalForCurrentNetwork()` : Make a proposal on the current network.
- `castProposalVoteForCurrentNetwork()` : Cast vote for a proposal on the current network.
- `proposeGlobal()` : Make a global proposal.
- `proposeGlobalProposalStructAndCastVotes()` : Make a global proposal and cast vote.

In the contract `VaultForwarder` , the `DEFAULT_ADMIN_ROLE` has the authority over the following function:

- `withdrawAll` : Withdraw all the RON in the vault.

In the contract `HasBridgeTrackingContract` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setBridgeTrackingContract()` : Modify the bridge tracking contract.

In the contract `HasMaintenanceContract` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setMaintenanceContract()` : Modify the maintenance contract address.

In the contract `HasRoninGovernanceAdminContract` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setRoninGovernanceAdminContract()` : Modify the Governance Admin Contract address.

In the contract `HasRoninTrustedOrganizationContract` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setRoninTrustedOrganizationContract()` : Set the ronin trusted organization contract.

In the contract `HasSlashIndicatorContract` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setSlashIndicatorContract()` : Set the slash indicator contract.

In the contract `HasStakingContract` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setStakingContract()` : Set the staking contract.

In the contract `HasStakingVestingContract` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setStakingVestingContract()` : Set the staking vesting contract.

In the contract `HasValidatorContract` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setValidatorContract()` : Set the validator contract.

In the contract `GatewayV2` , the Proxy Admin (`onlyAdmin`) has the authority over the following functions:

- `setThreshold()` : Set the threshold for the quorum.
- `pause()` / `unpause()` : Pause/Unpause the contract.

In the contract `MinimumWithdrawal` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setMinimumThresholds()` : Set the minimum threshold for withdrawing.

In the contract `WithdrawalLimitation` , the Proxy Admin (`onlyAdmin`) has the authority over the following functions:

- `setThreshold()` : Set the high-tier vote weight threshold.
- `setHighTierVoteWeightThreshold()` : Set the high-tier vote weight threshold.

- `setHighTierThresholds()` : Set the thresholds for high-tier withdrawals that require high-tier vote weights.
- `setLockedThresholds()` : Set the amount thresholds to lock withdrawal.
- `setUnlockFeePercentages()` : Set fee percentages to unlock withdrawal.

In the contract `MainchainGatewayV2` , the Proxy Admin (`onlyAdmin`) has the authority over the following functions:

- `replaceBridgeOperators()` : Replace the operators of the bridge.
- `setWrappedNativeTokenContract()` : Set the address of the wrapped native token contract.
- `mapTokens()` : Set the mapping correlation between tokens across different chains, such as connecting the mainchain with Ronin.
- `mapTokensAndThresholds()` : Set the token mappings and threshold together.

In the contract `RoninTrustedOrganization` , the Proxy Admin (`onlyAdmin`) has the authority over the following functions:

- `setThreshold()` : Set the threshold of the quorum.
- `addTrustedOrganizations()` : Add trusted organizations.
- `updateTrustedOrganizations()` : Update trusted organizations.
- `removeTrustedOrganizations()` : Remove trusted organizations.

In the contract `CreditScore` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setCreditScoreConfigs()` : Set the credit score configurations.

In the contract `SlashBridgeOperator` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setBridgeOperatorSlashingConfigs()` : Set the bridge operator slashing configurations.

In the contract `SlashBridgeVoting` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setBridgeVotingSlashingConfigs()` : Set the bridge voting slashing configurations.

In the contract `SlashDoubleSign` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setDoubleSignSlashingConfigs()` : Set the double sign slashing configurations.

In the contract `SlashUnavailability` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setUnavailabilitySlashingConfigs()` : Set the unavailability slashing configurations.

In the contract `CandidateStaking` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setMinValidatorStakingAmount()` : Set the minimum validator staking amount.

In the `ValidatorInfoStorage` contract, the Proxy Admin (`onlyAdmin`) has the authority over the following functions:

- `setMaxValidatorNumber()` : Set the maximum validator number.
- `setMaxPrioritizedValidatorNumber()` : Set the maximum prioritized validator number.

In the `CandidateManager` contract, the Proxy Admin (`onlyAdmin`) has the authority over the following functions:

- `setMaxValidatorCandidate()` : Set the maximum candidate number.
- `setMinEffectiveDaysOnwards()` : Set the minimum effective days.

In the contract `EmergencyExit` , the Proxy Admin (`onlyAdmin`) has the authority over the following functions:

- `setEmergencyExitLockedAmount()` : Set emergency exit locked amount.
- `setEmergencyExpiryDuration()` : Set emergency expiry duration.
- `execReleaseLockedFundForEmergencyExitRequest()` : Execute release locked fund for emergency exit request.

In the contract `Maintenance` , the Proxy Admin (`onlyAdmin`) has the authority over the following function:

- `setMaintenanceConfig()` : Set the maintenance configurations.

In the contract `RoninGatewayV2` , the following roles are adopted:

- The Proxy Admin (`onlyAdmin`) has the authority over the following functions:
  - `setValidatorContract()` : Set the validator contract.
  - `setBridgeTrackingContract()` : Set the bridge tracking contract.
  - `mapTokens()` : Set the mapping correlation between tokens across different chains, such as connecting the mainchain with Ronin.
- The `WITHDRAWAL_MIGRATOR` has the authority over the following function:
  - `migrateWithdrawals()` : Migrate the withdrawals.

In the contract `StakingVesting` , the Proxy Admin (`onlyAdmin`) has the authority over the following functions:

- `setBlockProducerBonusPerBlock()` : Set the bonus per block for the block producer.
- `setBridgeOperatorBonusPerBlock()` : Set the bonus per block for the bridge operator.

Additionally, the following roles are intended to be connected to their respective contracts within the codebase.

- Validator Contract (`onlyValidatorContract`)
- Slash Indicator Contract (`onlySlashIndicatorContract`)
- Staking Contract (`onlyStakingContract`)

- Bridge Contract (`onlyBridgeContract`)

However, since the dependencies are not guaranteed by the implementation, considering possible setups or upgrades, misoperations of these roles could potentially bring risks to the project.

In the contract `CreditScore` , the validator Contract (`onlyValidatorContract`) has the authority over the following function:

- `updateCreditScores()` : Update the credit score.

In the contract `CandidateStaking` , the Validator Contract (`onlyValidatorContract`) has the authority over the following function:

- `deprecatePools()` : Deactivate a pool.

In the `staking` contract, the Validator Contract (`onlyValidatorContract`) has the authority over the following functions:

- `execRecordRewards()` : Execute recording the rewards.
- `execDeductStakingAmount()` : Execute reducing the staking amount of an address.

In the `CandidateManager` contract, the Staking Contract (`onlyStakingContract`) has the authority over the following functions:

- `execApplyValidatorCandidate()` : Gant a candidate.
- `execRequestRenounceCandidate()` : Revoking a candidate.
- `execRequestUpdateCommissionRate()` : Request to update commission rate.

In the contract `EmergencyExit` , the Staking Contract (`onlyStakingContract`) has the authority over the following function:

- `execEmergencyExit()` : Execute emergency exit.

In the contract `SlashingExecution` , the Slash Indicator Contract (`onlySlashIndicatorContract`) has the authority over the following functions:

- `execSlash()` : Execute slash for a validator address.
- `execBailOut()` : Execute bailout for a validator address.

In the contract `BridgeTracking` , the Bridge Contract (`onlyBridgeContract`) has the authority over the following functions:

- `handleVoteApproved()` : Update record for approved votes.
- `recordVote()` : Update the state of the vote.

In the contract `RoninGovernanceAdmin` , the Validator Contract (`onlyValidatorContract`) has the authority over the following function:

- `createEmergencyExitPoll()` : Create an emergency exit poll.

In the contract `StakingVesting` , the Validator Contract (`onlyValidatorContract`) has the authority over the following function:

- `requestBonus()` : Request bonus.

Finally, certain privileged roles are associated with corresponding components/dependencies that are not within the scope of the current audit, including precompiled contracts. These dependencies are treated as a blackbox during the audit and presumed to be functionally correct. More information can be found in the Review Notes section.

If the aforementioned roles are not managed or secured appropriately, attackers could take advantage of the associated privileges, potentially resulting in unexpected losses for the project.

Recommendations

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 of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key being 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 of 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.

FINDINGS | RONIN DPOS CONTRACTS

14	0	1	2	4	7
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Ronin DPoS Contracts. Through this audit, we have uncovered 14 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CEU-01	Validators May Have The Wrong Block Producer Status	Logical Issue	Major	Resolved
CGU-01	Possible To Create A Proposal That Cannot Be Voted On	Logical Issue	Medium	Resolved
ROI-01	Possible To Acquire Credit Score While In Maintenance	Logical Issue	Medium	Resolved
BOP-01	For Loop Should Not Return Early When Casting Vote For Bridge Operators	Logical Issue	Minor	Resolved
DSU-01	Possible For A Pool Admin To Delegate To A Different Pool	Inconsistency	Minor	Resolved
PAC-01	Potential Out-Dated Openzeppelin Library Usage	Language Specific	Minor	Resolved
ROR-01	Lack Of Check When Updating Trusted Organization	Inconsistency	Minor	Resolved
CEH-01	Modifier `oncePerEpoch` Invalid On First Epoch	Volatile Code	Informational	Acknowledged
CON-01	Incompatibility With Deflationary Tokens	Volatile Code	Informational	Acknowledged
CSI-01	Potential DoS Attack On Candidate Application	Logical Issue	Informational	Acknowledged
ROG-01	Potential Reentrancy Attack	Volatile Code	Informational	Resolved

ID	Title	Category	Severity	Status
ROO-01	Purpose Of Voting For Bridge Operators	Inconsistency	Informational	Resolved
SLD-01	Implementation Of Double Sign Slashing	Logical Issue	Informational	Resolved
SUU-01	Lack Of Check When Slashing For Unavailability	Logical Issue	Informational	Resolved

CEU-01 VALIDATORS MAY HAVE THE WRONG BLOCK PRODUCER STATUS

Category	Severity	Location	Status
Logical Issue	Major	contracts/ronin/validator/CoinbaseExecution.sol: 430, 436	Resolved

Description

It is possible for validators to have an incorrect block producer role as the array used to check if a validator is in maintenance is incorrect. Since being a block producer is necessary to receive mining rewards, this issue can cause a validator and their delegators to not be able to acquire rewards.

At the end of each epoch, the function `_revampRoles()` is called to perform checks on each validator in the array `_currentValidators` to decide the next epoch's block producers.

```
430      bool[] memory _maintainedList =
_maintenanceContract.checkManyMaintained(_candidates, block.number + 1);
431
432      for (uint _i = 0; _i < _currentValidators.length; _i++) {
433        address _validator = _currentValidators[_i];
434        bool _emergencyExitRequested = block.timestamp <=
_emergencyExitJailedTimestamp[_validator];
435        bool _isProducerBefore = isBlockProducer(_validator);
436        bool _isProducerAfter = !(_jailed(_validator) || _maintainedList[_i] ||
_emergencyExitRequested);
```

To be a block producer for the upcoming epoch, the validator cannot be jailed, in maintenance, or requested an emergency exit. The maintenance check is done by calling `checkManyMaintained()` in the maintenance contract.

The issue is that the call to `checkManyMaintained()` uses the `_candidates` array, which contains all addresses that can be a validator, while the `_currentValidators` array contains addresses that are validators for the next epoch.

In general, these arrays are different since the `_candidates` array is never sorted to have current validators be at the beginning of the array. The `_candidates` array changes in two situations:

1. When a new candidate is added, they are added to the end of the array
2. When a candidate is removed, the removed candidate is first switched with the last candidate of the array and then removed from the array

Due to the above, there can be no expectations regarding the order of the `_candidates` array.

Hence, from the function call `checkManyMaintained()`, the address `_maintainedList[_i]` corresponds to `_candidates[_i]`, but not necessarily `_currentValidators[_i]`.

Consequently, if `_candidates[_i]` is in maintenance, then `_currentValidators[_i]` will not be a block producer, even if it should be. This would mean that `_currentValidators[_i]` and their delegators cannot earn mining rewards.

Scenario

The following is a scenario that demonstrates the above issue:

1. Suppose we have two validator candidates, `consensusA` and `consensusB`, where `consensusA` is a trusted organization and `consensusB` applied to be a candidate before consensusA
    - Hence the `_candidates` array looks like `[consensusB, consensusA]`
2. Both candidates are chosen to be validators, so both are block producers
    - Since `consensusA` is a trusted organization, it appears first in the validators array
3. `consensusB` schedules for maintenance in the next epoch
4. When the epoch finishes, `consensusB` will retain the block producer status but `consensusA` will have this status removed due to the above issue

Proof of Concept

A proof-of-concept written in foundry is done in the function `testIncorrectBlockProducer()`. Changes to the source code are listed in the comments.

```
// Changes made:
// - removed _disableInitializers() from the following:
//   - src/ronin/staking/Staking.sol
//   - src/ronin/validator/RoninValidatorSet.sol
//   - src/ronin/slash-indicator/SlashIndicator.sol
//   - src/ronin/StakingVesting.sol
//   - src/ronin/Maintenance.sol
//   - src/ronin/BridgeTracking.sol
// - src/libraries/Math.sol: Math changed to RoninMath.
//   References to Math in various contracts changed to RoninMath
// - src/ronin/RoninGovernanceAdmin.sol: removed TransparentUpgradeableProxyV2
//   in _getWeight()
// - src/extensions/GovernanceAdmin.sol: removed TransparentUpgradeableProxyV2
//   in _getMinimumVoteWeight()
// - src/extensions/GovernanceAdmin.sol: removed TransparentUpgradeableProxyV2
//   in _getTotalWeights()

pragma solidity ^0.8.9;

import "forge-std/Test.sol";
import "../src/ronin/staking/Staking.sol";
import "../src/ronin/validator/RoninValidatorSet.sol";
import "../src/ronin/slash-indicator/SlashIndicator.sol";
import "../src/ronin/StakingVesting.sol";
import "../src/ronin/Maintenance.sol";
import "../src/multi-chains/RoninTrustedOrganization.sol";
import "../src/ronin/RoninGovernanceAdmin.sol";
import "../src/ronin/BridgeTracking.sol";
import "../src/ronin/RoninGatewayV2.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract StakingValidatorTest is Test {

    Staking staking;
    RoninValidatorSet roninValidatorSet;
    SlashIndicator slashIndicator;
    StakingVesting stakingVesting;
    Maintenance maintenance;
    RoninTrustedOrganization roninTrustedOrganization;
    RoninGovernanceAdmin roninGovernanceAdmin;
    RoninGatewayV2 roninGateway;
    BridgeTracking bridgeTracking;

    // Trusted Org Config
    address consensusAddr = vm.addr(1);
    address governor = vm.addr(2);
    address bridgeVoter = vm.addr(3);
```

```
    // Token Config
    address roninToken = address(new ERC20("Ronin Token", "RNT"));
    address mainchainToken = address(new ERC20("Mainchain Token", "MCT"));

    function setUp() public {
        roninValidatorSet = new RoninValidatorSet();
        staking = new Staking();
        slashIndicator = new SlashIndicator();
        stakingVesting = new StakingVesting();
        maintenance = new Maintenance();
        bridgeTracking = new BridgeTracking();

        _deployRoninTrustedOrg();
        _deployRoninGateway();
        _deployRoninGovernanceAdmin();

        _initializeStaking();
        _initializeValidator();
        _initializeSlashIndicator();
        _initializeStakingVesting();
        _initializeMaintenance();
        _initializeBridgeTracking();

        _deployPickValidatorSet();
    }

    function testIncorrectBlockProducer() public {
        address poolAdminB = vm.addr(10);
        address consensusAddrB = vm.addr(11);
        address bridgeOperatorB = vm.addr(12);

        vm.deal(poolAdminB, 1e18);
        vm.prank(poolAdminB);
        staking.applyValidatorCandidate{ value: 1000 }(
            poolAdminB,
            consensusAddrB,
            payable(poolAdminB),
            bridgeOperatorB,
            10
        );

        // create validators
        vm.deal(governor, 1e18);
        vm.prank(governor);
        staking.applyValidatorCandidate{ value: 1000 }(
            governor,
            consensusAddr,
            payable(governor),
            bridgeVoter,
```

```
            10
        );

        // setup initial validators
        vm.coinbase(address(this));
        vm.roll(199); // to satisfy whenEpochEnding oncePerEpoch modifier
        vm.warp(block.timestamp + 1 days + 1); // to be in a new period
        roninValidatorSet.wrapUpEpoch();

        // check both are block producers
        assert(roninValidatorSet.isBlockProducer(consensusAddr));
        assert(roninValidatorSet.isBlockProducer(consensusAddrB));

        // put consensusAddrB validator in maintenance
        vm.prank(poolAdminB);
        maintenance.schedule(
            consensusAddrB,
            block.number + 1, // 200
            399
        );

        // revamp roles
        vm.roll(299);
        roninValidatorSet.wrapUpEpoch();

        // the validator in maintenance is still a block producer
        // the validator not in maintenance is no longer a block producer
        assert(!roninValidatorSet.isBlockProducer(consensusAddr));
        assert(roninValidatorSet.isBlockProducer(consensusAddrB));

        // this is because the validator and candidates array are different
        address[] memory validators = roninValidatorSet.getValidators();
        address[] memory candidates = roninValidatorSet.getValidatorCandidates();

        assert(validators[0] == candidates[1]);
        assert(validators[1] == candidates[0]);
    }

    function _initializeStaking() internal {
        staking.initialize(
            address(roninValidatorSet),
            20, // minValidatorStakingAmount
            3 * 86400, // cooldownSecsToUndelegate
            7 * 86400 // waitingSecsToRevoke
        );
    }

    function _initializeValidator() internal {
        uint256[2] memory emergencyExitConfigs;
```

```
        emergencyExitConfigs[0] = 500; // emergencyExitLockedAmount
        emergencyExitConfigs[1] = 14 * 86400; // emergencyExpiryDuration

        roninValidatorSet.initialize(
            address(slashIndicator),
            address(staking),
            address(stakingVesting),
            address(maintenance),
            address(roninTrustedOrganization),
            address(bridgeTracking),
            100, // maxValidatorNumber
            100, // maxValidatorCandidate
            100, // maxPrioritizedValidatorNumber
            1, // minEffectiveDaysOnwards
            100, // numberOfBlocksInEpoch
            emergencyExitConfigs
        );
    }

    function _initializeSlashIndicator() internal {
        uint256[4] memory _bridgeOperatorSlashingConfigs;
        _bridgeOperatorSlashingConfigs[0] = 5; // _missingVotesRatioTier1
        _bridgeOperatorSlashingConfigs[1] = 10; // _missingVotesRatioTier2
        _bridgeOperatorSlashingConfigs[2] = 5; //
_jailDurationForMissingVotesRatioTier2
        _bridgeOperatorSlashingConfigs[3] = 10; //
_skipBridgeOperatorSlashingThreshold

        uint256[2] memory _bridgeVotingSlashingConfigs;
        _bridgeVotingSlashingConfigs[0] = 10; // _bridgeVotingThreshold
        _bridgeVotingSlashingConfigs[1] = 100; // _bridgeVotingSlashAmount

        uint256[2] memory _doubleSignSlashingConfigs;
        _doubleSignSlashingConfigs[0] = 100; // _slashDoubleSignAmount
        _doubleSignSlashingConfigs[1] = 5000; // _doubleSigningJailUntilBlock

        uint256[4] memory _unavailabilitySlashingConfigs;
        _unavailabilitySlashingConfigs[0] = 5; // _unavailabilityTier1Threshold
        _unavailabilitySlashingConfigs[1] = 10; // _unavailabilityTier2Threshold
        _unavailabilitySlashingConfigs[2] = 100; //
_slashAmountForUnavailabilityTier2Threshold
        _unavailabilitySlashingConfigs[3] = 100; //
_jailDurationForUnavailabilityTier2Threshold

        uint256[4] memory _creditScoreConfigs;
        _creditScoreConfigs[0] = 5; // _gainCreditScore
        _creditScoreConfigs[1] = 100; // _maxCreditScore
        _creditScoreConfigs[2] = 0; // _bailOutCostMultiplier
        _creditScoreConfigs[3] = 10; // _cutOffPercentageAfterBailout
```

```
        slashIndicator.initialize(
            address(roninValidatorSet),
            address(maintenance),
            address(roninTrustedOrganization),
            address(roninGovernanceAdmin),
            _bridgeOperatorSlashingConfigs,
            _bridgeVotingSlashingConfigs,
            _doubleSignSlashingConfigs,
            _unavailabilitySlashingConfigs,
            _creditScoreConfigs
        );
    }

    function _initializeStakingVesting() internal {
        stakingVesting.initialize(
            address(roninValidatorSet),
            100, // blockProducerBonusPerBlock
            100 // bridgeOperatorBonusPerBlock
        );
        vm.deal(address(stakingVesting), 1e18);
    }

    function _initializeMaintenance() internal {
        maintenance.initialize(
            address(roninValidatorSet),
            0, // minMaintenanceDurationInBlock
            1000, // maxMaintenanceDurationInBlock
            1, // minOffsetToStartSchedule
            1000, // maxOffsetToStartSchedule
            100 // maxSchedules
        );
    }

    function _initializeBridgeTracking() internal {
        bridgeTracking.initialize(
            address(roninGateway),
            address(roninValidatorSet),
            block.number // startedAtBlock
        );
    }

    function _deployRoninTrustedOrg() internal {
        roninTrustedOrganization = new RoninTrustedOrganization();

        IRoninTrustedOrganization.TrustedOrganization memory trustedOrg =
            IRoninTrustedOrganization.TrustedOrganization(
                consensusAddr,
                governor,
```

```
                bridgeVoter,
                100, // weight
                0 // added block
            );

        IRoninTrustedOrganization.TrustedOrganization[] memory trustedOrgs =
            new IRoninTrustedOrganization.TrustedOrganization[](1);
        trustedOrgs[0] = trustedOrg;

        roninTrustedOrganization.initialize(
            trustedOrgs,
            1, // numerator
            1 // denominator
        );
    }

    function _deployRoninGateway() internal {
        roninGateway = new RoninGatewayV2();

        address[] memory _withdrawalMigrators = new address[](1);
        _withdrawalMigrators[0] = address(this);

        address[][2] memory _packedAddresses;
        _packedAddresses[0] = new address[](1);
        _packedAddresses[0][0] = roninToken;
        _packedAddresses[1] = new address[](1);
        _packedAddresses[1][0] = mainchainToken;

        uint256[][2] memory _packedNumbers;
        _packedNumbers[0] = new uint256[](1);
        _packedNumbers[0][0] = block.chainid; // ronin chain id
        _packedNumbers[1] = new uint256[](1);
        _packedNumbers[1][0] = 0; // min threshold

        Token.Standard[] memory _standards = new Token.Standard[](1);
        _standards[0] = Token.Standard.ERC20;

        roninGateway.initialize(
            address(this), // role setter
            1, // numerator
            1, // denominator
            _withdrawalMigrators,
            _packedAddresses,
            _packedNumbers,
            _standards
        );
    }

    function _deployRoninGovernanceAdmin() internal {
```

```
        roninGovernanceAdmin = new RoninGovernanceAdmin(
            2020, // ronin chain id
            address(roninTrustedOrganization),
            address(roninGateway),
            address(roninValidatorSet),
            100 // proposalExpiryDuration
        );
    }

    function _deployPickValidatorSet() internal {
        PickValidatorSet pickValidatorSet = new PickValidatorSet();
        bytes memory code = address(pickValidatorSet).code;
        address targetAddr = address(0x68);
        vm.etch(targetAddr, code);
    }
}

contract PickValidatorSet {
    function pickValidatorSet(
        address[] calldata candidates,
        uint256[] calldata weights,
        uint256[] calldata trustedWeights,
        uint256 maxValidatorNumber,
        uint256 maxPrioritizedValidatorNumber
    ) external pure returns (address[] memory) {
        uint256 len = candidates.length;
        address[] memory newValidators = new address[](len);

        for (uint256 i = 0; i < len; i++) {
            newValidators[i] = candidates[len - 1 - i];
        }

        return newValidators;
    }
}
```

Recommendation

Recommend checking maintenance on the array `_currentValidators` instead of `_candidates`.

Alleviation

[**Ronin Team**, 03/03/2023]: The team acknowledged this issue and fixed it in commit 6e2566235009c9e85f1869233ba6966d58ad6dd4 by using the correct array.

CGU-01 POSSIBLE TO CREATE A PROPOSAL THAT CANNOT BE VOTED ON

Category	Severity	Location	Status
Logical Issue	Medium	contracts/extensions/sequential-governance/CoreGovernance.sol: 116, 173	Resolved

Description

It is possible to create a proposal whose nonce does not match with the current round if the previous proposal expired, making it impossible to vote for the proposal.

When a proposal is created via `_proposeProposal()`, the nonce of the proposal is set to the current round **plus one**.

```
function _proposeProposal(
  ...
) internal virtual returns (Proposal.ProposalDetail memory _proposal) {
  require(_chainId != 0, "CoreGovernance: invalid chain id");

  _proposal = Proposal.ProposalDetail(
    round[_chainId] + 1,
    _chainId,
    _expiryTimestamp,
    _targets,
    _values,
    _calldatas,
    _gasAmounts
  );
  _proposal.validate(_proposalExpiryDuration);

  bytes32 _proposalHash = _proposal.hash();
  uint256 _round = _createVotingRound(_chainId, _proposalHash, _expiryTimestamp);
  emit ProposalCreated(_chainId, _round, _proposalHash, _proposal, _creator);
}
```

Even though the nonce of the proposal is set to `round[_chainId] + 1`, `_proposeProposal()` calls `_createVotingRound()`, which decides the current round and can be different from `round[chainId] + 1`.

```
  function _createVotingRound(
    ...
  ) internal returns (uint256 _round) {
    _round = round[_chainId];

    // Skip checking for the first ever round
    if (_round == 0) {
      _round = round[_chainId] = 1;
    } else {
      ProposalVote storage _latestProposalVote = vote[_chainId][_round];
      bool _isExpired = _tryDeleteExpiredVotingRound(_latestProposalVote);
      // Skip increase round number if the latest round is expired, allow the vote
to be overridden
      if (!_isExpired) {
        require(_latestProposalVote.status != VoteStatus.Pending, "CoreGovernance:
current proposal is not completed");
        _round = ++round[_chainId];
      }
    }

    vote[_chainId][_round].hash = _proposalHash;
    vote[_chainId][_round].expiryTimestamp = _expiryTimestamp;
  }
```

In particular, if the previous proposal expired, the round of the proposal will still be `round[chainId]` instead of `round[chainId] + 1`.

This prevents voting on the proposal because when casting a vote, there is a check to ensure that the proposal's nonce matches with the current round.

```
  function _castVote(
    ...
  ) internal virtual returns (bool _done) {
    uint256 _chainId = _proposal.chainId;
    uint256 _round = _proposal.nonce;
    ProposalVote storage _vote = vote[_chainId][_round];

    ...
    require(round[_proposal.chainId] == _round, "CoreGovernance: query for invalid
proposal nonce");
```

This also prevents functions that have a hash check as the hash of a proposal includes the proposal's nonce. For example, in `_castProposalVoteForCurrentNetwork()`:

```
  function _castProposalVoteForCurrentNetwork(
    ...
  ) internal {
    require(_proposal.chainId == block.chainid, "RoninGovernanceAdmin: invalid chain
id");
    require(
      vote[_proposal.chainId][_proposal.nonce].hash == _proposal.hash(),
      "RoninGovernanceAdmin: cast vote for invalid proposal"
    );
```

The same issue exists in `_proposeGlobal()`, where the nonce of the proposal is decided before the round.

```
  function _proposeGlobal(
    ...
  ) internal virtual returns (uint256 _round) {
    GlobalProposal.GlobalProposalDetail memory _globalProposal =
GlobalProposal.GlobalProposalDetail(
      round[0] + 1,
      _expiryTimestamp,
      _targetOptions,
      _values,
      _calldatas,
      _gasAmounts
    );
    Proposal.ProposalDetail memory _proposal = _globalProposal.into_proposal_detail(
      _roninTrustedOrganizationContract,
      _gatewayContract
    );
    _proposal.validate(_proposalExpiryDuration);

    bytes32 _proposalHash = _proposal.hash();
    _round = _createVotingRound(0, _proposalHash, _expiryTimestamp);
    emit GlobalProposalCreated(_round, _proposalHash, _proposal,
_globalProposal.hash(), _globalProposal, _creator);
  }
```

The issue can be fixed by proposing a proposal using the function `_proposeProposalStruct()`, or `_proposeGlobalStruct()` for global proposals`, as the nonce of the proposal can be manually set to match the expected round.

This fix does require the invalid proposal to first expire, which may take a long time.

Scenario

Two scenarios are provided to demonstrate the above issue.

The first scenario shows how voting can be prevented by performing the following:

1. Create a proposal using `RoninGovernanceAdmin.proposeProposalForCurrentNetwork()`
    - Note that this function also has the function caller vote on the proposal
2. After the proposal expires, create the same proposal using `RoninGovernanceAdmin.proposeProposalForCurrentNetwork()`
3. This call will revert with the message "RoninGovernanceAdmin: cast vote for invalid proposal" as the hash of the proposal is incorrect due to an incorrect nonce

The second scenario creates a proposal and directly shows that its nonce does not match the current round by performing the following:

1. Create a proposal using `RoninGovernanceAdmin.propose()`
2. After the proposal expires, create the same proposal using `RoninGovernanceAdmin.propose()`
3. Check that the proposal's nonce and current round are not the same

Proof of Concept

The following proof of concept written in foundry is provided to demonstrate the above two scenarios. The function `testCannotVoteOnProposal()` showcases the first scenario while `testIncorrectProposalNonce()` showcases the second scenario.

Changes to the source code are stated in the comments.

```
// Changes made:
// - src/libraries/Math.sol: Math changed to RoninMath (to deal with compiler
issue).
//   References to Math in various contracts changed to RoninMath
// - src/ronin/RoninGovernanceAdmin.sol: removed TransparentUpgradeableProxyV2
//   in _getWeight()
// - src/extensions/GovernanceAdmin.sol: removed TransparentUpgradeableProxyV2
//   in _getMinimumVoteWeight()
// - src/extensions/GovernanceAdmin.sol: removed TransparentUpgradeableProxyV2
//   in _getTotalWeights()

pragma solidity ^0.8.9;

import "forge-std/Test.sol";
import "../src/ronin/RoninGovernanceAdmin.sol";
import "../src/multi-chains/RoninTrustedOrganization.sol";

contract RoninGovernanceTest is Test {

    RoninGovernanceAdmin roninGovernanceAdmin;
    RoninTrustedOrganization roninTrustedOrganization;
    address bridgeContract = vm.addr(100); // placeholder as contract is unused
    address validatorContract = vm.addr(101); // placeholder as contract is unused

    address consensusAddr = vm.addr(1);
    address governor = vm.addr(2);
    address bridgeVoter = vm.addr(3);

    address consensusAddrB = vm.addr(11);
    address governorB = vm.addr(12);
    address bridgeVoterB = vm.addr(13);

    uint256 proposalExpiryDuration = 100;

    function setUp() public {
        _deployRoninTrustedOrg();
        _deployRoninGovernanceAdmin();
    }

    function testCannotVoteOnProposal() public {
        // proposal parameters
        address[] memory targets = new address[](1);
        targets[0] = vm.addr(7777);

        uint256[] memory values = new uint256[](1);
        values[0] = 7777;

        bytes[] memory calldatas = new bytes[](1);
        calldatas[0] = new bytes(7777);
```

```
        uint256[] memory gasAmounts = new uint256[](1);
        gasAmounts[0] = 7777;

        Ballot.VoteType support = Ballot.VoteType.Against;

        // create a proposal and vote on it
        vm.prank(governor);
        roninGovernanceAdmin.proposeProposalForCurrentNetwork(
            block.timestamp + 1, // expiry timestamp
            targets,
            values,
            calldatas,
            gasAmounts,
            support
        );

        // increase time so that previous proposal expires
        vm.warp(block.timestamp + 10);

        // create the same proposal, but call will revert due to invalid hash
        vm.prank(governor);
        vm.expectRevert("RoninGovernanceAdmin: cast vote for invalid proposal");
        roninGovernanceAdmin.proposeProposalForCurrentNetwork(
            block.timestamp + 1, // expiry timestamp
            targets,
            values,
            calldatas,
            gasAmounts,
            support
        );
    }

    // Change in src/ronin/RoninGovernanceAdmin.sol:
    // propose() returns Proposal.ProposalDetail

    function testIncorrectProposalNonce() public {
        // proposal parameters
        address[] memory targets = new address[](1);
        targets[0] = vm.addr(7777);

        uint256[] memory values = new uint256[](1);
        values[0] = 7777;

        bytes[] memory calldatas = new bytes[](1);
        calldatas[0] = new bytes(7777);

        uint256[] memory gasAmounts = new uint256[](1);
        gasAmounts[0] = 7777;
```

```
        // create a proposal
        vm.prank(governor);
        roninGovernanceAdmin.propose(
            1, // chain id
            block.timestamp + 1, // expiry timestamp
            targets,
            values,
            calldatas,
            gasAmounts
        );

        // increase time so that previous proposal expires
        vm.warp(block.timestamp + 10);

        // create the same proposal again
        vm.prank(governor);
        Proposal.ProposalDetail memory proposal = roninGovernanceAdmin.propose(
            1, // chain id
            block.timestamp + 1, // expiry timestamp
            targets,
            values,
            calldatas,
            gasAmounts
        );

        // check that current round is not the proposal's nonce
        uint256 currentRound = roninGovernanceAdmin.round(1);
        assert(currentRound == 1);
        assert(proposal.nonce == 2);
    }

    function _deployRoninTrustedOrg() internal {
        roninTrustedOrganization = new RoninTrustedOrganization();

        IRoninTrustedOrganization.TrustedOrganization memory trustedOrg =
            IRoninTrustedOrganization.TrustedOrganization(
                consensusAddr,
                governor,
                bridgeVoter,
                1, // weight
                0 // added block
            );
        IRoninTrustedOrganization.TrustedOrganization memory trustedOrgB =
            IRoninTrustedOrganization.TrustedOrganization(
                consensusAddrB,
                governorB,
                bridgeVoterB,
                100, // weight
```

```
                0 // added block
            );

        IRoninTrustedOrganization.TrustedOrganization[] memory trustedOrgs =
            new IRoninTrustedOrganization.TrustedOrganization[](2);
        trustedOrgs[0] = trustedOrg;
        trustedOrgs[1] = trustedOrgB;

        roninTrustedOrganization.initialize(
            trustedOrgs,
            1, // numerator
            2 // denominator
        );
    }

    function _deployRoninGovernanceAdmin() internal {
        roninGovernanceAdmin = new RoninGovernanceAdmin(
            2020, // ronin chain id
            address(roninTrustedOrganization),
            bridgeContract,
            validatorContract,
            proposalExpiryDuration
        );
    }
}
```

Recommendation

Recommend first checking if the current proposal has expired and if so, the nonce of the new proposal should be `round[chainId]`, otherwise if the proposal is not pending, `round[chainId] + 1`.

Alleviation

[**Ronin Team**, 03/03/2023]: The team acknowledged this issue and fixed it in commit ddbdfc803154f04c8e6eedb3e7073b2fb5142c0f by first checking for expiration and then deciding the round number.

ROI-01 POSSIBLE TO ACQUIRE CREDIT SCORE WHILE IN MAINTENANCE

Category	Severity	Location	Status
Logical Issue	Medium	contracts/ronin/slash-indicator/CreditScore.sol: 36; contracts/ronin/validator/CoinbaseExecution.sol: 107	Resolved

Description

Validators in maintenance will never be considered to be in maintenance when updating credit scores due to an incorrect value of the variable `_periodStartAtBlock`. This allows validators to acquire credit scores while in maintenance.

The function `updateCreditScores()` is called by `wrapUpEpoch()`, which occurs at the end of an epoch.

```
  function wrapUpEpoch() external payable virtual override onlyCoinbase
whenEpochEnding oncePerEpoch {
    ...

    if (_periodEnding) {
      _currentPeriodStartAtBlock = block.number + 1;
      ...
      _slashIndicatorContract.updateCreditScores(_currentValidators, _lastPeriod);
```

Note that `_currentPeriodStartAtBlock` is updated to `block.number + 1` before `updateCreditScores()` is called.

The function `updateCreditScores()` then calls `currentPeriodStartAtBlock()`, setting `_periodStartAtBlock == block.number + 1`, and this return value is used in `checkManyMaintainedInBlockRange()`.

```
  function updateCreditScores(address[] calldata _validators, uint256 _period)
external override onlyValidatorContract {
    uint256 _periodStartAtBlock = _validatorContract.currentPeriodStartAtBlock();

    bool[] memory _jaileds = _validatorContract.checkManyJailed(_validators);
    bool[] memory _maintaineds =
_maintenanceContract.checkManyMaintainedInBlockRange(
      _validators,
      _periodStartAtBlock,
      block.number
    );
```

The function `checkManyMaintainedInBlockRange()` checks to see if validators are in maintenance within the block range `[_periodStartAtBlock, block.number]`. Since `_periodStartAtBlock == block.number + 1 > block.number`, no validators will be considered to be in maintenance.

As validators in maintenance are to receive no credit score, this issue actually allows such validators to acquire credit scores for the period.

Proof of Concept

A proof of concept written in foundry is provided that demonstrates the above issue, where a validator in maintenance is able to acquire credit score. This is done by the function `testIncorrectMaintenance()`.

Changes to the source code are listed in the comments and a fake `PickValidatorSet` contract was created in order to choose validators.

```
// Changes made:
// - removed _disableInitializers() from the following:
//   - src/ronin/staking/Staking.sol
//   - src/ronin/validator/RoninValidatorSet.sol
//   - src/ronin/slash-indicator/SlashIndicator.sol
//   - src/ronin/StakingVesting.sol
//   - src/ronin/Maintenance.sol
//   - src/ronin/BridgeTracking.sol
// - src/libraries/Math.sol: Math changed to RoninMath.
//   References to Math in various contracts changed to RoninMath
// - src/ronin/RoninGovernanceAdmin.sol: removed TransparentUpgradeableProxyV2
//   in _getWeight()
// - src/extensions/GovernanceAdmin.sol: removed TransparentUpgradeableProxyV2
//   in _getMinimumVoteWeight()
// - src/extensions/GovernanceAdmin.sol: removed TransparentUpgradeableProxyV2
//   in _getTotalWeights()

pragma solidity ^0.8.9;

import "forge-std/Test.sol";
import "../src/ronin/staking/Staking.sol";
import "../src/ronin/validator/RoninValidatorSet.sol";
import "../src/ronin/slash-indicator/SlashIndicator.sol";
import "../src/ronin/StakingVesting.sol";
import "../src/ronin/Maintenance.sol";
import "../src/multi-chains/RoninTrustedOrganization.sol";
import "../src/ronin/RoninGovernanceAdmin.sol";
import "../src/ronin/BridgeTracking.sol";
import "../src/ronin/RoninGatewayV2.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract StakingValidatorTest is Test {

    Staking staking;
    RoninValidatorSet roninValidatorSet;
    SlashIndicator slashIndicator;
    StakingVesting stakingVesting;
    Maintenance maintenance;
    RoninTrustedOrganization roninTrustedOrganization;
    RoninGovernanceAdmin roninGovernanceAdmin;
    RoninGatewayV2 roninGateway;
    BridgeTracking bridgeTracking;

    // Trusted Org Config
    address consensusAddr = vm.addr(1);
    address governor = vm.addr(2);
    address bridgeVoter = vm.addr(3);
```

```
    // Token Config
    address roninToken = address(new ERC20("Ronin Token", "RNT"));
    address mainchainToken = address(new ERC20("Mainchain Token", "MCT"));

    function setUp() public {
        roninValidatorSet = new RoninValidatorSet();
        staking = new Staking();
        slashIndicator = new SlashIndicator();
        stakingVesting = new StakingVesting();
        maintenance = new Maintenance();
        bridgeTracking = new BridgeTracking();

        _deployRoninTrustedOrg();
        _deployRoninGateway();
        _deployRoninGovernanceAdmin();

        _initializeStaking();
        _initializeValidator();
        _initializeSlashIndicator();
        _initializeStakingVesting();
        _initializeMaintenance();
        _initializeBridgeTracking();

        _deployPickValidatorSet();
    }

    function testIncorrectMaintenance() public {
        // create a validator
        vm.deal(governor, 1e18);
        vm.prank(governor);
        staking.applyValidatorCandidate{ value: 1000 }(
            governor,
            consensusAddr,
            payable(governor),
            bridgeVoter,
            10
        );

        // setup initial validators
        vm.coinbase(address(this));
        vm.roll(199); // to satisfy whenEpochEnding oncePerEpoch modifier
        vm.warp(block.timestamp + 1 days + 1); // to be in a new period
        roninValidatorSet.wrapUpEpoch();
        assert(slashIndicator.getCreditScore(consensusAddr) == 0); // as was not a
validator yet

        // test first credit score
        vm.roll(299);
        vm.warp(block.timestamp + 1 days + 1);
```

```
        roninValidatorSet.wrapUpEpoch();
        assert(slashIndicator.getCreditScore(consensusAddr) == 5); // gained credit

        // put validator in maintenance
        vm.prank(governor);
        maintenance.schedule(
            consensusAddr,
            block.number + 1, // 300
            399
        );

        // gained credit score even though in maintenance
        vm.roll(399);
        vm.warp(block.timestamp + 1 days + 1);
        roninValidatorSet.wrapUpEpoch();
        assert(slashIndicator.getCreditScore(consensusAddr) == 10); // gained credit
    }

    function _initializeStaking() internal {
        staking.initialize(
            address(roninValidatorSet),
            20, // minValidatorStakingAmount
            3 * 86400, // cooldownSecsToUndelegate
            7 * 86400 // waitingSecsToRevoke
        );
    }

    function _initializeValidator() internal {
        uint256[2] memory emergencyExitConfigs;
        emergencyExitConfigs[0] = 500; // emergencyExitLockedAmount
        emergencyExitConfigs[1] = 14 * 86400; // emergencyExpiryDuration

        roninValidatorSet.initialize(
            address(slashIndicator),
            address(staking),
            address(stakingVesting),
            address(maintenance),
            address(roninTrustedOrganization),
            address(bridgeTracking),
            100, // maxValidatorNumber
            100, // maxValidatorCandidate
            100, // maxPrioritizedValidatorNumber
            1, // minEffectiveDaysOnwards
            100, // numberOfBlocksInEpoch
            emergencyExitConfigs
        );
    }

    function _initializeSlashIndicator() internal {
```

```
        uint256[4] memory _bridgeOperatorSlashingConfigs;
        _bridgeOperatorSlashingConfigs[0] = 5; // _missingVotesRatioTier1
        _bridgeOperatorSlashingConfigs[1] = 10; // _missingVotesRatioTier2
        _bridgeOperatorSlashingConfigs[2] = 5; //
_jailDurationForMissingVotesRatioTier2
        _bridgeOperatorSlashingConfigs[3] = 10; //
_skipBridgeOperatorSlashingThreshold

        uint256[2] memory _bridgeVotingSlashingConfigs;
        _bridgeVotingSlashingConfigs[0] = 10; // _bridgeVotingThreshold
        _bridgeVotingSlashingConfigs[1] = 100; // _bridgeVotingSlashAmount

        uint256[2] memory _doubleSignSlashingConfigs;
        _doubleSignSlashingConfigs[0] = 100; // _slashDoubleSignAmount
        _doubleSignSlashingConfigs[1] = 5; // _doubleSigningJailUntilBlock

        uint256[4] memory _unavailabilitySlashingConfigs;
        _unavailabilitySlashingConfigs[0] = 5; // _unavailabilityTier1Threshold
        _unavailabilitySlashingConfigs[1] = 10; // _unavailabilityTier2Threshold
        _unavailabilitySlashingConfigs[2] = 100; //
_slashAmountForUnavailabilityTier2Threshold
        _unavailabilitySlashingConfigs[3] = 100; //
_jailDurationForUnavailabilityTier2Threshold

        uint256[4] memory _creditScoreConfigs;
        _creditScoreConfigs[0] = 5; // _gainCreditScore
        _creditScoreConfigs[1] = 100; // _maxCreditScore
        _creditScoreConfigs[2] = 5; // _bailOutCostMultiplier
        _creditScoreConfigs[3] = 10; // _cutOffPercentageAfterBailout

        slashIndicator.initialize(
            address(roninValidatorSet),
            address(maintenance),
            address(roninTrustedOrganization),
            address(roninGovernanceAdmin),
            _bridgeOperatorSlashingConfigs,
            _bridgeVotingSlashingConfigs,
            _doubleSignSlashingConfigs,
            _unavailabilitySlashingConfigs,
            _creditScoreConfigs
        );
    }

    function _initializeStakingVesting() internal {
        stakingVesting.initialize(
            address(roninValidatorSet),
            100, // blockProducerBonusPerBlock
            100 // bridgeOperatorBonusPerBlock
        );
```

```
    }

    function _initializeMaintenance() internal {
        maintenance.initialize(
            address(roninValidatorSet),
            0, // minMaintenanceDurationInBlock
            1000, // maxMaintenanceDurationInBlock
            1, // minOffsetToStartSchedule
            1000, // maxOffsetToStartSchedule
            100 // maxSchedules
        );
    }

    function _initializeBridgeTracking() internal {
        bridgeTracking.initialize(
            address(roninGateway),
            address(roninValidatorSet),
            block.number // startedAtBlock
        );
    }

    function _deployRoninTrustedOrg() internal {
        roninTrustedOrganization = new RoninTrustedOrganization();

        IRoninTrustedOrganization.TrustedOrganization memory trustedOrg =
            IRoninTrustedOrganization.TrustedOrganization(
                consensusAddr,
                governor,
                bridgeVoter,
                100, // weight
                0 // added block
            );

        IRoninTrustedOrganization.TrustedOrganization[] memory trustedOrgs =
            new IRoninTrustedOrganization.TrustedOrganization[](1);
        trustedOrgs[0] = trustedOrg;

        roninTrustedOrganization.initialize(
            trustedOrgs,
            1, // numerator
            1 // denominator
        );
    }

    function _deployRoninGateway() internal {
        roninGateway = new RoninGatewayV2();

        address[] memory _withdrawalMigrators = new address[](1);
        _withdrawalMigrators[0] = address(this);
```

```
        address[][2] memory _packedAddresses;
        _packedAddresses[0] = new address[](1);
        _packedAddresses[0][0] = roninToken;
        _packedAddresses[1] = new address[](1);
        _packedAddresses[1][0] = mainchainToken;

        uint256[][2] memory _packedNumbers;
        _packedNumbers[0] = new uint256[](1);
        _packedNumbers[0][0] = block.chainid; // ronin chain id
        _packedNumbers[1] = new uint256[](1);
        _packedNumbers[1][0] = 0; // min threshold

        Token.Standard[] memory _standards = new Token.Standard[](1);
        _standards[0] = Token.Standard.ERC20;

        roninGateway.initialize(
            address(this), // role setter
            1, // numerator
            1, // denominator
            _withdrawalMigrators,
            _packedAddresses,
            _packedNumbers,
            _standards
        );
    }

    function _deployRoninGovernanceAdmin() internal {
        roninGovernanceAdmin = new RoninGovernanceAdmin(
            2020, // ronin chain id
            address(roninTrustedOrganization),
            address(roninGateway),
            address(roninValidatorSet),
            100 // proposalExpiryDuration
        );
    }

    function _deployPickValidatorSet() internal {
        PickValidatorSet pickValidatorSet = new PickValidatorSet();
        bytes memory code = address(pickValidatorSet).code;
        address targetAddr = address(0x68);
        vm.etch(targetAddr, code);
    }
}

contract PickValidatorSet {
    function pickValidatorSet(
        address[] calldata candidates,
        uint256[] calldata weights,
```

```
        uint256[] calldata trustedWeights,
        uint256 maxValidatorNumber,
        uint256 maxPrioritizedValidatorNumber
    ) external pure returns (address[] memory) {
        return candidates;
    }
}
```

Recommendation

Recommend first updating credit scores and then updating the `_currentPeriodStartAtBlock` variable in the `CoinbaseExecution` contract.

Alleviation

[**Ronin Team**, 03/03/2023]: The team acknowledged this issue and fixed it in commit f584d65c5534fa6577e41362d9f8dde1f008e9a1 by updating `_currentPeriodStartAtBlock` after updating credit scores.

BOP-01 FOR LOOP SHOULD NOT RETURN EARLY WHEN CASTING VOTE FOR BRIDGE OPERATORS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/extensions/isolated-governance/bridge-operator-governance/BOsGovernanceProposal.sol: 85	Resolved

Description

When governors cast votes for a set of bridge operators, a `for` loop is done on an array of signatures, casting a vote for each signature.

```
 69      for (uint256 _i = 0; _i < _signatures.length; _i++) {
 70        // Avoids stack too deeps
 71        {
 72          Signature calldata _sig = _signatures[_i];
 73          _signer = ECDSA.recover(_digest, _sig.v, _sig.r, _sig.s);
 74          require(_lastSigner < _signer, "BOsGovernanceProposal: invalid signer
order");
 75          _lastSigner = _signer;
 76        }
 77
 78        uint256 _weight = _getBridgeVoterWeight(_signer);
 79        if (_weight > 0) {
 80          _hasValidVotes = true;
 81          _lastVotedBlock[_signer] = block.number;
 82          _info.signatureOf[_signer] = _signatures[_i];
 83          _info.voters.push(_signer);
 84          if (_castVote(_v, _signer, _weight, _minimumVoteWeight, _hash) ==
VoteStatus.Approved) {
 85            return;
```

If the vote of a governor causes the proposal to pass, then the function returns early, causing later iterations of the loop to not occur. The loop updates the `_lastVotedBlock` of the voter, which is important as it is used when deciding whether or not to slash a governor for not voting enough.

```
  33    function slashBridgeVoting(address _consensusAddr) external {
  34      IRoninTrustedOrganization.TrustedOrganization memory _org =
_roninTrustedOrganizationContract
  35        .getTrustedOrganization(_consensusAddr);
  36      uint256 _lastVotedBlock =
Math.max(_roninGovernanceAdminContract.lastVotedBlock(_org.bridgeVoter),
_org.addedBlock);
  37      uint256 _period = _validatorContract.currentPeriod();
  38      if (block.number - _lastVotedBlock > _bridgeVotingThreshold &&
!_bridgeVotingSlashed[_consensusAddr][_period]) {
  39        _bridgeVotingSlashed[_consensusAddr][_period] = true;
  40        emit Slashed(_consensusAddr, SlashType.BRIDGE_VOTING, _period);
  41        _validatorContract.execSlash(_consensusAddr, 0,
_bridgeVotingSlashAmount);
```

As the `_signatures` array is sorted, it is possible that the `_lastVotedBlock` value for governors with lexicographically later addresses to not have their `_lastVotedBlock` value be updated. Such governors would need to vote on the same proposal again.

There are also no events emitted on which governor's vote counted, so governors would need to check if their `_lastVotedBlock` value was updated.

Recommendation

Recommend not returning early and have all votes be processed.

Alleviation

[**Ronin Team**, 03/03/2023]: The team acknowledged this issue and fixed it in commit 05a7fb8e7f4b0f7ef4afc0779cfa18eec5ba6329 by processing all votes.

DSU-01 POSSIBLE FOR A POOL ADMIN TO DELEGATE TO A DIFFERENT POOL

Category	Severity	Location	Status
Inconsistency	Minor	contracts/ronin/staking/DelegatorStaking.sol: 78	Resolved

Description

Normally, a pool admin cannot delegate to any other pools due to the following check in `delegate()`:

```
18    function delegate(address _consensusAddr) external payable noEmptyValue
poolExists(_consensusAddr) {
19      require(!isActivePoolAdmin(msg.sender), "DelegatorStaking: admin of an
active pool cannot delegate");
```

However, this check is not in `delegateRewards()` or `_delegateRewards()`, so a pool admin can have their rewards delegated to pool that they are not the admin of.

```
  function delegateRewards(address[] calldata _consensusAddrList, address
_consensusAddrDst)
    external
    override
    nonReentrant
    poolIsActive(_consensusAddrDst)
    returns (uint256 _amount)
  {
    return _delegateRewards(msg.sender, _consensusAddrList, _consensusAddrDst);
  }
```

```
  function _delegateRewards(
    ...
  ) internal returns (uint256 _amount) {
    _amount = _claimRewards(_user, _poolAddrList);
    _delegate(_stakingPool[_poolAddrDst], _user, _amount);
  }
```

The only check is in `_delegate()`, which requires the caller to not be the pool admin of the destination pool.

```
  function _delegate(
    ...
  ) internal notPoolAdmin(_pool, _delegator) {
```

Recommendation

Recommend including a check in `delegateRewards()` similar to the one in `delegate()` if the project intends to prevent every pool admin from delegating to any pool.

Alleviation

[**Ronin Team**, 03/03/2023]: The team acknowledged this issue and fixed it in commit f291728854130c0413de0ececa6765518695a71b by adding the required check when delegating rewards.

PAC-01 | POTENTIAL OUT-DATED OPENZEPPELIN LIBRARY USAGE

Category	Severity	Location	Status
Language Specific	Minor	package.json: 25	Resolved

Description

In the Ronin Network, the signatures are verified by the `recover()` function from OpenZeppelin's `ECDSA` module. For example,

```
    for (uint256 _i; _i < _signatures.length; _i++) {
      _sig = _signatures[_i];

      if (_supports[_i] == Ballot.VoteType.For) {
        _signer = ECDSA.recover(_forDigest, _sig.v, _sig.r, _sig.s);
      } else if (_supports[_i] == Ballot.VoteType.Against) {
        _signer = ECDSA.recover(_againstDigest, _sig.v, _sig.r, _sig.s);
      } else {
        revert("GovernanceProposal: query for unsupported vote type");
      }
```

However, according to the `package.json`, the version of OpenZeppelin is `^4.6.0`. For the OpenZeppelin version prior to 4.7.3, there is a vulnerability in signature malleability due to accepting EIP-2098 compact signatures in addition to the traditional 65-byte signature format.

Reference: https://github.com/advisories/GHSA-4h98-2769-gh6h

Recommendation

Recommend using latest stable version of the OpenZeppelin library during deployment to avoid the risk of potential vulnerabilities in an outdated version.

Alleviation

[**Ronin Team**, 03/03/2023]: The team acknowledged this issue and fixed it in commit 450241f8e4fa2be33c9f14ca6dca57f12af0e15a by using an updated library.

ROR-01 LACK OF CHECK WHEN UPDATING TRUSTED ORGANIZATION

Category	Severity	Location	Status
Inconsistency	Minor	contracts/multi-chains/RoninTrustedOrganization.sol: 332	Resolved

Description

When adding a trusted organization, there is a check to ensure that the consensus, governor, and bridge voter addresses are all different from each other.

```
268     address[] memory _addresses = new address[](3);
269     _addresses[0] = _v.consensusAddr;
270     _addresses[1] = _v.governor;
271     _addresses[2] = _v.bridgeVoter;
272     require(!AddressArrayUtils.hasDuplicate(_addresses),
"RoninTrustedOrganization: three addresses must be distinct");
```

However, this check is missing when updating a trusted organization, allowing the possibility of the governor or bridge voter address being equal to the consensus address.

Recommendation

Recommend adding a check when updating a trusted organization to ensure that the consensus, governor, and bridge voter addresses are all distinct from each other.

Alleviation

[**Ronin Team**, 03/03/2023]: The team acknowledged this issue and fixed it in commit 1d099af84e1bc0356e029508f2b9af570171939d by adding the required check when updating a trusted organization.

CEH-01 | MODIFIER `oncePerEpoch` INVALID ON FIRST EPOCH

Category	Severity	Location	Status
Volatile Code	Informational	contracts/ronin/validator/CoinbaseExecution.sol (01/18/2023-d722d7b): 42	Acknowledged

Description

The modifier `oncePerEpoch` ensures that a function can only be called once in an epoch.

```
42    modifier oncePerEpoch() {
43      if (epochOf(_lastUpdatedBlock) >= epochOf(block.number)) revert
ErrAlreadyWrappedEpoch();
44      _lastUpdatedBlock = block.number;
45      _;
46    }
```

However, in the scenario when `block.number < _numberOfBlocksInEpoch`, `epochOf(_lastUpdatedBlock) == epochOf(block.number)` will always be true as both of them are one.

```
39    function epochOf(uint256 _block) public view virtual override returns
(uint256) {
40      return _block / _numberOfBlocksInEpoch + 1;
41    }
```

This means that any function with the `oncePerEpoch` modifier, such as `wrapUpEpoch`, cannot be called in the first epoch.

Recommendation

Recommend adding logic to handle the case when `block.number < _numberOfBlocksInEpoch`.

Alleviation

[**Ronin Team**, 02/27/2023]: The team acknowledged the finding and decided not to change the current codebase. The `block.number` will be a large number (about ~11M) after the DPoS hardfork, so this check is unnecessary.

CON-01 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Volatile Code	Informational	contracts/mainchain/MainchainGatewayV2.sol; contracts/ronin/RoninGatewayV2.sol	Acknowledged

Description

The Ronin Bridge protocol may face potential compatibility issues with non-standard ERC20 tokens, such as deflationary tokens, as the exact amount of tokens locked in the bridge may not be precisely tracked.

For example, when bridging deflationary tokens with the ERC20 interface, the transaction fee can result in an unequal input and received amount. In a scenario where a user deposits 100 deflationary tokens (with a 10% transaction fee) to the mainchain bridge, only 90 tokens may arrive in the contract. However, on the Ronin chain, the user may still receive 100 wrapped tokens. If the user then bridges back the 100 wrapped deflationary tokens to the mainchain, they can still withdraw 100 tokens, causing the contract to lose 10 tokens.

Recommendation

Recommend regulating the tokens supported for the Ronin bridge and adding necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[**Ronin Team**, 02/27/2023]: The team acknowledged the finding and decided not to change the current codebase.

CSI-01 | POTENTIAL DOS ATTACK ON CANDIDATE APPLICATION

Category	Severity	Location	Status
Logical Issue	Informational	contracts/ronin/staking/CandidateStaking.sol: 36	Acknowledged

Description

The function `applyValidatorCandidate()` allows any individual to apply as a candidate, but the number of candidates is subject to an upper limit specified in the `maxValidatorCandidate()` function. If the number of candidates reaches this limit, any subsequent `applyValidatorCandidate()` invocations will fail due to a revert function in the CandidateManager contract (line 75).

```
  74      uint256 _length = _candidates.length;
  75      if (_length >= maxValidatorCandidate()) revert
ErrExceedsMaxNumberOfCandidate();
```

This creates an opportunity for an attacker to fill the candidate list with multiple addresses, effectively denying other addresses the ability to apply to become a candidate. This constitutes a Denial-of-service (DoS) attack.

Given that the attacker must stake no less than _minValidatorStakingAmount in order to become a candidate, a low setting of _minValidatorStakingAmount can cause a DoS attack.

Recommendation

Recommend including a check to ensure that `_minValidatorStakingAmount` is a high enough value.

Alleviation

[**Ronin Team**, 02/27/2023]: The team acknowledged the finding and decided not to change the current codebase. The maximum number of validator candidates will be 100 and the minimum staking amount will be 250,000 RON.

ROG-01 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Volatile Code	Informational	contracts/ronin/RoninGatewayV2.sol: 195, 204, 205, 278, 283, 284, 389, 390	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Although some of the "external" calls may be implemented in the provided smart contracts, considering possible component upgrades, it is still recommended to exclude the risks by adding proper protections instead of relying on dependencies.

`RoninGatewayV2` :

External call(s)

```
195
_bridgeTrackingContract.recordVote(IBridgeTracking.VoteKind.MainchainWithdrawal, _withdrawalId, _governor);
```

```
204
_bridgeTrackingContract.handleVoteApproved(IBridgeTracking.VoteKind.MainchainWithdrawal, _withdrawalId);
```

State variables written after the call(s)

```
205            _proposal.status = VoteStatus.Executed;
```

If reentrancy occurs in the call to `handleVoteApproved` such that the original caller and the reentrancy caller are different governors, the status of the proposal will still be `Approved`, meaning that the following block of code will again be executed:

```
203          if (_status == VoteStatus.Approved) {
204
_bridgeTrackingContract.handleVoteApproved(IBridgeTracking.VoteKind.MainchainWithdrawal, _withdrawalId);
205            _proposal.status = VoteStatus.Executed;
206            emit MainchainWithdrew(_hash, _withdrawal);
```

In particular `handleVoteAppproved` will be called again, possibly inflating the number of votes during a period, and the event `MainchainWithdrew` will be emitted again, which may be important to how the bridge operates.

External call(s)

```
278         _bridgeTrackingContract.recordVote(IBridgeTracking.VoteKind.Withdrawal,
_id, _validator);
```

```
283
_bridgeTrackingContract.handleVoteApproved(IBridgeTracking.VoteKind.Withdrawal,
_id);
```

State variables written after the call(s)

```
284           _proposal.status = VoteStatus.Executed;
```

If reentrancy occurs in the call to `handleVoteApproved` such that the original caller and the reentrancy caller are different governors, the status of the proposal will still be `Approved`, meaning that the following block of code will again be executed:

```
282         if (_status == VoteStatus.Approved) {
283
_bridgeTrackingContract.handleVoteApproved(IBridgeTracking.VoteKind.Withdrawal,
_id);
284           _proposal.status = VoteStatus.Executed;
```

In particular `handleVoteAppproved` will be called again, possibly inflating the number of votes during a period.

External call(s)

```
389
_bridgeTrackingContract.handleVoteApproved(IBridgeTracking.VoteKind.Deposit,
_receipt.id);
```

State variables written after the call(s)

```
390         _proposal.status = VoteStatus.Executed;
```

If reentrancy occurs in the call to `handleVoteApproved` such that the original caller and the reentrancy caller are different governors, the status of the proposal will still be `Approved`, meaning that the following block of code will again be executed:

```
388     if (_status == VoteStatus.Approved) {
389
_bridgeTrackingContract.handleVoteApproved(IBridgeTracking.VoteKind.Deposit,
_receipt.id);
390       _proposal.status = VoteStatus.Executed;
391       _receipt.info.handleAssetTransfer(payable(_receipt.ronin.addr),
_receipt.ronin.tokenAddr, IWETH(address(0)));
392       emit Deposited(_receiptHash, _receipt);
```

In particular `handleVoteAppproved` will be called again, possibly inflating the number of votes during a period, tokens are again transferred or minted to the recipient, and the `Deposited` event is again emitted.

Recommendation

Recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[**Ronin Team**, 03/03/2023]: The team acknowledged this issue and fixed it in commit 7cbae4761f05c24f2fbd7f2acbd8dfcac0d591d0.

ROO-01 | PURPOSE OF VOTING FOR BRIDGE OPERATORS

Category	Severity	Location	Status
Inconsistency	Informational	contracts/ronin/RoninGovernanceAdmin.sol: 296	Resolved

Description

There is currently no explicit relationship between bridge operator governance proposals and the actual bridge operators.

Bridge operators are required to ensure the gateways work properly by signing deposit and withdrawal receipts. On the Ronin chain, these bridge operators are determined by querying the validator contract.

```
403    function _getValidatorWeight(address _addr) internal view returns (uint256
_weight) {
404      _weight = _validatorContract.isBridgeOperator(_addr) ? 1 : 0;
405      require(_weight > 0, "RoninGatewayV2: unauthorized sender");
406    }
```

The Ronin validator contract lets a validator be a bridge operator as long as it has not requested an emergency exit.

```
445        bool _isBridgeOperatorAfter = !_emergencyExitRequested;
446        if (!_isBridgeOperatorBefore && _isBridgeOperatorAfter) {
447          _validatorMap[_validator] =
_validatorMap[_validator].addFlag(EnumFlags.ValidatorFlag.BridgeOperator);
```

On the other hand, the admin of the main chain decides who the bridge operators are.

```
 99    function replaceBridgeOperators(address[] calldata _list) external onlyAdmin
{
100      address _addr;
101      for (uint256 _i = 0; _i < _list.length; _i++) {
102        _addr = _list[_i];
103        if (_bridgeOperatorAddedBlock[_addr] == 0) {
104          _bridgeOperators.push(_addr);
105        }
106        _bridgeOperatorAddedBlock[_addr] = block.number;
107      }
108
109      {
110        uint256 _i;
111        while (_i < _bridgeOperators.length) {
112          _addr = _bridgeOperators[_i];
113          if (_bridgeOperatorAddedBlock[_addr] < block.number) {
114            delete _bridgeOperatorAddedBlock[_addr];
115            _bridgeOperators[_i] = _bridgeOperators[_bridgeOperators.length - 1];
116            _bridgeOperators.pop();
117            continue;
118          }
119          _i++;
120        }
121      }
122
123      emit BridgeOperatorsReplaced(_list);
124    }
```

In either case, the bridge operators do not look at the results of governors voting for bridge operators. When a bridge operator governance proposal is passed, only the `_lastSyncedBridgeOperatorSetInfo` variable is updated, but this variable has no influence on the bridge operators used by the gateways.

Recommendation

Recommend enforcing that the addresses in `_lastSyncedBridgeOperatorSetInfo` are the actual bridge operators for the gateways.

Alleviation

[**Ronin Team**, 02/17/2023]: The list of bridge operators is determined on Ronin chain by querying the validator contract.

This list is then relayed into the mainchain via a proposal. Note that the `admin` of the `MainchainGatewayV2` contract is the `GovernanceAdmin`. The call to update the bridge operator list only gets executed if there is a valid proposal.

When a new list of bridge operators is updated in `RoninGateway`, the _lastSyncedBridgeOperatorSetInfo is updated, along with the `BridgeOperatorsApproved` event being emitted. This event will trigger a worker that receives the new operator list

from `RoninGateway` and creates a new proposal to update the list on `MainchainGateway`.

Also note that the `_lastSyncedBridgeOperatorSetInfo` also helps the `MainchainGateway` keep up-to-date with the list on Ronin, preventing duplication updates and outdated updates.

SLD-01 | IMPLEMENTATION OF DOUBLE SIGN SLASHING

Category	Severity	Location	Status
Logical Issue	Informational	contracts/ronin/slash-indicator/SlashDoubleSign.sol: 24	Resolved

Description

There are some concerns regarding how the function `slashDoubleSign()` works:

1. The only check for provided evidence is `_pcValidateEvidence()` and it is unclear if in the next block, the provided evidence can be used again, possibly allowing repeated slashing of a validator.
2. Since validation of the evidence `_header1` and `_header2` does not include `_consensusAddr` as an input, there is a concern that the evidence can be used against any validator.
3. Regarding the jailing time, a validator slashed for double signing is jailed until the block `_doubleSigningJailUntilBlock` instead of something akin to `block.number + duration`. There is a concern that validators who conduct double signing after `_doubleSigningJailUntilBlock` will not be jailed.

Recommendation

Recommend changing the design of the function `slashDoubleSign()` if any of the above issues are valid.

Alleviation

[**Ronin Team**, 02/08/2023]:

1. It is not intended to be able to submit the same evidence twice. However, if a validator gets double-sign slashed, it will not be selected as a validator again so this does not need to be enforced.
2. The `_header1`, and `_header2` contains `_consensusAddr` so it will not be a problem.
3. `_doubleSigningJailUntilBlock` is set to be very big, so the block number cannot reach this point.

SUU-01 | LACK OF CHECK WHEN SLASHING FOR UNAVAILABILITY

Category	Severity	Location	Status
Logical Issue	Informational	contracts/ronin/slash-indicator/SlashUnavailability.sol: 45	Resolved

Description

There are no established criteria for slashing a validator for unavailability. In contrast, when slashing for double signing, the caller must provide evidence, and when slashing for bridge voting, a check is made to determine if the governor's voting activity meets the requirements.

However, with regards to unavailability, there are no provisions for incrementing the validator's `_unavailabilityIndicator` value.

Recommendation

Recommend including checks so that any slashes for unavailability are valid.

Alleviation

[**Ronin Team**, 02/08/2023]: The _unavailabilityIndicator can only be increased when the validator misses a block. We do have verification for this (but not in the smart contract).

APPENDIX | RONIN DPOS CONTRACTS

Finding Categories

Categories	Description
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

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

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

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.

CertiK | Securing the Web3 World

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.