Security Audit Report

Ripple Q1 2025: XRPL EVM Sidechain

Last revised 25.03.2025

Authors: Mirel Dalcekovic, Andrija Mitrovic

Contents

Audit overview

The Project

In February 2025, Ripple Labs engaged Informal Systems to conduct a security audit of the XRPL EVM Sidechain.

Scope of this report

The audit focused on evaluating the correctness and security properties of the changes introduced in the XRPL EVM node and the Evmos fork.

For the `node` implementation, the review covered the introduced diffs, with particular attention to their impact on validator management - specifically, the addition and removal of validators through governance proposals and the implementation of the PoA consensus, along with IBC v8 integration and upgrade handler implementations due to the Cosmos SDK v0.50 upgrade.

In the Evmos fork, the audit examined the extensions made to the ERC20 precompile contract and the `x/erc20` middleware, specifically the addition of mint, burn and ownership transfer functionalities. A key focus was ensuring that these modifications did not introduce unintended vulnerabilities in token pair management, whether executed via smart contract calls or Cosmos SDK messages.

Audit plan

The audit was conducted between February 27th, 2025 and March 14th, 2025 by the following personnel:

- Mirel Dalcekovic
- Andrija Mitrovic

Conclusions

The changes introduced in the node diff lack clarity. This stems from the earlier PoA consensus implementation, which retained its old code in the codebase. While this does not affect correctness, it creates confusion, as the communicated intended behavior differs from the existing code in the `x/poa` module. Additionally, the integration of the `x/crisis` module requires further review, and the team should decide on its intended impact - specifically, whether and under what cadence the chain should halt if the invariants introduced in the diff are violated.

All the findings are listed in the Findings section.

During the course of the audit, the development team independently discovered a critical issue that prevented the bank smart contract from minting ERC20 tokens as intended. The root cause was promptly identified and resolved by the development team through a PR.

Additionally, we recommend following best practices and guidance related to publicly disclosed security advisories that impact dependencies used in the XRPL EVM Sidechain node codebase. Addressing these advisories before deploying to mainnet is crucial for maintaining the overall security posture of the project:

- ibc-go related: ISA-2025-001; ASA-2025-004.
- CometBFT related: ASA-2025-002; ASA-2025-001.

Audit Dashboard

Target Summary

- **Type: Protocol and Implementation**
- **Platform: Go & Solidity**
- **Artifacts:**
  - xrlpevm node diff (later tag v6.0.0)
  - xrplevm evmos fork diff (later tag v20.0.0-exrp.4)

Engagement Summary

- **Dates**: February 27th, 2025 - March 14th, 2025
- **Method**: Manual code review, protocol analysis

Severity Summary

Finding Severity	Number
Critical	1
High	0
Medium	0
Low	2
Informational	5
Total	8

System Overview

The XRPL EVM Sidechain is an extension of the XRP Ledger (XRPL) that introduces Ethereum Virtual Machine (EVM) compatibility, enabling smart contract functionality within the XRPL ecosystem. Built on the Cosmos SDK, this sidechain employs a Proof-of-Authority (PoA) consensus model. The integration of Ethereum's smart contract capabilities with XRPL enhances its utility for decentralized applications (dApps) and cross-chain asset transfers.

Consensus Mechanism: Proof of Authority (PoA)

The XRPL EVM Sidechain employs a Proof of Authority (PoA) consensus model, where validators are pre-approved through a governance process rather than relying on economic incentives such as Proof of Work (PoW) or Proof of Stake (PoS). **Main goal with building the XRPL EVM Sidechain was Preserving the XRP Ledger's Security & Trust Philosophy:** where consensus participats define their own Unique Node List (UNL) and as a consequence the XRPL network minimizes the risk of a single point of failure or centralized control. PoA approach aligns with the core principles of the XRP Ledger, ensuring security, trust, and decentralization.

PoA Consensus is Built on CometBFT's PoS: The XRPL EVM sidechain adapts CometBFT's PoS consensus by introducing key restrictions to enforce PoA:

- Validators are added and removed exclusively through the governance process.
- A special `BondDenom` token is used solely for staking:
  - When a validator is created, a predefined amount is minted as stake.
  - When a validator is removed, the staked amount is burned.
- The system does not contain any unstaked (free) `BondDenom` tokens.
- Re-delegations and un-delegations are not permitted.
- Each validator has only a single self-delegation (created at the moment of adding the validator, when the staking amount is minted).
- Slashing penalties do not apply, meaning staking amounts remain unchanged.
- No `BondDenom` token rewards are distributed (no inflationary rewards), and since there are no delegations, rewards distribution and commission rate settings are irrelevant.

Interoperability and Asset Transfers

The XRPL EVM Sidechain is connected to the XRPL via the Axelar Network, utilizing Axelar's decentralized validator set and message-passing protocols. This integration enables seamless XRP transfers between XRPL's native ledger and its EVM-compatible sidechain.

- **IBC and ERC-20 Compatibility:** The sidechain leverages the Evmos framework to facilitate ERC-20 token transfers using Inter-Blockchain Communication (IBC). The `x/ibc/transfer` application converts ERC-20 tokens into a Cosmos-compatible format for IBC transactions, while the `x/erc20` middleware ensures bidirectional conversion between ERC-20 and Cosmos tokens.
- **Governance and Token Management:** Token pairs (ERC-20 <> Cosmos) must be pre-registered via governance using `MsgRegisterERC20`. Ownership transfers of token pairs can be executed through governance with `MsgTransferOwnership` or by the current owners using EVM precompiles, allowing designated accounts to mint and manage token supplies.
- **Burn, Mint, and Ownership Functions:** The Evmos fork extends the ERC-20 precompile contract and `x/erc20` middleware to support burnable, mintable, and transferable ownership functionalities, ensuring efficient asset movement across the Axelar bridge.

Threat Model

In our threat analysis, we start by defining a set of properties required for the correctness of the audited artifacts. For each property, we define one or more threats. We then analyzed them individually to see if they could be violated, resulting in the findings presented in the Findings section.

Node diff

Property: Validators can only be instantiated and activated via governance process

Violation consequences: Validators can be created not only through an authority (governance) thus violating the PoA protocol.

Threats:

- Non-authorized (someone other than governance) entity can call `AddValidator` (submit a `MsgAddValidator`) and add a validator.
- Validator can be created through a standard `MsgCreateValidator` message.

Conclusion:

The property holds.

- The check ensures that the `MsgAddValidator` msg's authority matches the PoA module keeper's authority. The PoA module keeper is instantiated here, where it is explicitly defined that its authority parameter is set to governance. Enforcement that the `MsgAddValidator` authority parameter can't be tampered with is done by setting this parameter to be filled with the msg signer.
- Creation of validators through a standard `MsgCreateValidator` is not possible due to the fact that a validator cannot acquire the tokens the bondable denomination necessary for staking in any other way than through `MsgAddValidator` msg.

Property: Only validators can be removed from the validator set through the governance process

Violation consequences: Validators can be removed not only through an authority (governance) thus violating the PoA protocol.

Threats:

- Non-authorized (someone other than governance) entity can call `RemoveValidator` (submit a `MsgRemoveValidator`) and remove a validator.
- Validator can leave or be removed from the current validator set by:
  - Unbonding: A validator can voluntarily remove themselves (A validator can choose to **stop** validating by submitting a `MsgUnbond` transaction).
  - Low stake: If their total stake falls below the top N validators, they are automatically removed.

Conclusion:

The property holds.

- The check ensures that the `MsgRemoveValidator` msg's authority matches the PoA module keeper's authority. The PoA module keeper is instantiated here, where it is explicitly defined that its authority parameter is set to governance. Enforcement that the `MsgRemoveValidator` authority parameter can't be tampered with is done by setting this parameter to be filled with the msg signer.
- Due to the fact that the undelegate and redelegate msgs are disabled there is no possibility for a validator to leave the validator set by removing staked coins thus falling outside the needed stake range.

Property: The total supply of `BondDenom` corresponds to `DefaultPowerReduction * len(active validators)` where each active validator has `DefaultPowerReduction` staked `BondDenom`

Violation consequences: Violating this property would lead to existence of free `BondDenom` tokens in circulation that can be used to bypass the authorized validator creation.

Threats:

- On removing a validator not all tokens are burned.
- `BondDenom` tokens can only be minted apart from validator creation.
- Transfer of `BondDenom` tokens from a validator to another arbitrary address.

Conclusion:

The property holds.

- No remnants of tokens are left after a validator is removed. All the tokens are burned no matter if these are:
  - On a balance in the bank.
  - Bonded.
  - Unbonded/Unbonding.
- `BondDenom` tokens can be only minted through validator creation by governance and always in the same amount (`DefaultPowerReduction`) here. Thus there is no possibility of minting tokens on another address apart from a validator.
- Transfer of `BondDenom` tokens from a validator to an arbitrary address is not possible because all the tokens are staked on validator creation and undelegation and redelegation are disabled.

Property: No redelegations and undelegations for validators are allowed

Violation consequences: Possibility of redelegations and undelegations can lead to an uneven distribution of power among the validators and at the same time free tokens that can be transferred to arbitrary addresses and then used to bypass the authorized validator creation.

Conclusion:

The property holds.

During validator creation in `ExecuteAddValidator`, the system verifies that the validator has no delegations to other validators or unbonding delegations. If either exists, the system returns an error and prevents validator creation. Additionally, undelegation and redelegation are prevented since these messages are disabled.

Property: If the authority (governance) initiates validator removal, validator cannot stop the removal

Violation consequences: Validator can evade removal from the validator set even though the authority has decided to do so.

Threat:

- Validator could jail himself in order to avoid being removed from the set.

Conclusion:

The property holds.

A validator cannot prevent their removal by the authority (governance) through self-jailing or any other action. This is because the authority removes validators through `ExecuteRemoveValidator` by burning all their `BondDenom` tokens, as explained above. Once these tokens are burned, the validator automatically falls out of the validator set through the staking module.

Property: Validator commission rates must be set to 0

Violation consequences: Basically, no impact if no delegations, but the validators should keep the initial PoA consensus settings. If delegations are somehow made, some validators would earn more rewards on account of their delegations.

Threat:

- Commission rate parameters can be changed to a value different than 0, leading to the misalignment in existing PoA validators settings.

Conclusion:

The property holds.

The only way that commission rate parameters can be changed is through the `MsgEditValidator` , which execution is allowed. However XRPL's Cosmos SDK fork was not altered and the vanilla implementation still stands: Since the `maxRate` and `maxChangeRate` are set to 0 initially, the validator is locked at 0% commission forever, They cannot earn staking rewards from their delegations. `UpdateValidatorCommission` can not be successful executed (code ref):

- `MaxRate` defines the absolute maximum a validator's commission can reach. If it's 0, the validator cannot set rate to anything other than 0 - this is checked within `ValidateNewRate` function (code ref)
- The validator can never change the commission rate because: `MaxChangeRate = 0` means that even if `MaxRate > 0`, the allowed daily increase is zero - this is checked within `ValidateNewRate` function (code ref).

Property: Validator's `minimumSelfDelegation` must be set to 1

Violation consequences: Changing the `minimumSelfDelegation` to a value that is lower than `DefaultPowerReduction` can lead to the fact that not enough tokens have been staked in order to have the same influence from all the validators.

Threat:

- `minimumSelfDelegation` value can be changed with editing validator parameters.

Conclusion:

The property holds.

After reviewing the XRPL Cosmos SDK fork, it was confirmed that the implementation remains consistent with the vanilla Cosmos SDK and that the `minimumSelfDelegation` can not be altered for the PoA validators. Specifically, the restrictions that prevent this are:

- `minSelfDelegation` from decreasing (code ref)
- `minSelfDelegation` from being increased above the validator's currently bonded stake remains intact in the XRPL fork (code ref)

```
if !msg.MinSelfDelegation.GT(validator.MinSelfDelegation) {
  return nil, types.ErrMinSelfDelegationDecreased
 }

if msg.MinSelfDelegation.GT(validator.Tokens) {
 return nil, types.ErrSelfDelegationBelowMinimum
}
```

This ensures that all validators maintain their expected voting power, preventing any validator from artificially reducing/increasing their self-delegation to manipulate stake-based governance or consensus participation.

Property: Validator edits must not allow modifications to default PoA-related validator parameters

Threats:

- A validator operator can modify `MinSelfDelegation`, setting it to an arbitrary value. This may result in an insufficiently bonded token amount relative to the expected `PowerReductionValue`, potentially affecting staking security guarantees.
- A validator operator can update commission rate parameters, which may introduce risks for existing delegations, particularly if excessive protocol changes occur.

Conclusion:

The property holds.

After reviewing the XRPL Cosmos SDK fork, as already shared with the previous property analysis conclusions for the properties: *Validator commission rates must be set to 0* and *Validator's `minimumSelfDelegation` must be set to 1* - it is impossible to alter the `CommissionRates` and `minimumSelfDelegation` values.

Invariant related

Staking Power Invariant: All validators have the same staking power as the default power reduction (`DefaultPowerReduction`)

Violation Consequences: Unequal staking power breaks the PoA guarantee, leading to validator centralization. This may allow a subset of validators to gain disproportionate influence over consensus and governance, potentially enabling censorship or malicious proposals.

Threats:

- Arbitrary user or validators could perform delegations of `BondDenom` tokens to other validators.
- Validator rewards are issued in `BondDenom` tokens and can be used to increase the validator power with self-delegation.

Conclusion:

The property holds.

After reviewing the code and the allowed messages, integrated modules we concluded:

- No undelegation and redelegation is allowed (code ref).
- Only self-delegations are allowed. All other delegations will be rejected with validation introduced in the `BeforeDelegationCreated` staking hook (code ref).
- There is no mechanism for a validator to acquire unbonded `BondDenom` tokens to perform self-delegation and increase their power. Our analysis confirms that no additional `apoa` tokens can be created outside of `MsgAddValidator`—there is no **mint** module present, and consequently, no inflation (which aligns with the expectation for a PoA-based chain). Furthermore, all fees are paid exclusively in the designated **token denom** (verified through localnet testing).

Self-Delegation Invariant: Each validator has exactly one self-delegation, and it matches their validator address

Violation consequences: Possibility of validators having different staking powers could lead to consequences listed in the property `StakingPowerInvariant`.

Threats:

- Arbitrary user or validators could perform delegations of `BondDenom` tokens to other validators.

- Validators and arbitrary users could perform undelegations and redelegations leading to fluctuations in validator power.

Conclusion:

The property holds.

Analysis performed within the property Staking Power invariant and threats analysis here, concluded that there is no unstaked `BondDenom` tokens, undelegations and re-delegations are not possible and only self-delegations records could be present in the state.

In addition, the invariant says "exactly one" self-delegation. If there is an existing record for the delegator-validator pair, this delegation record will be updated to increase their delegation shares (code ref1, ref2). However, the staking amount is minted when the validator is created and that is the only possible place of self-delegating `BondDenom` tokens to the validator - so the self-delegation record will be created at this point and this delegation record is the only one present.

Keeper Dependencies Parameters Invariant: Critical slashing parameters are initialized to expected values

Violation consequences: Possibility of validators having different staking powers, as result of slashing power reductions, could lead to consequences listed in the property `StakingPowerInvariant`.

Threats:

- Slashing parameters `SlashFractionDoubleSign` and `SlashFractionDowntime` could be updated and introduce power reduction after validator slashing.

Conclusion:

The property does not hold.

Slashing module contains the following parameters: `SignedBlocksWindow`, `MinSignedPerWindow`, `DowntimeJailDuration`, `SlashFractionDoubleSign` and `SlashFractionDowntime`. The slashing fraction parameter values are set to 0 (as we can see with the `localnet.sh` code ref1, ref2.) In POA it is expected that there are no slashing reductions that impact the validator power.

Slashing module parameters can be updated through governance process (XRPL Cosmos SDK fork slashing code ref1, node app wiring ref2, ref3) meaning that existing validators should approve this change with voting for the proposal. It is highly unlikely that this kind of proposal will be accepted, if it is placed.

These parameters can also be set from genesis, so it is possible that `SlashFractionDoubleSign` and `SlashFractionDowntime` have non zero values assigned at the chain start.

The invariant will not be triggered for assertion automatically from the `EndBlocker` since `invCheckPeriod` = 0 (code ref) and if called for assertion externally, it will only log that slashing parameters are not 0 (code ref).

Property: The state after the upgrade must be a valid continuation of the pre-upgrade state, ensuring all protocol invariants remain intact

Violation consequences: If not done accordingly, state migration during the upgrade from Cosmos SDK v0.47 to v0.50, can lead to breaking of the invariants.

Threats:

- Upgrade handlers contain coding issues.
- Proto definitions for Msgs do not have signer fields defined (`GetSigners` is deprecated)
- `ValidateBasic` confusion due to upgrading from 0.47 to 0.50

Conclusion:

The property holds.

Nonetheless, our analysis identified several minor observations, which are outlined in an informational issue.

The analysis focused on code changes related to the Evmos upgrade, as well as the Cosmos SDK upgrade to version 0.50. Auditors reviewed all the steps outlined in the v0.50.x upgrade guide.

General Properties and Threats

The following properties were systematically analyzed throughout the audit to assess the overall security, stability, and correctness of the system. The analysis confirms that the evaluated properties hold as expected, with no critical issues identified. Minor recommendations for code quality improvements and documentation clarity have been noted.

GT #1: Development & Testing Practices and Quality of Code

Conclusion: The code is well-structured, with only minor suggestions for improvement, which are outlined in the findings.

GT #2: Arithmetic Operations

Conclusion: This property holds.

GT #3: Chain Halts Due to Panics in the Begin and End Blocker

Conclusion: This property holds; no panics were identified that could cause a chain halt.

GT #4: DoS Attacks Due to Unbounded Iteration

Conclusion: This property holds.

GT #5: Non-Determinism Sources

Conclusion: This property holds.

GT #6: Protobuf Definition Implementation Issues

Conclusion: This property holds.

GT #7: Documentation Discrepancies

Conclusion: Some system properties should be more clearly outlined in the guideline documentation.

---

evmos fork related properties

Property: Token has only one owner (smart contract address)

Threats: Token can have multiple owners.

Conclusion:

The property holds.

This is enabled by design. Each token can have only one owner. When a token is created, its ownership address is initially left empty. Later, governance can assign ownership of this token to a desired contract address (particularly in cases involving IBC-transferred destination base tokens). From this point forward, either the owner or governance can transfer ownership to another address. Each ownership transfer moves control from one address to another, maintaining the principle that there can only be one token owner at any given time.

Property: Token owner is the only address allowed to mint tokens

Threats:

- It is possible for any address to mint tokens directly on Cosmos SDK layer.
- It is possible for ANY address to mint tokens on evm precompile layer.

Conclusion:

The property holds.

The MintCoins function verifies that the caller (msg sender) is the token owner. Additionally, the `MsgMint` message cannot be forged because the sender must be the signer of the message and must match the token owner in MintCoins, as enforced in the message definitions here and here.

Property: Token ownership can be set only to Native denoms

Threats: Owner for non native denoms can be changed.

Conclusion:

The property holds.

This is enforced by the `transferOwnership` function, which handles ownership transfers whether initiated through governance (TransferContractOwnership) or by the token owner (TransferOwnership). The function checks if the coin is native. A native coin (labeled with OWNER_MODULE) is a source chain base coin transferred over IBC to this chain.

Property: Token (Smart contract) ownership can be transferred through governance proposal (Cosmos SDK layer) or by the current owner (evm precompile)

Threats:

- Unauthorized (someone other than governance) entity can transfer ownership via the Cosmos SDK layer.
- Unauthorized (someone other than current owner) entity can transfer ownership via EVM precompiles.

Conclusion:

The property holds.

The `TransferOwnership` precompile function is possible to executed only if the sender is the token pair's `OwnerAddress` (code ref). Similarly, when the underlying Cosmos SDK erc20 `TransferOwnership` function is triggered, the same ownership checks are performed by retrieving the owner from the store (code ref). If all checks pass, the token pair `OwnerAddress` will be updated to the new owner (code ref). Ownership transfer for a token pair through the Cosmos SDK layer (erc20 module) is governed by the `TransferOwnershipProposal` function, processing the `MsgTransferOwnership`. This function can only be executed by an authority (code ref) specified during the chain's initialization (code ref), ensuring that only authorized entities can initiate the transfer.

Property: Revoke ownership is possible

Threats:

- Ownership can not be transferred to an non active (burned) address.
- Accidental ownership revocation is possible.

Conclusion:

The property holds. It is possible to revoke ownership by transferring it to a burned address with evm precompile `TransferOwnership` function or with governance proposal, making it possible for governance to make a contract owner-less, if needed. However, improvement is possible, since it is possible for current owner to accidentally revoke the ownership by setting zero as new owner.

Property: Transfer the ownership, minting and burning of ERC20 tokens can be executed through evm precompiles and through Cosmos SDK layer directly

Threats:

- Inability to execute minting, burning, or transferring ownership via:
  - EVM precompiles
  - Cosmos SDK layer

Conclusion:

The property holds.

Minting is possible to be executed from evm precompiles with calling the `Mint` function (code ref) and from Cosmos SDK layer with calling the `MintCoins` function (code ref) via `MsgMint`.

Burning is possible to be executed from the evm precompiles with calling the `Burn`, `Burn0` and `BurnFrom` functions (code ref) and from Cosmos SDK layer with calling the `BurnCoins` function (code ref) via `MsgBurn`.

Transferring of the token pair ownership is possible to be executed from evm precompiles with calling the `TransferOwnership` function (code ref) and from Cosmos SDK layer with calling the `TransferOwnershipProposal` function (code ref) via `MsgTransferOwnership`.

Property: Evm precompiles burn execution and `BurnCoins` on Cosmos SDK layer execution should behave the same

Conclusion:

The property holds.

Coin burning through EVM precompiles is handled by the burn function, while on the Cosmos SDK layer it is processed through the MsgServer by executing the Burn function for `MsgBurn` messages. Both methods utilize the `BurnCoins` function without significant parameter preparation, ensuring identical behavior.

Property: `Burn0` (evm precomplile) allows token owner to burn tokens from a spenders account

Threats: It is possible for any address to burn tokens with `Burn0` (evm precomplile).

Conclusion:

The property holds.

The `Burn0` function verifies that the spender is the token owner. Additionally, the `Burn0` parameters cannot be forged because the sender must be the signer of the transaction that initiates the burning process. The sender is derived from the transaction signer in the evmos go ethereum implementation.

Property: When spender and sender are the same accounts, `BurnFrom` will perform transfer without any authorization

Threats: An owner account is required authorization to burn coins.

Conclusion:

The property holds.

`BurnFrom` executes a burn of the caller's tokens by transferring the callers tokens to the zero address (call to transfer). In the case that the spender and the sender (token owner) are the same no authorization is necessary, tokens are sent to the zero address without any authorization here. As in the previous threat, the parameters to the `BurnFrom` function can not be forged in that way that the spender is the same as the token owner because the spender is the contract caller (here) and this argument is derived from the transaction signer that is enforced in the evmos go ethereum implementation.

Property: When spender and sender are NOT the same accounts, `BurnFrom` will perform an authorized transfer of the specific amount to zero address and burning

Threats: No authorization is needed if the spender and sender are NOT the same accounts on calling `BurnFrom`.

Conclusion:

The property holds.

`BurnFrom` executes a burn of the caller's tokens by transferring the callers tokens to the zero address (call to transfer). In the case where the sender and the spender are not the same account an authorization from the token owner should be given (check here). If the action is authorized by the token owner then the action is executed here.

Property: When spender and sender are NOT the same accounts, the authorized transfer amount cannot exceed the allowance

Threats: An authorized transfer can be executed for an amount that is greater than the allowed amount.

Conclusion:

The property holds.

This is enforced through authorization given by the token owner. The authorization includes an allowed amount to be transferred so called allowance that is returned when the authorization is checked here.

Findings

Finding	Type	Severity	Status
Invariant checks through crisis module may go unnoticed	Protocol	Critical	Resolved
Slashing parameters can be modified to unexpected values	Protocol	Medium	Resolved
Validator number validation	Implementation	Low	Resolved
Unnecessary token burning on validator removal	Implementation	Informational	Resolved
Messages MsgDelegate and MsgCancelUnbondingDelegation should be disabled	Protocol	Informational	Resolved
Separate revoke ownership from transfer ownership	Implementation	Informational	Acknowledged
Minor observations and improvements related to the Cosmos SDK v0.50 upgrade	Implementation	Informational	Resolved
Refactor mint.go by splitting responsibilities	Implementation	Informational	Resolved

Invariant checks through crisis module may go unnoticed

ID	IF-FINDING-001
Severity	Critical
Impact	3 - High
Exploitability	3 - High
Type	Protocol
Status	Resolved

Involved artifacts

- x/poa/keeper/invariants.go

Description

The `x/crisis` module is not advisable to use (see security advisory) because it does not actually halt the chain when an invariant check fails. While the module allows submitting a `MsgVerifyInvariant` transaction to confirm an invariant violation, this transaction does not cause nodes to panic. Instead, the SDK's panic-recovery mechanism treats it as an invalid transaction and allows the chain to continue processing blocks.

Additionally, only nodes running with `--inv-check-period X` will panic upon detecting an invariant violation in `EndBlock`, but this is rarely used in production due to the high resource cost of running invariant checks every `X` blocks and in the case of this project the check period is set to zero meaning that it will not get triggered automatically. As a result, invariant violations may go unnoticed, making the module ineffective in practice.

Problem scenarios

Invariant violations may go unnoticed.

Recommendation

Consider alternative approaches for checking invariants, such as monitoring with alerts. Additionally, carefully evaluate the desired behavior for each invariant violation. Some violations may warrant stopping the chain, while others may only require raising an alert.

Status

Resolved through:

- Pull request: https://github.com/xrplevm/node/pull/61

Slashing parameters can be modified to unexpected values

ID	IF-FINDING-002
Severity	Medium
Impact	3 - High
Exploitability	1 - Low
Type	Protocol
Status	Resolved

Involved artifacts

-

Description

In a Proof of Authority (PoA) system, validator power should remain unchanged after slashing events. If power reductions occur due to slashing, validators may end up with different staking powers, potentially violating the `StakingPowerInvariant`.

The Keeper Dependencies Parameters Invariant, which expects critical slashing parameters to be initialized to predefined values, does not hold. This is because the slashing parameters `SlashFractionDoubleSign` and `SlashFractionDowntime` can be modified through governance or set at genesis to unexpected values.

Problem scenarios

- Validators may lose power unexpectedly if slashing parameters are modified post-genesis.
- This could disrupt validator selection and consensus stability, especially in a PoA system where equal power is expected.

Recommendation

Implement restrictions on parameter changes through both governance and genesis configuration. This can be achieved by modifying the Cosmos SDK fork to prevent governance proposals from updating slashing factor parameters, and by adding genesis initialization checks that validate these values are non-zero before allowing chain startup.

Status

Resolved through:

- Pull requests
  - Add validation and change default values https://github.com/xrplevm/cosmos-sdk/pull/1
  - Slashing integration tests: https://github.com/xrplevm/node/pull/70

Validator number validation

ID	IF-FINDING-003
Severity	Low
Impact	2 - Medium
Exploitability	1 - Low
Type	Implementation
Status	Resolved

Involved artifacts

- /x/poa/keeper/keeper.go

Description

Taking into consideration that in Proof of Authority (PoA), all validators have the same voting power, and that the Cosmos SDK staking module determines which validators are included in the active set based on their voting power, an issue arises when the maximum number of validators has already been reached.

If a new validator is added under these conditions, since it has the same voting power as the existing ones, the selection process may be affected by the validator's address rather than its power. This means that a validator could be excluded from the active set not due to voting power differences, but simply due to address ordering.

To prevent this, a maximum validator count check should be enforced to block additional validators once the limit is reached—or at least until this limit is intentionally increased.

Problem scenarios

Validator admission to the active set can become dependent on its address.

Recommendation

There are two possible approaches:

- Introduce a minimum and a maximum number of validators check in methods to remove and add validator.
- Create a clear guideline documents for the governance that outlines the necessity to take care of the maximum number of validators and if the upper limit needs to be changed.

Status

Resolved through:

- Pull request: https://github.com/xrplevm/node/pull/62

Unnecessary token burning on validator removal

ID	IF-FINDING-004
Severity	Informational
Impact	0 - None
Exploitability	0 - None
Type	Implementation
Status	Resolved

Involved artifacts

- x/poa/keeper/keeper.go

Description

- The burning of free tokens tokens in `ExecuteRemoveValidator` is unnecessary. This is because `BondDenom` tokens can only be minted and burned during validator creation and removal by an authority, and these tokens are immediately staked and bonded. Additionally, since undelegate and redelegate messages have been disabled, there is no way for free tokens to exist.
- A related issue is that `ExecuteRemoveValidator` can be called for non-validator addresses, which is unnecessary since it is meant specifically for validator removal. As mentioned above, no `BondDenom` tokens can exist except for those bonded to validators.

Problem scenarios

Unnecessary code makes the codebase harder to understand and maintain. Additionally, `ExecuteRemoveValidator` can be called for non-validator addresses.

Recommendation

Remove the burning of free tokens tokens. Add a check at the beginning of `ExecuteRemoveValidator` to verify that it is only called for validator addresses.

Status

Resolved through:

- Pull request: https://github.com/xrplevm/node/pull/60

Messages MsgDelegate and MsgCancelUnbondingDelegation should be disabled

ID	IF-FINDING-005
Severity	Informational
Impact	0 - None
Exploitability	0 - None
Type	Protocol
Status	Resolved

Involved artifacts

- app/ante.go
- x/poa/keeper/hooks.go

Description

Since `BondDenom` tokens can only be minted and burned during validator creation (these tokens are immediately staked and bonded) and removal by an authority and both undelegate and redelegate messages are disabled, there are no free tokens available for delegation, nor any possibility of canceling unbonding delegation. This means the messages `MsgDelegate` and `MsgCancelUnbondingDelegation` are unnecessary and can be removed, simplifying the codebase.

Problem scenarios

No problem scenarios, just code clarity.

Recommendation

Explicitly disable `MsgDelegate` and `MsgCancelUnbondingDelegation` messages. If delegations are completely disabled, there is no need for `BeforeDelegationHook` implementation.

In order for authz-ed `MsgDelegate` and `MsgCancelUnbondingDelegation` messages to be disabled - add the messages among `AuthzDisabledMsgTypes` (code ref) in `HandlerOptions`.

Status

Resolved through:

- Pull request: https://github.com/xrplevm/node/pull/63

Separate revoke ownership from transfer ownership

ID	IF-FINDING-006
Severity	Informational
Impact	0 - None
Exploitability	0 - None
Type	Implementation
Status	Acknowledged

Involved artifacts

- precompiles/erc20/tx.go

Description

The current implementation of the `TransferOwnership` EVM precompile function is used to both transfer ownership and revoke ownership. There is no validation of the `newOwner` address, allowing the ownership to be transferred to a burned (zero) address, effectively revoking ownership.

Problem scenarios

It is possible to unintentionally revoke the token pair ownership, leading to a situation where the contract becomes unusable. While this issue can be resolved through a governance proposal, the voting period must first pass, leaving the contract inoperable during this time.

Recommendation

Separate the functionalities to prevent the unintentional transfer of ownership to the zero address and implement validation to prohibit ownership transfer to the zero address. This ensures that ownership is only transferred to an active entity, while a distinct function allows for intentional ownership removal when the contract needs to become ownerless. For best practices, refer to OpenZeppelin's `Ownable.sol` contract (here).

Minor observations and improvements related to the Cosmos SDK v0.50 upgrade

ID	IF-FINDING-007
Severity	Informational
Impact	0 - None
Exploitability	0 - None
Type	Implementation
Status	Resolved

Involved artifacts

- proto/poa/params.proto
- x/poa/types/params.go
- cmd/exrpd/cmd/root.go
- x/poa/module.go
- proto/evmos/erc20/v1/tx.proto
- x/erc20/types/msg.go

Description

The upgrade to v0.50.x guide suggests:

- *Removing `gogoproto.goproto_stringer = false` annotation from proto definitions and custom String() functions.*
  - For the `x/poa` module:
    - `x/poa` params contains this annotation (code ref) and
    - `String()` implementation (code ref).

    so these need to be removed, as suggested in the upgrade guide.
- *The `AppModuleBasic` interface has been simplified. Defining `GetTxCmd() *cobra.Command` and `GetQueryCmd() *cobra.Command` is no longer required. The module manager detects when module commands are defined. If AutoCLI is enabled, `EnhanceRootCommand()` will add the auto-generated commands to the root command, unless a custom module command is defined and register that one instead.*
  - For the `x/poa` module `GetTxCmd` and `GetQueryCmd` (code ref) can be removed, since `EnhanceRootCommand` is added (code ref).
  - After the sync meeting with the development team representatives, it was clarified that the current method for creating proposals can be changed. Proposals can be submitted using the standard approach:
    - Create a draft with `exrpd tx gov draft-proposal` and select `MsgAddValidator`.
    - Submit it with `exrpd tx gov submit-proposal draft_proposal.json`.

    This method works without relying on `GetTxCmd` and `NewSubmitAddValidatorProposalTxCmd`, meaning both can be safely removed. The team have been unaware of this alternative or simply preferred the convenience of the `tx poa add-validator` command.
- *`GetSigners()` is no longer required to be implemented on `Msg` types. The SDK will automatically infer the signers from the `Signer` field on the message. The signer field is required on all messages unless using a custom signer function.*
  - Signer fields are defined in proto definitions for `MsgBurn` (code ref) and `MsgTransferOwnership` (code ref), so the `GetSigners` functions (code ref1, ref2) can be removed as part of the upgrade Cosmos SDK from 0.47 to 0.50 version.

Problem Scenarios

No issues, only following the upgrade guide.

Recommendation

As listed in the Description section.

Status

Resolved through:

- Pull requests
  - Evmos changes (GetSigners function) https://github.com/xrplevm/evmos/pull/15
  - XRPLEVM changes (params annotation + removing GetQueryCmd) https://github.com/xrplevm/node/pull/64
  - XRPLEVM changes (removing GetTxCmd as suggested by @karolos) https://github.com/xrplevm/node/pull/64

Refactor mint.go by splitting responsibilities

ID	IF-FINDING-008
Severity	Informational
Impact	0 - None
Exploitability	0 - None
Type	Implementation
Status	Resolved

Involved artifacts

- x/erc20/keeper/mint.go

Description

The `mint.go` file currently contains multiple unrelated functionalities, including minting, burning, and ownership transfers. This lack of separation makes the code harder to maintain and understand.

Problem Scenarios

The current approach violates best coding practices, particularly the **Separation of Concerns** principle. Combining multiple responsibilities in a single file leads to:

- Reduced readability and maintainability
- Increased complexity when modifying specific functionalities
- Potential conflicts when working on different features

Recommendation

Refactor `mint.go` by splitting its functionalities into separate files:

- **`mint.go`** – Handles minting logic
- **`burn.go`** – Handles burning logic
- **`transfer_ownership.go`** – Manages ownership transfer

This restructuring will improve code clarity, maintainability, and adherence to best practices.

Status

Resolved through:

- Pull requests: https://github.com/xrplevm/evmos/pull/14

Appendix: Vulnerability classification

For classifying vulnerabilities identified in the findings of this report, we employ the simplified version of Common Vulnerability Scoring System (CVSS) v3.1, which is an industry standard vulnerability metric. For each identified vulnerability we assess the scores from the *Base Metric Group*, the Impact score, and the Exploitability score. The *Exploitability score* reflects the ease and technical means by which the vulnerability can be exploited. That is, it represents characteristics of the *thing that is vulnerable*, which we refer to formally as the *vulnerable component*. The *Impact score* reflects the direct consequence of a successful exploit, and represents the consequence to the *thing that suffers the impact*, which we refer to formally as the *impacted component*. In order to ease score understanding, we employ CVSS Qualitative Severity Rating Scale, and abstract numerical scores into the textual representation; we construct the final *Severity score* based on the combination of the Impact and Exploitability sub-scores.

As blockchains are a fast evolving field, we evaluate the scores not only for the present state of the system, but also for the state that deems achievable within 1 year of projected system evolution. E.g., if at present the system interacts with 1-2 other blockchains, but plans to expand interaction to 10-20 within the next year, we evaluate the impact, exploitability, and severity scores wrt. the latter state, in order to give the system designers better understanding of the vulnerabilities that need to be addressed in the near future.

Impact Score

The Impact score captures the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack.

ImpactScore	Examples
High	Halting of the chain; loss, locking, or unauthorized withdrawal of funds of many users; arbitrary transaction execution; forging of user messages / circumvention of authorization logic
Medium	Temporary denial of service / substantial unexpected delays in processing user requests (e.g. many hours/days); loss, locking, or unauthorized withdrawal of funds of a single user / few users; failures during transaction execution (e.g. out of gas errors); substantial increase in node computational requirements (e.g. 10x)
Low	Transient unexpected delays in processing user requests (e.g. minutes/a few hours); Medium increase in node computational requirements (e.g. 2x); any kind of problem that affects end users, but can be repaired by manual intervention (e.g. a special transaction)
None	Small increase in node computational requirements (e.g. 20%); code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation

Exploitability Score

The Exploitability score reflects the ease and technical means by which the vulnerability can be exploited; it represents the characteristics of the vulnerable component. In the below table we list, for each category, examples of actions by actors that are enough to trigger the exploit. In the examples below:

- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be
  - *legitimate*, e.g. submission of a transaction that follows protocol rules by a user; delegation/redelegation/bonding/unbonding; validator downtime; validator voting on a single, but alternative block; delays in relaying certain messages, or speeding up relaying other messages;
  - *illegitimate*, e.g. submission of a specially crafted transaction (not following the protocol, or e.g. with large/incorrect values); voting on two different alternative blocks; alteration of relayed messages.
- We employ also a *qualitative measure* representing the amount of certain class of power (e.g. possessed tokens, validator power, relayed messages): *small* for < 3%; *medium* for 3-10%; *large* for 10-33%, *all* for >33%. We further quantify this qualitative measure as relative to the largest of the system components. (e.g. when two blockchains are interacting, one with a large capitalization, and another with a small capitalization, we employ *small* wrt. the number of tokens held, if it is small wrt. the large blockchain, even if it is large wrt. the small blockchain)

ExploitabilityScore	**Examples**
High	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
Medium	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
Low	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
None	illegitimate actions taken in a coordinated fashion by all actors

Severity Score

The severity score combines the above two sub-scores into a single value, and roughly represents the probability of the system suffering a severe impact with time; thus it also represents the measure of the urgency or order in which vulnerabilities need to be addressed. We assess the severity according to the combination scheme represented graphically below.

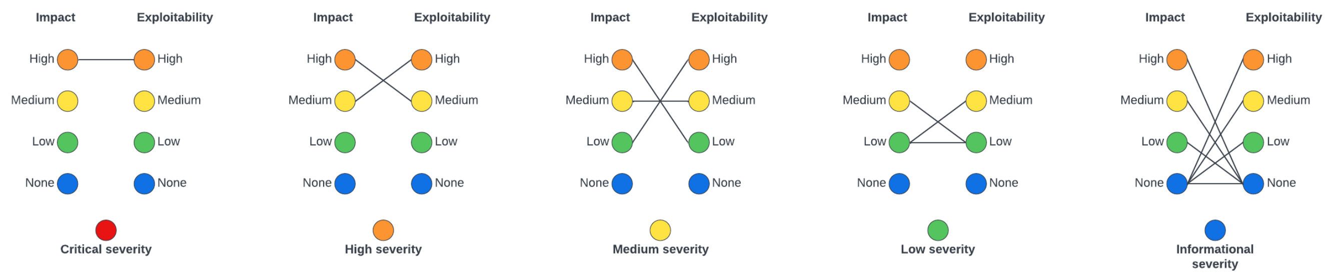

Figure 1: Severity classification

As can be seen from the image above, only a combination of high impact with high exploitability results in a Critical severity score; such vulnerabilities need to be addressed ASAP. Accordingly, High severity score receive vulnerabilities with the combination of high impact and medium exploitability, or medium impact, but high exploitability.

SeverityScore	Examples
Critical	Halting of chain via a submission of a specially crafted transaction
High	Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers
Medium	Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users
Low	2x increase in node computational requirements via coordinated withdrawal of all user tokens
Informational	Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability, etc.) set forth in the associated Services Agreement. This report provided in connection with the Services set forth in the Services Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This audit report is provided on an “as is” basis, with no guarantee of the completeness, accuracy, timeliness or of the results obtained by use of the information provided. Informal has relied upon information and data provided by the client, and is not responsible for any errors or omissions in such information and data or results obtained from the use of that information or conclusions in this report. Informal makes no warranty of any kind, express or implied, regarding the accuracy, adequacy, validity, reliability, availability or completeness of this report. This report should not be considered or utilized as a complete assessment of the overall utility, security or bugfree status of the code.

This audit report contains confidential information and is only intended for use by the client. Reuse or republication of the audit report other than as authorized by the client is prohibited.

This report is not, nor should it be considered, an “endorsement”, “approval” or “disapproval” of any particular project or team. This report is not, nor should it be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts with Informal 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 does it provide any indication of the client’s 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 it be leveraged as investment advice of any sort.

Blockchain technology and cryptographic assets in general and by definition present a high level of ongoing risk. Client is responsible for its own due diligence and continuing security in this regard.