Overview

Abstract

This document specifies the ContractManager module for the Neutron network.

The ContractManager module implements a mechanism and contains methods to make sudo calls to the contracts as well it helps to store sudo handler calls errors during IBC ACK.

Concepts

Due to the fact that contracts are allowed to make calls to the IBC, as well as process all received data, a problem arises in which a malicious contract can make a call to the IBC and, during the response of the ACK, make an error in the sudo handler, or simply do not implement it. Which will lead to disruption of the channel (in the case of a ORDERED channel), or force the relayer to send ACK requests over and over again, thereby loading the nodes serving the blockchain.

In order to avoid this problem, the code in the module from which the sudo handler is called should ignore the error and instead return the success status of the call. But this in turn exposes the problem of informing the owner of the contract that a failure has occurred. To do this, in case of an unsuccessful call, the module from which the sudo handler is called must call theAddContractFailure method of thekeeper of theContractManager module.

To ensure that the state of the contract is consistent, the call to the sudo handler takes place in a temporary state (usingCacheContext), which is written to the active state if the call succeeds.

SudoLimitWrapper

SudoLimitWrapper is a middleware wrapper with interface

type WasmKeeper interface

{ HasContractInfo (ctx sdk . Context , contractAddress sdk . AccAddress)

bool Sudo (ctx sdk . Context , contractAddress sdk . AccAddress , msg [ ] byte)

([ ] byte ,

error) } It performs two important functions:

1. Make sure that the sudo contract call does not use more gas than allowed by a chainparameter
2. . If the gas limit is exceeded and an out of gas panic occurs inside the sudo call, the wrapper intercepts the panic (only the out of gas panic is intercepted) and converts it to an error.
3. Capture an error from the sudo handler of the interchaintxs module, either directly initiated by the contract or an error that was received from an out of gas panic and write the error with the data toFailures
4. for further processing

Gas limitation

To make sure there are no exploits with infinite recursion of IBC messages which call other IBC messages in sudo handler we use constant gasLIMIT to spend. The LIMIT is small, so you can't place extensive work in Sudo handlers. As a workaround, in such cases you can use Sudo handlers to simply store required payload in contract's state, and use Execute messages to handle results separately.

If your contract exceeds this constantLIMIT , it will terminate sudo handler call and save aFailure with full call info. You canresubmit failure from this contract.

TheLIMIT is defined bySudoCallGasLimit module's parameter.

We provide an ability to resubmit bindings through the contract that initiated the IBC transaction.

Binding msgs

ResubmitFailure

Cosmwasm contracts that sent an IBC transfer or ICA transaction can resubmit their failures.

This binding is permissioned - only the contract that sent an IBC transfer or transaction can call it.

The format is as follows:

type ResubmitFailure struct

{ FailureId uint64

json:"failure_id" } Binding in cosmwasm described here .

It will call sudo handler with exact same arguments as the original handler that failed. The only difference is that thisSubmitHandler will be called not from relayer, so the gas limitations above do not apply.

Note that you can only resubmit failure through cosmwasm contract. See examples on how to resubmit failure for interchain txs and for ibc transfer

Binding queries

Failures for contract address can be queried using bindings .

Query request struct should be like:

type Failures struct

{ Address string

json:"address" Pagination * query . PageRequest json:"pagination,omitempty" } Response:

type FailuresResponse struct

{ Failures [ ] contractmanagertypes . Failure json:"failures" }

type Failure struct

{ // Address of the failed contract Address string

protobuf:"bytes,1,opt,name=address,proto3" json:"address,omitempty" // Id of the failure under specific address Id uint64

protobuf:"varint,2,opt,name=id,proto3" json:"id,omitempty" // Serialized MessageSudoCallback with Packet and Ack(if exists) SudoPayload [ ] byte

protobuf:"bytes,3,opt,name=sudo_payload,json=sudoPayload,proto3" json:"sudo_payload,omitempty" // Redacted error response of the sudo call. Full error is emitted as an event Error string

protobuf:"bytes,4,opt,name=error,proto3" json:"error,omitempty" } You're encouraged to use our neutron-sdk implementation in contracts -request and response

Failures details

In the case of a contract-initiated error, the error is stored in Failure . Since raw errors are not part of the consensus and cannot guarantee determinism when saved to the state, errors saved inFailure are redacted tocodespace + code_id using the RedactError function. At the same time, we emit the full text of the error into transaction events that do not affect the consensus and if you need to get detailed information about the error returned by the contract, you need to find the transaction in which the redacted error occurred and see the events in which we emit the error. To simplify this procedure we added a special cli query , the error can be found by theFailure.Address and theFailure.Id .Previous Messages Next Client