Audit Report

Router Chain 2

v1.0

April 30, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Kailaasa Infotech Pte Ltd to perform a security audit of several components of Router.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/router-protocol/router-chain
Commit	`f9ff8b44467d9d0b898b30726ffac421071d58e1`
Scope	All modules in the `x/` directory were in scope.
Fixes verified at commit	`4f44e99c38ea9fa4d63ef0c44ae26eabf7531138` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Router Chain is a layer one blockchain focusing on blockchain interoperability, enabling cross-chain communication with CosmWasm middleware contracts.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**High**	The complexity is flagged as high due to complex cross-chain requests, custom attestation mechanisms, and middleware implementations that interact with other components.
Code readability and clarity	**Low**	Multiple instances of unhandled errors, incorrect debugging comments, and unnecessary functions are found.
Level of documentation	**Medium**	Router Chain's whitepaper is available at https://www.routerprotocol.com/router-chain-whitepaper.pdf. Documentation is available at https://router-chain-docs.vercel.app/. However, the documentation lacks details for individual modules.
Test coverage	**Low**	Most of the test cases fail due to compilation errors.

Summary of Findings

No	Description	Severity	Status
1	Blocked cross-chain requests are indefinitely retried and could be exploited for denial-of-service attacks	Critical	Resolved
2	Non-deterministic iteration in `ClaimEventSlashing` can cause consensus failure	Critical	Resolved
3	Unapproved fee payers can block legitimate requests which can be exploited for denial-of-service attacks	Critical	Resolved
4	Anyone can submit forged cross-chain requests	Critical	Resolved
5	Anyone can submit forged cross-chain ack requests	Critical	Resolved
6	Unbonding validators can circumvent slashing	Critical	Resolved
7	Deactivated price feeders are not enforced	Critical	Resolved
8	Failed requests will not be reflected for `ErrorGasOverflow` panics	Major	Resolved
9	Failed cross-chain ack requests are incorrectly stored as successful	Major	Resolved
10	Executing crosstalk requests with a missing IBC relayer config results in indefinite retries and liveness issues	Major	Resolved
11	Anyone can create a new relayer configuration with a potentially invalid IBC channel, resulting in failed cross-chain requests via IBC	Major	Resolved
12	Symbol prices requested by the `BandChain` oracle are not namespaced by the price feeder name, resulting in the inability to retrieve prices	Major	Resolved
13	Custom application panics in `EndBlocker` can halt the chain	Major	Resolved
14	Unhandled errors in the codebase	Major	Resolved
15	Incomplete genesis handling causes loss of storage values	Major	Resolved

16	`GetLastUnBondingBlockHeight` will always return zero	Major	Resolved
17	Slashing risk due to sequential attestation processing	Major	Resolved
18	Zero values initialization affects gas handling and fee refunds	Major	Resolved
19	`HandleAckDefaultValidation` does not update the event block height, causing incorrect query responses to be returned	Minor	Resolved
20	Validators joining the network after attestations were submitted are subject to slashing and have to attest to previous claims	Minor	Resolved
21	`gas-prices` and `token-prices` CLI commands are ineffective due to not specifying the `PriceFeederName` and corresponding prices	Minor	Resolved
22	The `execute-cw-contract` CLI command does not supply `Msg` and `Funds` to the `MsgExecuteContract` message	Minor	Resolved
23	`rwasm` module does not support `InstantiateContract2`, `UpdateAdmin`, and `ClearAdmin`	Minor	Resolved
24	Price feeders can feed prices with older resolve times	Minor	Resolved
25	Amino codec must be registered to support end users with hardware devices like Ledger	Minor	Resolved
26	Errors are not propagated correctly	Minor	Resolved
27	`ListOrchestrators` might fail due to an out-of-gas error	Minor	Acknowledged
28	The `GetTokenPrice` query plugin lacks support for price staleness check	Informational	Resolved
29	`tallyChainAttestations` computation power can be reduced	Informational	Resolved
30	Unused chain type argument in `CmdValsetUpdatedClaim` can be removed	Informational	Resolved
31	`IterateAttestations` does not handle reverse argument implementation	Informational	Resolved
32	Incorrect comments and debug logging	Informational	Resolved

33	General code improvements in the codebase	Informational	Resolved
34	`HandleDeleteChainConfigProposal` can emit incorrect chain type	Informational	Resolved

Detailed Findings

1. Blocked cross-chain requests are indefinitely retried and could be exploited for denial-of-service attacks

Severity: Critical

In the `HandleFees` function in `x/crosschain/abci.go:18`, cross-chain requests are put in a blocked state (`CROSSCHAIN_TX_BLOCKED`) if fees cannot be deducted from the fee payer with the `DeductFeeInboundRequest`, `DeductFeeOutboundRequest` or `DeductFeeCrosstalkRequest` functions due to either not having a fee payer configured, insufficient fee payer balance, or the fee payer address not being approved yet. Such blocked cross-chain requests are then retried in the `EndBlocker` function in line `38` after every other block, specifically after every `params.BlockedRetryInterval` block interval.

However, if deducting fees from the fee payer continues to fail, the cross-chain request will be retried indefinitely. Consequently, this can potentially be exploited to cause denial-of-service attacks on Router Chain by submitting a large number of cross-chain requests that are blocked and retried indefinitely.

Similarly, the same issue persists for blocked ack requests if deducting fees fails in the `DeductFeeInboundAckRequest` and `DeductFeeCrosstalkAckRequest` functions when processing the blocked requests in the `ProcessBlockedCrosschainAckRequests` function in lines `551-592`.

Recommendation

We recommend implementing a timeout or exponential back-off strategy for retrying blocked requests before removing and stopping processing them. Additionally, we recommend modifying the implementation so users can retry the failed requests instead of executing them automatically in the `EndBlocker`.

Status: Resolved

2. Non-deterministic iteration in `ClaimEventSlashing` can cause consensus failure

Severity: Critical

In `x/attestation/abci.go:32-121`, attestations are processed by iterating over the `attmap` map. In Go, iterating over a map does not guarantee a consistent order of iteration, leading to non-deterministic behavior.

For instance, if the `UnpackAttestationClaim` function in line `78` errors, a panic will occur in line `80`. This could cause the resulting app hash computed by one validator to differ from

others. Consequently, the validators cannot reach a consensus within Tendermint's timeouts, causing a consensus failure and halting the chain.

Recommendation

We recommend sorting the attestation map keys before iteration to ensure they do not introduce non-deterministic behavior.

Status: Resolved

3. Unapproved fee payers can block legitimate requests which can be exploited for denial-of-service attacks

Severity: Critical

In `x/crosschain/abci.go:254` and `259`, any errors within the fee payer will cause the `Execute` function to `return` the execution early. This is problematic because other cross-chain requests ready to be executed will not be processed.

Consequently, an attacker can submit a cross-chain request without approving the fee payer to block other legitimate requests. The attacker can wait for the pending requests to accumulate to a large number and approve the fee payer to cause all pending requests to be processed at once. Since this function is executed in the `EndBlocker`, block production will be slowed, potentially halting the chain. This can be exploited for a denial of service attack.

Recommendation

We recommend replacing the `return` with a `continue` statement so other requests are processed accordingly.

Status: Resolved

4. Anyone can submit forged cross-chain requests

Severity: Critical

Cross-chain requests are submitted to Router Chain via the `MsgCrosschainRequest` message, handled in `x/crosschain/keeper/msg_server_crosschain_request.go:10`. Requests are processed differently based on the workflow type determined by calling the `WorkflowType` function in line `18`, which can `INBOUND`, `OUTBOUND`, or `CROSSTALK`.

The `INBOUND` workflow type is used for requests originating from a source chain and relayed to a CosmWasm contract deployed on Router Chain. The `OUTBOUND` workflow type is used for requests originating from CosmWasm contracts on Router Chain and relayed to a

destination chain. In any other case, the `CROSSTALK` workflow type is used for requests originating from a source chain and relayed to a destination chain.

`INBOUND` requests, constituting emitted events on a source chain, are submitted by the orchestrators of Router Chain and require a supermajority of orchestrators to attest to the request before it is considered valid. However, as there is currently no access control or in-depth validation of the submitted `MsgCrosschainRequest` message, anyone can send such a message, potentially with a malicious payload.

For instance, setting `SrcChainType` to `MultichainTypes.CHAIN_TYPE_ROUTER` to forge an `OUTBOUND` request does not require attestations as it is assumed that the request originates from Router Chain itself. This results in burning Router tokens from an arbitrary address specified in `RequestSender`, causing a loss of funds.

Additionally, it is also possible for an attacker to mint unlimited native Router tokens by forging an `INBOUND` request with an `IBC_VALIDATION` validation type. Such a validation type typically assumes that the request was received via IBC and thus is valid. In the case of a forged cross-chain request, the request is considered valid and processed in the `HandleNativeTransfer` function in `x/crosschain/abci.go:81`. Subsequently, the `HandleInboundNativeTokenFlow` function called in line `92` mints an arbitrary number of Router tokens to the attacker-controlled `RouteRecipient`.

The two depicted scenarios are just examples of how an attacker can exploit the lack of access control and validation of cross-chain requests. The ability to maliciously craft a `MsgCrosschainRequest` message leads to serious security implications for Router Chain.

Recommendation

We recommend validating the `MsgCrosschainRequest` to prevent forged requests from being processed. For instance, `OUTBOUND` cross-chain requests should only be accepted from Router Chain itself. Similarly, `IBC_VALIDATION` validation types should only be accepted if the request was received via IBC.

Status: Resolved

5. Anyone can submit forged cross-chain ack requests

Severity: Critical

As part of the acknowledgment workflow, cross-chain ack requests are submitted to Router Chain via the `MsgCrosschainAckRequest` message, handled in `x/crosschain/keeper/msg_server_crosschain_ack_request.go:10`. Ack requests are processed differently based on the workflow type determined by calling the `WorkflowType` function in line `17` and can be `INBOUND_ACK`, `OUTBOUND_ACK`, or `CROSSTALK_ACK`.

The `INBOUND_ACK` workflow type is used to send acknowledgments to a CosmWasm contract on Router Chain. The `OUTBOUND_ACK` workflow type is used for ack requests

originating from Router Chain and relayed to a destination chain. In any other case, the `CROSSTALK_ACK` workflow type is used.

`INBOUND_ACK` requests, constituting emitted ack events, i.e., `iReceive` on a source chain, are submitted by the orchestrators to Router Chain and require a supermajority of orchestrators to attest to the ack request before it is considered valid and executed. However, as there is currently no access control or in-depth validation of the submitted `MsgCrosschainAckRequest` message, anyone can send such a message, potentially with a malicious payload.

For instance, by sending a forged `MsgCrosschainAckRequest` message with an `INBOUND_ACK` workflow type and the validation type `IBC_VALIDATION`, the last observed event nonce will be updated by the `SetLastObservedEventNonce` function in `x/crosschain/keeper/ack_validation_handler.go:71`. This results in potentially griefing new legitimate cross-chain requests as they will be considered as already validated in `x/crosschain/keeper/validation_handler.go:45` due to the last observed event nonce being greater or equal to the submitted event nonce and thus not storing the request.

Subsequently, as soon as the supermajority of orchestrators attests and observes the event, it will not get further processed by the handlers in `x/crosschain/keeper/hooks.go` as there is no such request stored in the database.

The ability to maliciously craft a `MsgCrosschainAckRequest` message leads to serious security implications for Router Chain.

Recommendation

We recommend validating the `MsgCrosschainAckRequest` to prevent forged requests from being processed. `OUTBOUND_ACK` cross-chain ack requests should only be accepted from Router Chain itself. Similarly, `IBC_VALIDATION` validation types should only be accepted if the ack request was received via IBC.

Status: Resolved

6. Unbonding validators can circumvent slashing

Severity: Critical

During the `EndBlocker` slashing process, the `ClaimEventSlashing` function, implemented in `x/attestation/abci.go:27-122`, is called in line `22`. However, only bonded validators are considered for slashing if they submit conflicting attestations or miss attesting an observed claim. This is problematic because unbonding validators are not exposed to the slashing penalty.

Consequently, if an orchestrator reports such a slashable behavior, the corresponding validator can unbond from the validator set to avoid the slashing penalty.

Recommendation

We recommend slashing validators with unbonding status in the `ClaimEventSlashing` function.

Status: Resolved

7. Deactivated price feeders are not enforced

Severity: Critical

When configuring price feeders in `x/pricefeed/proposal_handler.go:34`, governance can set the `Active` boolean to indicate whether the price feeder is activated or deactivated (see `x/pricefeed/types/price_feeder_info.pb.go:28`). However, this is not enforced through the codebase.

Specifically, the `TokenPrices` and `GasPrices` functions in `x/pricefeed/keeper/msg_server_token_prices.go:11` and `x/pricefeed/keeper/msg_server_gas_prices.go:11` do not validate whether the price feeder's `Active` status is `true` before allowing them to update the prices. Consequently, governance cannot remove or disable malicious price feeders after adding them.

Recommendation

We recommend ensuring the price feeder's `Active` status is `true` in the `TokenPrices` and `GasPrices` functions.

Status: Resolved

8. Failed requests will not be reflected for `ErrorGasOverflow` panics

Severity: Major

In the following code locations, the `recover` function does not modify the exec data and exec flag when the `default` statement is entered:

- `x/crosschain/keeper/inbound_ack_execution.go:66-69`
- `x/crosschain/keeper/inbound_execution.go:65-68`
- `x/voyager/keeper/fund_deposit_execution.go:60-63`
- `x/voyager/keeper/fund_paid_execution.go:52-55`
- `x/voyager/keeper/update_deposit_info_execution.go:54-57`

In comparison, the function handles the `ErrorOutOfGas` panic case when executing the `ConsumeGas` function. If that happens, the exec flag will be set to `false`, and the exec data

will be updated with the error. However, this is not performed when the `default` statement is entered.

This will happen if the `ErrorGasOverflow` panic occurs for overflows when computing the gas consumption amount. Consequently, the exec flag and exec data will not be updated to indicate the failed request, causing incorrect events to be emitted.

Recommendation

We recommend updating the exec flag and data if the `default` statement is entered.

Status: Resolved

9. Failed cross-chain ack requests are incorrectly stored as successful

Severity: Major

In `x/crosschain/keeper/ack_workflow.go:93-100`, no `return` statement is implemented to exit the `HandleOutboundAck` function early after the cross-chain request has failed.

Specifically, if an error occurs during `SendIBCPayload` in line `86`, the cross-chain acknowledgment request status is stored as `CROSSCHAIN_ACK_TX_EXECUTION_FAILED`. Since no `return` statement is implemented after updating the status, lines `99-100` will continue to execute, causing the status to be incorrectly updated as `CROSSCHAIN_ACK_IBC_TRANSFERRED`. Consequently, a cross-chain request that fails will be incorrectly stored as a success.

This issue is also present in the `HandleOutbound` function in `x/crosschain/keeper/workflow.go:124-131`.

Recommendation

We recommend adding a `return` statement in both `HandleOutboundAck` and `HandleOutbound` functions after the request has failed.

Status: Resolved

10. Executing crosstalk requests with a missing IBC relayer config results in indefinite retries and liveness issues

Severity: Major

The `Execute` function in `x/crosschain/abci.go:321-361` iterates and processes all stored cross-chain requests with the `CROSSCHAIN_TX_READY_TO_EXECUTE` status in the `EndBlocker` function at the end of every block.

In the case of a `CROSSTALK` request, a cross-chain request originates from a source chain and is intended to get relayed to a destination chain. If the destination chain is a Cosmos chain, the relayer config is retrieved by the `CrosschainRequest.DestChainId` function to determine the IBC channel, `RelayerConfig.Channel`. Subsequently, this IBC channel is used to send the cross-chain request from Router Chain to the destination Cosmos chain via IBC.

However, if no such relayer configuration exists for the provided `CrosschainRequest.DestChainId`, for instance, because the chain ID is invalid or the relayer config is not yet stored, the request status remains unchanged, i.e., `CROSSCHAIN_TX_READY_TO_EXECUTE`. This results in the cross-chain request execution to be retried by the `Execute` function in the next block, potentially indefinitely. As there is currently no validation of `CrosschainRequest.DestChainId`, allowing anyone to define an arbitrary destination chain ID, this behavior can be exploited to spam Router Chain nodes with cross-chain requests that are repeatedly processed and retried, allowing a denial-of-service attack.

Similarly, the same issue persists for `CROSSTALK_ACK` requests processed by the `ExecuteAck` function in `x/crosschain/abci.go:443-478`.

Recommendation

We recommend validating the cross-chain destination chain ID, `DestChainId`, to ensure it is a valid chain ID with a corresponding relayer config. Otherwise, the request should be deleted.

Status: Resolved

11. Anyone can create a new relayer configuration with a potentially invalid IBC channel, resulting in failed cross-chain requests via IBC

Severity: Major

Cross-chain requests are sent to a Cosmos destination chain via IBC. The IBC channel used for sending the request is configured in the relayer configuration. Relayer configurations are created and managed via the `MsgCreateRelayerConfig`, `MsgUpdateRelayerConfig`,

and `MsgDeleteRelayerConfig` messages handled in `x/crosschain/keeper/msg_server_relayer_config.go:11-91`.

However, anyone can create a new relayer configuration (before the Router team is able to set the legitimate relayer config), as the `CreateRelayerConfig` function does not implement any access control mechanism.

Consequently, malicious users can create a new relayer config for a specific Cosmos destination chain with an invalid IBC channel. This will result in cross-chain requests failing when attempting to send them via IBC to the destination chain with the invalid IBC channel.

Once a relayer config is created, it can only be updated and deleted by the creator of said relayer config. An invalid relayer config will thus prevent a correctly configured relayer config from being created for the same Cosmos destination chain.

Recommendation

We recommend adding access control to allow only the Router team/governance to create and manage relayer configurations.

Status: Resolved

12. Symbol prices requested by the `BandChain` oracle are not namespaced by the price feeder name, resulting in the inability to retrieve prices

Severity: Major

The `StoreOracleResponsePacket` function in `x/pricefeed/keeper/keeper.go:210` receives an `OracleResponsePacketData` packet containing the requested symbol prices from the `BandChain` oracle. The contained prices are looped over, and the `UpdatePrice` function in line `220` updates the stored symbol prices. Symbol prices are namespaced by a price feeder name, `PriceFeeder`, which is usually either `ROUTER_PRICE_FEEDER = "router_price_feeder"` or `BAND_PRICE_FEEDER = "band_feeder"`.

Retrieving the price of a specific symbol with the `GetPriceBySymbol` function in `x/pricefeed/keeper/price.go:80` first attempts to return the price with the `BAND_PRICE_FEEDER` price feeder name or, if not found, with the `ROUTER_PRICE_FEEDER` price feeder name.

However, the `StoreOracleResponsePacket` function does not set the `PriceFeeder` field of the `Price` struct when updating the symbol prices. This leads to the symbol price being incorrectly stored without a prefix, resulting in the `GetPriceBySymbol` function not being able to retrieve the price by the symbol name.

Consequently, the `ConvertNativeTokenFeeToRouter` function, used within the `crosschain` module, is unable to retrieve the price of the chain's native token and returns the error "Token price not found" when processing cross-chain requests.

Recommendation

We recommend setting the `PriceFeeder` field of the `Price` struct when updating the symbol prices from the requested `BandChain` oracle in the `StoreOracleResponsePacket` function.

Status: Resolved

13. Custom application panics in `EndBlocker` can halt the chain

Severity: Major

Within the `x/attestation` and `x/crosschain` modules, custom panics are implemented in the `EndBlocker` if application-specific errors are encountered, which leads the chain to halt.

While the SDK itself might intentionally utilize panics as preventive measures against unintended bugs, custom panics within the application's logic in ABCI methods can accidentally introduce vulnerabilities. For example, a malicious actor can halt the chain if they trigger an application-specific logic error, causing a denial of service attack.

That said, it is important to note that panicking within ABCI methods can be appropriate under specific circumstances. For instance, if an error in the application-specific logic indicates a critical flaw in the protocol, halting the chain and alerting stakeholders might be beneficial to prevent the issue from causing further damage.

Recommendation

We recommend revising the custom panic logic to determine whether the chain needs to be halted in case an application-specific error occurs. If a chain halt is unnecessary, we recommend logging the error and implementing a defer function that recovers when a panic is triggered.

Status: Resolved

14. Unhandled errors in the codebase

Severity: Major

In several instances of the codebase, functions that return an error are not checked. This would cause silent failures as errors are not raised to notify users.

- `x/attestation/keeper/attestation.go:220, 329`

- x/attestation/keeper/msg_server_set_orchestrator_address.go:58
- x/attestation/keeper/msg_server_valset_confirm.go:53
- x/attestation/keeper/msg_server_valset_updated_claim.go:44
- x/attestation/keeper/valset.go:42
- x/attestation/module.go:80
- x/crosschain/keeper/ack_fee_handler.go:186,194
- x/crosschain/keeper/ack_workflow.go:17,19,110,113,120
- x/crosschain/keeper/event_handler.go:16, 36, 47, 56, 64, 77, 85, 93, 103,117,135,146,154,167,175,183,192,204,216
- x/crosschain/keeper/fee_handler.go:194,200
- x/crosschain/keeper/gov/gov.go:37,72
- x/crosschain/keeper/native_token_flow.go:52
- x/crosschain/keeper/workflow.go:153
- x/crosschain/module.go:74
- x/crosschain/types/metadata.go:61
- x/pricefeed/module.go:76
- x/multichain/keeper/gov/gov.go:40,86,125,168,220,259
- x/multichain/module.go:80
- x/rwasm/module.go:73
- x/voyager/keeper/event_handler.go:16, 38, 45, 53, 63, 81, 88, 96, 106, 123,130,138
- x/voyager/module.go:74
- x/metastore/abci.go:33
- x/metastore/keeper/msg_server_approve_feepayer_request.go:30
- x/metastore/keeper/msg_server_set_metadata_request.go:24,51
- x/metastore/module.go:82

Recommendation

We recommend handling the error of the above functions.

Status: Resolved

15. Incomplete genesis handling causes loss of storage values

Severity: Major

In several instances of the codebase, the genesis state does not handle the storage values properly in either the initialization or export functions.

Firstly, the `InitGenesis` function in `x/attestation/genesis.go:45-50` does not call `SetValsetUpdatedClaim` for the provided `genState.ValsetUpdatedClaimList`. This means that although it is exported in line `64`, it cannot be added back when initializing the genesis state, causing all existing `ValsetUpdatedClaimList` storage values to be lost.

Secondly, the `InitGenesis` function in `x/voyager/genesis.go:30-33` does not call `SetDepositUpdateInfoRequest` for the provided `genState.DepositUpdateInfoRequestList`. This means that although it is exported in line `46`, it cannot be added back when initializing the genesis state, causing all existing `DepositUpdateInfoRequestList` storage values to be lost.

Lastly, the `InitGenesis` and `ExportGenesis` functions in `x/crosschain/genesis.go:10-84` do not handle `BlockedCrosschainRequestList` and `BlockedCrosschainAckRequestList`. This means that if the genesis state was exported and the chain is reinitialized, all existing `BlockedCrosschainRequestList` and `BlockedCrosschainAckRequestList` storage values will be lost.

Recommendation

We recommend handling all storage values mentioned above.

Status: Resolved

16. `GetLastUnBondingBlockHeight` will always return zero

Severity: Major

In `x/attestation/abci.go:135`, the last unbonding block height is retrieved with `GetLastUnBondingBlockHeight` and compared in line `162` to determine whether `SetValsetRequest` needs to be called. However, there is no function that sets the storage value of `types.LastUnBondingBlockHeight`. Consequently, the last unbonding block height will always return zero in `x/attestation/keeper/valset.go:259`, rendering it useless because the value will never equal the current block height.

Recommendation

We recommend calling the `SetLastUnBondingBlockHeight` function when a validator starts unbonding.

Status: Resolved

17. Slashing risk due to sequential attestation processing

Severity: Major

In `x/attestation/keeper/attestation.go:192`, the `Attest` function ensures the claim event nonce equals the incremented last claim event nonce to process attestations sequentially. This can become problematic when a validator unbonds and later re-bonds, for example, months or even a year later.

In such cases, the validator is obligated to attest to all claims that occurred during their absence. Processing this backlog of in-between claims can be time-consuming, and there's a risk that the re-bonded validator might miss attesting to some claims, leading to slashing risks.

Recommendation

We recommend removing the `LastEventByValidator` when a validator has completed the unbonded process.

Status: Resolved

18. Zero values initialization affects gas handling and fee refunds

Severity: Major

In the following instances of the codebase, the variables are set to zero instead of the intended values.

Firstly, the `DestGasLimit` and `DestGasPrice` in `x/crosschain/abci.go:230-231` are set to zero instead of `gasLimit` and `gasPrice`. This is also the same for the `AckGasLimit` and `AckGasPrice` functions in `x/crosschain/abci.go:584-585`, as their value is also set to zero instead of `ackGasLimit` and `ackGasPrice`. This would cause the sudo message to fail due to an out-of-gas error because the computed gas fee limit will be zero.

Besides that, the `CreateCrosschainAckRequest` function in `x/crosschain/abci.go:313` sets the `feeConsumed` to zero instead of `feeConsumedInRoute`. This would cause the relayer to be refunded with all the fees even though the sudo message did consume gas.

Recommendation

We recommend updating the variables to their intended values mentioned above.

Status: Resolved

19. `HandleAckDefaultValidation` does not update the event block height, causing incorrect query responses to be returned

Severity: Minor

In `x/crosschain/keeper/ack_validation_handler.go:71`, the `HandleAckDefaultValidation` function does not update the last observed event height with `SetLastObservedEventBlockHeight` after updating the last observed event nonce with `SetLastObservedEventNonce`. This is inconsistent with others as the last observed event height is updated in `x/attestation/keeper/attestation.go:324`,

`x/crosschain/keeper/ack_workflow.go:37`, and `x/crosschain/keeper/workflow.go:67`.

Consequently, the lowest observed height retrieved in `x/attestation/keeper/attestation.go:244` and used in line `261` will be an outdated value, causing the `LastEventNonce` query to return incorrect event height in `x/attestation/keeper/grpc_query_last_event_nonce.go:25`.

Recommendation

We recommend calling `SetLastObservedEventBlockHeight` in the `HandleAckDefaultValidation` function.

Status: Resolved

20. Validators joining the network after attestations were submitted are subject to slashing and have to attest to previous claims

Severity: Minor

The `ClaimEventSlashing` function in `x/attestation/abci.go:73-120`, called at the end of every block in `EndBlocker`, slashes currently bonded validators for missing votes on observed attestations once the signing window has passed.

However, newly bonded validators who joined after an attestation (claim) was submitted have to attest to those previous claims as well, or they will be slashed.

Recommendation

We recommend only slashing validators that were already bonded and part of the validator set at the time the attestation was submitted.

Status: Resolved

21. `gas-prices` and `token-prices` CLI commands are ineffective due to not specifying the `PriceFeederName` and corresponding prices

Severity: Minor

The `gas-prices` CLI command in `x/pricefeed/client/cli/tx_gas_prices.go:33` is intended to set the gas prices for various destination chains. Similarly, the `token-prices` CLI command in

`x/pricefeed/client/cli/tx_token_prices.go:33` is intended to set the token prices.

However, both commands fail to provide the `PriceFeederName` as well as the gas prices and token prices to the corresponding message. This results in the gas prices and token prices not being set.

Recommendation

We recommend supplying the `PriceFeederName` and `GasPrices` to `NewMsgGasPrices` when constructing the `MsgGasPrices` message.

Status: Resolved

22. The `execute-cw-contract` CLI command does not supply `Msg` and `Funds` to the `MsgExecuteContract` message

Severity: Minor

The `execute-cw-contract` CLI command in `x/rwasm/client/cli/tx_execute_cw_contract.go` does not supply the `Msg` and `Funds` parameters to the `MsgExecuteContract` message. Consequently, this prevents sending funds and passing a message to the executed contract.

Recommendation

We recommend adding the `Msg` and `Funds` parameters to the `execute-cw-contract` CLI command and passing them to the `MsgExecuteContract` message.

Status: Resolved

23. `rwasm` module does not support `InstantiateContract2`, `UpdateAdmin`, and `ClearAdmin`

Severity: Minor

In `x/rwasm/types/tx.pb.go:616-621`, the `rwasm` module does not support `InstantiateContract2`, `UpdateAdmin`, and `ClearAdmin`. This means that it is not possible for users to instantiate contracts with precomputed addresses and modify contract admin configurations.

Recommendation

We recommend implementing the `InstantiateContract2`, `UpdateAdmin`, and `ClearAdmin` messages.

Status: Resolved

24. Price feeders can feed prices with older resolve times

Severity: Minor

In `x/pricefeed/keeper/msg_server_token_prices.go:24-27`, the `TokenPrices` function does not validate that the new price's resolve time is larger than the old price's resolve time. Ideally, only higher resolve time should be added to ensure the price retrieved is the latest one, as indicated in `x/pricefeed/keeper/price.go:72`.

If the price feeder updates the token price with an older resolve time, the `TokenPrices` function will not reject it. Consequently, older prices can be fed by price feeders, causing outdated prices to be used across the codebase.

We classify this issue as minor because only the governance can configure price feeders, which should be a trusted entity.

Recommendation

We recommend ensuring the new price's `ResolveTime` is larger than the old price, similar to `x/pricefeed/keeper/price.go:72`.

Status: Resolved

25. Amino codec must be registered to support end users with hardware devices like Ledger

Severity: Minor

In the following code lines, Amino is not used to register all interfaces and concrete types for the modules:

- `x/attestation/types/codec.go`
- `x/crosschain/types/codec.go`
- `x/metastore/types/codec.go`
- `x/pricefeed/types/codec.go`
- `x/voyager/types/codec.go`

This is necessary for legacy binary and JSON serialization to support hardware devices like Ledger since these devices do not support proto-transaction signing.

Recommendation

We recommend registering all interfaces and concrete types for the modules using Amino to support legacy binary and JSON serialization.

Status: Resolved

26. Errors are not propagated correctly

Severity: Minor

The following code lines illustrate instances where errors are not propagated correctly:

- `x/crosschain/abci.go:338`
  - Errors during marshal are not propagated.
  - The error should be handled similarly to `x/crosschain/abci.go:349-353`.
- `x/crosschain/abci.go:459`
  - Errors during marshal are not propagated.
  - The error should be handled similarly to `x/crosschain/abci.go:468-472`.
- `x/crosschain/keeper/ack_workflow.go:83, 88`
  - Errors during marshal and `SendIBCPayload` are not propagated.
  - The error should be handled similarly to lines `94` to `97` with a `return` statement.
- `x/crosschain/keeper/fee_handler.go:150`
  - Errors when computing fee payer address is not handled.
  - The error should be handled by returning the statement early.
- `x/crosschain/keeper/fee_handler.go:196`
  - Errors when computing relayer router address is not handled.
  - The error should be handled by returning the statement early.
- `x/crosschain/keeper/inbound_ack_execution.go:89`
  - Errors during marshal are not propagated.
  - The error should be handled similarly to lines `104` to `105` before returning the statement.
- `x/crosschain/keeper/workflow.go:114`
  - Errors during marshal are not propagated.
  - The error should be handled similarly to `x/crosschain/keeper/workflow.go:125-128`.

Recommendation

We recommend handling the mentioned errors above.

Status: Resolved

27. `ListOrchestrators` might fail due to an out-of-gas error

Severity: Minor

In `x/attestation/keeper/grpc_query_list_orchestrators.go:20`, the `ListOrchestrators` function attempts to list all orchestrators by iterating over all `EthAddressByValidatorKey` and `KeyOrchestratorAddress` prefixes.

Consequently, the query will fail if many validators are registered due to an out-of-gas error.

Recommendation

We recommend applying pagination to the `ListOrchestrators` function.

Status: Acknowledged

The client states that the orchestrator set will not be very big, with a size of 100-200 orchestrators.

28. The `GetTokenPrice` query plugin lacks support for price staleness check

Severity: Informational

In `wasmbinding/queries.go:35-41`, the `GetTokenPrice` function allows a CosmWasm contract to query token prices. However, the returned `TokenPriceResponse` struct does not include the token's resolution time. Consequently, the caller cannot determine whether the returned price is stale or fresh.

We classify this issue as informational because the usage of the `GetTokenPrice` function depends on the caller. Providing the resolution time allows the caller to have the option of handling outdated prices, such as erroring the transaction.

Recommendation

We recommend including `tokenPriceState.ResolveTime` in the `TokenPriceResponse` struct so the caller can check stale prices.

Status: Resolved

29. `tallyChainAttestations` computation power can be reduced

Severity: Informational

In `x/attestation/abci.go:189-215`, the `tallyChainAttestations` function uses a nested loop to iterate over all attestations from all nonces and then processes the attestation if the nonce equals the incremented last observed event nonce.

Since the attestation only needs to be processed after the nonce validation, computation power can be reduced by iterating all the attestations only if the nonce equals the incremented last observed event nonce.

Recommendation

We recommend only iterating the attestations if the nonce equals the incremented last observed event nonce, illustrated in the pseudocode below:

```
for _, nonce := range keys {
  if nonce == GetLastObservedEventNonce + 1 {
    for _, att := range attmap[nonce] {
      k.TryAttestation(ctx, att)
    }
  }
}
```

Status: Resolved

30. Unused chain type argument in `CmdValsetUpdatedClaim` can be removed

Severity: Informational

In `x/attestation/client/cli/tx_valset_updated_claim.go:60`, the chain type argument is required to be provided in `CmdValsetUpdatedClaim` function. However, it is not used when dispatching a `MsgValsetUpdatedClaim` message, as seen in `x/attestation/types/message_valset_updated_claim.go:16-25`.

Recommendation

We recommend removing the chain type argument from `CmdValsetUpdatedClaim`.

Status: Resolved

31. `IterateAttestations` does not handle `reverse` argument implementation

Severity: Informational

In `x/attestation/keeper/attestation.go:115`, the `IterateAttestations` function does not adjust the iteration order based on the `reverse` argument. Specifically, when the `reverse` argument is set to `true`, the function should use `sdk.KVStoreReversePrefixIterator` to ensure keys are iterated in descending order. Currently, the function iterates keys in ascending order irrespective of the `reverse` value.

We classify this issue as informational because the `IterateAttestations` function is only used in `x/attestation/keeper/attestation.go:92` with the `reverse` argument set to `false`. If future development implements the `IterateAttestations` function with the `reverse` argument as `true`, the iteration order will be incorrect.

Recommendation

We recommend using `sdk.KVStoreReversePrefixIterator` if the `reverse` argument is provided as `true`.

Status: Resolved

32. Incorrect comments and debug logging

Severity: Informational

The following code lines illustrate instances where comments and debugging are incorrect:

- In `x/crosschain/abci.go:579`, the debug logging should include "crosstalk or outbound" because the case statement includes `OUTBOUND_ACK` and `CROSSTALK_ACK`.
- In `x/crosschain/keeper/ack_workflow.go:45`, the debug logging should be "outbound ack" instead of "inbound ack".
- In `x/crosschain/keeper/fee_handler.go:73`, the error message should be "fee payer not found or not approved" because the statement would be entered if the fee payer is unapproved.
- In `x/voyager/abci.go:65` and `89`, the debug logging incorrectly emits "*DepositId*" as `fundPaidRequest.EventNonce`. This is incorrect as the "*DepositId*" should be "*EventNonce*".
- In `x/voyager/keeper/fund_paid_execution.go:58`, the descriptor for `ConsumeGas` mentions "*contract execution in Begin Blocker*". This is incorrect because `ExecuteFundPaidRequest` is executed in `EndBlocker`.
- In `x/voyager/keeper/update_deposit_info_execution.go:20`, the debug logging mentions it is processing fund deposit requests. This is incorrect because it is handling updated deposit info requests, not fund deposit requests.

Recommendation

We recommend correcting the incorrect comments and debug loggings above.

Status: Resolved

33. General code improvements in the codebase

Severity: Informational

The following code lines illustrate instances where code improvements can be made:

- The `if` condition in `x/crosschain/abci.go:467` can be combined directly with line `463`.
- The unused variables in `x/crosschain/keeper/crosschain_request.go:150-154` can be removed.
- The duplicate `SetMetaInfo` validation in `x/metastore/genesis.go:26-28` can be removed as it is already performed in lines `12` to `14`.

Additionally, multiple spelling issues have been observed. For example, the `ProcesValsetUpdatedClaimObservedHook` function in `x/attestation/keeper/hooks.go:38` should be named `ProcessValsetUpdatedClaimObservedHook`.

Recommendation

We recommend handling the mentioned code lines above to increase code readability and decrease code complexity. The `cspell` CLI tool can be used to detect all spelling errors.

Status: Resolved

34. `HandleDeleteChainConfigProposal` can emit incorrect chain type

Severity: Informational

In `x/multichain/keeper/gov/gov.go:127`, the `ChainType` emitted depends on the submitted value in `MultichainDeleteChainConfigProposal`. However, the `RemoveChainConfig` function only uses the `ChainId` when removing the chain configuration, leaving the `ChainType` unused.

Consequently, if the `ChainType` provided is not equivalent to `ChainId`, the emitted `EventDeletedChainConfig` value will be incorrect. For example, it is possible to remove a Cosmos chain with the chain type specified as Ethereum, causing incorrect events to be emitted.

Recommendation

We recommend emitting the correct `ChainType` based on the `GetChainConfig` return value.

Status: Resolved