Audit Report

Router Orchestrator

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 Router's orchestrator service.

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-orchestrator
Commit	`fee27164a82be6af1b366c1045ac09cc70b33827`
Scope	All files were in scope.
Fixes verified at commit	`34a84e26745ac6ecba3a603a6b19c7f281ef8a74` 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 orchestrators listen to incoming cross-chain requests from other chains, attest their validity, parse them into a unified format, and post them on Router Chain. These attested requests can then be picked up by the relayers and forwarded to the destination chain.

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	Medium	The orchestrator listens, attests, and dispatches requests via Router Chain to multiple chains whose implementation differs from each other.
Code readability and clarity	Low-Medium	There were many outstanding `TODO` comments and incomplete function and configuration implementations.
Level of documentation	Medium	Documentation is available at https://docs.routerprotocol.com/validators/orchestrators.
Test coverage	Low	There were minimal test cases in the codebase.

Summary of Findings

No	Description	Severity	Status
1	Events emitted from the EVM gateway and voyager contracts are potentially processed out of order, resulting in events being skipped and not sent to Router Chain	Critical	Resolved
2	Inbound and outbound `CROSSTALK` requests originating from non-Cosmos chains are ignored and not able to be relayed	Critical	Acknowledged
3	Transaction origin is hardcoded as an empty string when transforming an `iSend` event	Critical	Resolved
4	A potential unconfirmed block is processed in the NEAR event listener	Critical	Resolved
5	Suboptimal processing of attestations leading to missing attestations	Major	Resolved
6	Risk of liveness slashing due to inconsistent chain configurations	Major	Acknowledged
7	Unhandled errors in the codebase	Major	Partially Resolved
8	Incorrect gas price denom used	Major	Resolved
9	Undetected initialization errors and premature main termination complicate thread management	Minor	Resolved
10	`MqConsumer` lacks updating the status and error code of transactions in the database in case of an error	Minor	Resolved
11	Sequential transactions are potentially returned out of order from the database	Minor	Resolved
12	Voyager implementation for TRON and NEAR network is incomplete and incorrect	Minor	Resolved
13	Retrying failed Router Chain queries without delay between retries may overload the RPC node	Minor	Resolved
14	The NEAR network used for querying events is hard-coded to the testnet	Minor	Resolved
15	Dispatcher lacks updating the status of	Minor	Resolved

	transactions in the database		
16	Logger is hardcoded as debug level	Minor	Resolved
17	Unprotected Cosmos keys in configuration	Minor	Resolved
18	Insecure HTTP and GRPC exposing sensitive data	Informational	Resolved
19	Processed inbound requests are stored in the `ProcessedOutboundBatch` table intended for outbound requests	Informational	Acknowledged
20	Health check does not process voyager listeners	Informational	Resolved
21	Incorrect debug logging	Informational	Resolved
22	Errors are incorrectly displayed due to incorrect format printing	Informational	Resolved
23	Newly added database records are logged as errors	Informational	Resolved
24	Same log color is used for different logging levels	Informational	Resolved
25	Validator set updated event is not logged along with other fields	Informational	Resolved
26	`QueryGatewayValsetUpdatedEvents` computational complexity can be reduced	Informational	Resolved
27	Unused `path` variable during database initialization	Informational	Resolved

Detailed Findings

1. Events emitted from the EVM gateway and voyager contracts are potentially processed out of order, resulting in events being skipped and not sent to Router Chain

Severity: Critical

Emitted events from the gateway and voyager contracts on supported EVM chains, such as `iSend` and `iReceive`, are listened to by the `EvmGatewayEventProcessor`, and `EvmVoyagerEventProcessor`, respectively. Subsequently, events are sent to Router Chain. The `ProcessInboundEvents` function in `listener/evm/gatewayeventprocessor/gatewayeventprocessor.go:52` and `listener/evm/voyagereventprocessor/voyagereventprocessor.go:52` queries the events emitted within the specified block range by the source chain and sends them to Router Chain. Router Chain expects the events to be submitted in sequence, i.e., the events must be ordered by the `EventNonce`. This monotonically increasing number is incremented for each event emitted from the gateway and voyager contracts.

Internally, the various events, split up in individual arrays per event type, are sorted sequentially by using the merge sort algorithm, implemented in the `SortAndTransformInboundEventsByEventNonce` function in `gatewayeventprocessor.go:228-337` and `SortAndTransformVoyagerEventsByEventNonce` function in `voyagereventprocessor.go:189-281`.

Specifically, a `for` loop is implemented, which iterates `n` times, where `n` is the number of events. In each iteration, the loop iterates over all individual event arrays to find the next event in sequence by comparing the `EventNonce` of the current event with the `lastProcessedEventNonce` variable. If the `EventNonce` is equal to `lastProcessedEventNonce + 1`, the event is included in the `msgs` array, containing the sorted events. Otherwise, the event is considered to have already been processed and skipped. Afterward, the `lastProcessedEventNonce` variable is updated with the `EventNonce` of the current event.

To ensure the merge sort algorithm's correct functioning, the event arrays, e.g., `iSendEvents` or `iReceiveEvents`, must be sorted by the `EventNonce`. Otherwise, events will be incorrectly skipped and not processed.

However, contrary to the event listeners for the NEAR and TRON chains, the EVM gateway and voyager listener do not explicitly sort the individual event arrays by the `EventNonce`. Instead, the events are queried from the source EVM chain by the underlying `eth_getLogs` RPC call, which does not guarantee that the events are returned in order. Consequently, the events are not sorted by the `EventNonce`, and the merge sort algorithm does not work as intended, resulting in events being skipped and not sent to Router Chain.

Recommendation

We recommend explicitly sorting the individual event arrays in `listener/evm/gatewayeventprocessor/gatewayeventprocessor.go:228` and `listener/evm/voyagereventprocessor/voyagereventprocessor.go:189` before passing them to the merge sort algorithm, similar to the NEAR implementation in `listener/near/gatewayeventprocessor/gatewayeventprocessor.go:202-259`.

Status: Resolved

2. Inbound and outbound `CROSSTALK` requests originating from non-Cosmos chains are ignored and not able to be relayed

Severity: Critical

Inbound and outbound requests are processed and confirmed by orchestrators within the `Start` function implemented in `attester/inbound/inbound.go:43` and `attester/outbound/outbound.go:43`, respectively. The requests are queried in batches of 50 from Router Chain RPC until all available requests are retrieved. Subsequently, the requests are iterated and processed individually. Before confirming a request, i.e., signing the request data with the orchestrator's private key and sending it to Router Chain, the request is validated.

Validation includes checking if the request requires relaying to a destination chain and if the request is ready for execution on the destination chain. Furthermore, special validation is performed for `CROSSTALK` requests in `inbound.go:76` and `outbound.go:77`. Specifically, if the request's workflow type is of type `CROSSTALK` and the validation type is not `IBC_VALIDATION`, the request is filtered out and not processed.

Consequently, only `CROSSTALK` requests originating from a Cosmos chain are processed, and requests from other chain types, e.g., EVM or NEAR, are ignored and not processed, resulting in the inability to relay such `CROSSTALK` requests to a destination chain.

Recommendation

We recommend removing the condition checking if the request's workflow type is `CROSSTALK` and only skipping requests that are supposed to be sent to a Cosmos chain.

Status: Acknowledged

The client states that requests from other chain types, e.g., EVM or NEAR, are ignored and not processed because they will be validated by orchestrator validation.

3. Transaction origin is hardcoded as an empty string when transforming an `iSend` event

Severity: Critical

In `listener/near/gatewayeventprocessor/tranformer.go:76`, the transaction origin is hardcoded as an empty string when creating a cross-chain request. This is incorrect because the transaction origin should record the original sender of the transaction. Consequently, the orchestrator will submit an empty transaction origin when transforming an `iSend` event in the NEAR network via the gateway contract, causing Router Chain to be unable to parse it.

Recommendation

We recommend setting the transaction origin to the original sender of the transaction.

Status: Resolved

4. A potential unconfirmed block is processed in the NEAR event listener

Severity: Critical

In `listener/near/gatewayeventprocessor/querier.go:75` and `listener/near/voyagereventprocessor/querier.go:75`, the `QueryEventsFromGatewayContract` and `QueryEventsFromVoyagerContract` function computes the number of blocks as (`endBlock - startBlock + 1`). This is problematic because one extra block will be processed after the end block, which is unconfirmed.

An unconfirmed block can be replaced if block reorganization happens. If this happens, the orchestrator will process the unconfirmed block's event instead of the confirmed block, leading to incorrectly processed events.

Recommendation

We recommend removing the extra block when computing the number of blocks to process.

Status: Resolved

5. Suboptimal processing of attestations leading to missing attestations

Severity: Major

In `attester/inbound/inbound.go:49`, `attester/outbound/outbound.go:49`, and `attester/valset/valset.go:50`, a loop is used to continuously poll for new `crosschainRequests`, i.e., `CrosschainAckRequest`, `CrosschainRequest`, `valsetRequests`, and to provide attestations. The process involves querying all available requests until the pagination key, `nextKey`, becomes empty. Subsequently, requests are processed and queued for dispatching to Router Chain. After completion, the code enters a 10-second sleep period before resuming polling.

However, the `nextKey` variable is redeclared, leading to the following problems:

1. Increased load on Router Chain: The redeclaration of the `nextKey` variable (`var nextKey []byte`) causes unnecessary and repeated queries from the beginning each time polling resumes. This results in an increased load on Router Chain, which is suboptimal, particularly when all orchestrators run the same code.
2. Risk of missing attestations: The lack of persistence in the `nextKey` value between sleep cycles introduces the potential risk of missing attestations. This occurs because it takes time to query that the request has already been processed.
3. High memory usage for the orchestrator and potentially causing out-of-memory issues due to keeping a large number of requests in memory.

Recommendation

We recommend persisting the pagination cursor, `nextKey`, between batches in the database and retrieving it for the next polling cycle.

Alternatively, consider only querying the eligible requests, i.e., requests ready for execution on the destination chain, from Router Chain by adding appropriate queries to the `attestation` and `crosschain` Router Chain modules.

Status: Resolved

6. Risk of liveness slashing due to inconsistent chain configurations

Severity: Major

In `config/config.go:163`, the current configuration setup permits inconsistent chain configurations, which can potentially lead to slashing for liveness violations. This risk arises from the fact that local configurations can override the configurations fetched from Router Chain. Only configurations loaded from the JSON file are appended to `ChainSpecs`.

Consider a scenario where the orchestrator listens and processes all currently enabled chains, and Router Chain adds a new chain. In this case, the orchestrator should update their local JSON configuration file to process the new chain. Otherwise, the validator can be slashed due to liveness slashing rules.

The inconsistency in chain configurations, where the orchestrator's local configuration differs from that of Router Chain, introduces the risk of violating liveness slashing rules. To address this issue and mitigate the risk of slashing, it is essential to ensure that the orchestrator's configuration aligns with Router Chain's configurations, particularly in cases where attestation responsibilities are concerned. This alignment will help maintain the integrity and security of the system.

Recommendation

We recommend fetching the required configuration for the orchestrator directly from Router Chain via RPC.

Status: Acknowledged

The client states that they are fetching chain config (gateway address, confirmations, etc.) from Router Chain. Whenever a new chain is to be added, the client will inform validators upfront to update their config and provide an RPC endpoint for the new chain. Once the validators get the confirmation, the client will create an on-chain proposal to add the new chain.

7. Unhandled errors in the codebase

Severity: Major

In several instances of the codebase, functions that return an error are not checked nor handled properly. For example, some errors are printed out but not propagated to the caller. The following instances of unhandled errors have been found:

- `attester/inbound/inbound.go:86`
- `attester/outbound/outbound.go:164`
- `attester/valset/valset.go:80`
- `cmd/router-orchestrator/main.go:44, 77, 83, 102, 106, 299, 302`
- `config/config.go:158, 217, 220`
- `health/health.go:41, 47, 51`
- `listener/evm/gatewayeventprocessor/tranformer.go:150, 154, 159`
- `listener/evm/voyagereventprocessor/tranformer.go:19, 26, 60, 67, 101, 108, 138, 145, 175, 182`
- `listener/gatewaylistener.go:44, 59, 76, 81, 83`
- `listener/near/gatewayeventprocessor/querier.go:93, 104, 115, 126, 137`
- `listener/near/gatewayeventprocessor/tranformer.go:26, 86, 90, 94`

- `listener/near/voyagereventprocessor/querier.go:93, 104, 115, 126, 137`
- `listener/near/voyagereventprocessor/tranformer.go:17, 18, 54, 55`
- `listener/tron/gatewayeventprocessor/gatewayeventprocessor.go:62`
- `listener/tron/gatewayeventprocessor/querier.go:118`
- `listener/tron/gatewayeventprocessor/tranformer.go:44, 57, 59, 60, 124, 134-137, 174, 187, 188, 190, 191, 243, 244, 265, 292`
- `listener/tron/initializer/initializer.go:24`
- `listener/tron/initializer/voyagerlistener.go:22`
- `listener/tron/voyagereventprocessor/tranformer.go:17, 18, 23, 55, 56, 92, 147, 148`
- `listener/voyagerlistener.go:43, 57, 75, 80, 82`
- `store/store.go:32-35`
- `types/mqsender.go:59, 70, 96, 106`

Consequently, this would cause silent failures as errors are not raised to notify users.

Recommendation

We recommend handling errors of the above functions by returning the error, panicking, or calling `log.Fatal` to exit the process.

Status: Partially Resolved

8. Incorrect gas price denom used

Severity: Major

In `config/config.go:149`, the gas price is hardcoded to "`3000router`". This is incorrect because Router coins are denominated in "`route`" instead of "`router`". Consequently, incorrect gas fees will be used when dispatching transactions, causing the transactions to fail because "`router`" funds do not exist.

Recommendation

We recommend modifying the gas price denom to "`route`" and making the amount consistent with the SDK implementation.

Status: Resolved

9. Undetected initialization errors and premature `main` termination complicate thread management

Severity: Minor

In `cmd/router-orchestrator/main.go:312-317`, a `case` statement on the `errChn` channel exists to exit the function safely. However, it is important to note that not all errors that might occur during the initialization process are reported to `errChn`.

For instance, in `listener/tron/initializer/initializer.go:24`, an error triggers a panic, which bypasses `errChn`. This situation makes it difficult to terminate all threads gracefully. Consequently, this scenario introduces additional difficulties in ensuring comprehensive initialization validation and robust error handling.

Recommendation

We recommend sending all errors to `errChn` and handling all the errors properly.

Status: Resolved

10. `MqConsumer` lacks updating the status and error code of transactions in the database in case of an error

Severity: Minor

Queued transactions, i.e., `TxqStore` and `NoSequenceTxqStore`, are processed by the `MqConsumer` in two separate goroutines, running the `SubscribeToMsgsFromQueue` and `SubscribeToMsgsFromNoSequenceQueue` functions, respectively. Both functions, implemented in `types/mqconsumer.go`, fetch the unprocessed transactions from the database, parse the binary encoded data, and unpack it into an `sdk.Msg` message, and update the transaction `Status` field in the database to `Picked`. Subsequently, the message is passed to the `Dispatcher` via the `routerMsgChannel` Go channel.

However, if updating the status in the database or unmarshalling the binary encoded data errors, the `Status` is not updated to `Error`, and the error is not logged in the `ErrorCode` field of the `TxqStore` or `NoSequenceTxqStore` entry in the database. Consequently, the error is not visible in the database, resulting in the inability to identify the cause of the error.

Recommendation

We recommend updating the status of the transaction in the database to `Error` and adding an appropriate error code to the corresponding `TxqStore` or `NoSequenceTxqStore` entry in the database by calling the `AddTxqError` and `AddNoSequenceTxqError` functions, respectively.

Status: Resolved

11. Sequential transactions are potentially returned out of order from the database

Severity: Minor

Transactions are sent to Router Chain in either a sequential or non-sequential manner. Sequential messages, including emitted gateway events such as `iSend` or `iReceive`, are sent to Router Chain ordered by their `EventNonce`.

Internally, in `types/mqconsumer.go:52`, the `MqConsumer` goroutine fetches the queued transactions from the `TxqStore` database table by calling the `GetTxqByStatus` function. The `GetTxqByStatus` function, implemented in lines `store/txqStore.go:40-44`, queries the transactions by the specified status, in this case `Unprocessed`.

However, the transactions queried from the database are not guaranteed to be returned in sequential order sorted by the `Id`, a surrogate key consisting of the `chainId`, `contract`, and event `nonce`. While SQLite returns items ordered by the primary row ID, which is, in this case, the primary key `Id`, it is not guaranteed and should not be relied upon as this could result in transactions being sent to Router Chain out of order, leading to reverts.

Recommendation

We recommend adding a sequential integer nonce to `TxqStore` and querying the transactions ordered by nonce from the database.

Status: Resolved

12. Voyager implementation for TRON and NEAR network is incomplete and incorrect

Severity: Minor

In `cmd/router-orchestrator/main.go:184-201` and `236-249`, voyager listeners for TRON and NEAR network are not started if the voyager contract address is not an empty string. This is inconsistent with the EVM network as implemented in lines `221-233`.

Additionally, the TRON and NEAR network voyager implementation is incorrect. Firstly, the `InitializeVoyagerChainListener` function in `listener/near/initializer/voyagerinitialize.go:14` and `listener/tron/initializer/voyagerlistener.go:16` should return `listener.VoyagerListener` instead of `listener.GatewayListener`.

Secondly, the `QueryEventsFromVoyagerContract` function in `listener/near/voyagereventprocessor/querier.go:91-145` uses incorrect case statements to verify the events. For example, the `i_send_event` case statement is checked to parse a `NearVoyagerFundsDepositedEvent`, which is incorrect.

Thirdly, the `ProcessInboundEvents` function in `listener/near/voyagereventprocessor/voyagereventprocessor.go:116` and `129`, as well as in `listener/tron/voyagereventprocessor/voyagereventprocessor.go:75` and `88` incorrectly adds the messages to the gateway contract address instead of the voyager contract address.

Fourthly, the source chain is incorrectly hardcoded as `CHAIN_TYPE_EVM` when it should be `CHAIN_TYPE_NEAR` in lines `listener/near/voyagereventprocessor/tranformer.go:29`, `67`, `98`, `125`, and `155`.

Lastly, the `SortAndTransformInboundEventsByEventNonce` function in `listener/near/voyagereventprocessor/voyagereventprocessor.go:267`, `287`, `307`, and `328` determines the event nonce with incorrect variables. Ideally, they should be using the following variables to determine the event nonce:

- Line `267`: `fundPaidWithMessageEvents[j].Nonce`
- Line `287`: `depositUpdateInfoEvents[k].DepositId`
- Line `307`: `fundDepositedWithMessageEvents[l].DepositId`
- Line `328`: `fundPaidEvents[m].DepositId`

Consequently, the TRON and NEAR network voyager listener will not be started, causing inbound events to be unprocessed.

Recommendation

We recommend completing the TRON and NEAR network voyager listener and correcting the above-mentioned errors.

Status: Resolved

13. Retrying failed Router Chain queries without delay between retries may overload the RPC node

Severity: Minor

Inbound, outbound, and validator set requests are processed by orchestrators within the `Start` function in `attester/inbound/inbound.go:43`, `attester/outbound/outbound.go:43`, and `attester/valset/valset.go:43`. The requests are queried from Router Chain RPC with the `sdk-go` client by using pagination with a cursor (`Key`) and a limit of 50.

Once all requests are retrieved, specifically when the next pagination key `NextKey` equals zero, indicating that no further requests are available, the requests are processed and confirmed.

However, if attempting to retrieve the requests via Router Chain client errors, or the response is `nil`, the query is immediately and indefinitely retried without adding sufficient delay between retries. This can lead to many requests being sent to Router Chain, causing additional load on the RPC node, possibly leading to even further errors.

Recommendation

We recommend adding a delay between retries, e.g., using an exponential backoff to ensure sufficient time for the RPC node to recover before retrying.

Status: Resolved

14. The NEAR network used for querying events is hard-coded to the testnet

Severity: Minor

Events emitted from the gateway and voyager contracts deployed on the NEAR blockchain are queried via the Lake Go framework. The `Streamer` function, used to initialize the message channel for the received events, is configured with the `DefaultLakeConfigBuilder` in both `listener/near/voyagereventprocessor/querier.go:78` and `listener/near/gatewayeventprocessor/querier.go:78`.

However, NEAR's testnet is always used by calling the `Testnet()` function on the lake config builder instead of determining the network based on a configuration value.

Recommendation

We recommend configuring the used network instead of hardcoding the network to the testnet.

Status: Resolved

15. Dispatcher lacks updating the status of transactions in the database

Severity: Minor

The `MqConsumer` picks up unprocessed transactions stored in the database, runs in a goroutine, and forwards them to the `Dispatcher` in a separate goroutine via the `routerMsgChannel` Go channel. In `dispatcher/dispatcher.go:45`, the dispatcher receives a transaction, broadcasts it to Router Chain, and waits until the transaction is included in the block.

However, the status of the transaction after the broadcast is not updated in the database, even though the `Status` field in the `TxqStore` struct, representing the status of the transaction such as `Unprocessed`, `Picked`, `Dispatched`, `Error`, and `Completed`, suggests that the status should be updated.

Moreover, the transaction hash is not updated and stored in the `TxHash` field of the `TxqStore` struct.

Consequently, having inaccurate transaction statuses and missing transaction hashes in the database makes it difficult to track the status of transactions and to identify potential causes of errors.

Recommendation

We recommend updating the transaction status in the database, adding the transaction hash, and, in case of an error, adding the error code to the corresponding `TxqStore` or `NoSequenceTxqStore` entry in the database.

Status: Resolved

16. Logger is hardcoded as debug level

Severity: Minor

In `logger/logger.go:82`, the `InitLogger` function does not use the provided `logLevel` argument to set the logging level. Instead, it sets the logging level as `log.DebugLevel`, which is incorrect.

Consequently, the attesters in `cmd/router-orchestrator/main.go:260`, `272`, and `284` will not log as `log.InfoLevel` and the caller-provided `config.VerbosityFlag.Name` in `cmd/router-orchestrator/main.go:99` will not be used.

Recommendation

We recommend using the provided `logLevel` argument to set the logging level.

Status: Resolved

17. Unprotected Cosmos keys in configuration

Severity: Minor

When initializing Router Chain client in `cmd/router-orchestrator/main.go:137`, the passphrase protection for Cosmos keys is absent. These keys are loaded from an

unencrypted JSON configuration file. This exposes the Cosmos private keys, making them vulnerable to unauthorized access and compromise in the event of a security breach.

Recommendation

We recommend implementing passphrase protection, like in the case of Ethereum keys in lines `145-151`.

Status: Resolved

18. Insecure HTTP and GRPC exposing sensitive data

Severity: Informational

In `cmd/router-orchestrator/main.go:83`, there is a configuration for an HTTP server to handle a health check endpoint. However, the absence of HTTPS implementation for this health check endpoint introduces a security risk. This is because sensitive information, such as health status, could be transmitted in plaintext and potentially manipulated. Consequently, malicious actors may have the opportunity to intercept this data, potentially enabling them to assess the overall health of orchestrators, and although the extent of the threat remains undetermined, it could facilitate a potential attack.

Similarly, in `listener/tron/initializer/initializer.go:23`, GRPC is configured insecurely.

Recommendation

We recommend implementing HTTPS-secure GRPC.

Status: Resolved

19. Processed inbound requests are stored in the `ProcessedOutboundBatch` table intended for outbound requests

Severity: Informational

Inbound requests are attested in batches by an orchestrator in `attester/inbound/inbound.go`. Once a request within a batch is successfully processed and added to the message producer, the request is stored in the local SQLite database in line `183`.

However, the inbound request is stored in the `ProcessedOutboundBatch` table, which, as its name suggests, is intended to store outbound requests. While this does not lead to

collisions between storing inbound and outbound requests, it is semantically incorrect and confusing.

Recommendation

We recommend storing inbound requests in a separate table, e.g., `ProcessedInboundBatch`. Alternatively, consider renaming the `ProcessedOutboundBatch` table to a more neutral name, e.g., `ProcessedBatch`, indicating that it stores both inbound and outbound requests.

Status: Acknowledged

The client states that they agree that it is semantically incorrect, and they will rename it to `ProcessedBatch`.

20. Health check does not process voyager listeners

Severity: Informational

In `cmd/router-orchestrator/main.go:64`, the health checker only processes the gateway listeners when checking the health. On the other hand, voyager listeners started in line `228` are not included when checking the health. Consequently, the health endpoint will not reflect the status of the voyager listeners.

Recommendation

We recommend including the voyager listener in the health endpoint.

Status: Resolved

21. Incorrect debug logging

Severity: Informational

The following code locations contain incorrect debug loggings:

- `listener/tron/voyagereventprocessor/tranformer.go:172` and `listener/evm/voyagereventprocessor/tranformer.go:205` logs `MsgFundsPaid` while processing `MsgDepositInfoUpdated`.
- `listener/tron/voyagereventprocessor/voyagereventprocessor.go:167` logs "`setMetadataEvents`": `depositInfoUpdateEvents`, which should be "`depositInfoUpdateEvents`": `depositInfoUpdateEvents`.
- `listener/tron/voyagereventprocessor/voyagereventprocessor.go:357` emits the logging as "gateway events query", which should be "voyager events query".

Consequently, the debug logging will reflect the incorrect action.

Recommendation

We recommend modifying the logging to reflect the correct action.

Status: Resolved

22. Errors are incorrectly displayed due to incorrect format printing

Severity: Informational

The following code locations contain incorrect format printing:

- `listener/evm/initializer/voyagerinitialize.go:39`
- `listener/tron/gatewayeventprocessor/querier.go:36`
- `listener/tron/voyagereventprocessor/querier.go:30`

Since the called function in those locations does not support format printing, this would cause the error to be printed as "`%s`" instead of the intended error.

Recommendation

We recommend modifying the code locations mentioned above to support format printing.

Status: Resolved

23. Newly added database records are logged as errors

Severity: Informational

In `store/processedbatch.go:51`, the logger logs an error when creating a new `ProcessedOutboundBatch` record to the database. Logging the outbound batch as an error is misleading, as errors should reflect unintended or severe issues.

Recommendation

We recommend logging the record as an `Info` or `Debug` log entry instead of an error.

Status: Resolved

24. Same log color is used for different logging levels

Severity: Informational

In `logger/logger.go:50` and `54`, the level color is set to 31 for two different logging levels. Specifically, the debug and trace levels use the same level color as error, fatal, and panic levels. This is problematic because using the same color is confusing and makes it harder for the reader to differentiate the logging levels.

Recommendation

We recommend modifying the logging color to utilize different log-level colors.

Status: Resolved

25. Validator set updated event is not logged along with other fields

Severity: Informational

In `listener/near/gatewayeventprocessor/gatewayeventprocessor.go:73`, the `FetchAndTransformInboundEvents` function logs all `iAck`, `iReceive`, `iSend`, and `setDappMetadata` events when transforming the events into messages. However, the validator set updates being part of the events are not logged accordingly.

Recommendation

We recommend logging the `valsetUpdatedEvents` for consistency and verbosity.

Status: Resolved

26. `QueryGatewayValsetUpdatedEvents` computational complexity can be reduced

Severity: Informational

In `listener/tron/gatewayeventprocessor/querier.go:131`, the `QueryGatewayValsetUpdatedEvents` function iterates over all topics and sets the data if the topic key matches the event hash. If data is already set, the loop will continue until it finishes iterating all the topics. This consumes unnecessary computation power as there is no need to continue iterating once the data is found.

Recommendation

We recommend adding a `break` statement to reduce computation complexity after the data is set.

Status: Resolved

27. Unused `path` variable during database initialization

Severity: Informational

In `store/store.go:17`, the `InitialiseDB` function accepts a `path` parameter, but the parameter is not utilized within the function. Consequently, the function connects only to the "`orchestrator.db`" database instead of the expected `cfg.GlobalConfig.DbPath`.

Recommendation

We recommend using the `path` while connecting to the database.

Status: Resolved