Audit Report

Router Osmosis and Bitcoin Integration: Orchestrator

DRAFT – DO NOT PUBLISH

v0.2

May 15, 2024

Table of Contents

License

Disclaimer

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 Decimal FZC to perform a security audit of Router Osmosis and Bitcoin Integration: Orchestrator.

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	`32687e24d0ce69e46c0429d146e045ab60f1561e`
Scope	The scope was restricted to the changes compared to commit `fee27164a82be6af1b366c1045ac09cc70b33827`.

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	Medium	-
Code readability and clarity	Low-Medium	There were several `TODO` comments and unused functions and variables throughout the codebase.
Level of documentation	Medium	Documentation is available at https://docs.routerprotocol.com/validators/orchestrators.
Test coverage	-	Test coverage cannot be determined due to compile errors.

Summary of Findings

No	Description	Severity	Status
1	Funds are sent to the incorrect chain	Critical	Resolved
2	Prematurely calling `UpdateLastProcessedBlock` may cause missed transactions and orchestrator slashing	Critical	Resolved
3	Cross-chain requests may fail due to insufficient gas limit	Major	Acknowledged
4	`fetchHistoricTransactions` only processes one block, causing syncing delays for Bitcoin	Major	Resolved
5	Errors are ignored when decoding the depositor's address	Minor	Resolved
6	RPC call is executed when there are no new blocks to process	Informational	Resolved
7	Unhandled error in the defer function	Informational	Pending
8	Requests are not retried for `StatusTooManyRequests` errors	Informational	Pending
9	`Println` statements are used instead of log statements	Informational	Pending
10	Unnecessary loop implemented to reorder transactions	Informational	Pending
11	Usage of magic numbers decreases maintainability	Informational	Pending

Detailed Findings

1. Funds are sent to the incorrect chain

Severity: Critical

In `listener/bitcoin/gatewayeventprocessor/transformer.go:42`, the destination chain is determined from the hardcoded "*router_9625-1*" value, representing the Ethereum chain. This is incorrect because the destination chain should be determined by the data extracted in `listener/bitcoin/gatewayeventprocessor/querier.go:262`.

Consequently, funds are sent to the incorrect destination chain, causing depositors to lose funds as the recipient may not exist on the Ethereum chain.

This issue is also present in `listener/bitcoin/gatewayeventprocessor/transformer.go:94`, as the cross-chain request is always dispatched to Ethereum.

Recommendation

We recommend using `data.DestChainId` to determine the destination chain.

Status: Resolved

2. Prematurely calling `UpdateLastProcessedBlock` may cause missed transactions and orchestrator slashing

Severity: Critical

In `listener/bitcoin/gatewayeventprocessor/transformer.go:141`, the `TransformGatewayISendEvent` function updates the last processed block as soon as a transaction in the block is processed. This is problematic because there may be a possibility that not all transactions inside the block are fully processed in the event of a sudden system restart or recovery.

Consequently, the orchestrator may incorrectly interpret that all transactions in the block have been processed, causing missed transactions and the orchestrator slashed in Router Chain as other orchestrators attest to the transaction.

This issue is also present in `listener/bitcoin/voyagereventprocessor/transformer.go:176`.

Recommendation

We recommend updating the last processed block only if all the transactions inside the block are fully processed.

Status: Resolved

3. Cross-chain requests may fail due to insufficient gas limit

Severity: Major

In `listener/bitcoin/gatewayeventprocessor/transformer.go:73`, the `TransformGatewayISendEvent` function hardcodes the `requestMetadata` configurations, such as `destGasLimit` and `destGasPrice`, that enforce the gas limit when dispatching cross-chain requests. This is problematic because these values may spike dramatically on chains like Ethereum (e.g., due to high market demand), causing the request to fail due to an out-of-gas error, negatively impacting the protocol's performance.

Recommendation

We recommend retrieving the request metadata fields from extracted data.

Status: Acknowledged

4. `fetchHistoricTransactions` only processes one block, causing syncing delays for Bitcoin

Severity: Major

In `listener/bitcoin/gatewayeventprocessor/gatewayeventprocessor.go:66`, the `ProcessInboundEvents` function computes the blocks to process as `startBlock` and `endBlock` parameters and feeds them to the `fetchHistoricTransactions` function in line `85`. Internally, this function will call `fetchTransactionsViaAPI` to retrieve the transaction data in `listener/bitcoin/gatewayeventprocessor/querier.go:92`.

However, the `fetchTransactionsViaAPI` function ignores the `endBlock` parameter and only processes one block after the `startBlock`, as seen in `listener/bitcoin/gatewayeventprocessor/querier.go:117-121`. This is problematic because the orchestrator will only process one instead of all the confirmed blocks, preventing the system from successfully syncing with the current confirmed block height due to the 10-minute delay in `listener/bitcoin/gatewayeventprocessor/gatewayeventprocessor.go:91`.

Consequently, there will always be a delay when processing cross-chain messages from Bitcoin, negatively affecting the protocol's functionality.

This issue is also present in `listener/bitcoin/voyagereventprocessor/querier.go:85-90`.

Recommendation

We recommend modifying the `fetchTransactionsViaAPI` function in `listener/bitcoin/gatewayeventprocessor/querier.go:92` and `listener/bitcoin/voyagereventprocessor/querier.go:90` to compute the blocks based on the `endBlock` parameter.

Status: Resolved

5. Errors are ignored when decoding the depositor's address

Severity: Minor

In `listener/bitcoin/voyagereventprocessor/querier.go:252`, the `extractDataFromAPITransaction` function ignores the error when decoding the depositor address. This would cause the depositor address to be encoded as an empty string in line `277`, which may cause the cross-chain request to fail if the recipient chain attempts to decode the depositor's address.

Recommendation

We recommend returning an error message if an error occurs when decoding the depositor's address.

Status: Resolved

6. RPC call is executed when there are no new blocks to process

Severity: Informational

In `listener/bitcoin/gatewayeventprocessor/gatewayeventprocessor.go:143-145`, the `CalculateStartAndEndBlocksFromAPI` function returns an error when there are no confirmed blocks to process. Ideally, the orchestrator should wait until new blocks are mined before continuing to process blocks.

However, the error will be caught in lines `66-71`, causing the `CalculateStartAndEndBlocksFromRPC` function to query for new blocks and return the same error in lines `111-113`. This approach consumes unnecessary computation power for the orchestrator.

This issue is also present in `listener/bitcoin/voyagereventprocessor/voyagereventprocessor.go:60-65`.

Recommendation

We recommend only calling the `CalculateStartAndEndBlocksFromRPC` function if the error message does not read “no new confirmed blocks to process.”

Status: Resolved

7. Unhandled error in the defer function

Severity: Informational

In `listener/bitcoin/voyagereventprocessor/querier.go:166`, errors in the defer function are not handled.

Recommendation

We recommend returning the error in the defer statement and returning it like any other error that occurs in the function.

Status: Pending

8. Requests are not retried for `StatusTooManyRequests` errors

Severity: Informational

In `listener/bitcoin/voyagereventprocessor/querier.go:171`, a log statement suggests retrying the requests after some time due to the exceeded rate limit. However, this is not implemented.

Recommendation

We recommend implementing a delay mechanism before retrying the request when the rate limit is exceeded.

Status: Pending

9. `Println` statements are used instead of log statements

Severity: Informational

The codebase utilizes `Println` statements instead of log statements in multiple instances. For example, one can be found in `listener/bitcoin/voyagerevenprocessor/voyagereventprocessor.go:56`.

Recommendation

We recommend removing the `Println` statements if unnecessary or replacing them with logging statements.

Status: Pending

10. Unnecessary loop implemented to reorder transactions

Severity: Informational

In `listener/bitcoin/voyagereventprocessor/querier.go:132`, the transactions are looped in reverse and added into a slice, which is iterated subsequently.

However, this consumes unnecessary computation power and memory since the transactions can be iterated in reverse directly.

Recommendation

We recommend removing the loop and iterating the `apiResponse.TXs` in reverse directly.

Status: Pending

11. Usage of magic numbers decreases maintainability

Severity: Informational

In `utils/utils.go:123`, the magic number for "*router*" is used. Using such "magic numbers" goes against best practices as they reduce code readability and maintenance as developers cannot easily understand their use and may make inconsistent changes across the codebase.

Recommendation

We recommend replacing the magic number with the `RouterChainPrefix` constant defined in line `134`.

Status: Pending