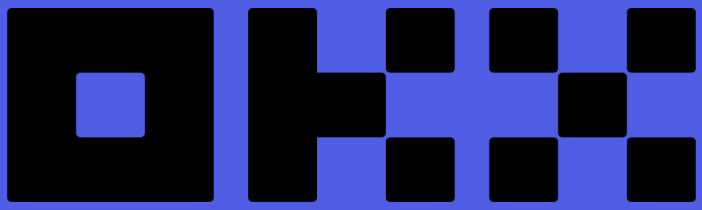
OpenZeppelin | security

OKX X Layer Reth Audit



January 29, 2026

Table of Contents

Summary

Type	Blockchain Infrastructure
Timeline	From 2026-01-05 To 2026-01-21
Languages	Rust

Total Issues	18 (18 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	3 (3 resolved)
Low Severity Issues	11 (11 resolved)
Notes & Additional Information	4 (4 resolved)

Scope

OpenZeppelin performed an audit of the okx/xlayer-reth repository at commit efa6523.

In scope were the following files:

```
├── bin
│   ├── node
│   │   └── src
│   │       ├── args_xlayer.rs
│   │       └── main.rs
│   └── tools
│       ├── export.rs
│       ├── import.rs
│       └── main.rs
└── crates
    └── chainspec/src
    │   ├── lib.rs
    │   ├── parser.rs
    │   ├── xlayer_mainnet.rs
    │   └── xlayer_testnet.rs
    └── flashblocks/src
    │   ├── handler.rs
    │   ├── lib.rs
    │   ├── pubsub.rs
    │   └── subscription.rs
    └── legacy-rpc/src
    │   ├── get_logs.rs
    │   ├── layer.rs
    │   ├── lib.rs
    │   └── service.rs
    └── rpc/src
    │   ├── lib.rs
    │   ├── pre_exec_ext_xlayer.rs
    │   ├── pre_exec_types.rs
    │   └── xlayer_ext.rs
    └── version
        ├── build.rs
        └── src
            └── lib.rs
```

System Overview

The audited codebase is a customized implementation of the Reth Ethereum client tailored for OKX's X Layer network. The codebase extends Reth's core functionality with X Layer-specific features for transaction simulation, RPC endpoints, blockchain specification, and real-time event streaming. The system is designed to provide pre-execution transaction analysis, legacy RPC compatibility, custom chainspec definitions, and flash block subscriptions.

RPC Module

The RPC layer provides both standard Ethereum JSON-RPC endpoints and extensions for transaction analysis and gas pricing. The system implements the `eth_transactionPreExec` RPC method, which simulates transaction execution and returns detailed traces including inner calls, gas metrics, and state changes. This module converts Geth call tracer output into X Layer's custom `PreExecInnerTx` format, recursively processing all nested calls to build a hierarchical execution tree.

Additionally, the `eth_minGasPrice` endpoint returns the minimum gas price by combining the base fee with a configured suggested fee. The codebase registers these custom RPC handlers while conditionally routing incoming requests to the legacy-rpc to provide backward-compatible RPC endpoints.

Node Module

The node binary bootstraps the X Layer client with custom command-line arguments. The node runs in full sync mode with Ethereum-compatible block validation and consensus while adding support for publishing or subscribing to flashblocks.

Chainspec Module

The chainspec module defines network parameters for X Layer mainnet and testnet. Specifications are loaded from genesis configuration files and include consensus rules, fork activation heights, and token parameters.

Flashblocks Module

The flashblocks module implements a WebSocket-based pub/sub system for real-time block and transaction event streaming. The handler receives flashblocks from the consensus layer and publishes them via WebSocket. The subscription module provides a JSON-RPC pub/sub interface that allows clients to subscribe to flashblocks with configurable filters for headers, transactions, and receipts.

Tools Module

The tools binary provides import and export utilities for blockchain data, similar to go-ethereum's import/export commands. The `export` subcommand reads blocks from the database, encodes them in RLP format, and writes them to files with optional `gzip` compression. Export operations support configurable block ranges (start/end blocks) and batch processing for efficiency, with read-only database access.

The `import` subcommand reverses this process, reading RLP-encoded blocks from files, validating them against the chain specification, and importing them into the database in batches. The import process skips the genesis block and already imported blocks to avoid duplicate processing.

Trust Assumptions

Throughout the in-scope codebase, the following trust assumptions were identified:

- The node operator has exclusive control over the execution environment, with no untrusted party able to inject or modify the `XLAYER_MAINNET_GENESIS` or `XLAYER_TESTNET_GENESIS` environment variables that override built-in genesis configurations.
- The operator-configured legacy endpoint in the log merging implementation returns unique and correctly ordered results, as the system only performs block number sorting without deduplication logic.
- Gzip-compressed chain import files are trusted inputs verified by the operator, as untrusted compressed files could expand to arbitrary sizes and cause resource exhaustion.

- The `eth_transactionPreExec` endpoint is not publicly exposed, as it allows per-transaction gas consumption up to `MAX_GAS_LIMIT` without batch size or total gas limits, which could lead to resource exhaustion if accessible to untrusted callers.

Medium Severity

M-01 High Complexity in Flashblocks Address Filtering Enables DoS

The `is_address_in_transaction` function performs `O(L × A)` operations per transaction by calling `Vec::contains()`: `O(A)` for each log address, where `L` is the number of logs and `A` is the size of `subscribe_addresses`. Since `subscribe_addresses` is user-supplied with no size limit, an attacker can submit a subscription with lots of addresses, causing the filtering in `collect_transactions` to consume excessive CPU cycles per block. The total complexity will be `O(T × L × A)`, where `T` is the number of transactions in the block. For example, a block with 500 transactions, 20 logs each, and a malicious filter containing 2,000 addresses would require 20 million comparison operations.

Consider converting `subscribe_addresses` to a `HashSet<Address>` to achieve O(1) lookups and imposing a maximum size limit on the address monitoring list.

Update: *Resolved in pull request #88 at commit fea9b86.*

M-02 Incorrect Per-Transaction Gas Calculation in Flashblocks Receipt Enrichment

The `enrich_receipt` function passes `cumulative_gas_used()` directly to `ConvertReceiptInput.gas_used`, which expects the gas consumed by a single transaction, not the cumulative total. As a result, flashblocks subscribers receive receipts with inflated `gasUsed` values, where each transaction's reported gas equals the sum of all preceding transactions plus its own.

Consider computing per-transaction gas by subtracting the previous transaction's cumulative gas.

Update: *Resolved in pull request #91 at commit f1fac63.*

M-03 Blocking Contract Creations in Pre-Execution RPC

The `validate_pre_args` function in the pre-execution extension includes a check that requires the `to` field to be present in all transaction requests. If the `to` address is missing (which is the standard way to denote a contract creation transaction), the function returns a `PreExecError::check_args`. This behavior prevents the `eth_transactionPreExec` API from being used to simulate contract deployments. In contrast, upstream Reth simulation RPCs allow the `to` field to be omitted for contract creation transactions.

Consider revising the mandatory check for the `to` field in `validate_pre_args` to treat it as a contract creation transaction if it is missing.

Update: *Resolved in pull request #120 at commit 84472c5 and in pull request #124 at commit 9889b57. The `eth_transactionPreExec` RPC API endpoint and all associated code have been removed.*

Low Severity

L-01 Legacy RPC Router Can Panic on Requests Without Parameters

The legacy routing middleware assumes request parameters are always present and unconditionally unwraps them in `handle_block_param_methods`, and similarly in `handle_eth_get_logs`. When legacy routing is enabled, a user can send an RPC request without parameters, causing `req.params().as_str()` to be `None` and the unwrap to panic.

Consider replacing all `req.params().as_str().unwrap()` occurrences in the legacy router with proper parameter validation and returning an error response.

Update: *Resolved in pull request #121 at commit 2761dac.*

L-02 Incorrect `--batch-size` Default Value in Documentation

The tools README states that `--batch-size` defaults to 1000, but the actual CLI default is 100000 in the argument definition. This discrepancy can mislead operators into running exports with a much larger in-memory working set than expected, increasing the risk of process out-of-memory conditions.

Consider aligning the documentation and code by lowering the default to a safer value.

Update: *Resolved in pull request #123 at commit 55511cd.*

L-03 Missing Range Validation When `start_block > end_block`

The `xlayer-reth-tools export` command computes `total_blocks` using unsigned arithmetic without validating whether `total_blocks = end_block - start_block + 1` underflows. If a user supplies a `--start-block` value greater than `--end-block`, this subtraction underflows: in debug builds, it can panic, and in release builds, it wraps to a huge number, after which the export loop is skipped because the loop condition `current_block <= end_block` is `false`.

Consider adding validation to ensure that `end_block` is greater than `start_block`.

Update: *Resolved in pull request #116 at commit 2980575.*

L-04 Incomplete State Diff Output Omits Self-Destructed and Newly Created Accounts

The `process_tracer_results` function constructs `state_diff` by only iterating over `diff.post` and requiring addresses to exist in both the `pre` and `post` states. This causes two categories of accounts to be omitted from the output: (1) self-destructed accounts, which exist in `diff.pre` but not `diff.post`, and (2) newly created accounts, which exist in `diff.post` but not `diff.pre`. Moreover, the current implementation only captures balance changes, ignoring nonce, storage, and code modifications. As such, users relying on `state_diff` for transaction analysis may therefore receive incomplete information.

Consider iterating over the addresses from both `diff.pre` and `diff.post` to capture all affected accounts, and including additional state fields (nonce, storage, and code) where they differ between the `pre` and `post` states.

Update: *Resolved in pull request #120 at commit 84472c5 and in pull request #124 at commit 9889b57. The `eth_transactionPreExec` RPC API endpoint and all associated code have been removed.*

L-05 Missing Legacy Fallback on Local Hash Resolution Failure

The `handle_block_param_methods` function does not trigger a legacy fallback when the local block hash resolution via `call_eth_get_block_by_hash` encounters an error. Instead, it only logs the diagnostic and proceeds to execute the original request against the local node.

Consider updating the `Err(err)` branch to invoke `forward_to_legacy`.

Update: *Resolved in pull request #110 at commit 71acb0e.*

L-06 Batch RPC Requests Bypass Legacy Routing

The `batch` method in the `LegacyRpcRouterService` implementation forwards all batch requests directly to the `inner` service without applying any routing logic. This causes historical data queries contained within a batch to bypass the legacy endpoint, leading to empty results or errors when the local node does not yet possess the requested historical state.

Consider implementing a batch handler that iterates over individual requests within the batch to apply the same routing logic used in the `call` method.

Update: *Resolved in pull request #117 at commit 5f4d3da.*

L-07 Panic on Malformed Block Identifiers in Legacy RPC Routing

The `parse_block_param` and `is_block_hash` functions perform insufficient validation of block identifiers, allowing any 66-character string that starts with "0x". This allows an attacker

to provide a 66-character string containing JSON-reserved characters (e.g., `"`) which is subsequently interpolated into a JSON template in `call_eth_get_block_by_hash` via the `params_str` variable. When this malformed payload is passed to `RawValue::from_string`, the JSON parser fails, triggering a panic via the `.expect("Valid JSON params")` call.

Consider replacing these instances with the already defined `is_valid_blockhash` function which validates that all characters following the `0x` prefix are valid hexadecimal digits, while renaming it to `is_valid_32_bytes_string`. In addition, consider replacing the `.expect()` call with proper error propagation.

Update: *Resolved in pull request #111 at commit ed04075 and in pull request #121 at commit 2761dac.*

L-08 Code Refactoring

Throughout the codebase, multiple instances of code that could benefit from refactoring were identified:

- The node-builder logic in lines 87 to 201 is partially duplicated across the flashblocks-enabled and flashblocks-disabled branches, differing only in payload builder and flashblocks subscription setup. Consider extracting the common setup into a shared function to reduce duplication.
- There is an unnecessary clone in the `flashblocks::subscription` module.
- The transaction hash cache check in line 287 of the `flashblocks::subscription` module occurs after creating `EnrichmentContext`, wasting allocations when the transaction is already cached. Consider placing the cache check before the `ctx` declaration to return early.
- The `PreExecInnerTx` struct has redundant fields (`value` and `value_wei`) that both store the same Wei-denominated value. Consider removing one field, as they contain identical data.
- The `AccountStateDiff` and `BalanceChange` structs are unused and can be removed.
- In `xlayer_ext.rs`, consider replacing the instances of hardcoded `-32603` error values with `jsonrpsee::types::error::INTERNAL_ERROR_CODE`.

Consider implementing the above-listed refactoring suggestions to improve the clarity and maintainability of the codebase.

Update: *Resolved.*

- *Node builder code duplication resolved in pull request #112 at commit df2e4ce.*
- *Unnecessary clone in subscription resolved in pull request #113 at commit 811be7b.*
- *Cache check after `EnrichmentContext` creation resolved in pull request #92 at commit f96c45e and pull request #126 at commit 905e516.*
- *Redundant `PreExecInnerTx` fields resolved in pull request #120 at commit 0cf1787.*
- *Unused `AccountStateDiff` and `BalanceChange` structs resolved in pull request #120 at commit 0cf1787.*
- *Hardcoded `-32603` error code resolved in pull request #125 at commit a3e4168.*

L-09 Improper Nonce Validation in Pre-execution Validation

The `validate_pre_args` function validates transaction nonces during pre-execution simulation by comparing the current transaction against the immediately previous transaction. When multiple transactions from the same sender are included in a batch, the validation only checks whether `msg_nonce <= pn`, where `pn` is the previous transaction's nonce. This approach fails to detect nonce ordering violations when transactions from the same address are non-consecutive within the batch.

Moreover, the `run_pre_exec_in_db` function uses a single `prev` request to enforce a per-sender monotonic nonce rule within `validate_pre_args`, but still sets `prev` even when a transaction is not executed/committed. This means that subsequent transactions from the same sender may be rejected based on the nonce of a prior failed request, leading to incorrect simulation results.

In addition, the error message reports "nonce decreases", whereas the actual violation is that the nonce did not increase from the previous transaction. Furthermore, the overflow check uses `st_nonce.checked_add(1).is_none()` to detect maximum nonce values, which incorrectly validates the edge case where both `st_nonce` and `msg_nonce` equal `u64::MAX`, allowing the simulation to proceed and fail later during execution instead of rejecting it during validation.

Consider maintaining a per-sender nonce tracker throughout the batch validation process, only updating that tracking after a successful execution commit. Doing so will ensure that all

transactions from the same sender have strictly increasing nonces regardless of their position in the batch. Apart from this, consider changing the state nonce comparison to `st_nonce != msg_nonce` for immediate rejection of non-sequential nonces. Consider also updating the error message to reflect that nonces must increase sequentially. Finally, for the maximum nonce case, consider checking that `msg_nonce == u64::MAX` first before the overflow check, rejecting the edge case during validation instead of allowing it to fail during execution.

Update: *Resolved in pull request #120 at commit 84472c5 and pull request #124 at commit 9889b57. The `eth_transactionPreExec` RPC API endpoint and all associated code have been removed.*

L-10 Incomplete Cleanup Leaves Empty Export Files on Failure

The `export` command creates the output file before executing any export work, then returns early on errors (e.g., failed block reads, interrupts, or invalid ranges) without removing or rolling back the newly created file. When the export loop is skipped or aborted, it leaves behind an empty or partial file. At the same time, the command still appears to have produced an artifact, increasing the likelihood that operators will treat incomplete exports as valid backups.

Consider ensuring that any early-return path removes the partially created file so that failed exports do not leave behind misleading artifacts.

Update: *Resolved in pull request #119 at commit 4d315d9.*

L-11 Missing Caller Context in DELEGATECALL Trace Output

The `convert_call_frame_recursive` function does not populate the `trace_address` field for `DELEGATECALL` frames. While `CALLCODE` frames receive special handling to set `code_address`, `DELEGATECALL` frames are left with an empty `trace_address`. This omits the `msg.sender` context that the delegatecalled code would observe during execution, which differs from the frame's `from` field. As a result, users relying on trace output for debugging or analysis would receive incomplete information about the execution context.

Consider tracking the caller address through the call chain recursion and populating `trace_address` for `DELEGATECALL` frames.

Update: *Resolved in pull request #120 at commit 84472c5 and in pull request #124 at commit 9889b57. The* `eth_transactionPreExec` *RPC API endpoint and all associated code have been removed.*

Notes & Additional Information

N-01 Possible Stack Overflow in `convert_call_frame_recursive` via Deep Call Frame Trees

The `eth_transactionPreExec` request can crash the RPC node process because it converts tracer call frames using unbounded recursion in `helpers::convert_call_frame_recursive`. A transaction that produces a deep call tree can drive the recursion to overflow the stack on RPC worker thread stack sizes. For the current configuration, the deepest call is ~500 frames under a 30M gas cap due to the 63/64 forwarding rule, even though the hard EVM limit is 1024. While this will not trigger a stack overflow for common Linux thread stacks (8MiB), it is still a potential DoS vector.

Consider rewriting `helpers::convert_call_frame_recursive` to use an iterative traversal approach or enforcing a maximum call-frame depth or size.

Update: *Resolved in pull request #120 at commit 84472c5 and in pull request #124 at commit 9889b57. The* `eth_transactionPreExec` *RPC API endpoint and all associated code have been removed.*

N-02 Missing Receipts Cause Silent Transaction Skipping

In the `collect_transactions` function, if a receipt is not found for a transaction, the transaction is silently skipped without any warning. While this scenario should be rare since receipts and transactions should always align, subsequent data inconsistencies could cause subscribers to miss transactions without any indication that an issue has occurred.

Consider adding a warning log when a receipt is unexpectedly missing.

Update: *Resolved in pull request #92 at commit f96c45e and merged into the main branch at commit f1fac63.*

N-03 Missing Empty Result Validation in `eth_getLogs` BlockHash Routing

The `handle_eth_get_logs` function does not check for empty results when `res.is_success()` is true by block hash. This deviates from the logic in `handle_try_local_then_legacy`, which falls back when a result is successful but empty.

Consider updating the routing logic so when the local response is successful but empty, the service proceeds to `forward_to_legacy` to ensure consistency with other hash-based routing methods.

Update: *Resolved in pull request #118 at commit 8b7f899.*

N-04 Incomplete Documentation

The `subscribe_addresses` filter in flashblocks subscriptions only matches the transactions wherein the specified address appears as the sender (`from`), recipient (`to`), or log emitter, as implemented in `is_address_in_transaction`. However, the current implementation does not match addresses involved in internal transactions, newly deployed contract addresses, or addresses appearing in indexed log topic parameters.

Without explicit documentation of the stated filtering boundaries, API consumers expecting comprehensive address monitoring may fail to capture all relevant on-chain activity involving their monitored addresses. In addition, the call_type field documentation uses "etc." to represent additional call types beyond the explicitly listed opcodes (`call`, `staticcall`, `delegatecall`), leaving the full set of possible values ambiguous.

Consider documenting the precise scope of the `subscribe_addresses` filter by explicitly stating which transaction components are included and excluded from address matching. Additionally, consider replacing "etc." with "callcode" to provide a complete enumeration of all supported call types in the API documentation.

Update: *Resolved in pull request #113 at commit 7941820 and in pull request #127 at commit a8ded6f. The pre-exec-related code has been removed in pull request #120 at commit 0cf1787.*

Conclusion

The audited codebase is a customized implementation of the Reth Ethereum client tailored for the X Layer network. It extends Reth's core functionality with custom RPC endpoints, legacy RPC routing, flashblocks streaming, and blockchain import/export utilities.

No critical- or high-severity issues were identified. Three medium-severity findings were reported: a DoS vector in flashblocks address filtering due to unbounded user-supplied filter lists, incorrect gas calculation in flashblocks receipt enrichment, and the pre-execution RPC blocking contract creation transactions. Several low-severity issues were also identified, primarily related to error handling in the legacy RPC routing layer, including potential panics on malformed requests and incomplete validation logic.

Overall, the codebase was found to be well-structured and modular, with a clear separation between different functionalities. The implementation includes comprehensive unit tests and helpful documentation comments. The legacy RPC routing middleware could benefit from more defensive error handling to better manage edge cases and malformed inputs.

The OKX team is appreciated for their excellent responsiveness throughout the audit process and for providing helpful context regarding the system's design and intended use cases.

Appendix

Issue Classification

OpenZeppelin classifies smart contract vulnerabilities on a 5-level scale:

- Critical
- High
- Medium
- Low
- Note/Information

Critical Severity

This classification is applied when the issue's impact is catastrophic, threatening extensive damage to the client's reputation and/or causing severe financial loss to the client or users. The likelihood of exploitation can be high, warranting a swift response. Critical issues typically involve significant risks such as the permanent loss or locking of a large volume of users' sensitive assets or the failure of core system functionalities without viable mitigations. These issues demand immediate attention due to their potential to compromise system integrity or user trust significantly.

High Severity

These issues are characterized by the potential to substantially impact the client's reputation and/or result in considerable financial losses. The likelihood of exploitation is significant, warranting a swift response. Such issues might include temporary loss or locking of a significant number of users' sensitive assets or disruptions to critical system functionalities, albeit with potential, yet limited, mitigations available. The emphasis is on the significant but not always catastrophic effects on system operation or asset security, necessitating prompt and effective remediation.

Medium Severity

Issues classified as being of medium severity can lead to a noticeable negative impact on the client's reputation and/or moderate financial losses. Such issues, if left unattended, have a moderate likelihood of being exploited or may cause unwanted side effects in the system.

These issues are typically confined to a smaller subset of users' sensitive assets or might involve deviations from the specified system design that, while not directly financial in nature, compromise system integrity or user experience. The focus here is on issues that pose a real but contained risk, warranting timely attention to prevent escalation.

Low Severity

Low-severity issues are those that have a low impact on the client's operations and/or reputation. These issues may represent minor risks or inefficiencies to the client's specific business model. They are identified as areas for improvement that, while not urgent, could enhance the security and quality of the codebase if addressed.

Notes & Additional Information Severity

This category is reserved for issues that, despite having a minimal impact, are still important to resolve. Addressing these issues contributes to the overall security posture and code quality improvement but does not require immediate action. It reflects a commitment to maintaining high standards and continuous improvement, even in areas that do not pose immediate risks.