F Y E O

Security Code Review of XRPL Permissioned DEX

Ripple

May 2025
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

Executive Summary

Overview

Ripple engaged FYEO Inc. to perform a Security Code Review of XRPL Permissioned DEX.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on May 05 - May 09, 2025, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

During our review process, we concluded that the reviewed code implements the documented functionality and zero security issues were identified.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review of XRPL DEX Permissioned Domains. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a public repository at https://github.com/XRPLF/rippled with the commit hash b7634410bac3b59e25e9ed8e1081bdcc0ab38451

The audit was done on a specific PR. Some files listed below were partially reviewed according to the PR: https://github.com/XRPLF/rippled/pull/5404

Files included in the code review
rippled/ ├── include/ │ └── xrpl/ │ └── protocol/ │ ├── Book.h

Files included in the code review

```
│           ├── ErrorCodes.h
│           ├── LedgerFormats.h
│           ├── TxFlags.h
│           └── detail/
│                ├── features.macro
│                ├── ledger_entries.macro
│                ├── sfields.macro
│                └── transactions.macro
└── src/
     ├── libxrpl/
     │   └── protocol/
     │     ├── ErrorCodes.cpp
     │     ├── Indexes.cpp
     │     └── InnerObjectFormats.cpp
     └── xrpld/
          ├── app/
          │   ├── ledger/
          │   │   ├── OrderBookDB.cpp
          │   │   └── OrderBookDB.h
          │   ├── misc/
          │   │   ├── PermissionedDEXHelpers.cpp
          │   │   └── PermissionedDEXHelpers.h
          │   ├── paths/
          │   │   ├── Flow.cpp
          │   │   ├── Flow.h
          │   │   ├── PathRequest.cpp
          │   │   ├── PathRequest.h
          │   │   ├── Pathfinder.cpp
          │   │   ├── Pathfinder.h
          │   │   ├── RippleCalc.cpp
          │   │   ├── RippleCalc.h
          │   │   └── detail/
          │   │       ├── BookStep.cpp
          │   │       ├── PaySteps.cpp
          │   │       └── Steps.h
          │   └── tx/
          │       └── detail/
          │           ├── CashCheck.cpp
          │           ├── CreateOffer.cpp
          │           ├── CreateOffer.h
          │           ├── InvariantCheck.cpp
          │           ├── InvariantCheck.h
          │           ├── OfferStream.cpp
```

Files included in the code review
│ ├── Payment.cpp │ └── XChainBridge.cpp ├── ledger/ │ └── detail/ │ └── View.cpp └── rpc/ ├── BookChanges.h ├── detail/ │ └── TransactionSign.cpp └── handlers/ ├── BookOffers.cpp └── Subscribe.cpp

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review of XRPL DEX Permissioned Domains, we discovered zero (0) issues with the audited code.

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

General Observations

Feature Summary

This PR implements permissioned domains for the XRP Ledger's decentralized exchange (DEX). It introduces domain-specific order books where only authorized participants (via credentials or ownership) can trade. Key components:
Domain-Centric Offers: Offers can be tied to a domain (via `sfDomainID`), restricting liquidity to participants within that domain.

1. Hybrid Offers: Offers can exist in both permissioned (domain) and open order books simultaneously, enabling flexible liquidity sharing.
2. Domain Validation: Functions like `accountInDomain` and `offerInDomain` enforce membership checks via on-ledger credentials or ownership.
3. Order Book Tracking: The `OrderBookDB` is extended to track both domain-specific and open books separately, ensuring pathfinding respects domain restrictions.
4. Transaction Support: Modified `OfferCreate` and `Payment` transactions to include domain parameters, with new error codes (`rpcDOMAIN_MALFORMED`).

Architecture Integration

- Domain Enforcement: During offer creation or payment execution, the ledger validates whether the sender/offer belongs to the specified domain.
- Hybrid Logic: Hybrid offers write to two order books (domain and open) via `sfAdditionalBooks`, ensuring liquidity is visible in both contexts.
- Pathfinding: The `Pathfinder` and `Flow` modules query the `OrderBookDB` with domain context, filtering liquidity based on the transaction's domain rules.
- RPC Extensions: Methods like `book_offers` and `subscribe` support domain filtering, enabling clients to interact with permissioned markets.

Coding Style

- Consistency: Uses modern C++ features (`std::optional`, type-safe `enum class`) appropriately. Domain logic is centralized in `PermissionedDEXHelpers`, promoting reusability.
- Readability: Complex logic (hybrid offer handling) is well-structured, though some functions (`operator<=>` in `Book.h`) could benefit from comments.
- Error Handling: New error codes (`tecNO_PERMISSION`, `rpcDOMAIN_MALFORMED`) are clearly defined, but some edge cases (domain expiry) lack explicit checks.
- Performance: Efficient data structures (`hash_set` for tracking domains) are used, but repeated ledger queries in `accountInDomain` might be optimizable.
- Safety: Invariant checks (`ValidPermissionedDEX`) prevent invalid hybrid offers, though shadowed variables (`c` for the spaceship operator in `Book.h`) could be renamed.

Overall Impression

The update introduces features for permissioned trading, balancing flexibility (hybrid offers) with strict access control. The architecture is logically layered, but minor improvements in code clarity and edge-case handling would strengthen maintainability.

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.