F Y E O

# Security Code Review of XRPL Token Escrow

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 Token Escrow.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on April 01 - April 21, 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

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-XRPL-01 – Future-Proofing Advisory for Overflow/Underflow in Escrow Balance Adjustments

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

## Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review of XRPL Token Escrow. 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/Transia-RnD/rippled/ with the commit hash 5408b942c2c4da5b86daf816a00fd9f988ab6f19

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/5185/

Files included in the code review

```
rippled/
├── include/
│   └── xrpl/
│       └── protocol/
│           ├── LedgerFormats.h
│           ├── STAmount.h
│           ├── TER.h
│           ├── TxFlags.h
│           ├── detail/
│           │   ├── features.macro
│           │   ├── ledger_entries.macro
│           │   ├── sfields.macro
│           │   └── transactions.macro
│           └── jss.h
└── src/
    ├── libxrpl/
    │   ├── basics/
    │   │   └── mulDiv.cpp
    │   └── protocol/
    │       ├── STAmount.cpp
    │       └── TER.cpp
    ├── test/
    │   ├── app/
    │   │   ├── AMM_test.cpp
    │   │   ├── AccountDelete_test.cpp
    │   │   ├── DepositAuth_test.cpp
    │   │   ├── EscrowToken_test.cpp
    │   │   ├── Escrow_test.cpp
    │   │   └── MPToken_test.cpp
    │   ├── jtx/
    │   │   ├── TestHelpers.h
    │   │   ├── escrow.h
    │   │   └── impl/
    │   │       ├── TestHelpers.cpp
    │   │       └── escrow.cpp
    │   ├── jtx.h
    │   ├── ledger/
    │   │   └── Invariants_test.cpp
    │   ├── protocol/
    │   │   └── STAmount_test.cpp
    │   └── rpc/
    │       ├── AccountLines_test.cpp
    │       ├── AccountSet_test.cpp
```

Files included in the code review

```
│       └── LedgerRPC_test.cpp
└── xrpld/
    ├── app/
    │   └── tx/
    │       └── detail/
    │           ├── Escrow.cpp
    │           ├── Escrow.h
    │           ├── InvariantCheck.cpp
    │           ├── MPTokenAuthorize.cpp
    │           └── SetAccount.cpp
    ├── ledger/
    │   ├── View.h
    │   └── detail/
    │       └── View.cpp
    └── rpc/
        └── handlers/
            └── GatewayBalances.cpp
```

Table 1: Scope

# Technical Analyses and Findings

During the Security Code Review of XRPL Token Escrow, we discovered:

- 1 finding with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

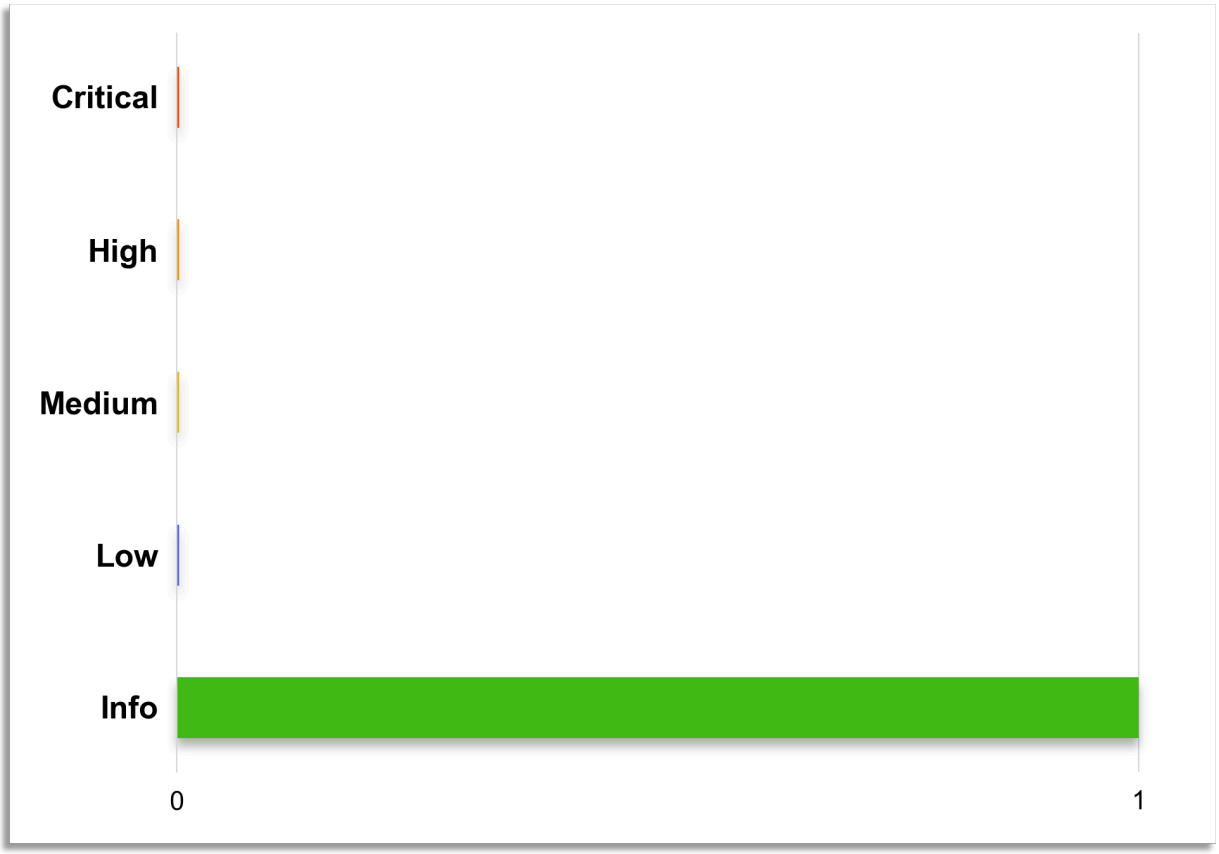


Figure 1: Findings by Severity

## Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

| Finding # | Severity | Description | Status |
|---|---|---|---|
| FYEO-XRPL-01 | Informational | Future-Proofing Advisory for Overflow/Underflow in Escrow Balance Adjustments | Acknowledged |

Table 2: Findings Overview

## 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.

# Technical Findings

## General Observations

A new transaction flag named asfAllowTokenLocking and related ledger fields such as sfEscrowedAmount have been introduced. The protocol macros and sfield definitions have been updated consistently. In RPC handlers such as GatewayBalances, escrowed amounts are aggregated and reported correctly.

Many functions including those in preflight, preclaim, and doApply methods for EscrowCreate, EscrowFinish, and EscrowCancel check that the new token escrow feature is enabled before proceeding with MPT-related logic. The use of std::visit to dispatch on the asset type (Issue versus MPTIssue) makes the code paths explicit.

For EscrowCreate, the new preflight helper templates for both Issue and MPTIssue types are logically separated. Checks are in place to validate net amount, currency, and issuer-account relations. The conversion of XRP amounts to IOU format in the addability check and consistent update of owner directories is handled.

In EscrowFinish and EscrowCancel, both preclaim and doApply functions have been extended to include non-XRP (MPT) escrow flows. The logic using rippleLockEscrowMPT and rippleUnlockEscrowMPT functions is clearly separated for Issue and MPTIssue. In doApply, escrows are removed from the correct owner directories, including the issuer's directory for non-XRP cases.

The rippleLockEscrowMPT and rippleUnlockEscrowMPT functions locate the MPT issuance and the corresponding MPT holder entries. In the lock function, the holder's MPT amount is checked for sufficiency before reducing it and increasing the escrowed amount. In the unlock function, the issuance's escrowed amount is reduced, and then either the holder's balance is increased if the issuer is not the receiver, or the issuance's outstanding amount is decreased if the issuer is the receiver.

Changes in InvariantCheck.cpp skip certain invariant failures if the token escrow feature is enabled or if specific transaction types such as escrow finish are involved.

The code follows the XRPL code base style, including macros, structured bindings, and lambdas. There is a consistent pattern of using feature flags such as featureTokenEscrow before executing new logic paths. Errors throughout the new functions are returned using the standard TER/NotTEC mechanisms.

Overall, the code changes are extensive and align with the architectural style of XRPL. The new token escrow feature is integrated into various components, including the protocol, ledger entries, transaction processing, consensus, nodestore rotation, and peer handling.

# Future-Proofing Advisory for Overflow/Underflow in Escrow Balance Adjustments

Finding ID: FYEO-XRPL-01
Severity: **Informational**
Status: **Acknowledged**

## Description

The escrow logic performs arithmetic operations on account balances using raw unsigned integers (uint64_t) without explicit overflow or underflow protections. For example, in EscrowCreate, the sender's XRP balance is reduced using:

```
(*sle)[sfBalance] = (*sle)[sfBalance] - ctx_.tx[sfAmount];
```

And in EscrowFinish or EscrowCancel, balances are increased using:

```
(*sled)[sfBalance] = (*sled)[sfBalance] + (*slep)[sfAmount];
```

Under today's ledger rules, these operations cannot overflow or underflow. This is due to:

- XRP invariant checks: no account balance may ever exceed the total drops in existence or fall below zero.
- Pre-debit reserve and fund checks: subtractions are only performed if the sender has at least that many drops available.

Consequently, no realistic transaction can trigger an arithmetic wrap under current rules.

## Proof of Issue

**File name:** src/xrpld/app/tx/detail/Escrow.cpp
Line Number: ~278
(function EscrowCreate::doApply)

```
(*sle)[sfBalance] = (*sle)[sfBalance] - ctx_.tx[sfAmount];
Overflow (Balance Increase on Finish)
```

Line Number: ~496
(function EscrowFinish::doApply)

```
(*sled)[sfBalance] = (*sled)[sfBalance] + (*slep)[sfAmount];
Overflow (Balance Increase on Cancel)
```

Line Number: ~594
(function EscrowCancel::doApply)

```
(*sle)[sfBalance] = (*sle)[sfBalance] + (*slep)[sfAmount];
```

## Severity and Impact Summary

**Current safety:** The built-in XRP invariant (0 ≤ balance ≤ max drops) guarantees these additions and subtractions cannot wrap, so the ledger remains consistent today.

**Future risk:** If protocol or code changes ever alter supply limits or disable/enhance invariant enforcement, raw 64-bit math could silently wrap and corrupt balances.

While there is no present vulnerability, it is prudent to guard against any future scenario in which invariants might shift or be bypassed.

### Recommendation

Centralize all XRP balance updates behind a helper that performs checked arithmetic.

Document in code comments that escrow operations rely on current supply invariants.

Add explicit assertions or saturating/checked-integer types around every balance adjustment.

Extend CI with static analysis or sanitizer builds, and include fuzz tests targeting escrow boundary cases.

# 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.