



Ripple
Multi-Purpose Token (MPT)
SECURITY ASSESSMENT

22.10.2024

Made in Germany by Softstack.io



Table of contents

1. Disclaimer.....	3
2. About the Project and Company	4
2.1 Project Overview.....	5
3. Vulnerability & Risk Level	6
4. Auditing Strategy and Techniques Applied.....	7
4.1 Methodology	7
5. Metrics	8
5.1 Tested Codebase Files.....	8
6. Scope of Work.....	11
6.1 Findings Overview	12
6.2 Manual and Automated Vulnerability Test.....	13
6.2.1 Potential Memory Safety Vulnerability in MPTIssue::getIssuer() Method	13
6.2.2 Potential Race Condition in MPToken Locking/Unlocking Operations.....	14
6.2.3 Lack of Explanation for isXRP() Function Behavior.....	16
6.2.4 Inconsistent Use of Commented-Out Code in requireAuth Function.....	18
6.3 Unit Tests.....	19
7. Executive Summary.....	21
8. About the Auditor	22



1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Ripple Labs Inc. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (23.09.2024)	Layout
0.4 (26.09.2024)	Create isolated testing environment
0.5 (02.10.2024)	Manual security assessment
0.9 (03.10.2024)	Summary and Recommendation
1.0 (07.10.2024)	Final document
1.1 (22.10.2024)	Re-check https://github.com/XRPLF/rippled/pull/5143/commits/6a7a8c27b9d98b0f40685345786d4803f8139098



2. About the Project and Company

Company address:

Ripple Labs Inc.
600 Battery St.
San Francisco, CA, 94111
United States of America



Website: <https://ripple.com/>

LinkedIn: <https://www.linkedin.com/company/rippleofficial>

Twitter (X): <https://twitter.com/Ripple>

Youtube: <https://www.youtube.com/channel/UCjok1uTSBUgvRYQaASz6YWw>

Facebook: <https://www.facebook.com/rippleofficial>

2.1 Project Overview

Ripple provides blockchain and cryptocurrency solutions for faster, cost-effective, and secure cross-border payments. Ripple developed the XRP Ledger (XRPL), an open-source, decentralized blockchain known for high transaction throughput, low fees, and support for trust lines. Ripple's solutions, including RippleNet, aim to streamline international payments and reduce costs in the traditional financial system.

The XRP Ledger supports the issuance of various tokens using trust lines, which are an essential mechanism for managing credit and asset issuance on the network. Trust lines enable accounts to establish a bilateral balance relationship with an issuer, facilitating the creation, holding, and transfer of tokens, including stablecoins, loyalty points, and other digital assets. Trust lines also provide users with control over which issuers they trust and how much value they are willing to receive, thereby ensuring greater security and preventing unwanted assets from being held in their accounts. Additionally, the XRP Ledger offers advanced features like rippling, which allows balances to flow through a chain of trust lines, enhancing liquidity and simplifying currency conversion.

Multi-Purpose Tokens (MPTs) represent a new development on the XRP Ledger, introduced to address scalability and storage limitations of trust lines while offering greater simplicity for token issuance and management. MPTs function as unidirectional trust lines that require significantly less storage space compared to traditional trust lines. They are designed to support a wide range of use cases, including monetary and non-monetary applications, such as stablecoins, utility tokens, and reputation points.

The specific focus of this security assessment will be the Multi-Purpose Tokens (MPTs) functionality on the XRP Ledger. This audit aims to evaluate the security, scalability, and efficiency of MPTs, ensuring that they meet the rigorous standards necessary to maintain network resilience and stability.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the codebase functioning in a number of scenarios, or creates a risk that the codebase may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a codebase, or provides the opportunity to use a codebase in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the codebase in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the codebase and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the codebase.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - i. Review of the specifications, sources, and instructions provided to softstack to make sure we understand the size, scope, and functionality of the codebase.
 - ii. Manual review of codebase, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - iii. Comparison to specification, which is the process of checking whether the codebase does what the specifications, sources, and instructions provided to softstack describe.
2. Testing and automated analysis that includes the following:
 - i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the codebase and how much code is exercised when we run those test cases.
 - ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the codebase to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your codebase.



5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Codebae Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Source: <https://github.com/XRPLF/rippled/pull/5143>

File	Fingerprint (MD5)
include/xrpl/basics/MPTAmount.h	d411ee1a3bf9a45d7037bba69c27d745
include/xrpl/basics/Number.h	f1492919562144d042a730744024ca4d
include/xrpl/basics/XRPAmount.h	198972a5ae80adf7e2ec81c8d564e0a5
include/xrpl/basics/base_uint.h	735e934e3fc0601281611645fd400849
include/xrpl/protocol/AmountConversions.h	cc97c80f48860361431ddaa200a0ea6c
include/xrpl/protocol/Asset.h	6aa63ebcbcbcd126e4b83a5476738f2b
include/xrpl/protocol/Feature.h	99f27993348b1cee747a968037d909df
include/xrpl/protocol/Indexes.h	133182012dd04b0d20c1d62743d414ac
include/xrpl/protocol/Issue.h	1faf713eb6179df8bc52d92d506cdaf4
include/xrpl/protocol/LedgerFormats.h	f0350ab3e82dd1852fab6a8f8da37ba1
include/xrpl/protocol/MPTIssue.h	3fed9197238b74e083110d75f1a7b2ba
include/xrpl/protocol/Protocol.h	2412a70dadac965dd7eb6417857482e8
include/xrpl/protocol/SField.h	d996eb238edde47229e71d487bb8a091
include/xrpl/protocol/SOTemplate.h	5a07b9442c6515496a627108f609753f
include/xrpl/protocol/STAmount.h	a81e8104b4ca62438921b73d04d90923
include/xrpl/protocol/STBitString.h	fb636c8e22aa9edcbdb44100291816f5

include/xrpl/protocol/STObject.h	428f10937adf3fd7b65af0696c4d729b
include/xrpl/protocol/Serializer.h	bed2a5510c3359d2ac2dbd6961709697
include/xrpl/protocol/TER.h	0c5541acc9779fb46960bbacad19ed0a
include/xrpl/protocol/TxFlags.h	a378f9f653a8afe108cbbdf4529f2b0c
include/xrpl/protocol/TxFormats.h	05d1e9d11364fc30f5fc8fc373720370
include/xrpl/protocol/detail/STVar.h	6cf9de9ed1f45fa1ef64848dd1119ad4
include/xrpl/protocol/jss.h	ae99dda1cf24124044e6d96317503dbe
src/libxrpl/basics/MPTAmount.cpp	f16e0ac0ce4b6a8ea632f3125b3ef403
src/libxrpl/protocol/Asset.cpp	ed0ddd74ddc24ba4782f0ebc6f4d1703
src/libxrpl/protocol/Feature.cpp	221eb26f8b43d80357882b9e62b7d57f
src/libxrpl/protocol/Indexes.cpp	a198fe3211cb9bef9bc6fa0e2dd8868f
src/libxrpl/protocol/Issue.cpp	2f93ac053379b58ff8567972108ea41d
src/libxrpl/protocol/LedgerFormats.cpp	ee5cdc281dacebe893d7fdae9f06ff21
src/libxrpl/protocol/Quality.cpp	eb4672debe8d06ad4f668383b0690741
src/libxrpl/protocol/Rate2.cpp	073825818bbc63dbd971f95f51112a71
src/libxrpl/protocol/SField.cpp	b1010829ce2590d995889c859e849132
src/libxrpl/protocol/STAmount.cpp	d3be746eed071bafd11ff3c5178327b9
src/libxrpl/protocol/STInteger.cpp	7e1db5cc6c7186ae2bd583f8d2111ab1
src/libxrpl/protocol/STObject.cpp	6df22abea5c604081070e8ca1fb83a61
src/libxrpl/protocol/STParsedJSON.cpp	409c1bbaa4320005946b6e4c898b99f8
src/libxrpl/protocol/STTx.cpp	a53f6a28108dc619dc0c96d7113a93f3
src/libxrpl/protocol/STVar.cpp	51054beba6828e875c7635bc7263fc1f
src/libxrpl/protocol/TER.cpp	40affeb33a442c0a65537941b40f8a92
src/libxrpl/protocol/TxFormats.cpp	dd6f6d2d6134375283097b46ed8ea662
src/xrpId/app/ledger/detail/LedgerToJson.cpp	c9374b8d602e60e2684597525fd28814
src/xrpId/app/misc/NetworkOPs.cpp	96a34852d2cd4caf8a70d7e3054579c9
src/xrpId/app/paths/Credit.cpp	ab6f304813eaca3e0568b278f996d91d
src/xrpId/app/paths/PathRequest.cpp	4e116a2aa67641210682a5ff06908c6a
src/xrpId/app/paths/Pathfinder.cpp	6c143597a052978a22c3455e0f510d3f
src/xrpId/app/tx/detail/Clawback.cpp	cdef32eeb7cd3e07c3d1a79cf7609ab3
src/xrpId/app/tx/detail/MPTokenAuthorize.cpp	9a98c30d4c0db5f213d364f79a362f29

src/xrpld/app/tx/detail/MPTokenAuthorize.h	83ddd873497c638feab9a69b2a2b02a9
src/xrpld/app/tx/detail/MPTokenIssuanceCreate.cpp	5e75dd88b6d6dc4a413d63091afcf928
src/xrpld/app/tx/detail/MPTokenIssuanceCreate.h	5085565d12b5026957b7d2a774652a49
src/xrpld/app/tx/detail/MPTokenIssuanceDestroy.cpp	7c06a4a697afcf8ba0ab7c644f0e9ecc
src/xrpld/app/tx/detail/MPTokenIssuanceDestroy.h	cc3e9a7b630208e5a86d1c900d64a076
src/xrpld/app/tx/detail/MPTokenIssuanceSet.cpp	3b241e890d0862d3dde2de4ccf19d7ba
src/xrpld/app/tx/detail/MPTokenIssuanceSet.h	056c7e5ca2a4d8f2eab649145dabd79e
src/xrpld/app/tx/detail/Payment.cpp	439a29f189dda16304e4f088704aa220
src/xrpld/app/tx/detail/SetTrust.cpp	94ef4adfe0c781eecf88de697c8bcde0
src/xrpld/app/tx/detail/applySteps.cpp	c19d35973ac42d5243524327b0825ee7
src/xrpld/ledger/View.h	a821d59fb532f46afd4374dcb27ac05e
src/xrpld/ledger/detail/View.cpp	46925a374423d3d66f174a48d8a6fdbb
src/xrpld/rpc/MPTokenIssuanceID.h	528eb27c3f2110e5bb5e40fc2bf62406
src/xrpld/rpc/detail/MPTokenIssuanceID.cpp	3a3e4764a2d8549d6b8b71f43173a286
src/xrpld/rpc/detail/RPCHelpers.cpp	c8e610cfbb5f0b918db146de6a3e6e96
src/xrpld/rpc/detail/TransactionSign.cpp	205e68b6e87f515d5355fdaaad7c652f
src/xrpld/rpc/detail/Tuning.h	fb4d7fc4651c64433c90585fade1f7be
src/xrpld/rpc/handlers/AccountObjects.cpp	b50e7b7770efed73bd1548c087014292
src/xrpld/rpc/handlers/AccountTx.cpp	34215673992d9e2d988996671b5d143c
src/xrpld/rpc/handlers/Handlers.h	7213a634eb05c6c1908bef8d19c9370a
src/xrpld/rpc/handlers/LedgerData.cpp	ce398036251a71b951d5cd1b39cd7a51
src/xrpld/rpc/handlers/LedgerEntry.cpp	7f123855e6d6f5d3406972a0c9521f1c
src/xrpld/rpc/handlers/Tx.cpp	0d1a5b37d74836e386786c4eb715c88c

6. Scope of Work

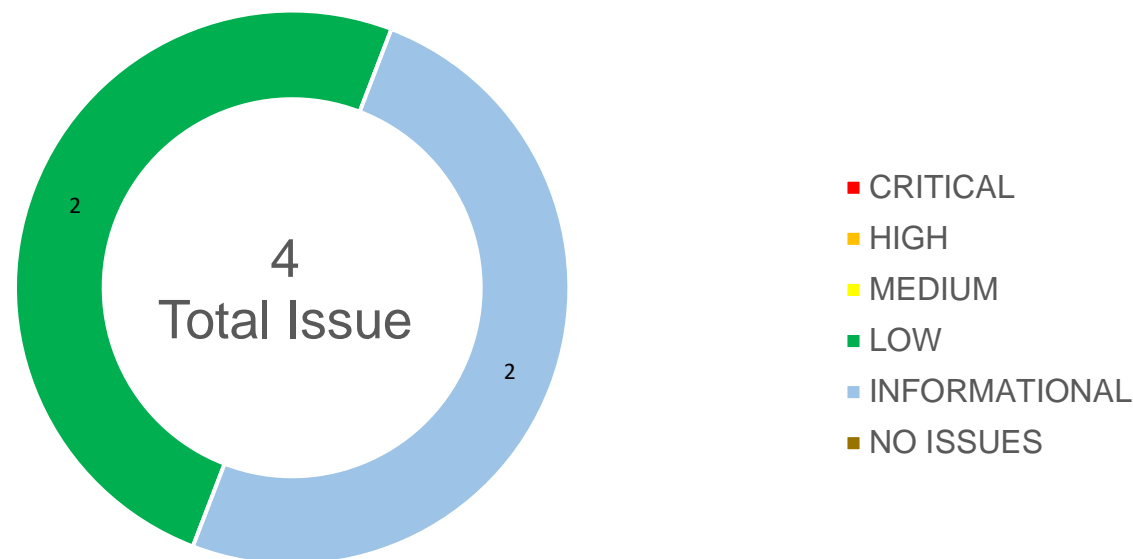
The Ripple team provided us with the files that need to be assessed. The scope of the audit includes the XLS-33d Multi-Purpose Tokens (MPT) functionality for the XRP Ledger.

The team put forward the following assumptions regarding the security, efficiency, and compliance of the MPT implementation:

1. **Code Efficiency:** The audit should ensure that MPT operations are optimized for performance on the XRP Ledger, focusing on how MPTs are stored and managed on the ledger to prevent potential performance bottlenecks.
2. **Space Efficiency Validation:** The audit should verify whether the claim of 52-byte storage per MPT is accurate in practice. Comparisons with traditional trustlines will be performed to validate the actual space savings.
3. **Compliance with Best Practices:** The MPT implementation should adhere to Ripple's best practices, ensuring maintainability, consistency, and minimized security risks. The code should also follow Ripple's style guidelines to ensure consistent formatting throughout the codebase.
4. **Security Assumptions Validation:** The audit should test the security assumptions made in the MPT proposal, particularly the unidirectional trustline concept, to ensure it does not introduce new vulnerabilities to the ledger.
5. **Functional Verification:** The audit should verify that MPT operations—creation, transfer, locking, and destruction—work as intended and align with the XRP Ledger's tokenization design. This includes testing various operations such as buying, selling, and holding tokens.

The primary goal of this audit is to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Potential Memory Safety Vulnerability in MPTIssue::getIssuer() Method	LOW	FIXED
6.2.2	Potential Race Condition in MPToken Locking/Unlocking Operations	LOW	ACKNOWLEDGED
6.2.3	Lack of Explanation for isXRP() Function Behavior	INFORMATIONAL	FIXED
6.2.4	Inconsistent Use of Commented-Out Code in requireAuth Function	INFORMATIONAL	FIXED



6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, softstack's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, softstack's experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, softstack's experts found **no Medium issues** in the code of the smart contract.

LOW ISSUES

During the audit, softstack's experts found **2 Low issues** in the code of the smart contract

6.2.1 Potential Memory Safety Vulnerability in MPTIssue::getIssuer() Method

Severity: LOW

Status: FIXED

File(s) affected: /rippled/src/libxrpl/protocol/MPTIssue.cpp

Update: <https://github.com/XRPLF/rippled/pull/5143/commits/6a7a8c27b9d98b0f40685345786d4803f8139098>

Attack / Description	The MPTIssue::getIssuer() method in the provided code uses memcpy() to extract the AccountID from the MPTID. This implementation assumes a specific memory layout and offset for the AccountID within the MPTID structure. This approach can lead to undefined behavior and potential security vulnerabilities if the MPTID structure changes or if the assumptions about memory layout are incorrect.
-----------------------------	--

Code	<p>Line 30 - 38 (MPTIssue.cpp):</p> <pre> AccountID const& MPTIssue::getIssuer() const { // copy from id skipping the sequence AccountID const* account = reinterpret_cast<AccountID const*>(mptID_.data() + sizeof(std::uint32_t)); return *account; } </pre>
Result/Recommendation	<p>We propose the following recommendations:</p> <ol style="list-style-type: none"> 1. Avoid using memcpy() for extracting structured data. Instead, implement a safer method to access the AccountID within the MPTID structure. 2. If possible, redesign the MPTID class to provide a safe method for accessing its components, such as getAccountID(). 3. If the current structure must be maintained, add runtime checks to ensure that the MPTID has the expected size and structure before performing the copy. 4. Consider using C++17's std::byte or a custom safe buffer class that provides bounds-checking.

6.2.2 Potential Race Condition in MPToken Locking/Unlocking Operations

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: /rippled/src/xrpld/app/tx/detail/MPTokenIssuanceSet.cpp

Update: This should not be addressed at the transactor layer, as it's going to be handled during the consensus, which makes sure that transactions like MPTokenIssuanceSet is executed sequentially to avoid race condition.

Attack / Description	The current implementation of the MPTokenIssuanceSet transaction, which handles locking and unlocking of MPTokens, lacks a mechanism to check for conflicts with other ongoing operations on the MPToken or MPTokenIssuance objects. This oversight could potentially lead to race conditions or inconsistent states in the ledger.
Code	Line 86 - 116 (MPTokenIssuanceSet.cpp): <pre> TER MPTokenIssuanceSet::doApply() { auto const mptIssuanceID = ctx_.tx[sfMPTokenIssuanceID]; auto const txFlags = ctx_.tx.getFlags(); auto const holderID = ctx_.tx[~sfMPTokenHolder]; std::shared_ptr<SLE> sle; if (holderID) sle = view().peek(keylet::mptoken(mptIssuanceID, *holderID)); else sle = view().peek(keylet::mptIssuance(mptIssuanceID)); if (!sle) return tecINTERNAL; std::uint32_t const flagsIn = sle->getFieldU32(sfFlags); std::uint32_t flagsOut = flagsIn; if (txFlags & tfMPTLock) flagsOut = IsfMPTLocked; else if (txFlags & tfMPTUnlock) flagsOut &= ~IsfMPTLocked; </pre>

	<pre> if (flagsIn != flagsOut) sle->setFieldU32(sfFlags, flagsOut); view().update(sle); return tesSUCCESS; } </pre>
Result/Recommendation	<p>We propose the following recommendations:</p> <ol style="list-style-type: none"> 1. Implement a locking mechanism: Before processing the lock/unlock operation, acquire a temporary lock on the MPToken and MPTokenIssuance objects. If the lock cannot be acquired (indicating an ongoing operation), the transaction should fail with an appropriate error code. 2. Add a status check: Before applying the lock/unlock operation, check the current status of the MPToken and MPTokenIssuance objects. If the status indicates an ongoing operation, abort the transaction with an appropriate error code. 3. Use a version number or timestamp: Implement a version number or timestamp for MPToken and MPTokenIssuance objects. Before applying changes, verify that the object's version matches the expected version when the transaction was initiated. If the versions don't match, it indicates a concurrent modification, and the transaction should be aborted.

INFORMATIONAL ISSUES

During the audit, softstack's experts found **2 Informational issue** in the code of the smart contract.

6.2.3 Lack of Explanation for isXRP() Function Behavior

Severity: INFORMATIONAL

Status: FIXED

File(s) affected: /rippled/include/xrpl/protocol/MPTIssue.h

Update: <https://github.com/XRPLF/rippled/pull/5143/commits/6a7a8c27b9d98b0f40685345786d4803f8139098>

Attack / Description	The isXRP() function always returns false for MPTID instances, but there is no explanation provided for this behavior. This lack of clarity can confuse developers who are unfamiliar with Multi-Purpose Tokens (MPTs) and their distinction from XRP.
Code	Line 79 - 83 (MPTIssue.h): <pre>inline bool isXRP(MPTID const&) { return false; }</pre>
Result/Recommendation	Add a detailed comment explaining why MPTs are never considered XRP to clarify the intended functionality: <pre>// MPTs are never considered XRP as they represent a distinct token type inline bool isXRP(MPTID const&) { return false; }</pre> This comment will help developers understand the design decision, reducing confusion and preventing potential misunderstandings regarding the relationship between MPTs and XRP.

6.2.4 Inconsistent Use of Commented-Out Code in requireAuth Function

Severity: INFORMATIONAL

Status: FIXED

File(s) affected: /rippled/src/xrpld/ledger/detail/View.cpp

Attack / Description	The presence of commented-out code in the requireAuth function reduces the readability and maintainability of the codebase. Specifically, the line (sle->getFlags() & IsfMPTRRequireAuth) is commented out, which could create confusion for future developers as to whether this check is intended to be used or permanently removed. Leaving such commented-out code in production code can make it unclear if there is an unresolved issue or if the code has become obsolete.
Code	Line 1716 – 1733 (View.cpp): TER requireAuth(ReadView const& view, MPTIssue const& mptIssue, AccountID const& account) { auto const mptID = keylet::mptIssuance(mptIssue.getMptID()); if (auto const sle = view.read(mptID); sle && sle->getFieldU32(sfFlags) & IsfMPTRRequireAuth) { auto const mptokenID = keylet::mptoken(mptID.key, account); if (auto const tokSle = view.read(mptokenID); tokSle && //(sle->getFlags() & IsfMPTRRequireAuth) && !(tokSle->getFlags() & IsfMPTAuthorized)) return TER{tecNO_AUTH}; } return tesSUCCESS; }




Result/Recommendation	Remove the commented-out line (sle->getFlags() & IsfMPTRRequireAuth) if it is no longer necessary. If the check is meant to be reintroduced later or is still under consideration, add a detailed comment explaining the purpose and future intent of this code instead of leaving it commented out.
------------------------------	--

6.3 Unit Tests


Name	Missed Lines	Patch %	Head %	Change %
src/libxrpl/protocol/STObject.cpp	2	0.00%	84.84%	-0.45%
src/xrpld/rpc/handlers/LedgerEntry.cpp	32	5.88%	71.90%	-5.71%
src/libxrpl/basics/MPTAmount.cpp	22	8.33%	8.33%	-
src/xrpld/rpc/handlers/LedgerData.cpp	2	33.33%	56.04%	-0.77%
src/xrpld/rpc/detail/RPCHelpers.cpp	3	50.00%	81.21%	-0.43%
src/libxrpl/protocol/MPTIssue.cpp	6	50.00%	50.00%	-
src/xrpld/app/ledger/detail/LedgerToJson.cpp	2	50.00%	76.10%	-0.67%
src/libxrpl/protocol/STParsedJSON.cpp	6	64.71%	64.35%	0.40%
src/xrpld/app/tx/detail/InvariantCheck.cpp	30	65.52%	86.01%	-5.08%
src/libxrpl/protocol/Asset.cpp	8	72.41%	72.41%	-
src/libxrpl/protocol/STAmount.cpp	47	73.99%	84.77%	-5.26%
src/xrpld/rpc/detail/MPTokenIssuanceID.cpp	4	83.33%	83.33%	-
src/xrpld/app/tx/detail/Payment.cpp	21	85.42%	84.05%	-2.35%

6.4 Verify Claims


6.4.1 Code Efficiency: The audit should ensure that MPT operations are optimized for performance on the XRP Ledger, focusing on how MPTs are stored and managed on the ledger to prevent potential performance bottlenecks.

Status: tested and verified 


6.4.2 Space Efficiency Validation: The audit should verify whether the claim of 52-byte storage per MPT is accurate in practice. Comparisons with traditional trustlines will be performed to validate the actual space savings.

Status: tested and verified 


6.4.3 Compliance with Best Practices: The MPT implementation should adhere to Ripple's best practices, ensuring maintainability, consistency, and minimized security risks. The code should also follow Ripple's style guidelines to ensure consistent formatting throughout the codebase.

Status: tested and verified 

6.4.4 Security Assumptions Validation: The audit should test the security assumptions made in the MPT proposal, particularly the unidirectional trustline concept, to ensure it does not introduce new vulnerabilities to the ledger.

Status: tested and verified 

6.4.5 Functional Verification: The audit should verify that MPT operations—creation, transfer, locking, and destruction—work as intended and align with the XRP Ledger's tokenization design. This includes testing various operations such as buying, selling, and holding tokens.

Status: tested and verified 

7. Executive Summary

Two independent experts from Softstack conducted an unbiased and isolated audit of the Ripple Multi-Purpose Token (MPT) implementation on the XRP Ledger. The main objective of the audit was to verify the security, efficiency, and functionality claims of the MPT implementation. The audit process involved a thorough manual code review combined with automated security testing. Overall, the audit identified a total of four issues, classified as follows:

- No critical issues were found.
- No high severity issues were found.
- No medium severity issues were found.
- 2 low severity issues were discovered.
- 2 informational issues were identified.

The audit report provides detailed descriptions of each identified issue, including severity levels, recommendations for mitigation, and applicable code snippets demonstrating the issues and proposed fixes. Based on the nature of the findings and adherence to the business logic, we recommend that the Ripple development team review the suggestions and implement the necessary mitigations.

8. About the Auditor

Established in 2017 under the name Chainsulting, and rebranded as softstack GmbH in 2023, softstack has been a trusted name in Web3 Security space. Within the rapidly growing Web3 industry, softstack provides a comprehensive range of offerings that include software development, cybersecurity, and consulting services. Softstack's competency extends across the security landscape of prominent blockchains like Solana, Tezos, TON, Ethereum and Polygon. The company is widely recognized for conducting thorough code audits aimed at mitigating risk and promoting transparency.

The firm's proficiency lies particularly in assessing and fortifying smart contracts of leading DeFi projects, a testament to their commitment to maintaining the integrity of these innovative financial platforms. To date, softstack plays a crucial role in safeguarding over \$100 billion worth of user funds in various DeFi protocols.

Underpinned by a team of industry veterans possessing robust technical knowledge in the Web3 domain, softstack offers industry-leading smart contract audit services. Committed to evolving with their clients' ever-changing business needs, softstack's approach is as dynamic and innovative as the industry it serves.

Check our website for further information: <https://softstack.io>

How We Work



1 -----

PREPARATION

Supply our team with audit ready code and additional materials



2 -----

COMMUNICATION

We setup a real-time communication tool of your choice or communicate via e-mails.



3 -----

AUDIT

We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.



4 -----

FIXES

Your development team applies fixes while consulting with our auditors on their safety.



5 -----

REPORT

We check the applied fixes and deliver a full report on all steps done.

