F Y E O

Security Code Review of Axelar XRPL

Axelar

January 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

Axelar engaged FYEO Inc. to perform a Security Code Review of Axelar XRPL.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on November 06 - November 20, 2024, and additional code was reviewed January 3rd - January 13th 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-AXELAR-01 – Wrong amount used for XRPL payments
- FYEO-AXELAR-02 – No tests implemented for multisig prover contract
- FYEO-AXELAR-03 – Payment flags are ignored
- FYEO-AXELAR-04 – Use of unwrap & expect in verifier code
- FYEO-AXELAR-05 – Exponent of 80 will fail in XRPL token amount
- FYEO-AXELAR-06 – Function update_tx_status accepts empty signer_public_keys
- FYEO-AXELAR-07 – Shadowed variable prevents event emission
- FYEO-AXELAR-08 – Unimplemented functionality

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 Axelar XRPL. 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/commonprefix/axelar-amplifier with the commit hash e7829b40533c341bce3bd371da614b1865d89aa4.

Remediations were submitted with the commit hash 84bd1001b53be380ac1d697736d414735cd26587.

Files included in the code review

```
axelar-amplifier/
├── ampd/
│   └── src/
│       ├── handlers/
│       │   ├── xrpl_multisig.rs
│       │   └── xrpl_verify_msg.rs
│       └── xrpl/
│           ├── error.rs
│           ├── finalizer.rs
│           ├── json_rpc.rs
│           ├── mod.rs
│           └── verifier.rs
├── contracts/
│   ├── xrpl-gateway/
│   │   ├── src/
│   │   │   ├── bin/
│   │   │   │   └── schema.rs
│   │   │   ├── contract/
│   │   │   │   ├── execute.rs
│   │   │   │   └── query.rs
│   │   │   ├── contract.rs
│   │   │   ├── events.rs
│   │   │   ├── lib.rs
│   │   │   ├── msg.rs
│   │   │   └── state.rs
│   ├── xrpl-multisig-prover/
│   │   └── src/
│   │       ├── bin/
│   │       │   └── schema.rs
│   │       ├── contract/
│   │       │   ├── execute.rs
```

Files included in the code review

```
│   │       │   ├── query.rs
│   │       │   └── reply.rs
│   │       ├── axelar_verifiers.rs
│   │       ├── contract.rs
│   │       ├── error.rs
│   │       ├── events.rs
│   │       ├── lib.rs
│   │       ├── msg.rs
│   │       ├── querier.rs
│   │       ├── state.rs
│   │       ├── xrpl_multisig.rs
│   │       └── xrpl_serialize.rs
│   └── xrpl-voting-verifier/
│       └── src/
│           ├── bin/
│           │   └── schema.rs
│           ├── contract/
│           │   ├── execute.rs
│           │   └── query.rs
│           ├── client.rs
│           ├── contract.rs
│           ├── error.rs
│           ├── events.rs
│           ├── lib.rs
│           ├── msg.rs
│           └── state.rs
└── packages/
    └── xrpl-types/
        └── src/
            ├── error.rs
            ├── lib.rs
            ├── msg.rs
            └── types.rs
```

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review of Axelar XRPL, we discovered:

- 1 finding with CRITICAL severity rating.
- 3 findings with MEDIUM severity rating.
- 4 findings with LOW severity rating.

The following chart displays the findings by severity.

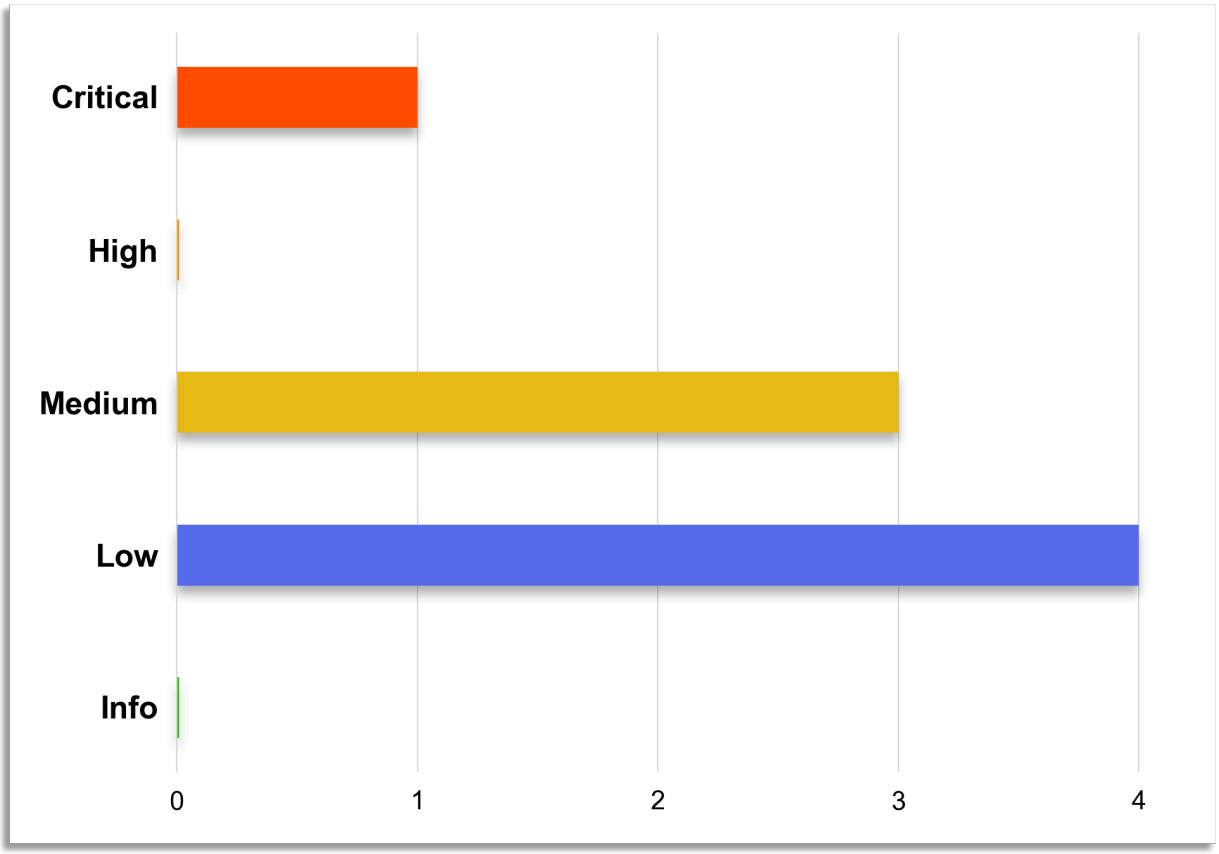

Figure 1: Findings by Severity

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
FYEO-AXELAR-01	Critical	Wrong amount used for XRPL payments
FYEO-AXELAR-02	Medium	No tests implemented for multisig prover contract
FYEO-AXELAR-03	Medium	Payment flags are ignored
FYEO-AXELAR-04	Medium	Use of unwrap & expect in verifier code
FYEO-AXELAR-05	Low	Exponent of 80 will fail in XRPL token amount
FYEO-AXELAR-06	Low	Function update_tx_status accepts empty signer_public_keys
FYEO-AXELAR-07	Low	Shadowed variable prevents event emission
FYEO-AXELAR-08	Low	Unimplemented functionality

Table 2: Findings Overview

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

The Axelar Amplifier is an interchain development platform that enables developers to easily establish connections within the Axelar network ecosystem. A key advantage lies in its cost-effectiveness - developers need to build just one connection to gain access to Axelar's network of interconnected chains. This reduces development overhead while allowing teams to deploy resources across many blockchain ecosystems. The platform supports permissionless setup of new connections and allows developers to enhance existing chain connections with improved security features and better message delivery capabilities. The project is built using Cosmos SDK and includes various smart contracts for managing cross-chain communication and verification.

This implementation extends Axelar Amplifier's capabilities by adding support for the XRP Ledger (Ripple) through three new Cosmos contracts: xrpl-gateway, xrpl-multisig-prover, and xrpl-voting-verifier, enabling seamless integration with the XRPL ecosystem by providing support for dynamic cross-chain messaging and converting assets for interoperability.

Wrong amount used for XRPL payments

Finding ID: FYEO-AXELAR-01
Severity: **Critical**
Status: **Remediated**

Description

The `amount` field in the payment transaction on the XRPL does not indicate the funds received if partial payments are used. A user can send a million XRP in the `amount` field but only deliver 1 XRP in the payment by using a partial payment, still they would be credited with a million XRP.

Proof of Issue

This has been remediated by checking that both amounts match:

File name: ampd/src/xrpl/verifier.rs
Line number: 45

```
let tx_amount = payment_tx.amount.clone();
if tx_meta.unwrap().delivered_amount != Some(tx_amount.clone()) {
    return false;
}
```

Severity and Impact Summary

Users can steal any amount of funds from the bridge.

Recommendation

Make sure to check the `delivered_amount` amount field as described by the documentation.

References

https://xrpl.org/docs/references/protocol/transactions/metadata#delivered_amount
https://xrpl.org/docs/concepts/payment-types/partial-payments#partial-payments-exploit

No tests implemented for multisig prover contract

Finding ID: FYEO-AXELAR-02
Severity: Medium
Status: Remediated

Description

There are some tests for serialization etc. but there are no functional tests implemented for the Multisig Prover contract.

Proof of Issue

There are no tests.

Severity and Impact Summary

Without proper contract testing, issues in the contract may go unnoticed.

Recommendation

Implement tests for this contract to verify its correct behaviour.

Payment flags are ignored

Finding ID: FYEO-AXELAR-03
Severity: Medium
Status: Remediated

Description

The payment flags are ignored. New payment flags could be added that may break the assumptions made by the payment verifier.

Proof of Issue

File name: ampd/src/xrpl/verifier.rs
Line number: 9

```
pub fn verify_message(
    multisig_address: &XRPLAccountId,
    tx: &Transaction,
    message: &XRPLMessage,
) -> Vote {
    // TODO: only allow certain tx flags
    if is_validated_tx(tx) && (is_valid_multisig_tx(tx, multisig_address, message) ||
is_valid_deposit_tx(tx, multisig_address, message)) {
        if is_successful_tx(tx) {
            Vote::SucceededOnChain
        } else {
            Vote::FailedOnChain
        }
    } else {
        Vote::NotFound
    }
}
```

Severity and Impact Summary

Similar to the `tfPartialPayment` flag, other flags may break the assumptions made in this code. This could enable users to steal funds.

Recommendation

Make sure to not accept payments that have unknown flags or flags that are not supported by the verifier.

References

https://xrpl.org/docs/references/protocol/transactions/types/payment#payment-flags

Use of unwrap & expect in verifier node

Finding ID: FYEO-AXELAR-04
Severity: Medium
Status: Remediated

Description

Using functions like unwrap() and expect() in contract code is generally acceptable because failures typically indicate critical bugs that should halt the contract immediately. However, in the context of an API, these functions can lead to denial of service (DoS) situations.

This occurs because any unexpected condition that triggers a failure will cause the entire API request to panic, potentially rendering the API unresponsive.

Proof of Issue

File name: ampd/src/xrpl/verifier.rs
Line number: 95

```
.map(|m| (String::from_utf8(hex::decode(m.memo_type.unwrap()).unwrap()).unwrap(),
m.memo_data.unwrap()))
```

This assumes the decode, utf8 conversion will always work.

Severity and Impact Summary

These could cause panics, which may stop the API service which could potentially stop the service completely.

Recommendation

Instead, errors should be handled gracefully to ensure the API remains robust and continues functioning under unexpected conditions.

Exponent of 80 will fail in XRPL token amount

Finding ID: FYEO-AXELAR-05
Severity: Low
Status: Remediated

Description

An exponent of 80 (= MAX_EXPONENT) will fail in this code, as it will be first incremented to 81 and then fail the second check.

Proof of Issue

File name: packages/xrpl-types/src/types.rs
Line number: 765

```
while mantissa > MAX_MANTISSA.into() && exponent > MIN_EXPONENT {
    if exponent > MAX_EXPONENT {
        return Err(XRPLError::InvalidAmount {
            reason: "overflow".to_string(),
        });
    }
    mantissa /= ten;
    exponent += 1;
}

...

if exponent > MAX_EXPONENT || mantissa > MAX_MANTISSA.into() {
    return Err(XRPLError::InvalidAmount {
```

Severity and Impact Summary

The `MAX_EXPONENT` of 80 could not be used.

Recommendation

Make sure this function is implemented as intended.

Function update_tx_status accepts empty signer_public_keys

Finding ID: FYEO-AXELAR-06
Severity: Low
Status: Remediated

Description

The `update_tx_status` function accepts an empty array and will still pass checks.

Proof of Issue

File name: contracts/xrpl-multisig-prover/src/contract/execute.rs
Line number: 85

```
if xrpl_signers.len() != signer_public_keys.len() {
    return Err(ContractError::SignatureNotFound);
}
```

Severity and Impact Summary

This issue has a low severity, but it can still cause the following problems:

- The system might incorrectly process transactions with no associated signers, leading to unexpected behavior.
- It introduces the risk of potential misuse or logic inconsistencies, especially in edge cases.

Recommendation

Make sure to verify that this isn't empty.

Shadowed variable prevents event emission

Finding ID: FYEO-AXELAR-07
Severity: Low
Status: Remediated

Description

The outer events variable is used to attach events to the response. This variable is however never written, as it is shadowed by a re-declaration inside an inner loop.

Proof of Issue

File name: contracts/xrpl-gateway/src/contract/execute.rs
Line number: 60

```
let events = Vec::new();
for (status, msgs) in &msgs_by_status {
    ...
    let mut events = Vec::new();
    for msg in msgs {
        ...
        events.extend(vec![
            into_route_event(status, its_msg),
            contract_called_event,
        ]);
```

Severity and Impact Summary

No events will be emitted.

Recommendation

Remove the inner declaration, update the outer one.

Unimplemented functionality

Finding ID: FYEO-AXELAR-08
Severity: Low
Status: Remediated

Description

There are several places that appear to be missing functionality.

Proof of Issue

File name: contracts/xrpl-gateway/src/contract/execute.rs
Line number: 267

```
// TODO: register deployment and don't allow duplicate deployments
```

This check appears to be missing for `DeployXrpToSidechain`, `DeployInterchainToken`.

File name: contracts/xrpl-multisig-prover/src/contract/execute.rs
Line number: 36

```
// TODO: check if trust set already set
```

Throughout the codebase, decimal conversion is not fully implemented. Decimals may be hardcoded:

```
// TODO: remove: this conversion is temporary--will be handled by ITS Hub
XRPLPaymentAmount::Drops(drops) => if user_message.destination_chain ==
ChainName::from_str("xrpl-evm-sidechain").unwrap() { // TODO: create
XRPL_EVM_SIDECHAIN_NAME const
    scale_up_drops(drops, 18u8)

let drops = if source_chain == config.xrpl_evm_sidechain_chain_name {
    scale_down_to_drops(amount.into(), 18u8)
// TODO: handle decimal precision conversion
```

```
message: its::Message::DeployInterchainToken {
    token_id,
    name: token_params.name,
    symbol: token_params.symbol,
    decimals: token_params.decimals,
```

The `token_params.decimals` is a user input parameter to the instruction.

Severity and Impact Summary

The documented functionality is important to the correct behavior of the program.

Recommendation

Make sure this logic is implemented correctly. Also add the required unit tests to verify proper behavior.

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.

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