F Y E O

Security Code Review of Axelar - Stacks Integration

Axelar Foundation

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 Stacks.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on December 20 - January 22, 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-AX-STA-01 – Function UpdateSourceGatewayAddress should be behind a build flag.
- FYEO-AX-STA-02 – TLS is not enforced in API communication
- FYEO-AX-STA-03 – Block finality is not specifically considered
- FYEO-AX-STA-04 – Use of unwrap and expect

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 Stacks. 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 repositories at:

https://github.com/axelarnetwork/axelar-amplifier/pull/728

https://github.com/buidly/axelar-stacks-relayer/pull/20

https://github.com/Trust-Machines/axelar-amplifier/pull/6

Remediations were done with commit hashes 304666d59c73f51b310f7619153920a50559cb1f and 017a077a35dd7b0e42aa7b16c4d092faac0b2397.

Files included in the code review

```
PR 728
ampd/
├── Cargo.toml
└── src/
      ├── config.rs
      ├── handlers/
      │    ├── config.rs
      │    ├── mod.rs
      │    ├── stacks_verify_msg.rs
      │    ├── stacks_verify_verifier_set.rs
      │    └── lib.rs
      ├── stacks/
      │    ├── error.rs
      │    ├── http_client.rs
      │    ├── mod.rs
      │    └──verifier.rs
      └── tests/
      └── config_template.toml


PR 20
├── apps/
│    ├── axelar-event-processor/
│    │    └── src/
│    │        ├── approvals.processor.module.ts
│    │        ├── approvals.processor.service.ts
│    │        ├── approvals.processor.spec.ts
│    │        ├── cosmwasm.service.ts
│    │        └── entities/
│    │             └── pending-cosm-wasm-transaction.ts
│    └── stacks-event-processor/
│     └── src/
│           ├── cross-chain-transaction-processor/
│           │    └── cross-chain-transaction.processor.service.ts
```

Files included in the code review

```
│           └── message-approved-processor/
│                 └── message-approved.processor.service.ts
└── libs/common/src/
      ├── api/entities/
      │    └── axelar.gmp.api.d.ts
      ├── assets/
      │    └── axelar-gmp-api.schema.yaml
      └── utils/
      ├── await-success.spec.ts
      └── cache.info.ts


PR 6
├── Cargo.toml
├── contracts/
│    ├── gateway/
│    │    └── src/
│    │       ├── contract.rs
│    │       └── contract/
│    │            └── execute.rs
│    ├── multisig-prover/
│    │    ├── Cargo.toml
│    │    └── src/
│    │       ├── contract.rs
│    │       ├── contract/
│    │       │    ├── execute.rs
│    │       │    └── its.rs
│    │       ├── encoding/
│    │       │    ├── mod.rs
│    │       │    ├── stacks/
│    │       │      ├── execute_data.rs
│    │       │      └── mod.rs
│    │       ├── error.rs
│    │       ├── events.rs
│    │       ├── msg.rs
│    │       └── state.rs
│    └── voting-verifier/
│      ├── Cargo.toml
│      └── src/
│           ├── client.rs
│           ├── contract.rs
│           ├── contract/
│           │    ├── execute.rs
```

Files included in the code review

```
│           │   ├── its.rs
│           │   └── migrations/
│           │     ├── mod.rs
│           │     └── v1_1_1.rs
│           ├── error.rs
│           ├── events.rs
│           ├── msg.rs
│           └── state.rs
└── packages/
    ├── axelar-wasm-std/
    │   ├── Cargo.toml
    │   └── src/
    │     └── address.rs
    ├── gateway-api/src/
    │   └── msg.rs
    └── stacks-clarity/
    ├── Cargo.toml
    └── src/
        ├── common/
        │   ├── address/
        │   │   ├── c32.rs
        │   │   └── mod.rs
        │   ├── codec/
        │   │   ├── macros.rs
        │   │   └── mod.rs
        │   ├── mod.rs
        │   ├── types/
        │   │   └── mod.rs
        │   └── util/
        │     ├── hash.rs
        │     ├── macros.rs
        │     ├── mod.rs
        │     └── pair.rs
        ├── lib.rs
        └── vm/
              ├── analysis/
              │   ├── errors.rs
              │   └── mod.rs
              ├── callables.rs
              ├── contexts.rs
              ├── errors.rs
              ├── mod.rs
              ├── representations.rs
```

Files included in the code review
└── types/ ├── mod.rs ├── serialization.rs └── signatures.rs

Table 1: Scope

Technical Analyses and Findings

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

- 1 finding with MEDIUM severity rating.
- 1 finding with LOW severity rating.
- 2 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

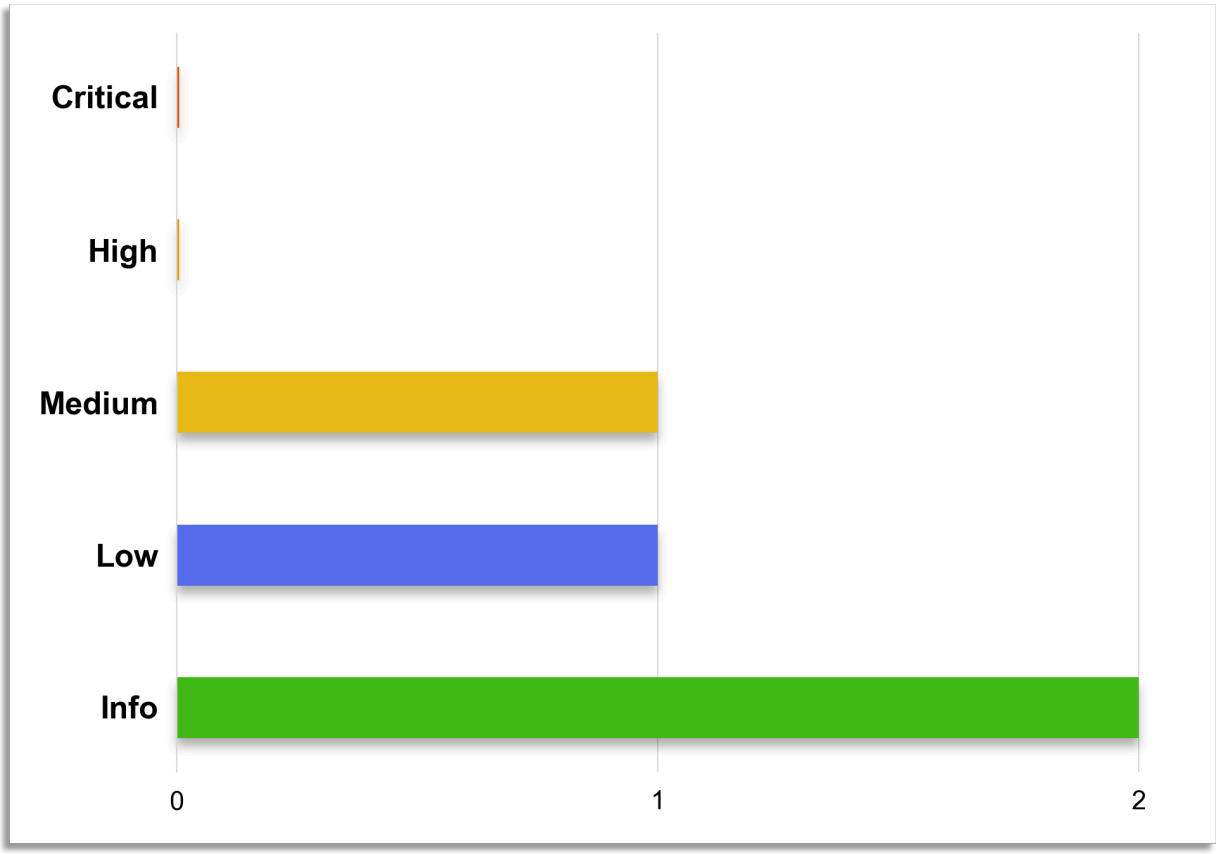

Figure 1: Findings by Severity

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
FYEO-AX-STA-01	Medium	Function UpdateSourceGatewayAddress should be behind a build flag.
FYEO-AX-STA-02	Low	TLS is not enforced in API communication
FYEO-AX-STA-03	Informational	Block finality is not specifically considered
FYEO-AX-STA-04	Informational	Use of unwrap and expect

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 designed to simplify blockchain interoperability by enabling developers to build a single connection to access Axelar's network of interconnected chains. This approach reduces development complexity and costs, empowering teams to allocate resources across multiple blockchain ecosystems effectively.

The platform facilitates permissionless setup of new connections while allowing developers to enhance existing connections with robust security and reliable message delivery. Built on the Cosmos SDK, it incorporates various smart contracts to manage cross-chain communication, asset transfer, and verification processes.

This implementation expands Axelar Amplifier by integrating support for the Stacks blockchain. The integration adds to the Axelar network by incorporating updated contracts and mechanisms for compatibility with the Stacks ecosystem and its encoding scheme. These updates allow for cross-chain messaging and asset interoperability between Stacks and other supported networks.

The codebase is well-organized, making it straightforward for developers to navigate and extend. A notable strength is the inclusion of comprehensive tests, which thoroughly validate the new functionality and ensure reliability across various scenarios. This approach to extensive testing shows the team's commitment to writing secure code.

Function UpdateSourceGatewayAddress should be behind a build flag.

Finding ID: FYEO-AX-STA-01
Severity: Medium
Status: Remediated

Description

There is a function to update the gateway address in the voting verifier.

Proof of Issue

File name: contracts/voting-verifier/src/contract.rs
Line number: 89

```
// TODO: Remove this, only for Devnet testing!
ExecuteMsg::UpdateSourceGatewayAddress {
    new_source_gateway_address,
} => Ok(execute::update_source_gateway_address(
    deps,
    new_source_gateway_address,
)?),
```

Severity and Impact Summary

This function should not be present in release builds as it allows modifying the gateway address.

Recommendation

Put this functionality behind a build flag.

Update:
The entry point has been removed, but the `update_source_gateway_address` function remains unused in the code.

TLS is not enforced in API communication

Finding ID: FYEO-AX-STA-02
Severity: Low
Status: Acknowledged

Description

The service contacts an endpoint via HTTP but does not enforce the use of TLS which may enable man-in-the-middle attacks. This may not be of too much concern on the same host.

Proof of Issue

The endpoint setup / configuration does not check the protocol used to connect and the network of the destination.

Severity and Impact Summary

Depending on the configuration, the API communication may be susceptible to man-in-the-middle attacks, which could be exploited to manipulate data.

Recommendation

Consider adding checks to ensure a safe connection with the endpoint.

Block finality is not specifically considered

Finding ID: FYEO-AX-STA-03
Severity: **Informational**
Status: **Remediated**

Description

The code checks that the `tx_status` is `success`, but does not consider block finality.

Proof of Issue

File name: ampd/src/stacks/http_client.rs
Line number: 110

```
fn is_valid_transaction(tx: &Transaction) -> bool {
    tx.tx_status == *STATUS_SUCCESS
}
```

Severity and Impact Summary

If blocks could roll back, transactions may be un-done.

Recommendation

Make sure to check this is in-line with finality requirements.

Use of unwrap and expect

Finding ID: FYEO-AX-STA-04
Severity: **Informational**
Status: **Remediated**

Description

In the contract there are several places using unwrap or expect. In general they appear to be unlikely to be of much concern.

Proof of Issue

File name: contracts/multisig-prover/src/contract/its.rs
File name: contracts/voting-verifier/src/contract/its.rs
Line number: 20

```
let its_hub_message = its::HubMessage::abi_decode(message_payload.as_slice()).unwrap();
```

File name: packages/stacks-clarity/src/common/address/c32.rs
Line number: 226, 309

```
String::from_utf8(result).unwrap()

...

Ok(String::from_utf8(c32_string).unwrap())
```

File name: packages/stacks-clarity/src/common/codec/mod.rs
Line number: 75

```
.expect("BUG: serialization to buffer failed.");
```

File name: packages/stacks-clarity/src/common/util/macros.rs
Line number: 27, 56

```
u8::try_from(self.as_str().len()).unwrap()

...

Self::try_from(value.to_string()).unwrap()
```

Severity and Impact Summary

These mostly relate to data processing and could trigger if bad input is encountered. It seems unlikely to happen but as the codebase and the protocol evolve, things may change. While unwrap and expect will terminate the execution, the main concern would be a type of DoS situation where certain messges can not be processed and every retry aborts the execution.

Recommendation

Consider improving error handling.

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