# SECURITY ANALYSIS REPORT

Generated By

Backstage Rookie

**Provider: Backstage Rookie**

**contact - swarajdarekar9@gmail.com**

**Client:** Chandrakant Darekar

darekar138@gmail.com

**Scan ID:** 6e641134-7c51-461a-85ba-b91fdb5ec0ec

**Date:** 2026-02-10 14:08:30

**Version:** 1.0.0

# TABLE OF CONTENTS



# EXECUTIVE SUMMARY

This security assessment of the provided codebase for a Technology & SaaS company identified a critical dependency vulnerability related to HTTP request processing. The finding highlights a significant risk stemming from the use of a vulnerable version of `aiohttp`, which could lead to request smuggling attacks. Such vulnerabilities can enable attackers to bypass security measures like firewalls or proxies, potentially leading to unauthorized access, data exfiltration, or other severe impacts on the application's integrity and confidentiality.

## Identified Vulnerabilities

| ID | Title | CVSS | Page |
|---|---|---|---|
| **H1** | Vulnerable `aiohttp` Dependency (Request Smuggling) | 9.1 | |

# METHODOLOGY

## Introduction

This report details the results of a security assessment conducted on the specified repository. The analysis involved a multi-layered approach, combining automated static analysis tools with advanced, AI-driven verification and enrichment to identify potential security vulnerabilities.

## Objective

The primary objective of this assessment was to identify security weaknesses, assess their potential impact, and provide actionable recommendations for remediation to improve the overall security posture of the application.

## Scope

The assessment was performed on the source code of the repository cloned at the time of the scan. The analysis focused on common web application vulnerabilities, insecure coding practices, and dependency risks.

## Systems in Scope

No systems explicitly defined.

## User Accounts

As this was a static source code analysis, no user accounts were provisioned or tested.

# FINDINGS

## H1 – Vulnerable `aiohttp` Dependency (Request Smuggling)

**Severity:** High

**CVSS Score:** 9.1

**CVSS Vector:** CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N

**Target:** Application's HTTP request processing layer

### Overview

The application utilizes a vulnerable version of the `aiohttp` library (3.11.10), which is susceptible to an HTTP request smuggling vulnerability (CVE-2025-53643). This flaw arises from `aiohttp`'s failure to correctly parse trailer sections of an HTTP request. This vulnerability can be exploited if a pure Python version of `aiohttp` is installed or if C extensions are explicitly disabled.

### Details

The `aiohttp` library, specifically version 3.11.10, does not properly handle HTTP trailer sections. An attacker can craft a malicious HTTP request that includes a malformed or unexpected trailer, which `aiohttp` might misinterpret. This misinterpretation can cause the application (and potentially upstream/downstream proxy servers or firewalls) to parse the request boundaries differently, leading to request smuggling. Such an attack allows an attacker to prepend arbitrary requests to other users' connections, bypass security mechanisms, and potentially gain unauthorized access to backend systems or sensitive data.

### Evidence

- **dependency_file:** The application relies on `aiohttp==3.11.10`, which is identified by `pip_audit` as vulnerable to CVE-2025-53643. The vulnerability exists due to incorrect parsing of HTTP trailer sections, enabling request smuggling.

### References

- https://github.com/aio-libs/aiohttp/commit/e8d774f635dc6d1cd3174d0e38891da5de0e2b6a
- https://portswigger.net/web-security/request-smuggling

### Recommendation

- Upgrade `aiohttp` to a version patched against CVE-2025-53643. Refer to the official `aiohttp` GitHub repository for the latest stable and patched releases. - If using `aiohttp` without C extensions (i.e., pure Python version) or with `AIOHTTP_NO_EXTENSIONS` enabled, re-evaluate this configuration given the increased risk, or ensure that all relevant security patches are applied promptly. - Implement robust input validation and sanitization at the application edge to prevent malformed requests from reaching the `aiohttp` server. - Consider implementing HTTP request normalization at proxy/load balancer layers to ensure consistent interpretation of requests across all components.

### Prompt to Solve the Vulnerability:

To address the 'Vulnerable `aiohttp` Dependency (Request Smuggling)' (H1) in your project, you need to upgrade the `aiohttp` library. First, identify where `aiohttp` is declared as a dependency (e.g., in `requirements.txt`,

`pyproject.toml`, or `setup.py`). Then, update the version constraint for `aiohttp` to a patched version. Based on the provided patch link (https://github.com/aio-libs/aiohttp/commit/e8d774f635dc6d1cd3174d0e38891da5de0e2b6a), you should upgrade to at least `aiohttp>=3.11.11` (or the latest stable version available after the patch). For example, if your `requirements.txt` has `aiohttp==3.11.10`, change it to `aiohttp>=3.11.11` or `aiohttp==X.Y.Z` where X.Y.Z is the latest patched version. After updating, run `pip install -r requirements.txt` (or equivalent for your package manager) and thoroughly test your application to ensure compatibility and stability with the new `aiohttp` version.

# ENDPOINT SECURITY ANALYSIS

# METRICS SUMMARY

**Total Findings:** 1

| Severity | Count |
|---|---|
| CRITICAL | 0 |
| HIGH | 1 |
| MEDIUM | 0 |
| LOW | 0 |

# BUSINESS RISK ADVICE

Based on a comprehensive security analysis, here are the prioritized recommendations to enhance the security posture and mitigate identified risks.

## Upgrade `aiohttp` Library to Patch Request Smuggling Vulnerability (Priority: Immediate)

**Description:** Address the high-severity HTTP request smuggling vulnerability (CVE-2025-53643) by immediately updating the `aiohttp` dependency.

**Why it Matters:** Prevents attackers from bypassing security controls, gaining unauthorized access, and exfiltrating data through request smuggling attacks, which directly impacts confidentiality, integrity, and compliance (SOC 2, ISO 27001).

### Recommended Actions:

- Update the `aiohttp` dependency to version `>=3.11.11` or the latest stable patched release.
- Execute comprehensive regression testing across the application to confirm stability and functionality post-upgrade.
- Verify that `aiohttp` C extensions are active; if using a pure Python version or with C extensions disabled, re-evaluate this configuration due to heightened risk.

**Expected Outcome:** Elimination of the known request smuggling vulnerability, significantly enhancing application security and reducing the attack surface.

## Implement Robust HTTP Request Validation and Normalization (Priority: Short-Term)

**Description:** Fortify the application's request processing by introducing stringent input validation and canonicalization at the network edge and application layers.

**Why it Matters:** Mitigates various HTTP-based attacks, including request smuggling and injection, by ensuring only well-formed and legitimate requests reach the application. This is crucial for maintaining data integrity, access control, and compliance (SOC 2, ISO 27001).

### Recommended Actions:

- Deploy or configure Web Application Firewalls (WAF) or API Gateways to enforce strict HTTP protocol validation and anomaly detection.
- Implement HTTP request normalization at load balancer or proxy layers to guarantee consistent interpretation of requests across all infrastructure components.
- Enforce strict parsing rules for all incoming HTTP headers and body content within the application code.

**Expected Outcome:** Enhanced resilience against malformed requests and protocol-level attacks, effectively preventing security control bypass.

## Automate Dependency Vulnerability Scanning and Updates (Priority: Short-Term)

**Description:** Establish a continuous and automated process for monitoring, scanning, and updating all third-party dependencies.

**Why it Matters:** Proactively identifies and remediates known vulnerabilities in external libraries before they can be exploited in production, maintaining a secure software supply chain and supporting compliance with secure development practices.

### Recommended Actions:

- Integrate Software Composition Analysis (SCA) tools into the CI/CD pipeline to perform automated scanning for vulnerable dependencies.
- Configure automated alerts for newly disclosed vulnerabilities affecting any currently used libraries.
- Define and enforce a policy for regular dependency updates, including patch and minor versions, accompanied by controlled testing.

**Expected Outcome:** Reduced exposure to third-party library vulnerabilities and a significantly more secure software supply chain.

## Enhance Security Logging, Monitoring, and Alerting (Priority: Mid-Term)

**Description:** Expand logging capabilities to capture detailed security-relevant events and implement real-time monitoring and alerting for suspicious activities.

**Why it Matters:** Enables early detection of ongoing attacks, provides critical forensic data for effective incident response, and supports compliance requirements for audit trails and continuous security monitoring (SOC 2, ISO 27001).

### Recommended Actions:

- Log all HTTP request anomalies, including parsing errors, header inconsistencies, and suspicious payloads.
- Centralize all security logs into a robust Security Information and Event Management (SIEM) system.
- Establish and configure specific alert rules for patterns indicative of request smuggling attempts or other protocol-level attacks.

**Expected Outcome:** Improved threat detection and response capabilities, leading to enhanced overall system observability and auditability.

## Integrate Security into the Software Development Lifecycle (SSDLC) (Priority: Long-Term)

**Description:** Embed comprehensive security practices throughout the entire software development lifecycle, from initial design to final deployment.

**Why it Matters:** Proactively identifies and mitigates security flaws at earlier, less costly stages of development, fostering a 'secure-by-design' culture and ensuring continuous adherence to industry standards like ISO 27001 and SOC 2.

### Recommended Actions:

- Conduct mandatory security architecture reviews and threat modeling exercises during the design and planning phases of all new features and major changes.
- Implement security code reviews and integrate Static Application Security Testing (SAST) tools as mandatory gates within development workflows.
- Provide continuous and targeted security training for development teams, focusing on OWASP Top 10, secure coding practices, and specific attack vectors like request smuggling.

**Expected Outcome:** Reduced introduction of new vulnerabilities, increased developer security awareness, and a more resilient and compliant application over time.