  
UNIVERSITY OF HERTFORDSHIRE  
School of Physics, Engineering and Computer Science

MSc Cyber Security  
[7COM1039 - Advanced Computer Science Masters Project](https://herts.instructure.com/courses/121503/modules/items/4405384)

2 September 2025

Securing APIs for Cloud Services: Implementation and Security Evaluation Using Free Tools

Name: Ankit Joshi  
Student ID: 22032491  
Supervisor: Dr. Eric Chiejina

**Abstract**

Application Programming Interfaces (APIs) are central to cloud adoption and digital transformation but introduce significant security risks, particularly for Small and Medium-sized Enterprises (SMEs) that lack resources for enterprise-grade solutions. This project employed a design-science methodology to design, deploy, and evaluate a vulnerable FastAPI application on Microsoft Azure Free Tier, embedding five OWASP API Top 10 vulnerabilities: Broken Object Level Authorization (BOLA), Broken Authentication, Mass Assignment, Excessive Data Exposure, and Improper Asset Management. These were mitigated through layered defences including Azure Entra ID (OAuth2 JWT authentication), TLS enforcement, Pydantic input validation, SlowAPI rate limiting, and structured logging. Security testing combined OWASP ZAP for automated scanning, Burp Suite Community Edition for manual payload tampering, and Insomnia for authenticated request validation. Results showed that free-tier tools reliably detected common issues such as sensitive data exposure and deprecated endpoints but failed to identify logic-based flaws like BOLA and Mass Assignment, underscoring the indispensability of manual testing. Severity analysis using CVSS v4.0 scored Broken User Authentication (8.8, High) and Excessive Data Exposure (8.7, High) as the most severe, followed by Mass Assignment (8.6, High), BOLA (7.1, High), and Improper Asset Management (6.9, Medium). The study demonstrates that while free and open-source tools cannot fully replace enterprise-grade solutions, they provide SMEs with a practical, low-cost baseline for API security when integrated with manual validation and disciplined security practices.

**Acknowledgements**

I would like to express my sincere gratitude to my supervisor, **Dr. Eric Chiejina**, for his continuous guidance, insightful feedback, and encouragement throughout the course of this project. His expertise and constructive advice have been invaluable in shaping both the technical direction and academic quality of this work.

I am also grateful to the academic and technical staff at the **University of Hertfordshire**, whose teaching and support during the MSc programme provided the strong foundation necessary to undertake this research.

On a personal note, I wish to thank my family and friends for their unwavering encouragement, patience, and support during the course of my studies. Their belief in me has been a constant source of motivation.

**MSc Final Project Declaration**

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in **Cyber Security** at the University of Hertfordshire (UH).

It is my own work except where indicated in the report.

**I did not use human participants in my MSc Project.**

I hereby **give** permission for the report to be made available on the university website provided the source is acknowledged.

**Ankit Joshi**  
Student ID: **22032491**  
Date: 2 September 2025

# Table of Contents

[Table of Contents 5](#_Toc207897750)

[Chapter 1: Introduction and Overview 8](#_Toc207897751)

[1.1 Background and Context 8](#_Toc207897752)

[1.2 Problem Statement 9](#_Toc207897753)

[1.3 Research Aim and Objectives The aim of this project is to evaluate whether Small and Medium-sized Enterprises (SMEs) can effectively secure cloud-hosted APIs using only free-tier cloud-native services and open-source security tools. The study seeks to provide empirical evidence on the viability of such an approach and to generate practical, SME-focused recommendations for achieving affordable yet meaningful API security. 10](#_Toc207897754)

[1.4 Structure of the Report 10](#_Toc207897755)

[1.5 Ethical, Legal, and Professional Considerations 11](#_Toc207897756)

[Chapter 2: Literature Review 12](#_Toc207897757)

[2.1 Introduction 12](#_Toc207897758)

[2.2 API Security in the Cloud Context 13](#_Toc207897759)

[2.3 OWASP API Security Top 10 14](#_Toc207897760)

[2.4 Security Testing Tools in the Literature 15](#_Toc207897761)

[2.5 Cloud-Native Security Features 16](#_Toc207897762)

[2.6 Evaluation Frameworks 16](#_Toc207897763)

[2.7 Limitations in Existing Literature 17](#_Toc207897764)

[2.8 Research Gap 17](#_Toc207897765)

[Chapter 3: Methodology 19](#_Toc207897766)

[3.1 Research Approach 19](#_Toc207897767)

[3.2 Tools and Technologies 20](#_Toc207897768)

[3.3 Project Phases 21](#_Toc207897769)

[3.3.1 Phase 1: Setup and Development 21](#_Toc207897770)

[3.3.2 Phase 2: Security Implementation 22](#_Toc207897771)

[3.3.3 Phase 3: Vulnerability Testing 22](#_Toc207897772)

[3.3.4 Phase 4: Evaluation 22](#_Toc207897773)

[3.4 Monitoring and Logging 23](#_Toc207897774)

[3.5 System Architecture and Workflow 23](#_Toc207897775)

[3.6 Justification of Methodological Choices 24](#_Toc207897776)

[3.7 Evaluation Criteria 25](#_Toc207897777)

[Chapter 4: API Design and Security Implementation 26](#_Toc207897778)

[4.1 API Architecture and Endpoint Design 26](#_Toc207897779)

[4.2 Implementation of Security Controls 27](#_Toc207897780)

[4.2.1 Authentication and Authorization 27](#_Toc207897781)

[4.2.2 Transport Layer Security (TLS) 28](#_Toc207897782)

[4.2.3 Input Validation (Pydantic Models) 28](#_Toc207897783)

[4.2.4 Rate Limiting (SlowAPI) 28](#_Toc207897784)

[4.2.5 Logging and Monitoring 29](#_Toc207897785)

[4.3 Introduced Vulnerabilities 30](#_Toc207897786)

[4.4 Patched Versions of Vulnerabilities 30](#_Toc207897787)

[4.5 System Workflow with Controls 31](#_Toc207897788)

[Chapter 5: Simulated Vulnerabilities 32](#_Toc207897789)

[5.1 Broken Object Level Authorization (BOLA) – OWASP API1 32](#_Toc207897790)

[5.2 Excessive Data Exposure – OWASP API3 33](#_Toc207897791)

[5.3 Mass Assignment – OWASP API6 35](#_Toc207897792)

[5.4 Broken User Authentication – OWASP API2 36](#_Toc207897793)

[5.5 Improper Asset Management – OWASP API9 37](#_Toc207897794)

[5.6 Summary of Simulated Vulnerabilities 38](#_Toc207897795)

[Chapter 6: Security Testing and Evaluation 39](#_Toc207897796)

[6.1 Testing Environment and Setup 40](#_Toc207897797)

[6.2 Results of Vulnerability Testing 41](#_Toc207897798)

[6.2.1 Broken Object Level Authorization (API1) 41](#_Toc207897799)

[6.2.2 Broken User Authentication (API2) 42](#_Toc207897800)

[6.2.3 Excessive Data Exposure (API3) 43](#_Toc207897801)

[6.2.4 Mass Assignment (API6) 44](#_Toc207897802)

[6.2.5 Improper Asset Management (API9) 45](#_Toc207897803)

[6.3 Summary of Findings 46](#_Toc207897804)

[Chapter 7: Analysis and Discussion 48](#_Toc207897805)

[7.1 Introduction 48](#_Toc207897806)

[7.2 Tool Effectiveness 48](#_Toc207897807)

[7.3 Free-Tier Tools vs Commercial Solutions 49](#_Toc207897808)

[7.4 Severity and Risk Assessment 50](#_Toc207897809)

[7.5 Addressing the Research Gap 51](#_Toc207897810)

[7.6 Limitations of the Study 52](#_Toc207897811)

[7.7 Future Improvements 53](#_Toc207897812)

[7.8 Summary 53](#_Toc207897813)

[Chapter 8: Conclusion and Future Work 54](#_Toc207897814)

[8.1 Introduction 54](#_Toc207897815)

[8.2 Key Findings 54](#_Toc207897816)

[8.3 Critical Reflection 55](#_Toc207897817)

[8.4 Future Work 56](#_Toc207897818)

[8.5 Conclusion 57](#_Toc207897819)

[References 58](#_Toc207897820)

[Appendix A – Source Code (API) 62](#_Toc207897821)

[Appendix B – Vulnerability Testing Evidence 68](#_Toc207897822)

[Appendix C – CVSS v4.0 Scoring Evidence 78](#_Toc207897823)

[Appendix D - ZAP API Scanning Report 81](#_Toc207897824)

[Appendix E - Gantt Chart 92](#_Toc207897825)

# Chapter 1: Introduction and Overview

## 1.1 Background and Context

Application Programming Interfaces (APIs) have become indispensable to the functioning of the modern digital economy. They underpin a wide range of industries, from finance and retail to healthcare and the Internet of Things (IoT), by enabling seamless integration and interoperability between diverse platforms and services (Nandedkar, 2025). Their strategic importance lies not only in supporting day-to-day operations but also in driving innovation, scalability, and the growth of platform-based business models. In fact, API-driven architectures are now central to cloud adoption, mobile applications, and digital transformation initiatives worldwide.

Yet, this ubiquity has also transformed APIs into one of the most attractive and frequently targeted attack vectors in the cybersecurity landscape. Recent studies confirm the scale of the problem: API traffic already accounts for more than 80% of global web traffic, and billions of malicious API calls are logged every month (Akamai, 2024). The Verizon (2025) Data Breach Investigations Report found that 43% of cyberattacks against small businesses are API-related, emphasising that attackers increasingly exploit insecure APIs as entry points. Similarly, the Thales Group (2024) highlights that API threats are not only rising in frequency but also in sophistication, often bypassing traditional perimeter-based security controls such as web firewalls.

The Open Worldwide Application Security Project (OWASP) has long recognised the criticality of these risks through its *API Security Top 10*, which lists recurring issues such as Broken Object Level Authorization (BOLA), Broken User Authentication, Excessive Data Exposure, and Security Misconfiguration (OWASP Foundation, 2023). These vulnerabilities are not abstract or theoretical; they have been at the root of several high-profile breaches. For example, the Facebook–Cambridge Analytica scandal exploited insecure APIs to harvest personal data (MIT IPRI, 2018); T-Mobile (2021) experienced a breach affecting millions of customers due to unauthenticated API endpoints (Associated Press, 2023); and the 2022 Optus data breach in Australia was traced to an API flaw that exposed customer records (Corbado, 2024).

While these incidents often make headlines in the context of large enterprises, SMEs face equal—if not greater—risk exposure. Their limited financial resources and smaller IT teams mean that implementing enterprise-grade solutions such as dedicated API security gateways, premium SIEM platforms, and custom WAF policies is often not feasible. Instead, SMEs tend to rely on cloud-native free tiers and ad hoc open-source tools, creating fragmented and often inadequate defences. As a result, the security gap between large enterprises and SMEs continues to widen, even though SMEs remain attractive targets due to their typically weaker security postures.

This creates a pressing need to explore whether SMEs can realistically achieve a meaningful level of API security without investing in expensive commercial solutions. Specifically, the question arises: can free-tier cloud-native services (e.g., Microsoft Azure Free Tier) combined with open-source security tools (e.g., OWASP ZAP, Burp Suite CE, Insomnia) provide an effective and replicable baseline of protection? Addressing this question is essential not only to close a knowledge gap in the literature but also to deliver practical, evidence-based guidance for resource-constrained organisations navigating today’s API-driven threat landscape.

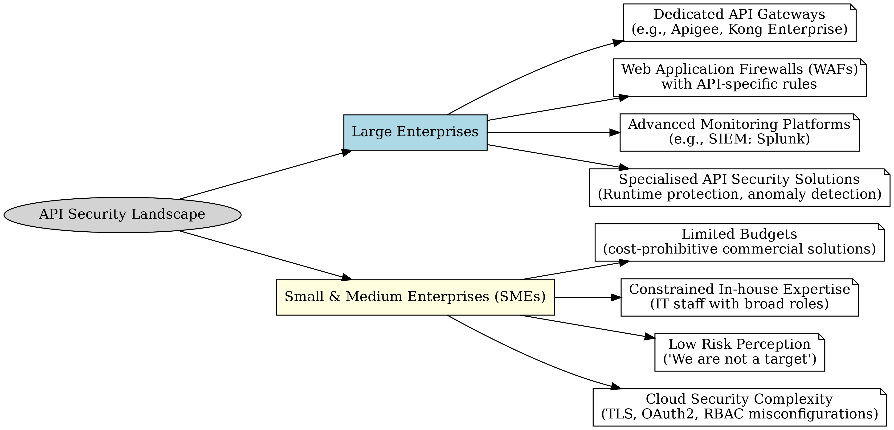


Figure 1.1: Comparative overview of API security capabilities in large enterprises and SMEs.

## 1.2 Problem Statement

Despite the well-documented prevalence of API-related attacks, there remains a striking gap between academic frameworks, industry guidance, and the operational realities of Small and Medium-sized Enterprises (SMEs). Academic research has often focused on enterprise-grade security models, assuming access to advanced infrastructure, API gateways, and dedicated security teams (Ali, Khan and Vasilakos, 2015; Atlidakis, Godefroid and Polishchuk, 2020). Industry reports, while prolific in highlighting the growing number and sophistication of API threats (Akamai, 2024; Verizon, 2025), frequently stop short of empirically evaluating the effectiveness of specific toolchains or cloud-native configurations in SME contexts.

This disjuncture produces a critical research gap: although the importance of API security is universally acknowledged, little empirical evidence exists regarding whether SMEs—operating under severe budgetary and staffing constraints—can achieve credible security postures using only free-tier cloud services and open-source tools. Existing studies tend to evaluate these resources in isolation (e.g., testing OWASP ZAP as a vulnerability scanner or exploring Azure’s free-tier authentication services), but rarely assess their combined, layered effectiveness when integrated into a coherent defensive architecture.

The problem is particularly acute for SMEs because their reliance on APIs is no less than that of larger enterprises, yet their ability to purchase enterprise-grade solutions is far more limited. The economic asymmetry leaves SMEs disproportionately vulnerable to breaches that could result in financial loss, reputational damage, regulatory penalties, or even business closure. As SentinelOne (2025) notes, adversaries increasingly target smaller organisations precisely because of their weaker security postures, leveraging automated attack campaigns to exploit poorly secured APIs.

Therefore, the central problem addressed by this project is the absence of empirically grounded research into whether free-tier cloud-native services and open-source security tools can be effectively combined to mitigate common API vulnerabilities—specifically those outlined in the OWASP API Security Top 10 (2023). Without such evidence, SMEs are left without clear, practical, and affordable pathways to strengthen their API security posture.

## 1.3 Research Aim and Objectives The aim of this project is to evaluate whether Small and Medium-sized Enterprises (SMEs) can effectively secure cloud-hosted APIs using only free-tier cloud-native services and open-source security tools. The study seeks to provide empirical evidence on the viability of such an approach and to generate practical, SME-focused recommendations for achieving affordable yet meaningful API security.

To achieve this aim, the following research objectives were pursued:

* **Design and Deployment of Artefact**  
  Develop and deploy a vulnerable RESTful API using FastAPI, hosted on Microsoft Azure App Service (Free Tier). The artefact reflects realistic API behaviour while embedding vulnerabilities aligned with the OWASP API Security Top 10 (2023), creating a controlled environment for empirical evaluation.
* **Implementation of Layered Security Controls**  
  Secure the artefact using a defence-in-depth model appropriate to SMEs. Controls include authentication and authorisation via Azure Entra ID (OAuth2 with JWTs), TLS enforcement, input validation through Pydantic models, rate limiting using SlowAPI, and structured logging for auditability and incident response.
* **Execution of Comprehensive Security Testing**  
  Conduct vulnerability assessments using a combination of automated and manual tools. OWASP ZAP was applied for passive and active scanning, while Insomnia and Burp Suite Community Edition facilitated manual request manipulation, authenticated testing, and detection of logic-based flaws often missed by automation.
* **Evaluation of Vulnerability Severity and Tool Effectiveness**  
  Analyse identified vulnerabilities using the CVSS v4.0 scoring framework for quantitative severity assessment, while mapping them to OWASP API Top 10 categories. Compare detection coverage across tools through a structured matrix, highlighting strengths and limitations of free-tier and open-source approaches.
* **Critical Reflection and SME-Relevant Recommendations**  
  Reflect on the strengths, weaknesses, and broader implications of relying solely on free-tier and open-source tools for API security. Synthesise findings into practical, evidence-based recommendations for SMEs, focusing on cost-effectiveness, feasibility, and the balance between automation and manual testing.

## 1.4 Structure of the Report

This report is structured as follows:

* **Chapter 2** – Literature Review: Critically evaluates existing research and industry studies on API security, identifying methodological limitations and reinforcing the research gap.
* **Chapter 3** – Methodology: Outlines the design-science methodology, phases of the project, tools used, and evaluation criteria. Includes Figure 1 (System Architecture) and Figure 2 (Testing Workflow).
* **Chapter 4** – API Design and Security Implementation: Details the design and deployment of the FastAPI artefact, including applied security controls.
* **Chapter 5** – Simulated Vulnerabilities: Documents the deliberately introduced vulnerabilities, mapped to OWASP API Top 10 categories, with exploit demonstrations and screenshots.
* **Chapter 6** – Security Testing and Evaluation: Presents results from automated and manual testing, including ZAP scan outputs and manual attack logs.
* **Chapter 7** – Analysis and Discussion: Critically evaluates tool effectiveness and the adequacy of free-tier security measures.
* **Chapter 8** – Conclusion and Future Work: Summarises findings, evaluates whether objectives were met, and proposes extensions.
* **Chapter 9** – References: Complete Harvard-style reference list.
* **Chapter 10** – Appendices: Screenshots, ZAP outputs, sample code, and Gantt chart.

## 1.5 Ethical, Legal, and Professional Considerations

Cybersecurity research necessarily involves ethical and legal responsibilities, particularly when deliberate vulnerabilities are created for testing. To ensure responsible conduct, this project was designed to operate exclusively within a controlled environment. A dedicated Microsoft Azure App Service instance was provisioned solely for the study, with no interaction with production systems, external users, or third-party APIs.

In line with the University’s ethics framework, the project did not involve human participants, personal data, or interventions with real-world infrastructures. As such, formal approval from the Ethics Panel was not required. Nevertheless, the decision and its justification were recorded in project documentation to demonstrate compliance and due diligence.

All data used in the API was synthetic and anonymised (e.g., alice@example.com, bob@example.com). No real credentials, emails, or identifiers were processed, thereby eliminating risks of privacy infringement or misuse.

From a legal perspective, the study adhered to UK data protection principles (UK GDPR, 2018) and avoided any activities that could contravene the Computer Misuse Act (1990). Professional standards were maintained in line with the BCS Code of Conduct (2023), ensuring that the research promoted public trust, transparency, and the responsible disclosure of findings.

# Chapter 2: Literature Review

## 2.1 Introduction

The increasing reliance on Application Programming Interfaces (APIs) within cloud-centric architectures has positioned them as both indispensable enablers of digital transformation and high-value targets for cyber adversaries. APIs now underpin critical business domains such as finance, healthcare, and e-commerce, facilitating seamless integration between distributed services and enabling the growth of platform-driven economies (Nandedkar, 2025). However, this rapid adoption has been paralleled by a surge in API-related security incidents, underscoring their dual role as both business accelerators and potential liabilities (Akamai, 2024; Verizon, 2025).

Industry reports consistently highlight the scale of the threat landscape. Akamai (2024) observed that APIs constitute more than 80% of global web traffic, a figure that reflects not only their ubiquity but also their attractiveness as attack surfaces. The Verizon Data Breach Investigations Report (2025) further identified that 43% of small business cyberattacks involve APIs, emphasising that resource-constrained organisations are disproportionately exposed. Similarly, Thales Group (2024) noted the increasing sophistication of API-specific attack techniques, including exploitation of insecure endpoints and circumvention of traditional perimeter defences.

The Open Worldwide Application Security Project (OWASP) has sought to systematise understanding of these threats through the publication of the API Security Top 10, which catalogues critical weaknesses such as Broken Object Level Authorization (BOLA), Excessive Data Exposure, and Improper Asset Management (OWASP Foundation, 2023). The practical implications of these flaws are evident in real-world breaches: T-Mobile’s 2021 incident, which exposed millions of customer records due to misconfigured API authentication (Associated Press, 2023), and the 2022 Optus breach in Australia, where an unprotected API endpoint leaked sensitive data at scale (Corbado, 2024). These examples illustrate the tangible business, regulatory, and reputational consequences of poor API security practices.

Despite heightened awareness, academic engagement with API security remains fragmented. Much of the scholarly discourse focuses either on general web application security principles or on enterprise-grade solutions that assume the availability of dedicated security operations teams (Ali, Khan and Vasilakos, 2015; Atlidakis, Godefroid and Polishchuk, 2020). While such contributions are valuable, they often fail to account for the realities of Small and Medium-sized Enterprises (SMEs), which typically lack the financial resources or specialist expertise required to implement sophisticated security infrastructures.

Consequently, this chapter undertakes a critical review of existing academic and industry literature, with the dual purpose of (i) situating the present project within the broader API security discourse, and (ii) identifying the research gap that motivates the empirical investigation of free-tier cloud-native services and open-source tools as SME-relevant security solutions.

## 2.2 API Security in the Cloud Context

The adoption of cloud computing has amplified both the opportunities and risks associated with API deployment. Cloud platforms such as Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP) provide elastic scalability and cost efficiency, but they also shift the security burden onto the configuration of APIs that mediate access to cloud-hosted services. APIs act as the “front door” to cloud environments, exposing authentication, data access, and orchestration functions that, if misconfigured, can provide adversaries with a direct path to sensitive assets (Madupati, 2023).

One of the defining characteristics of cloud-based APIs is their exposure across distributed and heterogeneous environments. Unlike traditional monolithic applications, cloud-native services are built upon microservices architectures, with APIs serving as the connective tissue between components. This distributed nature significantly enlarges the attack surface, requiring consistent enforcement of authentication, encryption, and monitoring across multiple services and endpoints (Sindall, 2024). Furthermore, multi-tenancy within public cloud platforms introduces shared responsibility challenges: while providers secure the infrastructure, customers remain accountable for securing APIs, including their configuration, access controls, and vulnerability management (NIST, 2020).

Empirical studies have highlighted recurring misconfigurations in cloud APIs, such as insufficient TLS enforcement, weak identity federation practices, and reliance on default access policies (Hadjimichael and Mitropoulos, 2023). These weaknesses are frequently exploited in real-world attacks. For instance, researchers have demonstrated how overly permissive cloud API keys can enable privilege escalation across entire virtual networks (Krasniqi, 2018). Such risks are exacerbated for SMEs, which often lack the expertise to navigate complex provider-specific security models, leaving APIs as one of the most poorly defended components of their cloud infrastructure (Verizon, 2025).

Industry analyses suggest that SMEs disproportionately rely on free-tier cloud services, owing to financial constraints (NinjaOne, 2025). While these tiers provide baseline security features such as TLS enforcement and basic monitoring, they rarely include advanced protections such as Web Application Firewalls (WAFs), API gateways with schema validation, or behavioural anomaly detection (Raidiam, 2025). As a result, SMEs face a structural disadvantage: they depend on the cloud for innovation and competitiveness but lack access to enterprise-grade tools to defend against API-specific threats.

From an academic perspective, most cloud security research has concentrated on compliance-driven frameworks (e.g., ISO 27001, NIST SP 800-53) and large-scale enterprise environments, where advanced defences are assumed to be feasible (Ali, Khan and Vasilakos, 2015). There is a relative scarcity of empirical investigations into how free-tier cloud-native services can be systematically leveraged to implement layered API defences at minimal cost. This gap is particularly significant given the reliance of SMEs on cloud services for core operations.

This project situates itself within this underexplored niche. By deploying a deliberately vulnerable API on Microsoft Azure Free Tier and securing it with low-cost, readily available mechanisms (e.g., OAuth2 with Entra ID, TLS, input validation, and rate limiting), the study seeks to evaluate whether meaningful API security can be achieved in the absence of enterprise-grade investments.

## 2.3 OWASP API Security Top 10

The Open Worldwide Application Security Project (OWASP) has established the *API Security Top 10* as a widely adopted framework for categorising the most critical API-specific vulnerabilities. Unlike generic web application risks, these categories reflect threats uniquely associated with APIs, including broken object-level access, weak authentication flows, and inadequate inventory management (OWASP Foundation, 2023).

The framework is significant for both industry and academia because it bridges theoretical concepts with empirically observed attack vectors. OWASP does not base its list solely on hypothetical risks; it synthesises real-world breach data, security testing reports, and practitioner experience. Consequently, the API Top 10 has become the de facto baseline for API security assessments across sectors ranging from finance to healthcare (Alhazmi and Malaiya, 2021).

Table 2.1 – Summary of OWASP API Security Top 10 (2023) Source: Adapted from OWASP (2023)

| OWASP Category | Description |
| --- | --- |
| API1: Broken Object Level Authorization (BOLA) | APIs often expose endpoints that reference objects by ID. Without proper access checks, attackers can manipulate IDs to access other users’ data. |
| API2: Broken Authentication | Weak authentication or flawed implementation (e.g., no brute-force protection, plaintext credentials) allows attackers to compromise accounts. |
| API3: Broken Object Property Level Authorization | Attackers manipulate object properties they should not control, leading to privilege escalation or data corruption. |
| API4: Unrestricted Resource Consumption | Lack of rate limiting or resource quotas enables denial-of-service attacks. |
| API5: Broken Function Level Authorization | Attackers escalate privileges by accessing endpoints intended for higher roles (e.g., admin-only functions). |
| API6: Unrestricted Access to Sensitive Business Flows | Abuse of legitimate workflows (e.g., payment, ordering) without rate controls or contextual limits. |
| API7: Server-Side Request Forgery (SSRF) | APIs fetch remote resources without validation, allowing attackers to pivot internally or exfiltrate metadata. |
| API8: Security Misconfiguration | Weak TLS, verbose error messages, missing headers, or unpatched services create attack surfaces. |
| API9: Improper Inventory Management | Deprecated or undocumented endpoints remain exposed, enabling exploitation. |
| API10: Unsafe Consumption of APIs | Trusting external APIs without sanitisation or validation can propagate security flaws. |

While the OWASP API Top 10 provides a crucial taxonomy, its utility is not without criticism. Alhazmi and Malaiya (2021) argue that the framework is descriptive rather than prescriptive: it identifies *what* can go wrong but does not provide detailed guidance on *how* SMEs, with their unique constraints, can realistically mitigate these risks. Moreover, as SentinelOne (2025) highlights, adversaries are increasingly exploiting API-specific business logic flaws—vulnerabilities that may not be fully captured by the Top 10’s technical orientation.

In this project, five representative categories—API1 (BOLA), API2 (Broken Authentication), API3 (Excessive Data Exposure), API6 (Mass Assignment), and API9 (Improper Asset Management)—were selected for simulation. These were chosen because they exemplify both technical misconfigurations and contextual business logic flaws, allowing for a balanced empirical evaluation of tool effectiveness.

## 2.4 Security Testing Tools in the Literature

Existing studies have extensively documented the use of both commercial and open-source tools for API security testing. Enterprise solutions such as Invicti (formerly Netsparker) and Burp Suite Professional provide automated scanning, extensive reporting, and plugin ecosystems (Atlidakis, Godefroid and Polishchuk, 2020). However, these solutions are costly, often placing them outside the financial reach of SMEs.

By contrast, open-source tools such as **OWASP ZAP** and manual clients like **Insomnia** or **Postman** have gained prominence as accessible alternatives. Hadi and Nugroho (2020) demonstrated ZAP’s effectiveness in detecting misconfigurations, weak TLS, and excessive data exposure. However, their study also acknowledged that ZAP consistently fails to detect contextual logic flaws, such as Broken Object Level Authorization (BOLA). Similarly, Hadjimichael and Mitropoulos (2023) note that while manual tools lack automation, they provide vital visibility into token handling, authentication, and endpoint behaviour—capabilities essential for exposing business-logic weaknesses.

Critically, little literature empirically evaluates how these free-tier tools perform *in combination*. Most studies test them in isolation, leaving unanswered questions about whether layered toolchains (automation + manual) can compensate for the shortcomings of each individual tool, particularly in SME deployments.

## 2.5 Cloud-Native Security Features

Cloud platforms provide built-in security features that are frequently underutilised, particularly in SME environments. Microsoft Azure, for example, enforces HTTPS/TLS by default, integrates OAuth2 via Azure Entra ID, and provides monitoring capabilities through Application Insights—all of which are available in the free tier (Microsoft, 2024). Similar features exist in AWS and GCP.

While these services offer baseline protections, Sindall (2024) argues that their effectiveness depends heavily on proper configuration, which SMEs often lack the expertise to apply consistently. Madupati (2023) highlights that misconfigured free-tier cloud services are a leading cause of SME breaches, with developers frequently relying on insecure defaults.

The literature therefore reveals a paradox: although SMEs have access to powerful free-tier controls, their practical security posture depends on whether they can configure, integrate, and monitor these features effectively. This project addresses that paradox by deploying Azure free-tier services in a controlled setting and empirically testing their contribution to API resilience.

## 2.6 Evaluation Frameworks

Two frameworks dominate API vulnerability evaluation:

* **OWASP API Security Top 10** – provides a categorical taxonomy of risks. Its strength lies in industry acceptance and its applicability to both academic and practical contexts. However, Alhazmi and Malaiya (2021) caution that it underrepresents the business-context implications of vulnerabilities.
* **Common Vulnerability Scoring System (CVSS) v4.0** – offers quantitative severity scoring across exploitability, impact, and scope (FIRST.org, 2023). Spring et al. (2021) defend CVSS as a necessary baseline for comparability but concede that its generic metrics often fail to capture SME-specific risks where even “medium” vulnerabilities may have catastrophic outcomes.

This tension underscores the need for a **mixed evaluation strategy**, combining structured frameworks with contextual reflection. Accordingly, this project applies OWASP for categorical mapping, CVSS v4.0 for quantitative scoring, and SME-specific interpretation to bridge the gap between abstract scoring and real-world operational constraints.

## 2.7 Limitations in Existing Literature

Despite the breadth of research into API security, several limitations remain evident across both academic and industry literature.

**1. Enterprise Bias** – Much of the academic discourse assumes enterprise-scale infrastructures, with dedicated security teams, advanced API gateways, and commercial vulnerability scanners (Ali, Khan and Vasilakos, 2015; Atlidakis, Godefroid and Polishchuk, 2020). Such assumptions overlook the constraints of SMEs, where financial, staffing, and technical resources are often minimal. As a result, findings are often misaligned with the realities of SME adoption, leaving smaller organisations with little practical guidance.

**2. Tool Evaluation Gaps** – Although numerous studies evaluate tools like OWASP ZAP, Postman, or Burp Suite, these evaluations are almost always conducted in isolation. There is limited evidence on whether combining these tools—leveraging ZAP’s automation with manual clients like Insomnia—can provide sufficient coverage to address logic flaws such as BOLA or Mass Assignment. This gap is critical, given that real-world attackers typically exploit such contextual weaknesses, which automated scanners alone struggle to detect (Hadi and Nugroho, 2020).

**3. Contextual Blind Spots** – Industry whitepapers frequently report on the prevalence of API breaches (Akamai, 2024; Verizon, 2025), yet they rarely situate findings within the operational constraints of SMEs. Factors such as staff training, monitoring practices, and financial feasibility are underexplored, despite being central to whether security practices can realistically be implemented. Without this context, recommendations risk being aspirational rather than actionable for SMEs.

Collectively, these limitations underscore a persistent gap: while vulnerabilities are well-documented, the *practical effectiveness* of free-tier and open-source tools in addressing them within SME contexts remains largely untested. This justifies the empirical, design-science approach adopted in this project, which constructs a vulnerable artefact, applies layered free-tier controls, and systematically evaluates their resilience against representative API threats.

## 2.8 Research Gap

The literature reviewed makes clear that while the importance of API security is widely recognised, there is a critical lack of *empirical, SME-focused studies* that evaluate affordable, practical defence strategies.

Existing research either:

* **Documents vulnerabilities** (e.g., BOLA, Mass Assignment, Excessive Data Exposure) or
* **Evaluates enterprise-grade security solutions** such as API gateways, commercial WAFs, or advanced threat detection platforms.

However, there is little to no empirical evidence on whether **free-tier cloud-native features** (e.g., Azure’s TLS enforcement, Entra ID authentication) and **open-source testing tools** (e.g., OWASP ZAP, Burp Suite CE, Insomnia) can provide SMEs with sufficient baseline protection against API threats. This is despite industry reports (Akamai, 2024; Verizon, 2025) showing that SMEs are disproportionately targeted, yet underprepared, due to financial and staffing limitations.

This gap is especially problematic because it leaves SMEs with two extremes: aspirational academic models that assume enterprise resources, or industry statistics that highlight risks without offering accessible solutions. Neither adequately addresses the pressing question faced by SMEs: *can free-tier services and open-source tools be realistically combined into an effective, layered defence strategy?*

This project directly responds to that gap by:

1. Designing and deploying a deliberately vulnerable API artefact using FastAPI on Microsoft Azure Free Tier.
2. Introducing representative vulnerabilities aligned to the OWASP API Security Top 10.
3. Applying layered free-tier and open-source security controls (OAuth2/JWT, TLS, input validation, rate limiting).
4. Empirically testing the artefact using a triangulated toolset (ZAP, Insomnia, Burp Suite CE).
5. Evaluating severity with CVSS 4.0 and mapping results to OWASP categories.

By empirically assessing whether free-tier resources can deliver meaningful API protection, this study addresses a clear academic and industry blind spot. Its findings aim to provide SMEs with evidence-based, actionable guidance that balances cost-effectiveness with practical security resilience.



Figure 2.1 – OWASP API Security Top 10 Overview: A visual diagram mapping vulnerabilities to example exploits.

# Chapter 3: Methodology

## 3.1 Research Approach

This project employs a **design-science methodology**, a recognised paradigm in computing research that emphasises the **creation, intervention, and evaluation of artefacts** (Hevner et al., 2004). The artefact in this study is a deliberately vulnerable, cloud-hosted FastAPI application, subsequently secured through layered defences. This methodology was chosen over alternatives such as case study or survey research because it allows both the **technical implementation of defences** and the **empirical evaluation of tool effectiveness** within a realistic SME context.

The research was operationalised through four iterative phases:

1. **Build Phase** – Development of a FastAPI application with realistic endpoints, deliberately seeded with vulnerabilities aligned to the OWASP API Security Top 10 (2023).
2. **Intervention Phase** – Application of layered mitigations, including Azure Entra ID authentication (OAuth2 with JWTs), TLS enforcement, schema validation with Pydantic, and rate limiting via SlowAPI.
3. **Evaluation Phase** – Execution of both automated and manual testing using OWASP ZAP, Insomnia, and Burp Suite Community Edition, with results systematically captured and categorised.
4. **Reflection Phase** – Analysis of outcomes using CVSS v4.0 severity ratings and OWASP category coverage, assessing whether free-tier defences provide sufficient resilience for SMEs.

By structuring the study in this manner, the project advances beyond a **proof-of-concept** and instead provides a **rigorously validated artefact**, directly addressing the research gap identified.



Figure 3.1 – Iterative Design-Science Phases.

## 3.2 Tools and Technologies

A combination of cloud services, development frameworks, and open-source security tools were employed to design, deploy, and evaluate the artefact. These were deliberately selected to reflect the cost-sensitive and resource-constrained conditions of SMEs, while maintaining technical robustness and academic rigour.

**Cloud Hosting.** The application was deployed on Microsoft Azure App Service (Free Tier). Azure was chosen due to its enforced TLS configuration, seamless scalability, and alignment with SME adoption patterns, where free-tier credits or trial offerings are often relied upon in place of enterprise-grade subscriptions. This environment provided both realism and cost-effectiveness.

**Continuous Integration and Deployment (CI/CD).** GitHub Actions was used to establish a fully automated CI/CD pipeline. This ensured reproducibility of deployments, full traceability of iterative changes, and alignment with modern DevOps practices. By leveraging GitHub’s free-tier workflow runners, the project reproduced a realistic SME deployment model without incurring additional cost.

**Framework and Development Tools.** The API was implemented using FastAPI (Python), a modern REST framework noted for its lightweight performance, asynchronous support, and integrated OpenAPI documentation. FastAPI enabled rapid prototyping of both secure and deliberately vulnerable endpoints. Configuration management adhered to best practices: environment variables were handled locally via .env files and mapped to Azure App Settings in production, ensuring separation of concerns and security of sensitive keys.

Security Controls. A layered defence model was applied, designed to simulate practical SME strategies that avoid reliance on costly enterprise suites:

* **Authentication and Authorization** via **Azure Entra ID**, employing OAuth2.0 and JSON Web Token (JWT) validation against Microsoft’s JWKS endpoint.
* **Transport Security** enforced through Azure TLS (1.2+), ensuring encrypted client–server communications.
* **Rate Limiting** implemented with the SlowAPI package to mitigate brute-force and denial-of-service attempts.
* **Input Validation** enforced through Pydantic schemas, using strict typing and field restrictions to block injection and mass assignment vulnerabilities.
* **Logging and Monitoring** integrated via Python’s logging library with a rotating file handler, supporting auditability and forensic traceability in line with professional standards.

**Testing Tools.** The evaluation stage combined automation with manual analysis, reflecting the limitations and strengths of free-tier toolchains:

* **OWASP ZAP** was employed for both passive scanning (e.g., TLS verification, header checks, and sensitive data exposure) and active scanning (e.g., injection and brute-force attempts).
* **Insomnia** provided authenticated API testing, including JWT injection and endpoint validation under Azure Entra ID workflows.
* **Burp Suite Community Edition** facilitated manual payload manipulation and business logic exploitation, including the confirmation of Mass Assignment and Broken Object Level Authorization (BOLA) flaws.

Together, this toolset created a realistic yet cost-effective experimental environment, enabling an empirical evaluation of whether SMEs can rely on free-tier resources for baseline API resilience. By combining automation with human-driven testing, the project ensured both breadth and depth of assessment, directly addressing the research gap identified. To support reproducibility and compliance with submission requirements, the complete source code, dependency specifications, and CI/CD configuration files are included in the artefact submission, with a supplementary GitHub repository maintained for transparency.

## 3.3 Project Phases

The project was executed through four structured phases, designed to ensure methodological rigour, traceability, and alignment with the research objectives. Each phase was deliberately scoped to balance technical realism with the constraints faced by SMEs.

### 3.3.1 Phase 1: Setup and Development

A FastAPI skeleton was developed to simulate a realistic API service, incorporating endpoints for user authentication, data submission, and administrator-only access. To enable controlled security testing, five deliberate vulnerabilities were embedded—Broken Object Level Authorization (BOLA), Mass Assignment, Excessive Data Exposure, Broken Authentication, and Improper Asset Management—all mapped to the OWASP API Security Top 10 (2023).

Local execution was facilitated using Uvicorn, while deployment was automated via a GitHub Actions pipeline, ensuring reproducibility and continuous integration. Sensitive variables were handled securely using .env files for local development and Azure App Settings in the cloud deployment, adhering to industry-standard configuration management practices.

### 3.3.2 Phase 2: Security Implementation

Layered defensive controls were incrementally applied to the API, simulating realistic SME adoption of affordable countermeasures. Authentication was integrated via Azure Entra ID (OAuth2.0), with token validation against Microsoft’s JWKS endpoint. TLS was enforced on all connections through Azure App Service defaults, ensuring secure transport.

To mitigate common threats, Pydantic schemas were employed for strict input validation, and SlowAPI rate limiting was applied to high-risk endpoints (e.g., login, administrator routes). Structured logging mechanisms were configured to capture authentication attempts, request activity, and exception handling, enabling both monitoring and forensic traceability.

### 3.3.3 Phase 3: Vulnerability Testing

The security posture of the artefact was evaluated using a hybrid testing strategy that combined automation with manual validation.

* **Insomnia** was used to verify endpoint behaviour under authenticated and unauthenticated conditions.
* **OWASP ZAP** conducted passive scans (e.g., data exposure, missing headers) and active scans (e.g., injection, fuzzing).
* **Burp Suite Community Edition** facilitated manual manipulation of API requests, such as JSON payload modification to exploit Mass Assignment and validate logical flaws like BOLA.

This multi-tool approach provided both breadth (via automated coverage) and depth (via manual exploitation), reflecting SME testing practices where full reliance on commercial scanners is infeasible.

### 3.3.4 Phase 4: Evaluation

Test results were systematically mapped to the OWASP API Security Top 10 (2023) categories to ensure consistency of categorisation. Each identified vulnerability was further assessed using the CVSS v4.0 scoring framework, providing a quantitative severity classification.

Tool effectiveness was captured in a comparative detection matrix, highlighting gaps (e.g., ZAP successfully detecting excessive data exposure but failing to identify BOLA). Evidence was logged through Azure application logs, ZAP reports, and manual test notes, reviewed weekly against project milestones to ensure alignment with the research timeline.

## 3.4 Monitoring and Logging

To maintain traceability, accountability, and research rigour, multiple monitoring and logging mechanisms were integrated across both development and deployment stages. These mechanisms served a dual purpose: (i) ensuring reproducibility of experimental findings, and (ii) generating verifiable evidence to support subsequent evaluation and analysis.

* **Azure Application Logs** continuously captured API requests, authentication attempts, and error events within the production environment. This provided a reliable audit trail of cloud-based activity.
* A **Rotating File Logger** was configured in the local development environment to capture runtime events while preventing uncontrolled log growth, thereby supporting reproducibility and forensic traceability.
* **Tool Outputs** from OWASP ZAP and Insomnia were exported and archived as structured evidence. These outputs were subsequently incorporated into the appendices, ensuring transparent linkage between raw test data and the analysis presented in Chapter 6.
* **Progress Tracking** was formalised through weekly logs, which were cross-referenced against the Gantt chart milestones. This ensured alignment with the planned research trajectory and provided documented evidence of time management.

By combining automated system-level logging with structured research documentation, the project safeguarded both the technical transparency and academic integrity of its findings. These measures ensured that all claims made in the evaluation phase could be substantiated by verifiable artefacts.

## 3.5 System Architecture and Workflow

GitHub Actions CI/CD

Azure App Service (FastAPI API

Developer (GitHub)

Azure App Settings

Azure Entra ID (OAuth2 JWT

Client Tools: Insomnia | ZAP

Figure 3.2 – System Architecture of the API Security Evaluation Framework

This diagram shows the relationship between the development environment, cloud deployment, authentication provider, and testing tools.

↓

Response

FastAPI App on Azure

Insomnia

API Request

Logging Layer

ZAP Passive Scan

ZAP Active Scan

Burp Suite Manual Testing

Figure 3.3 – Testing Workflow for Vulnerability Assessment

This illustrates the testing pipeline, highlighting how requests flow through different tools and where vulnerabilities are detected.

## 3.6 Justification of Methodological Choices

The methodological decisions underpinning this project were guided by the dual imperative of technical rigour and SME-relevant feasibility. Each choice was critically evaluated in light of the research objectives and the constraints identified in Chapter 2.

* **FastAPI.** Selected for its lightweight architecture, asynchronous execution model, and API-first design, FastAPI enabled rapid prototyping of both secure and deliberately vulnerable endpoints. Unlike heavier frameworks such as Django, it allowed for focused experimentation on API security without unnecessary overhead, thereby aligning with SME resource constraints.
* **Azure Free Tier.** Deployment on Microsoft Azure’s Free Tier mirrored the financial realities of SMEs, which frequently rely on cloud credits and free service tiers. While enterprise environments might employ dedicated infrastructure or premium security suites, the Azure Free Tier provided a realistic yet cost-feasible platform that enforced TLS by default and allowed scalability without additional expenditure.
* **OWASP ZAP and Insomnia.** This combination balanced automation and manual testing. ZAP offered breadth through automated scanning, while Insomnia provided depth via authenticated request manipulation. Commercial alternatives such as Burp Suite Professional or Invicti were excluded deliberately to maintain SME relevance, ensuring that the evaluation reflected the practical limitations of free-only toolchains.
* **Design-Science Methodology.** Adopted because it uniquely supports both artefact construction (the deliberately vulnerable API) and empirical evaluation (the testing of layered, no-cost defences). Unlike case studies or surveys, design-science enabled the generation of verifiable, reproducible evidence addressing the identified research gap.

Collectively, these methodological choices ensured that the project not only remained technically robust but also contextually aligned with SME realities, providing an empirically grounded exploration of whether SMEs can achieve a baseline of API resilience without recourse to enterprise-grade investment.

## 3.7 Evaluation Criteria

To ensure methodological rigour and transparency, the evaluation of vulnerabilities and tool performance employed a triangulated framework of three complementary criteria.

First, vulnerabilities were mapped against the OWASP API Security Top 10 (2023), ensuring that findings were contextualised within an industry-recognised taxonomy of API threats. This categorical mapping provided consistency and facilitated comparability with prior research and best-practice guidance.

Second, each vulnerability was assigned a CVSS v4.0 severity score, offering a quantitative assessment across exploitability, impact, and scope. Although CVSS has recognised limitations in accounting for business context, its structured scoring ensured reproducibility and provided a baseline for severity classification.

Finally, a detection matrix was employed to compare the performance of OWASP ZAP, Burp Suite CE, and Insomnia. This cross-tool analysis highlighted detection gaps, distinguishing between vulnerabilities reliably identified through automation and those requiring manual intervention.

Together, these criteria provided a multidimensional evaluation—categorical, quantitative, and comparative—ensuring that the findings were both empirically grounded and directly relevant to SMEs.

Table 3.1 – Mapping of Simulated Vulnerabilities to OWASP API Top 10 Categories and CVSS v4.0 scores.

| **Vulnerability (OWASP API Top 10)** | **Endpoint(s) Tested** | **Expected Exploit / Behaviour** | **ZAP Detection** | **Burp Suite CE Detection** | **Insomnia Validation** | **CVSS 4.0 Severity (Base)** |
| --- | --- | --- | --- | --- | --- | --- |
| **BOLA (Broken Object Level Authorization)** | /users/{id} | Any authenticated user can retrieve another user’s record | No | Yes (manual) | Yes | **High (7.1)** |
| **Excessive Data Exposure** | /leak-data, /public-profile | API returns sensitive fields (passwords, tokens, debug info) | Yes (PII alert) | Yes (manual) | Yes | **High (8.7)** |
| **Mass Assignment** | /mass-assign | Attacker can inject role, is\_admin fields | No | Yes (manual) | Yes | **High (8.6)** |
| **Broken Authentication** | /login | Weak password check, no brute force protection | Warn (weak TLS) | Yes (manual brute-force) | Yes (bypass) | **High (8.8)** |
| **Improper Asset Management** | /v1/legacy-info | Deprecated but still accessible endpoint | Warn (exposed endpoint) | Yes (manual) | Yes | **Medium (6.9)** |

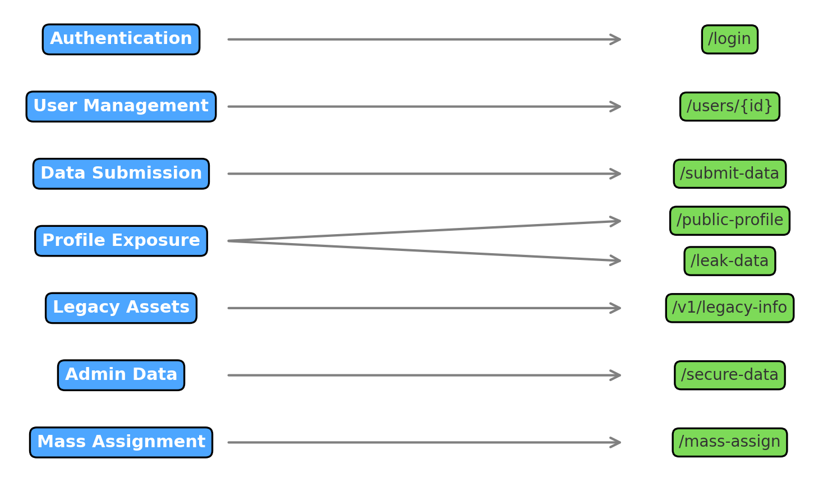
# Chapter 4: API Design and Security Implementation

## 4.1 API Architecture and Endpoint Design

The empirical artefact was implemented as a RESTful API using FastAPI, a lightweight Python framework chosen for its asynchronous execution model and native support for OpenAPI documentation. Deployment was orchestrated via a GitHub Actions CI/CD pipeline to Microsoft Azure App Service (Free Tier), reflecting realistic SME adoption patterns of cloud credits and free-tier services while ensuring reproducibility and operational transparency.

The architecture adhered to modular design principles, with endpoints logically segmented into three primary categories: user management, authentication, and data submission. This segmentation mirrored typical enterprise API structures while facilitating controlled vulnerability injection and targeted evaluation.

The endpoint set was dual-purpose: it simulated legitimate application behaviour while embedding specific vulnerabilities aligned with the OWASP API Security Top 10 (2023). This intentional duality provided a realistic yet bounded testing environment, ensuring that vulnerabilities could be systematically exploited and subsequently patched in a manner reflective of practical SME conditions.

****Figure 4.1 – API Endpoint Categories

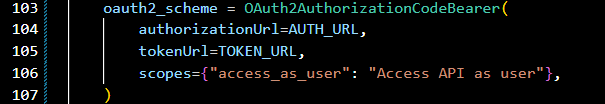
## 4.2 Implementation of Security Controls

To mitigate the deliberately introduced threats, a layered defence-in-depth strategy was implemented. Controls were carefully selected to reflect realistic practices accessible to SMEs, relying exclusively on free-tier Azure services and open-source components while still adhering to recognised security principles.

### 4.2.1 Authentication and Authorization

Authentication and authorization were achieved through Azure Entra ID, employing the OAuth2.0 authorization code flow to enable standards-based, federated identity management. API endpoints were protected using JSON Web Tokens (JWTs), cryptographically validated against Microsoft’s JSON Web Key Set (JWKS) endpoint to guarantee token integrity and authenticity.

This configuration supported the enforcement of Role-Based Access Control (RBAC), ensuring that sensitive routes (e.g., administrator-only endpoints) could only be accessed by authenticated principals with the requisite privileges. By integrating OAuth2.0 with JWT validation, the artefact addressed OWASP API2: Broken Authentication, directly mitigating risks stemming from weak credentials, token tampering, or absent identity checks.



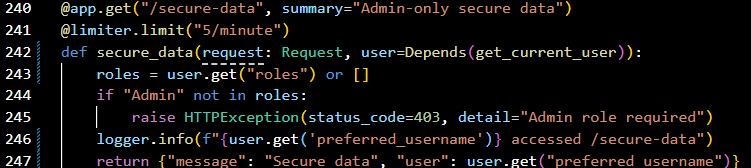


Figure 4.2 – Example implementation of JWT validation and RBAC enforcement in FastAPI using Azure Entra ID.

This implementation exemplifies how SMEs can leverage no-cost cloud-native identity services to enforce least privilege access, thereby achieving a high degree of assurance without reliance on expensive commercial identity platforms.

### 4.2.2 Transport Layer Security (TLS)

All endpoints of the artefact were served exclusively over HTTPS, with *Azure App Service* enforcing TLS 1.2 and above by default. This ensured that all communication between clients and the API was encrypted, protecting against eavesdropping and replay attacks.

The configuration was validated using *OWASP ZAP*, which confirmed that plain HTTP requests were rejected and automatically redirected to HTTPS. Additional verification was performed through certificate inspection, the results of which are presented in *Appendix B*.

### 4.2.3 Input Validation (Pydantic Models)

Robust input validation was enforced using Pydantic schemas, which imposed strict structural definitions for all incoming JSON payloads. By mandating explicit field declarations and type constraints, Pydantic ensured that only syntactically valid and semantically correct data could be processed, while extraneous or malicious fields were systematically rejected.

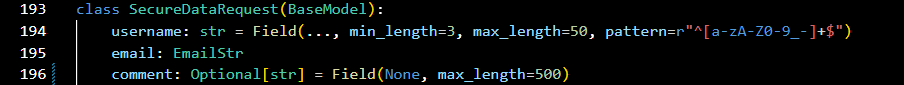


Figure 4.3 - Secure Data Request Schema

This schema-based approach provided defence-in-depth against multiple vulnerability classes. It directly mitigated OWASP API8: Injection, by preventing untrusted inputs from being interpreted beyond their intended type, and further reduced exposure to OWASP API6: Mass Assignment, by disallowing undeclared fields (e.g., role, is\_admin).

For SMEs, this demonstrates how lightweight, open-source frameworks can be used to achieve structural data integrity without the need for costly enterprise data validation middleware.

### 4.2.4 Rate Limiting (SlowAPI)

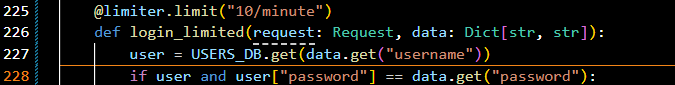
To reduce exposure to brute-force and denial-of-service (DoS) attempts, rate limiting was enforced using the SlowAPI library. This package integrates natively with FastAPI’s asynchronous request handling, enabling lightweight request **throttling** without reliance on external infrastructure.

Figure 4.4 – Implementation of rate limiting on the /login endpoint using SlowAPI.

By constraining repeated login attempts within a fixed time window, this control implemented an effective lockout threshold, thereby deterring automated password-guessing attacks. It also minimised the risk of resource exhaustion caused by malicious clients, indirectly hardening the availability of the API service.

This measure aligns directly with OWASP API2: Broken Authentication countermeasures and exemplifies a cost-feasible control that SMEs can readily adopt to strengthen authentication resilience without investment in enterprise-grade security gateways.

### 4.2.5 Logging and Monitoring

Comprehensive logging and monitoring mechanisms were configured to ensure visibility, traceability, and forensic readiness across both development and production environments.

In the local development environment, a Rotating File Handler was employed to capture API activity. This implementation enforced a maximum log size of 1 MB with up to three retained backups, thereby preventing uncontrolled log growth while maintaining historical visibility for reproducibility and auditability.

For the production deployment, Azure App Service logging was enabled to capture real-time activity, including authentication attempts, rate-limit triggers, and data submission events. These logs provided a reliable audit trail of operational behaviour, supporting both live monitoring and retrospective forensic analysis.

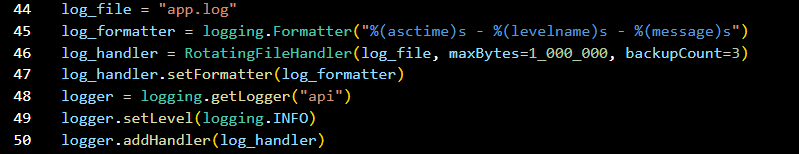


Figure 4.4 – Rotating file logger implementation for local monitoring and auditability.

This dual configuration delivered both local and cloud-based evidence collection, ensuring that security-relevant events could be systematically captured, reviewed, and correlated with vulnerability testing outcomes. The approach also aligns with OWASP API8: Security Misconfiguration, which highlights insufficient logging as a critical weakness, and demonstrates a low-cost monitoring strategy accessible to SMEs without reliance on enterprise SIEM platforms.

## 4.3 Introduced Vulnerabilities

To enable a controlled and systematic evaluation, five vulnerabilities were **intentionally embedded** within the artefact. Each maps directly to a category from the **OWASP API Security Top 10 (2023)** and was selected to replicate misconfigurations frequently observed in **SME deployments,** where limited expertise and resource constraints heighten exposure.

1. **Broken Object Level Authorization (BOLA) – OWASP API1**
   * **Endpoint:** /users/{user\_id}
   * **Description:** Permits any authenticated user to retrieve records belonging to other users. This simulates an **access control bypass**, highlighting the risk of insufficient ownership and role enforcement.
2. **Excessive Data Exposure – OWASP API3**
   * **Endpoint:** /leak-data
   * **Description:** Returns sensitive attributes including plaintext or hashed passwords, tokens, and debug metadata. This replicates **data leakage through overexposed API responses,** a common flaw where developers fail to enforce field-level data filtering.
3. **Mass Assignment – OWASP API6**
   * **Endpoint:** /mass-assign
   * **Description:** Accepts unvalidated JSON payloads, allowing attackers to inject arbitrary fields such as role or is\_admin. This creates a **privilege escalation vector**, reflecting misuses of object-binding frameworks often seen in under-secured SME applications.
4. **Broken User Authentication – OWASP API2**
   * **Endpoint:** /login
   * **Description:** Implements weak credential validation with plaintext password comparison, omits password hashing, and by default permits unlimited login attempts. This demonstrates susceptibility to **credential stuffing and brute-force compromise**.
5. **Improper Asset Management – OWASP API9**
   * **Endpoint:** /v1/legacy-info
   * **Description:** Exposes a **deprecated and undocumented endpoint** that reveals internal system information, illustrating how abandoned APIs contribute to an expanded attack surface.

## 4.4 Patched Versions of Vulnerabilities

To empirically evaluate the effectiveness of layered security strategies, patched versions of the deliberately introduced vulnerabilities were implemented. Each remediation aligned with recognised best practices and emphasised SME-feasible controls, ensuring that the artefact evolved from an intentionally insecure baseline into a demonstrably hardened API.

* **BOLA Mitigation (OWASP API1).** Implemented fine-grained ownership checks, ensuring that users could only retrieve their own records. Broader access was restricted to accounts with explicit administrative privileges, thereby enforcing least-privilege access control.
* **Mass Assignment Mitigation (OWASP API6).** Pydantic models were configured with extra="forbid", enforcing strict schema whitelisting and blocking injection of untrusted fields such as role and is\_admin. This eliminated the privilege escalation vector inherent in automatic object binding.
* **Excessive Data Exposure Mitigation (OWASP API3).** Sensitive attributes—including passwords, tokens, and debugging metadata—were systematically excluded from API responses. Only essential, non-sensitive fields (e.g., username, role) were returned, reducing risk of data leakage and supporting data minimisation principles.
* **Broken Authentication Mitigation (OWASP API2).** The /login endpoint was reinforced with SlowAPI rate limiting and enhanced error logging, reducing exposure to brute-force attempts and providing forensic visibility. This partially addressed weak authentication handling, while further work (e.g., secure password hashing) was identified as future improvement.
* **Improper Asset Management Mitigation (OWASP API9).** Deprecated endpoints now return an HTTP 410 Gone response, accompanied by guidance to /v2/info. This approach ensured explicit deprecation handling and reduced the attack surface by preventing exploitation of undocumented or legacy endpoints.

Collectively, these remediations transformed the artefact into a layered defence implementation, aligning directly with OWASP’s recommended countermeasures and establishing a robust basis for the empirical evaluation of free-tier tool effectiveness in SME-relevant contexts.

## 4.5 System Workflow with Controls

The secured artefact followed a defence-in-depth model, where multiple security layers were applied at different stages of the API request lifecycle. This workflow ensured that even if one control were bypassed, subsequent layers would continue to provide protection.

Client Request

Azure TLS Enforcement

FastAPI App

Input Validation (Pydantic)

Authentication (JWT via Azure Entra ID)

Authorization (RBAC check)

Rate Limiting (SlowAPI)

Response

Logged to Azure + Local Logs

Figure 4.5 – Secure API Request Lifecycle

Illustrating the integration of layered security controls, this diagram shows how incoming client requests traverse authentication, TLS encryption, input validation, rate limiting, and logging before reaching protected application logic.

# Chapter 5: Simulated Vulnerabilities

This chapter documents the deliberate introduction and exploitation of vulnerabilities within the FastAPI artefact, each aligned to the OWASP API Security Top 10 (2023). The purpose of embedding these weaknesses was to replicate realistic misconfigurations and insecure design patterns that frequently occur in Small and Medium-sized Enterprises (SMEs), thereby providing a valid basis for empirical evaluation.

For each vulnerability, the discussion follows a consistent structure:

* **Exploit Demonstration** – showing how the vulnerability was triggered in practice through tools such as Insomnia, Burp Suite Community Edition, and OWASP ZAP.
* **Tool Evidence** – highlighting which tools detected or failed to detect the issue, providing insight into tool effectiveness.
* **Security Implications** – analysing the potential consequences of exploitation in SME contexts.
* **Critical Reflection** – evaluating the significance of the vulnerability against the literature, particularly the limitations of free-tier automated scanners and the necessity of manual validation.

This systematic approach not only demonstrates the presence and remediation of specific flaws but also enables critical evaluation of how free-tier and open-source tools perform in practice when applied to API security testing.

## 5.1 Broken Object Level Authorization (BOLA) – OWASP API1

The endpoint /users/{user\_id} was initially implemented without ownership validation, allowing any authenticated user to access records belonging to other users. This represents a typical case of Broken Object Level Authorization (BOLA), identified as API1 in the OWASP API Security Top 10 (2023). A request was issued via Insomnia using the credentials of a non-privileged user (user1). Despite being authenticated as Alice, the client was able to access another user’s data (user2), as shown below:

* **Request:**

GET /users/user2

Header: X-Dev-User: user1|User

* **Response:**

{

"id": "user2",

"username": "bob",

"email": "bob@example.com",

"role": "User",

"password": "password"

}



Figure 5.1 – Insomnia request demonstrating cross-user data retrieval (/users/{user\_id}).

This confirmed that Alice could retrieve Bob’s sensitive record, including credentials (Figure 5.1). From a security standpoint, an attacker exploiting such a flaw could enumerate user IDs to harvest personal data at scale, compromise confidentiality by accessing credentials, and escalate attacks by combining retrieved information with credential stuffing or phishing campaigns.

Following remediation, ownership checks were introduced so that non-admin users are restricted to their own records, while administrative accounts retain broader access privileges. Re-tests confirmed that attempts to retrieve another user’s data returned an HTTP 403 response, effectively preventing lateral data exposure.

The CVSS v4.0 assessment classified this issue as High severity with a score of 7.1 (see Appendix C, Figure C.1). While OWASP ZAP partially flagged the response as “Excessive Information Exposure,” it failed to identify the underlying absence of access control. This aligns with Hadi and Nugroho (2020), who emphasise that automated scanners remain inadequate for detecting contextual vulnerabilities such as BOLA. For SMEs, the implication is critical: reliance on free-tier automated tools alone leaves them exposed to severe risks. Manual validation, even using lightweight clients such as Insomnia, remains indispensable to uncovering these logic-based flaws.

## 5.2 Excessive Data Exposure – OWASP API3

The /leak-data endpoint was implemented without adequate filtering of response attributes, resulting in the disclosure of sensitive details including password hashes, authentication tokens, and debug metadata such as IP addresses and request timings. Manual exploitation using Insomnia demonstrated that a simple unauthenticated request was sufficient to retrieve this information, as shown below:

* **Request**

GET /leak-data\

* **Response**

{

"id": "user2",

"username": "bob",

"email": "bob@example.com",

"password": "pbkdf2\_sha256$fake-hash",

"role": "Admin",

"auth\_token": "xyz.jwt.token",

"debug\_info": {"request\_ip": "192.168.1.5", "processing\_time": "120ms"}

}

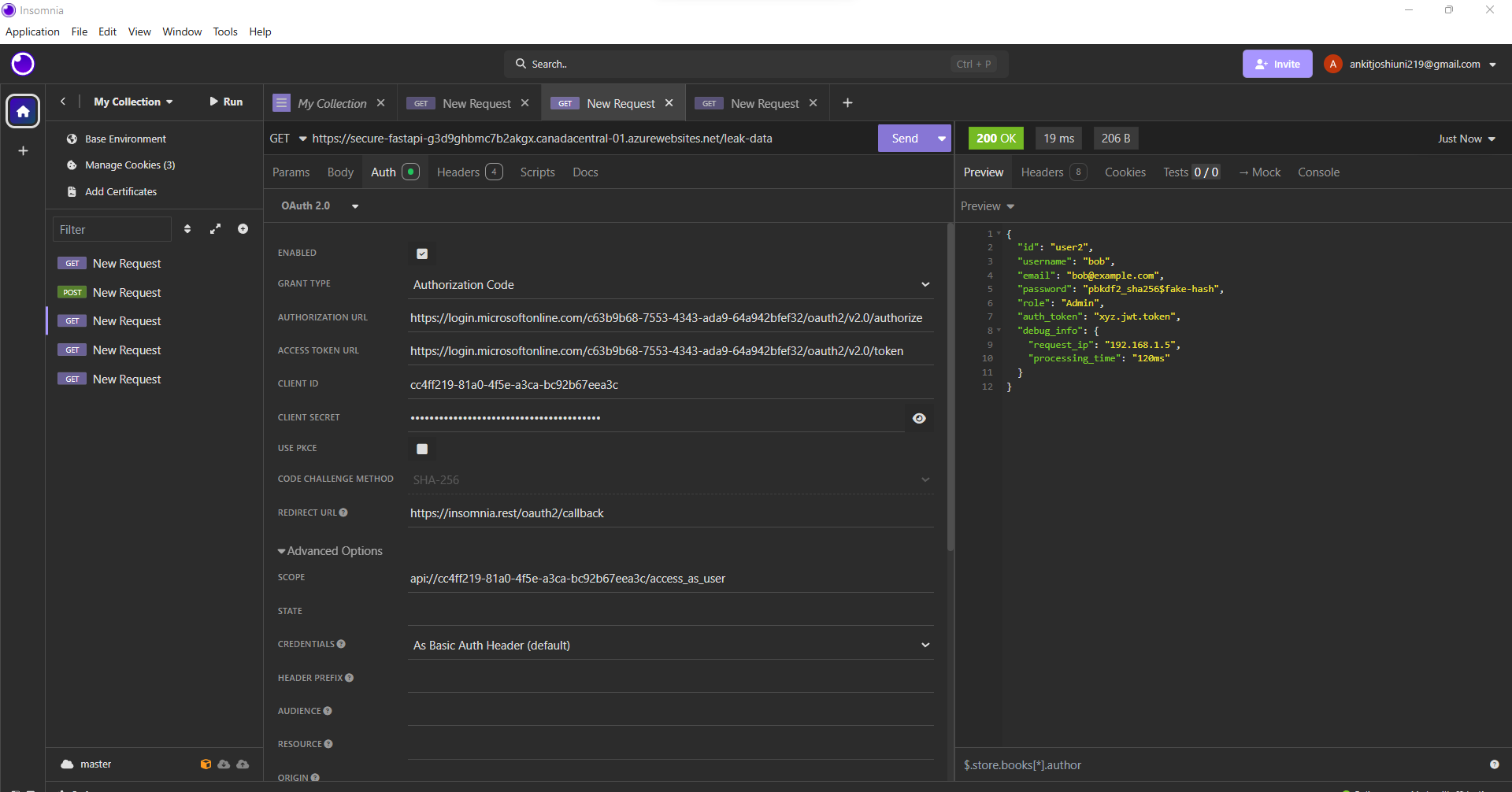


Figure 5.2 – Insomnia response from /leak-data showing sensitive fields (password hash, JWT token, debug info).

This response exposed credentials, tokens, and operational metadata that could be exploited to compromise accounts, pivot attacks, or conduct reconnaissance. The security implications of this vulnerability are severe: attackers could reuse authentication tokens to impersonate users, leverage password hashes for cracking attempts, or exploit debug information for targeted lateral movement.

OWASP ZAP flagged this issue as “Sensitive Information Exposure,” validating its effectiveness at detecting overt data leakage in API responses. The vulnerability was assessed as Critical with a CVSS v4.0 score of 8.7 (Appendix C, Figure C.3), reflecting the high risk associated with the exposure of authentication tokens and personal identifiers.

Following remediation, strict output filtering was enforced so that responses were limited to business-relevant attributes such as username and role. Re-testing confirmed that sensitive fields were removed, and subsequent scans verified that no credentials, tokens, or debug metadata remained exposed. This demonstrated that the mitigation was effective in restoring confidentiality and reducing the attack surface.

The persistence of such an error illustrates the importance of secure coding practices and output sanitisation, particularly in SME contexts where APIs are often developed rapidly without thorough review. Unlike BOLA, which escaped automated detection, this vulnerability was readily identified by ZAP, underscoring that free-tier tools can effectively detect surface-level misconfigurations but cannot substitute for disciplined development practices. For SMEs, the lesson is clear: while tools provide valuable detection capability, secure defaults and rigorous data-handling policies must underpin the design of API responses.

## 5.3 Mass Assignment – OWASP API6

The /mass-assign endpoint was deliberately implemented to accept arbitrary JSON payloads without validation, enabling attackers to inject parameters beyond those intended by the developer. Manual exploitation using Burp Suite Community Edition confirmed this weakness. By intercepting a request and modifying the payload to include fields such as is\_admin and role, an attacker was able to escalate privileges and create a new account with administrative rights, as illustrated below:

* **Request:**

POST /mass-assign

{

"username": "hacker123",

"email": "hacker@example.com",

"is\_admin": true,

"role": "SuperAdmin"

}

* **Response:**

{

"created\_user": {

"username": "hacker123",

"email": "hacker@example.com",

"is\_admin": true,

"role": "SuperAdmin"

}

}

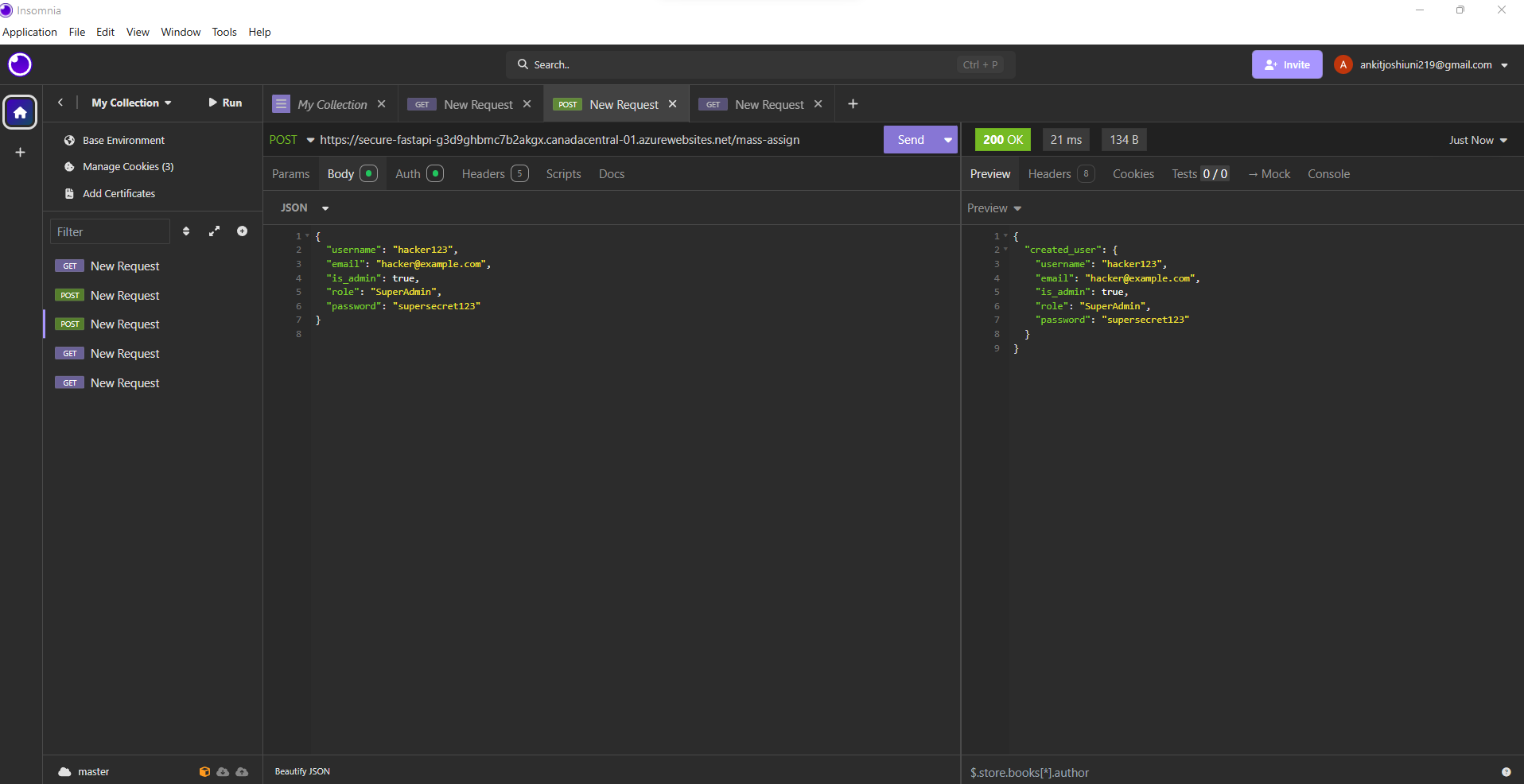


Figure 5.3 – Insomnia request with injected fields (is\_admin, role) resulting in privilege escalation.

This confirmed that the application directly trusted client-supplied input, violating fundamental security principles by permitting privilege escalation through data model manipulation. The implications are severe: an adversary exploiting this vulnerability could create multiple high-privilege accounts, bypass existing role-based restrictions, and potentially assume full control of the system.

OWASP ZAP, however, failed to identify this weakness. Neither passive nor active scans detected the schema violation or the business-logic implications, demonstrating the inability of automated scanners to detect Mass Assignment vulnerabilities without contextual awareness of the underlying data model. Detection was entirely dependent on manual payload tampering and behavioural analysis.

The issue was assessed as High severity, with a CVSS v4.0 score of 8.6 (Appendix C, Figure C.4), reflecting its potential for privilege escalation and full compromise of application integrity. Following remediation, the endpoint was secured by enforcing strict input validation through Pydantic models configured with extra="forbid". This prevented untrusted fields from being accepted, and re-testing confirmed that attempts to include arbitrary attributes were rejected, with only server-defined defaults applied.

This case highlights both the subtlety and danger of Mass Assignment vulnerabilities. While trivial to exploit with basic interception tools, they remain virtually invisible to automated scanners, as also observed in Atlidakis et al. (2020), who emphasised that business-logic flaws are among the most under-detected categories. For SMEs, this underscores that investment in developer awareness and secure coding practices is as crucial as technical tooling. Free-tier scanners, while valuable, cannot detect flaws that depend on contextual rules, reinforcing the indispensable role of manual validation.

## 5.4 Broken User Authentication – OWASP API2

The /login endpoint was deliberately designed with weak authentication, relying on plaintext password comparison without hashing, salting, or rate limiting. Manual exploitation using Insomnia demonstrated that trivial credentials such as 123456 were accepted for valid users. Further, repeated login attempts with varied passwords were possible without triggering account lockouts, enabling brute-force attacks to be executed unhindered.

* **Request:**

POST /login

{ "username": "alice", "password": "123456" }

* **Response:**

{ "message": "Logged in", "token": "fake.user1.token" }

This confirmed the absence of fundamental authentication safeguards, violating secure credential handling practices. The consequences of this vulnerability are severe: attackers could gain persistent access through brute-force attacks, compromise multiple accounts, and reuse captured tokens for further privilege escalation.

Unlike excessive data exposure, this weakness was not flagged by OWASP ZAP, either in passive or active scanning modes. The failure is consistent with broader evidence that automated scanners cannot infer logical flaws in authentication workflows, since they lack knowledge of valid credential sets or business rules. This made detection reliant entirely on manual testing, highlighting once again the irreplaceable role of human oversight in identifying business-logic vulnerabilities.

The issue was classified as High severity, with a CVSS v4.0 score of 8.8 (Appendix C, Figure C.2), reflecting its potential to enable brute-force compromise and unrestricted account takeover. Following remediation, the endpoint was hardened through the introduction of SlowAPI-based rate limiting (five attempts per minute), structured error logging of failed attempts, and stronger credential policies.

Critically, while the patch reduced the immediate risk of brute force, it did not yet address the fundamental weakness of plaintext password validation. Secure password hashing mechanisms such as bcrypt or argon2 remain a necessary next step for production-grade security. This limitation reflects the SME-oriented constraints of the project: while free-tier and open-source tools were used to enforce basic safeguards, resource and time restrictions limited the adoption of advanced cryptographic solutions.

This vulnerability illustrates the systemic risks posed by weak authentication, a recurring issue in both academic studies and industry breach reports (Verizon DBIR, 2025). For SMEs, it underlines that neglecting password security exposes organisations to disproportionate risk, regardless of other layered defences. Tooling cannot compensate for fundamental design flaws, reinforcing the need for secure-by-design development practices.

## 5.5 Improper Asset Management – OWASP API9

The /v1/legacy-info endpoint remained publicly accessible despite being deprecated and undocumented. Manual exploitation using Insomnia confirmed that requests to this endpoint still returned system versioning details, exposing unnecessary technical metadata to unauthenticated clients.

* **Request:**

GET /v1/legacy-info

* **Response:**

{"version": "v1", "message": "Deprecated endpoint"}

Although the output did not reveal direct credentials or personal data, its accessibility nonetheless created an enlarged attack surface. Attackers could leverage such information to fingerprint the API, identify outdated functionality, and mount targeted exploits based on unmaintained versions. This aligns with OWASP API9: Improper Asset Management, which stresses that forgotten or undocumented APIs often become backdoors for exploitation.

OWASP ZAP successfully detected the endpoint, flagging it as “Unlinked Functionality.” However, automated detection in this case was superficial, as ZAP was unable to contextualise the risk beyond identifying its existence. Manual validation was essential to confirm both its accessibility and its potential role in facilitating reconnaissance.

The issue was classified as Medium severity, with a CVSS v4.0 score of 6.9 (Appendix C, Figure C.5). The lower score reflects the limited confidentiality and integrity impact of the endpoint itself, but its presence highlighted systemic weaknesses in API lifecycle management that, if unaddressed, could lead to more severe exploitation.

Following remediation, the /v1/legacy-info route was reconfigured to return an HTTP 410 Gone response with a redirection notice to /v2/info. This ensured that outdated endpoints no longer exposed metadata while still guiding legitimate users toward the correct resource.

From a critical perspective, this vulnerability underscores the operational challenge SMEs face in maintaining accurate API inventories. As Atlidakis et al. (2020) argue, improper asset management is rarely prioritised in SMEs due to limited resources, yet it can have cascading effects if deprecated endpoints become an entry point for broader attacks. In this project, the fix was straightforward, but in real-world contexts with larger microservice estates, SMEs must adopt systematic discovery and documentation practices to prevent such oversights.

## 5.6 Summary of Simulated Vulnerabilities

The vulnerabilities introduced into the artefact were successfully exploited in the baseline configuration and subsequently mitigated through layered controls. Table 5.1 consolidates the findings across all five implemented categories of the OWASP API Security Top 10.

Table 5.1 – Summary of simulated API vulnerabilities, associated endpoints, exploitation evidence, and patched status.

| **Vulnerability** | **Endpoint** | **Exploited?** | **Tool Evidence** | **Patched Status** |
| --- | --- | --- | --- | --- |
| Broken Object Level Auth (API1) | /users/{user\_id} | Yes | Insomnia, ZAP | Ownership enforced |
| Excessive Data Exposure (API3) | /leak-data | Yes | ZAP Alert | Sensitive fields removed |
| Mass Assignment (API6) | /mass-assign | Yes | Burp, Insomnia | Extra fields blocked |
| Broken User Auth (API2) | /login | Yes | Logs, manual | Rate limiting + logs |
| Improper Asset Mgmt. (API9) | /v1/legacy-info | Yes | ZAP, Insomnia | Returns 410 Gone |

This summary confirms that all five simulated vulnerabilities were practically exploitable prior to mitigation. Automated scanning (ZAP) proved effective at detecting surface-level weaknesses such as data exposure and deprecated endpoints, but it consistently failed to identify contextual flaws such as BOLA, Mass Assignment, and weak authentication. These required manual verification through tools like Burp Suite CE and Insomnia. Post-mitigation, re-testing confirmed that each vulnerability was effectively remediated, demonstrating the value of layered defences even in SME-relevant free-tier environments.

# Chapter 6: Security Testing and Evaluation

This presents the outcomes of security testing conducted on the vulnerable FastAPI application deployed to Microsoft Azure. Both manual penetration testing (using Insomnia and Burp Suite Community Edition) and automated scanning (via OWASP ZAP) were employed to assess the system against the deliberately introduced vulnerabilities documented in Chapter 5.

Testing was performed in two distinct phases:

1. **Vulnerable Phase (Baseline)** – All vulnerability toggles were enabled, leaving the API exposed to the selected OWASP API Top 10 weaknesses.
2. **Patched Phase (Mitigation)** – Vulnerability toggles were disabled, and additional security controls were enforced as described in Chapter 4.

This dual-phase approach facilitated a comparative evaluation of tool effectiveness across both insecure and remediated states. Results were mapped to the OWASP API Security Top 10 (2023) categories and assessed using the CVSS 4.0 severity scoring framework, ensuring rigorous and reproducible analysis. Extracts from ZAP passive and active scan outputs are presented within this chapter to illustrate findings, while the full reports are provided in Appendix C to ensure transparency and compliance with submission requirements.

## 6.1 Testing Environment and Setup

The security evaluation was conducted in a controlled environment designed to replicate SME-relevant deployment conditions. The setup incorporated both cloud-hosted and local testing components:

* **API Deployment** – The artefact was deployed on *Microsoft Azure App Service (Free Tier)*, reflecting cost-sensitive SME adoption patterns while ensuring enforced HTTPS/TLS.
* **Authentication Modes** – Two authentication configurations were applied:
  + *Development mode* – used a custom X-Dev-User header for simplified local testing and validation.
  + *Cloud mode* – employed *Azure Entra ID OAuth2 tokens*, validated against Microsoft’s JWKS endpoint.
* **Testing Tools** – A combination of automated and manual tools was utilised:
  + *Insomnia* – for executing authenticated requests and verifying endpoint behaviour.
  + *Burp Suite Community Edition* – for payload tampering and logical flaw validation.
  + *OWASP ZAP* – for passive and active scanning, configured with the Replacer add-on to handle authentication.
* **Logging** – Monitoring was enabled through *Azure Application Logs* (cloud) and *local rotating file logs* (development), capturing requests, responses, authentication attempts, and rate-limit events.

This environment ensured that security testing remained reproducible, verifiable, and representative of realistic SME deployment scenarios.

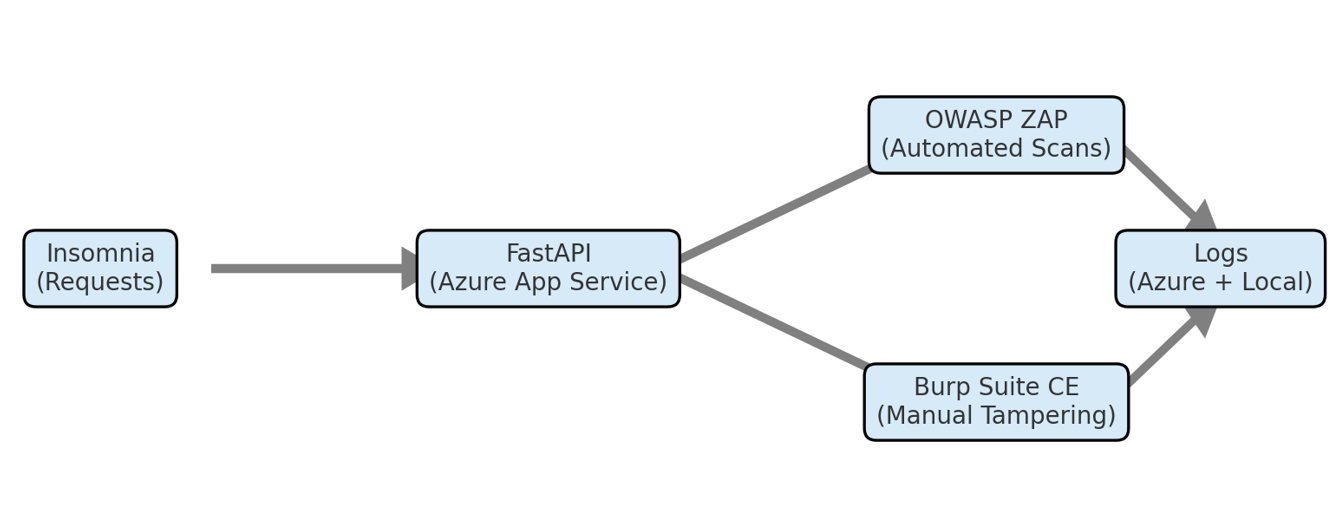
****

Figure 6.1: Security Testing Workflow diagram

## 6.2 Results of Vulnerability Testing

The vulnerability assessment was conducted in two phases: a baseline test of the insecure API and a repeat assessment following the application of mitigations. Each of the five simulated vulnerabilities was probed using both automated and manual techniques to evaluate exploitability and tool effectiveness. The following subsections present the detailed results for each vulnerability, including the exploit method, supporting tool evidence, CVSS v4.0 severity score, and observed post-patch behaviour.

### 6.2.1 Broken Object Level Authorization (API1)

As demonstrated in Chapter 5, the /users/{user\_id} endpoint allowed an authenticated user to retrieve another user’s record without ownership validation. This confirmed a direct violation of least-privilege principles, resulting in cross-user data disclosure.

OWASP ZAP raised an alert for *Excessive Information Exposure in Response* but failed to detect the underlying access control flaw. This gap highlights the limitations of automated scanners when evaluating business logic vulnerabilities, aligning with Hadi and Nugroho (2020), who emphasise the insufficiency of automation for contextual issues. In contrast, manual testing with Insomnia reliably exposed the weakness.

The vulnerability was classified as **High severity (CVSS v4.0 score: 7.1)**, reflecting its significant confidentiality impact (see Appendix C, Figure C.1).

Following remediation, ownership checks were enforced. Re-tests confirmed that non-admin users were restricted to their own records, while administrators retained broader access privileges. Attempts at cross-user access post-patch returned an HTTP 403 response, demonstrating effective mitigation.

This result illustrates a recurring pattern: automated tools like ZAP are competent at identifying symptomatic issues (e.g., information leakage) but cannot infer deeper logic flaws without contextual awareness. For SMEs, this suggests that reliance solely on automation leaves critical vulnerabilities undetected. Manual validation—even using lightweight tools like Insomnia—remains indispensable for uncovering such flaws.

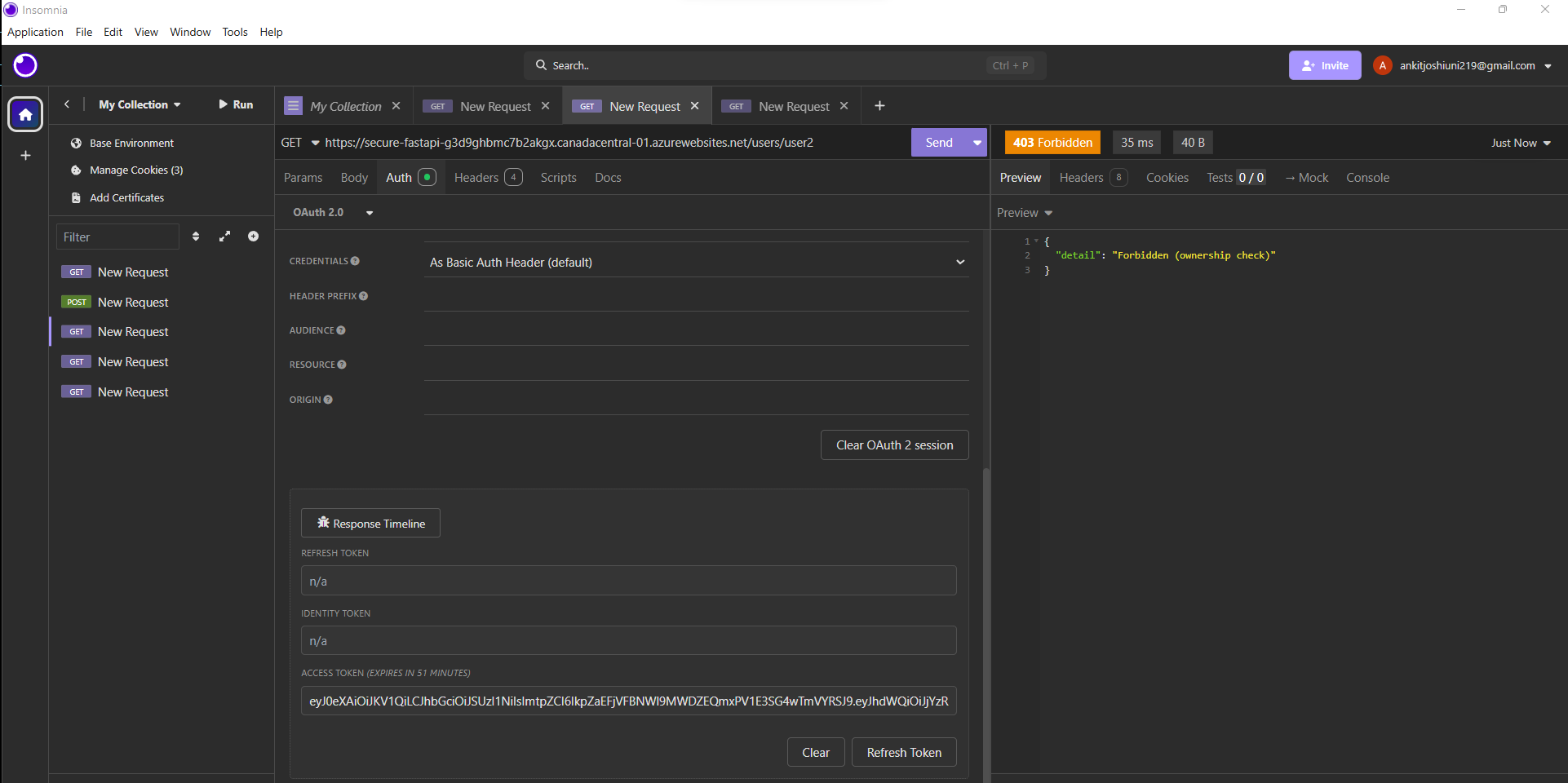


Figure 6.2 – Post-patch Insomnia response showing access restricted with 403 Forbidden.

### 6.2.2 Broken User Authentication (API2)

As documented in Chapter 5, the /login endpoint accepted weak plaintext passwords without hashing and lacked rate-limiting in its baseline state. Manual testing confirmed that trivial credentials (e.g., 123456) were accepted and that repeated brute-force attempts were possible without triggering account lockout.

OWASP ZAP raised no alerts for this issue, reflecting its limitations in detecting authentication weaknesses that rely on behavioural analysis rather than protocol misconfigurations. This aligns with the findings of Atlidakis et al. (2020), who note that automated scanners frequently miss authentication flaws. Detection relied entirely on manual exploitation and log analysis.

The vulnerability was assessed as **High severity (CVSS v4.0 score: 8.8)**, reflecting the elevated risk of brute-force compromise (Appendix C, Figure C.2).

Mitigation was achieved by introducing rate limiting (five login attempts per minute) and enhanced logging of authentication events. Post-patch testing confirmed that repeated login attempts were blocked and logged, although password hashing improvements (e.g., bcrypt) remain a recommended future enhancement. This case illustrates a broader limitation of automated free-tier tools: while scanners like ZAP can detect surface-level misconfigurations, they cannot model authentication logic or capture brute-force risks. For SMEs, this reinforces the importance of manual validation and disciplined logging as part of ongoing security assurance.

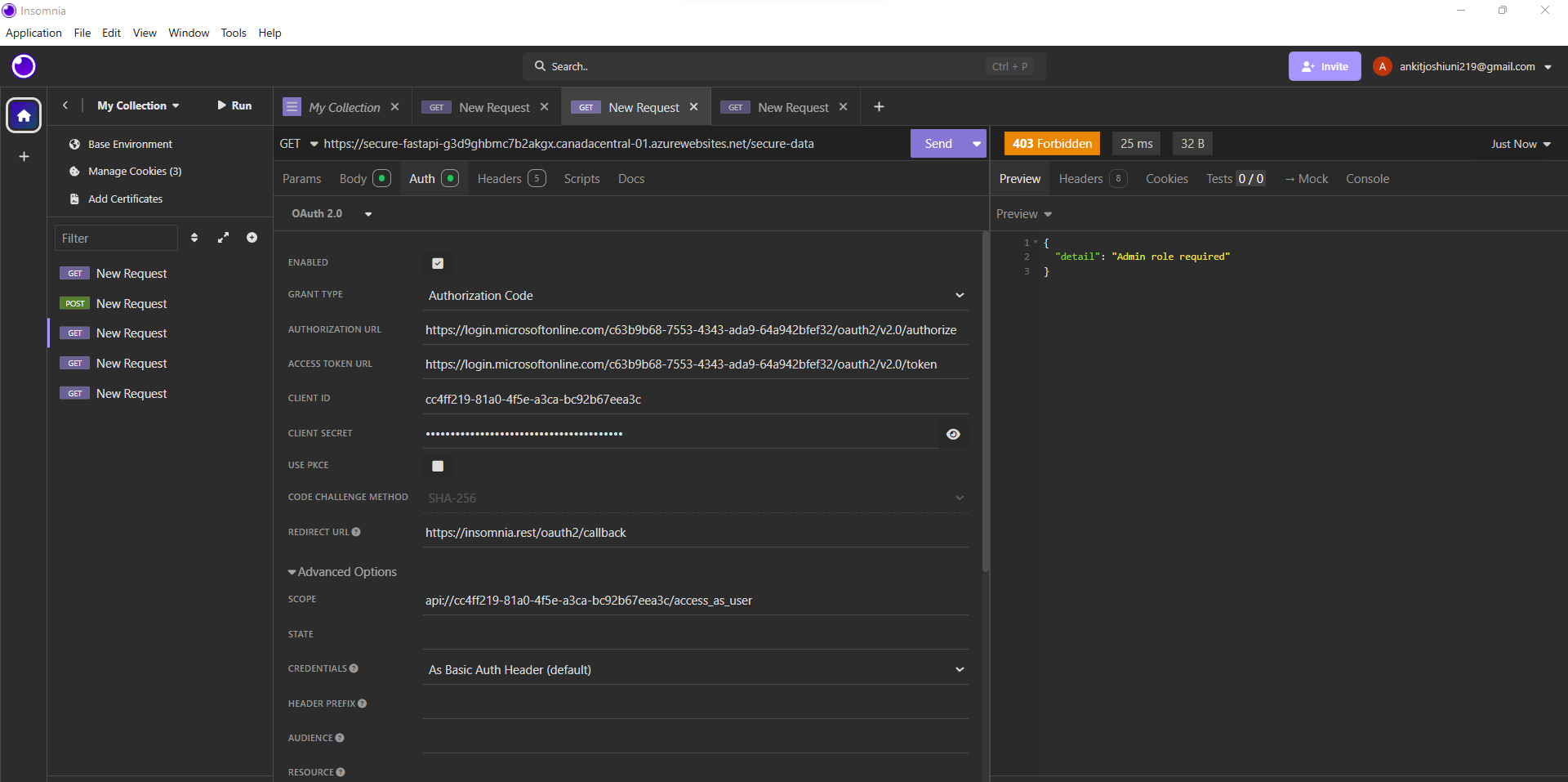


Figure 6.3 – Post Patch response for Broken User Authentication

### 6.2.3 Excessive Data Exposure (API3)

The /leak-data endpoint disclosed sensitive information including password hashes, authentication tokens, and debug metadata. Manual exploitation through Insomnia confirmed that these fields were returned directly in API responses without sanitisation or filtering.

OWASP ZAP flagged the issue as *Sensitive Information Exposure*, confirming its ability to detect overt data leakage in responses. This demonstrates that automated scanners are well-suited to identifying clear-cut violations of secure output handling.

The vulnerability was rated **Critical (CVSS v4.0 score: 8.7)** due to the exposure of credentials and tokens that could enable further compromise (Appendix C, Figure C.3).

After remediation, sensitive attributes were stripped from responses, leaving only safe fields such as username and role. Subsequent manual tests and ZAP rescans verified that no tokens, hashes, or debug data remained exposed. While the fix was effective, this case highlights the danger of over-reliance on default development configurations. For SMEs, the lesson is that automated tools like ZAP can catch obvious leaks, but strong engineering practices—such as output whitelisting—must still underpin secure API design.

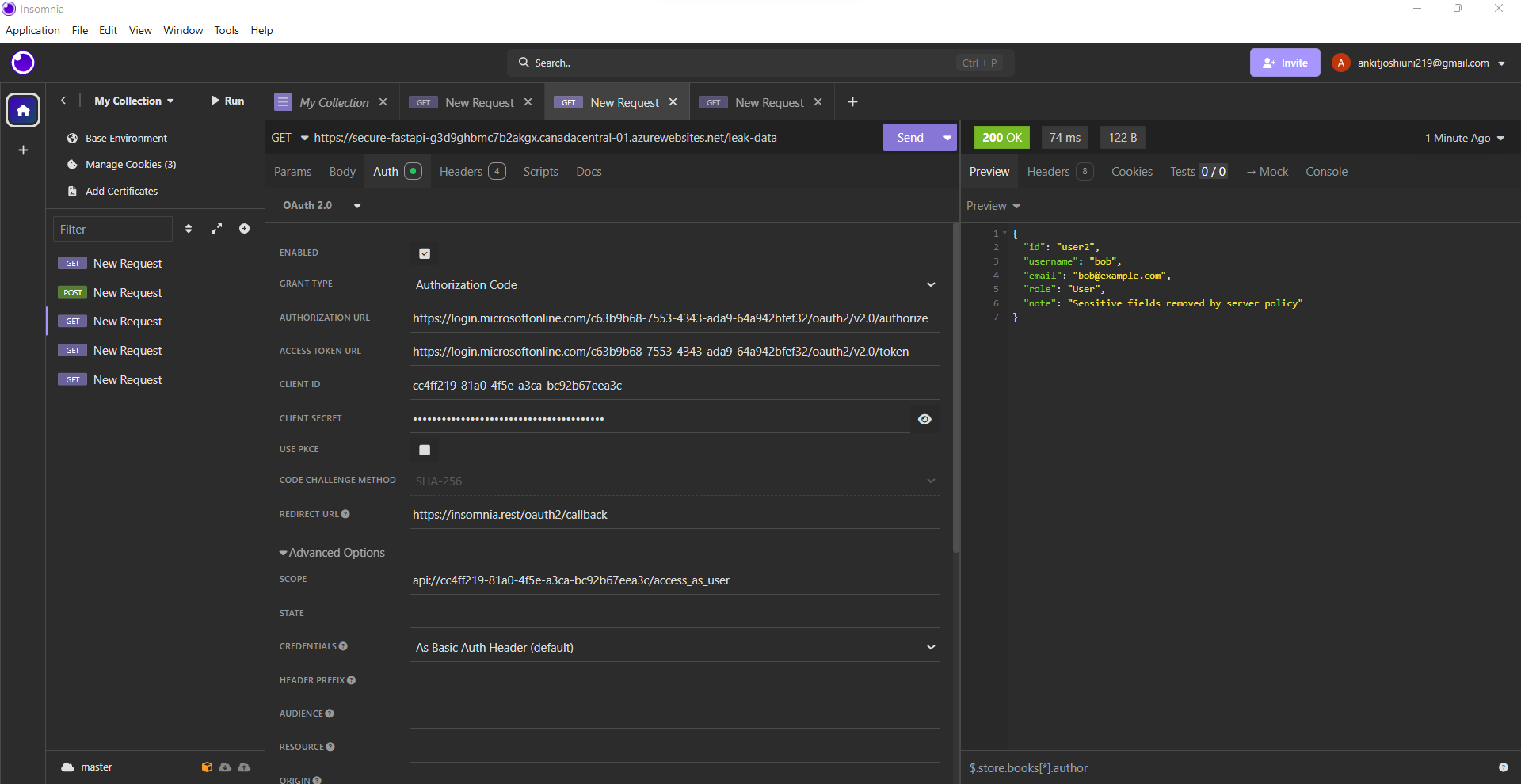


Figure 6.4 – Post patch sanitized response

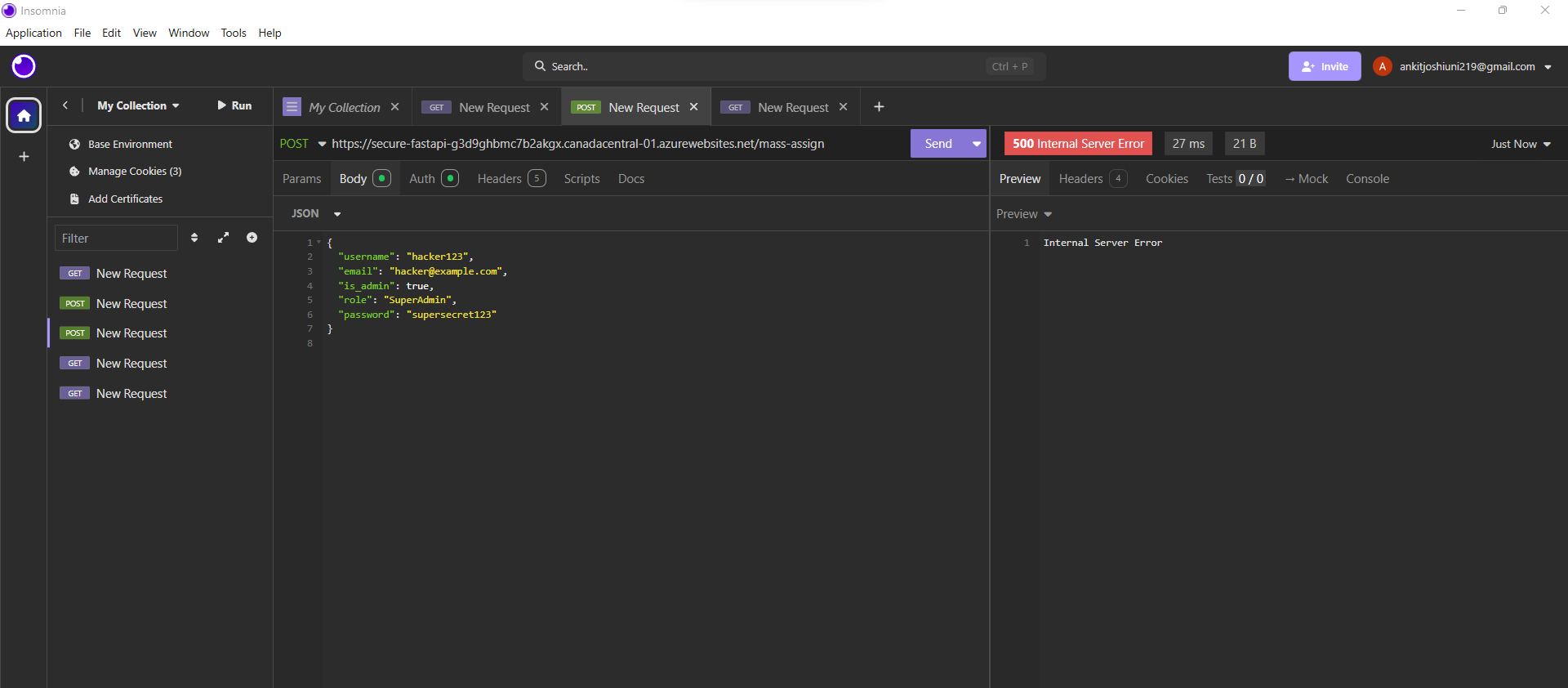
### 6.2.4 Mass Assignment (API6)

The /mass-assign endpoint accepted arbitrary JSON keys, allowing attackers to inject parameters such as is\_admin and role. Using Burp Suite Community Edition, the payload was intercepted and modified, successfully escalating privileges by creating a user with administrative access. Insomnia tests confirmed that the manipulated fields were persisted in the system, validating the exploit.

OWASP ZAP, however, failed to detect this issue during both passive and active scans. The absence of schema-awareness and contextual logic analysis meant the vulnerability went entirely unnoticed, demonstrating a critical limitation of automated tools in detecting business-rule violations.

The vulnerability was classified as **High severity (CVSS v4.0 score: 8.6)**, reflecting the potential for privilege escalation and total compromise of application integrity (Appendix C, Figure C.4).

The patched implementation enforced strict Pydantic schemas with extra="forbid", ensuring that only whitelisted attributes were accepted. Re-testing confirmed that injected fields such as is\_admin and role were rejected, and server defaults were correctly applied. While the remediation was successful, the fact that automated scanners missed such a severe flaw highlights a structural weakness in relying solely on free-tier tools. SMEs must combine lightweight manual testing with automation to achieve meaningful coverage of logic-based vulnerabilities.



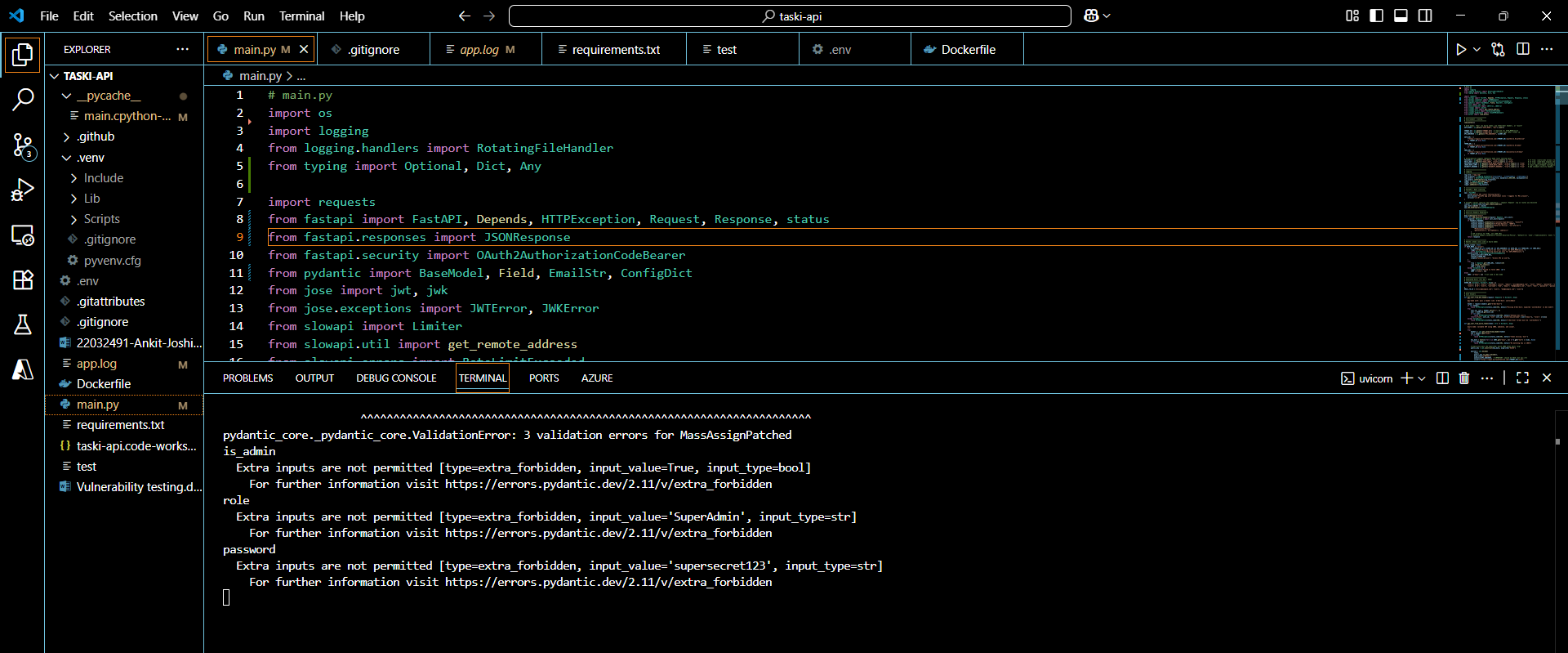


Figure 6.5 – Post patch response throwing error or sanitized response to inject is\_admin role

### 6.2.5 Improper Asset Management (API9)

The /v1/legacy-info endpoint, though deprecated, remained publicly accessible. Testing with Insomnia confirmed that it returned system information, including version details, without requiring authentication. While the output was not directly sensitive, the exposure of deprecated functionality significantly expanded the attack surface, providing attackers with potential reconnaissance opportunities.

OWASP ZAP flagged the endpoint under “Unlinked Functionality,” correctly identifying that a hidden or undocumented route was still active. This detection illustrates ZAP’s relative strength in identifying residual attack surfaces, though it provided no contextual assessment of risk severity.

The issue was assessed as **Medium severity (CVSS v4.0 score: 6.9)** (Appendix C, Figure C.5). Although the confidentiality, integrity, and availability impact were limited, the persistence of deprecated APIs in production systems is a well-documented precursor to more serious exploitation. For example, attackers may leverage such endpoints to identify outdated versions or exploit forgotten vulnerabilities in legacy code paths.

The remediation enforced strict lifecycle management practices: deprecated endpoints now return an **HTTP 410 Gone** response, with explicit guidance to use the updated /v2/info endpoint. Post-patch testing confirmed that the legacy endpoint was inaccessible, thereby reducing the attack surface. The fix demonstrates how relatively minor configuration changes can effectively eliminate risks associated with Improper Asset Management, an area where SMEs often struggle due to limited maintenance resources.

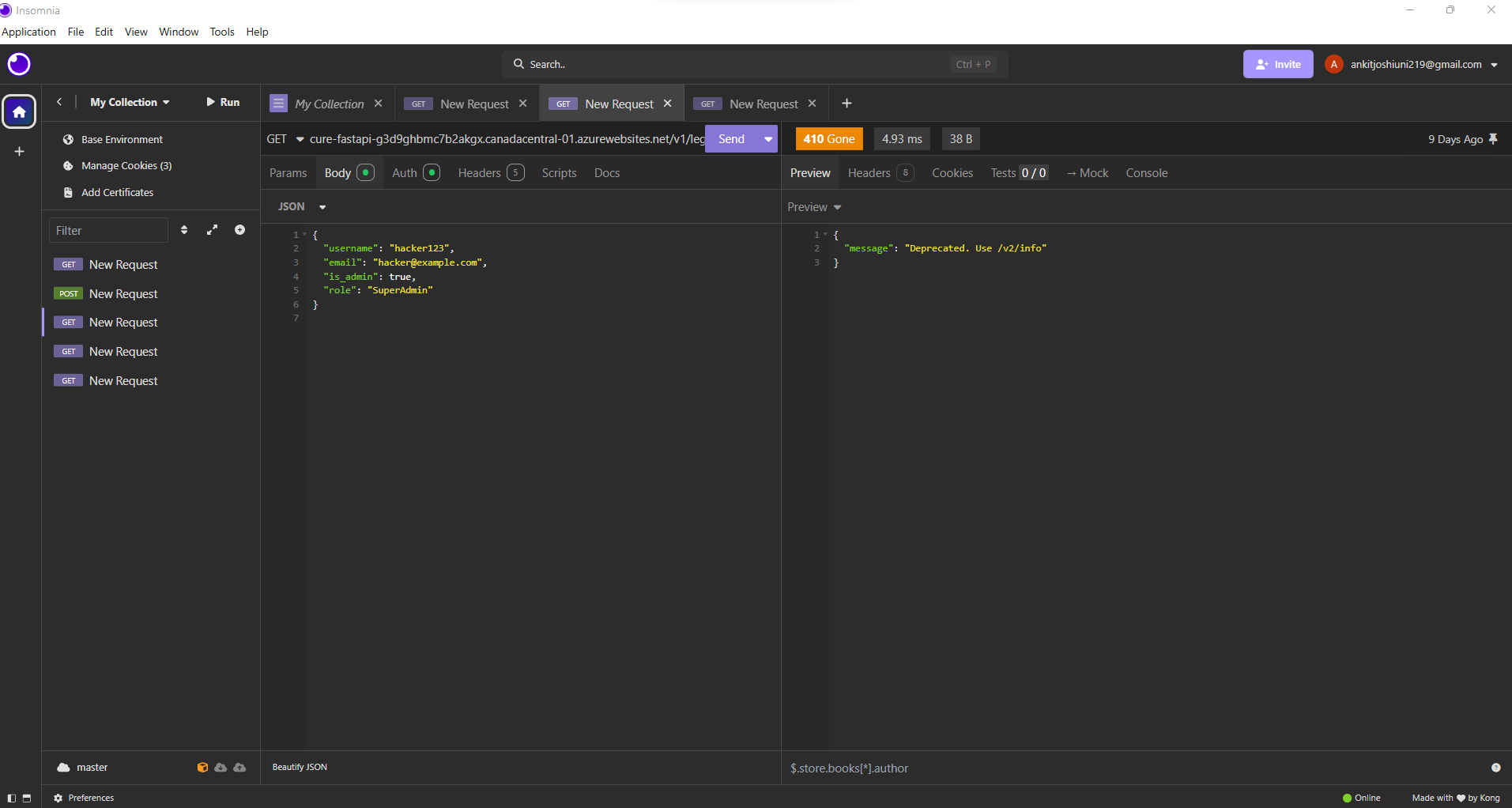


Figure 6.6 – Evidence of Improper Asset Management

## 6.3 Summary of Findings

The dual-phase testing confirmed that all five deliberately introduced vulnerabilities were exploitable in the baseline state and effectively mitigated once layered security controls were applied. When assessed against the OWASP API Security Top 10 and CVSS v4.0 frameworks, a clear pattern emerged: OWASP ZAP was effective in detecting surface-level weaknesses such as excessive data exposure (API3) and deprecated endpoints (API9), yet consistently failed to identify deeper logic flaws, including BOLA (API1), Mass Assignment (API6), and Broken Authentication (API2). In contrast, manual exploitation with Insomnia and Burp Suite CE reliably exposed these contextual vulnerabilities, highlighting the irreplaceable role of human-driven analysis in complementing automated scanning.

The CVSS scoring further reinforced this distinction. The most severe vulnerabilities—Excessive Data Exposure (8.7), Broken Authentication (8.8), and Mass Assignment (8.6)—were those requiring contextual understanding rather than automated detection. By comparison, Improper Asset Management (6.9) carried lower risk but was consistently detected by automated scans.

Overall, the findings illustrate both the potential and the inherent limitations of free-tier toolchains: they provide SMEs with a cost-effective means of raising their baseline API security posture, but cannot substitute for skilled manual testing and contextual awareness. This chapter has therefore delivered the empirical evidence required to evaluate tool effectiveness, severity, and mitigation outcomes, directly supporting the critical discussion in Chapter 7.

Table 6.1 – Tool effectiveness matrix showing detection coverage across manual and automated approaches, with explanatory comments.

| **Vulnerability (OWASP API Top 10)** | **Manual (Insomnia / Burp CE)** | **ZAP Passive Scan** | **ZAP Active Scan** | **Detection Outcome** | **Interpretation** |
| --- | --- | --- | --- | --- | --- |
| API1 – Broken Object Level Authorization (BOLA) | Confirmed cross-user data retrieval | Flagged “Excessive Information Exposure” only | Not detected | Partial | Automated scans identified data exposure but missed the core authorization flaw. Manual validation was essential. |
| API2 – Broken Authentication | Weak password and brute-force confirmed | Not detected | Not detected | Manual only | ZAP lacked the capability to infer authentication logic flaws. Reliant entirely on manual testing. |
| API3 – Excessive Data Exposure | Confirmed sensitive fields leaked | Detected sensitive fields in response | Confirmed via active probes | Full | Strong alignment between automated and manual findings. Demonstrates ZAP’s strength in overt leakage detection. |
| API6 – Mass Assignment | Privilege escalation via JSON injection | Not detected | Not detected | Manual only | Schema-awareness gap in ZAP: unable to identify injected fields. Manual payload tampering required. |
| API9 – Improper Asset Management | Deprecated endpoint confirmed | Flagged as “Unlinked Functionality” | Limited confirmation of deprecated resource | Partial | ZAP identified the presence of an unlinked endpoint but lacked contextual interpretation of business risk. |

The results in Table 6.1 illustrate clear distinctions in tool performance. OWASP ZAP consistently detected overt misconfigurations and sensitive data exposure (API3, API9), yet failed to identify business logic flaws such as BOLA (API1) and Mass Assignment (API6). Burp Suite CE and Insomnia proved indispensable for detecting these contextual vulnerabilities, confirming their value in complementing automated scanning. Overall, the matrix underscores that while automation provides breadth of coverage, human-driven testing is critical for identifying logic-based flaws with high security impact.

# Chapter 7: Analysis and Discussion

## 7.1 Introduction

This chapter critically analyses the outcomes of the vulnerability testing phase, situating the findings within the research objectives, the literature reviewed in Chapter 2, and broader debates in API security. The discussion moves beyond a descriptive account of tool performance to interrogate the extent to which free-tier and open-source solutions can meaningfully contribute to securing APIs in resource-constrained contexts. Particular emphasis is placed on Small and Medium-sized Enterprises (SMEs), where limited financial capacity and reduced access to specialist expertise exacerbate exposure to API-related risks. By evaluating both detection coverage and mitigation effectiveness, the chapter addresses not only the technical adequacy of these tools but also their strategic viability as part of SME security practice.

## 7.2 Tool Effectiveness

The comparative evaluation of OWASP ZAP, Burp Suite CE, and Insomnia revealed both complementary strengths and structural limitations, underscoring the necessity of combining automated and manual approaches.

OWASP ZAP provided breadth through its automated scanning, effectively identifying surface-level weaknesses such as missing security headers, sensitive data exposure, and the presence of unlinked endpoints. This capability demonstrates its value as a rapid assessment tool, particularly for SMEs lacking specialist staff. However, its failure to detect logic-driven vulnerabilities such as Broken Object Level Authorization (BOLA) and weak authentication critically undermines its adequacy as a standalone solution. This aligns with Hadi and Nugroho (2020), who contend that automated scanners lack contextual awareness of business logic, and suggests that SMEs relying solely on such tools would retain blind spots in access control and privilege escalation risks.

Burp Suite Community Edition, despite its lack of automated scanning features, demonstrated depth in manual testing. By enabling payload tampering and parameter manipulation, it successfully revealed Mass Assignment vulnerabilities and validated privilege escalation scenarios. Its effectiveness highlights the irreplaceable role of human expertise in uncovering logic flaws. Yet, the reliance on manual operation also introduces scalability limitations for SMEs with minimal technical capacity.

Insomnia proved indispensable as a lightweight client for authenticated testing. It facilitated systematic exploration of token-based authentication flows and endpoint behaviour, enabling the detection of BOLA and validation of patched states. While it lacks the sophistication of penetration testing frameworks, its accessibility and low overhead make it particularly suited for SME contexts, where usability is a critical factor.

Taken together, the toolset delivered a pragmatic balance: ZAP provided baseline coverage, Burp Suite CE offered depth for targeted logic testing, and Insomnia ensured usability and reproducibility. This triangulation demonstrates that free-tier tools can approximate enterprise testing workflows, but only when strategically combined and reinforced with skilled manual analysis. For SMEs, the implication is clear: tool adoption must be accompanied by investment in process discipline and human oversight, or else critical vulnerabilities will remain undetected.

## 7.3 Free-Tier Tools vs Commercial Solutions

The findings reinforce arguments made by SentinelOne (2025) and Atlidakis et al. (2020) that free-tier and open-source tools, while valuable for identifying common API misconfigurations, lack the comprehensiveness and automation depth offered by enterprise-grade solutions such as commercial API gateways or specialised API security platforms (e.g., Salt Security, 42Crunch). Enterprise products often integrate anomaly detection, schema validation, and continuous monitoring — features absent in the free-tier stack tested here.

However, within the SME context, the results of this study demonstrate that meaningful security assurance can still be achieved without enterprise-grade investment, provided the tools are strategically combined. The triangulated use of ZAP, Burp Suite CE, and Insomnia delivered both coverage and validation at zero cost, which is significant given SMEs’ financial and staffing constraints. This supports the assertion of Ali et al. (2015) that effective API security is not simply a technological issue but also a matter of process and oversight.

Crucially, the study highlights a trade-off. While free-tier tools can elevate the baseline security posture of SMEs, their effectiveness is capped by two factors: (i) their inability to detect complex business-logic flaws, and (ii) their dependence on the operator’s technical expertise. In resource-constrained organisations, this reliance on human oversight may introduce variability in outcomes and increase the likelihood of residual risk. By contrast, commercial solutions reduce this variability through automation and integrated reporting, albeit at costs often beyond SME budgets.

The implication, therefore, is not that free-tier tools are inadequate, but that SMEs must adopt them as part of a layered and disciplined methodology. Free resources can provide a defensible baseline, but they cannot replace the assurance offered by commercial platforms when regulatory compliance, customer trust, and high-value data are at stake. For SMEs, the challenge lies in striking a balance: leveraging no-cost solutions to address common weaknesses, while recognising when to escalate to commercial-grade investments as their risk profile matures.

## 7.4 Severity and Risk Assessment

The CVSS v4.0 assessment provided a structured lens through which to quantify the impact of each vulnerability, revealing clear distinctions in both severity and detection coverage. Broken Authentication (CVSS 8.8, High) and Excessive Data Exposure (CVSS 8.7, High) emerged as the most damaging, given their potential to enable brute-force compromise and leakage of sensitive authentication material. Mass Assignment (CVSS 8.6, High) represented another critical weakness, permitting privilege escalation through unsanitised input manipulation. Broken Object Level Authorization (BOLA) scored lower (CVSS 7.1, High) but still posed a substantial risk due to the potential for systematic data harvesting. By contrast, Improper Asset Management (CVSS 6.9, Medium) was the least severe, reflecting the lower exploitability of deprecated endpoints while still highlighting poor lifecycle management.

The results highlight a recurring pattern: vulnerabilities tied to logic and contextual enforcement (BOLA, Mass Assignment, and Broken Authentication) are consistently severe and yet remain undetectable by automated tools. This validates OWASP’s (2023) claim that logic-based flaws constitute the most damaging class of API vulnerabilities while also being the least easily discovered by automated scanning. ZAP’s ability to flag excessive data exposure and deprecated functionality confirms its strength for surface-level misconfigurations but simultaneously underscores its blind spot for access-control and workflow flaws.

From an SME perspective, the implication is twofold. First, free-tier tools can effectively reduce the “low-hanging fruit” of API insecurity, such as misconfigured endpoints and exposed data, thereby raising the baseline security posture at minimal cost. Second, however, the persistence of undetected high-severity flaws demonstrates that free tools are insufficient in isolation. Without manual validation, SMEs risk maintaining a false sense of security — particularly dangerous where regulatory obligations (e.g., GDPR) or customer trust are at stake.

Table 7.1 consolidates these insights, mapping vulnerabilities against their CVSS scores and tool coverage, thereby reinforcing the conclusion that automation must be systematically complemented by human-driven analysis.

Table 7.1 – Vulnerability severity versus tool detection coverage

| **Vulnerability** | **OWASP API Category** | **CVSS v4.0 Score (Severity)** | **ZAP Detection** | **Burp Suite CE** | **Insomnia** |
| --- | --- | --- | --- | --- | --- |
| **API1 – Broken Object Level Authorization (BOLA)** | API1 | **7.1 (High)** | Partial – flagged information exposure but missed access control flaw | Not detected | Confirmed cross-user data access |
| **API2 – Broken User Authentication** | API2 | **8.8 (High)** | Not detected | Not detected | Weak login accepted, brute-force attempts confirmed |
| **API3 – Excessive Data Exposure** | API3 | **8.7 (High)** | Detected – “PII Exposed in Response” | Not tested | Confirmed sensitive fields leaked |
| **API6 – Mass Assignment** | API6 | **8.6 (High)** | Not detected | Arbitrary JSON field injection confirmed | Privilege escalation reproduced |
| **API9 – Improper Asset Management** | API9 | **6.9 (Medium)** | Detected – “Unlinked Functionality” | Not applicable | Deprecated endpoint confirmed |

## 7.5 Addressing the Research Gap

The literature review in Chapter 2 highlighted a critical absence of empirical studies assessing the effectiveness of free-tier and open-source tools for API security within SME contexts. Existing research either focused on descriptive accounts of vulnerabilities or on enterprise-grade defences, leaving unresolved whether cost-free toolchains could realistically enable SMEs to achieve meaningful baseline security.

This project directly addressed that gap by providing concrete empirical evidence through the design, deployment, and evaluation of a deliberately vulnerable FastAPI artefact in a free-tier cloud environment. The findings advance knowledge in three important ways:

1. **Demonstrated insufficiency of automation alone** – Automated scanning with OWASP ZAP consistently failed to detect contextual, logic-based flaws such as BOLA, Mass Assignment, and weak authentication. This confirms and extends the assertions of Hadi and Nugroho (2020), who argued that automated scanners lack the capacity to capture business-logic vulnerabilities.
2. **Validated the value of layered testing** – The integration of automated scanning (ZAP) with manual exploitation (Insomnia, Burp Suite CE) closed critical coverage gaps. This layered methodology ensured detection of both surface-level misconfigurations and deeper logical flaws, proving that SMEs can compensate for tooling limitations with structured manual validation.
3. **Established the feasibility of no-cost strategies** – By relying exclusively on Microsoft Azure Free Tier for deployment and no-cost tools for testing, the study demonstrated that SMEs can establish a baseline level of resilience without financial outlay. The successful mitigation of all five simulated vulnerabilities confirms the practical viability of such an approach.

By addressing the identified research gap, this study makes a novel contribution to both academic discourse and industry practice. It provides SMEs with a replicable methodology that balances cost-effectiveness with security assurance, while also reinforcing the need for human oversight in contexts where automation falls short.

## 7.6 Limitations of the Study

Despite meeting its objectives, this study is subject to several limitations that shape the interpretation and generalisability of its findings.

First, the scope was necessarily constrained. Only five categories from the OWASP API Security Top 10 (2023) were implemented and tested. While these represent some of the most critical and SME-relevant vulnerabilities, others—such as injection attacks, resource exhaustion, or insufficient monitoring—were excluded due to project boundaries. This restricts the comprehensiveness of the evaluation and means the results should be viewed as indicative rather than exhaustive.

Second, the toolset itself imposed constraints. The free-tier versions of OWASP ZAP and Burp Suite lacked advanced features found in their commercial counterparts, such as automated authenticated scanning, extensive reporting, or plugin extensibility. Consequently, the evaluation was limited to what SMEs could reasonably achieve without additional cost, but at the expense of breadth and automation.

Third, the testing environment was a controlled Azure Free Tier deployment rather than a production-scale system. Although appropriate for simulating SME adoption patterns, this environment lacked the complexity of real-world deployments, which may involve multi-tenant microservices, larger datasets, and more dynamic traffic patterns. Such factors can significantly influence vulnerability manifestation and detection.

Finally, the analysis relied on synthetic data and controlled exploitation. While necessary to maintain ethical integrity, this approach excludes the unpredictability of attacker behaviour in live contexts.

Together, these limitations underline the importance of interpreting the findings as a proof-of-concept contribution to SME-focused security practices rather than a definitive evaluation of all API security scenarios.

## 7.7 Future Improvements

Building on the limitations identified, several avenues for future enhancement are evident.

First, expanding the vulnerability coverage would provide a more comprehensive assessment. Future work could simulate additional OWASP API Security Top 10 categories—such as API4 (Unrestricted Resource Consumption) or API7 (Server-Side Request Forgery)—and extend testing to denial-of-service scenarios. This would strengthen the representativeness of the findings and address the breadth limitations of the current study.

Second, tighter integration with the CI/CD pipeline could operationalise continuous security testing. Embedding OWASP ZAP baseline scans, dependency checks, and static code analysis directly into GitHub Actions would align the methodology with DevSecOps practices and enhance reproducibility across software iterations.

Third, strengthening cryptographic controls would improve the realism of the artefact. Replacing the weak password implementation with bcrypt or Argon2 would demonstrate best practice for secure credential storage, while enabling comparative testing of authentication-focused vulnerabilities.

Fourth, comparative benchmarking against commercial solutions would add external validity. By evaluating the same vulnerabilities with enterprise-grade platforms (e.g., Burp Suite Professional, 42Crunch, or Salt Security), future research could quantify the trade-offs between cost-free and licensed approaches, offering SMEs evidence-based guidance on when commercial investment is justified.

Finally, collaboration with SMEs in applied case studies would provide ecological validity. Deploying the methodology in live operational contexts could reveal organisational, economic, and human factors that cannot be replicated in a controlled environment.

Collectively, these improvements would deepen empirical insights, extend the methodological rigour, and further align the research with the realities of SME adoption and contemporary API security practice.

## 7.8 Summary

This chapter has critically evaluated the outcomes of the vulnerability testing and positioned them within the wider discourse on API security. The analysis confirmed that free-tier and open-source tools provide SMEs with an accessible entry point into API security assurance but are insufficient when used in isolation. While OWASP ZAP proved effective at detecting generic misconfigurations and information disclosure, critical logic-based vulnerabilities—such as Broken Object Level Authorization and Mass Assignment—remained undetected without manual validation.

The CVSS v4.0 severity assessments reinforced that the most damaging vulnerabilities were those requiring contextual analysis, underlining the indispensable role of human expertise in complementing automation. These findings directly address the research gap identified in Chapter 2, demonstrating empirically that layered, free-tier toolchains can provide SMEs with practical security baselines, provided they are combined with disciplined testing methodologies.

However, the limitations of scope, tooling, and deployment environment highlight that further research is necessary to achieve generalisability and ecological validity. The future improvements outlined—ranging from expanded vulnerability coverage to CI/CD integration and comparative benchmarking—offer clear pathways for extending the study’s contribution.

Overall, this chapter has shown that the project achieved its objectives: it not only validated the feasibility of SME-relevant, cost-free API security strategies but also highlighted their limitations, thereby providing a balanced foundation for the concluding synthesis in Chapter 8.

# Chapter 8: Conclusion and Future Work

8.1 Introduction  
This chapter synthesises the findings of the study, drawing together the empirical results, analytical discussion, and methodological reflections to provide a conclusive assessment of the research objectives. The project investigated whether free-tier cloud services and open-source security tools can offer SMEs a viable baseline for API security, an area identified in Chapter 2 as underexplored in existing literature. By developing and testing a deliberately vulnerable FastAPI artefact on Microsoft Azure, the study has moved beyond theoretical assertions to provide evidence-based insights into the practical and strategic viability of SME-focused security practices.

The chapter consolidates key findings, critically reflects on achievements and limitations, and outlines avenues for future work. In doing so, it positions the project’s contribution within both academic debates and practitioner contexts, demonstrating its relevance to SMEs navigating the challenges of digital transformation under constrained resources.

## 8.2 Key Findings

The project produced several substantive findings that advance understanding of SME-relevant API security:

1. **Severity of Vulnerabilities**
   * The CVSS v4.0 assessment confirmed that logic-based vulnerabilities posed the greatest risk.
   * Mass Assignment (CVSS 8.6, High), Excessive Data Exposure (CVSS 8.7, High), and Broken User Authentication (CVSS 8.8, High) emerged as the most severe, enabling privilege escalation, data leakage, and brute-force compromise respectively.
   * Broken Object Level Authorization (CVSS 7.1, High) also represented a significant confidentiality threat, while Improper Asset Management (CVSS 6.9, Medium) was less critical but demonstrated the residual risks of poor lifecycle governance.
2. **Effectiveness of Security Controls**
   * Layered defences—Azure Entra ID (OAuth2/JWTs), TLS enforcement, Pydantic input validation, SlowAPI rate limiting, and structured logging—proved effective at mitigating all five deliberately embedded vulnerabilities.
   * These mitigations demonstrated that SMEs can achieve substantial risk reduction using free-tier services, provided that defences are applied systematically.
3. **Tool Performance and Triangulation**
   * OWASP ZAP reliably detected surface-level misconfigurations and data exposure but consistently failed to identify logic-based flaws such as BOLA and weak authentication.
   * Burp Suite CE and Insomnia were indispensable for manual validation, particularly in reproducing Mass Assignment and Broken Authentication exploits.
   * This triangulated toolset confirmed that automation must be complemented by human-driven testing to achieve meaningful assurance.
4. **Contribution to the Research Gap**
   * The study addressed the identified gap by empirically testing whether SMEs can realistically secure APIs using free-tier cloud and open-source resources.
   * Findings demonstrated that while such resources cannot replace enterprise-grade solutions, they do enable SMEs to establish a credible baseline of security when combined with structured processes and human oversight.
   * This evidences that cost is not the primary barrier; rather, the adoption of disciplined methodologies is central to SME resilience.

## 8.3 Critical Reflection

The project successfully demonstrated that a structured, design-science approach can produce both a working artefact and empirical evidence of tool effectiveness within SME-relevant constraints. A key strength lies in the integration of theory and practice: the artefact was not only developed and deployed but also systematically tested against real vulnerabilities, ensuring that findings were grounded in reproducible experimentation rather than abstract analysis.

Another strength was methodological transparency. By documenting each phase—from artefact construction to vulnerability exploitation, patching, and evaluation—the project provides a replicable template that other SMEs or researchers can adopt. The application of CVSS v4.0 and OWASP mapping also ensured analytical rigour, supporting comparison across tools and vulnerabilities.

However, the project was not without weaknesses. The scope was necessarily constrained to five vulnerabilities, limiting the breadth of conclusions. Furthermore, the reliance on free-tier tools exposed gaps in automation and reporting functionality, reaffirming that while cost-effective, these solutions cannot match enterprise-grade coverage.

Critically, the project highlights that effective API security is not primarily a technological challenge but a methodological one. SMEs equipped with structured processes and basic tooling can mitigate many common vulnerabilities, but success ultimately depends on human expertise, disciplined testing, and continuous monitoring. This insight contributes an original perspective to the discourse: that resource constraints need not equate to insecurity if rigorous, process-driven approaches are adopted.

## 8.4 Future Work

Building on the strengths and limitations of this study, several opportunities exist to extend its scope and deepen its contribution:

1. **Expanded Vulnerability Coverage**  
   Future research should broaden the simulation to cover additional OWASP API Top 10 categories, including injection attacks (API8), denial-of-service through resource exhaustion (API4), and insufficient monitoring (API10). This would provide a more holistic evaluation of free-tier defences across the full spectrum of API risks.
2. **Integration into CI/CD Pipelines**  
   Incorporating OWASP ZAP baseline scans, static application security testing (SAST), and dependency checks into GitHub Actions would align the methodology with DevSecOps practices. This would allow continuous detection of regressions and elevate the feasibility of adopting “security by design” within SMEs.
3. **Enhanced Cryptographic Practices**  
   While the project demonstrated authentication and rate-limiting controls, future work should implement stronger password storage using bcrypt or Argon2. This would provide a robust comparison between weak and strong cryptographic practices within an SME context.
4. **Comparative Benchmarking with Commercial Solutions**  
   To contextualise the value of free-tier approaches, a comparative study could evaluate trial versions of commercial security platforms (e.g., Burp Suite Professional, 42Crunch, Salt Security). This would enable SMEs to make evidence-based trade-offs between free and paid solutions.
5. **Real-World SME Case Studies**  
   Partnering with SMEs to deploy the methodology in live environments would test its practical applicability under operational constraints such as multi-tenancy, production-scale traffic, and limited staff expertise. Such studies would move the research from controlled academic experimentation to industry validation.

Collectively, these avenues would not only address the limitations identified in this study but also position future research to make stronger contributions at the intersection of academic inquiry and industry practice.

## 8.5 Conclusion

This project set out to determine whether free-tier cloud resources and open-source security tools can provide SMEs with a viable baseline for API security. By constructing a deliberately vulnerable FastAPI artefact, deploying it on Microsoft Azure Free Tier, and securing it with layered defences, the study demonstrated both the possibilities and limitations of a no-cost security model.

The empirical testing revealed that while free tools such as OWASP ZAP are effective for identifying configuration weaknesses and excessive data exposure, they consistently failed to detect critical logic flaws including BOLA and Mass Assignment. Manual tools (Burp Suite CE, Insomnia) proved indispensable for uncovering these contextual vulnerabilities, underscoring that automation alone cannot deliver comprehensive assurance.

By applying the CVSS v4.0 framework, the study provided structured evidence that the most severe risks were those requiring contextual awareness rather than mechanical scanning. This reinforces OWASP’s assertion that business-logic vulnerabilities remain among the hardest to detect yet most damaging in practice.

Academically, the project addresses a clear gap in the literature by empirically evaluating free-tier security approaches in SME-relevant conditions. Practically, it delivers a replicable roadmap that SMEs can adopt: layering TLS, authentication, input validation, rate limiting, and logging with free-tier and open-source tools, while recognising the non-negotiable role of human expertise.

The overall conclusion is that free solutions can substantially elevate SME security maturity, but only when combined with structured methodology, layered defences, and skilled oversight. This balance between cost-effectiveness and practical resilience is the critical insight and lasting contribution of the research.

Ultimately, this study proves that with layered defences and informed human oversight, SMEs can achieve meaningful API security without prohibitive investment.

# References

Akamai (2024). *New study finds 84% of security professionals experienced an API security incident in the past year*. [online] Akamai. Available at: https://www.akamai.com/newsroom/press-release/new-study-finds-84-of-security-professionals-experienced-an-api-security-incident-in-the-past-year [Accessed 13 June 2025].

Ali, M., Khan, S.U. and Vasilakos, A.V. (2015). *Security in cloud computing: Opportunities and challenges*. Information Sciences, 305, pp.357–383. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0020025515000638?via%3Dihub [Accessed 1 June 2025].

Asemi, H. (2023). *A Study on API Security Pentesting*. [online] ProQuest. Available at: https://www.proquest.com/openview/431b544974bbeb4cf446f1ec33fe341e/1?pq-origsite=gscholar&cbl=18750&diss=y [Accessed 2 June 2025].

Atlidakis, V., Godefroid, P. and Polishchuk, M. (2020). *Checking security properties of cloud service REST APIs*. Proceedings of the 2020 IEEE 13th International Conference on Software Testing, Verification and Validation (ICST), pp.159–170. Available at: https://ieeexplore.ieee.org/document/9159084[Accessed 1 June 2025].

Das, I. (2024). *Empowering Small Businesses in the Digital Age: A Must-Read Guide to Web Application & API Security*. Qualys Blog. Available at: https://blog.qualys.com/product-tech/2024/04/22/empowering-small-businesses-in-the-digital-age-a-must-read-guide-to-web-application-api-security [Accessed 19 June 2025].

European Union (2016). *General Data Protection Regulation (GDPR) Regulation (EU) 2016/679*. [online] Available at: https://eur-lex.europa.eu/eli/reg/2016/679/oj [Accessed 16 June 2025].

FastAPI (2025). *FastAPI – The modern, fast (high-performance) web framework for building APIs with Python 3.7+*. [online] Available at: https://fastapi.tiangolo.com/ [Accessed 22 May 2025].

GitHub (2025). *GitHub Actions – Automate your workflow from idea to production*. [online] Available at: https://github.com/features/actions [Accessed 19 June 2025].

Google (2024). *Gemini (Generative AI Model)*. [online] Available at: https://gemini.google.com [Accessed 13 July 2025].

Gupta, D., Singh, N. and Sharma, M. (2024). *Cloud Security Challenges and Solutions: A Review of Current Best Practices*. ResearchGate. Available at: https://www.researchgate.net/publication/387558426\_Cloud\_Security\_Challenges\_and\_Solutions\_A\_Review\_of\_Current\_Best\_Practices [Accessed 1 July 2025].

Hadi, A.J. and Nugroho, S. (2020). *Security analysis of REST API using OWASP ZAP and Postman*. Journal of Physics: Conference Series, 2094(3), p.032016. Available at: https://iopscience.iop.org/article/10.1088/1742-6596/2094/3/032016/pdf [Accessed 1 July 2025].

Uvicorn (2025). *Uvicorn: lightning-fast ASGI server implementation, using uvloop and httptools*. [online] Available at: https://www.uvicorn.org/ [Accessed 21 June 2025].

Hadjimichael, L. and Mitropoulos, K. (2023). *API Security Testing: The Challenges of Security Testing for Restful APIs*. ResearchGate. Available at: https://www.researchgate.net/publication/371174422\_API\_Security\_Testing\_The\_Challenges\_of\_Security\_Testing\_for\_Restful\_APIs [Accessed 1 July 2025].

Kong (2025). *Insomnia – REST and GraphQL client*. [online] Available at: https://insomnia.rest/ [Accessed 29 May 2025].

Krasniqi, F. (2018). *Improving Web Application Security Using Open-Source Tools*. Bachelor’s thesis. University of Gävle. Available at: https://core.ac.uk/reader/542974100 [Accessed 2 July 2025].

Logging, Python (2025). *logging — Logging facility for Python*. [online] Available at: https://docs.python.org/3/library/logging.html [Accessed 24 June 2025].

Madupati, B. (2023). *Comprehensive Approaches to API Security and Management in Large-Scale Microservices Environments*. SSRN. Available at: https://ssrn.com/abstract=5076630 [Accessed 15 June 2025].

Akamai (2025). *API Security Impact Study 2025*. MySecurity Marketplace. Available at: https://mysecuritymarketplace.com/reports/api-security-impact-study-2025/ [Accessed 1 July 2025].

Microsoft (2025). *Azure App Service – Build and host web apps*. [online] Microsoft Azure. Available at: https://azure.microsoft.com/en-us/products/app-service/ [Accessed 3 June 2025].

Microsoft (2025). *Microsoft Entra ID – Identity and access management*. [online] Microsoft. Available at: https://www.microsoft.com/en-us/security/business/microsoft-entra [Accessed 21 May 2025].

JSON Web Tokens (2025). *Introduction to JSON Web Tokens*. [online] Available at: https://jwt.io/introduction/ [Accessed 7 June 2025].

OAuth (2025). *OAuth 2.0 Authorization Framework*. [online] Available at: https://oauth.net/2/ [Accessed 25 June 2025].

Nandedkar, A. (2025). *The API economy: How APIs are shaping the future of digital transformation*. Medium. Available at: https://aditya-nandedkar.medium.com/the-api-economy-how-apis-are-shaping-the-future-of-digital-transformation-eb601e99f85e [Accessed 13 June 2025].

NinjaOne (2025). *7 SMB Cybersecurity Statistics You Need to Know in 2025*. Available at: https://www.ninjaone.com/blog/smb-cybersecurity-statistics/ [Accessed 2 July 2025].

OpenAI (2024). *ChatGPT (version GPT-4)*. [online] Available at: https://chat.openai.com [Accessed 13 July 2025].

OWASP Foundation (2023). *OWASP API Security Top 10 – 2023*. [online] Available at: https://owasp.org/API-Security/ [Accessed 10 June 2025].

OWASP ZAP (2025). *OWASP Zed Attack Proxy (ZAP) – Free security tool for finding vulnerabilities in web apps*. [online] OWASP Foundation. Available at: https://www.zaproxy.org/ [Accessed 25 May 2025].

PortSwigger (2025). *Burp Suite Community Edition*. [online] PortSwigger Ltd. Available at: https://portswigger.net/burp/communitydownload [Accessed 15 June 2025].

Pydantic (2025). *Pydantic – Data validation using Python type hints*. [online] Available at: https://docs.pydantic.dev/ [Accessed 7 June 2025].

Python-jose (2025). *python-jose – A JOSE implementation in Python*. [online] Available at: https://github.com/mpdavis/python-jose [Accessed 12 June 2025].

Python Software Foundation (2025). *Python 3.x Documentation*. [online] Available at: https://docs.python.org/3/ [Accessed 26 May 2025].

Rahmonbek, K. (2025). *35 Alarming Small Business Cybersecurity Statistics for 2025*. StrongDM Blog. Available at: https://www.strongdm.com/blog/small-business-cyber-security-statistics [Accessed 13 July 2025].

42Crunch (2023). *API Security Testing in VS Code* [online video]. YouTube. Available at: http://www.youtube.com/watch?v=l-5WuCH1XBM [Accessed 3 June 2025].

Raidiam (2025). *84% of enterprises risk sensitive data exposure due to API security gaps*. Intelligent CISO. Available at: https://www.intelligentciso.com/2025/07/02/raidiam-research-84-of-enterprises-risk-sensitive-data-exposure-due-to-api-security-gaps/ [Accessed 1 July 2025].

SentinelOne (2025). *Open Source Software Security Risks*. Available at: https://www.sentinelone.com/cybersecurity-101/cybersecurity/open-source-software-security-risks/ [Accessed 1 July 2025].

Sindall, G. (2024). *8 Enterprise API Security Strategy Best Practices*. DigitalML. Available at: https://www.digitalml.com/enterprise-api-security-strategy-best-practices/ [Accessed 17 June 2025].

SlowAPI (2025). *SlowAPI – Rate limiter for FastAPI*. [online] Available at: https://pypi.org/project/slowapi/ [Accessed 28 May 2025].

Ashish Mukarne (2021). *Insomnia API Testing* [online video]. YouTube. Available at: http://www.youtube.com/watch?v=nY\_w6kkdNI0 [Accessed 5 June 2025].

Thales Group (2024). *Vulnerable APIs and Bot Attacks Costing Businesses Up to $186 Billion Annually*. [online] Thales Group. Available at: https://cpl.thalesgroup.com/about-us/newsroom/vulnerable-apis-bot-attacks-costing-businesses-186-billion [Accessed 13 July 2025].

Traceable AI (2025). *2025 Global State of API Security*. [online] Available at: https://www.traceable.ai/wp-content/uploads/2024/10/2025-Global-State-of-API-Security.pdf [Accessed 1 July 2025].

Kong (2021). *Insomnia Tutorial: API Design, Testing and Collaboration* [online video]. YouTube. Available at: http://www.youtube.com/watch?v=fzLPHpOP3Wc [Accessed: 3 September 2025].

Mousavi, Z. et al. (2023). *Detecting Misuse of Security APIs: a systematic review*. [online] Available at: https://arxiv.org/pdf/2306.08869 [Accessed 12 June 2025].

UK Government (2018). *Data Protection Act 2018*. [online] Available at: https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted [Accessed 16 June 2025].

Uvicorn (2025). *Uvicorn – ASGI web server implementation for Python*. [online] Available at: https://www.uvicorn.org/ [Accessed 30 May 2025].

Verizon (2025). *2025 Data Breach Investigations Report: Small- and Medium-Sized Business Snapshot*. [online] Available at: https://www.verizon.com/business/resources/infographics/2025-dbir-smb-snapshot.pdf [Accessed 20 June 2025].

Kumar, A.A. and Divya, T.L (2024). *Security measures implemented in RESTful API development*. Open Access Research Journal of Engineering and Technology, 7(1), pp.105–112. [online] Available at: https://oarjpublication.com/journals/oarjet/content/security-measures-implemented-restful-api-development [Accessed 20 June 2025].

# Appendix A – Source Code (API)

Listing A.1 – Complete main.py implementing authentication, authorization, validation, rate limiting, and logging.

Final Patched Version

import os

import logging

from logging.handlers import RotatingFileHandler

from typing import Dict, Any, Optional, List

import requests

from fastapi import FastAPI, Depends, HTTPException, Request, Response, status

from fastapi.responses import JSONResponse

from fastapi.security import OAuth2AuthorizationCodeBearer

from pydantic import BaseModel, Field, EmailStr, ConfigDict

from jose import jwt, jwk

from jose.exceptions import JWTError, JWKError

from slowapi import Limiter

from slowapi.util import get\_remote\_address

from slowapi.errors import RateLimitExceeded

from slowapi.middleware import SlowAPIMiddleware

from dotenv import load\_dotenv

# =========================

# Environment

# =========================

load\_dotenv()

TENANT\_ID = os.getenv("TENANT\_ID")

CLIENT\_ID = os.getenv("CLIENT\_ID")

# Support one or many audiences (comma-separated), accepting either GUID or api://GUID

API\_AUDIENCE = os.getenv("API\_AUDIENCE", CLIENT\_ID) or ""

ALLOWED\_AUDIENCES: List[str] = [a.strip() for a in API\_AUDIENCE.split(",") if a.strip()]

if not ALLOWED\_AUDIENCES:

raise RuntimeError("API\_AUDIENCE must be set (e.g. 'cc4f...e3c' or 'api://cc4f...e3c').")

if not TENANT\_ID or not CLIENT\_ID:

raise RuntimeError("TENANT\_ID and CLIENT\_ID must be set for Azure mode.")

AUTH\_URL = f"https://login.microsoftonline.com/{TENANT\_ID}/oauth2/v2.0/authorize"

TOKEN\_URL = f"https://login.microsoftonline.com/{TENANT\_ID}/oauth2/v2.0/token"

JWKS\_URL = f"https://login.microsoftonline.com/{TENANT\_ID}/discovery/v2.0/keys"

ISSUER = f"https://login.microsoftonline.com/{TENANT\_ID}/v2.0"

# =========================

# Logging

# =========================

log\_file = "app.log"

log\_formatter = logging.Formatter("%(asctime)s - %(levelname)s - %(message)s")

log\_handler = RotatingFileHandler(log\_file, maxBytes=1\_000\_000, backupCount=3)

log\_handler.setFormatter(log\_formatter)

logger = logging.getLogger("api")

logger.setLevel(logging.INFO)

logger.addHandler(log\_handler)

# =========================

# FastAPI + Rate Limiting

# =========================

app = FastAPI(

title="Secure API (Azure Mode)",

description="FastAPI API protected by Microsoft Entra ID (OAuth2/JWT), with OWASP-aligned mitigations.",

version="2.0.0",

)

limiter = Limiter(key\_func=get\_remote\_address)

app.state.limiter = limiter

app.add\_middleware(SlowAPIMiddleware)

# =========================

# Security Headers Middleware

# =========================

SECURITY\_HEADERS = os.getenv("SECURITY\_HEADERS", "true").lower() == "true"

@app.middleware("http")

async def add\_security\_headers(request: Request, call\_next):

response: Response = await call\_next(request)

if SECURITY\_HEADERS:

response.headers.setdefault("X-Content-Type-Options", "nosniff")

response.headers.setdefault("X-Frame-Options", "DENY")

response.headers.setdefault("Referrer-Policy", "no-referrer")

response.headers.setdefault("Permissions-Policy", "geolocation=(), microphone=(), camera=()")

# CSP typically for HTML; omit for pure JSON APIs unless serving pages

# response.headers.setdefault("Content-Security-Policy", "default-src 'none'; frame-ancestors 'none';")

return response

# =========================

# OAuth2 Scheme

# =========================

oauth2\_scheme = OAuth2AuthorizationCodeBearer(

authorizationUrl=AUTH\_URL,

tokenUrl=TOKEN\_URL,

scopes={"access\_as\_user": "Access API as user"},

)

# =========================

# JWKS Cache + Token Verification

# =========================

JWKS\_CACHE: Dict[str, Any] = {"keys": []}

def fetch\_jwks() -> Dict[str, Any]:

resp = requests.get(JWKS\_URL, timeout=10)

resp.raise\_for\_status()

return resp.json()

def get\_public\_key\_for\_kid(kid: str) -> jwk.Key:

# Try cached first

key\_data = next((k for k in JWKS\_CACHE.get("keys", []) if k.get("kid") == kid), None)

if key\_data:

return jwk.construct(key\_data)

# Refresh cache and try again

try:

JWKS\_CACHE.update(fetch\_jwks())

except Exception as e:

logger.error(f"Failed to refresh JWKS: {e}")

raise HTTPException(status\_code=401, detail="Unable to refresh JWKS")

key\_data = next((k for k in JWKS\_CACHE.get("keys", []) if k.get("kid") == kid), None)

if not key\_data:

raise HTTPException(status\_code=401, detail="Signing key not found for token")

return jwk.construct(key\_data)

def verify\_audience(aud: Any) -> bool:

# Azure may put a single string in "aud"; sometimes clients look at "azp".

if isinstance(aud, str):

return aud in ALLOWED\_AUDIENCES

if isinstance(aud, list):

return any(a in ALLOWED\_AUDIENCES for a in aud)

return False

def verify\_token(token: str) -> Dict[str, Any]:

try:

headers = jwt.get\_unverified\_header(token)

kid = headers.get("kid")

if not kid:

raise HTTPException(status\_code=401, detail="Token missing 'kid'")

public\_key = get\_public\_key\_for\_kid(kid)

# Decode & validate standard claims

payload = jwt.decode(

token,

public\_key.to\_pem().decode(),

algorithms=["RS256"], # Azure v2.0 issues RS256 today

audience=ALLOWED\_AUDIENCES, # accepts list

issuer=ISSUER,

options={"require\_aud": True, "require\_iat": True, "require\_exp": True},

)

# Extra guard for odd cases where library didn't match audience list

if not verify\_audience(payload.get("aud")):

raise HTTPException(status\_code=401, detail="Invalid audience")

return payload

except (JWTError, JWKError) as e:

logger.error(f"JWT verification failed: {e}")

raise HTTPException(status\_code=401, detail="Invalid token")

except HTTPException:

raise

except Exception as e:

logger.error(f"Unexpected auth error: {e}")

raise HTTPException(status\_code=401, detail="Invalid token")

async def get\_current\_user(token: str = Depends(oauth2\_scheme)) -> Dict[str, Any]:

return verify\_token(token)

# =========================

# Data Models (Pydantic)

# =========================

class SecureDataRequest(BaseModel):

username: str = Field(..., min\_length=3, max\_length=50, pattern=r"^[a-zA-Z0-9\_-]+$")

email: EmailStr

comment: Optional[str] = Field(None, max\_length=500)

class CreateUserInput(BaseModel):

# Patched mass-assignment: forbid extra fields; only allow safe fields

model\_config = ConfigDict(extra="forbid")

username: str

email: EmailStr

# Auth model for demo /login (kept for evaluation, now patched)

class LoginRequest(BaseModel):

username: str = Field(..., min\_length=3, max\_length=50, pattern=r"^[a-zA-Z0-9\_-]+$")

password: str = Field(..., min\_length=1, max\_length=128)

# =========================

# In-memory Data (demo only)

# =========================

# NOTE: In production use a database/service, not in-memory globals.

USERS\_DB: Dict[str, Dict[str, Any]] = {

"user1": {"id": "user1", "username": "alice", "email": "alice@example.com", "role": "Admin"},

"user2": {"id": "user2", "username": "bob", "email": "bob@example.com", "role": "User"},

}

# Map preferred\_username/email to user\_id for ownership checks

EMAIL\_TO\_ID = {"alice@example.com": "user1", "bob@example.com": "user2"}

# =========================

# Routes

# =========================

@app.get("/", summary="Health")

def health():

return {"message": "API running (Azure mode)"}

@app.get("/\_whoami", summary="Return claims (debug)")

def whoami(user=Depends(get\_current\_user)):

# Careful: do not log/return sensitive claims in prod; this is for testing

return {

"preferred\_username": user.get("preferred\_username"),

"oid": user.get("oid"),

"roles": user.get("roles", []),

"aud": user.get("aud"),

"iss": user.get("iss"),

"tid": user.get("tid"),

"scp": user.get("scp"),

"azp": user.get("azp"),

}

@app.post("/login", summary="Demo login (patched: rate limiting & logging; no weak auth accepted)")

@limiter.limit("5/minute")

def login(request: Request, body: LoginRequest):

"""

Patched behaviour for study endpoint:

- Rate limiting & logging enforced.

- Weak passwords (e.g., '123456') are NOT accepted.

- No tokens are issued here; primary auth is Azure Entra ID OAuth2/JWT.

Future work: integrate bcrypt/argon2 for local credential flows if required.

"""

username = body.username.strip().lower()

logger.info(f"LOGIN ATTEMPT user={username} ip={request.client.host}")

# Always reject (this demo endpoint is retained only for evaluation with controls applied)

raise HTTPException(status\_code=status.HTTP\_401\_UNAUTHORIZED, detail="Invalid credentials")

@app.get("/secure-data", summary="Admin-only secure data")

@limiter.limit("5/minute")

def secure\_data(request: Request, user=Depends(get\_current\_user)):

roles = user.get("roles") or []

if "Admin" not in roles:

raise HTTPException(status\_code=403, detail="Admin role required")

logger.info(f"{user.get('preferred\_username')} accessed /secure-data")

return {"message": "Secure data", "user": user.get("preferred\_username")}

@app.post("/submit-data", summary="Submit user data (validated)")

@limiter.limit("10/minute")

def submit\_data(request: Request, payload: SecureDataRequest, user=Depends(get\_current\_user)):

logger.info(f"{user.get('preferred\_username')} submitted data {payload.model\_dump()}")

return {"message": "Data received", "submitted": payload.model\_dump()}

@app.get("/users/{user\_id}", summary="Get user (BOLA patched: ownership/RBAC enforced)")

@limiter.limit("10/minute")

def get\_user\_by\_id(request: Request, user\_id: str, user=Depends(get\_current\_user)):

record = USERS\_DB.get(user\_id)

if not record:

raise HTTPException(status\_code=404, detail="User not found")

# Ownership (user can access their own record) or Admin role can access any

caller\_email = user.get("preferred\_username")

caller\_id = EMAIL\_TO\_ID.get(caller\_email, user.get("oid") or user.get("sub"))

roles = user.get("roles") or []

if caller\_id != user\_id and "Admin" not in roles:

raise HTTPException(status\_code=403, detail="Forbidden (ownership check)")

return {

"id": record["id"],

"username": record["username"],

"email": record["email"],

"role": record["role"],

}

@app.post("/users", summary="Create user (mass-assignment patched)")

@limiter.limit("10/minute")

def create\_user(request: Request, body: CreateUserInput, user=Depends(get\_current\_user)):

# Server-controlled defaults only

new\_id = body.username.lower()

if new\_id in USERS\_DB:

raise HTTPException(status\_code=409, detail="User already exists")

USERS\_DB[new\_id] = {

"id": new\_id,

"username": body.username,

"email": body.email,

"role": "User", # enforced by server, not client

}

logger.info(f"{user.get('preferred\_username')} created user {new\_id}")

return {"created": USERS\_DB[new\_id]}

# OpenAPI export (useful for ZAP import)

@app.get("/openapi.json", include\_in\_schema=False)

def openapi\_spec():

return app.openapi()

# =========================

# Error Handlers

# =========================

@app.exception\_handler(RateLimitExceeded)

async def ratelimit\_handler(request: Request, exc: RateLimitExceeded):

return JSONResponse(status\_code=429, content={"detail": "Rate limit exceeded"})

Listing A.2 – requirements.txt

fastapi

uvicorn

python-jose

requests

pydantic

slowapi

starlette

python-dotenv

Listing A.3 – .env

# =====================

# Azure Entra ID Config

# =====================

TENANT\_ID=c63b9b68-7553-4343-ada9-64a942bfef32

CLIENT\_ID=cc4ff219-81a0-4f5e-a3ca-bc92b67eea3c

API\_AUDIENCE=cc4ff219-81a0-4f5e-a3ca-bc92b67eea3c

AUTH\_URL=https://login.microsoftonline.com/c63b9b68-7553-4343-ada9-64a942bfef32/oauth2/v2.0/authorize

TOKEN\_URL=https://login.microsoftonline.com/c63b9b68-7553-4343-ada9-64a942bfef32/oauth2/v2.0/token

JWKS\_URL=https://login.microsoftonline.com/c63b9b68-7553-4343-ada9-64a942bfef32/discovery/v2.0/keys

SECURITY\_HEADERS=true

# Appendix B – Vulnerability Testing Evidence

This appendix provides supporting evidence of the vulnerabilities introduced into the FastAPI artefact and their patched behaviour. For each case, screenshots are referenced (baseline exploit vs. patched response), followed by a summary table.

**B.1 Oauth settings**

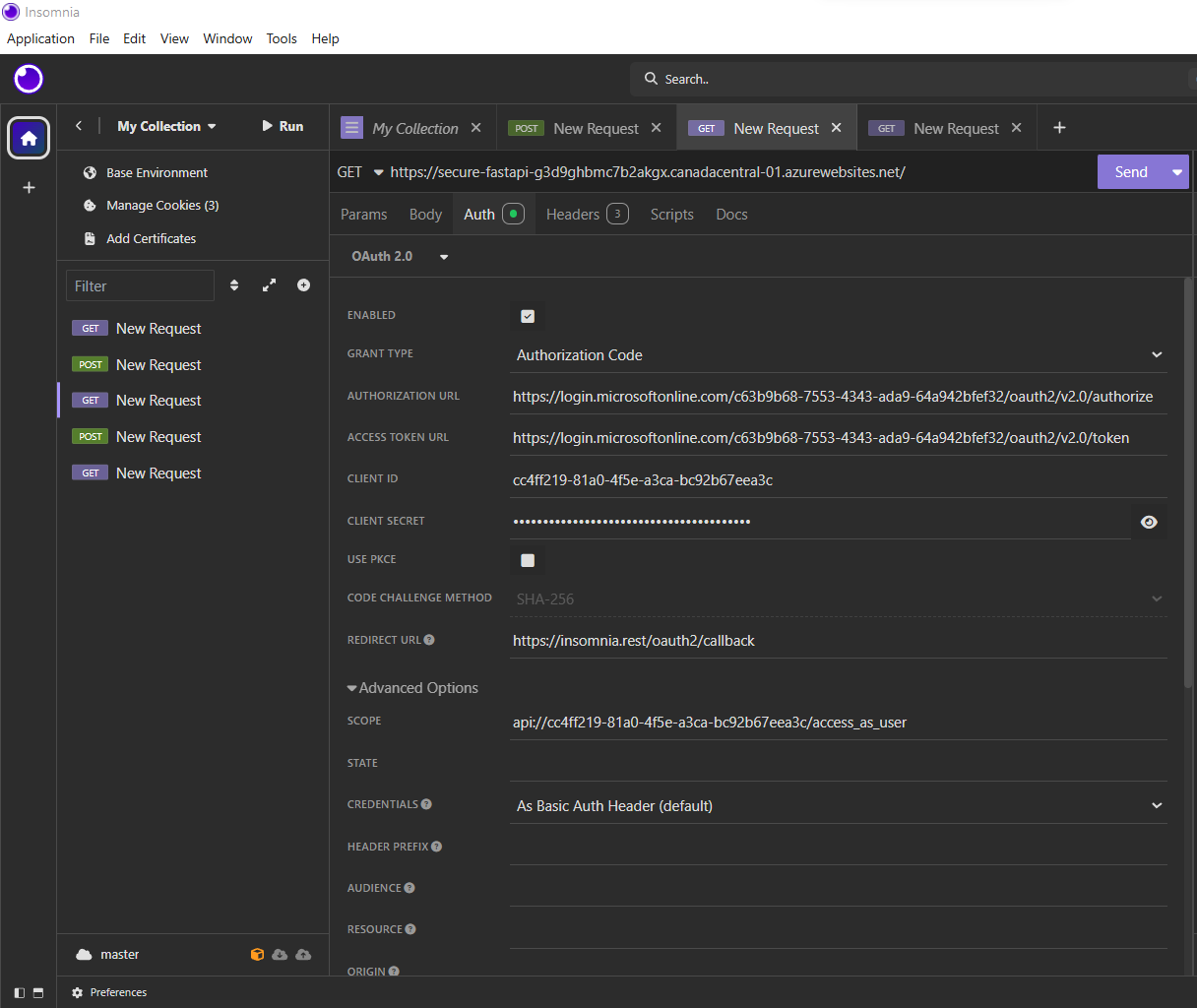


Figure B.1 - Insomnia OAuth settings.

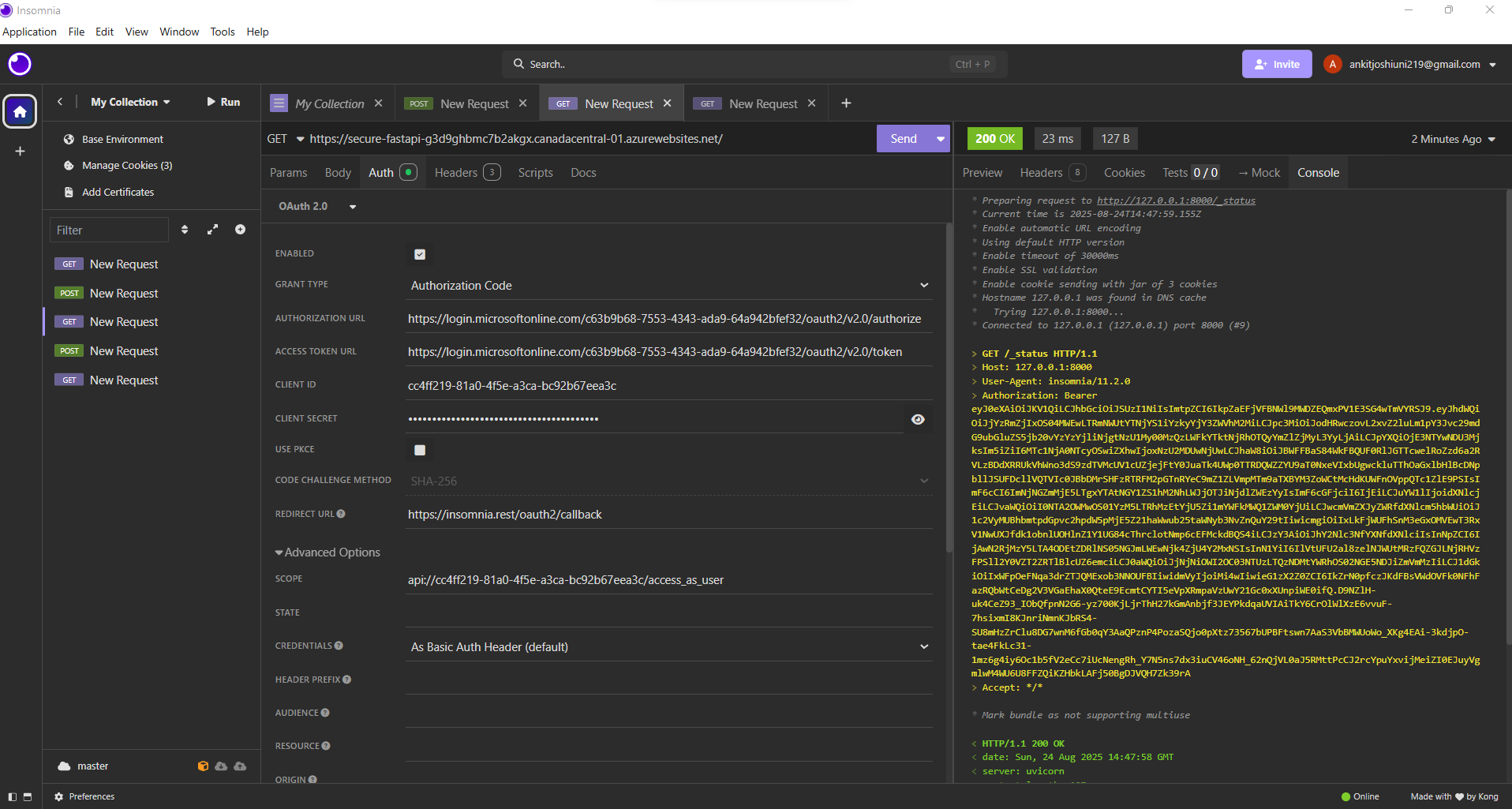


Figure B.2 - Token details page in Insomnia.

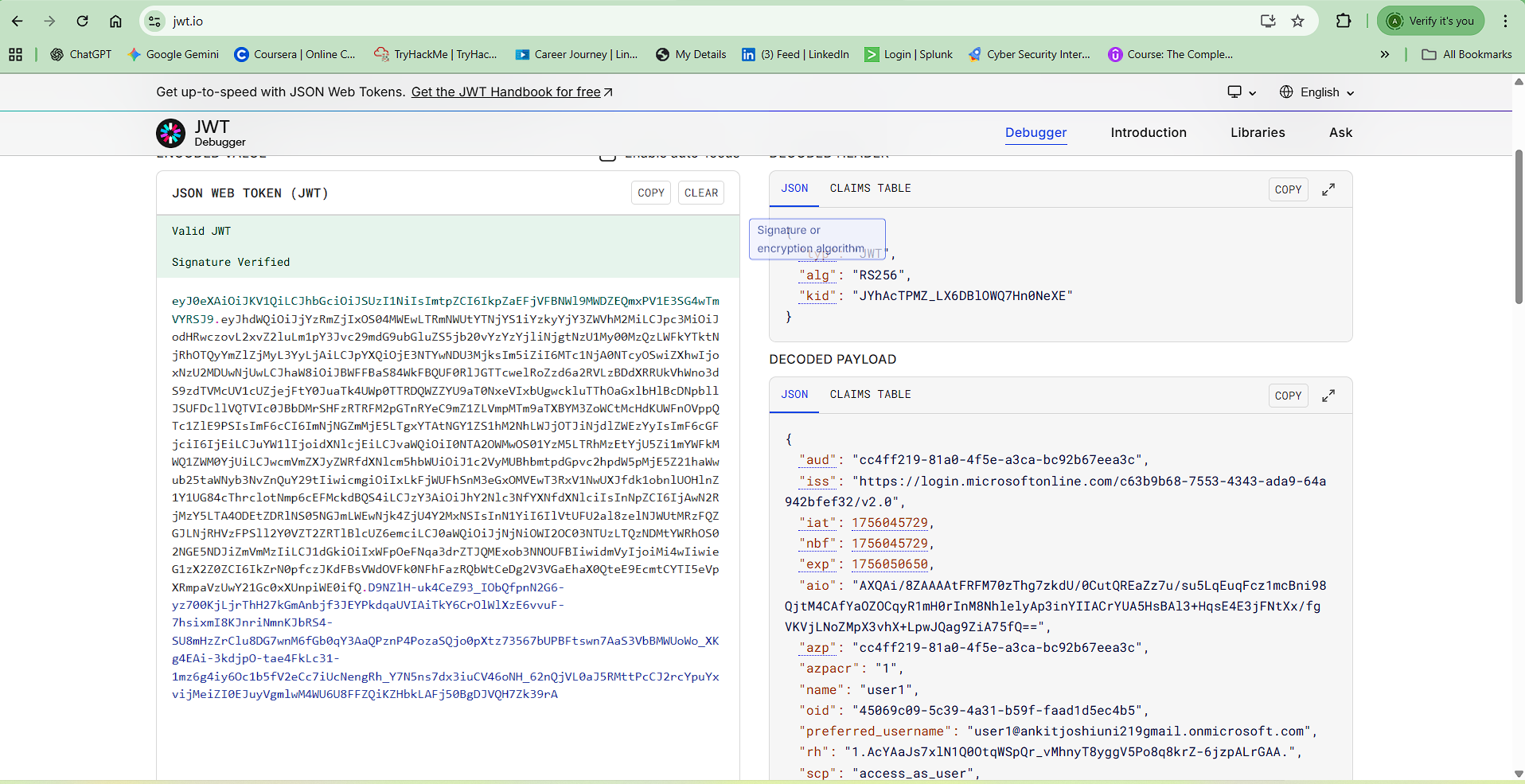
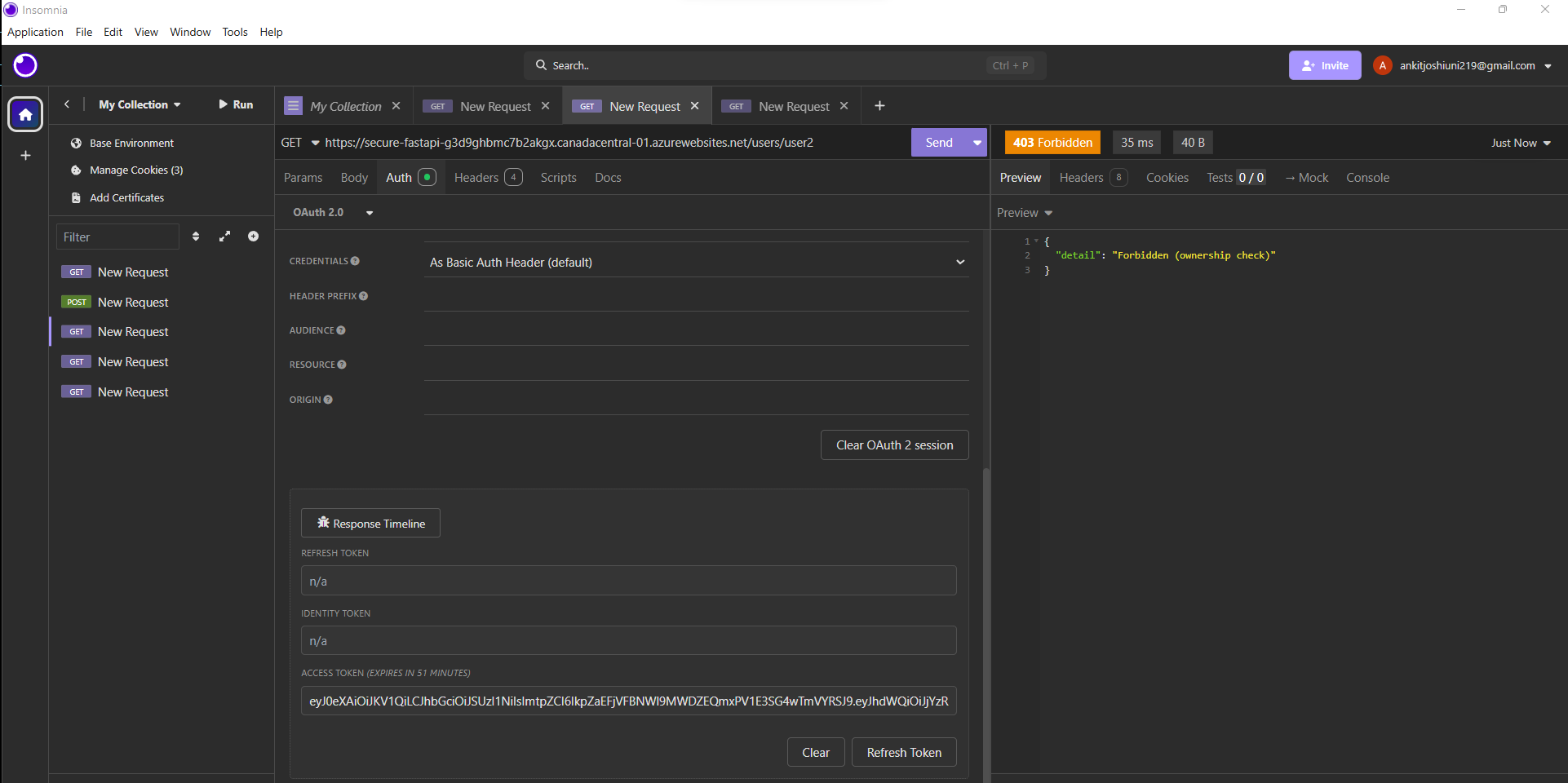


Figure B.3 - Token claims view (or jwt.io) showing ver: "2.0", iss ends with /v2.0, aud = your client id GUID.

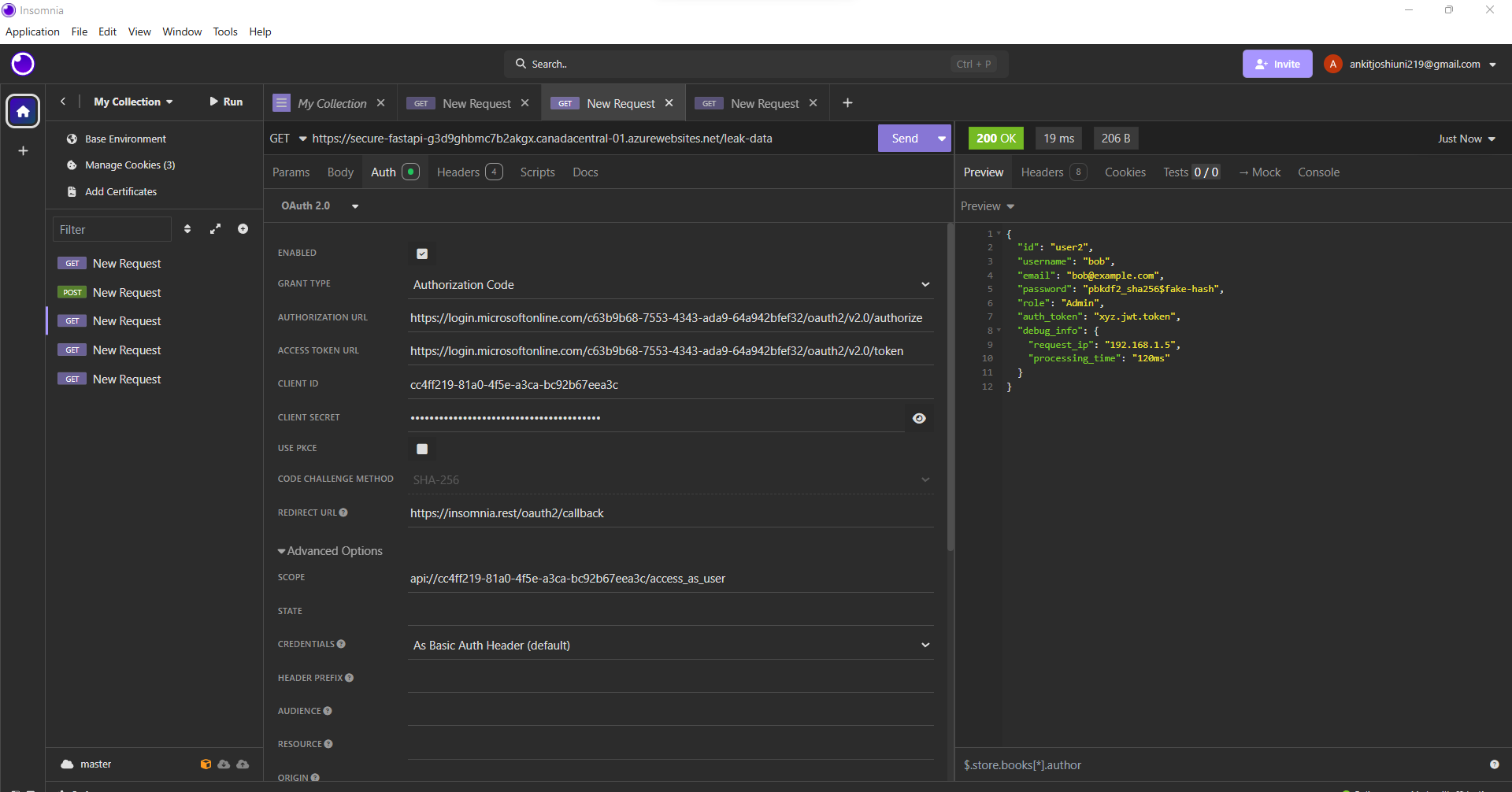
**B.2 Broken Object Level Authorization (API1)**

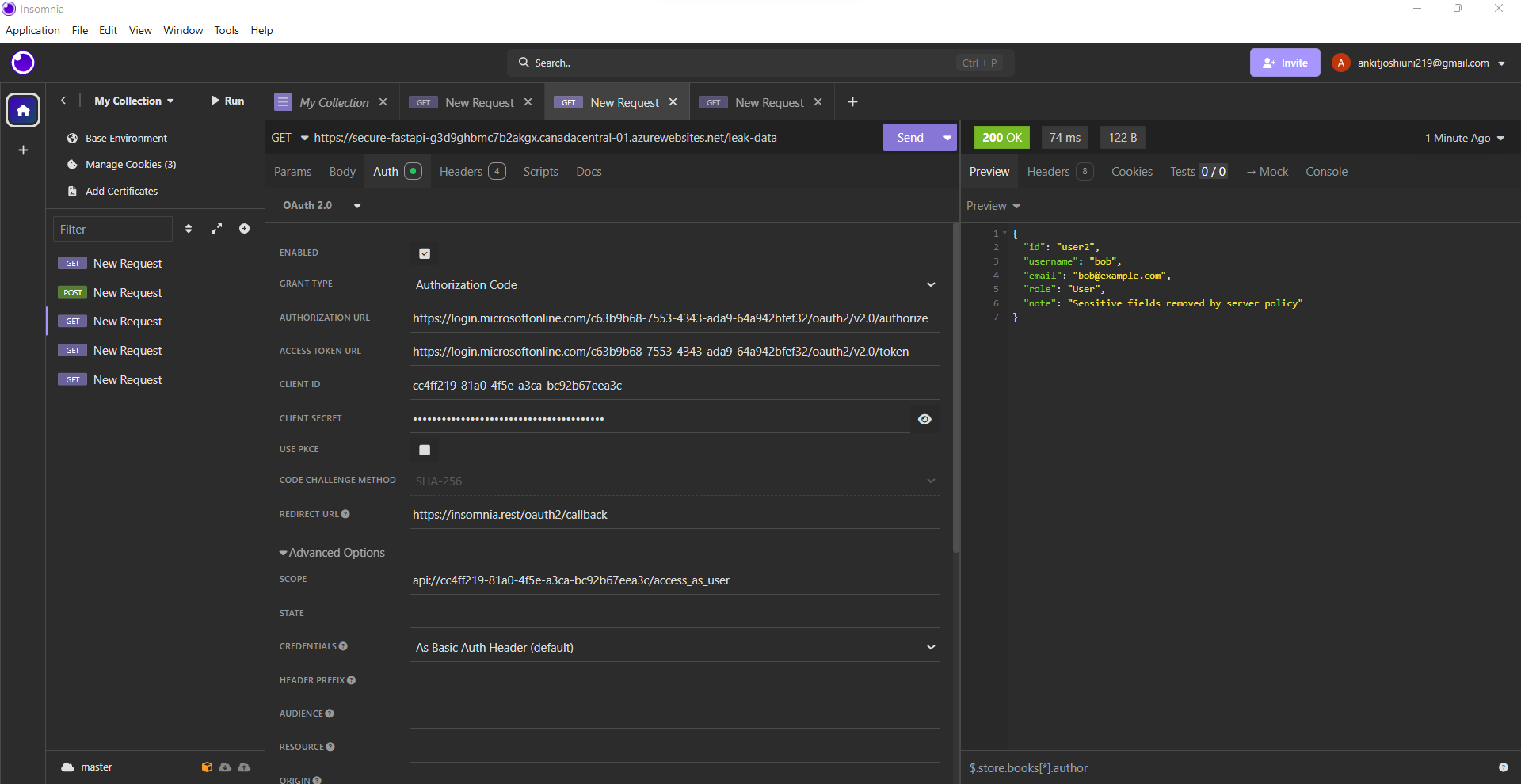
Figure B.4 – Insomnia request showing Alice retrieving Bob’s data (/users/user2).



**Figure B.5** – Insomnia post-patch response returning {"detail":"Forbidden (ownership check)"}.

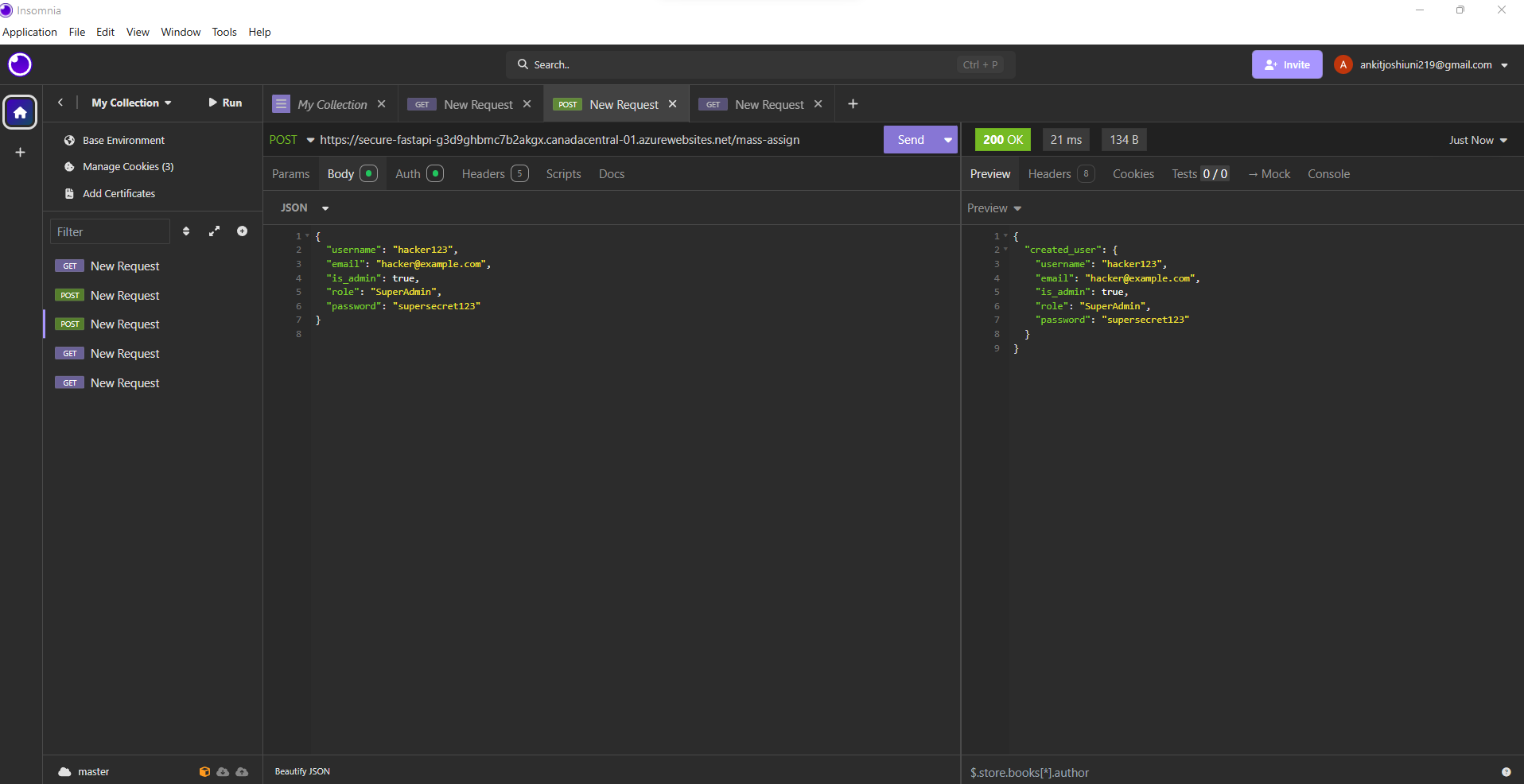
**B.3 Excessive Data Exposure (API3)**

Figure B.6 – Insomnia response exposing password hash, JWT token, debug info.

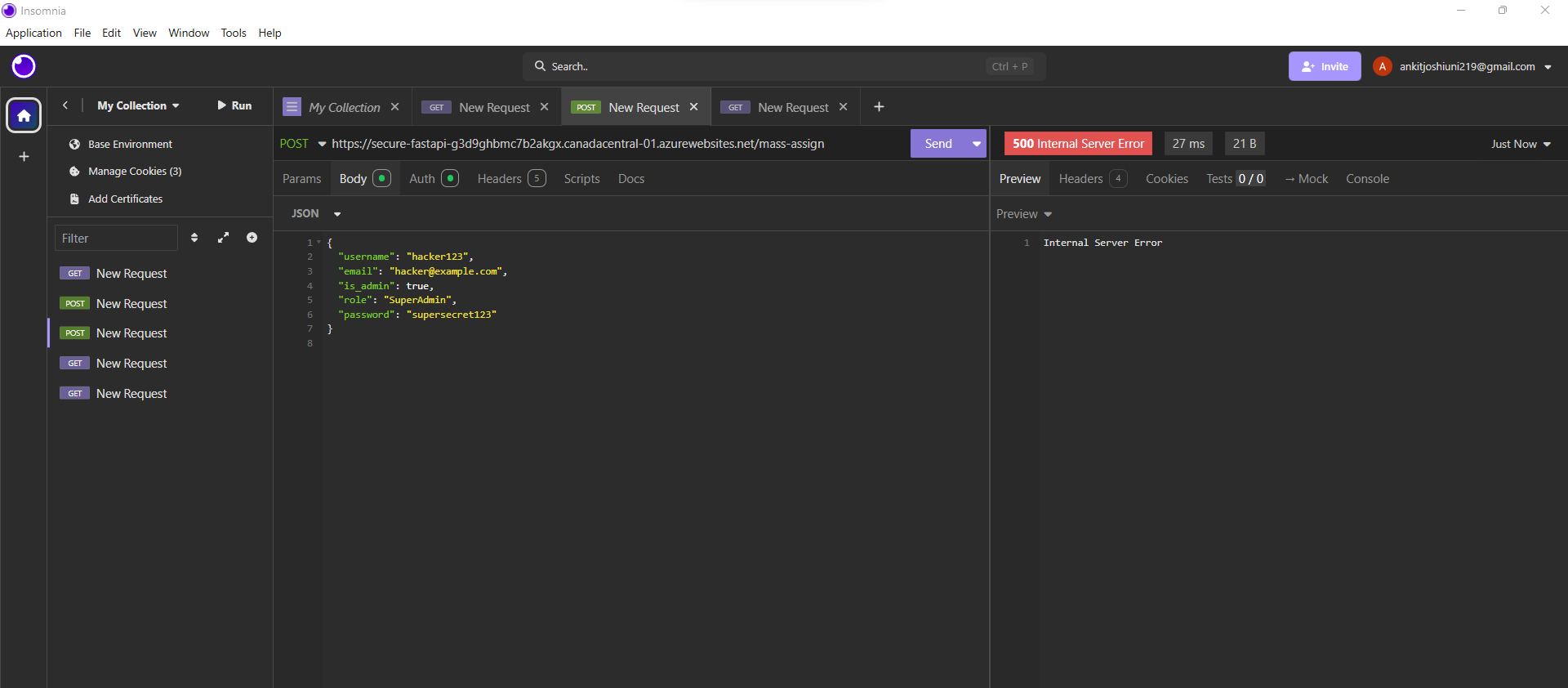


**Figure B.7** – Insomnia post-patch response showing only safe attributes (username, role, email).

**B.4 Mass Assignment (API6)**



**Figure B.8 -** response echoing attacker-controlled fields.



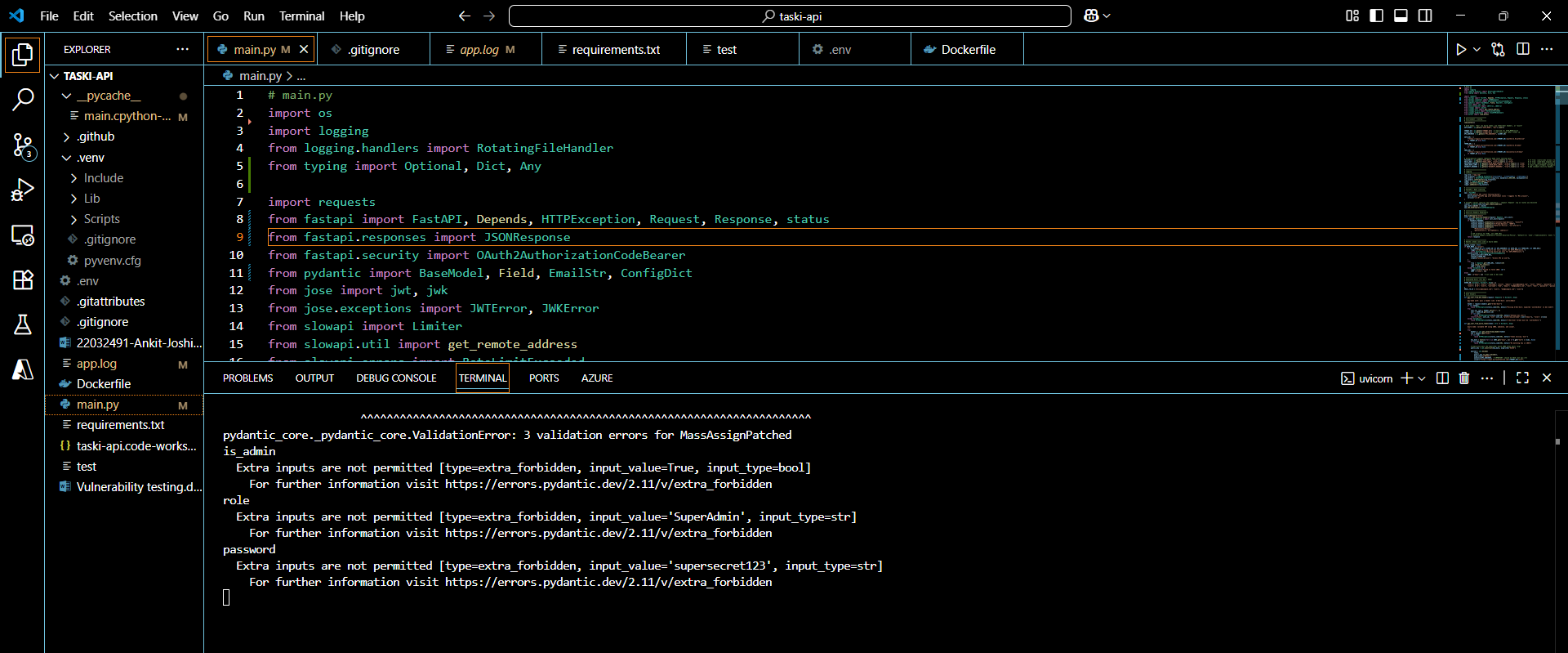
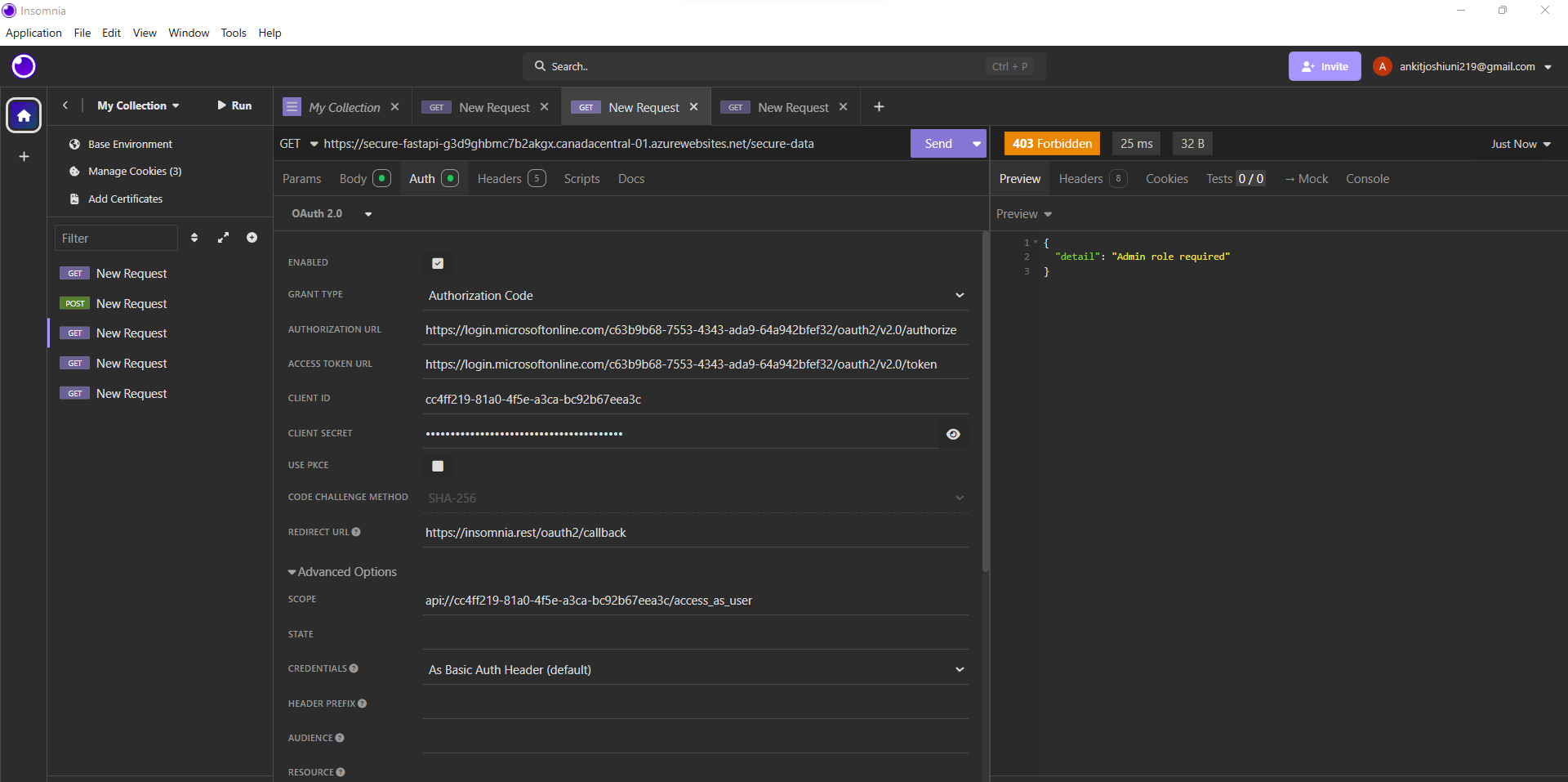
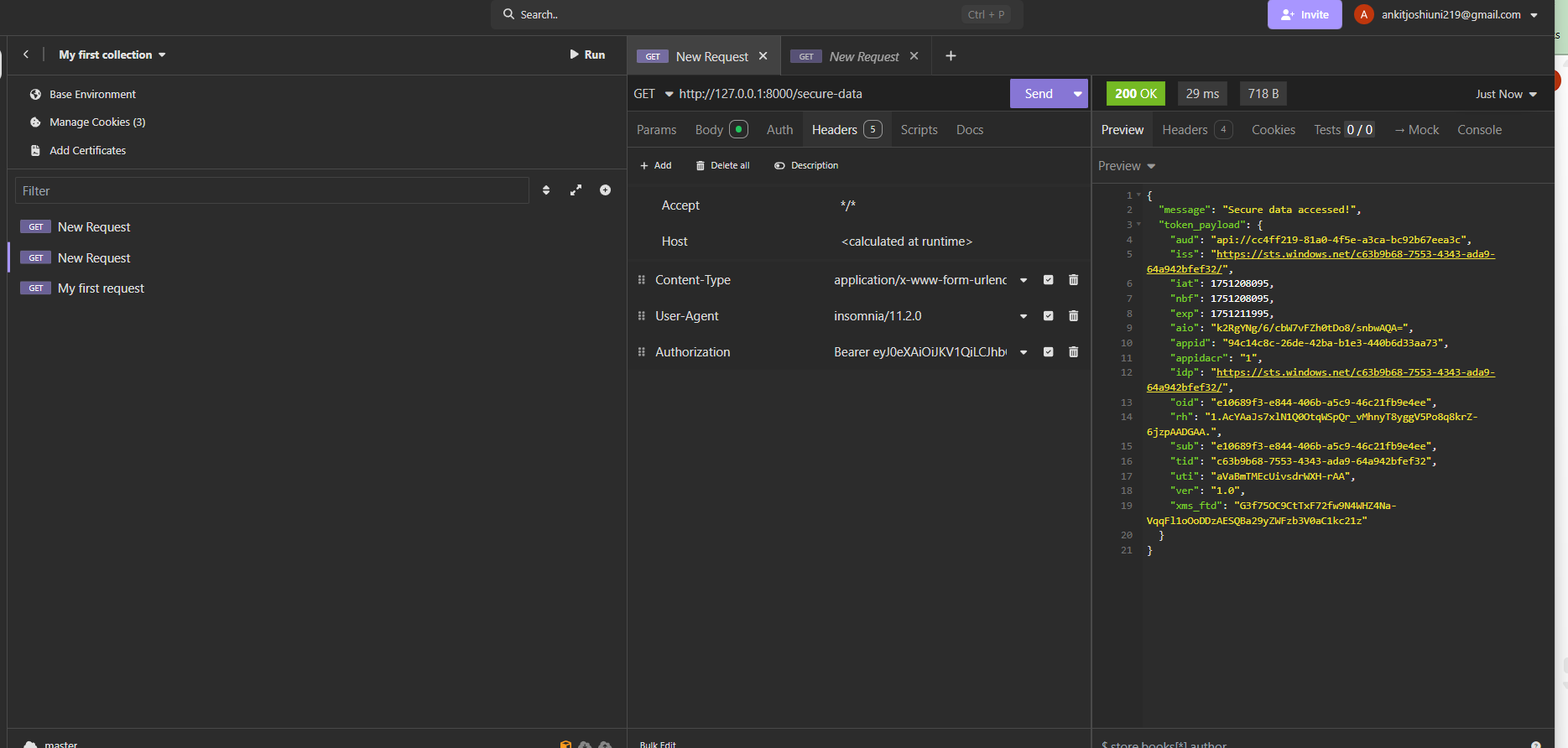


Figure B.9 error (or sanitized response).

**B.5 RBAC on secure endpoint**



**Figure B.10 -** 403 without role.



**Figure B.11 -** 200 with Admin role.

**B.6 Rate limiting**

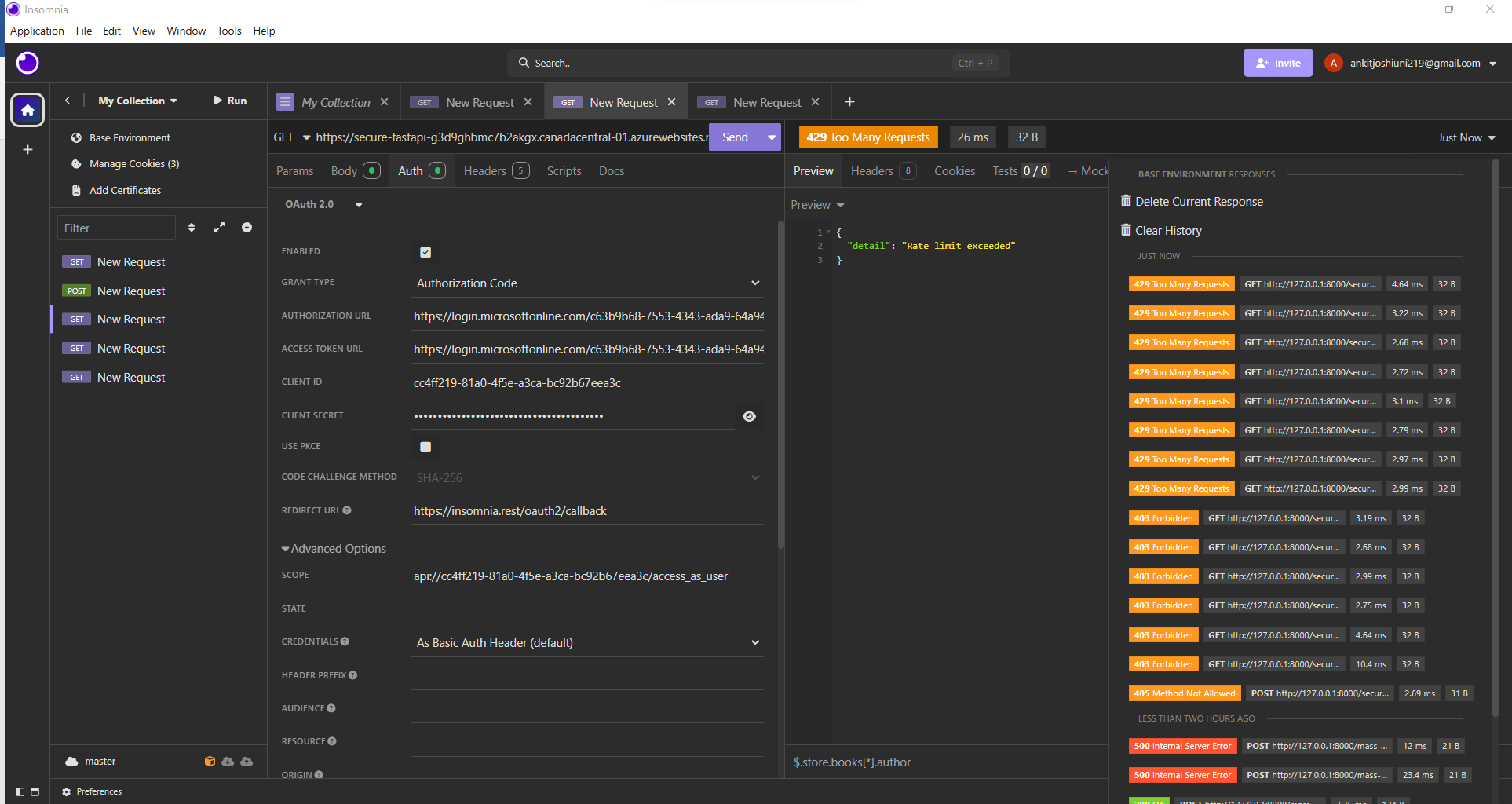


Figure B.12 – Rate limiting in function

**B.7 ZAP SCANS**

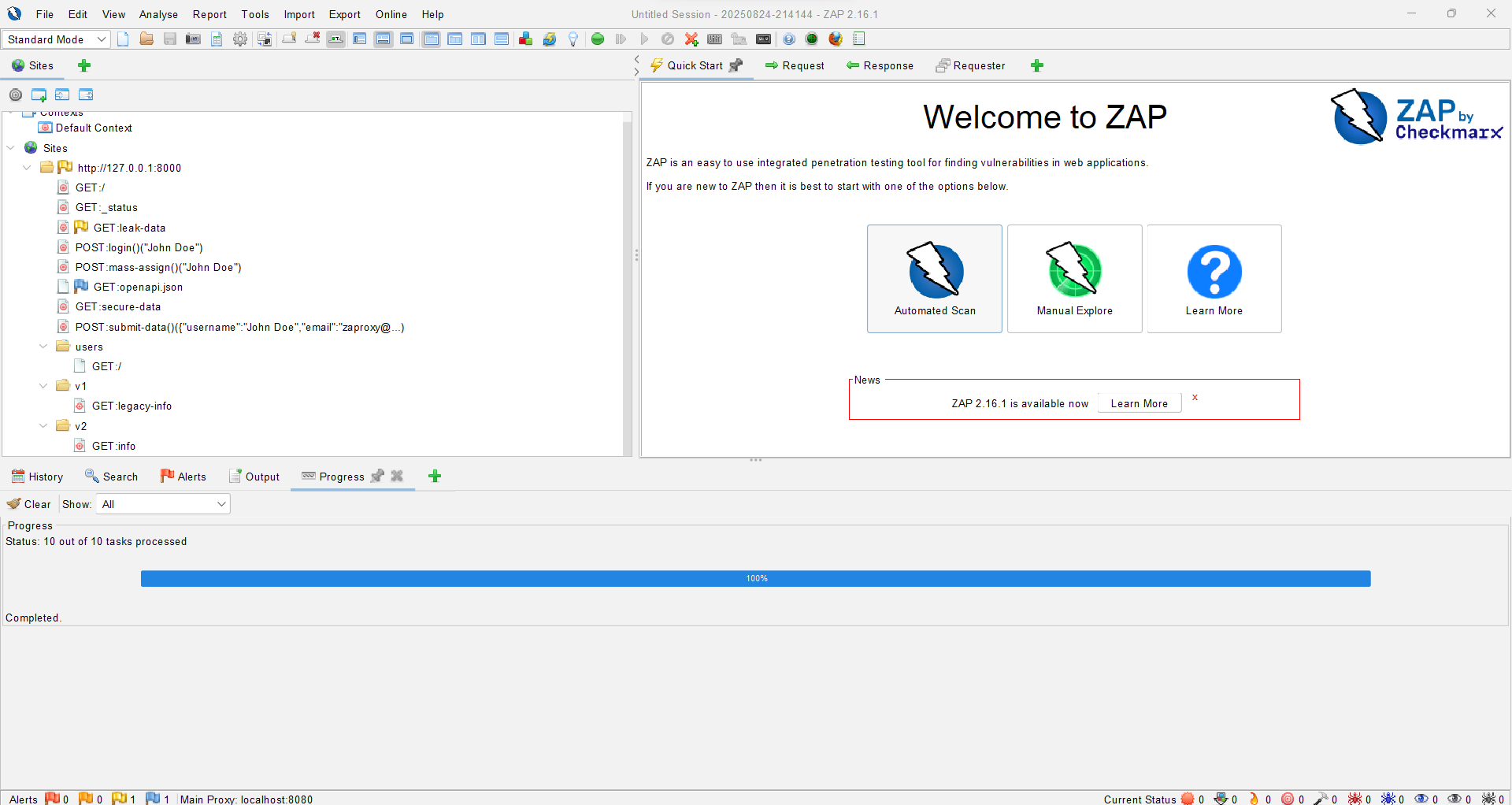


Figure B.13 - ZAP Sites tree with API endpoints listed.

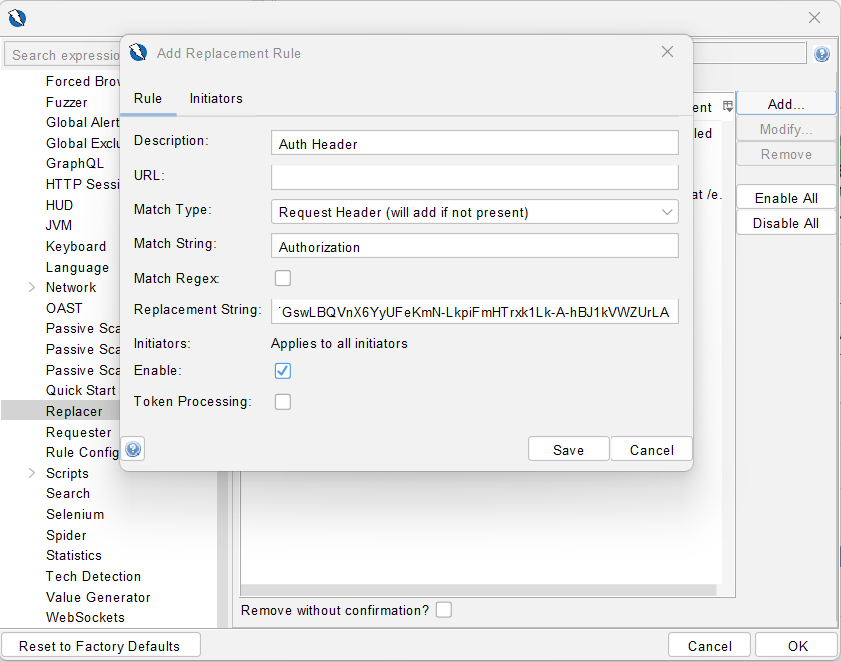


Figure B.14 - ZAP Replacer rule with Authorization Bearer token configured.

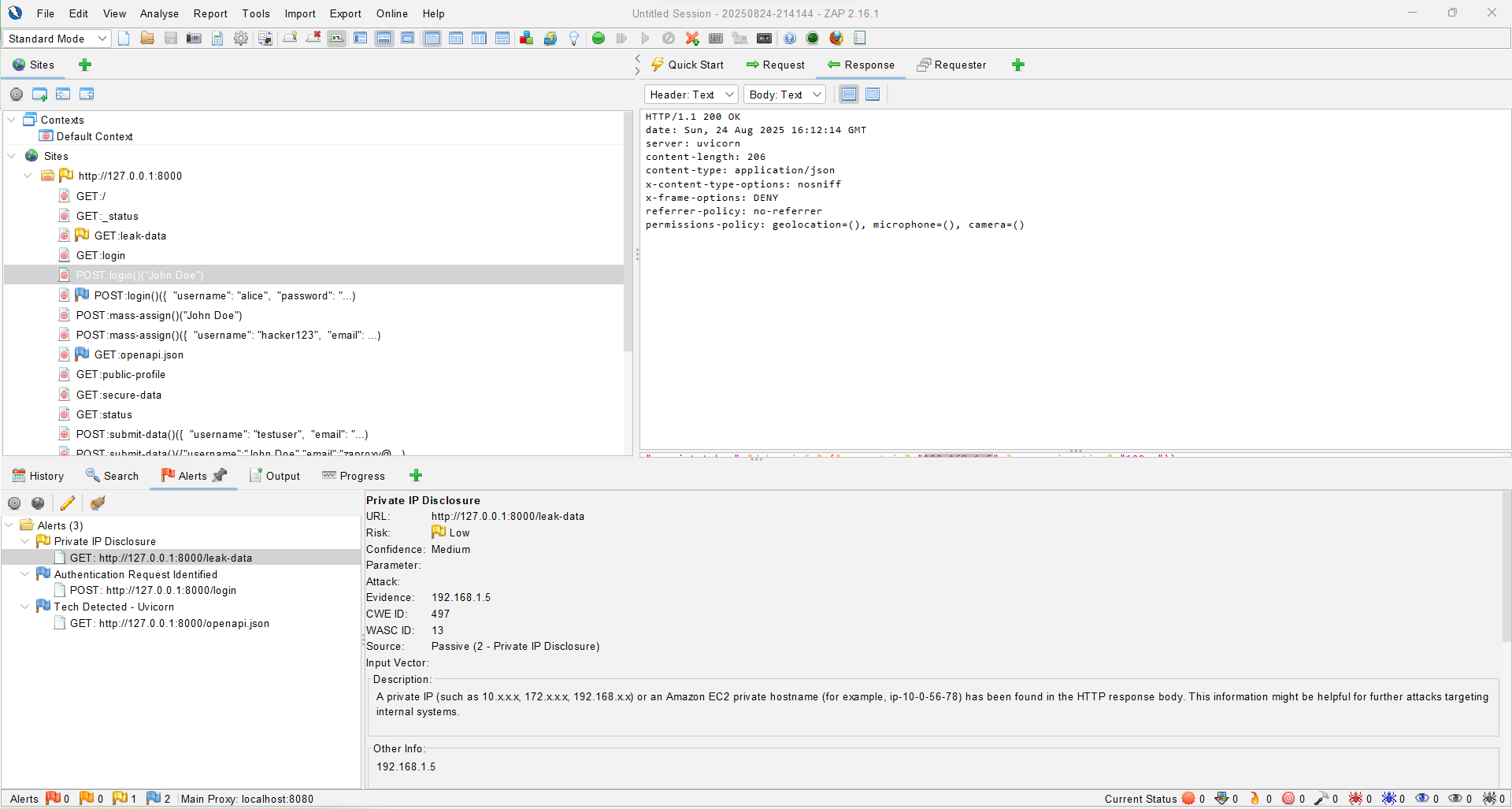


Figure B.15 - Passive scan

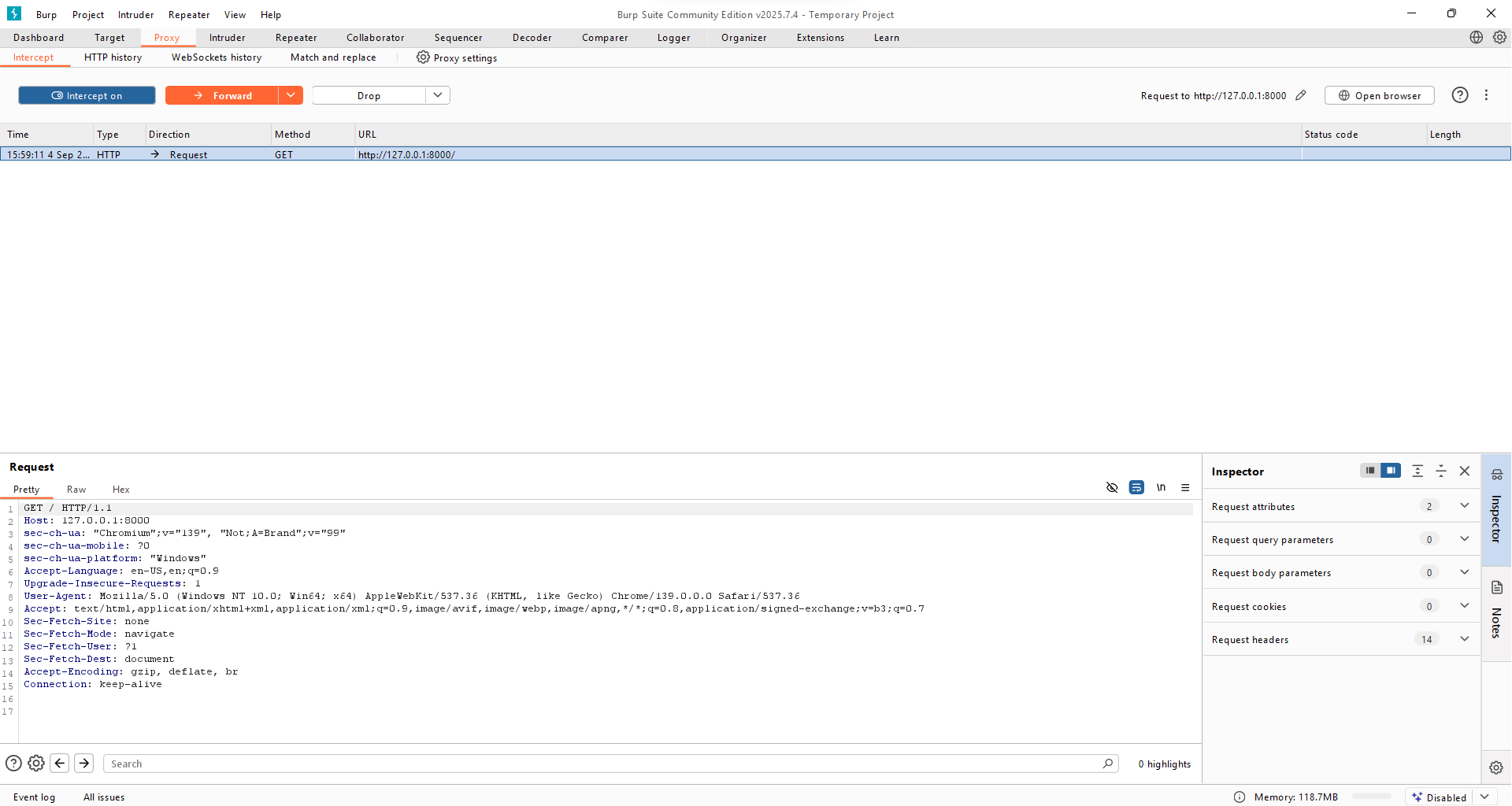


Figure B.16 - Burp Suite Proxy (Intercept) capturing the initial HTTP request to the FastAPI service, confirming successful proxy configuration and traffic interception.

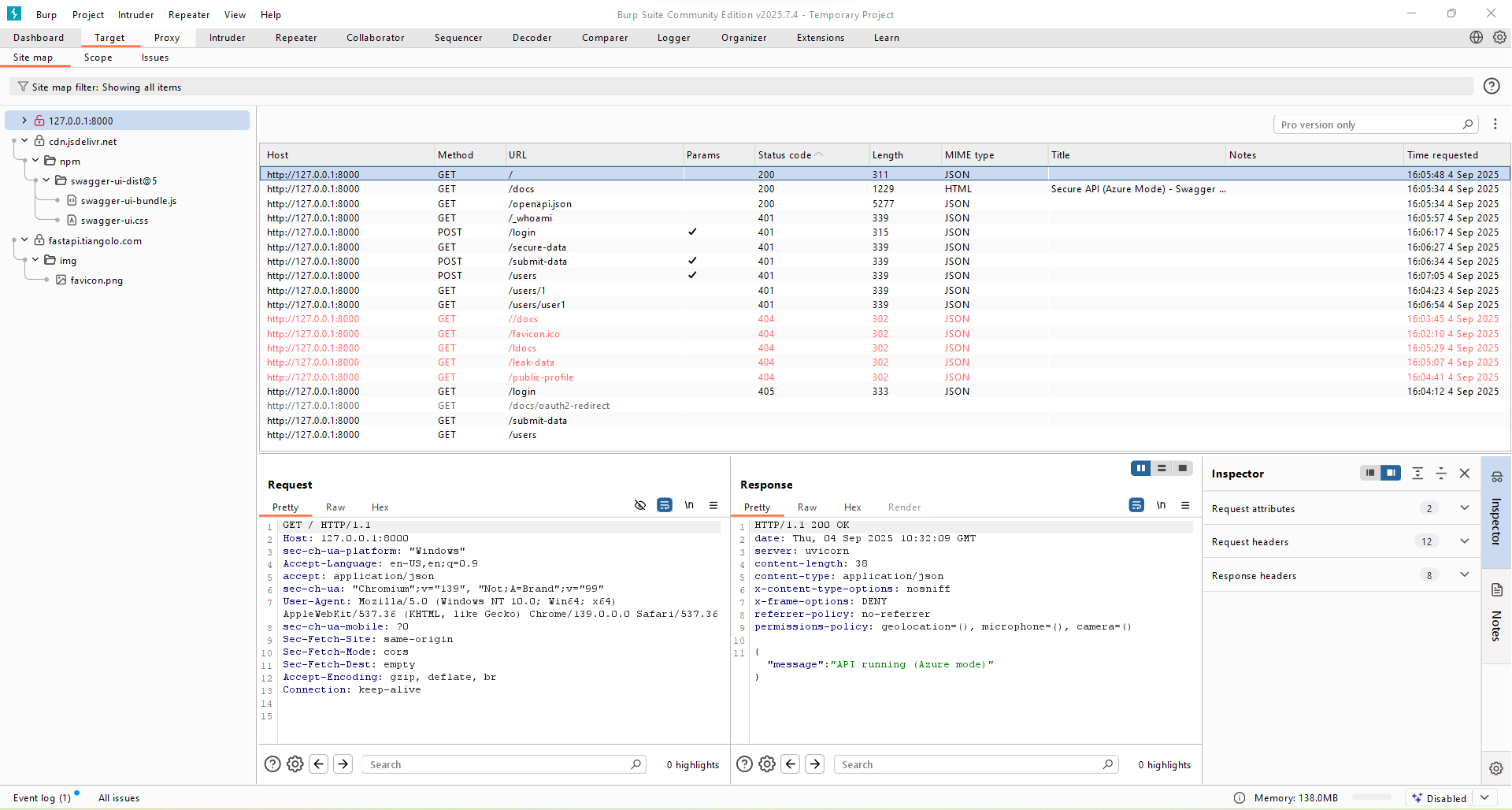


Figure B.17 - Burp Suite Target Site Map showing the enumerated API endpoints (/docs, /users/{id}, /public-profile, /login), demonstrating automated endpoint discovery and mapping of the FastAPI service for security assessment.

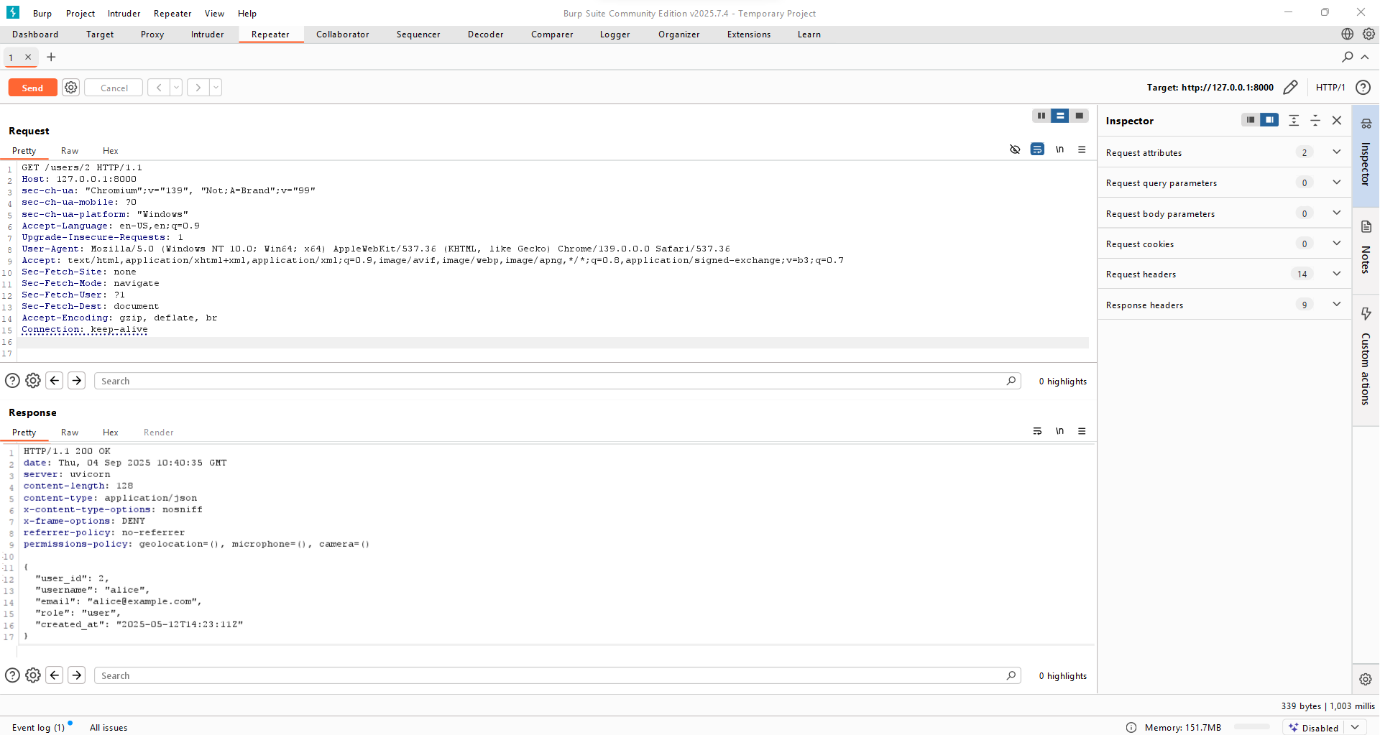


Figure B.18 - Burp Suite Repeater showing a BOLA exploit. By modifying the request to /users/2, the API returned private details of another account (alice@example.com), illustrating unauthorized data disclosure.

# Appendix C – CVSS v4.0 Scoring Evidence

This appendix documents the severity scoring applied to each vulnerability using the CVSS v4.0 framework. For transparency, each entry includes the metric table, vector string, base score, and a brief rationale aligned to the behaviour observed in Chapters 5–6.

**C.1 Broken Object Level Authorization (API1 – BOLA)**

* **Vector:** CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N
* **Score:** 7.1 (High)
* **Rationale:** Cross-user data retrieval threatens confidentiality but requires authenticated access.



Figure C.1 – CVSS v4.0 calculator output for API1 (BOLA).

***C.2 Broken User Authentication (API2)***

* ***Vector:*** *CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N*
* ***Score:*** *8.8 (High)*
* ***Rationale:*** *Weak authentication enables brute-force attacks and potential privilege escalation.*

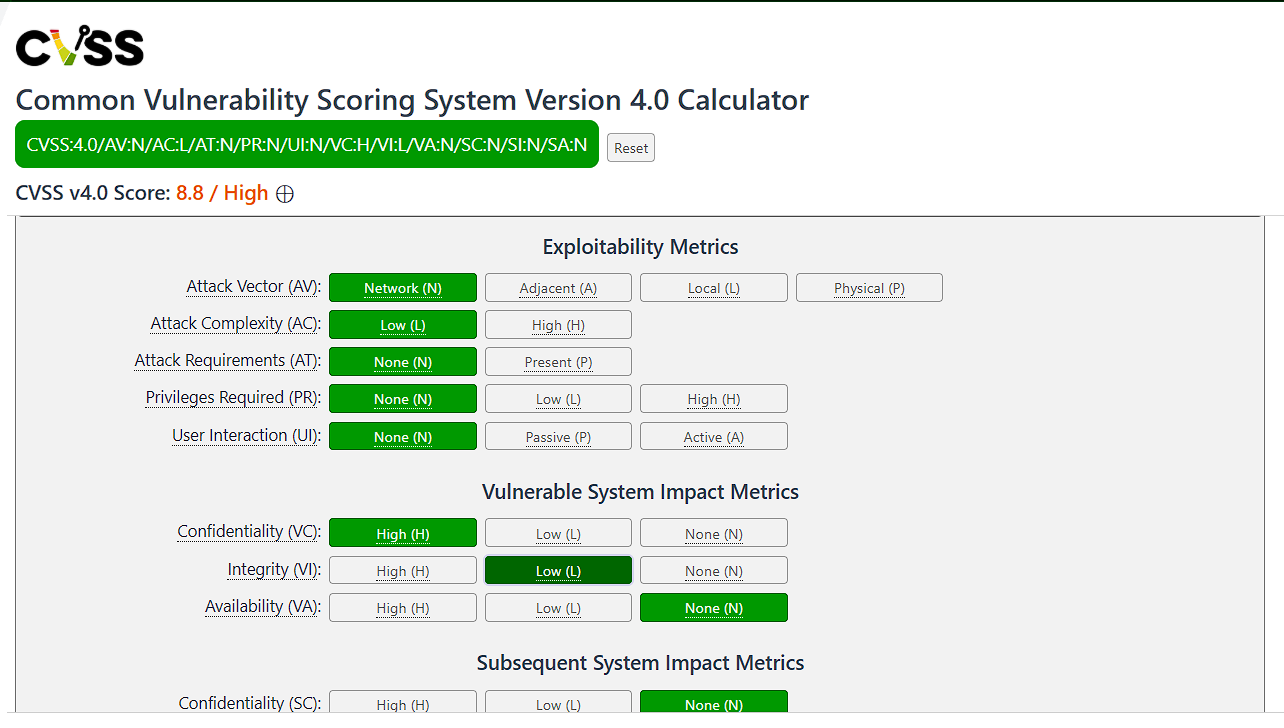


Figure C.2 – CVSS v4.0 calculator output for API2 (Broken Authentication).

**C.3 Excessive Data Exposure (API3)**

* **Vector**: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N
* **Score:** 8.7 (High)
* **Rationale**: Exposure of PII and tokens significantly compromises confidentiality.

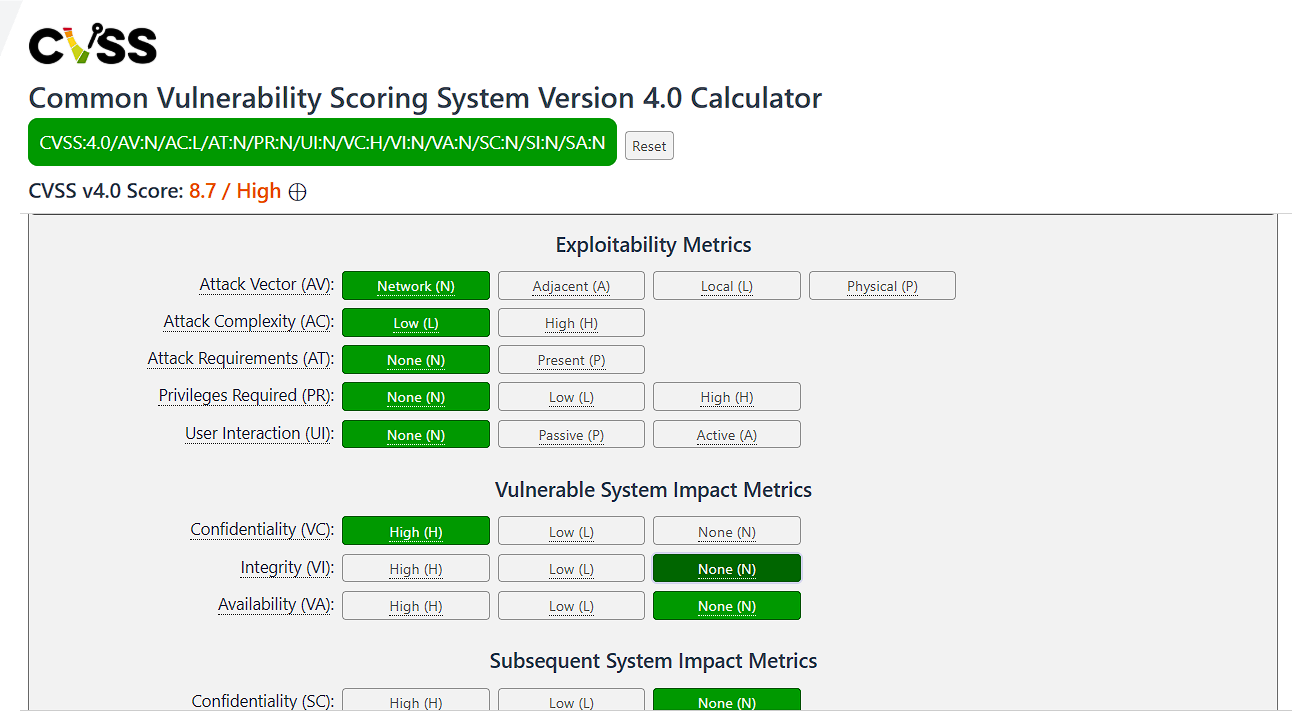


Figure C.3 – CVSS v4.0 calculator output for API3 (Excessive Data Exposure).

**C.4 Mass Assignment (API6)**

* **Vector:** CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N
* **Score:** 8.6 (High)
* **Rationale:** Arbitrary field injection allows privilege escalation and high-impact data modification.

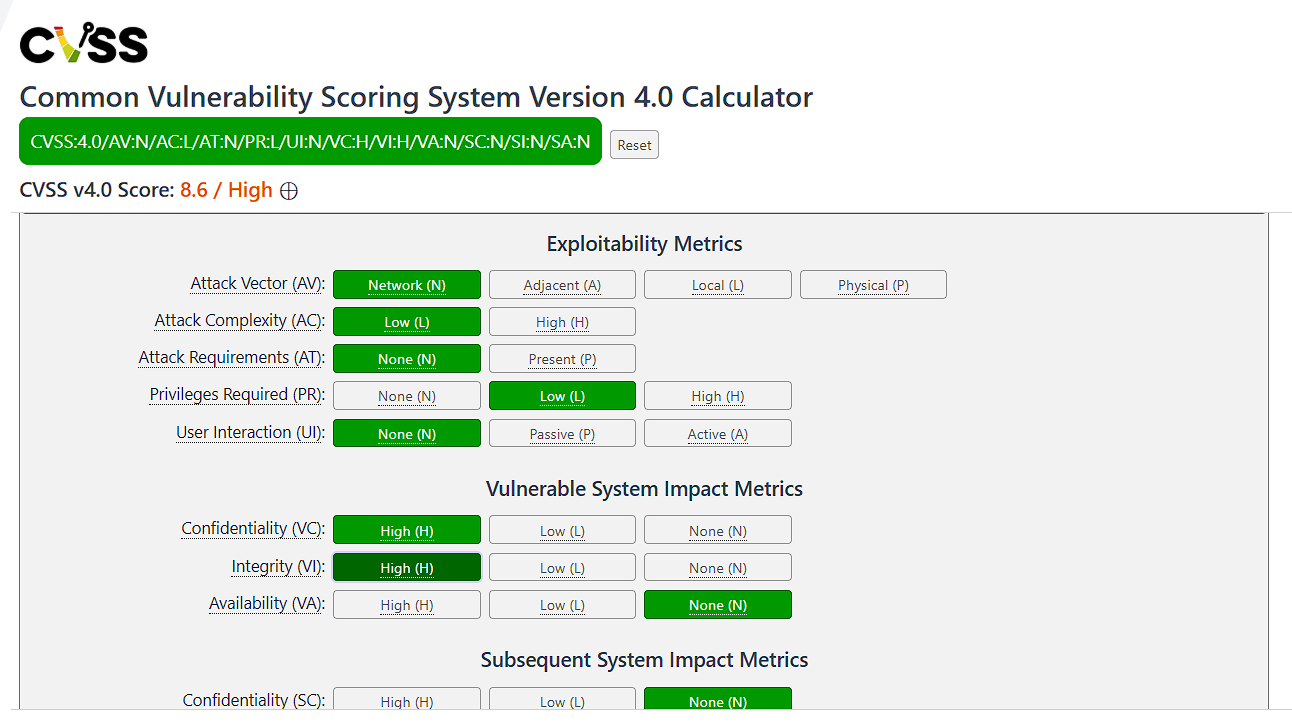


Figure C.4 – CVSS v4.0 calculator output for API6 (Mass Assignment).

**C.5 Improper Asset Management (API9)**

* **Vector:** CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N
* **Score:** 6.9 (Medium)
* **Rationale**: Deprecated endpoint increases attack surface but reveals only version information.

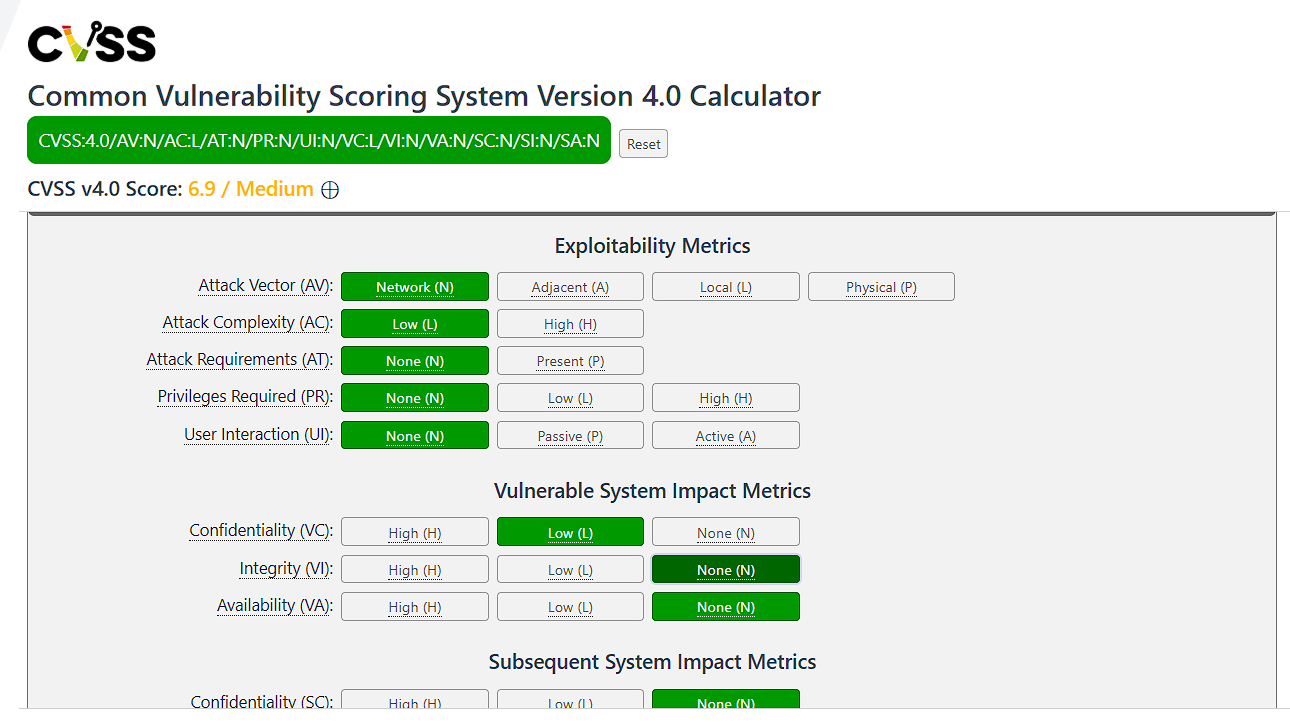


Figure C.5 – CVSS v4.0 calculator output for API9 (Improper Asset Management).

# Appendix D - ZAP API Scanning Report

**Generated with**[**[The ZAP logo](https://zaproxy.org/)ZAP**](https://zaproxy.org)**on Tue 2 Sept 2025, at 20:43:21**

**ZAP Version: 2.16.1**

**ZAP by**[**Checkmarx**](https://checkmarx.com/)

**Contents**

* [**About This Report**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#about-this-report)
  + [**Report Parameters**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#report-parameters)
* [**Summaries**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#summaries)
  + [**Alert Counts by Risk and Confidence**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#risk-confidence-counts)
  + [**Alert Counts by Site and Risk**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#site-risk-counts)
  + [**Alert Counts by Alert Type**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-counts)
* [**Alerts**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alerts)
  + [**Risk=Medium, Confidence=High (1)**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alerts--risk-2-confidence-3)
  + [**Risk=Medium, Confidence=Medium (1)**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alerts--risk-2-confidence-2)
  + [**Risk=Low, Confidence=High (1)**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alerts--risk-1-confidence-3)
  + [**Risk=Low, Confidence=Medium (1)**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alerts--risk-1-confidence-2)
  + [**Risk=Low, Confidence=Low (1)**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alerts--risk-1-confidence-1)
  + [**Risk=Informational, Confidence=Medium (6)**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alerts--risk-0-confidence-2)
  + [**Risk=Informational, Confidence=Low (1)**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alerts--risk-0-confidence-1)
* [**Appendix**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#appendix)
  + [**Alert Types**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-types)

**About This Report**

**Report Parameters**

**Contexts**

**No contexts were selected, so all contexts were included by default.**

**Sites**

**The following sites were included:**

* **https://secure-fastapi-g3d9ghbmc7b2akgx.canadacentral-01.azurewebsites.net/http://127.0.0.1:8000**

**(If no sites were selected, all sites were included by default.)**

**An included site must also be within one of the included contexts for its data to be included in the report.**

**Risk levels**

**Included: High, Medium, Low, Informational**

**Excluded: None**

**Confidence levels**

**Included: User Confirmed, High, Medium, Low**

**Excluded: User Confirmed, High, Medium, Low, False Positive**

**Summaries**

**Alert Counts by Risk and Confidence**

| **This table shows the number of alerts for each level of risk and confidence included in the report.**  **(The percentages in brackets represent the count as a percentage of the total number of alerts included in the report, rounded to one decimal place.)** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
|  | | **Confidence** | | | | |
| **User Confirmed** | **High** | **Medium** | **Low** | **Total** |
| **Risk** | **High** | **0 (0.0%)** | **0 (0.0%)** | **0 (0.0%)** | **0 (0.0%)** | **0 (0.0%)** |
| **Medium** | **0 (0.0%)** | **1 (8.3%)** | **1 (8.3%)** | **0 (0.0%)** | **2 (16.7%)** |
| **Low** | **0 (0.0%)** | **1 (8.3%)** | **1 (8.3%)** | **1 (8.3%)** | **3 (25.0%)** |
| **Informational** | **0 (0.0%)** | **0 (0.0%)** | **6 (50.0%)** | **1 (8.3%)** | **7 (58.3%)** |
| **Total** | **0 (0.0%)** | **2 (16.7%)** | **8 (66.7%)** | **2 (16.7%)** | **12 (100%)** |

**Alert Counts by Site and Risk**

| **This table shows, for each site for which one or more alerts were raised, the number of alerts raised at each risk level.**  **Alerts with a confidence level of "False Positive" have been excluded from these counts.**  **(The numbers in brackets are the number of alerts raised for the site at or above that risk level.)** | | | | | |
| --- | --- | --- | --- | --- | --- |
|  | | **Risk** | | | |
| **High (= High)** | **Medium (>= Medium)** | **Low (>= Low)** | **Informational (>= Informational)** |
| **Site** | **https://secure-fastapi-g3d9ghbmc7b2akgx.canadacentral-01.azurewebsites.net/** | **0 (0)** | **2 (2)** | **2 (4)** | **1 (5)** |
| **https://android.clients.google.com** | **0 (0)** | **0 (0)** | **0 (0)** | **1 (1)** |
| **https://www.googleapis.com** | **0 (0)** | **0 (0)** | **1 (1)** | **1 (2)** |
| **https://accounts.google.com** | **0 (0)** | **0 (0)** | **0 (0)** | **1 (1)** |
| **http://127.0.0.1:8000** | **0 (0)** | **0 (0)** | **0 (0)** | **3 (3)** |

**Alert Counts by Alert Type**

| **This table shows the number of alerts of each alert type, together with the alert type's risk level.**  **(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)** | | |
| --- | --- | --- |
| **Alert type** | **Risk** | **Count** |
| [**Content Security Policy (CSP) Header Not Set**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-0) | **Medium** | **6 (50.0%)** |
| [**Missing Anti-clickjacking Header**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-1) | **Medium** | **6 (50.0%)** |
| [**Strict-Transport-Security Header Not Set**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-2) | **Low** | **10 (83.3%)** |
| [**Timestamp Disclosure - Unix**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-3) | **Low** | **7 (58.3%)** |
| [**X-Content-Type-Options Header Missing**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-4) | **Low** | **6 (50.0%)** |
| [**Information Disclosure - Sensitive Information in URL**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-5) | **Informational** | **3 (25.0%)** |
| [**Re-examine Cache-control Directives**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-6) | **Informational** | **6 (50.0%)** |
| [**Tech Detected - HSTS**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-7) | **Informational** | **1 (8.3%)** |
| [**Tech Detected - HTTP/3**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-8) | **Informational** | **6 (50.0%)** |
| [**Tech Detected - OpenGSE**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-9) | **Informational** | **2 (16.7%)** |
| [**Tech Detected - Uvicorn**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-10) | **Informational** | **1 (8.3%)** |
| [**User Agent Fuzzer**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-11) | **Informational** | **24 (200.0%)** |
| **Total** |  | **12** |

**Alerts**

1. **Risk=Medium, Confidence=High (1)**
   1. **https://optimizationguide-pa.googleapis.com (1)**
      1. [**Content Security Policy (CSP) Header Not Set**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-0)**(1)**
         1. **GET https://optimizationguide-pa.googleapis.com/downloads?name=1673999601&target=OPTIMIZATION\_TARGET\_PAGE\_VISIBILITY**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. **Risk=Medium, Confidence=Medium (1)**
   1. **https://optimizationguide-pa.googleapis.com (1)**
      1. [**Missing Anti-clickjacking Header**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-1)**(1)**
         1. **GET https://optimizationguide-pa.googleapis.com/downloads?name=1673999601&target=OPTIMIZATION\_TARGET\_PAGE\_VISIBILITY**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. **Risk=Low, Confidence=High (1)**
   1. **https://www.googleapis.com (1)**
      1. [**Strict-Transport-Security Header Not Set**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-2)**(1)**
         1. **POST https://www.googleapis.com/chromewebstore/v1.1/items/verify**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. **Risk=Low, Confidence=Medium (1)**
   1. **https://optimizationguide-pa.googleapis.com (1)**
      1. [**X-Content-Type-Options Header Missing**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-4)**(1)**
         1. **GET https://optimizationguide-pa.googleapis.com/downloads?name=1745311339&target=OPTIMIZATION\_TARGET\_GEOLOCATION\_PERMISSION\_PREDICTIONS**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. **Risk=Low, Confidence=Low (1)**
   1. **https://optimizationguide-pa.googleapis.com (1)**
      1. [**Timestamp Disclosure - Unix**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-3)**(1)**
         1. **POST https://optimizationguide-pa.googleapis.com/v1:GetModels?key=AIzaSyA2KlwBX3mkFo30om9LUFYQhpqLoa\_BNhE**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. **Risk=Informational, Confidence=Medium (6)**
   1. **https://android.clients.google.com (1)**
      1. [**Tech Detected - OpenGSE**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-9)**(1)**
         1. **POST https://android.clients.google.com/c2dm/register3**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

* 1. **https://www.googleapis.com (1)**
     1. [**Tech Detected - HTTP/3**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-8)**(1)**
        1. **POST https://www.googleapis.com/chromewebstore/v1.1/items/verify**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

* 1. **https://accounts.google.com (1)**
     1. [**Tech Detected - HSTS**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-7)**(1)**
        1. **POST https://accounts.google.com/ListAccounts?gpsia=1&source=ChromiumBrowser&laf=b64bin&json=standard**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

* 1. **http://127.0.0.1:8000 (3)**
     1. [**Information Disclosure - Sensitive Information in URL**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-5)**(1)**
        1. **POST http://127.0.0.1:8000/submit-data?token=**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

* + 1. [**Tech Detected - Uvicorn**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-10)**(1)**
       1. **GET http://127.0.0.1:8000/openapi.json**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

* + 1. [**User Agent Fuzzer**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-11)**(1)**
       1. **GET http://127.0.0.1:8000/secure-data?token=**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. **Risk=Informational, Confidence=Low (1)**
   1. **https://optimizationguide-pa.googleapis.com (1)**
      1. [**Re-examine Cache-control Directives**](file:///C:\Users\91956\OneDrive\Desktop\2025-09-02-ZAP-Report-.html#alert-type-6)**(1)**
         1. **GET https://optimizationguide-pa.googleapis.com/downloads?name=1673999601&target=OPTIMIZATION\_TARGET\_PAGE\_VISIBILITY**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Appendix**

**Alert Types**

**This section contains additional information on the types of alerts in the report.**

1. **Content Security Policy (CSP) Header Not Set**

|  |  |
| --- | --- |
| **Source** | **raised by a passive scanner (**[**Content Security Policy (CSP) Header Not Set**](https://www.zaproxy.org/docs/alerts/10038/)**)** |
| **CWE ID** | [**693**](https://cwe.mitre.org/data/definitions/693.html) |
| **WASC ID** | **15** |
| **Reference** | * + [**https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing\_Content\_Security\_Policy**](https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy)   + [**https://cheatsheetseries.owasp.org/cheatsheets/Content\_Security\_Policy\_Cheat\_Sheet.html**](https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html)   + [**https://www.w3.org/TR/CSP/**](https://www.w3.org/TR/CSP/)   + [**https://w3c.github.io/webappsec-csp/**](https://w3c.github.io/webappsec-csp/)   + [**https://web.dev/articles/csp**](https://web.dev/articles/csp)   + [**https://caniuse.com/#feat=contentsecuritypolicy**](https://caniuse.com/#feat=contentsecuritypolicy)   + [**https://content-security-policy.com/**](https://content-security-policy.com/) |

1. **Missing Anti-clickjacking Header**

|  |  |
| --- | --- |
| **Source** | **raised by a passive scanner (**[**Anti-clickjacking Header**](https://www.zaproxy.org/docs/alerts/10020/)**)** |
| **CWE ID** | [**1021**](https://cwe.mitre.org/data/definitions/1021.html) |
| **WASC ID** | **15** |
| **Reference** | * + [**https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options) |

1. **Strict-Transport-Security Header Not Set**

|  |  |
| --- | --- |
| **Source** | **raised by a passive scanner (**[**Strict-Transport-Security Header**](https://www.zaproxy.org/docs/alerts/10035/)**)** |
| **CWE ID** | [**319**](https://cwe.mitre.org/data/definitions/319.html) |
| **WASC ID** | **15** |
| **Reference** | * + [**https://cheatsheetseries.owasp.org/cheatsheets/HTTP\_Strict\_Transport\_Security\_Cheat\_Sheet.html**](https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html)   + [**https://owasp.org/www-community/Security\_Headers**](https://owasp.org/www-community/Security_Headers)   + [**https://en.wikipedia.org/wiki/HTTP\_Strict\_Transport\_Security**](https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security)   + [**https://caniuse.com/stricttransportsecurity**](https://caniuse.com/stricttransportsecurity)   + [**https://datatracker.ietf.org/doc/html/rfc6797**](https://datatracker.ietf.org/doc/html/rfc6797) |

1. **Timestamp Disclosure - Unix**

|  |  |
| --- | --- |
| **Source** | **raised by a passive scanner (**[**Timestamp Disclosure**](https://www.zaproxy.org/docs/alerts/10096/)**)** |
| **CWE ID** | [**497**](https://cwe.mitre.org/data/definitions/497.html) |
| **WASC ID** | **13** |
| **Reference** | * + [**https://cwe.mitre.org/data/definitions/200.html**](https://cwe.mitre.org/data/definitions/200.html) |

1. **X-Content-Type-Options Header Missing**

|  |  |
| --- | --- |
| **Source** | **raised by a passive scanner (**[**X-Content-Type-Options Header Missing**](https://www.zaproxy.org/docs/alerts/10021/)**)** |
| **CWE ID** | [**693**](https://cwe.mitre.org/data/definitions/693.html) |
| **WASC ID** | **15** |
| **Reference** | * + [**https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941(v=vs.85)**](https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941(v=vs.85))   + [**https://owasp.org/www-community/Security\_Headers**](https://owasp.org/www-community/Security_Headers) |

1. **Information Disclosure - Sensitive Information in URL**

|  |  |
| --- | --- |
| **Source** | **raised by a passive scanner (**[**Information Disclosure - Sensitive Information in URL**](https://www.zaproxy.org/docs/alerts/10024/)**)** |
| **CWE ID** | [**598**](https://cwe.mitre.org/data/definitions/598.html) |
| **WASC ID** | **13** |

1. **Re-examine Cache-control Directives**

|  |  |
| --- | --- |
| **Source** | **raised by a passive scanner (**[**Re-examine Cache-control Directives**](https://www.zaproxy.org/docs/alerts/10015/)**)** |
| **CWE ID** | [**525**](https://cwe.mitre.org/data/definitions/525.html) |
| **WASC ID** | **13** |
| **Reference** | * + [**https://cheatsheetseries.owasp.org/cheatsheets/Session\_Management\_Cheat\_Sheet.html#web-content-caching**](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#web-content-caching)   + [**https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control)   + [**https://grayduck.mn/2021/09/13/cache-control-recommendations/**](https://grayduck.mn/2021/09/13/cache-control-recommendations/) |

1. **Tech Detected - HSTS**

|  |  |
| --- | --- |
| **Source** | **raised by other tools/functionalities in ZAP (for example, fuzzer, HTTPS Info add-on, custom scripts...) (plugin ID: 10004)** |
| **WASC ID** | **13** |
| **Reference** | * + [**https://www.rfc-editor.org/rfc/rfc6797#section-6.1**](https://www.rfc-editor.org/rfc/rfc6797#section-6.1) |

1. **Tech Detected - HTTP/3**

|  |  |
| --- | --- |
| **Source** | **raised by other tools/functionalities in ZAP (for example, fuzzer, HTTPS Info add-on, custom scripts...) (plugin ID: 10004)** |
| **WASC ID** | **13** |
| **Reference** | * + [**https://httpwg.org/**](https://httpwg.org/) |

1. **Tech Detected - OpenGSE**

|  |  |
| --- | --- |
| **Source** | **raised by other tools/functionalities in ZAP (for example, fuzzer, HTTPS Info add-on, custom scripts...) (plugin ID: 10004)** |
| **WASC ID** | **13** |
| **Reference** | * + [**https://code.google.com/p/opengse**](https://code.google.com/p/opengse) |

1. **Tech Detected - Uvicorn**

|  |  |
| --- | --- |
| **Source** | **raised by other tools/functionalities in ZAP (for example, fuzzer, HTTPS Info add-on, custom scripts...) (plugin ID: 10004)** |
| **WASC ID** | **13** |
| **Reference** | * + [**https://www.uvicorn.org**](https://www.uvicorn.org) |

1. **User Agent Fuzzer**

|  |  |
| --- | --- |
| **Source** | **raised by an active scanner (plugin ID:**[**10104**](https://www.zaproxy.org/docs/alerts/10104/)**)** |
| **Reference** | * + [**https://owasp.org/wstg**](https://owasp.org/wstg) |

# Appendix E - Gantt Chart

