API Penetration Testing Report — VAmPI (Vulnerable API)

Prepared by: Adeola Odunlade
Date: November 21, 2025
Target Application: VAmPI (Mock API via Postman)
Test Environment: Windows 11 + Postman Mock Server + Burp Suite Community
Testing Type: API Design Security Assessment & Mock Interaction Testing

Executive Summary

This report documents a security assessment conducted on the VAmPI API, a purposely vulnerable API provided in the form of an OpenAPI/Swagger specification.

What I Learned

As part of the Week 1 Cybersafe API Security training, I completed the API Security Fundamentals course from APISec University, which covered:

- The importance of API security
- The OWASP API Security Top 10
- API threat modeling
- The three pillars of API security: Governance, Monitoring, and Testing
- The modern API application security technology landscape

This assessment allowed me to apply those concepts in a hands-on environment. By analyzing the Swagger file, creating a Postman mock server, and testing traffic through Burp Suite, I was able to practically apply theoretical skills such as identifying design-layer vulnerabilities, evaluating access control gaps, and understanding how API weaknesses can be exploited. The project reinforced how API security principles are applied in real API testing workflows.

Scenario: You have been brought in to review a company's API prior to launch. Your responsibility is to analyze the provided API specification and identify

potential security issues before the system is exposed to real users or connected to production databases.

The primary objectives of this assessment were to:

- Review the Swagger/OpenAPI file to understand the API's structure, expected behavior, and security model
- Identify three (3) security vulnerabilities focusing on:
  - Broken Object Level Authorization (BOLA)
  - Excessive Data Exposure
  - Lack of access controls on sensitive endpoints
- Explain the risks using the OWASP API Security Top 10
- Recommend realistic fixes or mitigations to improve the API's security posture before deployment

Testing was performed by importing the Swagger file into Postman, generating a mock server, and routing all traffic through Burp Suite to observe and validate the API's behavior. Since the mock server reflects the specification's documented responses, all findings are based strictly on design-level weaknesses present in the Swagger file.

These design flaws particularly around authentication, authorization, and data exposure would pose major security risks if implemented as is. Addressing these issues during the design phase is crucial to ensuring a secure implementation ahead of the API's launch.

Scope

- Swagger/OpenAPI YAML file for VAmPI
- Postman mock server generated from the Swagger file
- API endpoints defined in the specification
- Local traffic inspection using Burp Suite

Boundaries

- Only design-level issues were evaluated
- Mock server responses were used strictly for verification
- No destructive actions were performed

Methodology

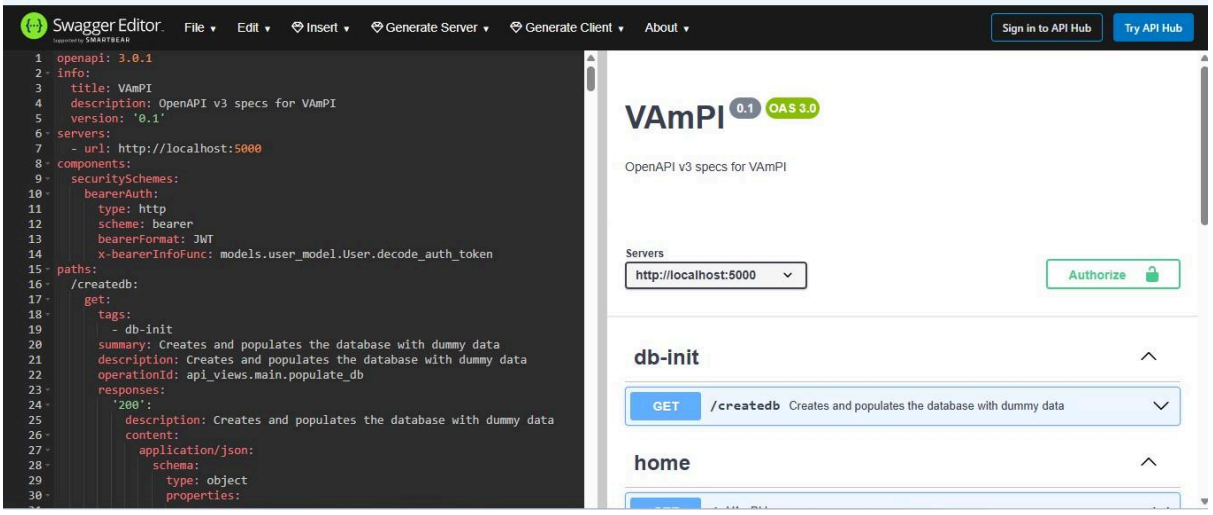
The testing approach simulated an API review using safe tools.
The following tools were used:

Tools Used

- **Postman**: Import Swagger, create mock server, issue requests
- **Burp Suite Community**: Proxy traffic, capture HTTP history
- **Swagger/OpenAPI Specification**: Primary evidence source

Process Overview

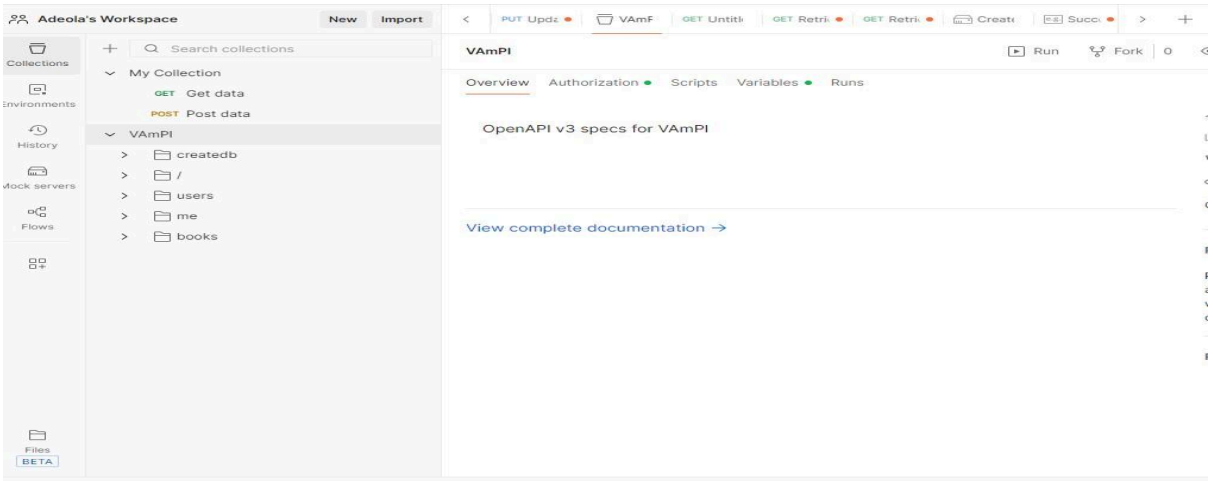
Download & Review Swagger File: The provided YAML file was reviewed line-by-line, focusing on:
Authentication requirements
Sensitive data fields
Path parameters
Security definitions
Response schemas



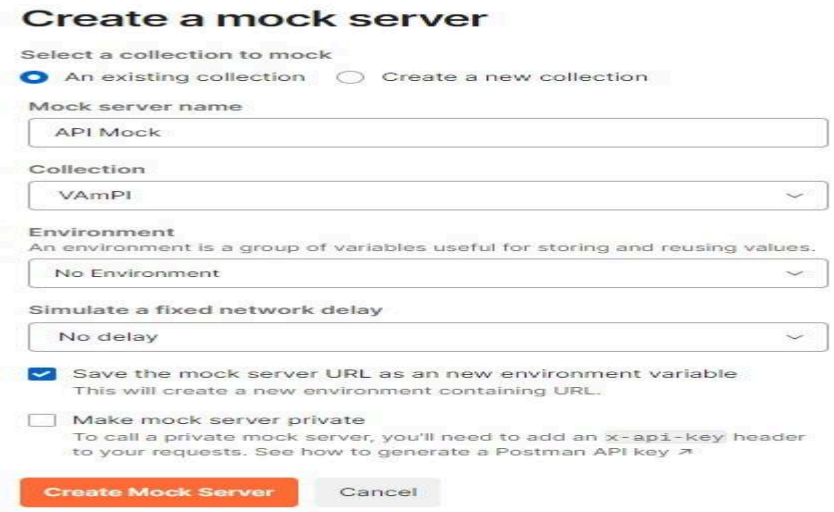
Swagger YAML preview

Import into Postman & Create Mock Server

- Swagger YAML imported into Postman
- Postman generated all endpoints automatically
- A mock server was created to emulate API behavior
- An environment variable {{base_url}} was automatically generated



Successful Postman Import Confirmation



Mock Server Configuration Page

Configure Burp Suite

- Burp installed on Windows
- Proxy listener enabled on 127.0.0.1:8080
- Intercept turned OFF

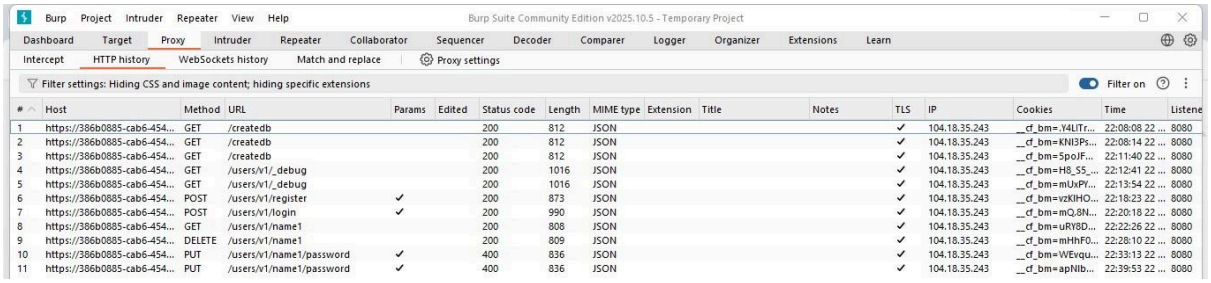
- Postman configured to route requests through Burp proxy

Burp Proxy Listener Running (127.0.0.1:8080)

Send Test Requests to Confirm Behavior

Requests such as:

GET /
GET /users/v1
GET /users/v1/_debug
GET /createdb
GET /users/v1/{username}
were sent through Postman, and responses validated through Burp's HTTP History.



Burp HTTP History Showing Captured Requests

Key Findings

Below are the three required findings, each aligned with the OWASP API Security Top 10 and backed by design-level evidence.

Finding 1: Excessive Data Exposure

Severity: High
OWASP Mapping: API3:2023 – Excessive Data Exposure
Endpoint: GET /users/v1/_debug

Description

The /users/v1/_debug endpoint returns complete user records, including:

Plaintext password
Admin flag
Email address
Username
This is visible directly in the Swagger YAML definitions.

Business Impact

If implemented:

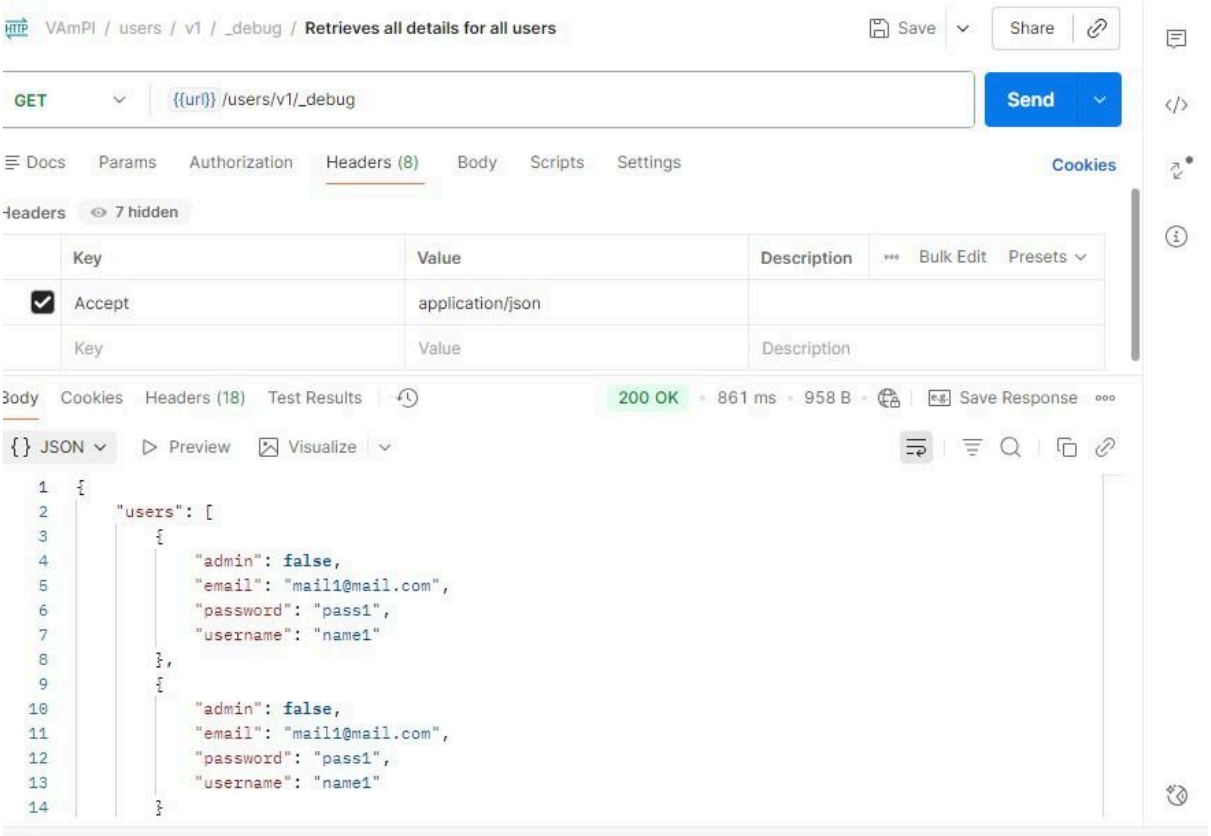
- Attackers could harvest all user passwords instantly
- Admin accounts could be compromised
- In real systems, this leads to identity theft, privilege escalation, full system compromise

Evidence

A. Swagger Evidence

YAML Showing 'password' and 'admin' Fields

B. Postman Mock Server Response



Mock Response Revealing Passwords

C. Burp Suite Capture

Burp Log Showing Exposed Sensitive Data

Mitigation

- Remove debug endpoints from production
- NEVER store or return plaintext passwords
- Hash passwords using bcrypt or Argon2
- Implement response filtering (return only required fields)

Finding 2: Broken Object Level Authorization (BOLA)

Severity: High
OWASP Mapping: API1:2023 – Broken Object Level Authorization
Endpoints:

GET /users/v1/{username}
PUT /users/v1/{username}/email
DELETE /users/v1/{username}

Description

These endpoints allow direct access to user-specific data based solely on the username path parameter.
The Swagger file does not define authentication or authorization controls.

There is no:

security:
- bearerAuth: []

Therefore any user (or attacker) could potentially request:

GET /users/v1/alice
GET /users/v1/bob

and retrieve user information.

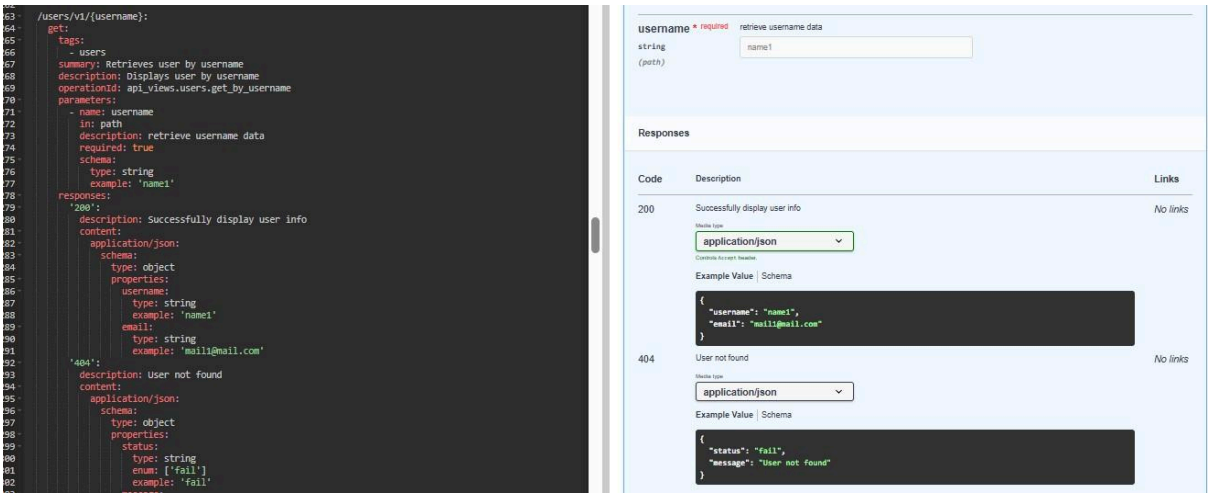
Business Impact

This flaw leads to:

- Unauthorized access to other users' data
- Account manipulation
- User data scraping
- Full profile takeover in real implementations

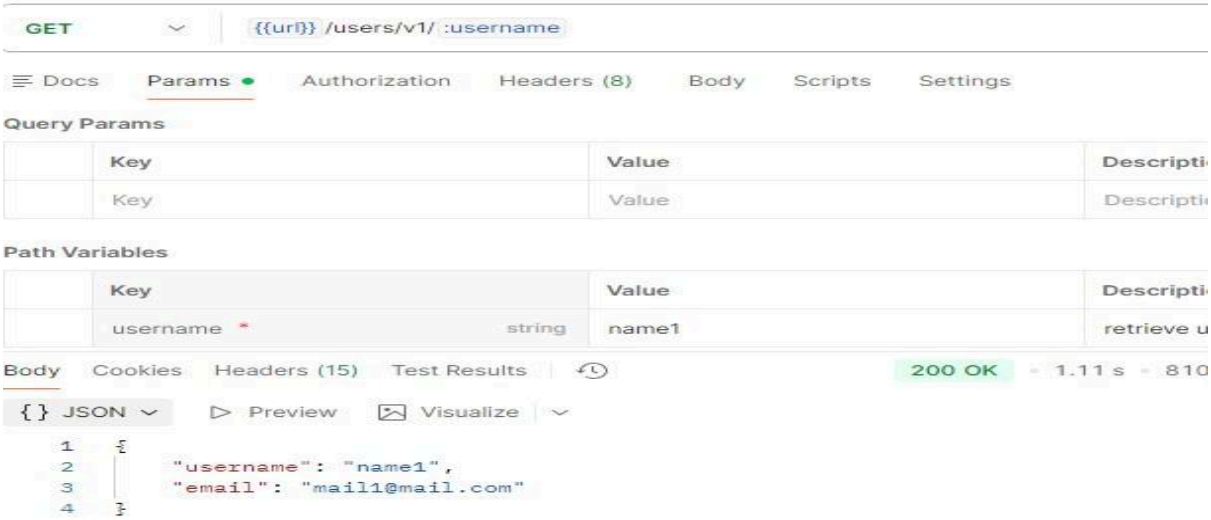
Evidence

A. Swagger Evidence



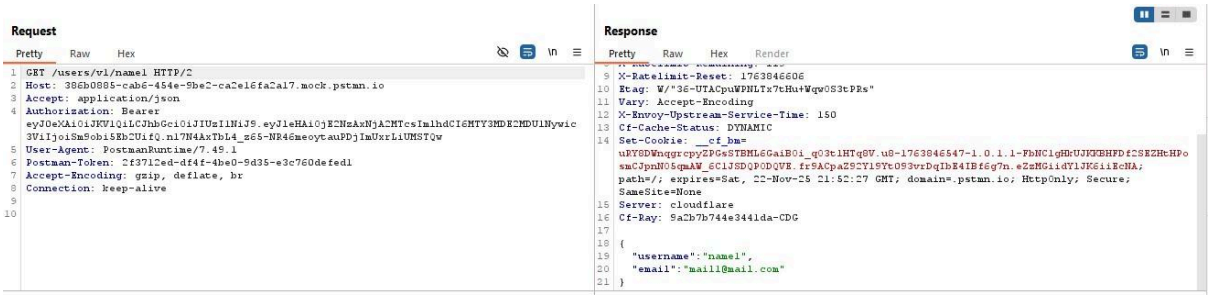
YAML Showing No Security Block on User Endpoints

B. Postman Mock Response



GET /users/v1/name1 Returning Data Without Authentication

C. Burp Suite Log



Burp Log Showing Request Without Authorization Header

Mitigation

- Require JWT authentication on all user-specific routes
- Enforce user ownership checks (“user can only access their own resources”)
- Restrict DELETE and modification operations to admins only

Finding 3: Lack of Access Control on Sensitive Endpoints

Severity: High

OWASP Mapping: API2:2023 – Broken Authentication / Missing Access Control

Endpoints:

/createdb
/users/v1
/users/v1/_debug

Description

The Swagger file defines no authentication requirements for endpoints that perform sensitive operations, including:

Database initialization
Debug data retrieval

User listing

The absence of access control results in sensitive functionality being available to the public.

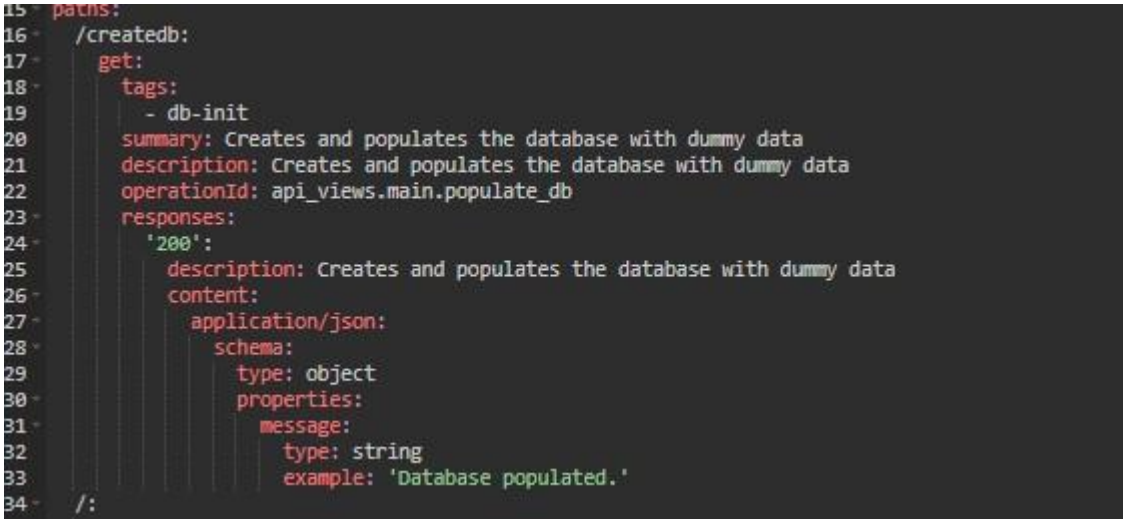
Business Impact

If implemented:

- Anyone could wipe or repopulate the database
- Anyone could retrieve full user lists
- Attackers could use these endpoints to enumerate users

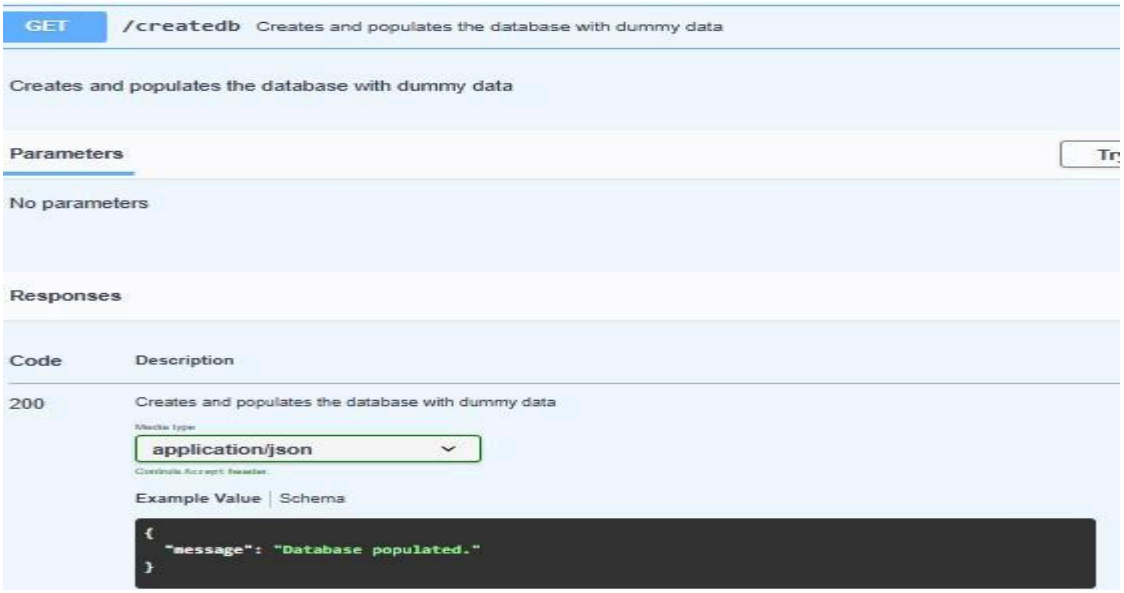
Evidence

A. Swagger Evidence



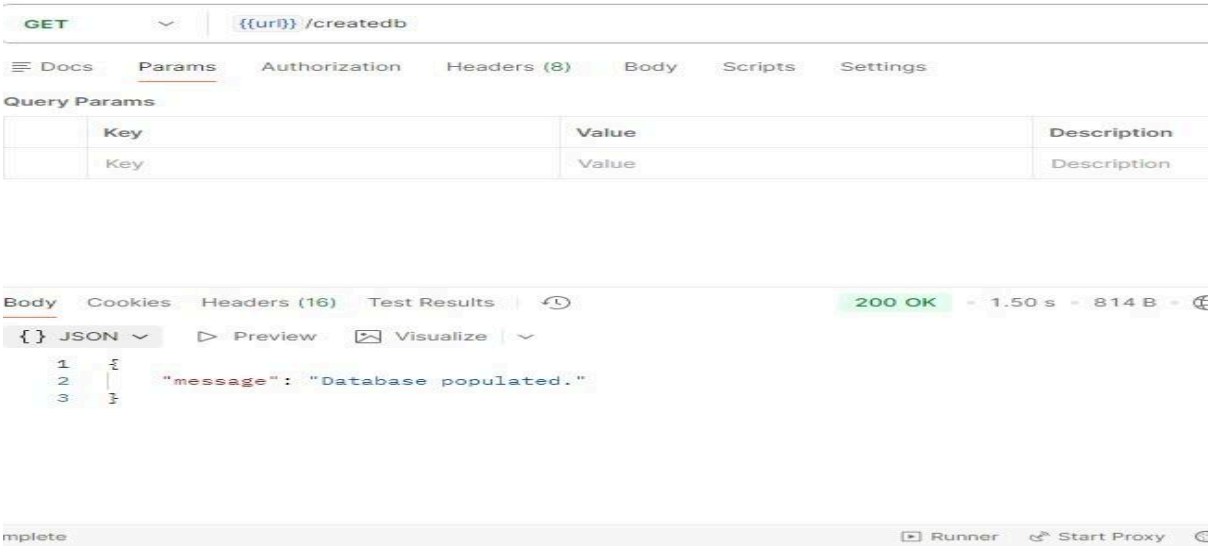
```yaml
paths:
  /createdb:
    get:
      tags:
        - db-init
      summary: Creates and populates the database with dummy data
      description: Creates and populates the database with dummy data
      operationId: api_views.main.populate_db
      responses:
        '200':
          description: Creates and populates the database with dummy data
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: 'Database populated.'
  /:
```

YAML Showing /createdb Without Security Defined



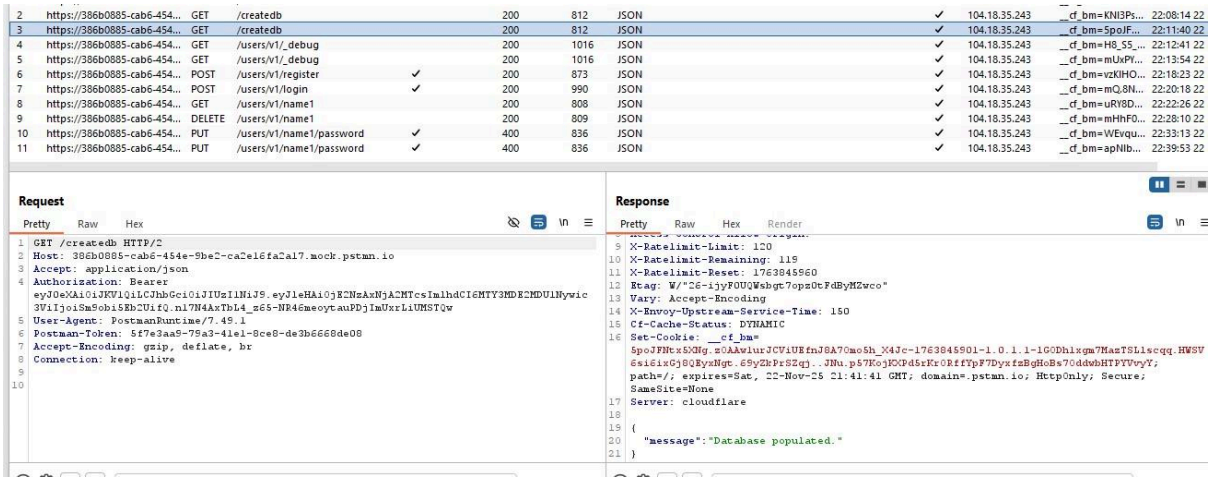
YAML Showing /createdb Without Security Defined

B. Postman Request



200 OK Without Authorization

C. Burp Suite Log



Unauthenticated Access to Sensitive Route

Mitigation

- Protect all sensitive endpoints with JWT-based authentication
- Restrict database-reset endpoints to development environments
- Implement RBAC (Role-Based Access Control)
- Explicitly define required permissions in Swagger/OpenAPI

Recommendations

1) Excessive Data Exposure: To address the excessive exposure of sensitive user information—especially plaintext passwords—the API should remove all debug endpoints such as /users/v1/_debug from production environments. Sensitive fields must never be returned to clients, and response filtering should be implemented to ensure only the minimal required data is exposed. Passwords must be securely hashed using strong algorithms such as Argon2 or bcrypt, and never stored or transmitted in plaintext.

2) Broken Object Level Authorization (BOLA): User-specific endpoints such as /users/v1/{username}, /users/v1/{username}/email, and deletion routes must enforce strict access control. The API should require JWT authentication on all user-related operations, and implement ownership checks to ensure a user can only access or modify their own account. Destructive actions like deleting a

user must be restricted to administrators, with authorization validated on the server side rather than relying on client input.

3) Lack of Access Controls on Sensitive Endpoints: Endpoints like /createdb, /users/v1, and debug routes must not be accessible without authentication. These should be protected by defining explicit security: blocks within the Swagger specification, enforcing that a valid token is required. Role-Based Access Control (RBAC) should be implemented to limit administrative operations, and highly sensitive endpoints such as /createdb should be available only in development environments and never exposed publicly.

Blockers and Challenges

At the beginning of the project, I encountered issues importing the Swagger file into Postman on Kali Linux. Postman required me to create an account before I could proceed, and even after signing in, the application continued to lag heavily and freeze during basic actions like importing the file and navigating the collection. This made the workflow slow and unreliable. To resolve the problem, I switched to Windows, where Postman ran smoothly and I was able to continue the assessment without performance issues.

Conclusion

The VAmPI API Swagger specification contains deliberately vulnerable design elements for educational purposes, but the issues identified closely resemble real-world API security failures. These include missing or weak access controls, inadequate object-level authorization, and the unnecessary exposure of sensitive data. Such vulnerabilities can lead to serious consequences in a production environment, including unauthorized account access, data breaches, and full system compromise.

Addressing these weaknesses at the API design stage is essential. Implementing strong authentication and authorization, enforcing least-privilege access, and ensuring proper data handling practices will significantly enhance the API's security posture before development or deployment begins.

APISEC UNIVERSITY

CERTIFICATE
OF COMPLETION

This certificate is issued to:

Adeola Odunlade

for completing the course:

API Security Fundamentals '25 (2 hours)

Issued on: Nov 20, 2025

https://www.credly.com/badges/1eb16f5a-d63c-4bb3-b707-c56267086107

Certificate of Completion