

# 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

The screenshot shows the Swagger Editor interface split into two main sections. On the left, the 'Swagger Editor' tab is active, displaying the raw Swagger YAML code. The code defines an API titled 'VAmPI' with version '0.1', a single endpoint '/createdb' that creates and populates a database, and security requirements involving JWT tokens and a bearer auth scheme. On the right, the 'VAmPI' tab is active, showing the generated OpenAPI 3.0 documentation. It includes a summary of the API, a 'Servers' section with the URL 'http://localhost:5000', and two endpoints: 'db-init' (a GET request to '/createdb') and 'home'. The 'db-init' endpoint is described as creating and populating the database with dummy data. The 'home' endpoint is also listed.

```
openapi: 3.0.1
info:
  title: VAmPI
  description: OpenAPI v3 specs for VAmPI
  version: '0.1'
servers:
  - url: http://localhost:5000
components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
      x-bearerInfoFunc: models.user_model.User.decode_auth_token
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:
```

VAmPI 0.1 OAS 3.0

OpenAPI v3 specs for VAmPI

Servers

http://localhost:5000 Authorize

db-init

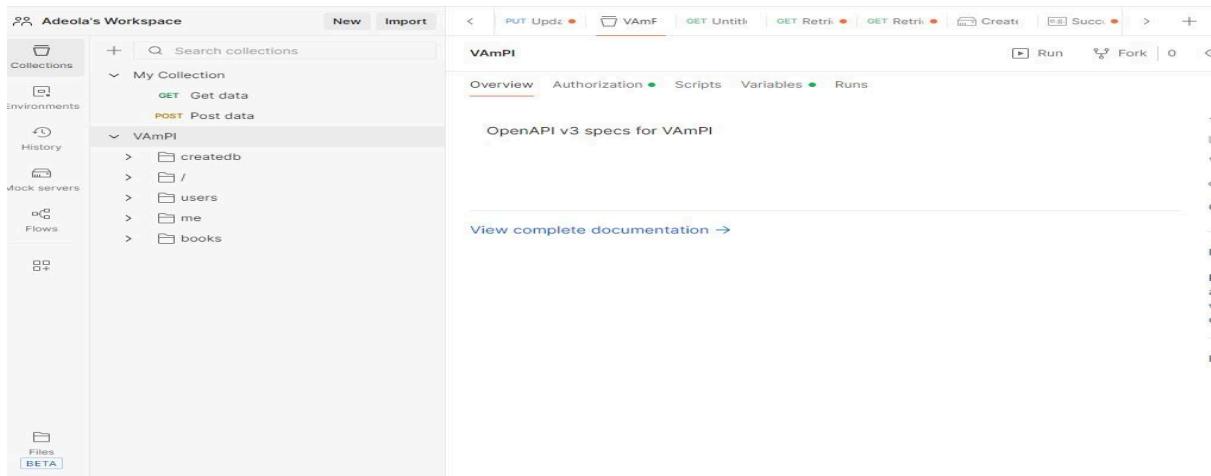
GET /createdb Creates and populates the database with dummy data

home

*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*

### Create a mock server

Select a collection to mock

An existing collection  Create a new collection

Mock server name: API Mock

Collection: VAmPI

Environment  
An environment is a group of variables useful for storing and reusing values.  
No Environment

Simulate a fixed network delay  
No delay

Save the mock server URL as an new environment variable  
This will create a new environment containing URL.

Make mock server private  
To call a private mock server, you'll need to add an `x-api-key` header to your requests. See how to generate a Postman API key ↗

**Create Mock Server** **Cancel**

*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

Proxy listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

Add	Running	Interface	Invisible	Redirect	Certificate	TLS Protocols	Support HTTPS
<input type="button" value="Edit"/>	<input checked="" type="checkbox"/> 127.0.0.1:8080				Per-host	Default	<input checked="" type="checkbox"/>
<input type="button" value="Remove"/>							

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate for other tools or another installation of Burp.

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.

#	Host	Method	URI	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listene
1	https://396b0885-cab6-454...	GET	/createdb			200	812	JSON			✓	104.18.35.243	_d_bm=Y4UTr...	22:08:08 22 ... 8080		
2	https://396b0885-cab6-454...	GET	/createdb			200	812	JSON			✓	104.18.35.243	_d_bm=KN9Pb...	22:08:14 22 ... 8080		
3	https://396b0885-cab6-454...	GET	/createdb			200	812	JSON			✓	104.18.35.243	_d_bm=Sp0fJ...	22:11:40 22 ... 8080		
4	https://396b0885-cab6-454...	GET	/users/v1/_debug			200	1016	JSON			✓	104.18.35.243	_d_bm=H8_S5...	22:12:41 22 ... 8080		
5	https://396b0885-cab6-454...	GET	/users/v1/_debug			200	1016	JSON			✓	104.18.35.243	_d_bm=mUpV...	22:13:54 22 ... 8080		
6	https://396b0885-cab6-454...	POST	/users/v1/register		✓	200	873	JSON			✓	104.18.35.243	_d_bm=vxkIHO...	22:18:23 22 ... 8080		
7	https://396b0885-cab6-454...	POST	/users/v1/login		✓	200	990	JSON			✓	104.18.35.243	_d_bm=mQ_8N...	22:20:18 22 ... 8080		
8	https://396b0885-cab6-454...	GET	/users/v1/name1			200	808	JSON			✓	104.18.35.243	_d_bm=uRV8D...	22:22:26 22 ... 8080		
9	https://396b0885-cab6-454...	DELETE	/users/v1/name1			200	809	JSON			✓	104.18.35.243	_d_bm=mHfH0...	22:28:10 22 ... 8080		
10	https://396b0885-cab6-454...	PUT	/users/v1/name1/password		✓	400	836	JSON			✓	104.18.35.243	_d_bm=WEVequ...	22:33:13 22 ... 8080		
11	https://396b0885-cab6-454...	PUT	/users/v1/name1/password		✓	400	836	JSON			✓	104.18.35.243	_d_bm=apNlb...	22:39:53 22 ... 8080		

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.

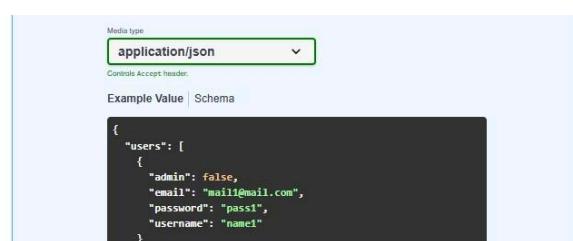
#### Business Impact

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



The screenshot shows a Swagger UI interface. On the left, there is a code editor window displaying a portion of a YAML file. The code is as follows:

```
105
106     items:
107       type: object
108       properties:
109         admin:
110           type: boolean
111           example: false
112         email:
113           type: string
114           example: 'malli@mail.com'
115         password:
116           type: string
117           example: 'pass1'
118         username:
119           type: string
119           example: 'name1'
```

On the right, there is a preview window showing a JSON example. The JSON is:

```
{
  "users": [
    {
      "admin": false,
      "email": "malli@mail.com",
      "password": "pass1",
      "username": "name1"
    }
  ]
}
```

YAML Showing 'password' and 'admin' Fields

## B. Postman Mock Server Response

The screenshot shows the Postman interface with the following details:

- HTTP:** VAmPI / users / v1 / \_debug / Retrieves all details for all users
- Method:** GET
- URL:** {{url}} /users/v1/\_debug
- Headers:** (8)
  - Accept: application/json
- Body:** JSON response showing two user objects with sensitive data:

```
1 {  
2     "users": [  
3         {  
4             "admin": false,  
5             "email": "mail1@mail.com",  
6             "password": "pass1",  
7             "username": "name1"  
8         },  
9         {  
10            "admin": false,  
11            "email": "mail1@mail.com",  
12            "password": "pass1",  
13            "username": "name1"  
14        }  
15    ]  
16}
```
- Response Status:** 200 OK
- Time:** 861 ms
- Size:** 958 B
- Save Response:** Available

*Mock Response Revealing Passwords*

## C. Burp Suite Capture

The Burp Suite capture shows the following:

- Request:** GET /users/v1/\_debug
- Response:** JSON response showing two user objects with sensitive data:

```
17 {  
18     "users": [  
19         {  
20             "admin": false,  
21             "email": "mail1@mail.com",  
22             "password": "pass1",  
23             "username": "name1"  
24         },  
25         {  
26             "admin": false,  
27             "email": "mail1@mail.com",  
28             "password": "pass1",  
29             "username": "name1"  
30         }  
31     ]  
32 }
```

*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

```
63: /users/v1/{username}:  
64:   get:  
65:     tags:  
66:       - users  
67:     summary: Retrieves user by username  
68:     description: Displays user by username  
69:     operationId: api_views.users.get_by_username  
70:     parameters:  
71:       - name: username  
72:         type: string  
73:         description: retrieve username data  
74:         required: true  
75:         schema:  
76:           type: string  
77:           example: 'name1'  
78:     responses:  
79:       '200':  
80:         description: Successfully display user info  
81:         content:  
82:           application/json:  
83:             schema:  
84:               type: object  
85:               properties:  
86:                 username:  
87:                   type: string  
88:                   example: 'name1'  
89:                 email:  
90:                   type: string  
91:                   example: 'mail1@mail.com'  
92:       '404':  
93:         description: User not found  
94:         content:  
95:           application/json:  
96:             schema:  
97:               type: object  
98:               properties:  
99:                 status:  
100:                type: string  
101:                enum: ['fail']  
102:                example: 'fail'  
103:               
```

Code	Description	Links
200	Successfully display user info	No links
404	User not found	No links

*YAML Showing No Security Block on User Endpoints*

### B. Postman Mock Response

GET {{url}} /users/v1/{:username}

Params

Key	Value	Description
username	name1	retrieve user info

Path Variables

Key	Value	Description
username	name1	retrieve user info

Body

200 OK • 1.11 s • 810

{ } JSON ▾ Preview Visualize

```
1 {  
2   "username": "name1",  
3   "email": "mail1@mail.com"  
4 }
```

*GET /users/v1/name1 Returning Data Without Authentication*

## C. Burp Suite Log



The screenshot shows a Burp Suite interface with two panes. The left pane, titled 'Request', displays a GET request to '/users/v1/\_debug'. The right pane, titled 'Response', shows a JSON response with various headers and a cookie section.

```
Request
Pretty Raw Hex Render
1 GET /users/v1/_debug HTTP/2
2 Host: 0BBD0B8C-ccb6-454e-9be2-ca2e16fa2a17.mock.postman.io
3 Accept: application/json
4 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NmAxNjACMTcsImhdCi6MTY3MDExMDU1Nyiwic3ViIjoiSm9obisRkCUIifQ.r1tNAxThI4_z65-NB46meoytauPdJImUxrLiUMSTQw
5 User-Agent: PostmanRuntime/7.49.1
6 Postman-Token: Cf371Ded-df4f-4be0-9d35-e3c760defd1
7 Accept-Encoding: gzip, deflate, br
8 Connection: keep-alive
9
10
11
12
13
14 Set-Cookie: _juf=eyJhbGciOiJIUzI1NiJ9.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NmAxNjACMTcsImhdCi6MTY3MDExMDU1Nyiwic3ViIjoiSm9obisRkCUIifQ.r1tNAxThI4_z65-NB46meoytauPdJImUxrLiUMSTQw; expires=Sat, 22-Nov-25 01:52:27 GMT; domain=.postman.io; HttpOnly; Secure; SameSite=None
15 Server: cloudflare
16 Cf-Ray: 9acb7b744e3441da-CDG
17
18 {
19   "username": "name1",
20   "email": "mail1@mail.com"
21 }
```

*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

```
15 paths:
16   /createdb:
17     get:
18       tags:
19         - db-init
20       summary: Creates and populates the database with dummy data
21       description: Creates and populates the database with dummy data
22       operationId: api_views.main.populate_db
23       responses:
24         '200':
25           description: Creates and populates the database with dummy data
26           content:
27             application/json:
28               schema:
29                 type: object
30                 properties:
31                   message:
32                     type: string
33                     example: 'Database populated.'
34   /:
```

*YAML Showing /createdb Without Security Defined*

**GET** /createdb Creates and populates the database with dummy data

Creates and populates the database with dummy data

**Parameters**

No parameters

**Responses**

Code	Description
200	Creates and populates the database with dummy data Media Type: application/json Content-Type: application/json Example Value   Schema <pre>{ "message": "Database populated." }</pre>

YAML Showing /createdb Without Security Defined

## B. Postman Request

**GET** {{url}}/createdb

Docs Params Authorization Headers (8) Body Scripts Settings

**Query Params**

	Key	Value	Description
	Key	Value	Description

**Body** Cookies Headers (16) Test Results ⏪ 200 OK - 1.50 s - 814 B - ⏪  
{} JSON ▾ Preview Visualize ▾  
1 {  
2 "message": "Database populated."  
3 }

complete ✘ Runner ⌂ Start Proxy ⏪

200 OK Without Authorization

## C. Burp Suite Log

Request	Response
2 https://386b0885-cab6-454.. GET /createdb	200 812 JSON ✓ 104.18.35.243 cf_bm=KN3Ps... 22:08:14 22
3 https://386b0885-cab6-454.. GET /createdb	200 812 JSON ✓ 104.18.35.243 cf_bm=SpwJF... 22:11:40 22
4 https://386b0885-cab6-454.. GET /users/v1/_debug	200 1016 JSON ✓ 104.18.35.243 cf_bm=H8_S5... 22:12:41 22
5 https://386b0885-cab6-454.. GET /users/v1/_debug	200 1016 JSON ✓ 104.18.35.243 cf_bm=mlUpV... 22:13:54 22
6 https://386b0885-cab6-454.. POST /users/v1/register	✓ 200 873 JSON ✓ 104.18.35.243 cf_bm=vzKHO... 22:18:23 22
7 https://386b0885-cab6-454.. POST /users/v1/login	✓ 200 990 JSON ✓ 104.18.35.243 cf_bm=mQ8H... 22:20:18 22
8 https://386b0885-cab6-454.. GET /users/v1/name1	200 808 JSON ✓ 104.18.35.243 cf_bm=uRY8D... 22:22:26 22
9 https://386b0885-cab6-454.. DELETE /users/v1/name1	200 809 JSON ✓ 104.18.35.243 cf_bm=mHfR0... 22:28:10 22
10 https://386b0885-cab6-454.. PUT /users/v1/name1/password	✓ 400 836 JSON ✓ 104.18.35.243 cf_bm=WtVqu... 22:33:13 22
11 https://386b0885-cab6-454.. PUT /users/v1/name1/password	✓ 400 836 JSON ✓ 104.18.35.243 cf_bm=spNlb... 22:39:53 22

### 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.

# 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*