

# API Security Risk Analysis Report

---

Saghana K S  
6/02/2026

# 1. Introduction

Application Programming Interfaces (APIs) enable communication between different software systems. In cybersecurity, APIs must be tested to ensure they handle requests securely and do not expose sensitive data. This report documents basic API security testing performed using Postman. The objective is to analyze HTTP methods, headers, response data, and identify potential security risks without exploiting the API.

## 2. Tools Used

- Postman (API testing tool)
- httpbin.org (Public API testing service)

## 3. Testing Methodology

The testing was performed by sending HTTP requests using Postman. Public endpoints provided by httpbin.org were selected to safely analyze API behavior. GET and POST requests were tested, headers were inspected, and responses were analyzed using HTTP status codes and JSON output.

## 4. Test Case 1 – GET Request

### Objective:

To retrieve data from the server and analyze the response.

## API Details:

- **Method:** GET
- **Endpoint:** <https://httpbin.org/get>

## Result:

- **Status Code:** 200 OK
- **Response Type:** JSON

## Observation:

The server successfully returned data without requiring authentication, indicating an open endpoint.

The screenshot shows the Postman application interface. On the left, the sidebar displays 'Saghana K S's Workspace' with collections, environments, history, and flows. The main workspace shows a 'GET https://httpbin.org/get' request. The 'Params' tab is selected, showing an empty table. Below it, the 'Body' tab shows a JSON response with the following content:

```
{}  
{"args": {},  
 "headers": {  
     "Accept": "*/*",  
     "Accept-Encoding": "gzip, deflate, br",  
     "Host": "httpbin.org",  
     "Postman-Token": "c73186d4-7ab6-43e5-b33e-b42ee8c167ef",  
     "User-Agent": "PostmanRuntime/7.51.1",  
     "X-Amzn-Trace-Id": "Root=1-6985f612-49c634542c033ca758c0cd02",  
     "origin": "27.5.12.98",  
     "via": "HTTP/2.0 proxy" }  
,"json": {},  
"script": {},  
"status": 200,  
"statusText": "OK",  
"time": "2.10 s",  
"type": "raw",  
"url": "https://httpbin.org/get"}  
200 OK  
2.10 s · 601 B
```

To the right of the request, a troubleshooting panel titled 'Troubleshooting Invalid Protocol Error' provides steps to resolve the issue. It includes a code editor with a failing POST test and a note about running the real response. A sidebar at the bottom right contains an AI feature and a 'GET Untitled Request' section.

# 5. Test Case 2 – POST Request

## Objective:

To retrieve data from the server and analyze the response.

## API Details:

- **Method:** POST
- **Endpoint:** <https://httpbin.org/post>
- **Payload:** Username and password in JSON format

## Result:

- **Status Code:** 200 OK
- **Response Type:** JSON (data echoed by server)

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Saghana K S's Workspace' containing 'Collections', 'Environments', 'History', 'Flows', and 'Files (BETA)'. The main area shows a collection named 'My Collection' with a 'Get data' and a 'POST Post data' request. The 'POST Post data' request is selected, showing its details. The 'Body' tab is active, displaying the JSON payload:

```
1 {
2   "username": "testuser",
3   "password": "testpass"
4 }
```

The 'Headers' tab shows the following headers:

```
1 {
2   "Accept": "*/*",
3   "Accept-Encoding": "gzip, deflate, br",
4   "Content-Length": "59",
5   "Content-Type": "application/json",
6 }
7 
```

The 'Body' tab shows the raw JSON response received from the server:

```
1 {
2   "args": {},
3   "data": "{'username': 'testuser', 'password': 'testpass'}",
4   "files": {},
5   "form": {},
6   "headers": {
7     "Accept": "*/*",
8     "Accept-Encoding": "gzip, deflate, br",
9     "Content-Length": "59",
10    "Content-Type": "application/json",
11  }
12 }
13 
```

The status bar at the bottom indicates a '200 OK' response with a duration of '1.03 s' and a size of '867 B'. To the right, there's a 'Troubleshooting Invalid Protocol Error' panel with code snippets for handling JSON responses and a note about running on real responses.

## Observation:

The API accepted and processed the data successfully. No authentication or validation was enforced on the submitted data.

## 6. Test Case 3 – Header Analysis

### Objective:

To retrieve data from the server and analyze the response.

### API Details:

- **Method:** GET
- **Endpoint:** <https://httpbin.org/headers>
- **Custom Header Added:** Authorization: Bearer sampletoken123

### Result:

- **Status Code:** 200 OK
- **Response:** Server reflected received headers

## Observation:

The Authorization header was accepted without validation, demonstrating how APIs receive authentication-related headers.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (My Collection), 'Environments', 'History', and 'Flows'. The main area shows a collection named 'My Collection' with two requests: 'Get data' and 'Post data'. The 'Get data' request is selected, pointing to the URL <https://httpbin.org/headers>. The 'Headers' tab is active, showing one header: 'Authorization' with the value 'Bearer samplertoken123'. The 'Body' tab shows a JSON response with various headers and tokens. On the right, there's a 'Troubleshooting Invalid Protocol Error' section with steps 4 and 5, and a code editor with a failing test script.

```

4. Keep your body as raw JSON:
</>
{
  "email": "eve.holt@reqres.in",
  "password": "cityslicka"
}

5. Send the request again.
Your existing test:
pm.test("Successful POST request", function() {
  pm.expect(pm.response.code).to.be.oneOf([200, 201]);
});

```

will then run on the real response.  
If you'd like, next I can:

- Add a test that checks the response body (e.g. token field exists), or
- Help you create a similar Get data request in your My Collection.

**GET Untitled Request**  
Describe what you need. Press @ for context, / for help.

## 7. Identified Security Risks

Risk Identified	Description	Severity
Open Endpoints	API endpoints accessible without authentication	Medium
Missing Authentication	No enforcement of authentication tokens	High
Header Misuse	Authorization headers accepted without validation	Medium
Input Validation	User input accepted without checks	Medium
Rate Limiting	No visible rate limiting mechanism	Low

## 8. Risk Severity Classification

- **High Risk:** Missing authentication mechanisms
- **Medium Risk:** Open endpoints, header misuse, input validation issues
- **Low Risk:** Rate limiting not enforced

## 9. Suggested Remediation Steps

- Implement proper authentication and authorization mechanisms
- Use secure token-based authentication (JWT, OAuth)
- Validate user input to prevent injection attacks
- Apply rate limiting to prevent abuse
- Restrict access to sensitive endpoints

## 10. Conclusion

This task provided practical exposure to API security testing using Postman. By testing GET and POST requests and analyzing headers, several potential security risks were identified. The exercise improved understanding of API behavior, HTTP status codes, and common security weaknesses. This analysis was conducted safely without exploiting the API.

## 11. References

- <https://httpbin.org>
- Postman Documentation