

# API Security Testing - Missing Security Headers & Token Exposure

Recently, as part of my continuous learning in cybersecurity, I performed a basic API penetration test on a sample application. The goal was to understand common API security weaknesses and how they impact overall system security.

Here are some of the key issues I identified and what they mean:

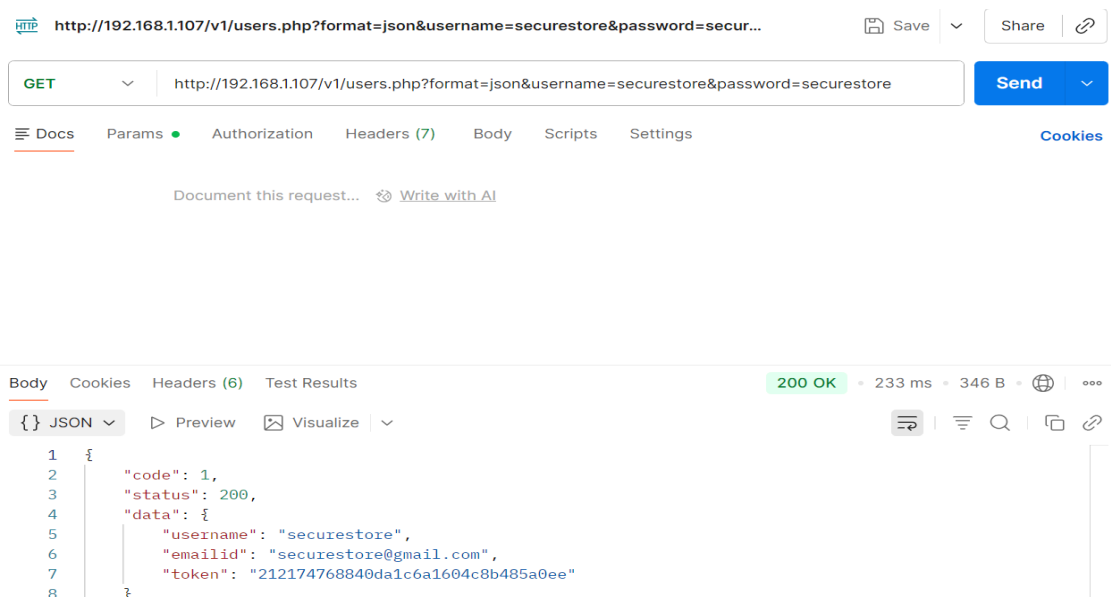
## 1. Insecure Authentication Method (Using GET Instead of POST)-Failed

The API was accepting login credentials through a GET request, meaning the username and password were visible in the URL.

This is insecure because:

- URLs are stored in browser history
- They can appear in server logs
- They may leak through referrers
- Sensitive data becomes easily exposed

Proper authentication workflows must use POST, where credentials are sent in the request body and can be protected using TLS/HTTPS.



screenshot of Postman showing 200 OK for a GET request with sensitive data.

---

## 2. Server Version Disclosure (Information Leakage)-Failed

The server response headers exposed the full server version:

Apache/2.4.18 (Ubuntu)

Attackers can use this information to:

- Identify known vulnerabilities for that exact version
- Tailor exploits more effectively
- Plan targeted attacks

Servers should disable unnecessary header disclosure to limit recon and fingerprinting.

### Response

Pretty	Raw	Hex	Render
1	HTTP/1.1 200 OK		
2	Date: Wed, 19 Nov 2025 06:32:42 GMT		
3	Server: Apache/2.4.18 (Ubuntu)		
4	Content-Length: 134		
5	Connection: close		
6	Content-Type: application/json; charset=utf-8		
7			
8	{		
	"code":1,		
	"status":200,		
	"data":{		
	"username":"securestore",		
	"emailid":"securestore@gmail.com",		
	"token":"212174768840dalc6a1604c8b485a0ee"		
	}		
	}		

API response from the repeater showing the Server version.

---

API Returns Sensitive Data in Response (Sensitive Data Exposure)

## 3. Sensitive Data Exposure- Failed

- Email address

- Authentication token
- Username

**This allows account takeover if the token is valid.**

The API returns sensitive user information, including username, email ID, and authentication token in plaintext JSON. This is a direct sensitive data exposure vulnerability and could lead to account takeover if intercepted or replayed.

```
[
  "code":1,
  "status":200,
  "data":{
    "username":"securestore",
    "emailid":"securestore@gmail.com",
    "token":"212174768840dalc6a1604c8b485a0ee"
  }
]
```

---

Exposed username ,email id and token in the response.

#### **4. Missing Security Headers (CSP, HSTS, XSS Protections)-Failed**

The API and website lacked essential security headers:

- **Content Security Policy (CSP)** - helps prevent XSS
- **HSTS (HTTP Strict Transport Security)** - forces HTTPS
- **X-Frame-Options / X-Content-Type-Options** - prevent clickjacking & MIME sniffing

These headers form the foundation of modern web security.

Without these headers:

- The site could be loaded over HTTP and here it can lead to the token leakage.
- Man-in-the-middle attacks become easier
- XSS risks increase
- Session security is weaker

---

## 5. Username Enumeration Risk-**Passed**

The API didn't respond differently depending on whether the username existed.

Example:

- Valid username + wrong password = "Password incorrect"
- Invalid username = "User does not exist"

This behaviour allows attackers to:

- Confirm valid usernames
- Prepare targeted brute-force attacks
- Reduce effort required for credential stuffing

A secure system should always return a generic error message such as: "Invalid username or password" like below.

---

### Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1 401 Unauthorized			
2	Date: Wed, 19 Nov 2025 06:57:02 GMT			
3	Server: Apache/2.4.18 (Ubuntu)			
4	Content-Length: 61			
5	Connection: close			
6	Content-Type: application/json; charset=utf-8			
7				
8	{			
	"code":4,			
	"status":401,			
	"data":"Invalid Username or Password"			
	}			

---

## 6. No Rate Limiting => Password Brute Force Vulnerability-**Failed**

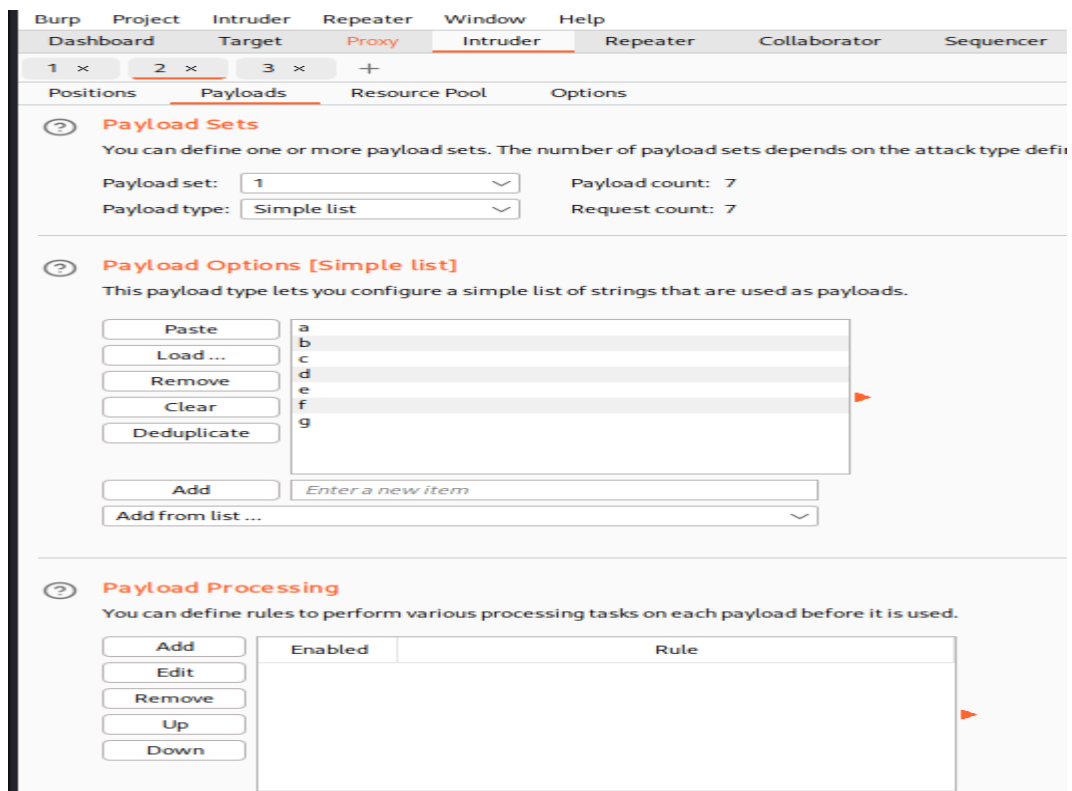
There was no rate limiting or lockout mechanism on login attempts.

Using an intruder attack, I noticed:

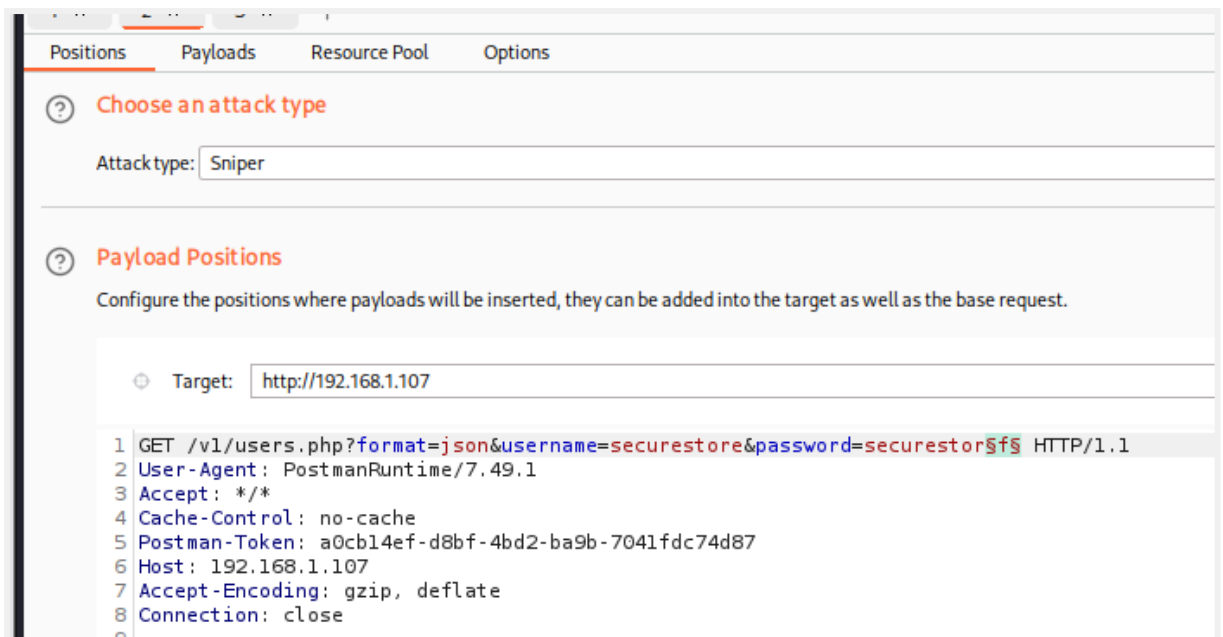
- Response sizes changed when the correct character was used
- This can leak information during brute-force attempts
- Attackers can automate login attempts at scale

Proper protections include:

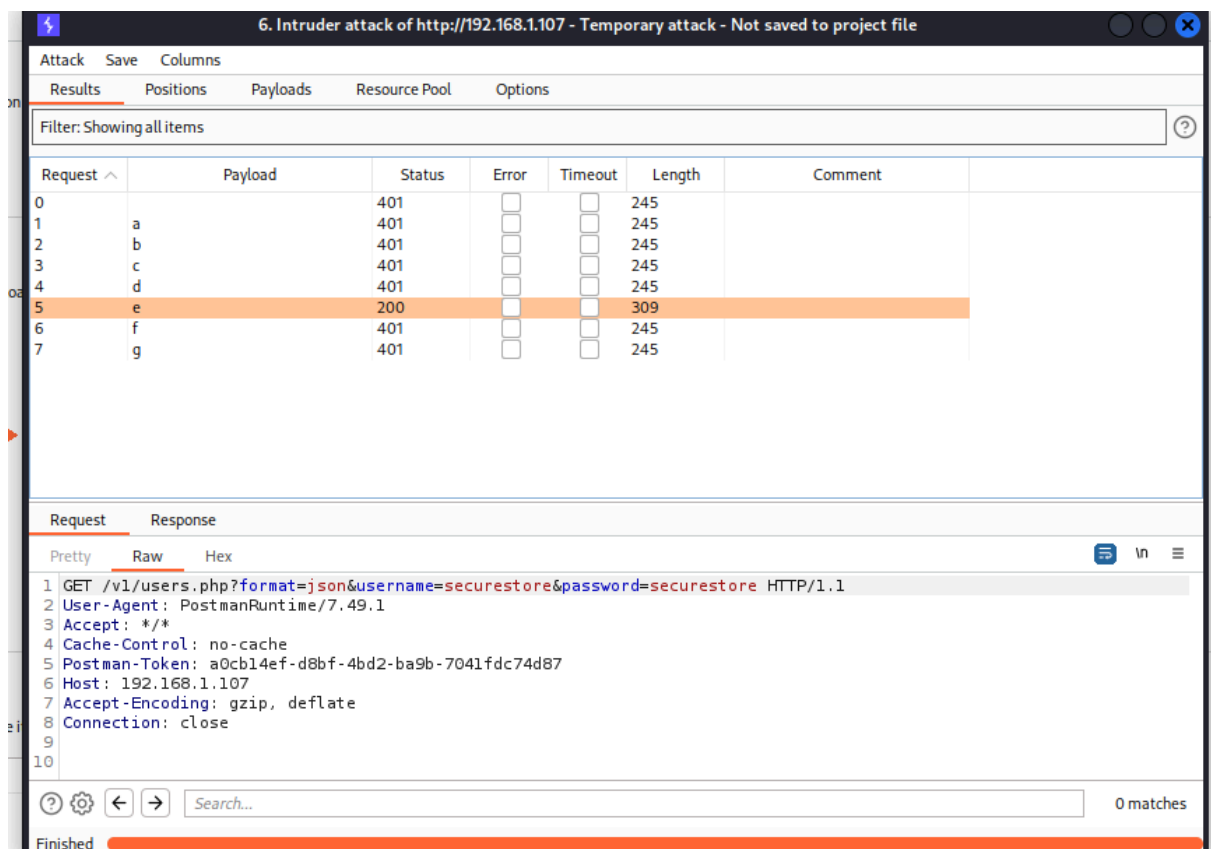
- Rate limiting (429 Too Many Requests)
- Account lockout
- CAPTCHA
- Monitoring abnormal login behaviour



Screenshot showing the payload



Screenshot showing modified password with attack position markers.



Attack result showing different status and length parameters for letter e, confirming the letter is correct.

## Conclusion

This self-driven API security assessment helped strengthen my understanding of:

- API attack surfaces
- Authentication weaknesses
- Header security
- Enumeration risks
- Brute-force defence mechanisms

API security is a critical component of modern application security, and even small misconfigurations can significantly increase risk. I am continuing to explore more areas of penetration testing and API hardening as part of my learning journey