Skills Assessment

Scenario

You are given access to a web application with basic protection mechanisms. Use the skills learned in this module to find the SQLi vulnerability with SQLMap and exploit it accordingly. To complete this module, find the flag and submit it here.

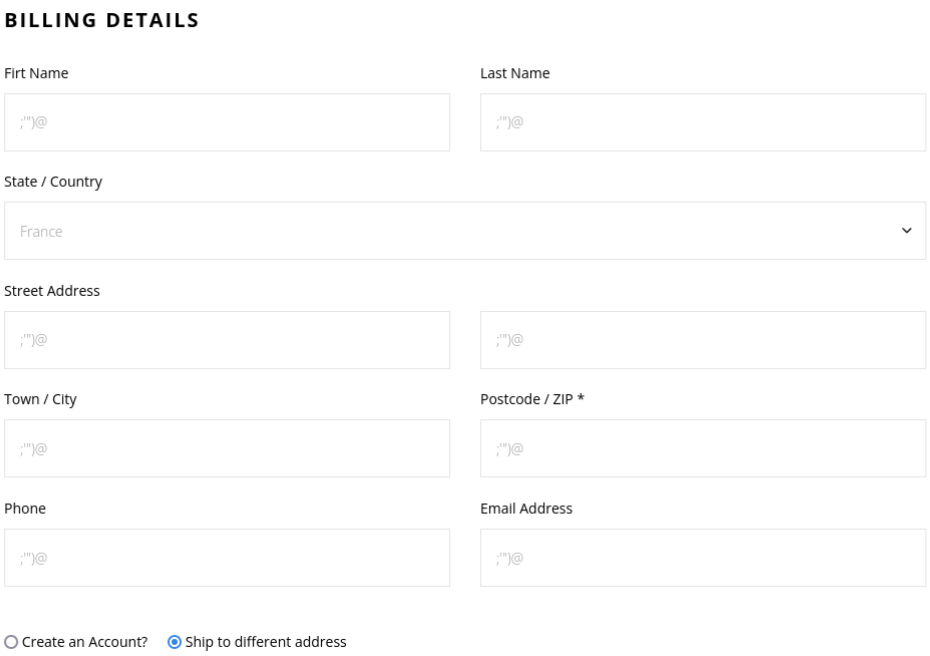
What's the contents of table final_flag?

Target: 83.136.253.44:44276

I started by loading up the page and looking for input fields as obvious places for my injection.

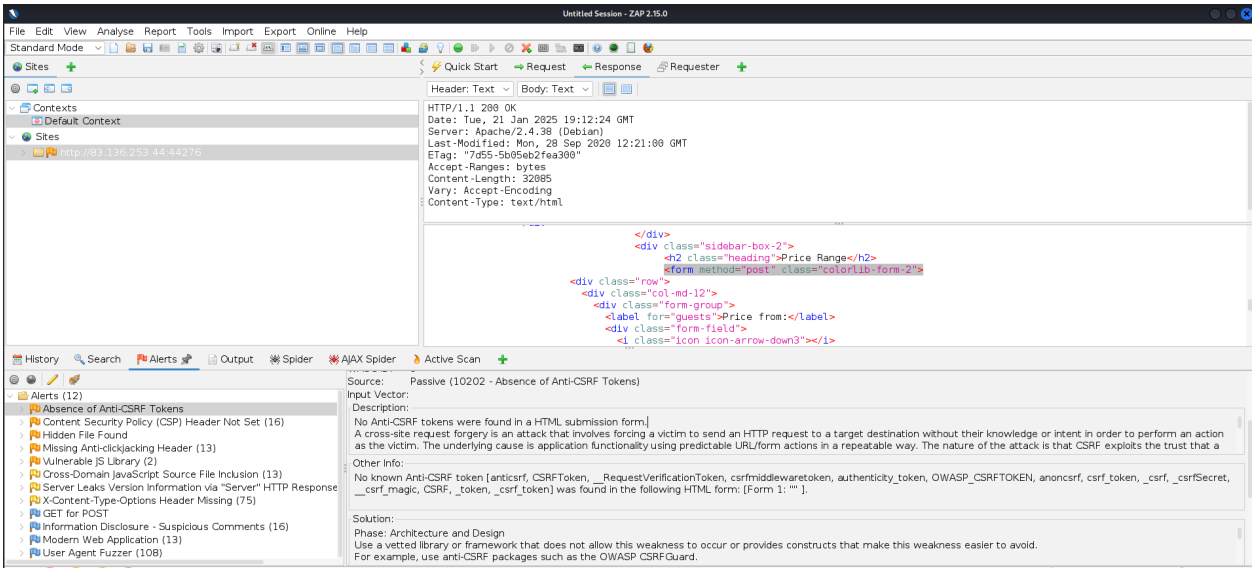
After quickly looking around I found 2 input field that was taking using input instead of utilizing drop downs or options.

One being the quantity field when placing an order, and the other being the billing details page after inserting a quantity and clicking buy. Neither option threw an error for me when manually injecting bad characters.



For the sake of utilizing knowledge in other sections, I thought using ZAP would be an interesting way of automating finding a vulnerable parameter even though I'm sure SQLmap would identify one.

So I ran the automating vulnerability scanned in ZAP against the site, and didn't find anything particularly of value



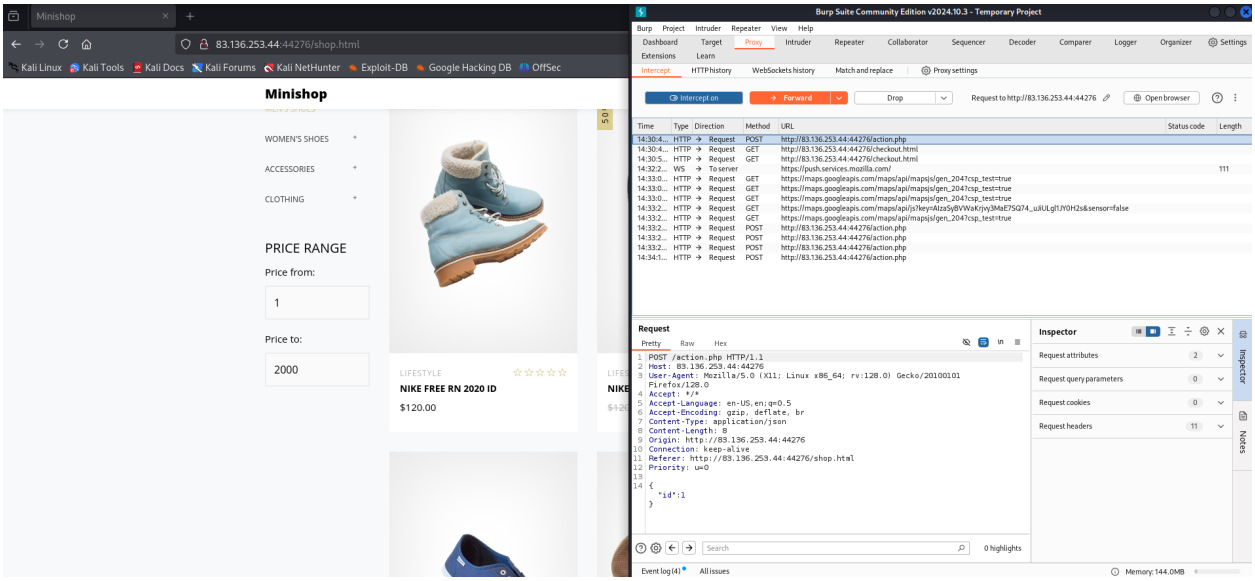
Looking into the CVE's in the vulnerable JavaScript libraries, they turned out to be XSS CVEs so they don't seem applicable to this module

So with that not identifying an SQL injectable point and my manual skimming not finding anything directly I decided to just point SQL map at it

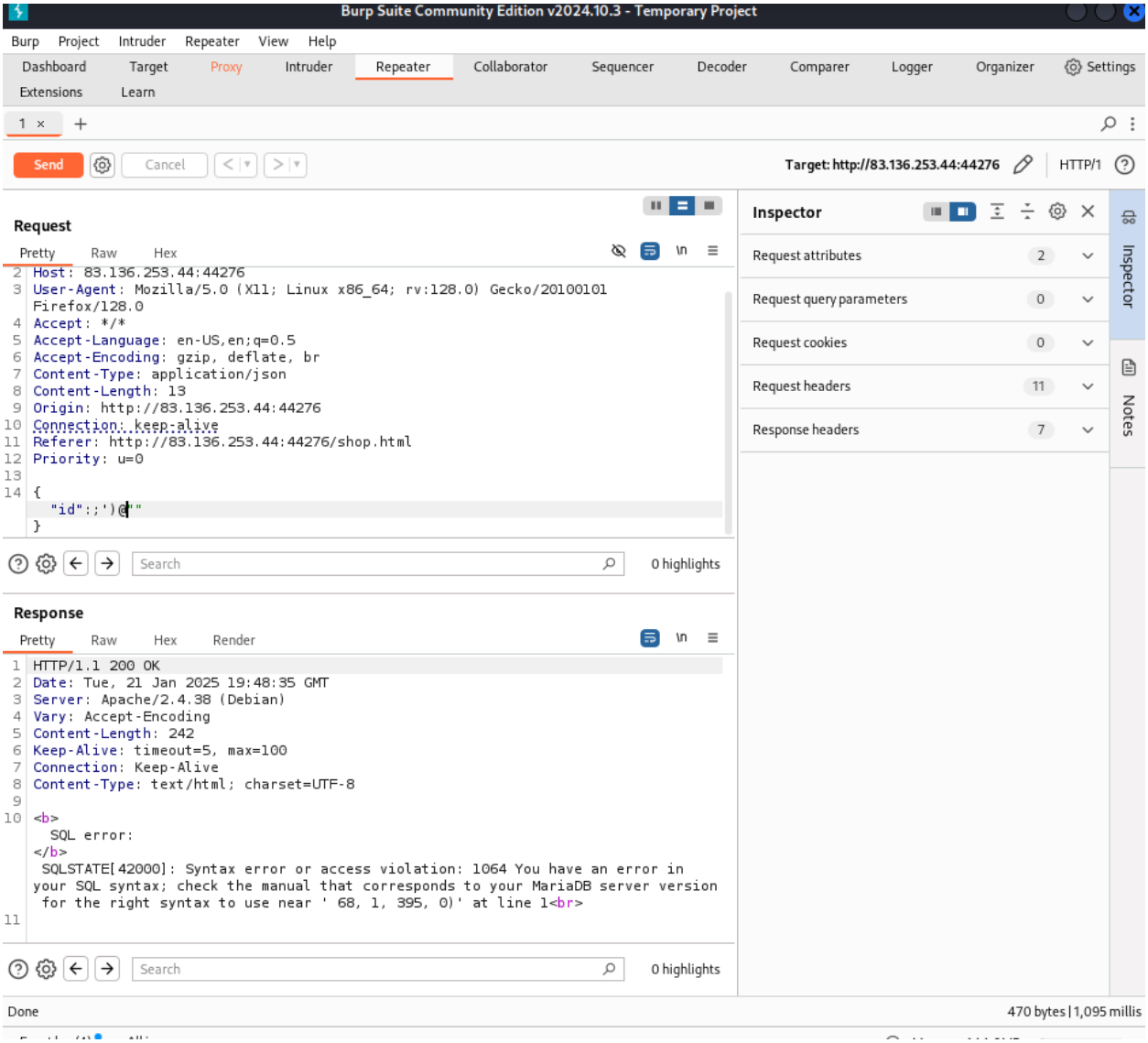
```
sqlmap -u http://83.136.253.44:44276/index.html --level=5 --risk=3 --dump -vv --batch --threads=10
```

and while that was running I kept looking at the site to see if I could find a place to narrow down sqlmap to

With my Burp proxy open, I continue clicking around the site to try a injectable parameter and I found a POST request being sent to /action.php being referred from the shop page when you click "add to cart"



Sending that request to repeated and manually injecting some bad characters into the parameter, the server throws an error in the response. This confirms my suspiscious that this is the injectable parameter.



In burp I right clicked on the request and then “saved item” to save that post request. Then I ran SQLmap against the request

```
sqlmap -r injection.req --level=5 --risk=3 --batch -vv
```

that was running for awhile just as the base sqlmap run, but the prompt does imply that there should be some basic protections in place.

Going through the list of bypassing web application protections:
There is no Anti-CSRF token, so we don’t need to bypass one

There aren’t any unique values being passed into the post request

No calculated parameters

So, I could try IP address concealing using a proxy, but that one seemed like a bit more work so I figured I’d come around to this one if the others don’t work

User-agent blacklist bypassing: I Tried running sqlmap with the —random-agent flag, but nothing of interest occurred

So at this point I figured it was probably a tamper script being required. The module mentions that between is one of the most popular tamper scripts utilized so i went with that one first and it ended up clearing up some errors I was getting.

(its injection3 because I had to recapture the web request after restarting the instance twice)

```
sqlmap -r injection3.req --level=5 --risk=3 --batch -v6 --tam
per=between --dump --threads=10
```

This took a LONG TIME. In hindsight it would have been better to use the —schema flag and then specify the db / table I wanted to dump.