CTF#2 - REPORT

Sujith Bellam

MAY 11, 2021

45664455

Abstract

This report contains the COMP2320 CTF-2 report of Group-9 (Sujith Bellam, Beau Williams, Daniel Zappala, Liam Strang, Udit Mahajan, and Harsh Patel). CTF-2 consisted of three challenges, Reach, Courses and Learna. Reach uses curl post request, and it is used for getting the flag; courses have a website called Omega, and the flag was found by SQL injection, and Learna is a website that will accept image files once logged in, the flag was found by uploading PHP scripts.

Table of Contents

1. Ethical Disclosure

The CTF-2 was done in a controlled environment in a virtual box for educational purposes only. We hacked the websites from hack the box, and we were authorized to use them to learn vulnerabilities and hacking for educational purposes. My team and I understand the complications of gained knowledge and know that this knowledge must not be used on real websites and servers under any circumstance without permission.

2. Scope of Work

For CTF – 2, my team was given three challenges to solve: Reach, Courses, and Learna. Each challenge contains a flag, and we can find the flags by hacking the websites. Reach website challenge was to give the host the curl command. Course challenge is a website for the university named Omega. The flag can be found once we log in to the website. Sqlmap is used for SQL injection for this website. Learna flag can be found by using Local File Inclusion(LFI) exploit. The website doed not verify the file type, so it was easy to get the flag.

3. Test Team Details

Liam Strang solved the Reach challenge. I was a minute away from getting the Reach flag, but Liam got it first. I solved the Courses challenge, and Liam Strang solved the Learna challenge. Beau Williams, Daniel Zappala, and Udit Mahajan worked on all the challenges providing helpful insights and ideas. Harsh Patel did not contribute anything neither communicated.

4. List of Tools Used

1. Kali Linux in Virtual Box
2. Google Chrome

I used Kali Linux 2021.1 OS (Operating System) to run different tools and exploits on the challenge websites on the virtual box. I used google chrome on my Windows 10 OS to accept team members of my group and start the servers of the challenges.

5. Identified Vulnerabilities

5.1 Information Gathering and CTF Steps

5.1.1 Reach Challenge

Commands Used: curl and its parameters

Figure 1 - Curl Functionality on Reach Website

$$http\ file://localhost/../flag$$

The command will print the flag HTB{ch3ck_0ut_th1s_pw4g3!}.

Figure 2 - Entering the curl host to get the flag

Figure 3 - Got the Flag for Reach Challenge

We can download the contents of the Reach challenge. The folder contains the following content

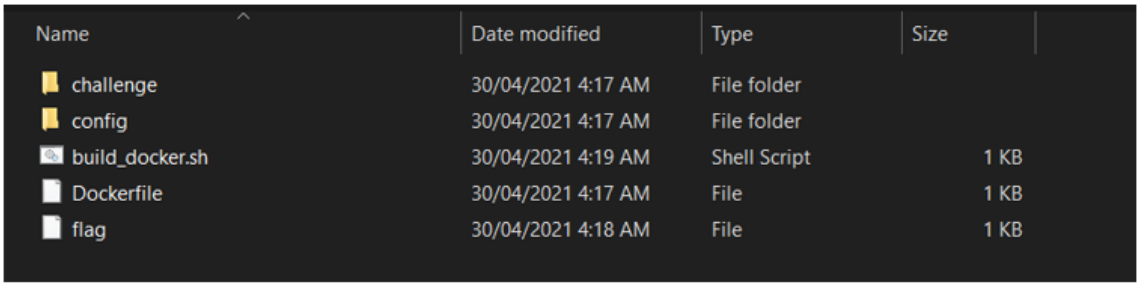
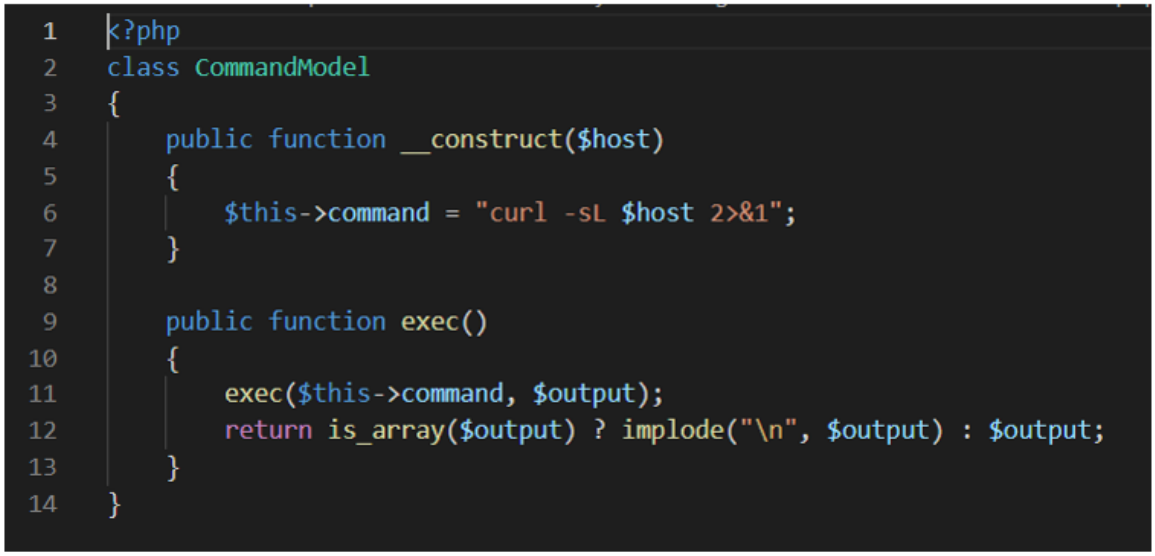


Figure 4 - Files inside the Reach Challenge

The flag file here contains a fake flag. The commandModel.php file from challenges >> models folder will show the curl command

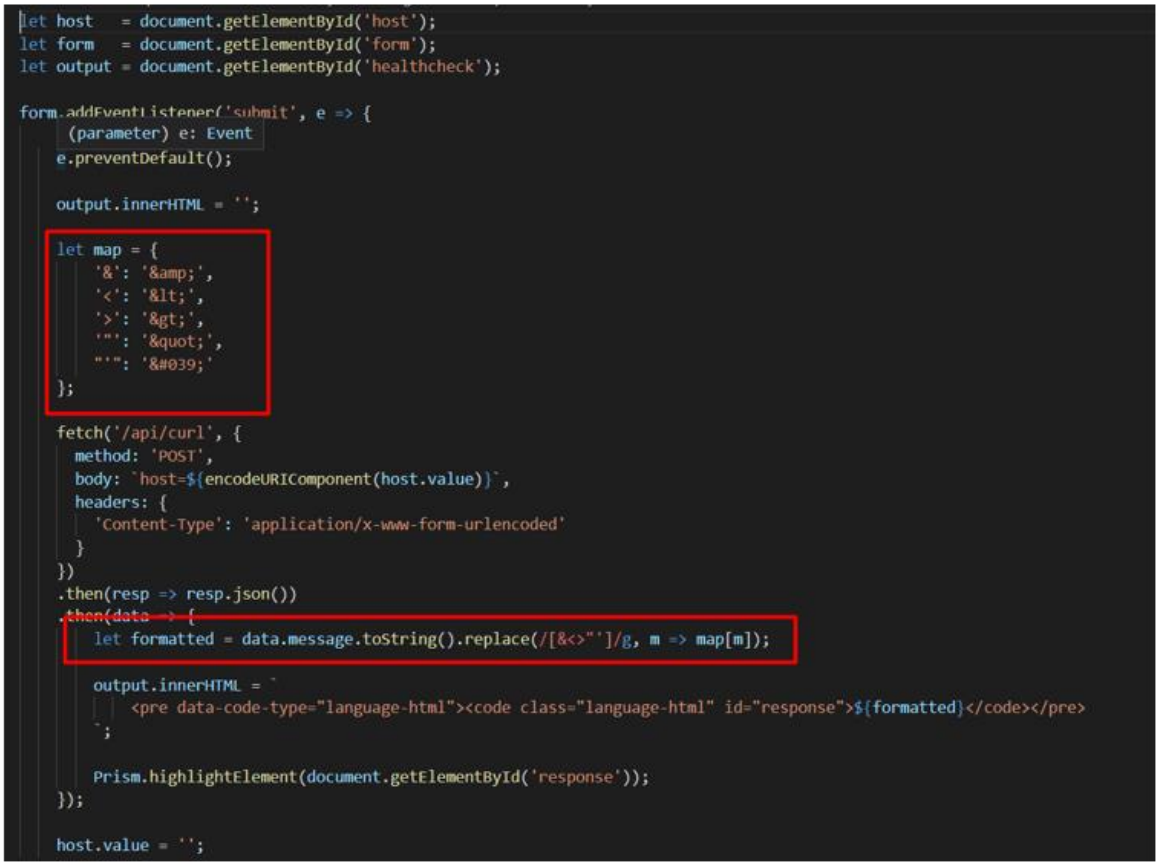


```php
<?php
class CommandModel
{
    public function __construct($host)
    {
        $this->command = "curl -sL $host 2>&1";
    }

    public function exec()
    {
        exec($this->command, $output);
        return is_array($output) ? implode("\n", $output) : $output;
    }
}
```

Figure 5 - The code inside commandModel.php file that is located in the models directory which is located in the challenges directory.

The above code tells us that we can get the flag file by giving the correct input into the host field.



```
let host   = document.getElementById('host');
let form   = document.getElementById('form');
let output = document.getElementById('healthcheck');

form.addEventListener('submit', e => {
    (parameter) e: Event
    e.preventDefault();

    output.innerHTML = '';

    let map = {
        '&': '&amp;',
        '<': '&lt;',
        '>': '&gt;',
        '"': '&quot;',
        "'": '&#039;'
    };

    fetch('/api/curl', {
      method: 'POST',
      body: `host=${encodeURIComponent(host.value)}`,
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      }
    })
    .then(resp => resp.json())
    .then(data => {
        let formatted = data.message.toString().replace(/[&<>"']/g, m => map[m]);

        output.innerHTML = `
            <pre data-code-type="language-html"><code class="language-html" id="response">${formatted}</code></pre>
        `;

        Prism.highlightElement(document.getElementById('response'));
    });

    host.value = '';
```

Figure 6 - The code is from main.js file that is located in the static directory which is in the challenges directory.

The code in the main.js file in challenge >> static >> js tells us that we cannot use any special characters mentioned in the map variable, as they will be removed. So will have to traditional curl to access the file.

```php
<?php
class CurlController
{
    public function index($router)
    {
        return $router->view('index');
    }

    public function execute($router)
    {
        $host = (isset($_POST['host'])) ? $_POST['host'] : 'http://localhost/';

        if (substr($host, 0, 4) === 'http') {
            $safeHost = escapeshellcmd($host);
        } else {
            $safeHost = 'http://localhost/';
        }

        $curl = new CommandModel($safeHost);
        return json_encode([ 'message' => $curl->exec() ]);
    }
}
```

Figure 7 - The code is from CurlController.php file that is located in the controller directory which is inside the challenges directory

The CurlController.php file on challenge >> controllers checks for the first four letters of the host and will only run the command if the first four letters entered are "HTTP" and curl http://localhost if not. So, the first four letters of the entered host have to be HTTP. As we need the contents inside the flag file, we will have to use the file as the header, and the flag file is only one directory behind it.

$$http\ file://localhost/../flag$$

Here, HTTP is required to pass through the CurlController.php file is used as we are requesting for a file and ../ mentions that the directory is the previous one and flag is the file we are requesting. Thus, we will get the flag by entering $http\ file://localhost/../flag$ as host.

5.1.2 Course Challenge

Commands used: SQLMAP and its parameters

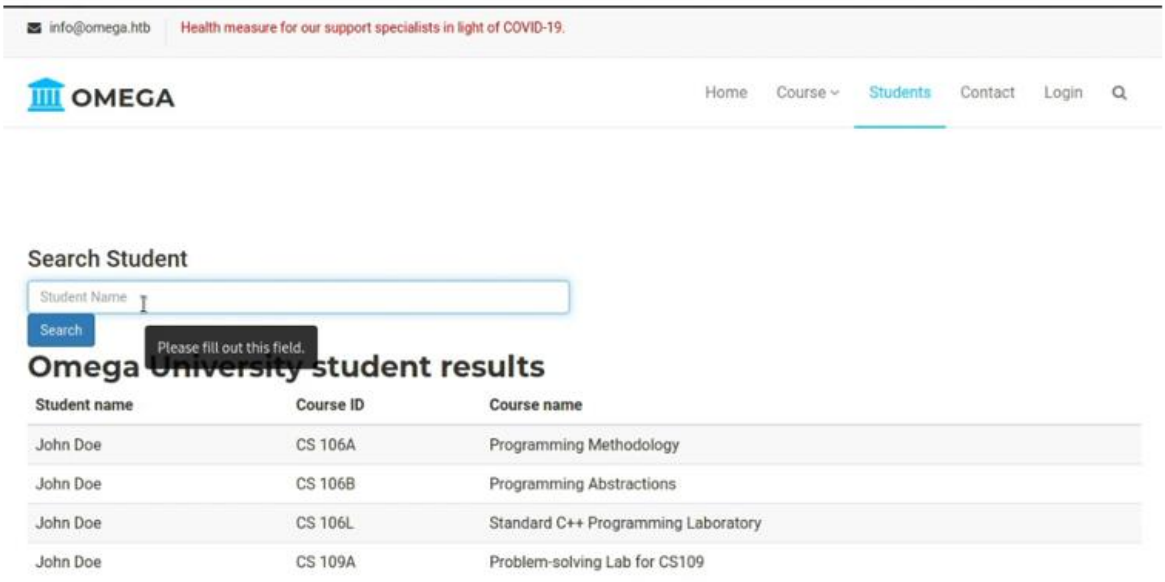


Figure 8 - students.php page from the courses challenge website

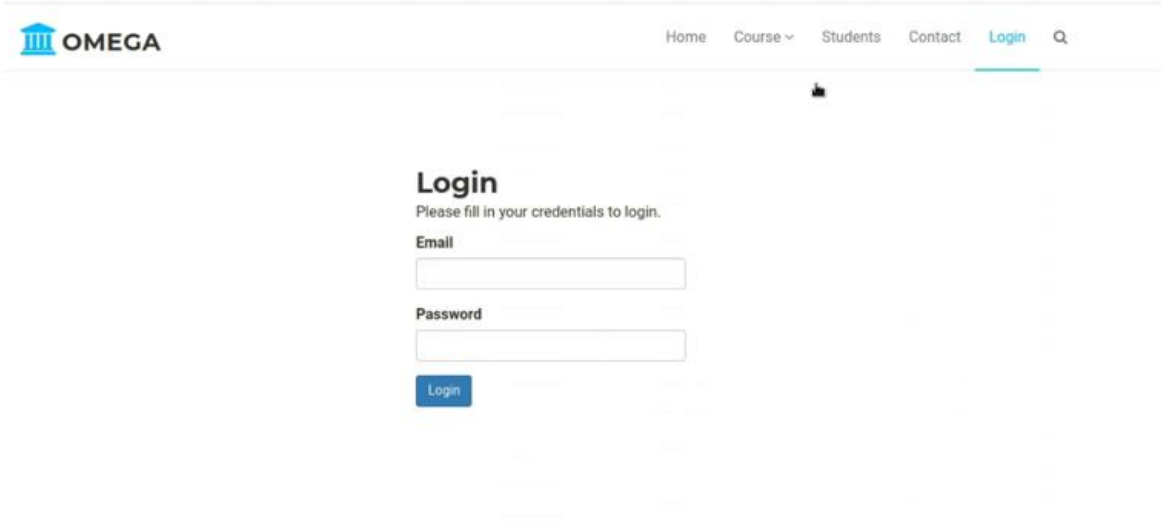


Figure 9 - login.php webpage from the courses challenge website

The Course challenge has a website. There are only two interesting pages on this website, and they are students.php and login.php in the portal directory. The best way to get the flag is to perform SQL Injection to either search form of students.php or username and password of login.php. There is no knowledge on which database is being used, neither if the query is a prepared statement or not. This

is a blind injection, and I decided to use SQLMAP.

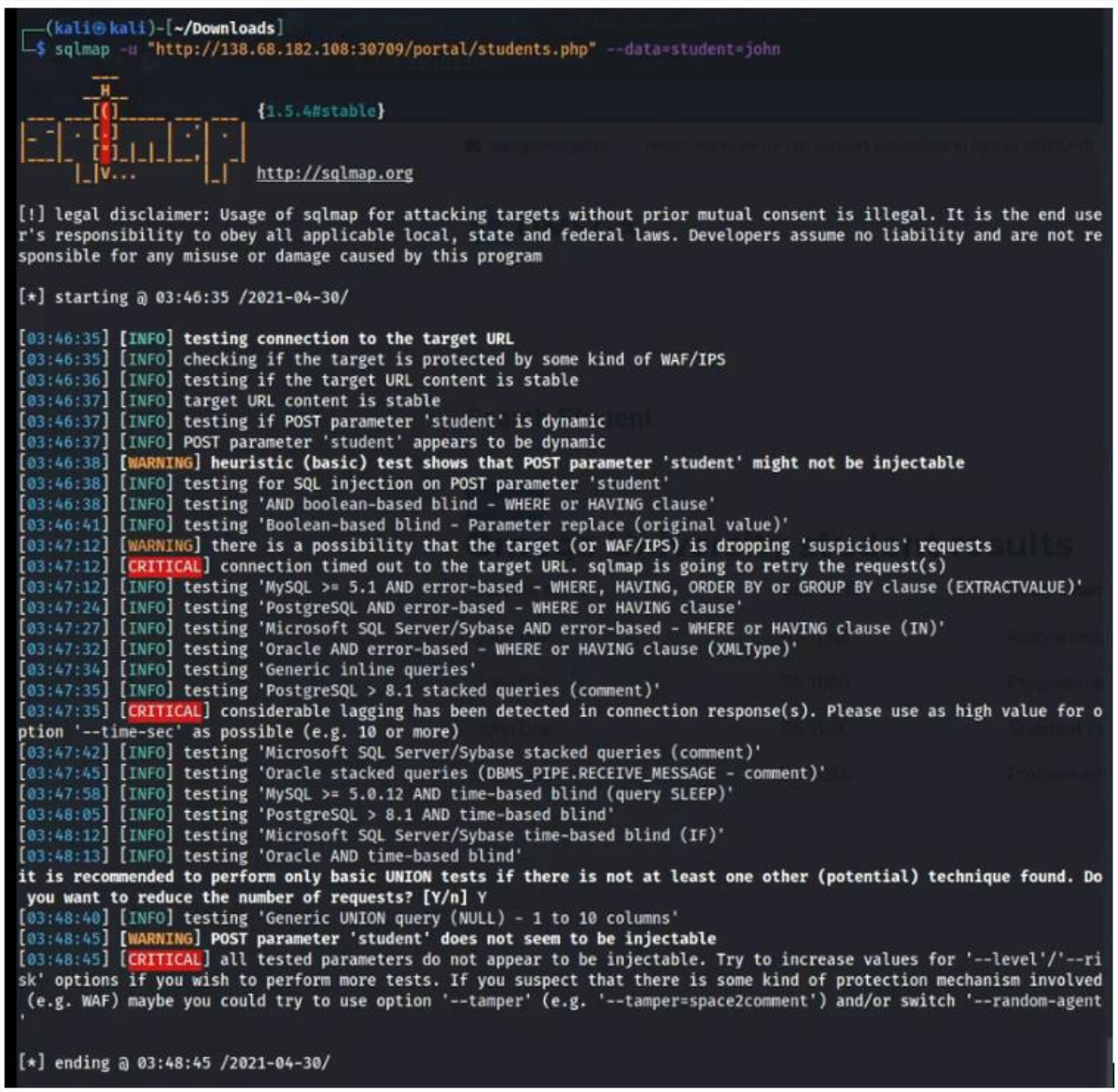


Figure 10 - Giving sqlmap URL and data

Here, -u says that the URL is provided and the URL is http://138.68.182.108:30709/portal/students.php and –data to give data, here my data is student=john.

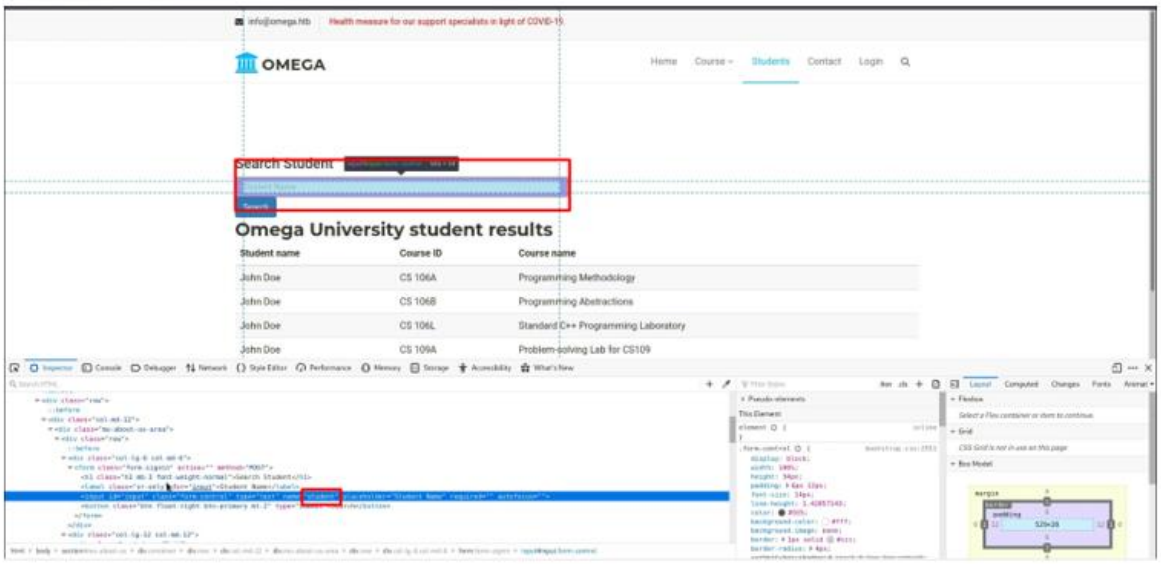


Figure 11 - Name of the input field for the search is student

The student is the name of the input box, and I'm passing the keyword john. I chose John as this name is displayed by default, and I know that it will return student(s) with the name John.

The result of the command asked to mention level and risk as it could not find any vulnerability on

the student.

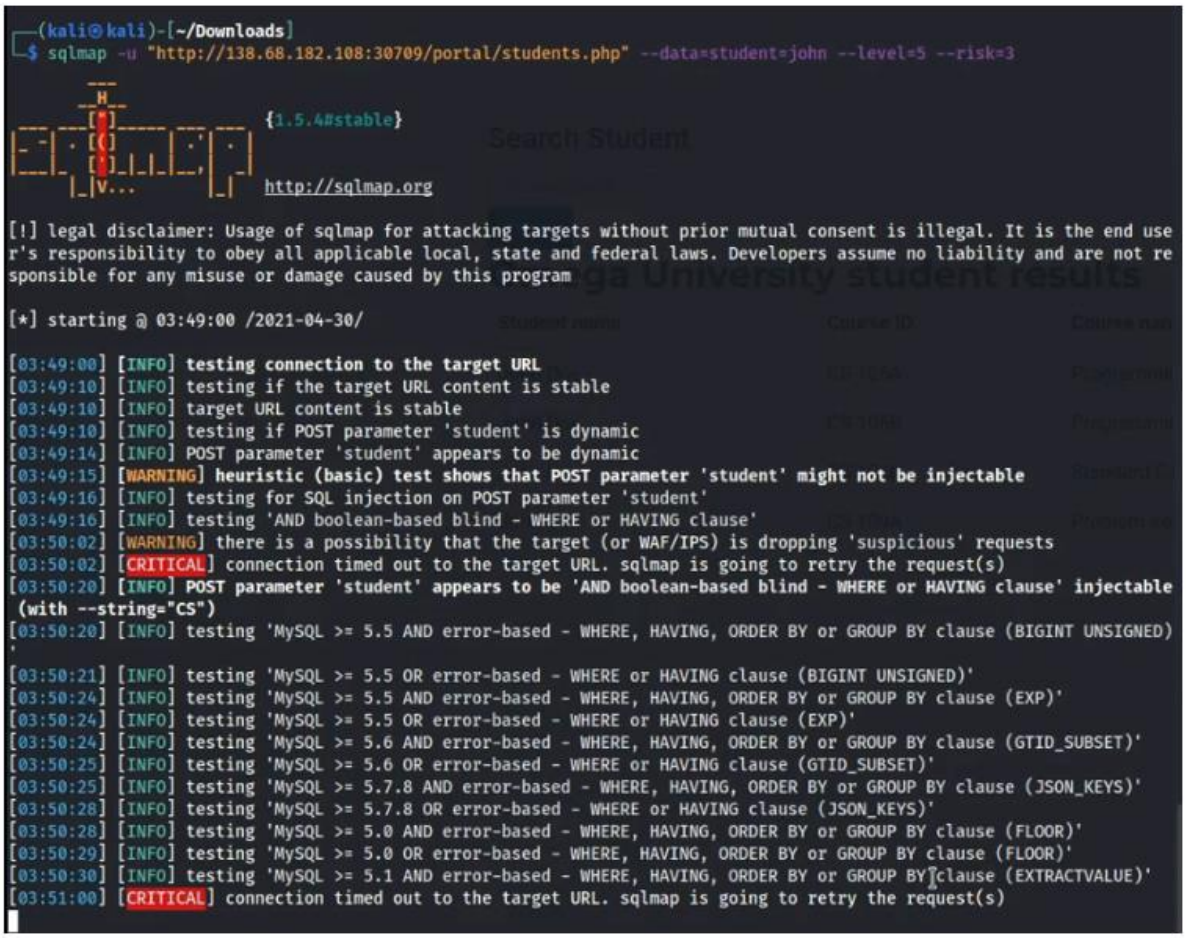


Figure 12 - Adding level and risk to sqlmap

I decided not to mention the --dbms field as I don't know what database is being used. MySQL is the database that is most commonly used with PHP.

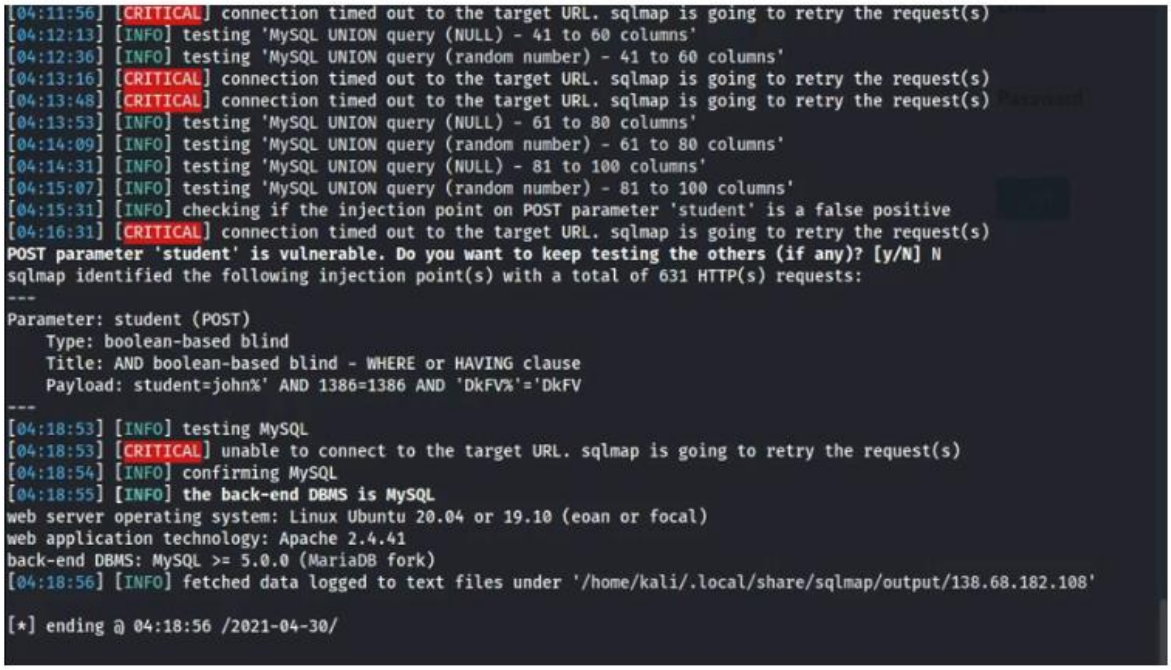


Figure 13 - sqlmap successfully identified the vulnerability as john%' AND 1=1 AND 'a%'='a

SQLMAP found the vulnerability "john%' AND 1386=1386 AND 'DkFV%'='DKFV "and the database as MySQL. Here 1386=1386 and 'DkFV%' = 'DkFV will return true. Using this, we can find all the databases.

Figure 14 - Giving sqlmap --dbs command to list all the databases and to use 10 threads (max)

I did not add --dbms as the SQLMAP already knows the database used, --dbs is used to list all the databases, and --threads is used to give more threads to SQLMAP. There are four databases, and they are

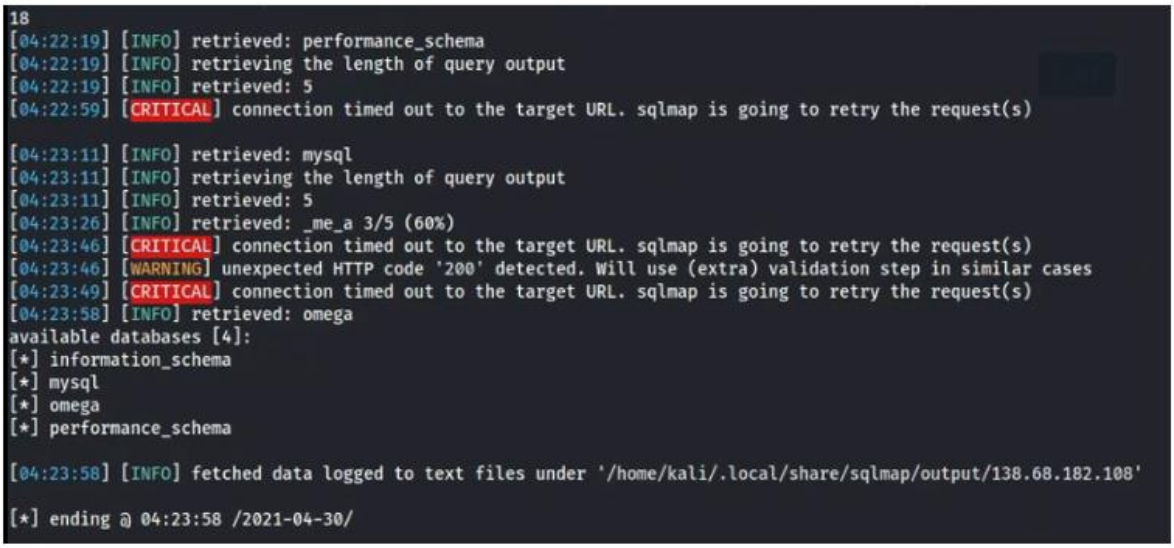


Figure 15 - sqlmap found 4 databases

mysql, information_schema, and performance_schema are the most common databases in MySQL. Omega is the database that we need.

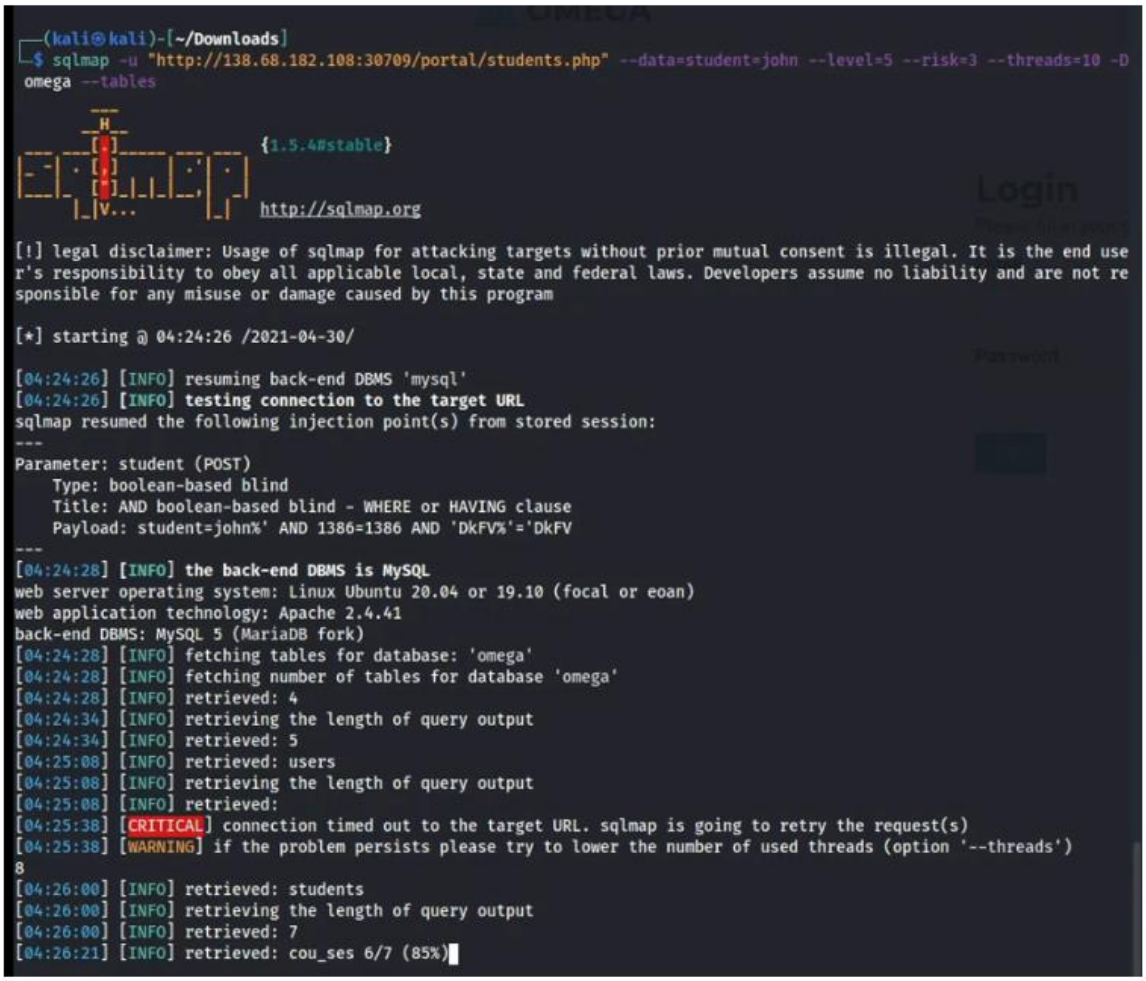


Figure 16 – Selecting omega database and giving –tables command to list all the tables inside omega database.

--D will let us choose the database, and --tables will list all the tables in that database. There are four tables in the omega database, and they are,

Figure 17 - sqlmap found 4 tables in the omega database

We are interested in the users table, as it might contain an email ID and password to log in.

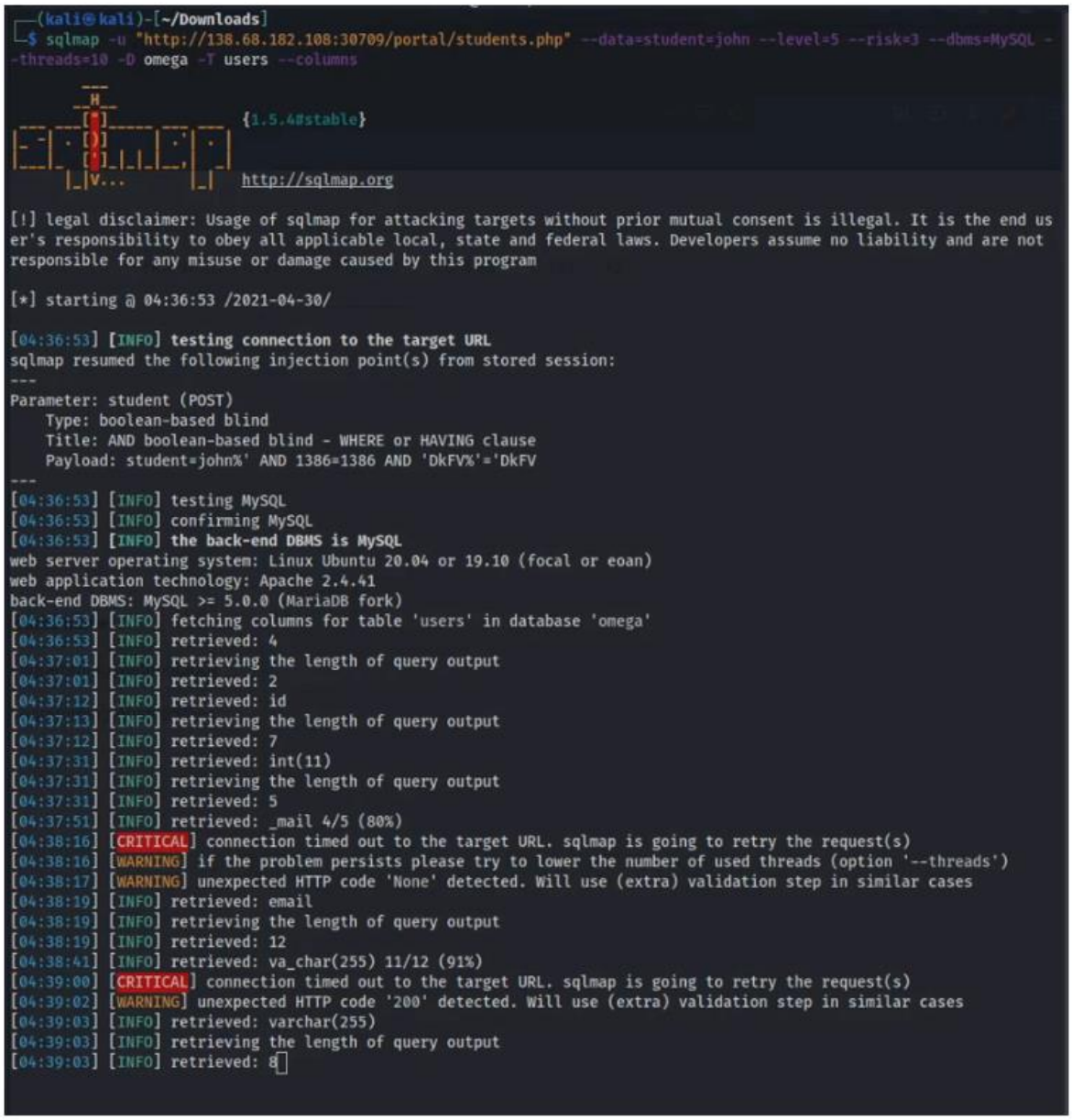


Figure 18 - Selecting the table and giving --columns to sqlmap to list all the columns in the users table

Here, -T will select the tables as users, and --columns will give all the columns inside that table. There are four columns, and they are,

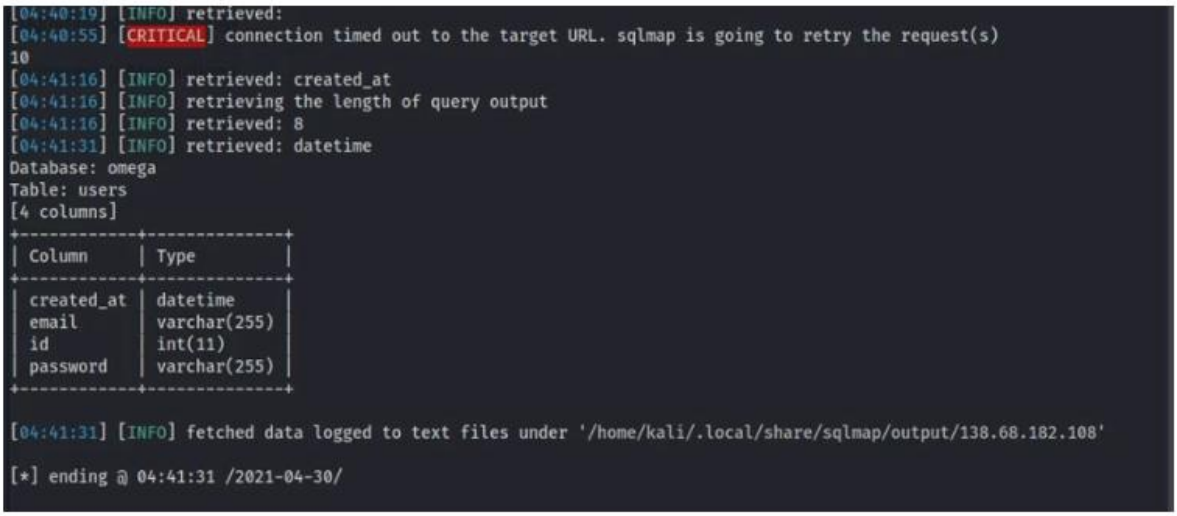


Figure 19 - sqlmap found 4 columns in the users table

We know that the users table contains email and password. Now we can get the data from the users table.

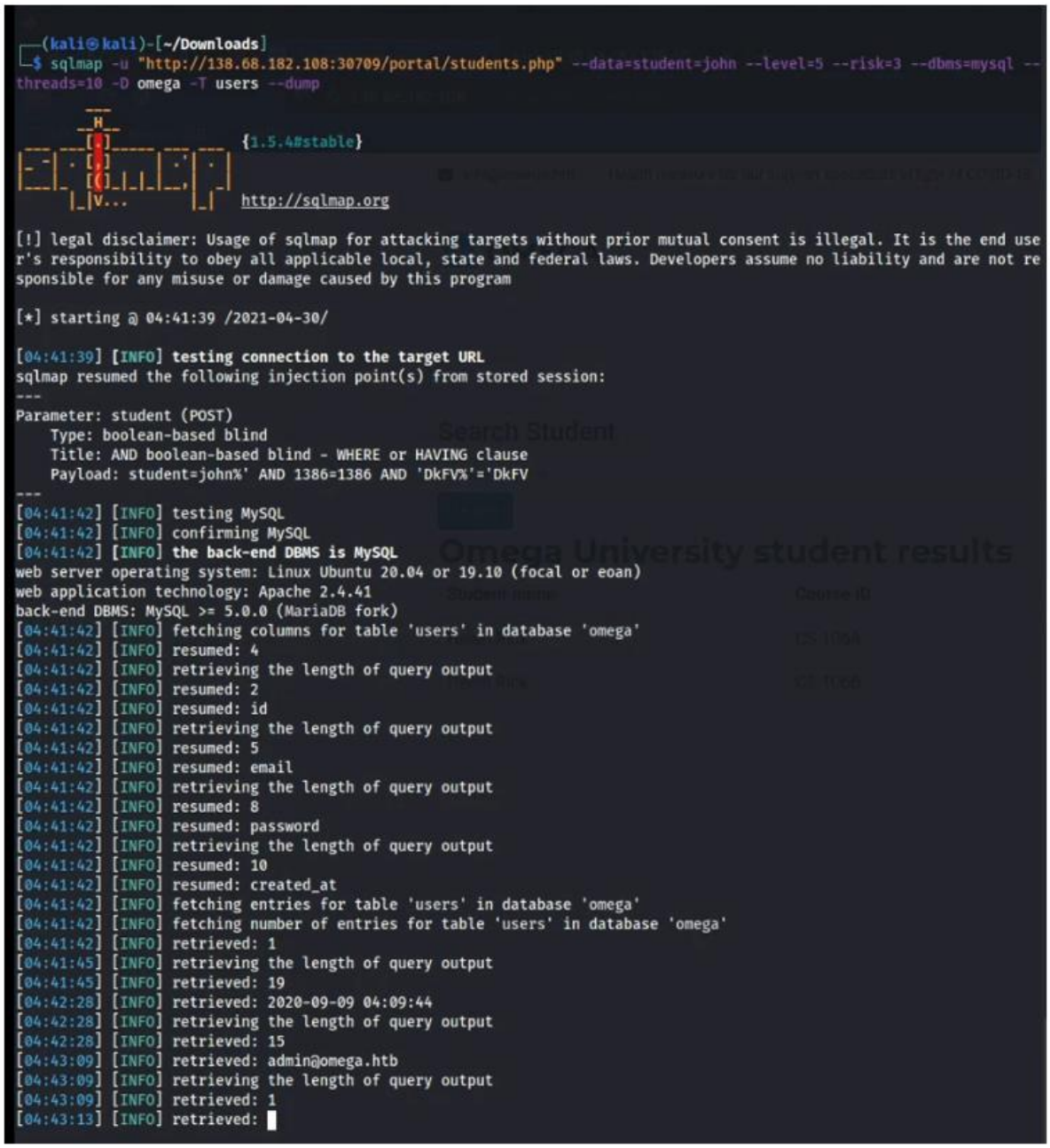


Figure 20 - Giving sqlmap --dump command to get all the information inside the users table

--dump will get all the data from the users table.

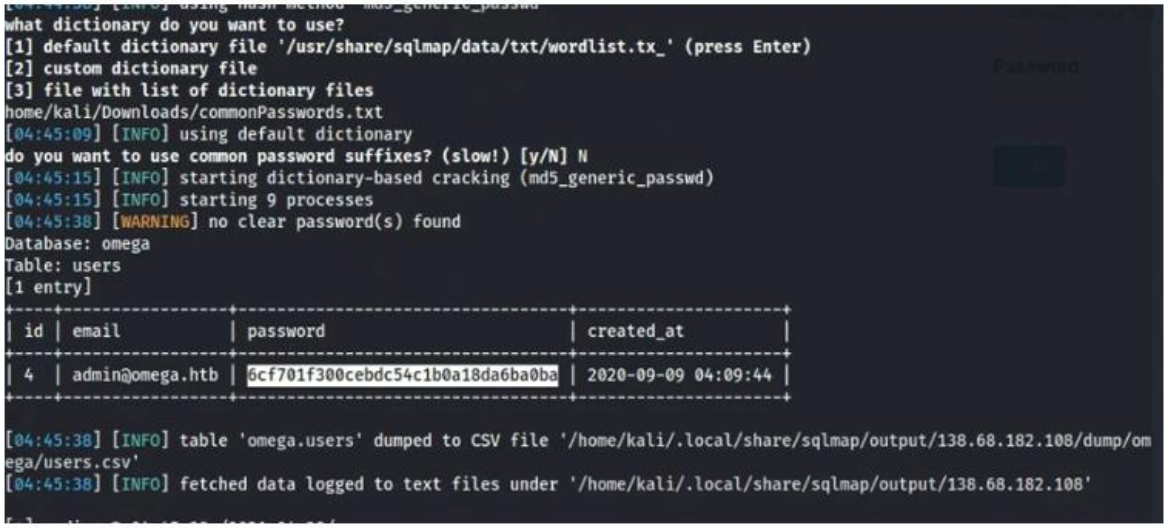


Figure 21 - sqlmap found that there is only 1 entry and its contents. The stored password is a hash.

There is only 1 row inside the users table. We have the email ID and the password as a hash. I used an online hash cracker tool to crack the hash and get the admin user's password.

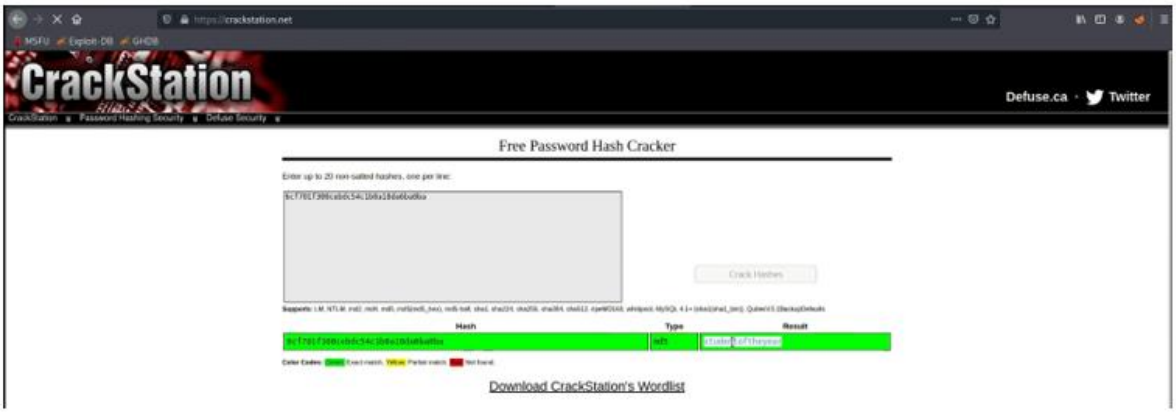


Figure 22 - Used crackstation.net website to crack the hash

The password is studentoftheyear. We can use these credentials to log in.

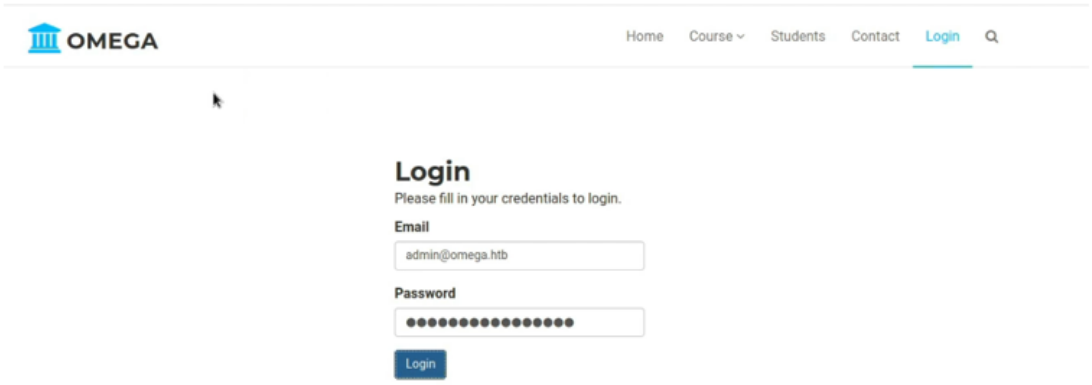


Figure 23 - Used the credentials to login to the website

Figure 24 - Found the flag for the courses challenge

Once logged in, we can find the flag “HTB{SQLi_c4n_b3_v3ry_f4n}” present on the flag.php page.

5.1.3 Learna Challenge

Commands Used: burpsuite, nano, system, get, ls cat

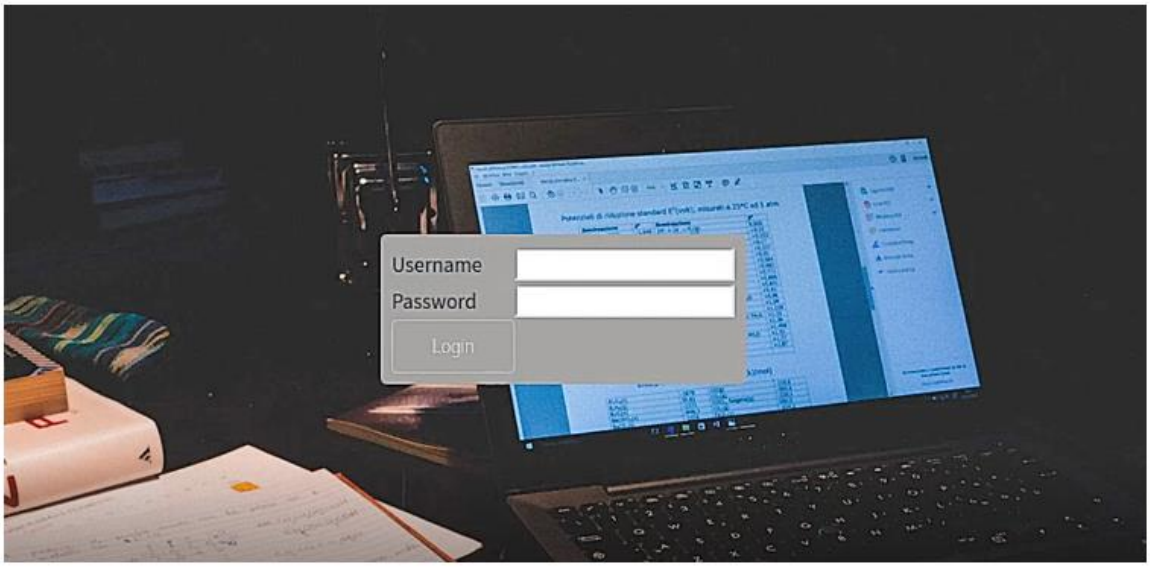


Figure 25 - Homepage of the learna challenge website requires login

This challenge opens with a login page by default. Similar to the course challenge, we can use SQLMAP to get the username and password. But I used burp suite to brute force the password. I entered the username as admin and the password as admin and sent the request through burp. The password can be anything as we are brute-forcing the password. I don't know if the username is admin; it was a guess.

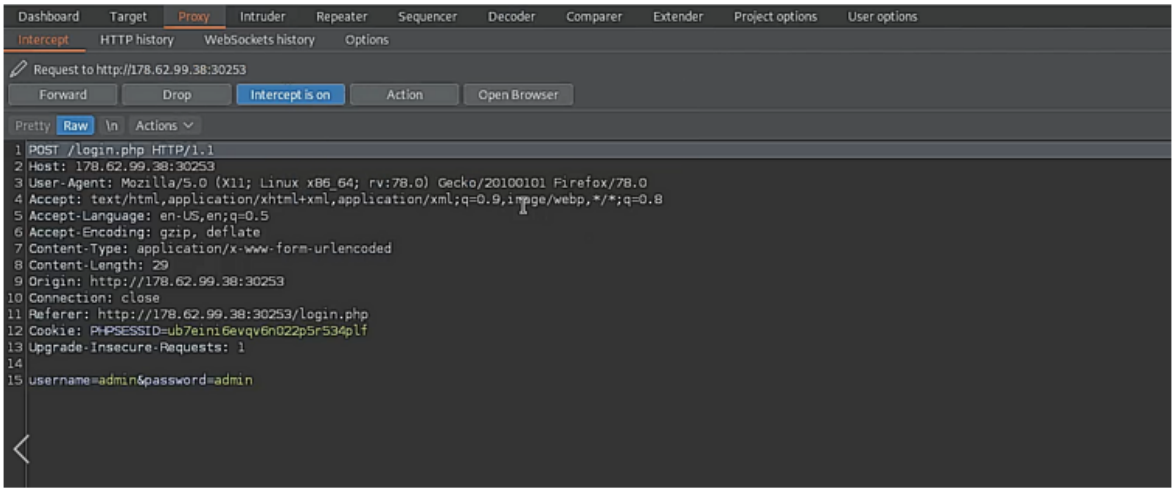


Figure 26 - Using burpsuite to intercept the login request

I then sent the request to the Intruder

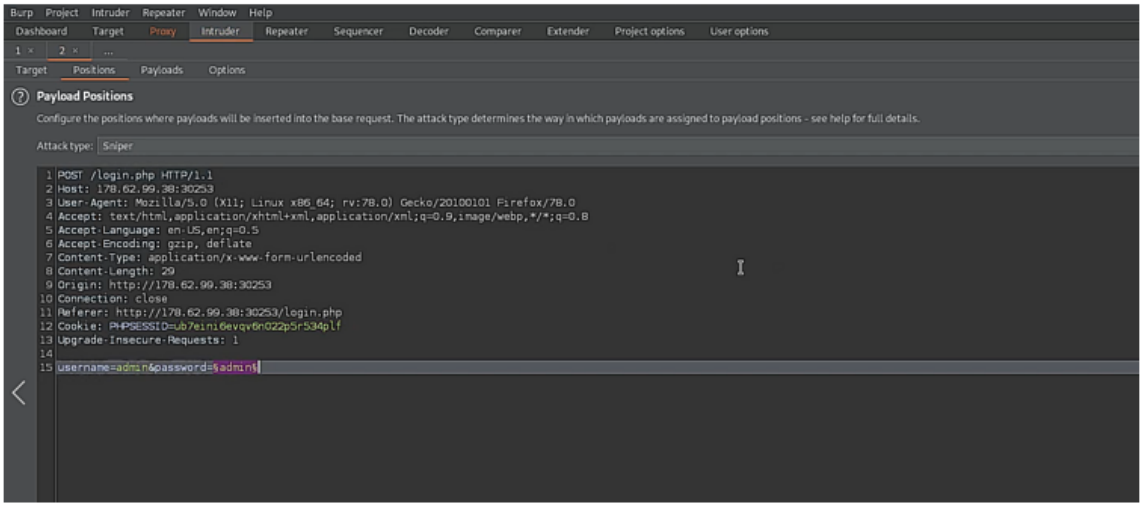


Figure 27 - Sent the login request to the intruder

and only selected the password to be brute-forced (position). The attack type will be sniper as we are only brute forcing one field.

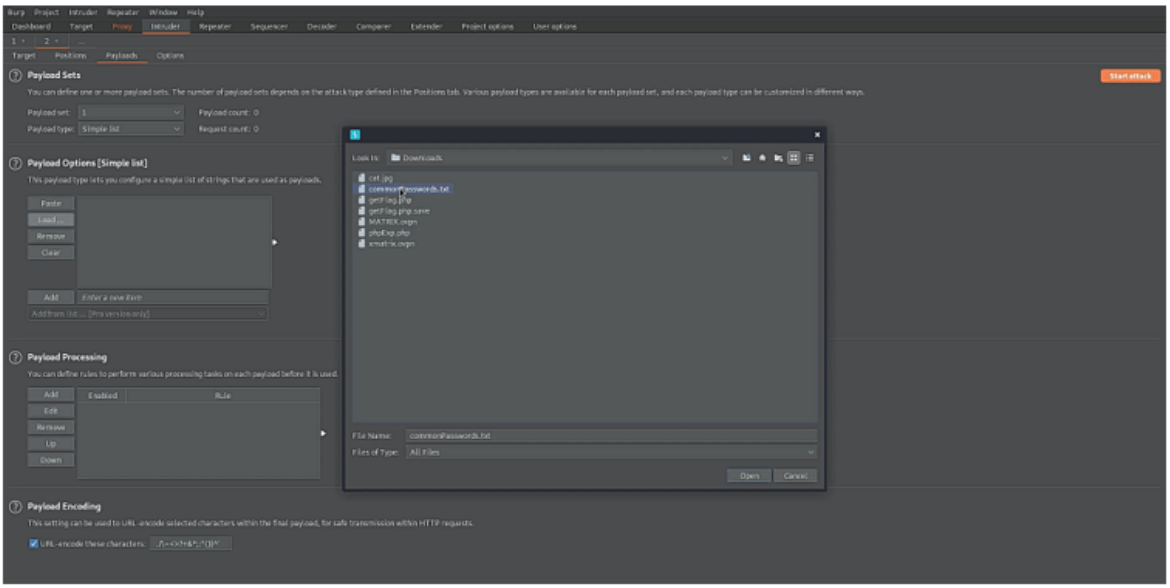


Figure 28 - Brute forcing the password with username admin

There is only one payload, i.e., password, and the payload type is a simple list. I selected my payload list, commonPasswords.txt.

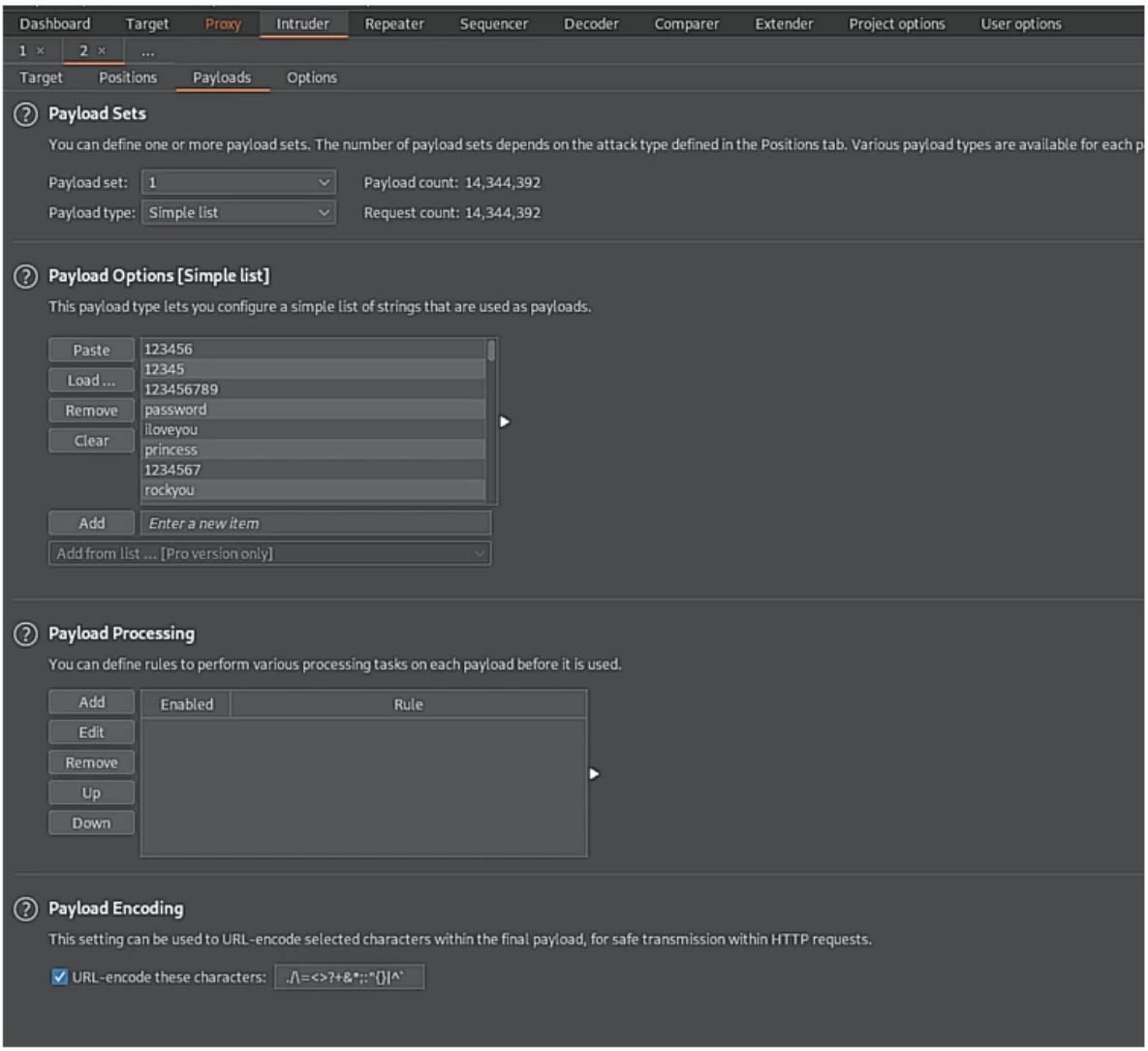


Figure 29 - loaded passwords list

All the passwords from my commonPasswords.txt file are now loaded into the payload list. Then I started the attack.

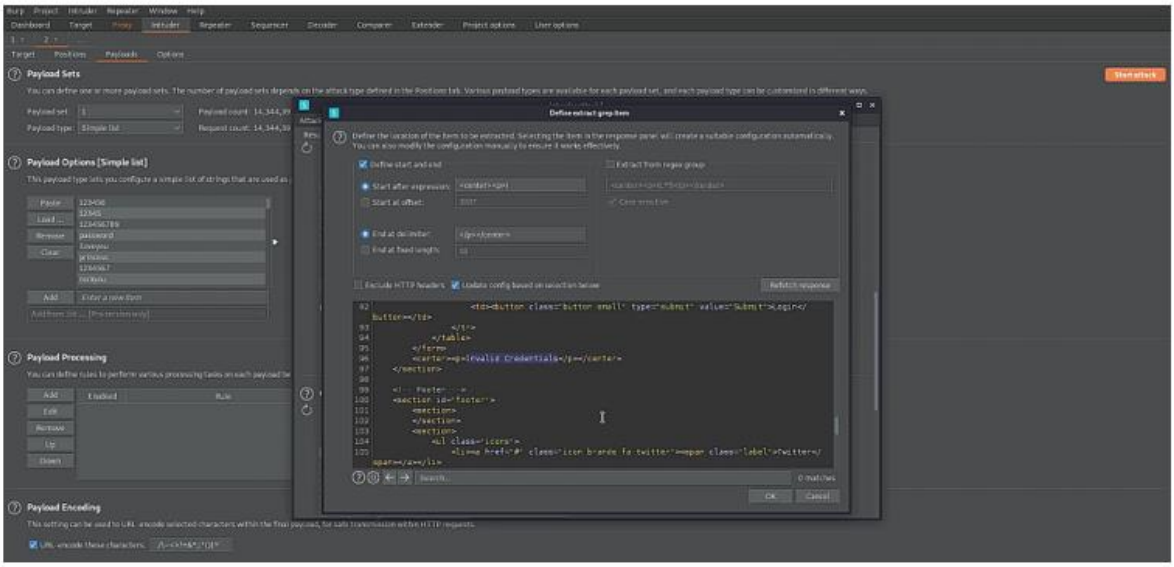


Figure 30 - Grep extracting the "Invalid Credentials" on website to locate the password easily when found

I also added a grep extract of “Invalid Credentials” so that it will be easier to know if the password is detected.

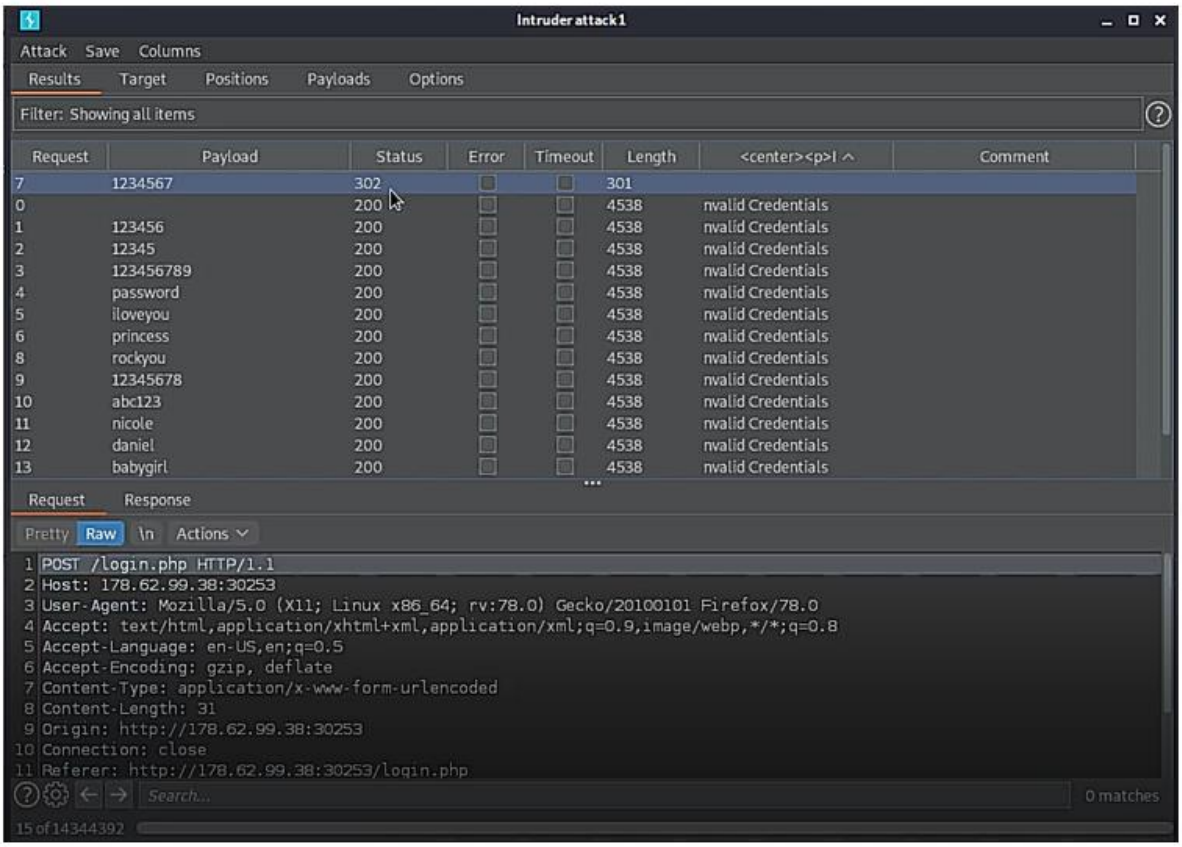


Figure 31 - Burpsuite successfully found the password of admin as 1234567

Only the payload “1235467” did not show Invalid credentials, so 1234567 has to be the password for the user admin.

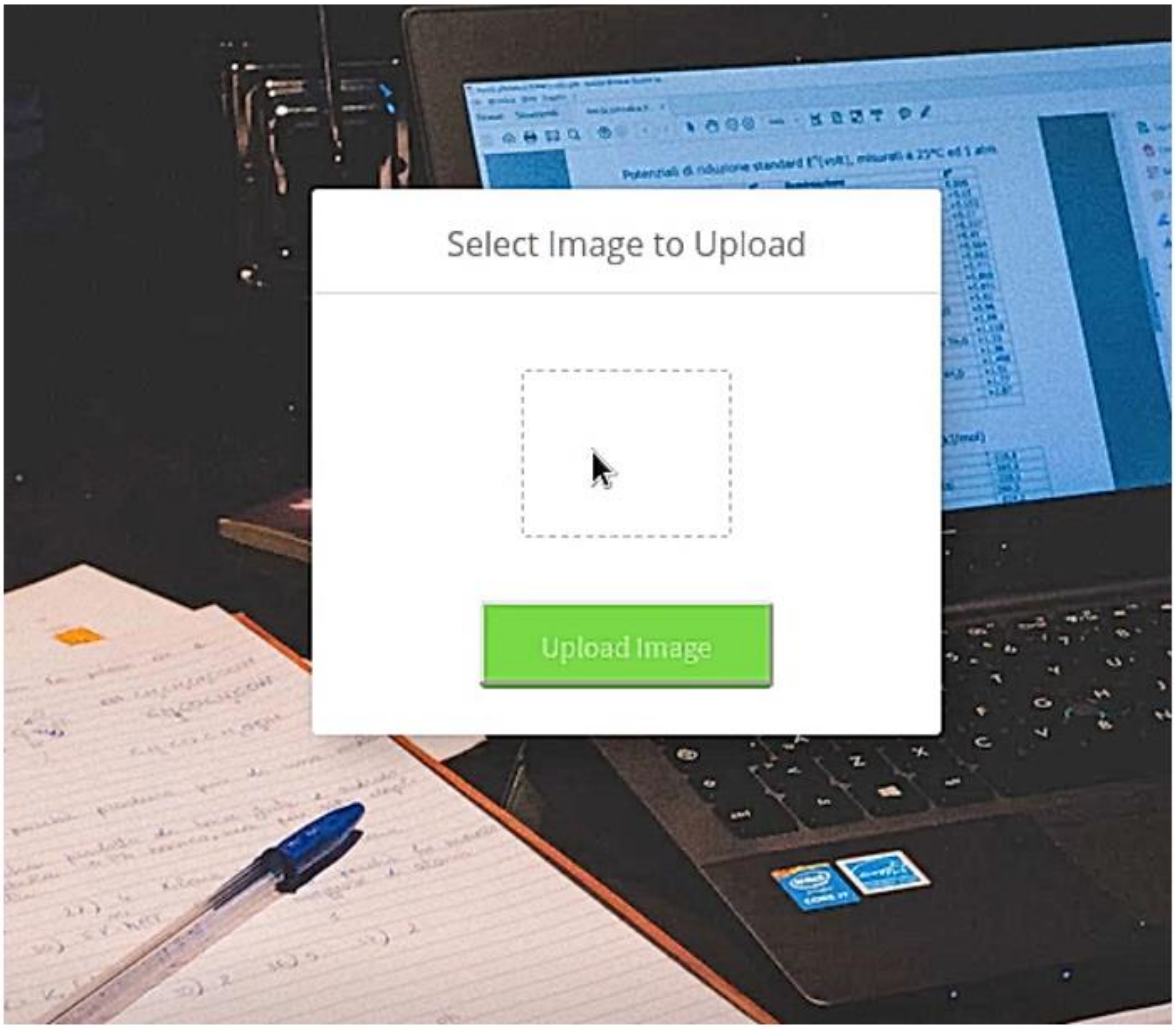


Figure 32 - Webpage to upload images was loaded after logging in

After logging in with those credentials, we can see an image upload option. Thus, we can use Local File Inclusion exploit to get the flag.

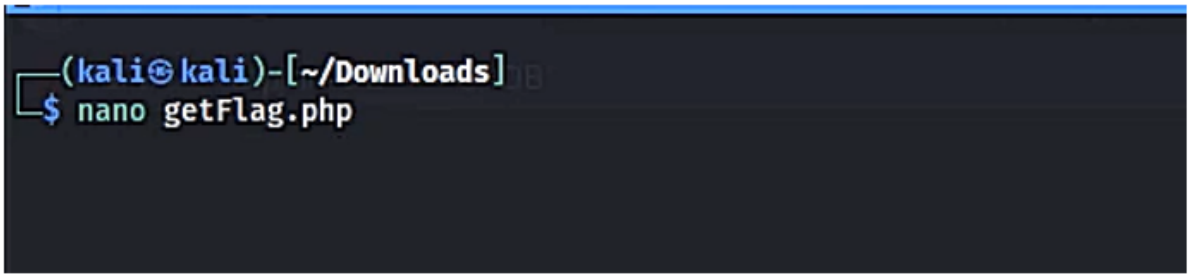


Figure 33 - Creating a getFlag.php script to get the Flag

I made a file “getFlag.php” to get the flag from the root directory.

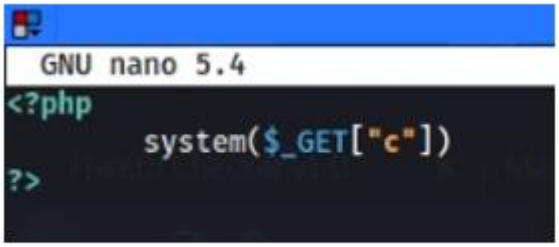


Figure 34 - getFlag.php script

The above code will take the value c as input and processes the command as prints out the output. Once we submit the getFlag.php file with the above code,

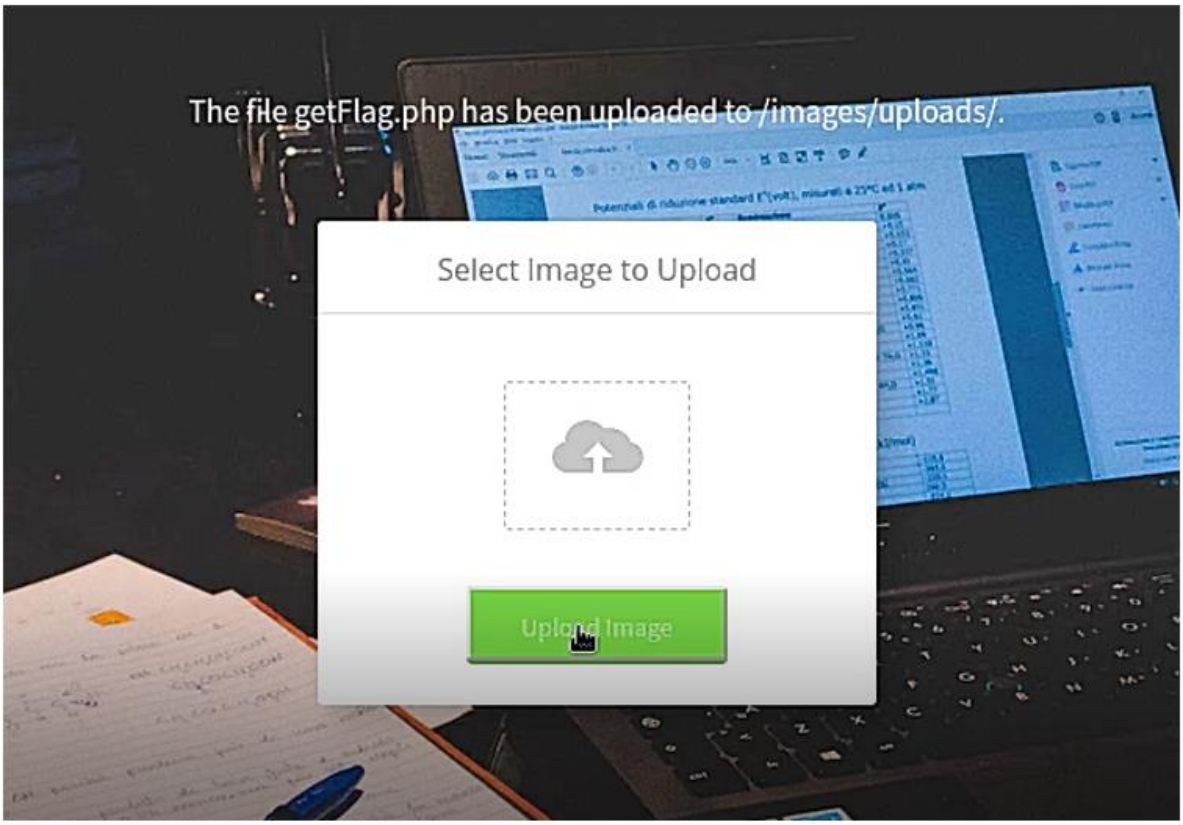


Figure 35 - Successfully uploaded the getFlag.php file

We get a confirmation saying that the script is successfully uploaded to /images/uploads/.

Opening the php page,

Figure 36 - Opening the uploaded file

The page does not show anything as there is no command given. To give a command, we just have to mention the variable c and the command. For example,

…../getFlag.php?c=whoami will give the command whoami to the variable c which then gets processed and return the output with the user.

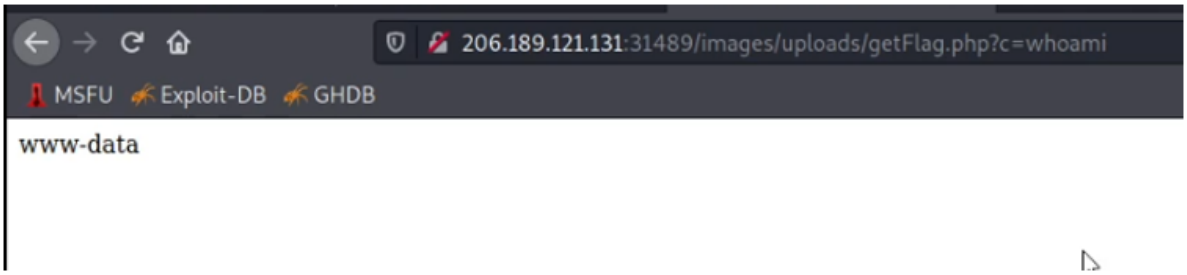


Figure 37 - Giving whoami command to the script which returned www-data

Figure 38 - Giving ls command to the script

Just giving ls command will show all the files that have been uploaded into the server. getFlag.php is the file that I uploaded, and the rest were uploaded by other teammates.

We can list the contents inside the previous directory by using ls ../

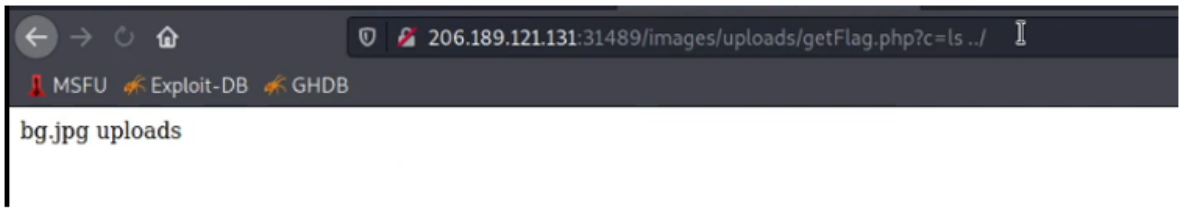


Figure 39 - Giving ls ../ command to the script

The previous folder has a directory uploads where the getFlag.php is uploaded and a bg.jpg image.

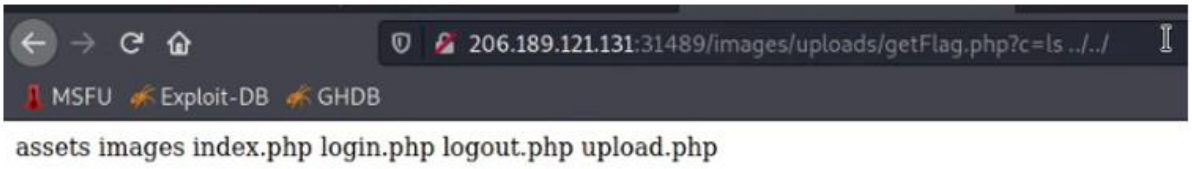


Figure 40 - Giving ls ../../ command to the script

Going a folder back, we can see the files that support the website.

Figure 41 - Giving ls ../../../ command to the script

Going another folder back will show html directory

Figure 42 - Giving ls ../../../../ command to the script

Going another folder back, we can see a bunch of directories. I guessed that the flag is located in the root directory, so I have to go back more

Figure 43 - Giving ls ../../../../../ command to the script

Going back one more folder, we can see a file named flag. So, I decided to print out the flag

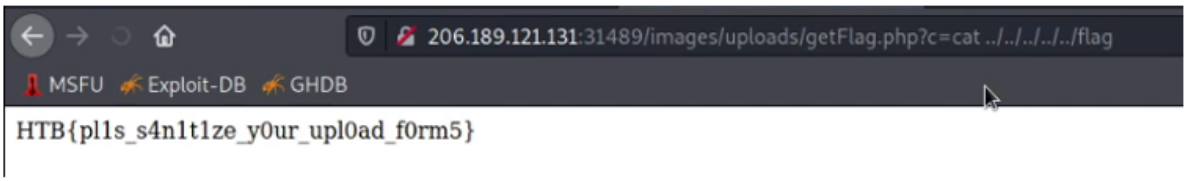


Figure 44 - Getting the flag by giving the command cat ../../../../../flag

$$?c = cat\ ../../../../../flag$$

Will give us the flag "HTB{pl1s_s4n1t1ze_y0ur_upl0ad_f0rm5}"

6. Reflection

CTF – 2 is the best environment to verify and acquire realistic understanding of what we have learned up to now. CTF – 2 being a group assessment allowed us to look very differently at the same challenge. Each one of us could bring different experience and solution to a challenge, and how others thought, or process information was fascinating and knowledgeable. In this CTF, I learnt a great deal from Liam. I am happy with the results and the knowledge I have gained through this CTF and my group.

7. Conclusion

CTF-2 assessment helped me put all my practical knowledge on web penetration testing to test. I learnt my strengths and weaknesses in this area. I am satisfied with my performance. However, I need to improve on the time taken to solve these challenges. I think that with enough practise in the future, I will be able to solve such challenges quickly. I and my team failed to find all the flags within the two-hour window which is disappointing. I will be practicing web pen-testing more on HTB challenges only. I will not use this knowledge on real websites under any circumstance. I learnt the importance of sanitization of the upload forms i.e., not to let any files that are not images or not of a particular file type to be uploaded and to check for the mime of the files. Also having the scripts at server side to check the file uploads will give hackers less chance to gain access to exploit the

website as there no knowledge of the code that is used. Similarly, a prepared statement will reduce the SQL injections.

8. Glossary

- OS – Operating System
- sqlmap - sqlmap is a platform to automatically discover and use SQL injection bugs while doing penetration testing and taking over database servers [1].
- Burpsuite – burpsuite is a tool that is used for penetration testing and finding vulnerabilities on websites [2].

9. References

[1] "sqlmap: automatic SQL injection and database takeover tool", Sqlmap.org, 2021. [Online]. Available: https://sqlmap.org/. [Accessed: 10- May- 2021].

[2]"How to Use Burp Suite Professional for Web Application Security [Part One]", 2021. [Online]. Available: https://deltarisk.com/blog/how-to-use-burp-suite-professional-for-web-application-security-part-one/. [Accessed: 10- May- 2021].

10. Appendix – A

- CommonPasswords.txt – commonPasswords.txt is the file that contains most of the common passwords used. I downloaded this file few months ago and add a password into this when I find one.

Linux/Unix Command Line Cheat Sheet - GettingGeneticsDone.blogspot.com

Command	Description
`pwd`	prints working directory (prints to screen, ie displays the full path, or your location on the filesystem)
`ls`	lists contents of current directory
`ls -l`	lists contents of current directory with extra details
`ls /home/user/*.txt`	lists all files in /home/user ending in .txt
`cd`	change directory to your home directory
`cd ~`	change directory to your home directory
`cd /scratch/user`	change directory to user on scratch
`cd -`	change directory to the last directory you were in before changing to wherever you are now
`mkdir mydir`	makes a directory called mydir
`rmdir mydir`	removes directory called mydir. mydir must be empty
`touch myfile`	creates a file called myfile. updates the timestamp on the file if it already exists, without modifying its contents
`cp myfile myfile2`	copies myfile to myfile2. if myfile2 exists, this will overwrite it!
`rm myfile`	removes file called myfile
`rm -f myfile`	removes myfile without asking you for confirmation. useful if using wildcards to remove files ***
`cp -r dir newdir`	copies the whole directory dir to newdir. -r must be specified to copy directory contents recursively
`rm -rf mydir`	this will delete directory mydir along with all its content without asking you for confirmation! ***
`nano`	opens a text editor. see ribbon at bottom for help. ^x means CTRL-x. this will exit nano
`nano new.txt`	opens nano editing a file called new.txt
`cat new.txt`	displays the contents of new.txt
`more new.txt`	displays the contents of new.txt screen by screen. spacebar to pagedown, q to quit
`head new.txt`	displays first 10 lines of new.txt
`tail new.txt`	displays last 10 lines of new.txt
`tail -f new.txt`	displays the contents of a file as it grows, starting with the last 10 lines. ctrl-c to quit.
`mv myfile newlocdir`	moves myfile into the destination directory newlocdir
`mv myfile newname`	renames file to newname. if a file called newname exists, this will overwrite it!
`mv dir subdir`	moves the directory called dir to the directory called subdir
`mv dir newdirname`	renames directory dir to newdirname
`top`	displays all the processes running on the machine, and shows available resources
`du -h --max-depth=1`	run this in your home directory to see how much space you are using. don't exceed 5GB
`ssh servername`	goes to a different server. this could be queso, brie, or provolone
`grep pattern files`	searches for the pattern in files, and displays lines in those files matching the pattern
`date`	shows the current date and time
`anycommand > myfile`	redirects the output of anycommand writing it to a file called myfile
`date > timestamp`	redirects the output of the date command to a file in the current directory called timestamp
`anycommand >> myfile`	appends the output of anycommand to a file called myfile
`date >> timestamp`	appends the current time and date to a file called timestamp. creates the file if it doesn't exist
`command1 | command2`	"pipes" the output of command1 to command2. the pipe is usually shift-backslash key
`date | grep Tue`	displays any line in the output of the date command that matches the pattern Tue. (is it Tuesday?)
`tar -zxf archive.tgz`	this will extract the contents of the archive called archive.tgz. kind of like unzipping a zipfile. ***
`tar -zcf dir.tgz dir`	this creates a compressed archive called dir.tgz that contains all the files and directory structure of dir
`time anycommand`	runs anycommand, timing how long it takes, and displays that time to the screen after completing anycommand
`man anycommand`	gives you help on anycommand
`cal -y`	free calendar, courtesy unix
`CTRL-c`	kills whatever process you're currently doing
`CTRL-insert`	copies selected text to the windows clipboard (n.b. see above, ctrl-c will kill whatever you're doing)
`SHIFT-insert`	pastes clipboard contents to terminal

*** = use with extreme caution! you can easily delete or overwrite important files with these.

Absolute vs relative paths.

Let's say you are here: /home/turnersd/scripts/. If you wanted to go to /home/turnersd/, you could type: `cd /home/turnersd/`. Or you could use a relative path. `cd ..` (two periods) will take you one directory "up" to the parent directory of the current directory.

`.`	(a single period) means the current directory
`..`	(two periods) means the parent directory
`~`	means your home directory

A few examples

`mv myfile ..`	moves myfile to the parent directory
`cp myfile ../newname`	copies myfile to the parent directory and names the copy newname
`cp /home/turnersd/scripts/bstrap.pl .`	copies bstrap.pl to "." i.e. to dot, or the current directory you're in
`cp myfile ~/subdir/newname`	copies myfile to subdir in your home, naming the copy newname
`more ../../../myfile`	displays screen by screen the content of myfile, which exists 3 directories "up"

Wildcards (use carefully, especially with rm)

`*`	matches any character. example: `ls *.pl` lists any file ending with ".pl" ; `rm dataset*` will remove all files beginning with "dataset"
`[xyz]`	matches any character in the brackets (x, y, or z). example: `cat do[or]m.txt` will display the contents of either doom.txt or dorm.txt