EDU CTF course - Computer Security 2022 Fall

CTF Write up by B10832008 蔡芸軒

Reverse

Challenge Name : Sacred Arts

Points : 11

Category : Reverse

Description :

Tool : IDA freeware, gdb

Approach :

In this challenge we are given an elf 64 file. I used **IDA freeware** to disassemble the file. In below section I found data recognized as a byte is actually an array with length 7.

```
.text:000000000040108B byte_40108B      db 0B3h, 0BAh, 0BEh, 0B8h, 84h
.text:000000000040108B                                          ; DATA XREF: .text:loc_4010CA↓o
.text:0000000000401090                  dq 9E98D18B928D9099h, 0D19290D29C8D9A92h, 8F978FBDD1D0888Bh
.text:0000000000401090                  dq 0CCCDCB92C28C9DC0h, 0CEC2BE8D91D9C7C7h, 0FFFFFFCF82C8CFC7h
.text:00000000004010C0                  db 3 dup(0FFh)
```

So I reformed it and got the array data.

```
.text:000000000040108B qword_40108B     dq 8D909984B8BEBAB3h, 8D9A929E98D18B92h, 0D0888BD19290D29Ch
.text:000000000040108B                                          ; DATA XREF: .text:loc_4010CA↓o
.text:000000000040108B                  dq 8C9DC08F978FBDD1h, 0D9C7C7CCCDCB92C2h, 0C8CFC7CEC2BE8D91h
.text:000000000040108B                  dq 0FFFFFFFFFFFFCF82h
```

Finally, I found the section below which processes the array data. For each element in the array, it does **neg**, which is equal to two's complement, and then **xchg**, which means exchange the first two bytes and the last two bytes.

```
.text:00000000004010D1 loc_4010D1:                              ; CODE XREF: .text:00000000004010EC↓j
.text:00000000004010D1                  lea     rdx, ds:0FFFFFFFFFFFFFFF8h[rcx*8]
.text:00000000004010D9                  mov     rax, [rsp+rdx]
.text:00000000004010DD                  neg     rax
.text:00000000004010E0                  xchg    al, ah
.text:00000000004010E2                  cmp     rax, [rbx+rdx]
.text:00000000004010E6                  jnz     loc_401035
.text:00000000004010EC                  loop    loc_4010D1
.text:00000000004010EE                  jmp     short loc_4010FD
```

Code :

```python
qword = [0x8D909984B8BEBAB3, 0x8D9A929E98D18B92, 0x0D0888BD19290D29C, 0x8C9DC08F978FBDD1,
         0x0D9C7C7CCCDCB92C2, 0x0C8CFC7CEC2BE8D91, 0x0FFFFFFFFFFFFCF82]

for val in qword:
    exch_val = ((val >> 16) << 16) + (val >> 8) % (1 << 8) + val % (1 << 8) * (1 << 8)
    flag = (1 << 64) - exch_val
    print(flag.to_bytes(math.ceil(math.log(flag, 256)), 'big').decode("utf-8")[::-1], end='')
```

Result :

```
FLAG{forum.gamer.com.tw/C.php?bsn=42388&snA=18071}
```

Challenge Name : Kekkou

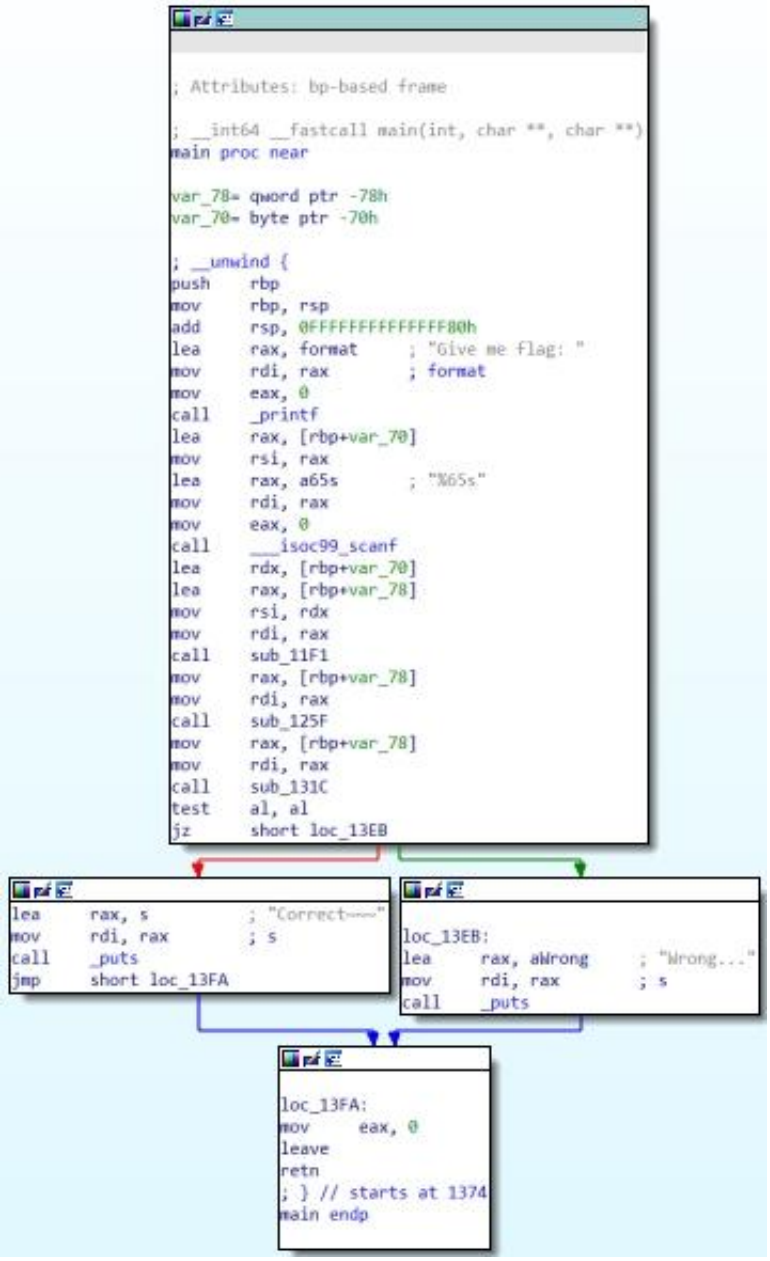
Points : 11

Category : Reverse

Description :

Approach :

In this challenge we are given an x86 64 file., I used **IDA freeware** to disassemble and decompile the file.



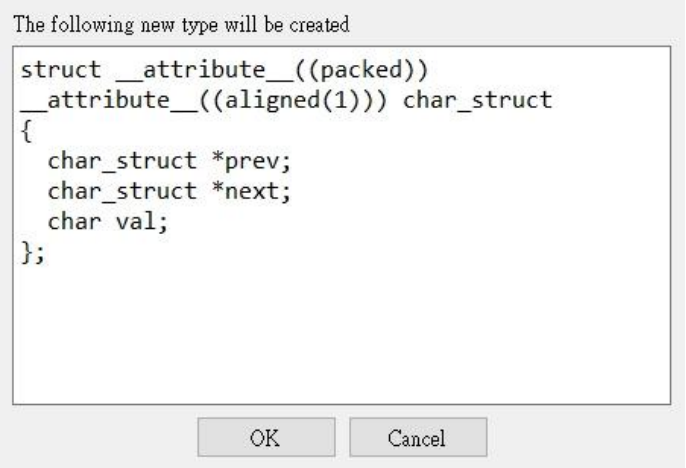
As IDA has recognized the functions correctly, I firstly looked into each function and modified some code to make it more understandable.

In the main function, I found that this is a program that includes inputting a string and verifying if it is the FLAG.

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  __int64 v4; // [rsp+8h] [rbp-78h] BYREF
  char v5[112]; // [rsp+10h] [rbp-70h] BYREF

  printf("Give me flag: ");
  __isoc99_scanf("%65s", v5);
  sub_11F1(&v4, v5);
  sub_125F(v4);
  if ( (unsigned __int8)sub_131C(v4) )
    puts("Correct~~~");
  else
    puts("Wrong...");
  return 0LL;
}
```

In function **sub_1169,** it is hard to understand what it does at first. However, I found that the structure is similar to **struct** data type, so I did "**reset pointer type"** and "**create new struct type"** to make it clearer. Also, from other sections, I observed that this is a linked list structure. Because of this, I created a new struct named char_struct, which contains a character value and two pointers points to previous and next structure objects in a linked list.




```
_QWORD *__fastcall sub_1169(char a1)
{
  _QWORD *result; // rax

  result = malloc(0x18uLL);
  *result = result;
  result[1] = result;
  *((_BYTE *)result + 16) = a1;
  return result;
}
```

```
char_struct *__fastcall char_to_struct(char a1)
{
  char_struct *result; // rax

  result = (char_struct *)malloc(0x18uLL);
  result->prev = result;
  result->next = result;
  result->val = a1;
  return result;
}
```

In the function **sub_11AC**, I redeclare type of some variables :
a1 from int_64 to char_struct*
a2 from QWORD to char_struct*
After this, it is obvious that what function **sub_11AC** does is connect a2 to a1, so I renamed it to **connect_a_struct**

```
1  __int64 __fastcall sub_11AC(__int64 a1, _QWORD *a2)
2  {
3    __int64 result; // rax
4
5    **(_QWORD **)(a1 + 8) = a2;
6    *a2 = a1;
7    a2[1] = *(_QWORD *)(a1 + 8);
8    result = a1;
9    *(_QWORD *)(a1 + 8) = a2;
10   return result;
11 }
```

```
1  char_struct *__fastcall connect_a_struct(char_struct *a1, char_struct *a2)
2  {
3    char_struct *result; // rax
4
5    a1->next->prev = a2;
6    a2->prev = a1;
7    a2->next = a1->next;
8    result = a1;
9    a1->next = a2;
10   return result;
11 }
```

In function **sub_11F1**, I redeclare type of some variables :
a2 from int64 to char* : since v5 is a string
v2 from int64 to char_struct* : since function sub_1169 return a char_struct*
a1 from int64* to char_struct** : since v2 is assigned to *a1.
In function **sub_11F1**, for each character in the input string, it generates a struct for each character and links them together, so I renamed it to **string_to_linked_list**.

```
1  void __fastcall sub_11F1(__int64 *a1, __int64 a2)
2  {
3    __int64 v2; // [rsp+10h] [rbp-10h]
4    int i; // [rsp+1Ch] [rbp-4h]
5
6    for ( i = 0; i <= 64; ++i )
7    {
8      v2 = sub_1169(*(unsigned __int8 *)(i + a2));
9      if ( i )
10       sub_11AC(*a1, v2);
11     else
12       *a1 = v2;
13   }
14 }
```

```
1  void __fastcall string_to_linked_list(char_struct **a1, char *a2)
2  {
3    char_struct *v2; // [rsp+10h] [rbp-10h]
4    int i; // [rsp+1Ch] [rbp-4h]
5
6    for ( i = 0; i <= 64; ++i )
7    {
8      v2 = sub_1169(a2[i]);
9      if ( i )
10       connect_a_struct(*a1, v2);
11     else
12       *a1 = v2;
13   }
14 }
```

Finally, in function **sub_125F**, it XOR an array of data to the FLAG. Because of the reflexivity of XOR, we can get FLAG by **do_xor(enc_flag, array data)**

```
__int64 __fastcall sub_125F(__int64 *a1)
{
  __int64 v1; // rdx
  __int64 result; // rax
  __int16 v4; // [rsp+Dh] [rbp-Bh]
  int j; // [rsp+10h] [rbp-8h]
  int i; // [rsp+14h] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = byte_4041[3 * i];
    if ( !(_BYTE)result )
      break;
    v1 = 3LL * i;
    v4 = *(_WORD *)((char *)&unk_4040 + v1);
    LOBYTE(v4) = v4 & 1;
    for ( j = 0; j < HIBYTE(v4); ++j )
    {
      if ( (_BYTE)v4 )
        a1 = (__int64 *)*a1;
      else
        a1 = (__int64 *)a1[1];
    }
    *((_BYTE *)a1 + 16) ^= *((_BYTE *)&unk_4040 + v1 + 2);
  }
  return result;
}
```

```
__int64 __fastcall do_xor(char_struct *a1)
{
  __int64 v1; // rdx
  __int64 result; // rax
  __int16 v4; // [rsp+Dh] [rbp-Bh]
  int j; // [rsp+10h] [rbp-8h]
  int i; // [rsp+14h] [rbp-4h]

  for ( i = 0; ; ++i )
  {
    result = byte_4041[3 * i];
    if ( !(_BYTE)result )
      break;
    v1 = 3LL * i;
    v4 = *(_WORD *)((char *)&off_4040 + v1);
    LOBYTE(v4) = v4 & 1;
    for ( j = 0; j < HIBYTE(v4); ++j )
    {
      if ( (_BYTE)v4 )
        a1 = a1->prev;
      else
        a1 = a1->next;
    }
    a1->val ^= *((_BYTE *)&off_4040 + v1 + 2);
  }
  return result;
}
```

Code :

```python
i, index = 0, 0
while True:
    if 3 * i + 2 >= len(xor_list) or not xor_list[3 * i]: break
    lobyte = xor_list[3 * i] & 1
    for j in range(xor_list[3 * i + 1]):
        if lobyte: index = (index + 1) % 65
        else: index = (index + 64) % 65
    enc_flag[index] ^= xor_list[3 * i + 2]
    i += 1

for c in enc_flag:
    print(chr(c), end='')
```

Result :

```
FLAG{kekkou_muri_jya_nai?nai_ai_kazoe_te_cyotto_kurushi_ku_na_ru}
```

Challenge Name : Why

Points : 11

Category : Reverse

Description :

Approach :

In this challenge we are given an x86 64 file., I used **IDA freeware** to disassemble and decompile the file.

This is a pretty simple challenge. The only thing is that the main function doesn't call the check function **sub_11f8**.

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int i; // [rsp+Ch] [rbp-4h]
4
5    printf("Give me flag: ");
6    __isoc99_scanf("%25s", buf);
7    for ( i = 0; i <= 24; ++i )
8    {
9      if ( buf[i] - 10 != enc_flag[i] )
10       return 0;
11   }
12   pass = 1;
13   return 0;
14 }
```

```
.data:0000000000004030 enc_flag        db 50h, 56h, 4Bh, 51h, 85h, 73h, 78h, 73h, 7Eh, 69h, 70h
.data:0000000000004030                                                 ; DATA XREF: main+5F↑o
.data:0000000000004030                 db 73h, 78h, 73h, 69h, 77h, 7Ah, 7Ch, 79h, 7Eh, 6Fh, 6Dh
.data:0000000000004030                 db 7Eh, 2Bh, 87h
.data:0000000000004030 _data           ends
```

```
1  int sub_11f8()
2  {
3    if ( pass )
4      return puts("Correct :)");
5    else
6      return puts("Wrong :(");
7  }
```

Finally I found the reference and it is in the fini section.

```
.fini_array:0000000000003DD0 _fini_array     segment qword public 'DATA' use64
.fini_array:0000000000003DD0                 assume cs:_fini_array
.fini_array:0000000000003DD0                 ;org 3DD0h
.fini_array:0000000000003DD0 __do_global_dtors_aux_fini_array_entry dq offset __do_global_dtors_aux
.fini_array:0000000000003DD0                                         ; DATA XREF: __libc_csu_init+19↑o
.fini_array:0000000000003DD0                                         ; Alternative name is '__init_array_end'
.fini_array:0000000000003DD8                 dq offset _sub_11f8
.fini_array:0000000000003DD8 _fini_array     ends
```

Code :

```python
enc_flag = [0x50, 0x56, 0x4B, 0x51, 0x85, 0x73, 0x78, 0x73, 0x7E, 0x69, 0x70, 0x73,
    0x78, 0x73, 0x69, 0x77, 0x7A, 0x7C, 0x79, 0x7E, 0x6F, 0x6D, 0x7E, 0x2B, 0x87]

for c in enc_flag:
    print(chr(c - 10), end='')
```

Result :

```
FLAG{init_fini_mprotect!}
```

✌Challenge Name : trace

Points : 100

Category : Reverse

Description : Can you trace me?

Approach :

Code :

Result :

✌Challenge Name : pwn_myself

Points : 200

Category : Reverse

Description :

Approach :

Code :

Result :

✌Challenge Name : AMessageBox

Points : 11

Category : Reverse

Description :

Approach :

Code :

Result :

✌Challenge Name : IAT Hook

Points : 11

Category : Reverse

Description :

Approach :

Code :

Result :

✌Challenge Name : GetProcAddress

Points : 11

Category : Reverse

Description :

Approach :

Code :

Result :

Challenge Name : ooxx

Points : 150

Category : Reverse

Description : Win this game to get flag!

Approach :

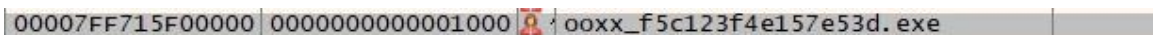
In this challenge we are given an exe file. I used **IDA freeware** together with **x64dbg** to disassemble, decompile, and trace the file.

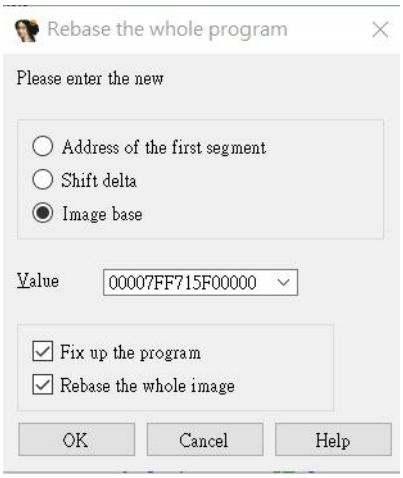
I executed this file and found that this is a Tic-tac-toe game, while we can never win by usual ways. However, as the description says, we have to win the game, so we have to do some tricks to it.

First, I looked at the structure of the code in IDA. Seems that *dword_7FF715F0D338* stores the chess board data and 1 represents O, 2 represents X.

```
1  __int64 __fastcall sub_7FF715F01C50(__int64 a1, __int64 a2, int a3, int a4)
2  {
3    __int64 result; // rax
4    int v5; // [rsp+38h] [rbp-10h]
5    int v6; // [rsp+3Ch] [rbp-Ch]
6
7    v6 = a3 / 200 + 3 * (a4 / 200);
8    result = v6;
9    if ( !dword_7FF715F0D338[v6] )
10   {
11     dword_7FF715F0D338[v6] = 1;
12     sub_7FF715F047C0(a1, 200 * (a3 / 200) + 20, 200 * (a4 / 200) + 20, 160, 160);
13     if ( (unsigned int)sub_7FF715F017F0() )
14       return sub_7FF715F049B0(a1, sub_7FF715F01C50);
15     v5 = sub_7FF715F01C10();
16     dword_7FF715F0D338[v5] = 2;
17     draw_line(a1, 200 * (v5 % 3) + 20, 200 * (v5 / 3) + 20, 160, 160);
18     draw_line(a1, 200 * (v5 % 3) + 180, 200 * (v5 / 3) + 20, -160, 160);
19     result = sub_7FF715F017F0();
20     if ( (_DWORD)result )
21       return sub_7FF715F049B0(a1, sub_7FF715F01C50);
22   }
23   return result;
24 }
```

Next, I use x64dbg to see what happened when executing and want to modify some values in memory to get the FLAG. To begin, I checked the memory address of execution in x64dbg and rebase memory to it in IDA.

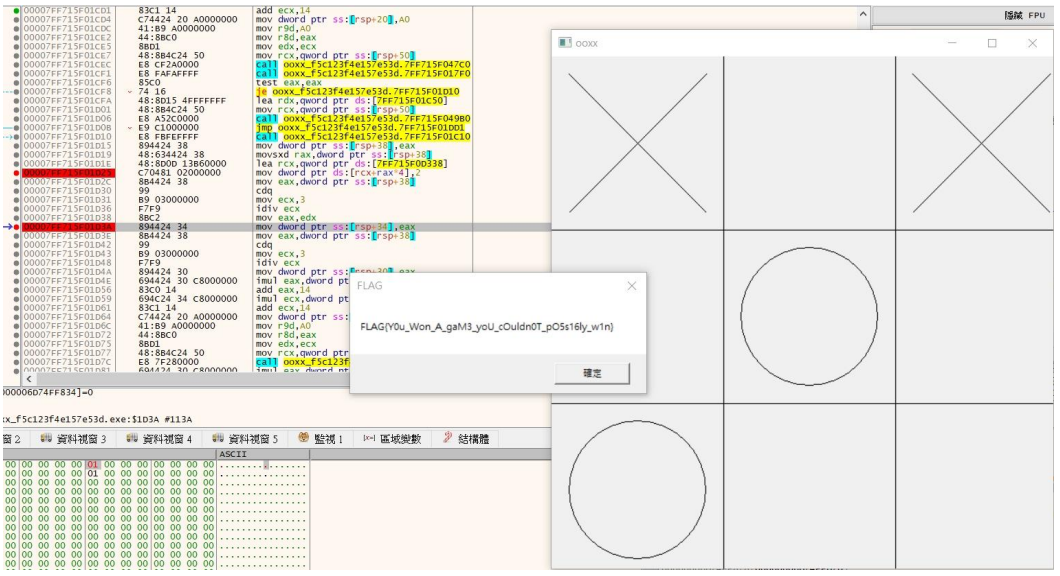




After setting two breaking points, I found that after the program drawing X, the below address became 02. With this observation, I modified them to 01, which represents O, and finally got the FLAG.

Result :

Because I modified the data after Xs being drew, the graph displayed on screen didn't change.



Challenge Name : trojan

Points : 150

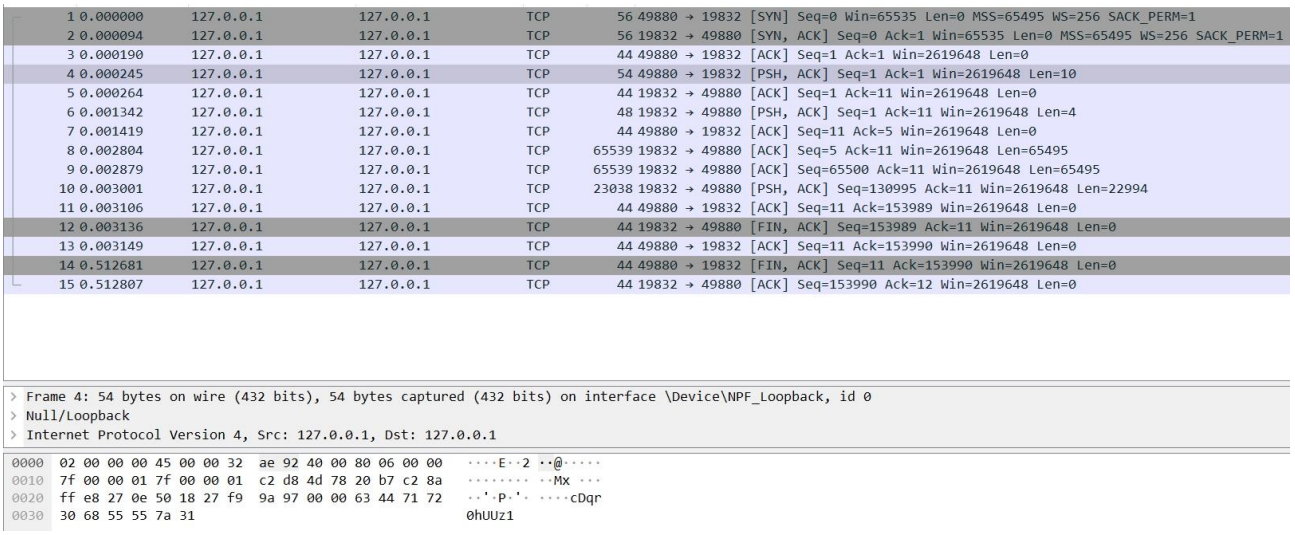
Category : Reverse

Description :

Approach :

In this challenge we are given an exe file and a wireshark file. I used **IDA freeware** and **Wireshark** to solve this. I studied the code in IDA and realized that the executable exe is used to write the wireshark file (like output).

In the wireshark file, I saw there's something abnormal - a string with length 10.

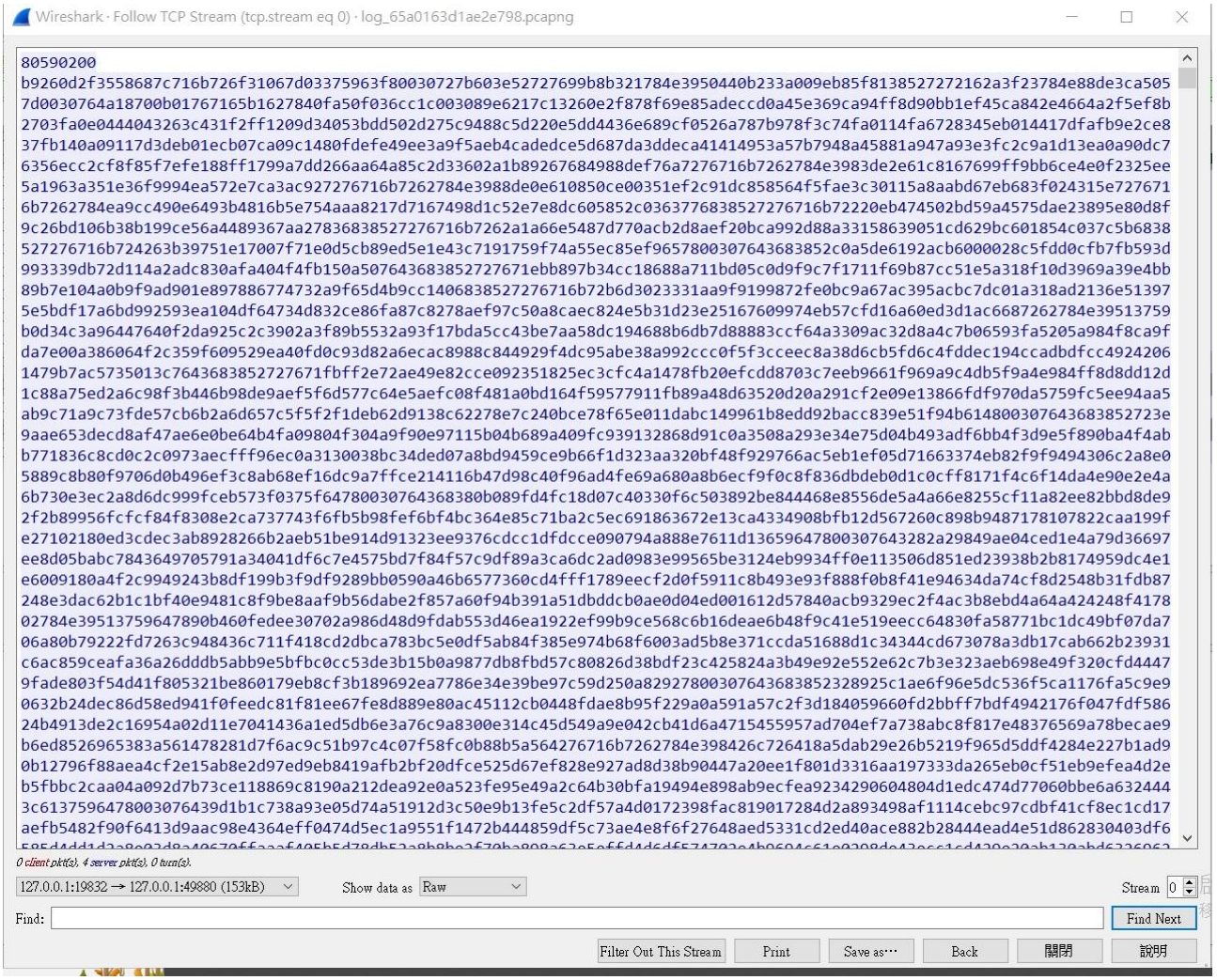


1	0.000000	127.0.0.1	127.0.0.1	TCP	56	49880 → 19832 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000094	127.0.0.1	127.0.0.1	TCP	56	19832 → 49880 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000190	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	0.000245	127.0.0.1	127.0.0.1	TCP	54	49880 → 19832 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=10
5	0.000264	127.0.0.1	127.0.0.1	TCP	44	19832 → 49880 [ACK] Seq=1 Ack=11 Win=2619648 Len=0
6	0.001342	127.0.0.1	127.0.0.1	TCP	48	19832 → 49880 [PSH, ACK] Seq=1 Ack=11 Win=2619648 Len=4
7	0.001419	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [ACK] Seq=11 Ack=5 Win=2619648 Len=0
8	0.002804	127.0.0.1	127.0.0.1	TCP	65539	19832 → 49880 [ACK] Seq=5 Ack=11 Win=2619648 Len=65495
9	0.002879	127.0.0.1	127.0.0.1	TCP	65539	19832 → 49880 [ACK] Seq=65500 Ack=11 Win=2619648 Len=65495
10	0.003001	127.0.0.1	127.0.0.1	TCP	23038	19832 → 49880 [PSH, ACK] Seq=130995 Ack=11 Win=2619648 Len=22994
11	0.003106	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [ACK] Seq=11 Ack=153989 Win=2619648 Len=0
12	0.003136	127.0.0.1	127.0.0.1	TCP	44	19832 → 49880 [FIN, ACK] Seq=153989 Ack=11 Win=2619648 Len=0
13	0.003149	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [ACK] Seq=11 Ack=153990 Win=2619648 Len=0
14	0.512681	127.0.0.1	127.0.0.1	TCP	44	49880 → 19832 [FIN, ACK] Seq=11 Ack=153990 Win=2619648 Len=0
15	0.512807	127.0.0.1	127.0.0.1	TCP	44	19832 → 49880 [ACK] Seq=153990 Ack=12 Win=2619648 Len=0

```
> Frame 4: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
```

```
0000  02 00 00 00 45 00 00 32  ae 92 40 00 80 06 00 00   ····E··2 ··@·····
0010  7f 00 00 01 7f 00 00 01  c2 d8 4d 78 20 b7 c2 8a   ········ ··Mx ···
0020  ff e8 27 0e 50 18 27 f9  9a 97 00 00 63 44 71 72   ··'·P·'· ····cDqr
0030  30 68 55 55 7a 31                                 0hUUz1
```

Also, I checked the TCP stream, showing data as raw



Using this, I located the position in IDA of the above string and found that it does below operations to it. I wrote a small program to get the result (as shown in the code area below).

```
1  __int64 __fastcall sub_1400014B0(__int64 a1, unsigned int a2
2  {
3    __int64 result; // rax
4    int i; // [rsp+0h] [rbp-48h]
5    char v4[24]; // [rsp+10h] [rbp-38h] BYREF
6
7    strcpy(v4, "0vCh8RrvqkrbxN9Q7Ydx");
8    for ( i = 0; ; ++i )
9    {
10     result = a2;
11     if ( i >= (int)a2 )
12       break;
13     *(_BYTE *)(a1 + i) ^= v4[i % 0x15ui64];
14   }
15   return result;
16 }
```

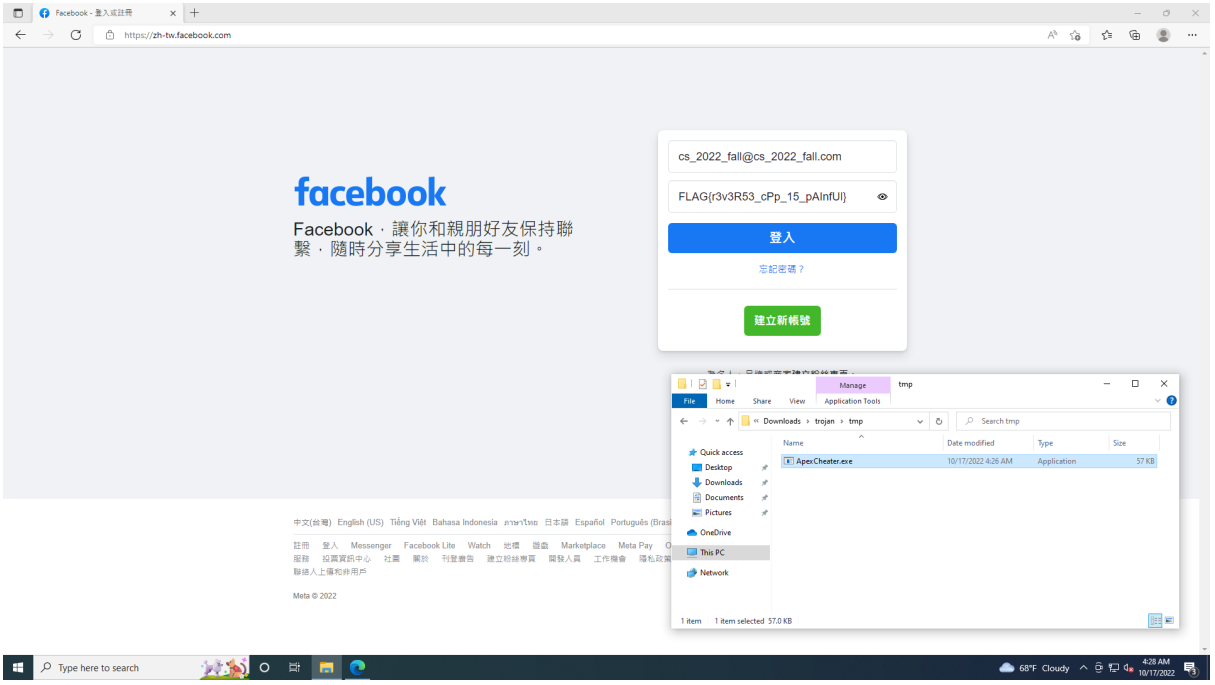
After solving it, I couldn't find any FLAG{xxx} format in the FLAG binary file. However, when I opened it by Hxd, I found that it contained the PNG magic header!! Then I renamed the format of the file to .png and got the FLAG.

Code :

```python
key = "0vCh8RrvqkrbxN9Q7Ydx\0"

with open("test") as fr:
    file = fr.read()
    enc_flag = bytearray(file)
    for i in range(len(enc_flag)):
        enc_flag[i] = enc_flag[i] ^ ord(key[i % len(key)])
    with open('flag') as fw:
        fw.write(enc_flag)
```

Result :


facebook
Facebook，讓你和親朋好友保持聯繫，隨時分享生活中的每一刻。
cs_2022_fall@cs_2022_fall.com
FLAG{r3v3R53_cPp_15_pAlnfUl}
登入
忘記密碼？
建立新帳號
tmp
ApexCheater.exe
10/17/2022 4:26 AM
Application
57 KB
1 item
1 item selected 57.0 KB
Type here to search
68°F Cloudy
4:28 AM
10/17/2022

Challenge Name : dropper

Points : 300

Category : Reverse

Description :

Approach :

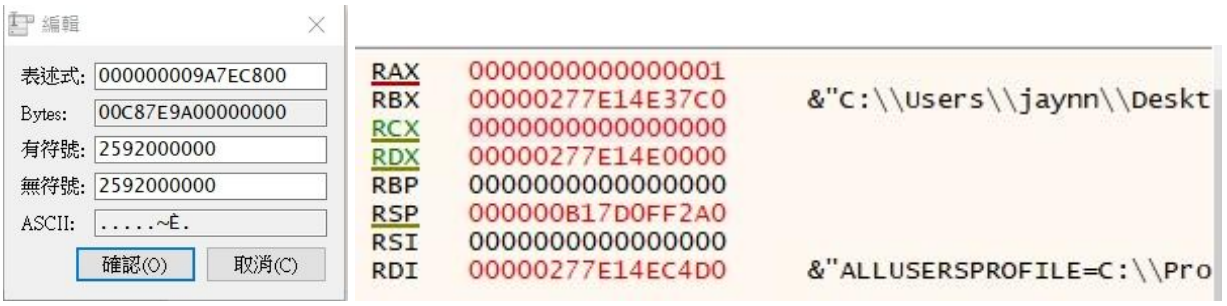
In this challenge, I used UPX (ultimate packer for executables) to unpack the file and IDA freeware to disassemble and decompile it and x64dgb to trace the memory.

Looking into IDA, I found that this executable is packed.

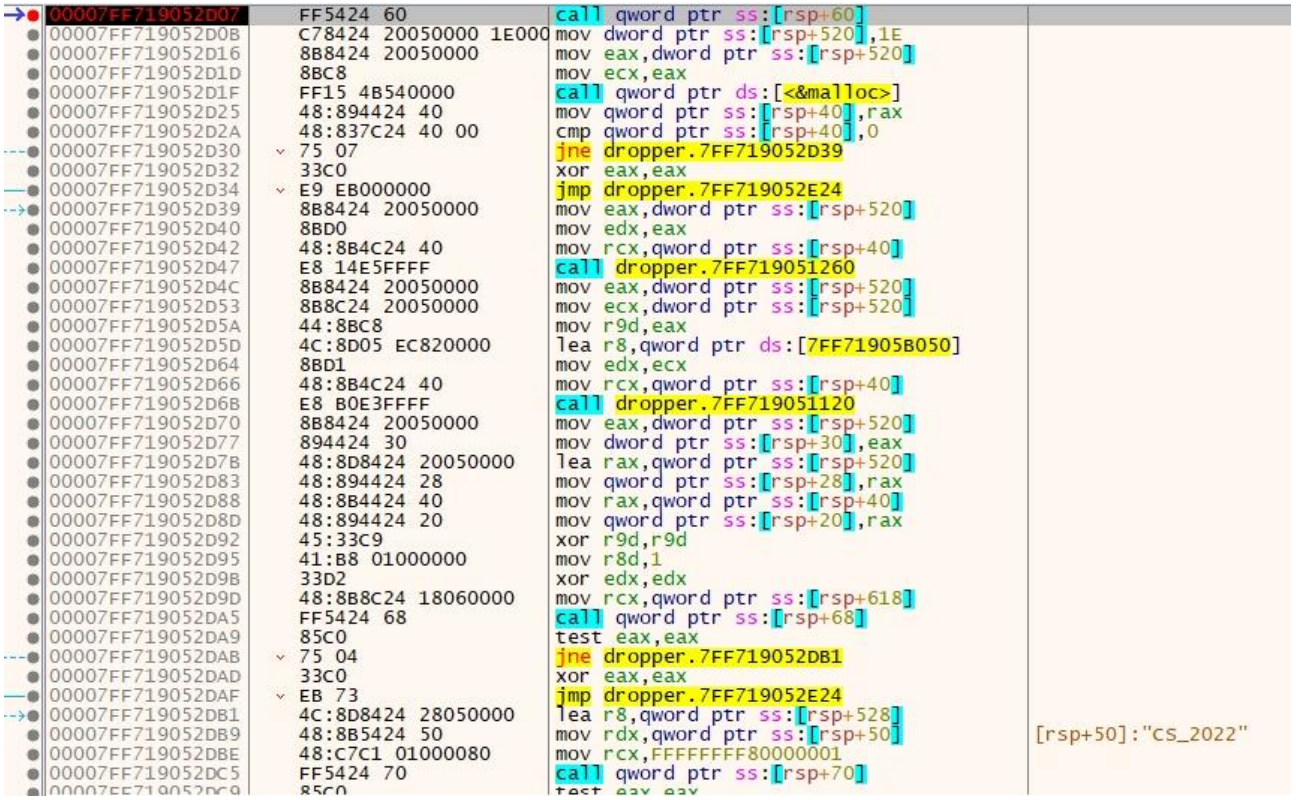
I used UPX to unpack it.

Using x64dgb to execute, I found that it stuck in one line. Seems that it has called a sleep function.

By modifying the value in rax to zero, it can keep going.



I got the flag. What a surprise!!



Result :