Computer Security Homework 0x02

Real name: 涂世昱

Nick name: R07922115

Student ID: R07922115

IDAmudamudamuda

Introduction

起初先將程式執行起來, 首先要輸入一個seed, 然後再輸入flag, 看似只要flag輸入正確它就會告訴我這是正確的flag。用Ida Pro打開之後一開始也不知道main function在何處, 之後想盡辦法看到了main function之後觀察一下, 它呼叫了兩個function, func1, func2, func1跟seed有關, func2則跟flag有關, 而func2做了一個function calling, funcx, 但從Ida Pro似乎看不出來這funcx在做什麼, 因為它用了function pointer, 用變數來定義funcx的區段在何處, 而這個變數似乎又是在執行的過程當中決定的, 所以還是要用動態分析的工具來觀察它。

Part 1. main

我用之前助教提出的方法, main function應該是被夾在`call <JMP.&__p___argc>`和`call <JMP.&_cexit>`之間, 於是就進入了main function, 在Ida Pro則是對應到`4012f0`的位置, 就可以知道程式的流程大概是怎樣了, 在printf一些訊息之後, 首先, 先將seed輸入傳入seed變數當中, 將seed當作參數傳入func1, 離開func1後在一個if條件當中又呼叫了func2, 如果它return為true則輸出`OH yeah, you got the flag\n\n\n`, 反之則輸出`That's not right...`, 就結束了。

於是就再進一步分析func1及func2

Part 2. func1(seed)

```
if(){
    if(){
        if(){
            if(){
                for(){                  // 第一個 for loop
                    ...
                }
                ...
                for(){                  // 第二個 for loop
                    ...
                }
                for(){                  // 第三個 for loop
                    ...
                }
            }

        }
    }
}
```

這邊有4個if statement, 但因為每次用debugger觀察它都進得去所以就不去探討它, 在裡面則是由三個大的for loop所組成的, 在Ida Pro中可發現第三個for loop裡有一條指令會用到seed, `*(_BYTE *)(l + v8 + m) += seed;`, 其中`l`, `m`為for loop用的index, 而`v8`再往上看可發現`v8 = *(_DWORD *)(i + 12) + dword_40448C;`, 於是用x32dbg去尋找`v8`到底是什麼, 發現它指到一個記憶體位置`104000 (最左邊兩個數字不一定每次都一樣)`, 當在執行第二個for loop時, 它會從`104000`的位置開始尋找值為`0x0F`的byte, 然後從那個byte開始一直到它後面的31個bytes都`+=`它32bytes之後的位置的byte, 如以下所示,

第二個 **for loop** 之前

```
00104000  31 3A AD 21 CE C5 52 DE FF FF FF FF 01 00 00 00  1:.!ÎÅRÞÿÿÿÿ....
00104010  2F 00 00 00 01 00 00 00 0F 09 31 0C F8 14 ED 36  /.........1.ø.í6
00104020  FA EE E2 ED 36 1E 36 0C 35 3C 36 3C ED 30 36 EF  úîâí6.6.5<6<í06ï
00104030  31 E8 EE EF E9 E2 EC C6 00 00 D1 00 00 E6 43 BA  1èîïéâìÆ..Ñ..æC°
00104040  28 34 18 43 BA 04 EC EE FB B4 EC E6 48 BD AE 07  (4.C°.ìîû´ìæH½®.
00104050  C9 FC FE 46 F8 40 36 00 45 7B DC 41 B7 35 EC F0  ÉüþFø@6.E{ÜA·5ìð
00104060  F0 F0 F0 DB F9 7B 35 EC 73 B0 F1 79 35 EC 7B 3D  ðððÛù{5ìs°ñy5ì{=
00104070  FC F3 3D EC FF AE 01 75 C2 64 15 7B 35 F8 F3 35  üó=ìÿ®.uÂd.{5øó5
00104080  EC FF AE F8 73 B1 13 73 E1 56 FF AE C1 7B 35 FC  ìÿ®øs±.sáVÿ®Á{5ü
00104090  F3 35 EC FF AE F8 2B C1 64 F4 23 B0 DB F7 DB B5  ó5ìÿ®ø+ÁdôÛ÷Ûµ
001040A0  A8 F1 F0 F0 F0 7B D5 4D B3 00 00 00 00 00 00 00  ¨ñððð{ÕM³.......
```

第二個 **for loop** 之後

```
00104000  31 3A AD 21 CE C5 52 DE FF FF FF FF 01 00 00 00  1:.!ÎÅRÞÿÿÿÿ....
00104010  2F 00 00 00 01 00 00 00 0F 09 02 0C F8 FA 30 F0  /...........øú0ð
00104020  22 22 FA 30 F0 22 22 FA 30 F0 22 22 35 ED E4 F6  ""ú0ð""ú0ð""5íäö
00104030  FA E4 EC 35 E1 22 22 C6 00 00 D1 00 00 E6 43 BA  úäì5á""Æ..Ñ..æC°
00104040  28 34 18 43 BA 04 EC EE FB B4 EC E6 48 BD AE 07  (4.C°.ìîû´ìæH½®.
00104050  C9 FC FE 46 F8 40 36 00 45 7B DC 41 B7 35 EC F0  ÉüþFø@6.E{ÜA·5ìð
00104060  F0 F0 F0 DB F9 7B 35 EC 73 B0 F1 79 35 EC 7B 3D  ðððÛù{5ìs°ñy5ì{=
00104070  FC F3 3D EC FF AE 01 75 C2 64 15 7B 35 F8 F3 35  üó=ìÿ®.uÂd.{5øó5
00104080  EC FF AE F8 73 B1 13 73 E1 56 FF AE C1 7B 35 FC  ìÿ®øs±.sáVÿ®Á{5ü
00104090  F3 35 EC FF AE F8 2B C1 64 F4 23 B0 DB F7 DB B5  ó5ìÿ®ø+ÁdôÛ÷Ûµ
001040A0  A8 F1 F0 F0 F0 7B D5 4D B3 00 00 00 00 00 00 00  ¨ñððð{ÕM³.......
```

再接著往下看到第三個 for loop, 它會從`104039`開始, 尋找值為`0x45`的byte, 從那個byte開始, 每個byte都`+= seed`, 直到碰到`0x00`,

```
00104000  31 3A AD 21 CE C5 52 DE FF FF FF FF 01 00 00 00  1:.!ÎÅRÞÿÿÿÿ....
00104010  2F 00 00 00 01 00 00 00 0F 09 02 0C F8 FA 30 F0  /...........øú0ð
00104020  22 22 FA 30 F0 22 22 FA 30 F0 22 22 35 ED E4 F6  ""ú0ð""ú0ð""5íäö
00104030  FA E4 EC 35 E1 22 22 C6 00 00 D1 00 00 E6 43 BA  úäì5á""Æ..Ñ..æC°
00104040  28 34 18 43 BA 04 EC EE FB B4 EC E6 48 BD AE 07  (4.C°.ìîû´ìæH½®.
00104050  C9 FC FE 46 F8 40 36 00 55 8B EC 51 C7 45 FC 00  ÉüþFø@6.U.ìQÇEü.
00104060  00 00 00 EB 09 8B 45 FC 83 C0 01 89 45 FC 8B 4D  ...ë..EüÀ..EüM
00104070  0C 03 4D FC 0F BE 11 85 D2 74 25 8B 45 08 03 45  ..Mü.¾..Òt%.E..E
00104080  FC 0F BE 08 83 C1 23 83 F1 66 0F BE D1 8B 45 0C  ü.¾..Á#.ñf.¾Ñ.E.
00104090  03 45 FC 0F BE 08 3B D1 74 04 33 C0 EB 07 EB C5  .Eü.¾.;Ñt.3Àë.ëÅ
001040A0  B8 01 00 00 00 8B E5 5D C3 00 00 00 00 00 00 00  ¸.....å]Ã.......
```

就結束了func1。先把`104000`這個位置先記著, 之後有可能會用到。

Part 2. func2(flag)

檢查flag字數是否為32, 是的話才會繼續執行, 在最後一行`return ((int (__cdecl *)(const char *, void *))v2)(flag, &unk_404018);`呼叫了一個function pointer, `v2`, 而`v2`的值, 在x32dbg中會對應到`[ebp-c]`, 其值是由下列assembly而來的,

```
001012C3 | B9  | mov ecx,32                          |
001012C8 | BE  | mov esi,idamudamudamuda.104058      |
001012CD | 8B7 | mov edi,dword ptr ss:[ebp-C]        |
001012D0 | F3: | rep movsd                           |
```

它在做的事情是把`104058`開始的`0x32 (50d)`個double word複製到[ebp-c] (BA0000)開始到對應結束的位置, 而`104058`就是func1開始`+= seed`的地方, 隨後它就`call dword ptr ss:[ebp-C]`, 進入到funcx。

```
00BA0000  55 8B EC 51 C7 45 FC 00 00 00 00 EB 09 8B 45 FC  U.ìQÇEü....ë..Eü
00BA0010  83 C0 01 89 45 FC 8B 4D 0C 03 4D FC 0F BE 11 85  .À..Eü.M..Mü.¾..
00BA0020  D2 74 25 8B 45 08 03 45 FC 0F BE 08 83 C1 23 83  Òt%.E..Eü.¾..Á#.
00BA0030  F1 66 0F BE D1 8B 45 0C 03 45 FC 0F BE 08 3B D1  ñf.¾Ñ.E..Eü.¾.;Ñ
00BA0040  74 04 33 C0 EB 07 EB C5 B8 01 00 00 00 8B E5 5D  t.3ÀëëÅ¸.....å]
00BA0050  C3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  Ã...............
```

到這邊可以做個小結論, 如果`BA0000`之後開始的binary code不符合一個function該有的樣子(e.g. function prologue 通常為`push ebp; mov ebp,esp`), 就不會順利執行, 又通常function一開始的machine code通常都是55 (`push ebp`)為首, 為了使它順利執行, 就要調整seed, 使得`104058`的byte`+= seed`會等於0x55, 由於在尚未加seed之前, 它的值為0x45, 所以得出seed = 0x10, 就可以看看funcx到底是什麼東西。

Part 3. funcx(flag, &unk_404018)

這邊相較之下比較沒什麼問題, 直接從x32dbg可以觀察出是個for loop, 它將flag的每個byte都加上0x23然後xor 0x66, 然後跟`104018`開始之後的32個byte比較是否相等, 於是將以下的32個bytes,

```
00104018  0F 09 02 0C F8 FA 30 F0 22 22 FA 30 F0 22 22 FA  ....øú0ð""ú0ð""ú
00104028  30 F0 22 22 35 ED E4 F6 FA E4 EC 35 E1 22 22 C6  0ð""5íäöúäì5á""Æ
```

用python就可回推正確的flag, 讓funcx最後可return true, 也得到真正的flag。