Secure Programming 2020

HW7 write-up

Go Crazy

腳本附在zip裡 exploit.py跟exploit.gdb

usage: gdb --nx -batch-silent -x ./exploit.gdb (gogo在同一個目錄下)

用ghidra開gogo看會發現,gogo會要求輸入,這個輸入的第一個字元跟最後一個字元必須是'x',才能進到下一個basic block.

```
...
if ((3 < (long)pcVar4) && (*pcVar3 == 'x')) {
...
if ((pcVar4 + -(long)pcVar2 == (char *)0x1) &&
(pcVar3[-(long)(pcVar4 + -(long)pcVar2) >> 0x3f & (ulong)pcVar2] == 'x')) {
...
```

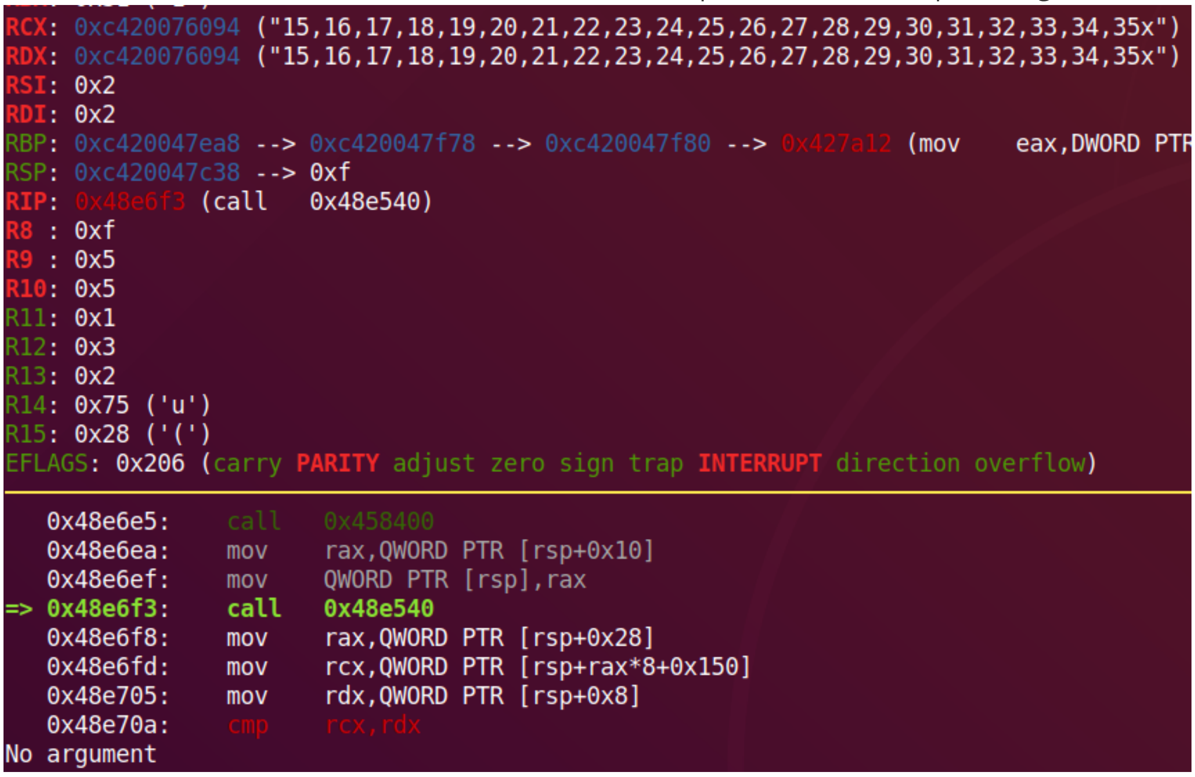
跟ida交互對照一下, gogo會先call strings_Split,再call main_check_input. main_check_input function會進入一個loop,loop的次數是36次,每次loop會先過一個判斷式,然後再call strconv_Atoi跟main_bezu,之後還有一個判斷式,如果沒過就會跳出loop.

```
//main
...
call    strings_Split
...
call    main_check_input
...
//main_check_input
...
cmp     rax, 24 //loop condition
...
cmp     rcx, rdx //if fail, jump out loop
...
call    main_bezu
...
cmp     rcx, rdx //if fail, jump out loop
...
```

根據golang的calling convention,function的參數會存在stack裡,用gdb看一下strings_split的參數跟回傳值會發現strings_split會依照','切分input,而且傳入main_check_input時不會是input的一開始.再看一下main_check_input的第一個cmp rcx rdx會發現rcx是f,而rdx是輸入的用','隔開的string的數量,再結合main_check_input的資訊可以總結出大概的程式邏輯:

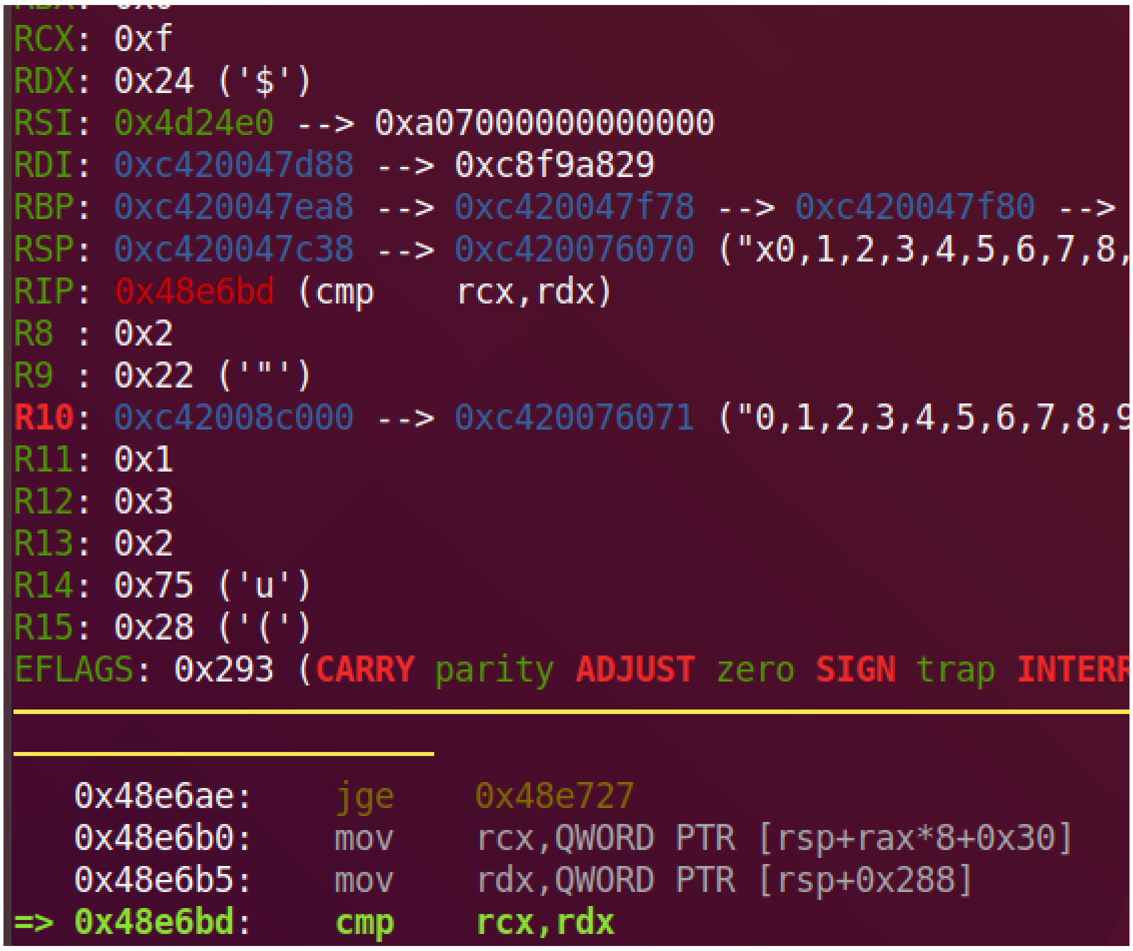
strings_split會把input用','切成數個string,再傳入main_check_input,然後main_check_input會把每一個字串轉成integer再呼叫main_bezu作運算,運算完之後,再跟某個值比較,如果一致就檢查下一個string,如果不一致就結束檢查,所以loop的次數是36,那個input的string個數也應該是36.

用
x0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35x
實際跑跑看,會發現進入main_check_input時,rcx裡的input變成
15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35x,而main_check_input的第一個cmp rcx rdx的rcx也是15,因此可以知道,第一個判斷式是在檢查loop的index是否小於input string的個數.



```
RCX: 0xc420076094 ("15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35x")
RDX: 0xc420076094 ("15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35x")
RSI: 0x2
RDI: 0x2
RBP: 0xc420047ea8 --> 0xc420047f78 --> 0xc420047f80 --> 0x427a12 (mov    eax,DWORD PTR
RSP: 0xc420047c38 --> 0xf
RIP: 0x48e6f3 (call   0x48e540)
R8 : 0xf
R9 : 0x5
R10: 0x5
R11: 0x1
R12: 0x3
R13: 0x2
R14: 0x75 ('u')
R15: 0x28 ('(')
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)

   0x48e6e5:    call   0x458400
   0x48e6ea:    mov    rax,QWORD PTR [rsp+0x10]
   0x48e6ef:    mov    QWORD PTR [rsp],rax
=> 0x48e6f3:    call   0x48e540
   0x48e6f8:    mov    rax,QWORD PTR [rsp+0x28]
   0x48e6fd:    mov    rcx,QWORD PTR [rsp+rax*8+0x150]
   0x48e705:    mov    rdx,QWORD PTR [rsp+0x8]
   0x48e70a:    cmp    rcx,rdx
No argument
```

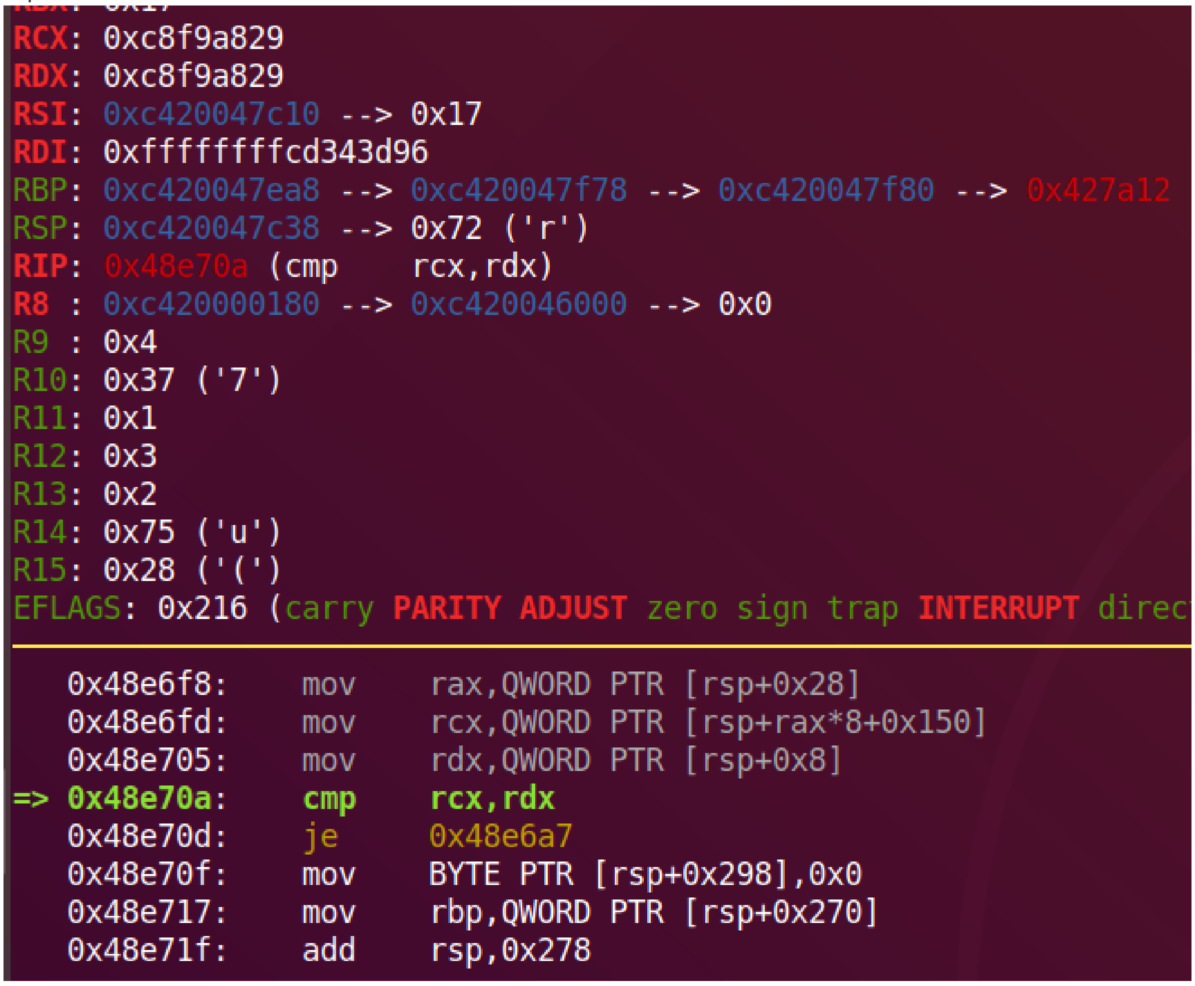


```
RCX: 0xf
RDX: 0x24 ('$')
RSI: 0x4d24e0 --> 0xa070000000000000
RDI: 0xc420047d88 --> 0xc8f9a829
RBP: 0xc420047ea8 --> 0xc420047f78 --> 0xc420047f80 -->
RSP: 0xc420047c38 --> 0xc420076070 ("x0,1,2,3,4,5,6,7,8,
RIP: 0x48e6bd (cmp    rcx,rdx)
R8 : 0x2
R9 : 0x22 ('"')
R10: 0xc42008c000 --> 0xc420076071 ("0,1,2,3,4,5,6,7,8,9
R11: 0x1
R12: 0x3
R13: 0x2
R14: 0x75 ('u')
R15: 0x28 ('(')
EFLAGS: 0x293 (CARRY parity ADJUST zero SIGN trap INTERR

   0x48e6ae:    jge    0x48e727
   0x48e6b0:    mov    rcx,QWORD PTR [rsp+rax*8+0x30]
   0x48e6b5:    mov    rdx,QWORD PTR [rsp+0x288]
=> 0x48e6bd:    cmp    rcx,rdx
```

再看一下第二個cmp rcx rdx,會發現rcx是定值,而rdx會根據input的不同而改變,如果我們可以找到input能夠通過每一次loop的cmp rcx rdx,就能夠得到唯一的input,那麼就可以進到下一步(結果input就是flag)

那麼現在的問題就是如何找值,最直接的方法就是看懂main_bezu做了什麼運算,然後直接逆推結果,得到input,然而想了很久還是想不出來bezu到底在幹麻,最後決定乾脆直接暴搜.



```
RCX: 0xc8f9a829
RDX: 0xc8f9a829
RSI: 0xc420047c10 --> 0x17
RDI: 0xffffffffcd343d96
RBP: 0xc420047ea8 --> 0xc420047f78 --> 0xc420047f80 --> 0x427a12
RSP: 0xc420047c38 --> 0x72 ('r')
RIP: 0x48e70a (cmp    rcx,rdx)
R8 : 0xc420000180 --> 0xc420046000 --> 0x0
R9 : 0x4
R10: 0x37 ('7')
R11: 0x1
R12: 0x3
R13: 0x2
R14: 0x75 ('u')
R15: 0x28 ('(')
EFLAGS: 0x216 (carry PARITY ADJUST zero sign trap INTERRUPT direc

   0x48e6f8:    mov    rax,QWORD PTR [rsp+0x28]
   0x48e6fd:    mov    rcx,QWORD PTR [rsp+rax*8+0x150]
   0x48e705:    mov    rdx,QWORD PTR [rsp+0x8]
=> 0x48e70a:    cmp    rcx,rdx
   0x48e70d:    je     0x48e6a7
   0x48e70f:    mov    BYTE PTR [rsp+0x298],0x0
   0x48e717:    mov    rbp,QWORD PTR [rsp+0x270]
   0x48e71f:    add    rsp,0x278
```

暴搜第16個string,最後得到114,發現好像不是很大,那就繼續暴搜下去.發現第二次是第33個string,而且剛好就是33,搜到第三次是第2個string發現是76,突然覺得好像那裡怪怪的,怎麼數字都這麼小,而且76感覺很熟悉,拿去ascii比對發現是'L',估計input就是flag,所以就繼續往下搜,最後得到flag.雖然到最後也不知道bezu在幹麻.

```
FLAG{gogo_p0werr4ng3r!you_did_IT!!!}
```