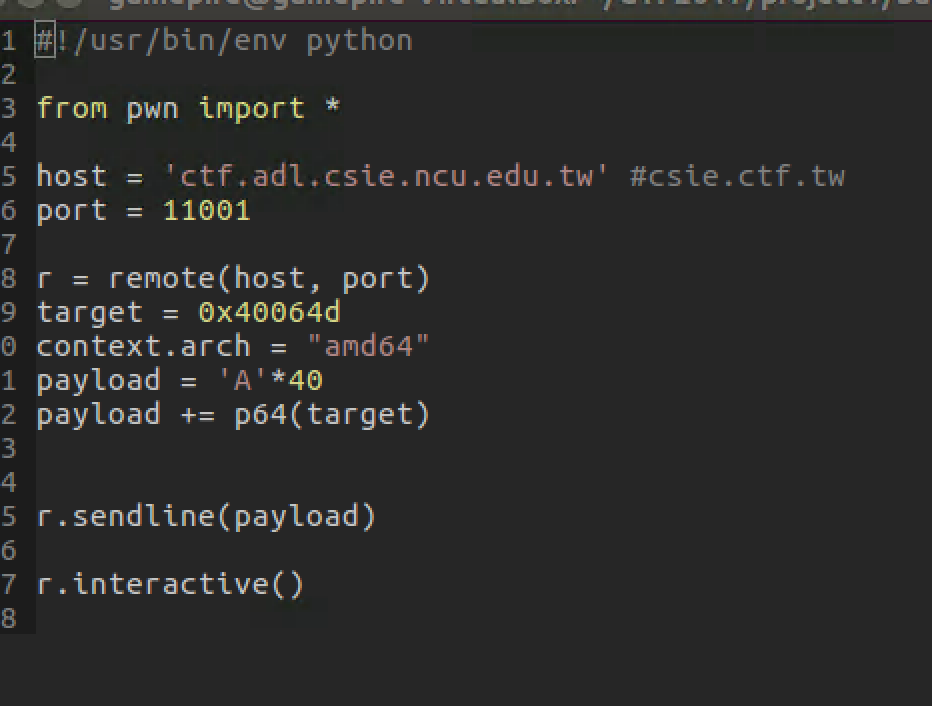
電腦攻防期中專題報告

106522077 詹勳和

1. baby\_bof

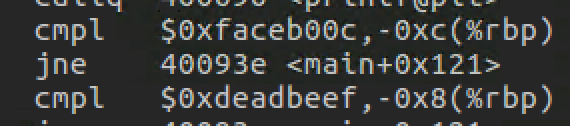
這一題就是一個簡單的bof題目，source code中有一個function 叫you\_cant\_see\_this\_its\_too\_evil直接開shell給你，那他吃input是用get又沒有檢查字串長度，就算一下offset讓input溢位把rsp蓋成you\_cant\_see\_this\_its\_too\_evil的位址就能成功執行shell

exploit code:

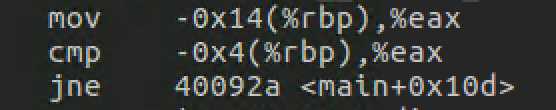


2.luck

也是簡單的bof，要你輸入a然後檢查b, c,跟password,過了就給shell,漏洞在read一個int卻給100bytes, 然後看到組語裡面

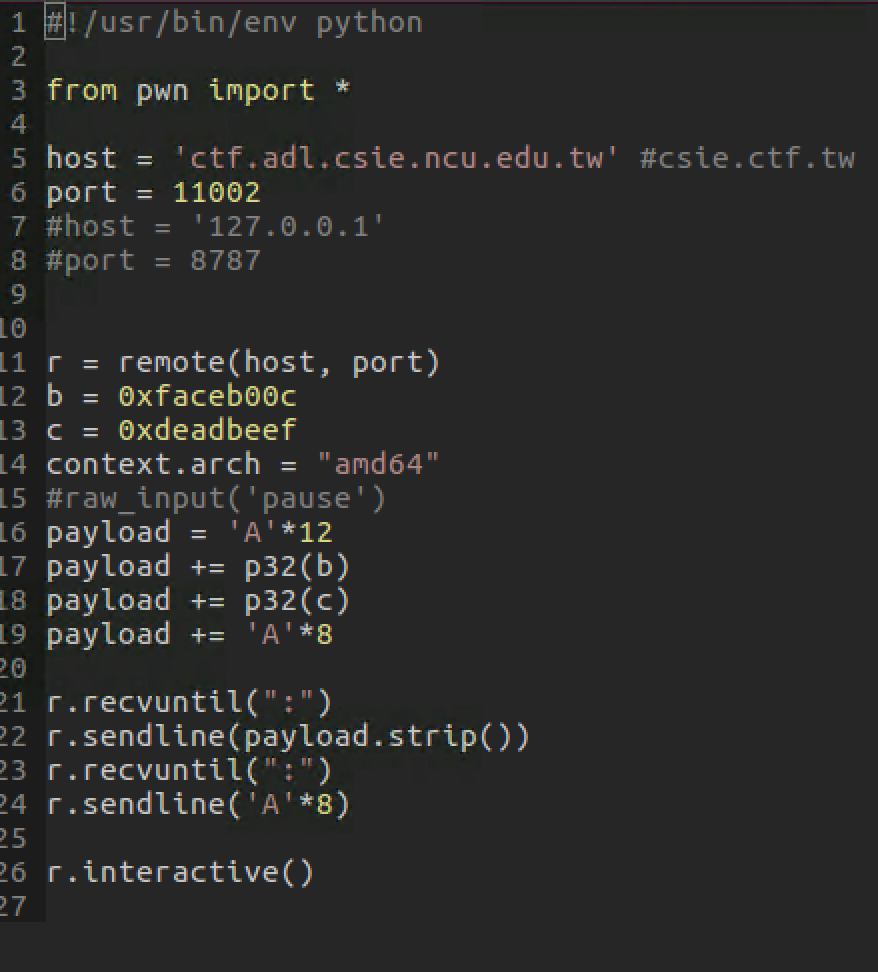


算一下offset(=12)然後放0xfaceb00c 再放 0xdeadbeef,之後看到



接著再隨便放自己訂的密碼，然後送payload, 送完後再送自己訂的密碼就拿到shell了

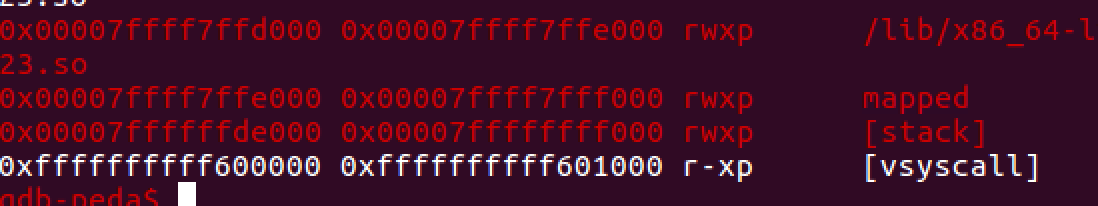
exploit code:



3.shellcode

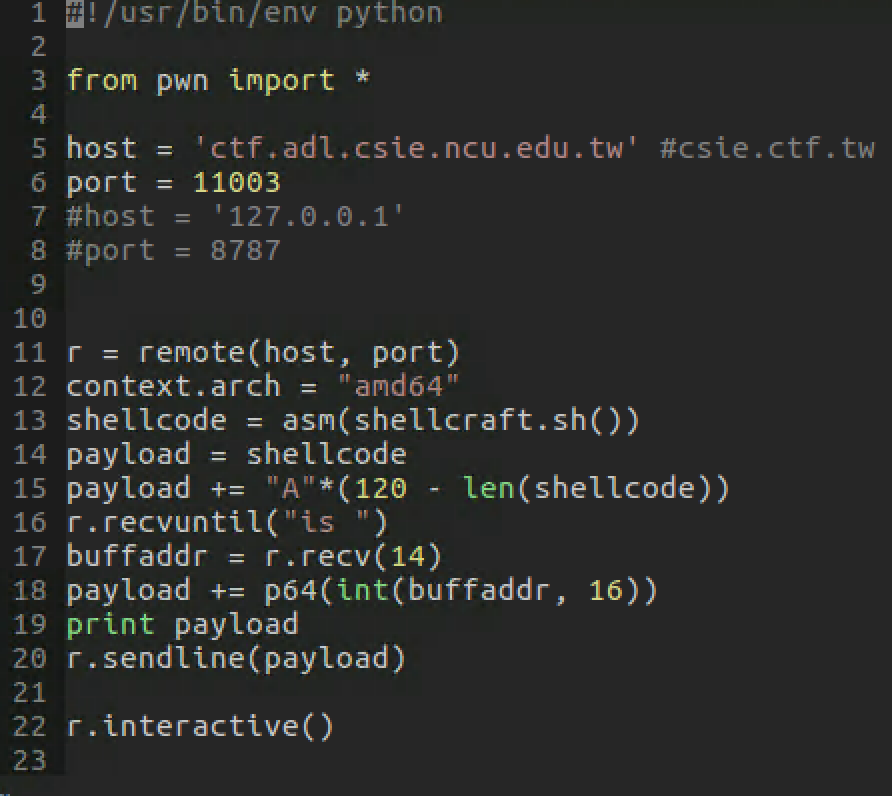
這一題是要我們練習寫shellcode，方便起見直接用pwntools 裡面的shellcraft.sh()

這題的buffer size是100但卻可以read 0x80顯然是個洞，然後這一題很善良的會print出buffer 的 base address而且用vmmap去看發現stack可執行



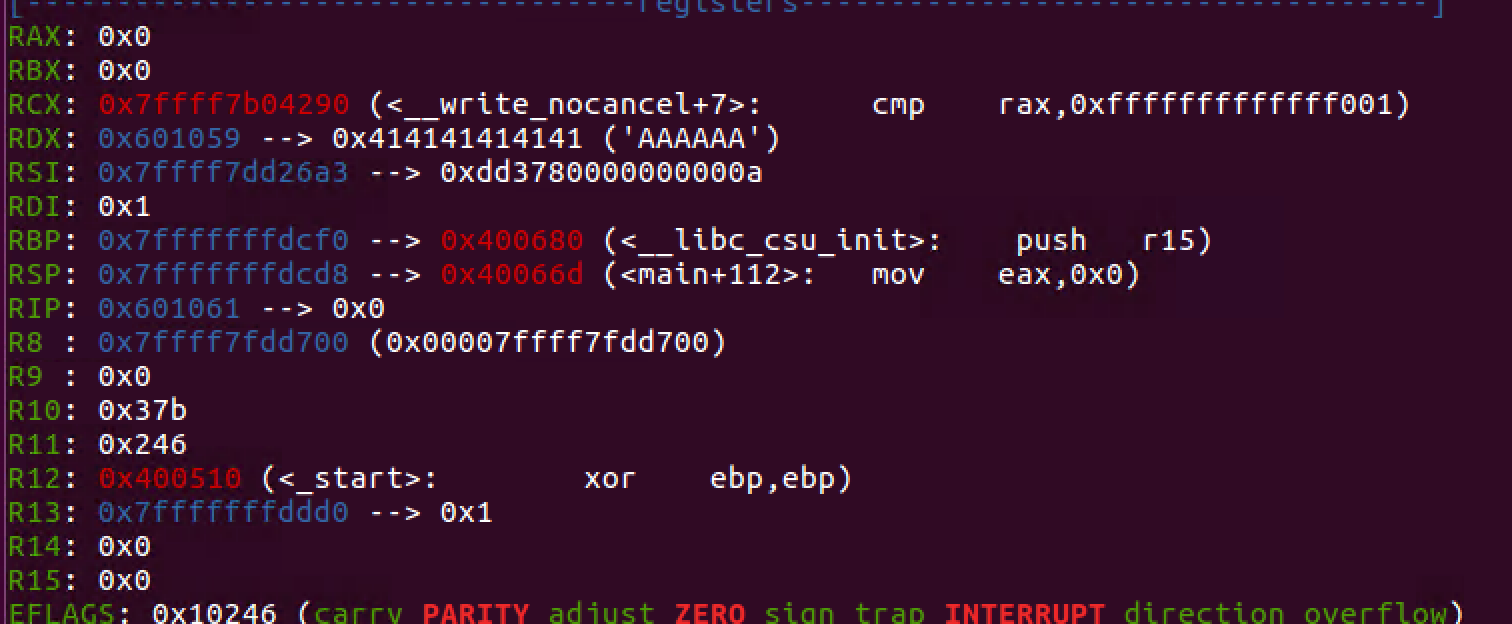
那就不客氣把shellcode寫在input的開頭，然後補滿offset蓋掉ret address 變成buffer的開頭(從程式接回來的address)去執行shellcode就完成了

exploit code:



4.shellcode revenge

這題一開始讓我們輸入一段6bytes的東東然後會直接執行，那顯然6bytes的code無法直接拿到shell，所以分兩段，第一段先讓他重新read一次，並設法讓他能read多一點，那首先看一下執行到要run第一段6bytes的code的register狀態



我們要read的話：



rax = 0, rdi = 0, rsi = buffer位置,

rdx = buffer size

執行前發現rax, rdx的值不用再設定了所以只需要把rdi, rsi設好就行，那用mov的話會用到太多bytes,所以用push,pop來完成

第一段shellcode:

push rax

pop rdi

push rdx

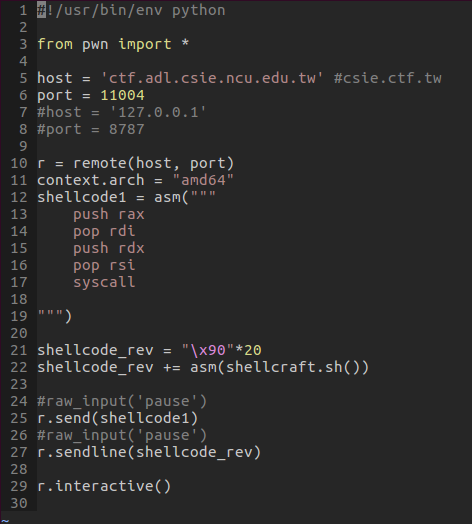
pop rsi

syscall

那第二段shellcode就直接用shellcraft.sh()就好只是因為rsp跳的位置會稍微後面一點所以為了怕直接從shellcode的中間讀到，在前面先墊一些nop(\x90)

還有一點要注意，第一段的payload不能送換行進去，用地二段的換行去觸發read就好，不然夾帶/n進去會失效。

exploit code:



5.ROP

題目就是要輸入一段input這樣，然後有boa的漏洞

由於這題有開NX不能用shellcode，所以要用rop去解題

那目標是要將

1=>rsi,rdx設為0

2=>rax設為0x3b(execve)

3=>rdi設為/bin/sh所在位置

4=>syscall的gadget

1+2 找一下gadget



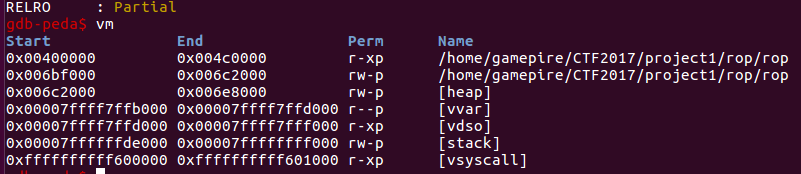
運氣不錯，兩個pop一次完成



pop rax 要放0x3b

3比較麻煩

先找個可寫讀的空間



就找data段後面0x100來用好了 => 0x6c2000 – 0x100

把這個segment存到rdi然後將/bin/sh\x00寫進其他隨便一個register，再將此register寫進rdi指向的地方，由於最後一部最難找，所以先找其指令



找到了~

再來找pop rdi 跟 pop rcx





4.syscall



Ok全找到了

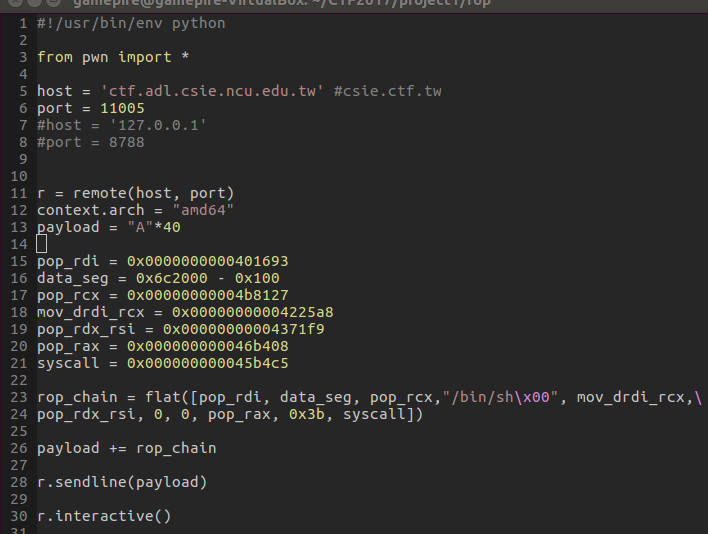
接下來就兜起來就可以嘞~

Rop chain=

pop\_rdi, data\_seg, pop\_rcx, “/bin/sh\x00”, mov\_drdi\_rcx,

pop\_rdx\_rsi, 0, 0, pop\_rax, 0x3b, syscall

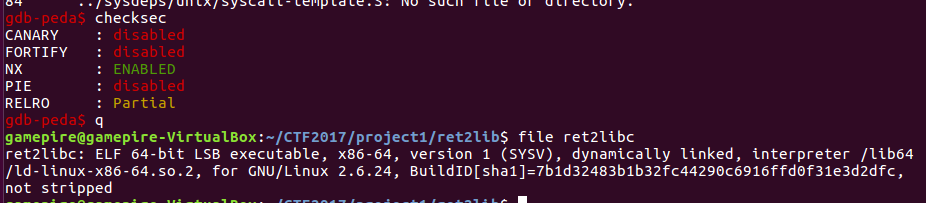
exploit code:



6.ret2libc

這題一開始要輸入一個address然後噴回該位址的內容，之後會要你輸入一段input會用strlen檢查長度，但還是有boa的漏洞

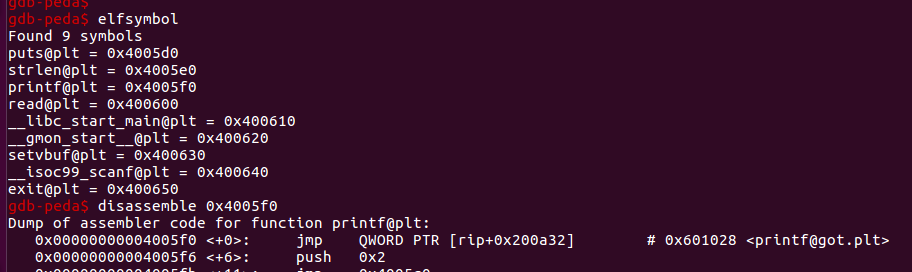
有開NX然後是dynamic link



這代表要兜rop chain，但gadgets又很少，連systcall都沒有，所以只好用libc 裡面的system

那因為第一段的input，我們只要給printf的got，程式就會噴回printf存code的位置，在減去printf 在libc中的offset就能得到libc在記憶體中的base address，最後再加上system的offset就能得知libc 的system在記憶體的位置了

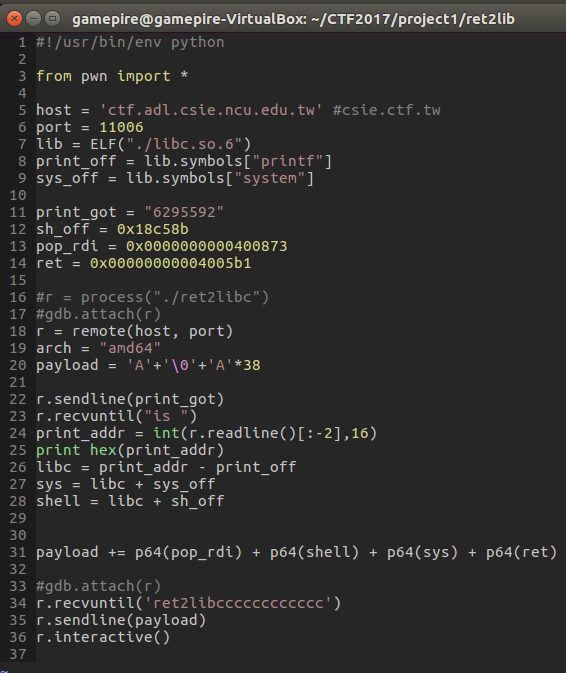
Printf got:



拿到system的address後再用能用的address兜一兜然後bypass調strlen的檢查就get shell了

Bypass strlen 就在payload中塞個\0騙過去就好了

exploit code



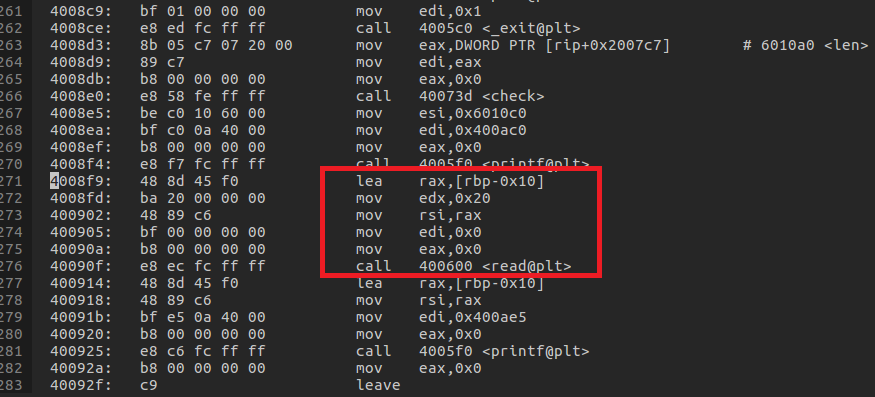
7.shellcode revenge++

這一題一開始會要你輸入一個name並檢查，是否為惡意字串，之後再讓你輸入一段input那這段輸入有bof漏洞，但是能overflow的大小有限，所以決定採用migration的方式來做，那首先看一下可以migrate到哪裡

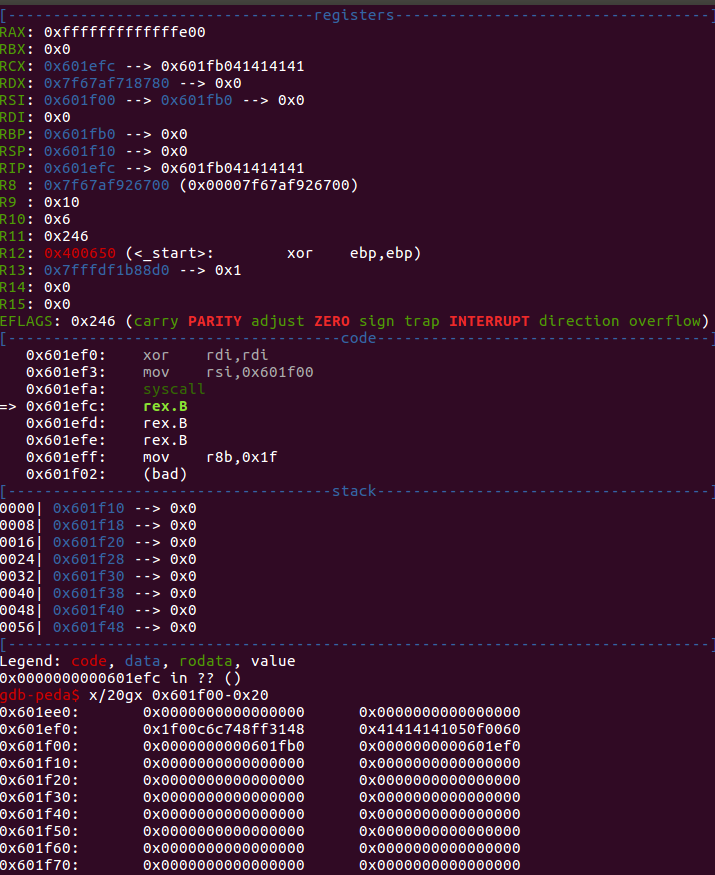


Vmmap看一下決定用data段後半段當作migration的buffer -> buf = 0x602000 – 0x100

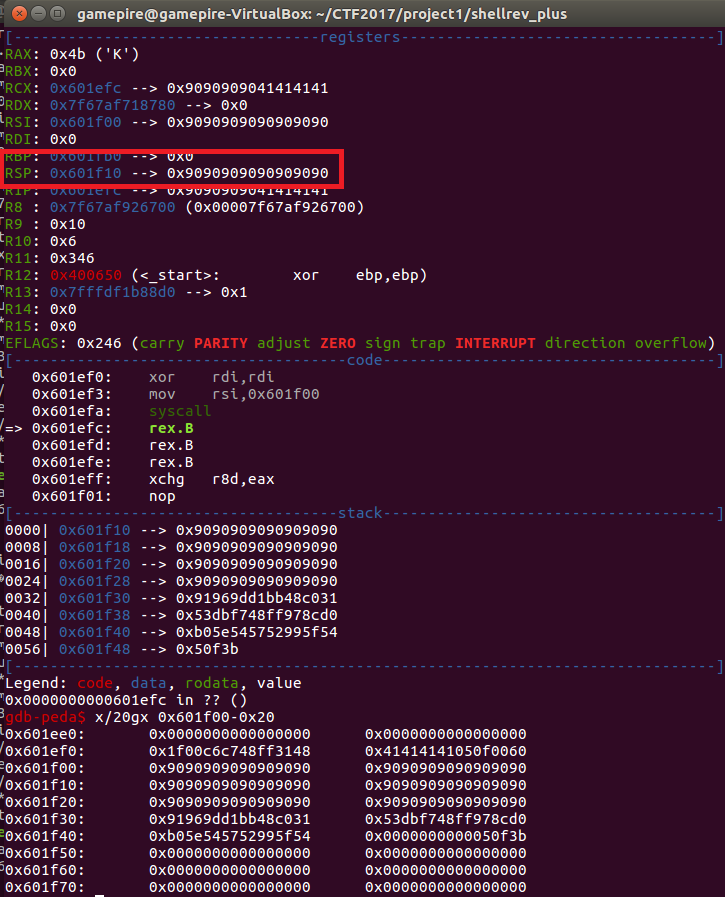
接下來應為第一段的input不能用所以直接跳回程式碼中的read讓他幫我們read到buf中



那這個read因為是程式碼給的，大小只有0x20有點不太夠，所以我們要在migration一次，這次自己造一個read一次可以輸入多點東西，此外記得還要overflow到起始位子去執行，從上面程式我們可以看到rax, [rbp – 0x10]所以這次read進來的位置其實不是我們一開始希望migrate到的buf，而是buf-0x10。

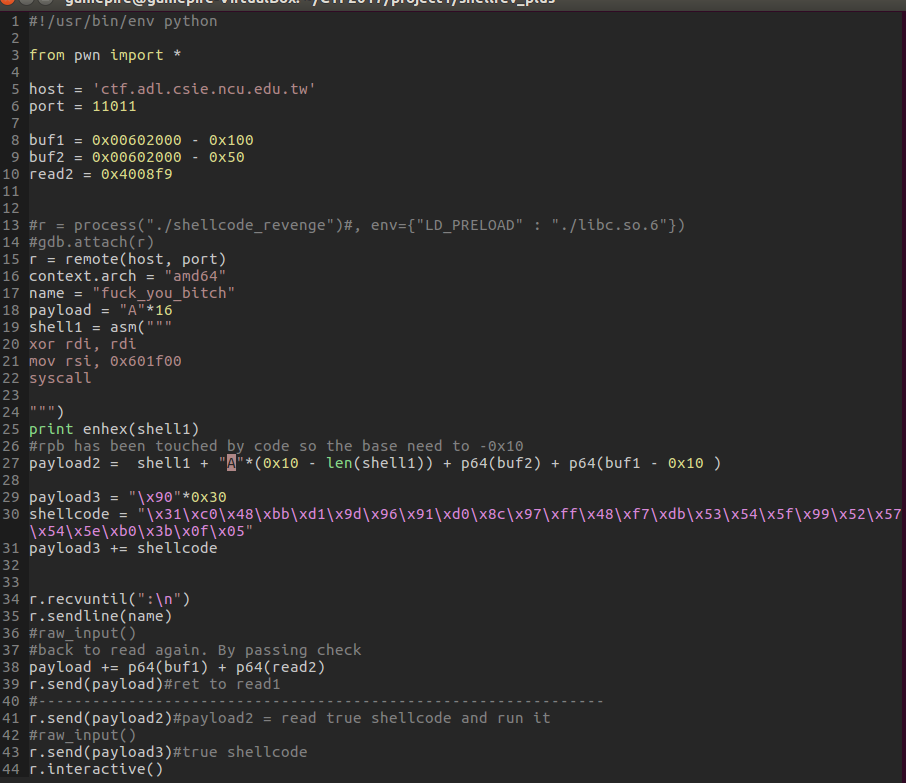


第二段有用的input丟進去後用裡面的shellcode(read)讓真正的shellcode讀進buf1，剛好接在後面rsp要執行的地方為了，保險起見依然在前面塞NOP



最後成功執行shellcode

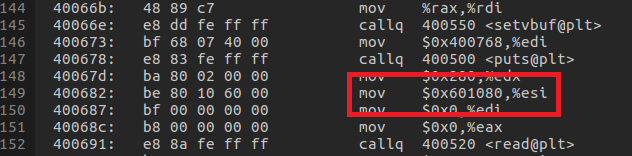
exploit code:



8.ROP revenge

這題跟上一題差不多只是不能用shell code要用rop chain但一樣要用migration，一開始給你輸入0x280然後再給你一個可以overflow 0x10的read

看一下sourcecode



存第一段input的read的位址在0x601080所以知道第一次migration的位置了，那邊的話要想辦法leak出libc的address(NX ENABLED)，之後再migrate一次同時read第二段rop chain拿到shell，

Leak libc base:

這程式之前用過puts，所以

pop rdi, puts got, puts plt

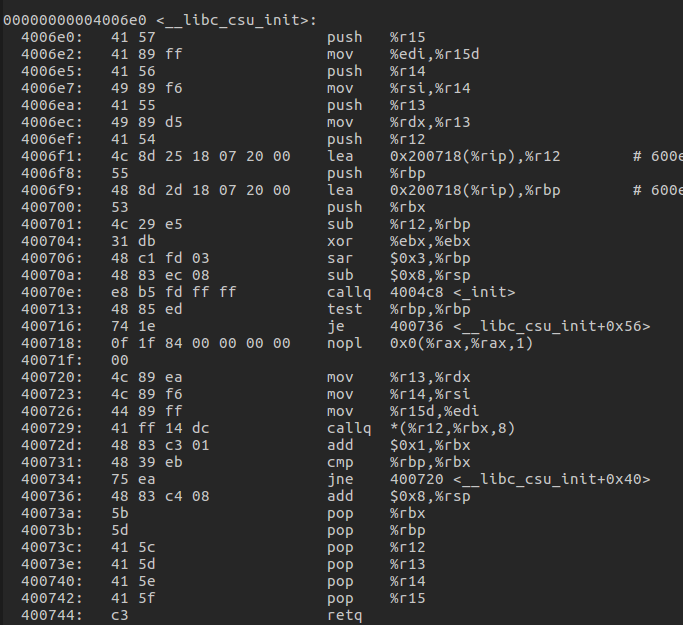
再接回來減掉puts的 offset得libc base address

這部分出現了一點問題，就是puts在執行時，stack會往下長超出原本stack space，導致puts出來一些亂七八糟的東西，解決方式就是在前面先墊一些垃圾，這樣stack就不會長出去了。

Run read:

這部分因為找不到pop rdx的gadget，所以只好用另一個麻煩的方法，

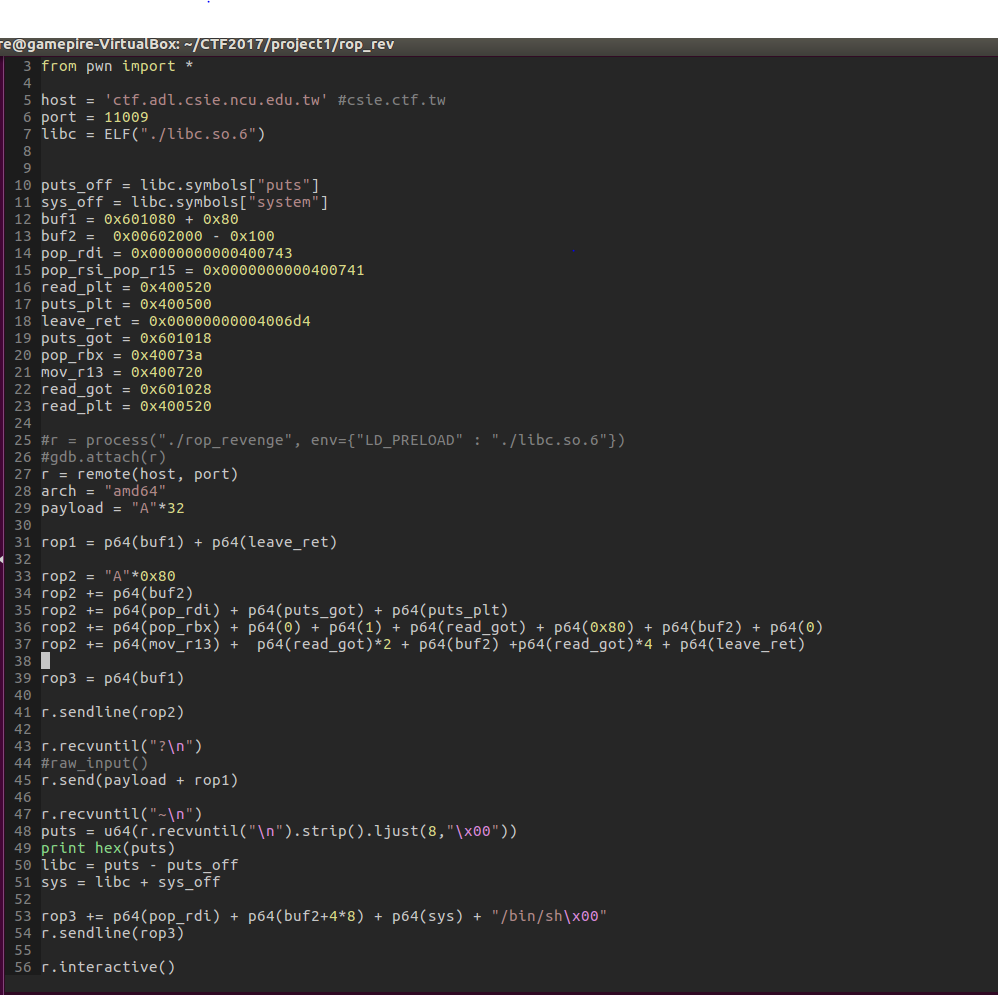
很多binary檔有這個function



那從40073a開始ret跳回400720最後在leave ret

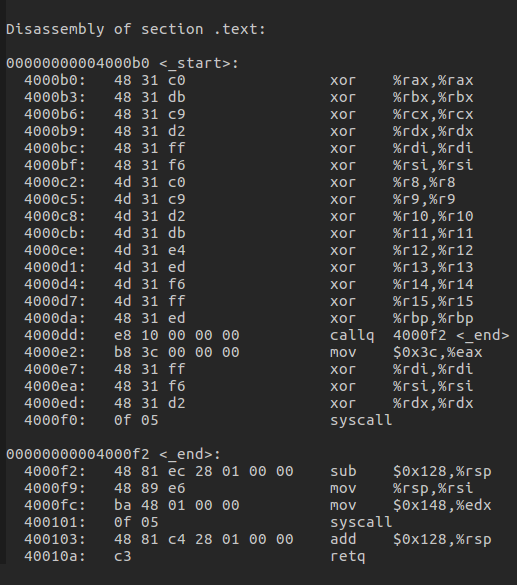
可以執行任意syscall，還可以順便控rpb migrate 去想去的地方，這裡看一下vm用data段的後半段0x602000 –0x100 的地方，都完這段rop chain會需要再input真正要開shell的rop chain(2)，最後migrate去rop chain拿到shell。

exploit code:

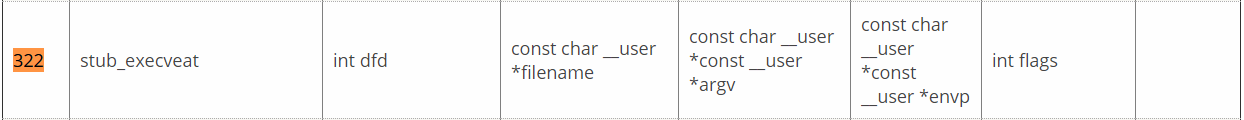


9.end

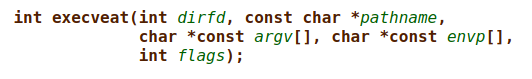
這題objdump之後可以看到，一開始先把各register清零之後call end function，end function會開一個，大小為0x128的buffer，然後讓你輸入0x148



有開NX加上有syscall可見要都rop chain，但由於gadgets實在太少，很難用一般的execv，因為一堆register不可控，那上網找了一下發現有一個神奇的syscall叫stub\_execveat



這個syscall的參數是長:



然後

https://2.bp.blogspot.com/-ta8ADQz6cY4/WbCHwQ1Dw-I/AAAAAAAAB7g/4FQiTMAEHGovN91Nd9NJZNWB_niXfWbMQCLcBGAs/s1600/Selection_024.png

所以說，原本不可控的rdi不須用了(/bin/sh是絕對路徑)~~

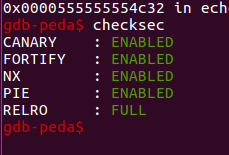
那rsi是可控(input的開頭)的又變成第一個參數，所以說只要把/bin/sh\x00放在開頭，然後控一下rax然後執行syscall(overflow 蓋掉rsp)就行了，阿因為read執行後會把字串長度存在rax所以控制在串長等於322就好了!

exploit code:



10.NCU center

有開的保護機制:



需要leak出

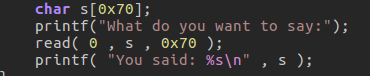
-> stack base address

-> libc base address

-> canary

這題一執行起來會給出一個選單，也就是進入一個迴圈，那有提供幾個簡單的function那漏洞在幾個地方:

1

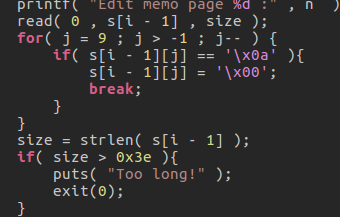


2



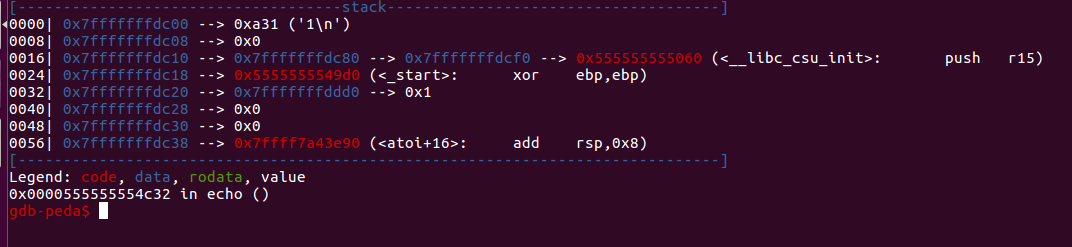
3

 加



1跟2的洞在於，原本buffer拿來用的時候可能殘留一些有用的資訊

Ex.



這裡是呼叫echo後進去read但還沒input東西的stack的狀態，能看到<atoi+16>(libc某行)

然後他是直接print一整個buffer所以說如果中間都沒有\x00終止，就能把這些有用的東西一起印出來~~

3的漏洞在於它的size是可以調整的，最大到0x3e，buffer大小只有0x30，剛好可以蓋過ret address

解題思路:

Step1->利用1 leak出libc base

Step2->利用2+3 leak 出 canary跟stack base

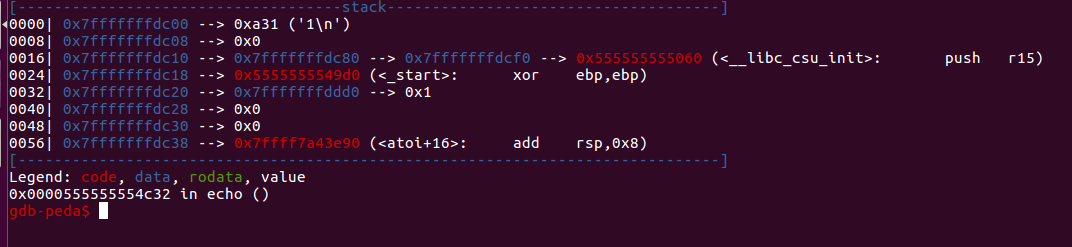
Step3->做出rop chain(用到libc base)後

Step4->利用3 jump to stack base去改掉ret的address

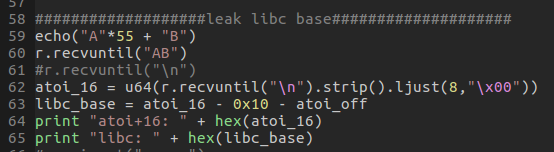
Step5->塞好rop chain

Step6->call exit()讓成是去執行ret address -> get shell!

Step1:呼叫echo



可以看到填滿56個字元後echo出來的東西最後6bytes是libc的東西，補成8bytes後再減掉他在libc的offset後就拿到libc base address了

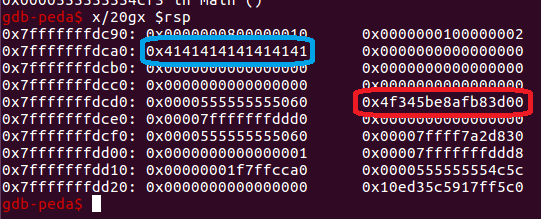


Step2:

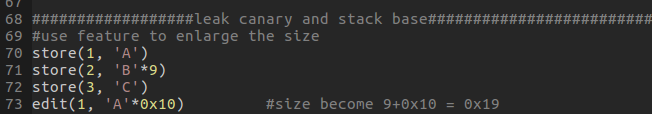
先找一下canary跟buffer的相對位置，先在第一段buffer輸入AAAAAAAA

看一下要讓buffer3補齊0x18 + 1(最後2bytes是0)個東西才能利用漏洞2 leak出canary

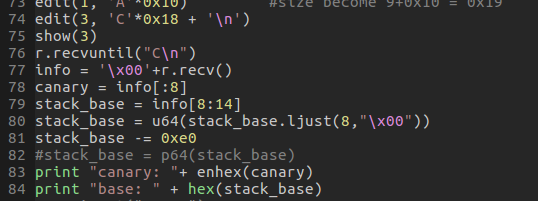
藍buffer1



那要讓size變大才能read進0x18到buffer3，所以先讓size變大



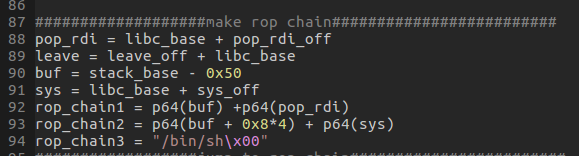
edit buffer1 變A\*0x10後再加上原本存在buffer2的0x9個b，size 會因為 = strlen的關係變成0x19此時再edit buffer3變0x19個就能leak出canary了



因為發現接在canary後面的6bytes(一開始的殘餘資訊)跟stack的base一值相差0xe0所以就順便leak出stack base了~

Step3:

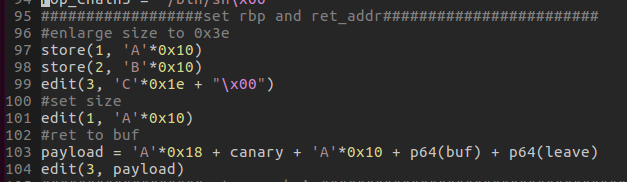
利用之前leak出的東西組好rop chain



Step4:

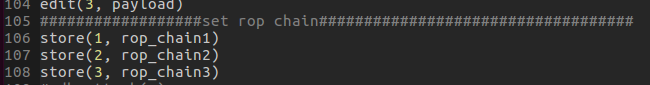
要改到ret address 需要把size 擴大到0x3e，先把buffer123塞到剛好0x3e + \x00然後call 隨便一個edit，完成擴大size，然後edit buffer3塞進 payload

payload = A\*0x18 + canary + A\*0x10 + buffer1\_address + leave\_ret



Step5:

把step3寫好的rop chain依次塞進buffer中



Step6:

Call exit()

Done!

exploit code:



