

xctf pwn 踩坑笔记(一): 以xctf入门题level3为例

原创

ホワイトアルバム  于 2019-10-30 18:49:45 发布  366  收藏 3

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/xiongzixun/article/details/102820136>

版权

操作环境: ubuntu16.04 + 惠普台式机

IDA环境: win10 + 联想笔记本**

1. 打开文件

下面以level3为例:

xctf中的附件一般为application/octet-stream

而且为了能够使用, 我们必须要先通过chmod命令获得X权限

如果是压缩文件还要先gunzip或者tar等试一下

题目附件: 

```
lcs@nau8200:~/下载$ chmod -x c42e36c561b844fe91c0d086fbecbf50.gz
lcs@nau8200:~/下载$ mv c42e36c561b844fe91c0d086fbecbf50.gz l3.gz
```

```
lcs@nau8200:~/下载$ gunzip l3.gz
```

```
lcs@nau8200:~/下载$ tar -zxvf l3
./level3
./libc_32.so.6
```

然后, 通过checksec 这个文件以后, 一定要先关注Arch, 也就是这个ELF究竟是64位, 还是32位。

(下面是level2的checksec结果)

```
Arch:      i386-32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x8048000)
```

因为IDA分为64位和32位, 64位的IDA是无法decompile32位的elf得到伪代码的

2. 基本概念

1. **canary**没开，意味着可以用stack overflow
2. **NX**开了，意味着不能直接注入shellcode，要利用现成的system等lib函数和写好了的参数（比如“binsh”）
3. **RELRO**没有full relro：意味着我们可以修改got表
4. **PS :ALSR**意味着libc地址是随机的，也就是说，每次调用plt.entry的地址不一样
5. **PIE**：“如果程序开启了PIE保护的话，在每次加载程序时都变换加载地址，从而不能通过ROPgadget等一些工具来帮助解题”

3.gdb调试

进入gdb以后设置断点的小技巧

```
(gdb)lay asm
```

我觉得这比odbdump方便哈

```

0x80484a3 <main+6>    sub    esp,0x20
0x80484a6 <main+9>    mov    eax,ds:0x804a028
0x80484ab <main+14>   mov    DWORD PTR [esp+0xc],0x0
0x80484b3 <main+22>   mov    DWORD PTR [esp+0x8],0x2
0x80484bb <main+30>   mov    DWORD PTR [esp+0x4],0x0
0x80484c3 <main+38>   mov    DWORD PTR [esp],eax
0x80484c6 <main+41>   call  0x8048390 <setvbuf@plt>
0x80484cb <main+46>   lea   eax,[esp+0x16]
0x80484cf <main+50>   mov    DWORD PTR [esp],eax
0x80484d2 <main+53>   call  0x8048350 <gets@plt>
0x80484d7 <main+58>   lea   eax,[esp+0x16]
0x80484db <main+62>   mov    DWORD PTR [esp],eax
0x80484de <main+65>   call  0x8048360 <puts@plt>
0x80484e3 <main+70>   mov    eax,0x0
0x80484e8 <main+75>   leave
0x80484e9 <main+76>   ret
0x80484ea     xchg  ax,ax
  
```

exec No process in: Line: ?? PC: ??
(gdb) b *main+

<https://blog.csdn.net/xiongzixun>

图片来自互联网

我们先要让level3连接到提供的libc文件上

输入下面的命令

```
lcs@nau8200:~/下载$ ldd ./level3
linux-gate.so.1 => (0xf7f42000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf7d6b000)
/lib/ld-linux.so.2 (0xf7f44000)
```

我们可以看到level3目前使用的libc（也就

是我们电脑本地上的）

可以通过

```
lcs@nau8200:~/下载/l3l$ LD_LIBRARY_PATH=. ldd ./level3
linux-gate.so.1 => (0xf7f43000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf7d6c000)
/lib/ld-linux.so.2 (0xf7f45000)
```

来修改到本地目录，不过由于libc.so要和

ld-linux.so版本相同，那么我们就用ubuntu14.04的虚拟机了

这里插题说一句，level3中提供了一个libc的文件，要看elf文件中plt函数的entry的地址，可以使用下面的命令

```
ics@nau8200:~/下载$ readelf -s ./libc_32.so.6 | grep 'write@'
2323: 000d43c0 101 FUNC WEAK DEFAULT 13 write@@GLIBC 2.0
```

当ALSR开启以后，entry的地址是会变化的，但是我们可以通过上面的命令得到函数关于libc头部的offset（也就是偏移地址）

再通过下面的命令：

```
pidof [elf程序的名称]
```

```
cat /proc/[通过上一行命令得到的pid]/maps
```

得到如同下面的效果：（这里以level3文件为例）：

```
7d7e000-f7d7f000 rw-p 00000000 00:00 0
7d7f000-f7f2f000 r-xp 00000000 08:02 34865906 /lib/i3
6-linux-gnu/libc-2.23.so
7f2f000-f7f31000 r--p 001af000 08:02 34865906 /lib/i3
```

这就是level3调用libc的

顶部

用这个地址+前面得到的offset，就是我们需要的函数地址了