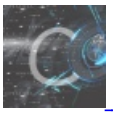
强网杯2019 Copperstudy

原创

无名函数 于 2021-08-13 22:54:50 发布 47 收藏 1

分类专栏： Buu-crypto 文章标签： python

本文链接：https://blog.csdn.net/m0_57291352/article/details/119685392



强网杯2019 Copperstudy

靶机：node4.buuoj.cn:29678

第一次见靶机的题，找题目找了半天

打开VMware中的浏览器，输入靶机地址，得到：

```
[+]proof: skr=os.urandom(8)
[+]hashlib.sha256(skr).hexdigest()=67fbc84508c4087471605c92639387450c6d41fa97936c8a64405408403b0f72
[+]skr[0:5].encode('hex')=32f574e3af
[-]skr.encode('hex')=
bye~
```

os.urandom（）函数：获取一个指定长度的随机bytes对象
hashlib.sha256(x).hecdigest()：使用sha256算法计算x的散列并返回十六进制的值。

爆破得到skr

```
from Crypto.Util.number import *
import hashlib

for i in range(10000,20000000):
 tar = 0x32f574e3af
 payload = long_to_bytes(i)
 payload = long_to_bytes(tar) + payload
 if hashlib.sha256(payload).hexdigest() == '67fbc84508c4087471605c92639387450c6d41fa97936c8a64405408403b0f72' :
  print(i)
  break
```

运行得到：`15740973`
转为十六进制为f0302d
加上前面的为32f574e3aff0302d

然后呢，，，，，把它输到哪啊啊啊

在菜菜的不懈努力下终于下载好了nc（并且删完了所以由于下载而附带的一系列软件）

然后发现在虚拟机上早就有安装好的nc

但是，有了nc之后呢，，，，

咳咳咳，打开虚拟机终端，输入：`nc node4.buuoj.cn 27362`

得到：

第一题

```
[+]proof: skr=os.urandom(8)
[+]hashlib.sha256(skr).hexdigest()=c8753a2c2e65194731a5c821803c197b018b157d3b43920dbc0e632ad108e835
[+]skr[0:5].encode('hex')=4f2d16389b
[-]skr.encode('hex')=
```

终于不直接出bye~了
（靶机题目数据每次打开是不同的）

按照前面程序，求得：`4f2d16389b9d3681`

输入后得到：

第二题

```
proof completed

[+]Generating challenge 1


  [+]n=131120618206856432396638311669283271195794258306324585688015444065067694612795909627723402491835694375593
942006355261836986045823857693811595637108236894172744795496275960953986211829958914545169537220250689262935125053831252275791697789466313699617535878563445822576836723132303786033240053377889139024340234318870614543685661007476185825902703859182046561560890535197095360019069640086357085106725502195468940060914835203554360910538663127184313184987836377127738784237774673166058655162481762487806371326158078862720298437701868334257920491081874873382378508062037282173748487992504198596468710570962970206709042110

[+]e=3

[+]m=random.getrandbits(512)

[+]c=pow(m,e,n)=159875547240031002953260760364131636343986009476950968578039379989694417630147317203751961040107945558680690243936479660405932582678884637321844950207094575600430505771989883637547037416360880894724889710503246541621666576783765571104927037122863068688437284662248875508271624420262621633409353337217052674327902685170

[+]((m>>72)<<72)=25191885942717592057578644860976055401354075015710786272388494435612190577518431705402618426772396819087360

[-]long_to_bytes(m).encode('hex')=
```

很明显的RSA，已知n、e、c，而且e=3，用小明文攻击

```
import gmpy2
from Crypto.Util.number import long_to_bytes

c =159875547240031002953260760364131636343986009476950968578039379989694417630147317203751961040107945558680690243936479660405932582678884637321844950207094575600430505771989883637547037416360880894724889710503246541621666576783765571104927037122863068688437284662248875508271624420262621633409353337217052674327902685170
m = gmpy2.iroot(c,3)[0]
print(long_to_bytes(m))
```

运行得到：`b'FLAG{2^8rsa7589693fc689c77c5f5262d654272427}'`

然后在用在线hex加密，或者

```
print(binascii.hexlify(long_to_bytes(m)))
```

得到：`b'464c41477b325e3872736137353839363933666336383963373763356635323632643635343237323432377d'`

输入（记得不要把b''输入上去）得到：

第三题

```
challenge1 completed

[+]Generating challenge 2


  [+]n=12784625729032789592766625203074018101354917751492952685083808825504221816847310910447532133616954262271205877651255598995305639194329607493047941212754523879402744065076183778452640602625242851184095546100200565113016690161053808950384458996881574266573992526357954507491397978278604102524731393059303476350167738237822647246425836482533150025923051544431330502522043833872580483142594571802189321599016725741260254170793393772931450105256865619044276136481848436193012414142643430573681924165511344041003861557512974246162546970410438518520810713062194629919698491236682483211303822317692508651902276300091817592190

[+]e=65537

[+]m=random.getrandbits(512)

[+]c=pow(m,e,n)=62782408615711924505647887580059895955377425016167078750608325396078823073758876178738568612582876566561756788790422803083953531798758960876153450000312824716423377479478423151821280427005640456571042661393826430299801542115339387972926355129202454375642270295647002295953722126917208461908136849869393055045615354362817030632420626621634838670700866112871743142623748651130976728617551823862023050720195286726128388098686875267654961395878528891498942922458284921839547167229541003685888183636336488516427698323731223583159185804490836937685548412761493354595554478716035204231837858803958791174102806757672279077 8

[+]((p>>128)<<128)=97522826022187678545924975588711975512906538181361325096919121233043973599759518562689050415761485716705615149641768982838255403594331293651224395590747133152128042950062103156564440155088825926440460692084053603243720571408903175188021300811980600935768415380089605603913803956970989644118217166645069086 72

[-]long_to_bytes(m).encode('hex')=
```

依旧是RSA，已知n、e、c

分解n然后按基础解法就可

```
import gmpy2
import binascii
from Crypto.Util.number import long_to_bytes

e = 65537
n = 12784625729032789592766625203074018101354917751492952685083808825504221816847310910447532133616954262271205877651255598995305639194329607493047941212754523879402744065076183778452640602625242851184095546100200565113016690161053808950384458996881574266573992526357954507491397978278604102524731393059303476350167738237822647246425836482533150025923051544431330502522043833872580483142594571802189321599016725741260254170793393772931450105256865619044276136481848436193012414142643430573681924165511344041003861557512974246162546970410438518520810713062194629919698491236682483211303822317692508651902276300091817592190
c = 62782408615711924505647887580059895955377425016167078750608325396078823073758876178738568612582876566561756788790422803083953531798758960876153450000312824716423377479478423151821280427005640456571042661393826430299801542115339387972926355129202454375642270295647002295953722126917208461908136849869393055045615354362817030632420626621634838670700866112871743142623748651130976728617551823862023050720195286726128388098686875267654961395878528891498942922458284921839547167229541003685888183636336488516427698323731223583159185804490836937685548412761493354595554478716035204231837858803958791174102806757672279077 8
p = 97522826022187678545924975588711975512906538181361325096919121233043973599759518562689050415761485716705615149641768982838255403594331293651224395590747133152128042950062103156564440155088882592644046069208405360324372057140890317518802130081198060093576841538008960560661715295741651653499691458486798196487
q = 13109367571161366116147627547344520668259755944700657138548225572760923878659695216580181402160269974987671268230778956811337476868963264272898657321177652647365177110443244350129466837244152598717439147299427105487330532434366627942674189761282788952544042858259221615158613888180619633192075896840350853163 7
phi = (q-1) * (p-1)
d = gmpy2.invert(e,phi)
m = gmpy2.powmod(c,d,n)

print(long_to_bytes(m))
print(binascii.hexlify(long_to_bytes(m)))
```

运行得到：`b'464c41477b325e3872736136653237376633353564626536646133656464366633353664326462366436667d'`

输入得到：

第四题

```
challenge2 completed

[+]Generating challenge 3


  [+]n=92896523979616431783569762645945918751162321185159790302085768095763248357146198882641160678623069857011832929179987623492267852304178894461486295864091871341339490870689110279720283415976342208476126414933914026436666789270209690168581379143120688241413470569887426810705898518783625903350928784794371176183

[+]e=3

[+]m=random.getrandbits(512)

[+]c=pow(m,e,n)=56164378185049402404287763972280630295410174183649054805947329504892979921131852321281317326306506444145699012788547718091371389698969718830761120076359634262880912417797038049510647237337251037070369278596191506725812511682495575589039521646062521091457438869068866365907962691742604895495670783101319608530

[+]d&((1<<512)-1)=78767399629537629766817107517095585210981493944224204980081160175300189731755602265399765187489720848791332103134071113833136035063396542064204538364495 5

[-]long_to_bytes(m).encode('hex')=
```

看起来跟前面第二题一样，但是用相同代码运行出来的

是：`1b5b7fc18b0db820211ba55ceeb56dcbbf1e181447f65f3b40dd8119dab89864ac15a28db8495fb0aa0692`

输入显示bye~，很显然错了，又要重来一次

还有个事：

运算时<< >>位移运算，也就是p右移128位再左移128位，右移直接吞左移不足位补零

例如：

```
>>> n = 249586783108217497039850947
>>> n>>5
779958697213179678249534 2
>>> (n>>5)<<5
249586783108217497039850944
>>> bin(n)
'0b1100111001110100000011101001111110011100011000001100101110101011101001101100110111000011'
>>> bin(n>>5)
'0b1100111001110100000011101001111110011100011000001100101110101011101001101100101110'
>>> bin((n>>5)<<5)
'0b1100111001110100000011101001111110011100011000001100101110101011101001101100110111000000'
>>>
```


然后第四题是已知d的地位

能力不够百度来凑

```
def partial_p(p0, kbits, n):
    PR.<x> = PolynomialRing(Zmod(n))
    nbits = n.nbits()

    f = 2^kbits*x + p0
    f = f.monic()
    roots = f.small_roots(X=2^(nbits//2-kbits), beta=0.3)  # find root < 2^(nbits//2-kbits) with factor >= n^0.3
    if roots:
        x0 = roots[0]
        p = gcd(2^kbits*x0 + p0, n)
        return ZZ(p)

def find_p(d0, kbits, e, n):
    X = var('X')

    for k in range(1, e+1):
        results = solve_mod([e*d0*X - k*X*(n-X+1) + k*n == X], 2^kbits)
        for x in results:
            p0 = ZZ(x[0])
            p = partial_p(p0, kbits, n)
            if p:
                return p


if __name__ == '__main__':
    n = 92896523979616431783569762645945918751162321185159790302085768095763248357146198882641160678623069857011
832929179987623492267852304178894461486295864091871341339490870689110279720283415976342208476126414933914026436 6
667892702096901685813791431206882414134705698874268107058985187836259033509287847943711761 83
    e = 3
    d = 78767399629537629766817107517095585210981493944224204980081160175300189731755602265399765187489720848791
332103134071113833136035063396542064204538364 4955

    nbits = n.nbits()
    kbits = d.nbits()

    print ("lower %d bits (of %d bits) is given" % (kbits, nbits))

    p = find_p(d, kbits, e, n)
    q = n//p
    print ("d0 = %d" % d)
    print ("d = %d" % inverse_mod(e, (p-1)*(q-1)))
```