深入浅出RSA在CTF中的攻击套路

看到请叫我滚去学习 / 2019-10-05 11:00:32 / 浏览数 31727

0x01 前言

本文对RSA中常用的模逆运算、欧几里得、拓展欧几里得、中国剩余定理等算法不展开作详细介绍，仅对遇到的CTF题的攻击方式，以及使用到的这些算法的python实现进行介绍。目的是让大家能轻松解决RSA在CTF中的套路题目。

0x02 RSA介绍

介绍

首先，我这边就不放冗长的百度百科的东西了，我概括一下我自己对RSA的看法。
RSA是一种算法，并且广泛应用于现代，用于保密通信。
RSA算法涉及三个参数,n,e,d，其中分为私钥和公钥，私钥是n,d，公钥是n,e
n是两个素数的乘积，一般这两个素数在RSA中用字母p,q表示
e是一个素数
d是e模 varphi(n) 的逆元，CTF的角度看就是，d是由e,p,q可以求解出的
一般CTF就是把我们想要获得的flag作为明文，RSA中表示为m。然后通过RSA加密，得到密文，RSA中表示为c。
加密过程
c=m^e mod n

```
c=pow(m,e,n)
```

解密过程
m=c^d mod n

```
m=pow(c,d,n)
```

求解私钥d

```
d = gmpy2.invert(e, (p-1)*(q-1))
```

一般来说，n，e是公开的，但是由于n一般是两个大素数的乘积，所以我们很难求解出d，所以RSA加密就是利用现代无法快速实现大素数的分解，所存在的一种安全的非对称加密。

基础RSA加密脚本

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

msg = 'flag is :testflag'
hex_msg=int(msg.encode("hex"),16)
print(hex_msg)
p=getPrime(100)
q=getPrime(100)
n=p*q
e=0x10001
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
print("d=",hex(d))
c=pow(hex_msg,e,n)
print("e=",hex(e))
print("n=",hex(n))
print("c=",hex(c))
```

基础RSA解密脚本

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import binascii
import gmpy2
n=0x80b32f2ce68da974f25310a23144977d76732fa78fa29fdcbf
#这边我用yafu分解了n
p=780900790334269659443297956843
q=1034526559407993507734818408829
e=0x10001
c=0x534280240c65bb1104ce3000bc8181363806e7173418d15762


phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(hex(m))
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

0x03 p和q相差过大或过小

利用条件

因为n=p*q
其中若p和q的值相差较小，或者较大，都会造成n更容易分解的结果
例如出题如下

```
p=getPrime(512)
q=gmpy2.next_prime(p)
n=p*q
```

因为p和q十分接近，所以可以使用yafu直接分解

yafu分解

使用

```
factor(*)
```

括号中为要分解的数

在线网站分解

http://factordb.com/
通过在此类网站上查询n，如果可以分解或者之前分解成功过，那么可以直接得到p和q

0x04 公约数分解n

利用条件

当题目给的多对公钥n是公用了一个素数因子的时候，可以尝试公约数分解
出题一般如下

```
p1=getPrime(512)
p2=getPrime(512)
q=getPrime(512)
n1=p1*q
n2=p2*q
```

所以当题目给了多个n，并且发现n无法分解，可以尝试是否有公约数。

欧几里得辗转相除法

求公约数可以使用欧几里得辗转相除法，实现python脚本如下

```
def gcd(a, b):   #求最大公约数
    if a < b:
        a, b = b, a
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return a
```

用例

```
def gcd(a, b):   #求最大公约数
    if a < b:
        a, b = b, a
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return a

n1=0x6c9fb4bf11344e4c818be178e3d3db352797099f929e4ba8fa86d9c4ce3d8f71e3daa8c795b67dc2dabe1e1608836904386c364ecec759c27ea
n2=0x46733cc071bdee0d178fb32836a6b0a2f145a681df47d31ea9d9fc5b5fa0cc7ddbcd34531aefeace9840fc890f7a111f73593c9a41886b9a6f9
print(gcd(n1,n2))
```

使用欧几里得辗转相除得到共有的因子，然后n1和n2除以这个因子，即可得到另一个素数因子。

0x05 模数分解

场景

已知e,d,n求p,q
例如

```
('d=', '0x455e1c421b78f536ec24e4a797b5be78df09d8d9e3b7f4e2244138a7583e810adf6ad056bb59a91300c9ead5ed77ea6bafdebf7ab2d9ec
('e=', '0x10001')
('n=', '0x71ee0f4883690893ab503e97e25e6308d4c1e0a050cbea7b9c040f7a5b5b484afcecc8a9b3cc6bf089a1e83281562df217caab7220e3df
```

模数分解

私钥d的获取是通过

```
d = gmpy2.invert(e, (p-1)*(q-1))
```

分解p,q python实现如下

```
import random
def gcd(a, b):
   if a < b:
     a, b = b, a
   while b != 0:
     temp = a % b
     a = b
     b = temp
   return a
def getpq(n,e,d):
    p = 1
    q = 1
    while p==1 and q==1:
        k = d * e - 1
        g = random.randint ( 0 , n )
        while p==1 and q==1 and k % 2 == 0:
            k /= 2
            y = pow(g,k,n)
            if y!=1 and gcd(y-1,n)>1:
                p = gcd(y-1,n)
                q = n/p
    return p,q
```

完整用例

```
import random
def gcd(a, b):
   if a < b:
     a, b = b, a
   while b != 0:
     temp = a % b
     a = b
     b = temp
   return a
def getpq(n,e,d):
    p = 1
    q = 1
    while p==1 and q==1:
        k = d * e - 1
        g = random.randint ( 0 , n )
        while p==1 and q==1 and k % 2 == 0:
            k /= 2
            y = pow(g,k,n)
            if y!=1 and gcd(y-1,n)>1:
                p = gcd(y-1,n)
                q = n/p
    return p,q

n=0x71ee0f4883690893ab503e97e25e6308d4c1e0a050cbea7b9c040f7a5b5b484afcecc8a9b3cc6bf089a1e83281562df217caab7220e3dfc14399
e=0x10001
d=0x455e1c421b78f536ec24e4a797b5be78df09d8d9e3b7f4e2244138a7583e810adf6ad056bb59a91300c9ead5ed77ea6bafdebf7ab2d9ec200127
p,q=getpq(n,e,d)
print("p=",p)
print("q=",q)
print(p*q==n)
```

0x06 dp&dq泄露

介绍

首先了解一下什么是dp、dq

```
dp=d%(p-1)
dq=d%(q-1)
```

这种参数是为了让解密的时候更快速产生的

场景

假设题目仅给出p，q，dp，dq，c，即不给公钥e

```
('p=', '0xf85d730bbf09033a75379e58a8465f8048b8516f8105ce2879ce774241305b6eb4ea506b61eb7e376d4fcd425c76e80cb748ebfaf3a852
('q=', '0xc1f34b4f826f91c5d68c5751c9af830bc770467a68699991be6e847c29c13170110ccd5e855710950abab2694b6ac730141152758acbec
('dp=', '0xf7b885a246a59fa1b3fe88a2971cb1ee8b19c4a7f9c1a791b9845471320220803854a967a1a03820e297c0fc1aabc2e1c40228d502287
('dq=', '0x865fe807b8595067ff93d053cc269be6a75134a34e800b741cba39744501a31cffd31cdea6078267a0bd652aeaa39a49c73d9121fafdf
('c=', '0xae05e0c34e2ba4ca3536987cc2cfc3f1f7f53190164d0ac50b44832f0e7224c6fdeebd2c91e3991e7d179c26b1b997295dc9724925ba43
```

解密代码如下

```
InvQ=gmpy2.invert(q,p)
mp=pow(c,dp,p)
mq=pow(c,dq,q)
m=(((mp-mq)*InvQ)%p)*q+mq
print '{:x}'.format(m).decode('hex')
```

解题完整脚本

```
import gmpy2
import binascii
def decrypt(dp,dq,p,q,c):
    InvQ = gmpy2.invert(q,p)
    mp = pow(c,dp,p)
    mq = pow(c,dq,q)
    m=(((mp-mq)*InvQ)%p)*q+mq
    print (binascii.unhexlify(hex(m)[2:]))

p=0xf85d730bbf09033a75379e58a8465f8048b8516f8105ce2879ce774241305b6eb4ea506b61eb7e376d4fcd425c76e80cb748ebfaf3a852b5cf31
q=0xc1f34b4f826f91c5d68c5751c9af830bc770467a68699991be6e847c29c13170110ccd5e855710950abab2694b6ac730141152758acbeca0c5a5
dp=0xf7b885a246a59fa1b3fe88a2971cb1ee8b19c4a7f9c1a791b9845471320220803854a967a1a03820e297c0fc1aabc2e1c40228d50228766ebeb
dq=0x865fe807b8595067ff93d053cc269be6a75134a34e800b741cba39744501a31cffd31cdea6078267a0bd652aeaa39a49c73d9121fafdfa7e113
c=0xae05e0c34e2ba4ca3536987cc2cfc3f1f7f53190164d0ac50b44832f0e7224c6fdeebd2c91e3991e7d179c26b1b997295dc9724925ba431f527f
decrypt(dp,dq,p,q,c)
```

0x07 dp泄露

场景介绍

假设题目给出公钥n,e以及dp

```
('dp=', '0x7f1344a0b8d2858492aaf88d692b32c23ef0d2745595bc5fe68de384b61c03e8fd054232f2986f8b279a0105b7bee85f74378c7f5f35c
('n=', '0x5eee1b4b4f17912274b7427d8dc0c274dc96baa72e43da36ff39d452ff6f2ef0dc6bf7eb9bdab899a6bb718c070687feff517fcf537743
('e=', '0x10001')
('c=', '0x510fd8c3f6e21dfc0764a352a2c7ff1e604e1681a3867480a070a480f722e2f4a63ca3d7a92b862955ab4be76cde43b51576a128fba493
```

给出密文要求解明文
我们可以通过n，e，dp求解私钥d

求解公式推导

公式推导参考简书
https://www.jianshu.com/p/74270dc7a14b
首先dp是

```
dp=d%(p-1)
```

以下推导过程如果有问题欢迎指正
现在我们可以知道的是

```
c≡m^e mod n
m≡c^d mod n
φ(n)=(p−1)*(q−1)
d*e≡1 mod φ(n)
dp≡d mod (p−1)
```

由上式可以得到

```
dp*e≡d*e mod (p−1)
```

因此可以得到

```
d*e=k*(p−1)+dp*ed*e≡1 mod φ(n)
```

我们将式1带入式2可以得到

```
k*(p−1)+dp*e≡1 mod (p−1)*(q−1)
```

故此可以得到

```
k2*(p−1)*(q−1)+1=k1*(p−1)+dp*e
```

变换一下

```
(p−1)*[k2*(q−1)−k1]+1=dp*e
```

因为

```
dp<p-1
```

可以得到

```
e>k2*(q-1)-k1
```

我们假设

```
x=k2*(q-1)-k1
```

可以得到x的范围为

```
(0,e)
```

因此有

```
x*(p-1)+1=dp*e
```

那么我们可以遍历

```
x∈(0,e)
```

求出p-1，求的方法也很简单，遍历65537种可能，其中肯定有一个p可以被n整除那么求出p和q，即可利用

```
φ(n)=(p-1)*(q-1)d*e≡1 mod φ(n)
```

推出

```
d≡1*e-1 mod φ(n)
```

注：这里的-1为逆元，不是倒数的那个-1

公式的python实现

求解私钥d脚本如下

```
def getd(n,e,dp):
    for i in range(1,e):
        if (dp*e-1)%i == 0:
            if n%(((dp*e-1)/i)+1)==0:
                p=((dp*e-1)/i)+1
                q=n/(((dp*e-1)/i)+1)
                phi = (p-1)*(q-1)
                d = gmpy2.invert(e,phi)%phi
                return d
```

解题完整脚本

```
import gmpy2
import binascii

def getd(n,e,dp):
    for i in range(1,e):
        if (dp*e-1)%i == 0:
            if n%(((dp*e-1)/i)+1)==0:
                p=((dp*e-1)/i)+1
                q=n/(((dp*e-1)/i)+1)
                phi = (p-1)*(q-1)
                d = gmpy2.invert(e,phi)%phi
                return d

dp=0x7f1344a0b8d2858492aaf88d692b32c23ef0d2745595bc5fe68de384b61c03e8fd054232f2986f8b279a0105b7bee85f74378c7f5f35c3fd505
n=0x5eee1b4b4f17912274b7427d8dc0c274dc96baa72e43da36ff39d452ff6f2ef0dc6bf7eb9bdab899a6bb718c070687feff517fcf5377435c56c2
e=0x10001
c=0x510fd8c3f6e21dfc0764a352a2c7ff1e604e1681a3867480a070a480f722e2f4a63ca3d7a92b862955ab4be76cde43b51576a128fba49348af7a



d=getd(n,e,dp)
m=pow(c,d,n)
print (binascii.unhexlify(hex(m)[2:]))
```

0x08 e与φ(n)不互素

场景介绍

假设题目给出两组公钥n,e以及第一组、第二组加密后的密文

```
('n1=', '0xbf510b8e2b169fbce366eb15a4f6c71b370f02f2108c7feb482234a386185bce1a740fa6498e04edbdf2a639e320619d9f39d3e740eba
('e1=', '0x15d6439c6')
('c1=', '0x43e5cc4c99c3040aef2ccb0d4c45266f6b75cd7f9f1be105766689283f0886061c9cd52ac2b2b6c1b7d250c2079f354ca9b988db55563
('n2=', '0xba85d38d1bfc3fb281927c9246b5b771ac3344ca9fe1c2d9c793a886bffb5c84558f4a578cd5ba9e777a4e08f66d0cabe05b9aa2ae8d0
('e2=', '0x2c09848c6')
('c2=', '0x79ec6350649377f69b475eca83a7d9d5356a1d62e29933e9c8e2b19b4b23626a581037aba3be6d7f73d5bed049350e41c1ed4cdc3e10e
```

首先用公约数分解可以分解得到n1、n2的因子
但是发现e和φ(n)是不互为素数的，所以我们无法求出私钥d。

解题公式推导

```
gcd(e1,(p-1)*(q1-1))
gcd(e2,(p-1)*(q2-1))
```

得到结果为79858
也就是说，e和φ(n)不互素且具有公约数79858

1、首先我们发现n1、n2可以用公约数分解出p、q
但是由于e与φ(n)不互素，所以我们无法求解得到私钥d
只有当他们互素时，才能保证e的逆元d唯一存在。
公式推导过程参考博客
https://blog.csdn.net/chenzzhenguo/article/details/94339659

2、下面进行等式运算，来找到解题思路
还是要求逆元，则要找到与φ(n)互素的数

$$gcd(e,\varphi(n)) = b$$

$$ed \equiv 1 mod\ \varphi(n)$$

$$e = a * b$$

$$abd \equiv 1\ mod\ \varphi(n)$$

$$mab \equiv c\ mod\ n$$

$$cbd \equiv mabbd\ \equiv mb\ mod\ n$$

我们已知b=79858
从上面的推算，可得a与φ(n)互素，于是可唯一确定bd
于是求出bd
gmpy2.invert(a,φ(n))
然后想到bd/b，求出d，然后求明文。可是，经测试求出的是乱码，这个d不是我们想要的

3、想一下，给两组数据，应该有两组数据的作用，据上面的结论，我们可以得到一个同余式组

$$res_1 \equiv m^{79858}\ mod\ n_1$$
$$res_2 \equiv m^{79858}\ mod\ n_2$$

进一步推导

$$res1 \equiv m^{79858} \bmod p$$
$$res1 \equiv m^{79858} \bmod q_1$$
$$res2 \equiv m^{79858} \bmod q_2$$

可以计算出特解m
m=solve_crt([m1,m2,m3], [q1,q2,p])
我们想到模n1,n2不行那模q1*q2呢，
这里res可取特值m

$$res \equiv m^{79858} \bmod q_1q_2$$

那么问题就转化为求一个新的rsa题目
e=79858，经计算发现此时e与φ(n)=(q1-1)(q2-1)，还是有公因数2。
那么，我们参照上述思路，可得出m^2，此时直接对m开方即可。

$$c \equiv m^e \bmod q_1q_2$$

$$e = 2*39929$$

$$2*39929*d \equiv 1 \bmod q_1q_2$$

$$m^2 \equiv c^{2d} \bmod q_1q_2$$

完整解题脚本

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import gmpy2
import binascii


def gcd(a, b):
    if a < b:
        a, b = b, a
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return a

n1=0xbf510b8e2b169fbce366eb15a4f6c71b370f02f2108c7feb482234a386185bce1a740fa6498e04edbdf2a639e320619d9f39d3e740ebaf578af
n2=0xba85d38d1bfc3fb281927c9246b5b771ac3344ca9fe1c2d9c793a886bffb5c84558f4a578cd5ba9e777a4e08f66d0cabe05b9aa2ae8d075778b

p=gcd(n1,n2)
q1=n1//p
```

```
q2=n2//p



c1=0x43e5cc4c99c3040aef2ccb0d4c45266f6b75cd7f9f1be105766689283f0886061c9cd52ac2b2b6c1b7d250c2079f354ca9b988db5556336201f
c2=0x79ec6350649377f69b475eca83a7d9d5356a1d62e29933e9c8e2b19b4b23626a581037aba3be6d7f73d5bed049350e41c1ed4cdc3e10ee34ec5
e1 =0x15d6439c6
e2 =0x2c09848c6

#print(gcd(e1,(p-1)*(q1-1)))
#print(gcd(e2,(p-1)*(q2-1)))


e1=e1//gcd(e1,(p-1)*(q1-1))
e2=e2//gcd(e2,(p-1)*(q2-1))


phi1=(p-1)*(q1-1);phi2=(p-1)*(q2-1)
d1=gmpy2.invert(e1,phi1)
d2=gmpy2.invert(e2,phi2)
f1=pow(c1,d1,n1)
f2=pow(c2,d2,n2)


def GCRT(mi, ai):
    curm, cura = mi[0], ai[0]
    for (m, a) in zip(mi[1:], ai[1:]):
        d = gmpy2.gcd(curm, m)
        c = a - cura
        K = c // d * gmpy2.invert(curm // d, m // d)
        cura += curm * K
        curm = curm * m // d
        cura %= curm
    return (cura % curm, curm)


f3,lcm = GCRT([n1,n2],[f1,f2])
n3=q1*q2
c3=f3%n3
phi3=(q1-1)*(q2-1)

d3=gmpy2.invert(39929,phi3)#39929是79858//gcd((q1-1)*(q2-1),79858) 因为新的e和ϕ(n)还是有公因数2
m3=pow(c3,d3,n3)

if gmpy2.iroot(m3,2)[1] == 1:
    flag=gmpy2.iroot(m3,2)[0]
    print(binascii.unhexlify(hex(flag)[2:].strip("L")))
```

0x09 公钥n由多个素数因子组成

场景介绍

题目如下

```
('n=', '0xf1b234e8a03408df4868015d654dcb931f038ef4fc0be8658c9b951ee6c60d23689a1bfb151e74df0910fa1cf8a542282a65')
('e=', '0x10001')
('c=', '0x22fda6137013bac19754f78e8d9658498017f05a4b0814f2af97dc2c60fdc433d2949ea27b13337961ef3c4cf27452ad3c95')
```

因为这题的公钥n是由四个素数相乘得来的，

其中四个素数的值相差较小，或者较大，都会造成n更容易分解的结果
例如出题如下

```
p=getPrime(100)
q=gmpy2.next_prime(p)
r=gmpy2.next_prime(q)
s=gmpy2.next_prime(r)
n=p*q*r*s
```

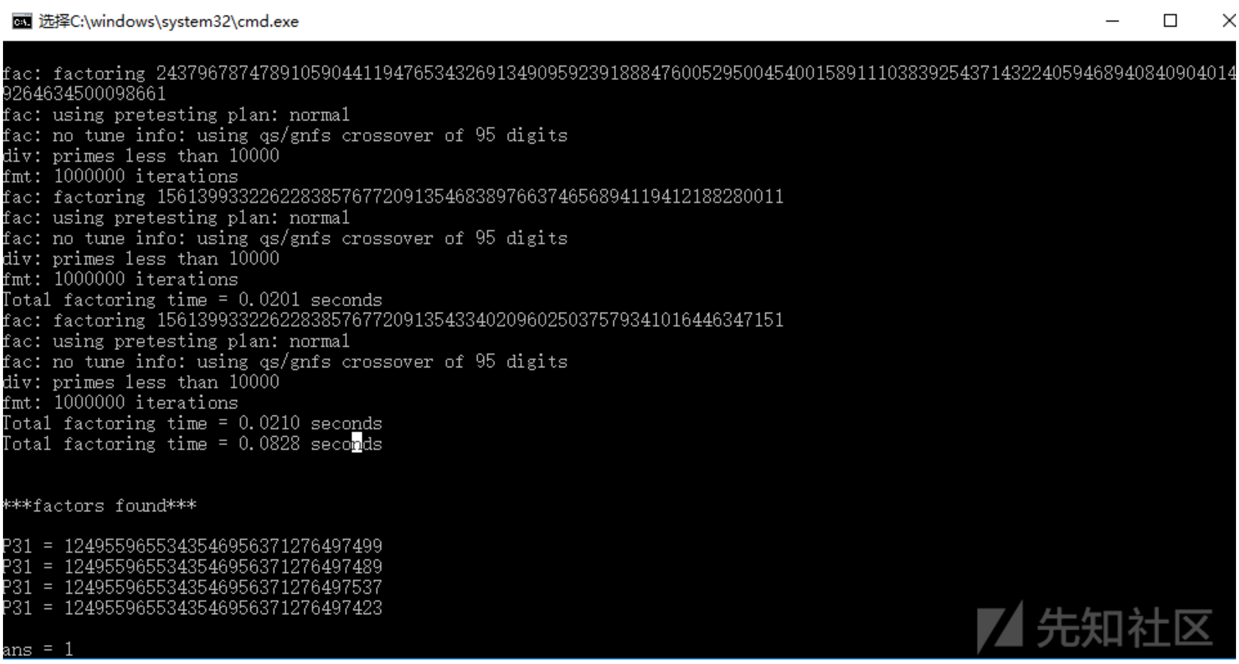
因为p、q、r、s十分接近，所以可以使用yafu直接分解

yafu分解

使用

```
factor(*)
```

括号中为要分解的数



公钥n由多素数相乘解题脚本

```
import binascii
import gmpy2
p=1249559655343546956371276497499
q=1249559655343546956371276497489
r=1249559655343546956371276497537
s=1249559655343546956371276497423
e=0x10001
c=0x22fda6137013bac19754f78e8d9658498017f05a4b0814f2af97dc2c60fdc433d2949ea27b13337961ef3c4cf27452ad3c95
n=p*q*r*s

phi=(p-1)*(q-1)*(r-1)*(s-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

0x10 小明文攻击

小明文攻击是基于低加密指数的，主要分成两种情况。

明文过小，导致明文的e次方仍然小于n

```
('n=', '0xad03794ef170d81aad370dccb7b92af7d174c10e0ae9ddc99b7dc5f93af6c65b51cc9c40941b002c7633caf8cd50e1b73aa942c8488d46
('e=', '0x3')
('c=', '0x10652cdf6f422470ea251f77L')
```

这种情况直接对密文e次开方，即可得到明文

解题脚本

```
import binascii
import gmpy2
n=0xad03794ef170d81aad370dccb7b92af7d174c10e0ae9ddc99b7dc5f93af6c65b51cc9c40941b002c7633caf8cd50e1b73aa942c8488d46c00320
e=0x3
c=0x10652cdf6f422470ea251f77

m=gmpy2.iroot(c, 3)[0]
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

明文的三次方虽然比n大，但是大不了多少

```
('n=', '0x9683f5f8073b6cd9df96ee4dbe6629c7965e1edd2854afa113d80c44f5dfcf030a18c1b2ff40575fe8e222230d7bb5b6dd8c419c9d4bca
('e=', '0x3')
('c=', '0x8541ee560f77d8fe536d48eab425b0505e86178e6ffefa1b0c37ccbfc6cb5f9a7727baeb3916356d6fce3205cd4e586a1cc407703b3f70
```

爆破即可，每次加上一个n

```
i = 0
while 1:
    res = iroot(c+i*n,3)
    if(res[1] == True):
        print res
        break
    print "i="+str(i)
    i = i+1
```

完整脚本

```
import binascii
import gmpy2

n=0x9683f5f8073b6cd9df96ee4dbe6629c7965e1edd2854afa113d80c44f5dfcf030a18c1b2ff40575fe8e222230d7bb5b6dd8c419c9d4bca1a7e84
e=0x3
c=0x8541ee560f77d8fe536d48eab425b0505e86178e6ffefa1b0c37ccbfc6cb5f9a7727baeb3916356d6fce3205cd4e586a1cc407703b3f709e2011

i = 0
while 1:
    res = gmpy2.iroot(c+i*n,3)
    if(res[1] == True):
        m=res[0]
        print(binascii.unhexlify(hex(m)[2:].strip("L")))
        break
    print "i="+str(i)
    i = i+1
```

0x11 低加密指数广播攻击

场景介绍

如果选取的加密指数较低，并且使用了相同的加密指数给一个接受者的群发送相同的信息，那么可以进行广播攻击得到明文。
这个识别起来比较简单，一般来说都是给了三组加密的参数和明密文，其中题目很明确地能告诉你这三组的明文都是一样的，并且e都取了一个较小的数字。

```
('n=', '0x683fe30746a91545a45225e063e8dc64d26dbf98c75658a38a7c9dfd16dd38236c7aae7de5cbbf67056c9c57817fd3da79dc4955217f43
('n=', '0xa39292e6ad271bb6a2d1345940dfab8001a53d28bc7468f285d2873d784004c2653549c589dae91c6d8238977ff1c4bea4f17d424a0fc4
('n=', '0x52c32366d84d34564a5fdc1650fc401c41ad2a63a2d6ef57c32c7887bb25da9d42c0acfb887c6334c938839c9a43aca93b2c7468915d18
```

解题脚本

```
import binascii,gmpy2

n =  [
0x683fe30746a91545a45225e063e8dc64d26dbf98c75658a38a7c9dfd16dd38236c7aae7de5cbbf67056c9c57817fd3da79dc4955217f43caefde3b
0xa39292e6ad271bb6a2d1345940dfab8001a53d28bc7468f285d2873d784004c2653549c589dae91c6d8238977ff1c4bea4f17d424a0fc4d5587661
0x52c32366d84d34564a5fdc1650fc401c41ad2a63a2d6ef57c32c7887bb25da9d42c0acfb887c6334c938839c9a43aca93b2c7468915d1846576f92
]
c =  [
0x673c72ace143441c07cba491074163c003f1a550eab56b1255e5ea9fa2bbd68fd6a9ccb48db9fd66d5dfc6a55c79cad3d9de53f700a1e3c2a29731
0x6111357d180d966a495f38566ebe4ea51fa0d54159b22bbd443cde9387687d87c08638483b39221883453a5ad09f6a0e3726b214e8e333037d178a
0x26cd2225c0229b6a3f1d1d685e53d114aa3d792737d040fbc14189336ac12fb780872792b0c0b259847badffd1427897ede0d60247aa5e79633f27
]
def CRT(mi, ai):
    assert(reduce(gmpy2.gcd,mi)==1)
    assert (isinstance(mi, list) and isinstance(ai, list))
    M = reduce(lambda x, y: x * y, mi)
    ai_ti_Mi = [a * (M / m) * gmpy2.invert(M / m, m) for (m, a) in zip(mi, ai)]
    return reduce(lambda x, y: x + y, ai_ti_Mi) % M
e=0x7
m=gmpy2.iroot(CRT(n, c), e)[0]
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

0x12 低解密指数攻击

场景介绍

主要利用的是私钥d很小，表现形式一般是e很大

```
n = 924760662352384777269895316161645566482186718357121805697009975130168220512311571608948679983744739792530888797677759
e = 27587468384672288862881213094354358587433516035212531881921186101712498639965289973292625430363076074737388345935775
```

攻击脚本

github上有开源的攻击代码https://github.com/pablocelayes/rsa-wiener-attack

求解得到私钥d

```
def rational_to_contfrac (x, y):
    '''
    Converts a rational x/y fraction into
    a list of partial quotients [a0, ..., an]
    '''
    a = x//y
    if a * y == x:
        return [a]
    else:
        pquotients = rational_to_contfrac(y, x - a * y)
        pquotients.insert(0, a)
        return pquotients
def convergents_from_contfrac(frac):
    '''
    computes the list of convergents
    using the list of partial quotients
    '''
    convs = [];
    for i in range(len(frac)):
        convs.append(contfrac_to_rational(frac[0:i]))
    return convs

def contfrac_to_rational (frac):
    '''Converts a finite continued fraction [a0, ..., an]
     to an x/y rational.
     '''
    if len(frac) == 0:
        return (0,1)
    elif len(frac) == 1:
        return (frac[0], 1)
    else:
        remainder = frac[1:len(frac)]
        (num, denom) = contfrac_to_rational(remainder)
        # fraction is now frac[0] + 1/(num/denom), which is
        # frac[0] + denom/num.
        return (frac[0] * num + denom, num)

def egcd(a,b):
    '''
    Extended Euclidean Algorithm
    returns x, y, gcd(a,b) such that ax + by = gcd(a,b)
    '''
    u, u1 = 1, 0
    v, v1 = 0, 1
```

```
    while b:
        q = a // b
        u, u1 = u1, u - q * u1
        v, v1 = v1, v - q * v1
        a, b = b, a - q * b
    return u, v, a

def gcd(a,b):
    '''
    2.8 times faster than egcd(a,b)[2]
    '''
    a,b=(b,a) if a<b else (a,b)
    while b:
        a,b=b,a%b
    return a

def modInverse(e,n):
    '''
    d such that de = 1 (mod n)
    e must be coprime to n
    this is assumed to be true
    '''
    return egcd(e,n)[0]%n

def totient(p,q):
    '''
    Calculates the totient of pq
    '''
    return (p-1)*(q-1)

def bitlength(x):
    '''
    Calculates the bitlength of x
    '''
    assert x >= 0
    n = 0
    while x > 0:
        n = n+1
        x = x>>1
    return n


def isqrt(n):
    '''
    Calculates the integer square root
    for arbitrary large nonnegative integers
    '''
    if n < 0:
        raise ValueError('square root not defined for negative numbers')

    if n == 0:
        return 0
    a, b = divmod(bitlength(n), 2)
    x = 2**(a+b)
    while True:
        y = (x + n//x)//2
        if y >= x:
            return x
        x = y


def is_perfect_square(n):
    '''
    If n is a perfect square it returns sqrt(n),
```

```
    otherwise returns -1
    '''
    h = n & 0xF; #last hexadecimal "digit"

    if h > 9:
        return -1 # return immediately in 6 cases out of 16.

    # Take advantage of Boolean short-circuit evaluation
    if ( h != 2 and h != 3 and h != 5 and h != 6 and h != 7 and h != 8 ):
        # take square root if you must
        t = isqrt(n)
        if t*t == n:
            return t
        else:
            return -1

    return -1

def hack_RSA(e,n):
    frac = rational_to_contfrac(e, n)
    convergents = convergents_from_contfrac(frac)

    for (k,d) in convergents:
        #check if d is actually the key
        if k!=0 and (e*d-1)%k == 0:
            phi = (e*d-1)//k
            s = n - phi + 1
            # check if the equation x^2 - s*x + n = 0
            # has integer roots
            discr = s*s - 4*n
            if(discr>=0):
                t = is_perfect_square(discr)
                if t!=-1 and (s+t)%2==0:
                    print("\nHacked!")
                    return d

def main():
    n = 92476066235238477726989531616164556648218671835712180569700997513016822051231157160894867998374473979253088879
    e = 27587468384672288862881213094354358587433516035212531881921186101712498639965289973292625430363076074737388345
    d=hack_RSA(e,n)
    print ("d=")
    print (d)

if __name__ == '__main__':
    main()
```

0x13 共模攻击

场景介绍

识别：若干次加密，e不同，n相同，m相同。就可以在不分解n和求d的前提下，解出明文m。

```
('n=', '0xc42b9d872f8ecf90b4832199771bbd8d9bafb213747d905a644baa42144f316dc224e7914f8a5d361eeab930adf5ea7fbe1416e58b3fae
('e1=', '0xc21000af014a98b2455dec479L')
('e2=', '0x9935842d63b75899ddd81b467L')
('c1=', '0xc0204d515a275954bbc8390d80efa1cca3bb29724ed7ba18f861913e28b6400298603b920d484284ad9c1c175587496300355395cb06b
('c2=', '0xc4053ed3455c15174e5699ab6eb09b830a98b79e92e7518b713e828faca4d6d02306a65a8ec70893ca8a56943a7074e6de8649f099164
```

推导过程

```
首先，两个加密指数互质：
gcd(e1,e2)=1

即存在s1、s2使得：
s1+*e1+s2*e2=1

又因为：
c1≡m^e1 mod n
c2≡m mod n

代入化简可得：
c1^s1 * c2^s2 ≡ m mod n

即可求出明文
```

公式的python实现如下

```
def egcd(a, b):
    if a == 0:
      return (b, 0, 1)
    else:
      g, y, x = egcd(b % a, a)
      return (g, x - (b // a) * y, y)
def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
      raise Exception('modular inverse does not exist')
    else:
      return x % m
s = egcd(e1, e2)
s1 = s[1]
s2 = s[2]
if s1<0:
   s1 = - s1
   c1 = modinv(c1, n)
elif s2<0:
   s2 = - s2
   c2 = modinv(c2, n)
m=(pow(c1,s1,n)*pow(c2,s2,n)) % n
```

完整解题脚本

```
import sys
import binascii
sys.setrecursionlimit(1000000)
def egcd(a, b):
    if a == 0:
      return (b, 0, 1)
    else:
      g, y, x = egcd(b % a, a)
      return (g, x - (b // a) * y, y)
def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
      raise Exception('modular inverse does not exist')
    else:
      return x % m

c1=0xc0204d515a275954bbc8390d80efa1cca3bb29724ed7ba18f861913e28b6400298603b920d484284ad9c1c175587496300355395cb06b32603e
n=0xc42b9d872f8ecf90b4832199771bbd8d9bafb213747d905a644baa42144f316dc224e7914f8a5d361eeab930adf5ea7fbe1416e58b3fae34ca7e
e1=0xc21000af014a98b2455dec479
c2=0xc4053ed3455c15174e5699ab6eb09b830a98b79e92e7518b713e828faca4d6d02306a65a8ec70893ca8a56943a7074e6de8649f099164cad33b
e2=0x9935842d63b75899ddd81b467

s = egcd(e1, e2)
s1 = s[1]
s2 = s[2]

if s1<0:
   s1 = - s1
   c1 = modinv(c1, n)
elif s2<0:
   s2 = - s2
   c2 = modinv(c2, n)
m=(pow(c1,s1,n)*pow(c2,s2,n)) % n
print(m)
print (binascii.unhexlify(hex(m)[2:].strip("L")))
```

0x14 Stereotyped messages攻击

场景介绍

```
('n=', '0xf85539597ee444f3fcad07142ecf6eaae5320301244a7cedc50b2beed7e60ffa11ccf28c1a590fb81346fb16b0cecd046a1f63f0bf9318
('e=', '0x3')
('c=', '0xa75c3c8a19ed9c911d851917e442a8e7b425e4b7f92205ca532a2ab0f5abe6cb86d164cc61374877f9e88e7bca606b43c79f1d59deadfc
('m=', '0x666c6167206973203a7465737431323131313131313131313131313334353600000000000000000L')
```

给了明文的高位，可以尝试使用Stereotyped messages攻击
我们需要使用sage实现该算法
可以安装SageMath
或者在线网站https://sagecell.sagemath.org/

攻击脚本

```
e = 0x3
b=0x666c6167206973203a7465737431323131313131313131313131313334353600000000000000000
n = 0xf85539597ee444f3fcad07142ecf6eaae5320301244a7cedc50b2beed7e60ffa11ccf28c1a590fb81346fb16b0cecd046a1f63f0bf93185c10
c=0xa75c3c8a19ed9c911d851917e442a8e7b425e4b7f92205ca532a2ab0f5abe6cb86d164cc61374877f9e88e7bca606b43c79f1d59deadfcc68c3d
kbits=72
PR.<x> = PolynomialRing(Zmod(n))
f = (x + b)^e-c
x0 = f.small_roots(X=2^kbits, beta=1)[0]
print "x: %s" %hex(int(x0))
```

可以求解出m的低位

0x15 Factoring with high bits known攻击

场景介绍

```
('n=', '0xb50193dc86a450971312d72cc8794a1d3f4977bcd1584a20c31350ac70365644074c0fb50b090f38d39beb366babd784d6555d6de3be54
('p=', '0xd7e990dec6585656512c841ac932edaf048184bac5ebf9967000000000000000000L')
('e=', '0x3')
('c=', '0x428a95e5712e8aa22f6d4c39ee5ec85f422608c2f141abf22799c1860a5e343068ab55dfb5c99a7085714f4ce8950e85d8ed0a11fce351
```

题目给出p的高位

攻击脚本

该后门算法依赖于Coppersmith partial information attack算法, sage实现该算法

```
p = 0xd7e990dec6585656512c841ac932edaf048184bac5ebf9967000000000000000000
n = 0xb50193dc86a450971312d72cc8794a1d3f4977bcd1584a20c31350ac70365644074c0fb50b090f38d39beb366babd784d6555d6de3be54dad3

kbits = 60
PR.<x> = PolynomialRing(Zmod(n))
f = x + p
x0 = f.small_roots(X=2^kbits, beta=0.4)[0]
print "x: %s" %hex(int(x0))
p = p+x0
print "p: ", hex(int(p))
assert n % p == 0
q = n/int(p)
print "q: ", hex(int(q))
```

其中kbit是未知的p的低位位数
x0为求出来的p低位

0x16 Partial Key Exposure Attack

场景介绍

```
('n=', '0x56a8f8cbc72ff68e67c72718bd16d7e98150aea08780f6c4f532d20ca3c92a0fb07c959e008cbcbeac744854bc4203eb9b2996e9cf6301
('d&((1<<256)-1)=', '0x594b6c9631c4987f588399f22466b51fc48ed449b8aae0309b5736ef0b741893')
('e=', '0x3')
('c=', '0xca2841cbc52c8307e0f2c48f8b14bc0846ece4111453362e6aee4b81f44f2a14df1c58836d4937f3b868148140ee36e9a7e910dd84c2dc
```

题目给出一组公钥n,e以及加密后的密文
给私钥d的低位

攻击脚本

记N=pq为n比特RSA模数,e和d分别为加解密指数,ν为p和q低位相同的比特数,即p≡qmod2ν且p≠qmod2ν+1.
1998年,Boneh、Durfee和Frankel首先提出对RSA的部分密钥泄露攻击:当ν=1,e较小且d的低n/4比特已知时,存在关于n的多项式时间算法分解N.
2001年R.Steinfeld和Y.Zheng指出,当ν较大时,对RSA的部分密钥泄露攻击实际不可行.

当ν和e均较小且解密指数d的低n/4比特已知时,存在关于n和2ν的多项式时间算法分解N.

```
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 xrange(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 =0x56a8f8cbc72ff68e67c72718bd16d7e98150aea08780f6c4f532d20ca3c92a0fb07c959e008cbcbeac744854bc4203eb9b2996e9cf63013
    e = 0x3
    d = 0x594b6c9631c4987f588399f22466b51fc48ed449b8aae0309b5736ef0b741893
    beta = 0.5
    epsilon = beta^2/7

    nbits = n.nbits()
    kbits = 255
    d0 = d & (2^kbits-1)
    print "lower %d bits (of %d bits) is given" % (kbits, nbits)

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

kbits是私钥d泄露的位数255

0x17 Padding Attack

场景介绍

```
('n=', '0xb33aebb1834845f959e05da639776d08a344abf098080dc5de04f4cbf4a1001dL')
('e=', '0x3')
('c1=pow(hex_flag,e,n)', '0x3aa5058306947ff46b0107b062d75cf9e497cdb1f120d02eaeca30f76492c550L')
('c2=pow(hex_flag+1,e,n)', '0x6a645739f25380a5e5b263ff5e5b4b9324381f6408a11fdaab0488209145fb3eL')
```

原理参考
https://www.anquanke.com/post/id/158944

意思很简单

1.pow(mm, e) != pow(mm, e, n)
2.利用rsa加密m+padding
值得注意的是，e=3，padding可控
那么我们拥有的条件只有
n,e,c,padding
所以这里的攻击肯定是要从可控的padding入手了

攻击脚本

```
import gmpy
def getM2(a,b,c1,c2,n,e):
    a3 = pow(a,e,n)
    b3 = pow(b,e,n)
    first = c1-a3*c2+2*b3
    first = first % n
    second = e*b*(a3*c2-b3)
    second = second % n
    third = second*gmpy.invert(first,n)
    third = third % n
    fourth = (third+b)*gmpy.invert(a,n)
    return fourth % n
e=0x3
a=1
b=-1
c1=0x3aa5058306947ff46b0107b062d75cf9e497cdb1f120d02eaeca30f76492c550
c2=0x6a645739f25380a5e5b263ff5e5b4b9324381f6408a11fdaab0488209145fb3e
padding2=1
n=0xb33aebb1834845f959e05da639776d08a344abf098080dc5de04f4cbf4a1001d
m = getM2(a,b,c1,c2,n,e)-padding2
print hex(m)
```

通过上面介绍的那篇文章的推导过程我们可以知道
a等于1
b=padding1-padding2
这边我们的padding1是第一个加密的明文与明文的差，本题是0
padding2是第二个加密的明文与明文的差，本题是1
所以b是-1
我们这边是用的那篇文章的Related Message Attack

0x18 RSA LSB Oracle Attack

场景介绍

参考博客https://www.sohu.com/a/243246344_472906
适用情况：可以选择密文并泄露最低位。
在一次RSA加密中，明文为m，模数为n，加密指数为e，密文为c。
我们可以构造出c'=((2^e)c)%n=((2^e)(m^e))%n=((2m)^e)%n， 因为m的两倍可能大于n，所以经过解密得到的明文是 m'=(2m)%n 。
我们还能够知道 m' 的最低位lsb 是1还是0。
因为n是奇数，而2m是偶数，所以如果lsb 是0，说明(2m)%n 是偶数，没有超过n，即m<n/2.0，反之则m>n/2.0 。
举个例子就能明白2%3=2 是偶数，而4%3=1 是奇数。
以此类推，构造密文c"=(4^e)c)%n 使其解密后为m"=(4m)%n ，判断m" 的奇偶性可以知道m 和 n/4 的大小关系。
所以我们就有了一个二分算法，可以在对数时间内将m的范围逼近到一个足够狭窄的空间。

攻击脚本

```
def brute_flag(encrypted_flag, n, e):

    flag_count = n_count = 1
    flag_lower_bound = 0
    flag_upper_bound = n
    ciphertext = encrypted_flag
    mult = 1
    while flag_upper_bound > flag_lower_bound + 1:
        sh.recvuntil("input your option:")
        sh.sendline("D")
        ciphertext = (ciphertext * pow(2, e, n)) % n
        flag_count *= 2
        n_count = n_count * 2 - 1

        print("bit = %d" % mult)
        mult += 1


        sh.recvuntil("Your encrypted message:")
        sh.sendline(str(ciphertext))

        data=sh.recvline()[:-1]
        if(data=='The plain of your decrypted message is even!'):
            flag_upper_bound = n * n_count / flag_count
        else:
            flag_lower_bound = n * n_count / flag_count
            n_count += 1
    return flag_upper_bound
```


上一篇： windows样本高级静态分析之识...

下一篇： Webmin远程命令执行漏洞（CV...