2018 HITCON Crypto lost key

原创

Risker_GML 于 2018-10-24 21:56:21 发布 711 收藏

分类专栏： wp 文章标签： CTF Crypto

lost key

赛后学习了一下这道密码题的攻击姿势。

题目脚本如下：

```python
#!/usr/bin/env python
from Crypto.Util.number import *
from gmpy2 import *
from random import *
import sys,os


sys.stdin  = os.fdopen(sys.stdin.fileno(), 'r', 0)
sys.stdout = os.fdopen(sys.stdout.fileno(), 'w', 0)
rnd = SystemRandom()

def calcA(g,n,data):
  num = bytes_to_long(data)
  res = pow(g,num,n*n)
  r = rnd.randint(0,n-1)
  magic = pow(r,n,n*n)
  res = (res*magic)%(n*n)
  return long_to_bytes(res).encode('hex')

def calcB(phi,n,u,data):
  num = bytes_to_long(data)
  res = pow(num,phi,n*n)
  res = (res - 1)/n
  res = (res*u)%n
  return long_to_bytes(res).encode('hex')

if __name__ == '__main__':
  p = getPrime(512)
  q = getPrime(512)
  n = p*q
  phi = (p-1)*(q-1)
  g = n+1
  u = invert(phi,n)
  flag = open('flag').read()
  print 'Here is the flag!'
  print calcA(g,n,flag)
  for i in xrange(2048):
    m = raw_input('cmd: ')
    if m[0] == 'A':
      m = raw_input('input: ')
      try:
        m = m.decode('hex')
        print calcA(g,n,m)
      except:
        print 'no'
        exit(0)
    if m[0] == 'B':
      m = raw_input('input: ')
      try:
        m = m.decode('hex')
        print calcB(phi,n,u,m)[-2:]
      except:
        print 'no'
        exit(0)
```

服务器随机生成 RSA 中的 n 和 e，并将 flag 的密文发给我们。服务器提供了两个服务：对于我们给出的明文服务器返回密文；对于我们给出的密文返回解密后的明文最后一个字节。

这个题考点有两个：获取公钥和 LSB Oracle Attrack。

获取公钥

首先我们需要获得公钥，这里可以通过选择明文攻击的典型手法获取 n，原理大致如下：

我们首先发送 2 让服务器进行加密，返回 **$c1=2^e \bmod n$**
继续发送 2**2，也就是 4，让服务器进行加密，返回 **$c2=4^e \bmod n$**
我们有：**$c1^2-c2=kn$**

为什么呢？不妨设 $2^e=a+bn$,我们有 **$c1=(a+bn) \bmod n=a$**,同时 **$c2=(a^2+2abn+b^2n^2) \bmod n=a^2 \bmod n$**,所以 $c1^2$ 和 c2 模 n 同余。

一般来说 a^2 会比 n 大，所以 k 不为 0。

同理我们分别发送 3 和 9，用同样的方法得到了 n 的又一个倍数,计算他们的公因数，这样大概率得到的就是 n。

为了准确我们也可以多发送几组，取他们的公因数。

获取 flag

得到了 n,我们就可以获取 flag 了。这个题提供的解密返回解密结果的最后一个字节，很像之前 Xman 选拔赛出国的经典的 LSB Oracle Attrack（不了解的可以看一下 这篇 以及 ctf-wiki 的相关推导 ）

之前的是给我们最后一个 bit 的值，也就是对应明文的奇偶，现在给我们的不仅仅是 1 bit 而是 8 bit。

如果我们考虑用同样的方法，即浪费他给的那 7 个 bit，我们会发现由于 n 是 1024 位，采用二分逼近需要 1024 次，题目连接一次只提供 150 次交互，显然条件不能浪费。

给了我们最后一字节，我们也可以采用利用最后 1bit 攻击类似的手法，详细的推导过程 iromise 师傅已经在 ctf-wiki 相关部分中写的很详细了。

下面是 exp:(我用的是 socket，pwn 师傅们可能习惯用 pwntools，可以参考 wiki 里的脚本)

```
# _*_ coding:utf-8 _*_
import socket
import gmpy2
from Crypto.Util.number import  bytes_to_long, long_to_bytes
from fractions import Fraction
import re

s = socket.socket()
s.connect(("18.179.251.168", 21700))

def enc(data):  #加密
    s.send("A"+"\n")
    s.recv(1024).strip()
    s.send(long_to_bytes(data).encode("hex")+"\n")
    result=s.recv(1024).strip()
    s.recv(1024).strip()
    return bytes_to_long(result.decode("hex"))

def dec(data):  #解密
    s.send("B" + "\n")
    s.recv(1024).strip()
    s.send(long_to_bytes(data).encode("hex") + "\n")
    result = s.recv(1024).strip()
    s.recv(1024).strip()
    return bytes_to_long(result.decode("hex"))

def recover_pubkey(): #获取公钥 n
```

```
    two = enc(2)
    three = enc(3)
    power_two = enc(2**2)
    power_three = enc(3**2)
    n = gmpy2.gcd(two ** 2 - power_two, three ** 2 - power_three)
    while n % 2 == 0:
        n = n / 2
    while n % 3 == 0:
        n = n / 3
    return n

def recover_flag(flagcipher,n): #获取 flag
    submap = {}
    for i in range(0, 256):
        submap[-n * i % 256] = i
    cipher256 = enc(256)
    last_bytes=dec(flagcipher)
    L = Fraction(0, 1)
    R = Fraction(1, 1)
    for i in range(128):
        print i
        flagcipher = flagcipher * cipher256 % n
        b=dec(flagcipher)
        k = submap[b]
        L, R = L + (R - L) * Fraction(k, 256), L + (R - L) * Fraction(k + 1, 256)
    low = int(L * n)
    return long_to_bytes(low - low % 256 +last_bytes)

def main():
    print s.recv(1024).strip()
    data=s.recv(1024).strip()
    print data
    flagcipher = bytes_to_long(re.findall('\s*(.*)\s*cmd:', data)[0].decode("hex"))
    n=recover_pubkey()
    print n
    print recover_flag(flagcipher,n)
main()
```

> hitcon{1east_4ign1f1cant_BYTE_0racle_is_m0re_pow3rfu1!}

参考：

https://ctf-wiki.github.io/ctf-wiki/crypto/asymmetric/rsa/rsa_chosen_plain_cipher

https://github.com/p4-team/ctf/tree/master/2018-10-20-hitcon/crypto_rsa