RSA Least-Significant-Bit Oracle Attack

发表于 2018-03-27 | 分类于 crypto |

此攻击方式是从rsa-least-significant-bit-oracle-attack 看到的，刚好用于Backdoor CTF的一道密码学的题目！

0x1 问题描述

假如用户知道公钥中N, e, c，并且可以任意构造密文c_1，返回此密文解密后p_1的末尾某些比特位的性质（记为函数f），求原始明文信息！

最简单的函数f是表示p_1的奇偶性。

0x2 原理

攻击者得到密文$C = P^e (mod\ n)$，将其乘以$2^e (mod\ N)$,并作为密文发送出去，若返回$f(2P)$

> 如果$f(2P)$返回的最后一位是0，那么$2P < N$，即$P < N/2$
>
> 如果$f(2P)$返回的最后一位是1，那么$2P > N$，即$P > N/2$

接着我们来看看$2P$和$4P$

> 如果返回的是（偶，偶），那么有$P < N/4$
>
> 如果返回的是（偶，奇），那么有$N/4 < P < N/2$
>
> 如果返回的是（奇，奇），那么有$3N/4 < P < N$

从这里基本上就可以找到规律了，如果我们循环下去，基本上就可以得到P所处在的空间。当次数不断叠加，最终所处在的空间将会十分的小，于是就可以解出对应的解！

0x3 方法

$P \in [0, P]$ 也即$LB = 0$，$UB = N$

使用$log_2\ N$次可以根据密文C求解出明文P

$C' = (2^e\ mod\ N) * C$

```
if (Oracle(C') == even)
    UB = (UB + LB)/2;
else
    LB = (UB + LB)/2;
```

0x4 实例

Backdoor CTF 2018 题目 BIT-LEAKER

service.py

```
#!/usr/bin/python -u
from Crypto.Util.number import *
from Crypto.PublicKey import RSA
import random
#from SECRET import flag
flag = "CTF{this_is_my_test_flag}"
m = bytes_to_long(flag)

key = RSA.generate(1024)

c = pow(m, key.e, key.n)
print("Welcome to BACKDOORCTF17\n")
print("PublicKey:\n")
print("N = " + str(key.n) + "\n")
print("e = " + str(key.e) + "\n")
print("c = " + str(c) + "\n")

while True:
    try:
        temp_c = int(raw_input("temp_c = "))
        temp_m = pow(temp_c, key.d, key.n)
    except:
        break
    l = str(((temp_m&5) * random.randint(1,10000))%(2*(random.randint(1,10000))))
    print "l = "+l
```

solve.py

```
# -*- coding: utf-8 -*-
#/usr/bin/env python

from pwn import *
import libnum
import Crypto
import re
from binascii import hexlify,unhexlify

if len(sys.argv)>1:
    p=remote("127.0.0.1",2334)
else:
    p=remote('127.0.0.1',2333)

#context.log_level = 'debug'

def oracle(c):
    l = []
    for i in range(20):
        p.sendline(str(c))
        s = p.recvuntil("temp_c")
        l.append(int(re.findall("l\s*=\s*([0-9]*)",s)[0]))
    flag0 = 0
    flag2 = 0
    for i in range(20):
        if l[i]%2 != 0:
            flag0 = 1
        if l[i] > 10000:
            flag2 = 1
    return [flag2,flag0]


def main():
    ss = p.recvuntil("temp_c")
    N = int(re.findall("N\s*=\s*(\d+)",ss)[0])
    e = int(re.findall("e\s*=\s*(\d+)",ss)[0])
    c = int(re.findall("c\s*=\s*(\d+)",ss)[0])
    size = libnum.len_in_bits(N)
    print "N=",N
    print "e=",e
    print "c=",c
    c = (pow(2,e,N)*c)%N
    LB = 0
    UB = N
    i = 1
    while LB!=UB:
        flag = oracle(c)
        print i,flag
        if flag[1]%2==0:
            UB = (LB+UB)/2
        else:
            LB = (LB+UB)/2
        c = (pow(2,e,N)*c)%N
        i += 1
    print LB
    print UB
    for i in range(-128,128,0):
        LB += i
        if pow(LB,e,N)==c:
            print unhexlify(hex(LB)[2:-1])
            exit(0)

if __name__ == '__main__':
    main()
    p.interactive()
```

这道题有个问题，就是远程比较慢，所以可能需要很长时间！

另外就是算法的正确度不能保证，所以可能需要中途断开，多跑几次！

还有就是这个算法因为都是整除运算，导致的最终结果可能有一定误差！

rsa

hexo使用hexo-math插件支持MathJax

线性分析法