2020JTWLB-个人CTF-CRYPTO-weakrsa

原创

大熊何在 于 2020-10-28 10:07:25 发布 294 收藏
分类专栏： CRYPTO

本文链接：https://blog.csdn.net/zippo1234/article/details/109326365

2020JTWLB-个人CTF-CRYPTO-weakrsa

每天一题，只能多不能少

weakrsa

题目分析

其实是昨天做的，偷个懒，今天再补上一题。
确实很weak。但是这是相对于大神而言□

1. 高位相同的m
2. 短填充攻击Coppersmith Shortpad Attack
3. FranklinReiter
4. epsilon调整

开始

1.题目

给出server.py

```
class Unbuffered(object):
    def __init__(self, stream):
        self.stream = stream

    def write(self, data):
        self.stream.write(data)
        self.stream.flush()

    def __getattr__(self, attr):
        return getattr(self.stream, attr)
import sys
sys.stdout = Unbuffered(sys.stdout)
import signal
signal.alarm(600)
import os
from Crypto.PublicKey import RSA
import random
n = 7407655489792730984675988039582123851515700539213342374301714913573084503059389253950580485715586133640029480042911355182403295727895254874951478707796193799828650113287755658500482485268633743760242969219377981697378605840470655939282958574174523727449369475448168042337019929196114881405933971591501521609206390639
e = 3
os.chdir("/home/ctf")
flag = open("flag", "rb").read().replace('\n', '')
flag_padding = "padding padding padding padding padding padding {} padding padding padding padding".format(
    flag)
c = pow(int(flag_padding.encode('hex')[
        2:], 16) + random.randint(1000, 999999), e, n)
print("c:" + str(c))
m = raw_input("> ")
if m == flag:
    print "- success"
else:
    print "- failed"
```

nc过去会反弹一个c

2.分析

server的加密过程其实很简单，flag加一些‘pading’再加一段随机数，扔进去RSA。e=3。
也就是说每次nc过去，得到的结果是同一段flag再加不一样的随机数的RSA结果。这就符合coppersmith的条件了。
大神说这题是crypto签到水平，应该就是基于这个吧，逻辑简单，有现成脚本。

3.原理

还是贴一下原理吧，避免自己是个脚本小子（虽然改变不了这个事实）。

Coppersmith’s short-pad attack

1. 攻击条件
   目前在大部分消息加密之前都会进行 padding，但是如果 padding 的长度过短，也有可能被很容易地攻击。
   这里所谓 padding 过短，其实就是对应的多项式的根会过小。

2.攻击原理
我们假设爱丽丝要给鲍勃发送消息，首先爱丽丝对要加密的消息 M 进行随机 padding，然后加密得到密文 C1，发送给鲍勃。这时，中间人皮特截获了密文。一段时间后，爱丽丝没有收到鲍勃的回复，再次对要加密的消息 M 进行随机 padding，然后加密得到密文 C2，发送给 Bob。皮特再一次截获。这时，皮特就可能可以利用如下原理解密。
这里我们假设模数 N 的长度为 k，并且 padding 的长度为

$$m = \lfloor \frac{k}{e^2} \rfloor 。$$

此外，假设要加密的消息的长度最多为 k-m 比特，padding 的方式如下

$$M_1 = 2^m M + r_1, 0 \leq r_1 \leq 2^m$$

消息 M2 的 padding 方式类似。
那么我们可以利用如下的方式来解密。
首先定义

$$g_1(x, y) = x^e - C_1 g_2(x, y) = (x + y)^e - C_2$$

其中

$$y = r_2 - r_1$$

显然这两个方程具有相同的根 M1。然后还有一系列的推导。

Known High Bits Message Attack

1. 攻击条件
   这里我们假设我们首先加密了消息 m，如下

$$C \equiv m^d \bmod N$$

   并且我们假设我们知道消息 m 的很大的一部分M_0，即

$$m = m_0 + x,$$

   ，但是我们不知道 x。那么我们就有可能通过该方法进行恢复消息。这里我们不知道的 x 其实就是多项式的根，需要满足 Coppersmith 的约束。

4.上脚本

链接: github脚本
直接套用即可，这里eps用默认的就可以了。

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/27 12:26
# @Author : A.James
# @FileName: tt3.py
# Franklin-Reiter attack against RSA.
# If two messages differ only by a known fixed difference between the two messages
# and are RSA encrypted under the same RSA modulus N
# then it is possible to recover both of them.

# Inputs are modulus, known difference, ciphertext 1, ciphertext2.
# Ciphertext 1 corresponds to smaller of the two plaintexts. (The one without the fixed difference added to it)
def franklinReiter(n,e,r,c1,c2):
```

```
    R.<X> = Zmod(n)[]
    f1 = X^e - c1
    f2 = (X + r)^e - c2
    # coefficient 0 = -m, which is what we wanted!
    return Integer(n-(compositeModulusGCD(f1,f2)).coefficients()[0])

  # GCD is not implemented for rings over composite modulus in Sage
  # so we do our own implementation. Its the exact same as standard GCD, but with
  # the polynomials monic representation
def compositeModulusGCD(a, b):
    if(b == 0):
        return a.monic()
    else:
        return compositeModulusGCD(b, a % b)

def CoppersmithShortPadAttack(e,n,C1,C2,eps=1/30):
    """
    Coppersmith's Shortpad attack!
    Figured out from: https://en.wikipedia.org/wiki/Coppersmith's_attack#Coppersmith.E2.80.99s_short-pad_attack
    """
    import binascii
    P.<x,y> = PolynomialRing(ZZ)
    ZmodN = Zmod(n)
    g1 = x^e - C1
    g2 = (x+y)^e - C2
    res = g1.resultant(g2)
    P.<y> = PolynomialRing(ZmodN)
    # Convert Multivariate Polynomial Ring to Univariate Polynomial Ring
    rres = 0
    for i in range(len(res.coefficients())):
        rres += res.coefficients()[i]*(y^(res.exponents()[i][1]))

    diff = rres.small_roots(epsilon=eps)
    recoveredM1 = franklinReiter(n,e,diff[0],C1,C2)
    print(recoveredM1)
    print("Message is the following hex, but potentially missing some zeroes in the binary from the right end")
    print(hex(recoveredM1))
    print("Message is one of:")
    for i in range(8):
        msg = hex(Integer(recoveredM1*pow(2,i)))
        if(len(msg)%2 == 1):
            msg = '0' + msg
        if(msg[:2]=='0x'):
            msg = msg[:2]
        print(binascii.unhexlify(msg))

def testCoppersmithShortPadAttack(eps=1/25):
    from Crypto.PublicKey import RSA
    import random
    import math
    import binascii
    M = "flag{This_Msg_Is_2_1337}"
    M = int(binascii.hexlify(M),16)
    e = 3
    nBitSize =  8192
    key = RSA.generate(nBitSize)
    #Give a bit of room, otherwhise the epsilon has to be tiny, and small roots will take forever
    m = int(math.floor(nBitSize/(e*e))) - 400
    assert (m < nBitSize - len(bin(M)[2:]))
```

```
    r1 = random.randint(1,pow(2,m))
    r2 = random.randint(r1,pow(2,m))
    M1 = pow(2,m)*M + r1
    M2 = pow(2,m)*M + r2
    C1 = Integer(pow(M1,e,key.n))
    C2 = Integer(pow(M2,e,key.n))
    CoppersmithShortPadAttack(e,key.n,C1,C2,eps)

def testFranklinReiter():
    p = random_prime(2^512)
    q = random_prime(2^512)
    n = p * q # 1024-bit modulus
    e = 11

    m = randint(0, n) # some message we want to recover
    r = randint(0, n) # random padding

    c1 = pow(m + 0, e, n)
    c2 = pow(m + r, e, n)
    print(m)
    recoveredM = franklinReiter(n,e,r,c1,c2)
    print(recoveredM)
    assert recoveredM==m
    print("They are equal!")
    return True

if __name__ == '__main__':
    n = 74076554897927309846759880395821238515157005392133423743017149135730845030593892539505804857155861336400
294800429113551824032957278952548749514787077961937998286501132877556585004824852686337437602429692193779816973786058404706559392829585741745237274493694754481680423370199291961148814059339715915015216092063 9
    c1 = 87062931409343127198398475902311269231919252928114078121918973331164534330616742069807018711434203169565663360460876575389736021237752186210787024416588643886638547419844656463261633445625505649793789071612148428421740762499025031346773046106155367974957859160122388009534240210400986187645459264829214801007328
    c2 = 39314871626527834153364259503300754912928993417507725398378065270212517252719716035522377356460378355051268492502714367257576757400152523023453301763545004465493317757051684404443294796094051897863669324105465432090124221691187466528630780322628208852585991345912027132291217680901406914354529957198434257608387 0
    e = 3
    CoppersmithShortPadAttack(e, n, c1, c2, eps=1 / 30)
```

5.get flag

```
Message is the following hex, but potentially missing some zeroes in the binary from the right end
616464696e6720706164696e6720706164696e6720706164696e6720706164696e6720706164696e6720706164696e6720666c61677b386263333764666652d
32d333462622d313634393863646232386566 7d20706164696e6720706164696e6720706164696e6720706164696e6720706164646d1681
Message is one of:
adding padding padding padding padding padding flag{8bc37dfe-0273-29dc-34bb-16498cdb28ef} padding padding padding paddm
```

```
flag{8bc37dfe-0273-29dc-34bb-16498cdb28ef}
```

结语

说复杂不复杂，但是你能说不复杂？□
反正我是不会。