PlaidCTF 2017: Multicast (175 pts)

Apr 23, 2017

PlaidCTF, PPP's annual capture the flag, was held this past weekend. Because of Google Games and other work, I didn't spend a large amount of time on it, but I and others on TechSec did manage to solve a few of the challenges.

Multicast was one of the earliest challenges released. We were given two files: a sage program called `generate.sage` and a file with 20 large integers called `data.txt`. You can find both of these files and my exploit here.

This was `generate.sage`:

```
nbits = 1024
e = 5
flag = open("flag.txt").read().strip()
assert len(flag) <= 64
m = Integer(int(flag.encode('hex'),16))
out = open("data.txt","w")

for i in range(e):
  while True:
    p = random_prime(2^floor(nbits/2)-1, lbound=2^floor(nbits/2-1), proof=False)
    q = random_prime(2^floor(nbits/2)-1, lbound=2^floor(nbits/2-1), proof=False)
    ni = p*q
    phi = (p-1)*(q-1)
    if gcd(phi, e) == 1:
      break

  while True:
    ai = randint(1,ni-1)
    if gcd(ai, ni) == 1:
      break

  bi = randint(1,ni-1)
  mi = ai*m + bi
  ci = pow(mi, e, ni)
  out.write(str(ai)+'\n')
  out.write(str(bi)+'\n')
  out.write(str(ci)+'\n')
  out.write(str(ni)+'\n')
```

Essentially, a message m is encrypted using RSA and sent to each of $e = 5$ individuals, each with a different RSA keypair. This is the setup for Håstad's Broadcast Attack, a classic attack on RSA, which (like almost every RSA attack that shows up on CTFs) is described in detail in Boneh's paper 20 Years of Attacks on the RSA Cryptosystem.

Vanilla Broadcast Attack

Given

$$c_i = m^5 \pmod{n_i}$$

for i between 1 and 5, we can compute the integer c^* such that

$$c^* = c_i \pmod{n_i}$$

for each i, using the Chinese Remainder Theorem (CRT).

CRT guarantees that

$$c^* < \prod_i n_i,$$

but since $m < n_i$ for all i, then

$$m^5 < \prod_i n_i$$

as well, which means that $c^* = m^5$. Now m can be computed as $\sqrt[5]{c^*}$ using binary search.

However, there's a twist.

In this challenge, each participant is actually given the encryption of a different message

$$m_i = a_i \cdot m + b_i$$

for randomly chosen integers a_i and b_i. Now,

$$c_i = (a_i m + b_i)^5 \pmod{n_i},$$

and `data.txt` contains the values a_i, b_i, c_i, n_i for five different participants.

This change renders the standard broadcast attack infeasible, but never fear: the Wikipedia page describes this so-called "linear-padding" variant just a few paragraphs below the original attack.

Coppersmith's Theorem

The new attack relies on a very powerful theorem from Coppersmith, which is the cornerstone of many of the more difficult attacks on RSA:

Consider a monic polynomial $p(x)$ with integer coefficients and with degree k. If there exists an $x_0 < M^{1/k}$ such that

$$p(x_0) = 0 \pmod{M},$$

then we can find x_0 in polynomial time.

The details of how this works are unimportant, and we can treat Coppersmith's method as a black box. Indeed, the small_roots function in sage will perform all the heavy lifting for us.

The New Attack

First, for each i we use CRT to compute T_i such that:

$$T_i = 1 \pmod{n_i}$$

$$T_i = 0 \pmod{n_{j \neq i}}$$

for all j. Then, we construct the polynomial:

$$g(x) = \sum_i T_i[(a_i x + b_i)^5 - c_i]$$

Note that

$$g(m) \pmod{n_i} = 0$$

for all i because either $T_j = 0 \pmod{n_i}$ for $i \neq j$, or

$$(a_i m + b_i)^5 = c_i \pmod{n_i}$$

Thus, m is a root of $g(x)$ modulo $M = \prod_i n_i$, and furthermore since $m < n_i$, we must have that $m^5 < \prod_i n_i$, or

$$m < \left(\prod_i n_i\right)^{1/5} = M^{1/\deg g(x)}.$$

Thus, we can find m using Coppersmith's method. (Actually Coppersmith's method requires that g is monic, and ours is not, but we can multiply g by the inverse of its leading coefficient without changing the value of the roots).

The Code

Sage makes this attack trivial to implement. It only required 22 lines of code:

```
import binascii

data = open('data.txt', 'r')
y = data.read().split()

y = [Integer(a) for a in y]
z = [(y[4*i + 0], y[4*i + 1], y[4*i + 2], y[4*i + 3]) for i in range(5)]

ns = [a[3] for a in z]
cs = [a[2] for a in z]
bs = [a[1] for a in z]
ass = [a[0] for a in z]

ts = [crt([int(i == j) for j in range(5)], ns) for i in range(5)]

P.<x> = PolynomialRing(Zmod(prod(ns)))
gs = [(ts[i] * ((ass[i] * x + bs[i])**5 - cs[i])) for i in range(5)]
g = sum(gs)
g = g.monic()
roots = g.small_roots()

print binascii.unhexlify(hex(int(roots[0]))[2:-1])
```

After running for about 5 seconds, our program gives us the flag:

```
[multicast]> /Applications/SageMath/sage exploit.sage
PCTF{L1ne4r_P4dd1ng_w0nt_s4ve_Y0u_fr0m_H4s7ad!}
```