

NEG9

NEGATIVE NINE SECURITY // NEWS

GOOGLE CTF 2016 - RSACALC (CRYPTO 300) WRITEUP

Hint:

Connect to `ssl-added-and-removed-here.ctfcompetition.com:59999`

Try our new Cloud Computing Service.

Note: *Encrypt and decrypt operations expect base64-encoded input.*

Connecting to the server (via SSL!) and solving a simple proof of work gives us an "RSA Calculator" that supports some basic arithmetic and Boolean operations along with encrypting and decrypting simple messages:

Usage: [operand1] operator operand2

*Supported operations: +, -, *, /, ^, |, &, *, sqrt, encrypt, decrypt*

After 8 operations or a timeout, the system will kick you out with an encrypted copy of the flag for your troubles.

Unfortunately, messages give a different encrypted value every time they're submitted suggesting that there's some sort of random padding being applied. As you would expect the previous flag can't be decrypted on subsequent connections leading to the likely conclusion that the private key changes on every connection attempt.

There's no obvious way to even get the public key parameters, until you notice that asking for certain operations (like 2^{128}) gives suspiciously low results. It turns out that all operations are being performed over a modulus, and the obvious guess is that the modulus they're using is the N parameter of for their RSA encryption.

There are several ways to get N back out from this operation, but the simplest (thanks to Meta!) is to just find a number that equals zero (modulo N). The operation $1/2$ gives a number that (although surprisingly large) when multiplied by 2 will be congruent to 1 (modulo N). So, because the calculator performs all the calculations modulo N we were able to verify that $1 == (1/2)*2$ and $0 == ((1/2)*2)-1 == N$.






Next we needed information about the padding. Requesting encryption of increasingly large plaintexts reveals that a minimum of 11 bytes of padding are mandatory (based on the 2048-bit N we discovered earlier). This is the same number of bytes as RSAES-PKCS1-v1_5 requires, which was also used in Spotted Wobbeong.

MEETINGS

- There are no upcoming meetings scheduled. Check back soon.

WITTER

LEADERBOARD

-  Javantea - 386
-  technicaltom - 379
-  CryptoMonkey - 269
-  coldwaterq - 264
-  Commod0re - 254

To finish discovering the public parameters we need e . Fortunately since we know N and a likely padding format we can simply encrypt a chosen plaintext locally and see if it decrypts correctly on the server. This quickly identifies 65537 as the proper value for e and confirms that PKCS v1.5 padding is indeed being used.

At this point the public portion of the key is fully discovered, but without the private key it doesn't seem possible to decrypt the flag...

But one of the more curious operations that rsacalc provides is `sqrt`. This allows you to reverse a quadratic residue of an arbitrary value mod N . This is easy when N is prime but computationally hard when N is a composite of two large primes...unless you happen to know and use knowledge of the component primes. In fact, it's a known equivalent to integer factorization, as described by:

- https://en.wikipedia.org/wiki/Quadratic_residue#Composite_modulus
- <http://crypto.stackexchange.com/questions/9950/quadratic-residue-problem-on-composite-integers>

The second link in particular gives you a direct recipe for calculating one particular component prime of N :

1. Choose an element ' A ' between $2 \dots N$ uniformly at random and compute $A = a^2 \pmod{N}$
2. Apply the square root computation technique on A , get result a'
3. If $a' = \pm A$ retry, else $\gcd(A - a', N)$ is a non-trivial factor of N

This generally gives us one prime factor of N in two or three attempts. Once one prime is discovered you can do simple division to get the other prime, and then just implement RSA private key derivation to discover the private key, d .

From here decrypt the flag by calculating $(m^d) \pmod{N}$, and strip off the PKCS padding to get the flag:

CTF{What.kind.of.flower.should.never.be.put.in.a.vase-Cauliflower}

And here's the automated solver in Python:

```
1 import ssl, socket, re, itertools, string, fractions, random, gmpy2
2 from hashlib import sha1
3 from base64 import b64encode, b64decode
4 from Crypto.Util.number import bytes_to_long, long_to_bytes
5
6 def proof_of_work(base):
7     base = bytes(base)
8     letters = [bytes(x) for x in string.ascii_letters]
9
10    for c in itertools.combinations_with_replacement(letters, 7):
11        work = b"".join(c)
12        digest = sha1(base + work).hexdigest()
13        if digest.startswith("00000"):
14            return work
15
16 e = 65537
17 host, port = "ssl-added-and-removed-here.ctfcompetition.com", 59999
18
19 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```

REAM)
20 s.connect((host, port))
21 sslSocket = socket.ssl(s)
22
23 #Throw away messages until you hit the proof of
work
24 data = sslSocket.read(1024)
25 data = sslSocket.read(1024)
26 proof = re.findall('proof\=(.*?)\', sha1', str(da
ta))[0]
27 print('proof is: ' + proof)
28
29 work = proof_of_work(proof)
30 print('work is: ' + str(work))
31 sslSocket.write(work + b'\n')
32
33 data = sslSocket.read(1024)
34 print data
35 if(-1 != data.find('Nope')):
36     print 'Proof of work failed.'
37     raise
38
39 #Throw away banner
40 data = sslSocket.read(1024)
41 data = sslSocket.read(1024)
42
43 sslSocket.write('1 / 2 \n')
44 data = sslSocket.read(1024)
45
46 N = (int(data) * 2) - 1
47 print 'N: ' + str(N)
48
49 p = 0
50 q = 0
51
52 while(True):
53     value = random.randint(2, N - 1)
54     A = pow(value, 2, N)
55
56     sslSocket.write('sqrt ' + str(A))
57     data = sslSocket.read(1024)
58
59     aPrime = int(data)
60
61     if(aPrime == value or aPrime == (value + N)
):
62         print "+- A satisfied. :( Retrying"
63         continue
64
65     if(fractions.gcd(value - aPrime, N) != 1):
66         p = fractions.gcd(value - aPrime, N)
67         print 'Found factor: \n' + str(p)
68         q = N / p
69         print 'q: \n' + str(q)
70
71         if(0 == ((p * q) - N)):
72             print '(p * q) == N! Private key fou
nd.'
73             break
74         else:
75             print 'Found candidate p that did no
t result in N. Retrying'
76
77 d = gmpy2.invert(e, (p-1)*(q-1))
78
79 flag = ''
80
81 while(True):
82     sslSocket.write(b'1 + 1\n')
83     data = sslSocket.read(1024)
84     if data.find("flag") != -1:
85         flag = re.findall(b'forget your flag

```

```
\!\n(.*)\n', data)[0]
86         print(b"flag: " + flag)
87         break
88
89 flagB = bytes_to_long(b64decode(flag))
90
91 decrypted = pow(flagB, d, N)
92 decryptedB = long_to_bytes(decrypted)
93
94 #Remove PKCS padding
95 for i in range(2, len(decryptedB)):
96     if('\0' == decryptedB[i]):
97         print 'Decryption: ' +
str(decryptedB[i:])
98         break
99
100 print 'Done!'
```

Posted on May 4, 2016, 12:24 a.m. by [reidb](#)

Copyright © 2010 Neg9.org. Design by [Free CSS Templates](#)
[XHTML](#) | [CSS](#)