Security Lab 4

Some examples of attacks that can be carried out against systems that rely on time via NTP are

Certificate Attacks
Authentication Attacks
DNSSEC Attacks
NTP Amplification

Certificate: Modern encryption schemes rely on certificates signed by third parties to secure communications between clients and servers. These certificates form the basis of TLS/SSL and are time stamped. If an attacker can spoof NTP and cause the client to accept a time in the past for example 2014, they can fool the client into accepting certificates that have been revoked or expired, plenty of these certificates are available online due to bugs such as heartbleed or hacking of CA authorities. NTP can also be used to perform protocol downgrade attacks by setting the date far into the future which will cause all certificates to be invalid and insecure clients may then try communicate over plain text.

Authentication Attacks: Many authentication services use time stamped tokens to negotiate access and prevent the user from having to continually re authenticate. These timestamps cover very short periods to keep the window of attack small however if an attacker can control the time of the server, they can use tokens that have been revoked or expired to be accepted.

DNSSEC: DNS Secure is a protocol to secure DNS queries to prevent the numerous DNS spoofing attacks. With control of time an attacker can easily invalidate any DNSSEC records for a domain, preventing the victim from resolving the domain names and potentially relying on insecure records.

NTP Amplification: This is a type of amplification attack where packets are deliberately spoofed with the victims IP address to NTP servers, these packets are designed to produce large outputs many times greater than the initial packet, these responses are then routed to the victim and can easily overwhelm the victim.

RSA Implementation

```
#Encoding: UTF-8
import random

def gcd(a, b):
    '''Return the greatest common divisor (gcd) of two numbers.
    Use the recursive variant of the extended Euclidean algorithm
    to compute the GCD of a and b. Also find two numbers x and y
    that satisfy ax + by = gcd(a,b).
    '''
    if b == 0:
        return (a, 1, 0)
    (d_, x_, y_) = gcd(b, a % b)
    (d, x, y) = (d_, y_, x_ - (a // b) * y_)
    return (d, x, y)

#Fermat's primality test used with base 2. This will return reliable
prime numbers up to prime<=340.
def isPrime(number):
    if ( (2 ** (number -1) ) % number ) == 1:
        return (True )# and isPrime(number,base-1))
    else:
        return False

def gen_prime(max):
    for i in xrange(max,2,-1):
        if(isPrime(i)):
            return i

def gen_privkey(p, q, e):
    ''' Return the private key exponent d.
    Compute the private exponent d using the extended Euclidean
algorithm.
    d is the multiplicative inverse of e % ((p - 1) * (q - 1)),
    so it satisfies d * e == 1 % ((p - 1) * (q - 1)).
    '''
    phi_n = (p - 1) * (q - 1)
    (x, d, y) = gcd(e, phi_n)
```

```
    if d < 0: d += phi_n
    return d

def pubkey_encrypt(text, pubkey, factor):
    return text ** pubkey % factor

def privkey_decrypt(chiffre, privkey, factor):
    return chiffre **  privkey % factor

print "Generating prime p"
prime_p = gen_prime(random.randint(10,50))
print ">>Prime p:" + str(prime_p)

print "Generating prime q"
prime_q = gen_prime(random.randint(10,50))
print ">>Prime q:" + str(prime_q)

assert prime_p != prime_q

print "Calculating factor n"
factor_n = prime_p*prime_q
print ">>Factor n:" + str(factor_n)

print "Calculating Phi(N)"
phi_n = (prime_p-1)*(prime_q-1)
print ">>Phi(N):" + str(phi_n)

# e doesn't have to be choosen at random. any prime number < phi_n will
work.
e = 17
print ">>Public exponent e:" + str(e)

assert e < phi_n

print "Generating pubkey d"
privkey_d = gen_privkey(prime_p, prime_q, e)
print ">>Privkey d:" + str(privkey_d)

print "Generating text"
text_clear = random.randint(0,100)
```

```
print ">>Text:" + str(text_clear)

print "Encrypting"
text_encrypted = pubkey_encrypt(text_clear, e, factor_n)
print ">>Text (encrypted):" + str(text_encrypted)

print "Decrypting"
text_decrypted = privkey_decrypt(text_encrypted, privkey_d, factor_n)
print ">>Text (decrypted):" + str(text_decrypted)
```