LosFuzzys

don't feed the bugs!

SharifCTF 7: LSB oracle (crypto 150)

A writeup by creed

- **Category:** crypto
- **Points:** 150
- **Description:**

  see the attachment http://ctf.sharif.edu/ctf7/api/download/27

Write-up

For this crypto challenge, we were given a python file detailing the encrpytion process, which was standard RSA with PKCS1 v1.5 padding.

```
#! /usr/bin/env python3

from Crypto.Cipher import PKCS1_v1_5
from Crypto.PublicKey import RSA
from Crypto.Util.number import bytes_to_long

n = -1    # get it from the provided EXE file
e = -1    # get it from the provided EXE file

flag = b'' # redacted
key = RSA.construct((n, e))
cipher = PKCS1_v1_5.new(key)
ctxt = bytes_to_long(cipher.encrypt(flag))

print(ctxt)
# output is:
# 2201077887205099886799419505257984908140690335465327695978150425602737431754769971309809434546937184700758848191008699273369652758836177602723960420
```

The second file was an executable, which gave you the public key parameters. Additionally you could ask the executable for the least significant bit of the decryption of a ciphertext. For this the executable needs to know the private parameters, and it should be possible to reverse the binary and get them, but since the binary was protected with vmprotect, it would have been quite annoying to do so.

The goal was to decrypt the ciphertext given in the `description.py` file. With a LSB oracle, like we are given, we can fully recover the plaintext. We can multiply the ciphertext by `2**e`, essentially doubling the plaintext. With the bit from the LSB oracle, we can now decide if the plaintext would have been reduced modulo `N`, when multiplied with 2. If it was not reduced, the LSB is 0, since it is an even number. If it is 1, then the even number got reduced modulo N, giving an odd number. Therfore we can now say if P is less or greater than N/2.

We can now repeat this process for 2P,4P,8P⋯, further constricting P, until we got the correct value for P. The below script executes this process. We found that the LSB oracle does not give you the correct value for the last 2 bytes. However, these are quickly found by bruteforcing them.

```
#!/usr/bin/env python2
from pwn import *
from Crypto.Util.number import long_to_bytes,bytes_to_long

n = 1203578556777954033268993258325992234600815518203519667649603868437558081566271313454647957139232716788352564228895677492302483898506438012639722231
e = 65537

C = 220107788720509988679941950525798490814069033546532769597815042560273743175476997130980943454693718470075884819100869927336965275883617760272396042

oracle_process = process(["wine", "./lsb_oracle.vmp.exe", "/decrypt"])
def oracle(CIN):
   oracle_process.recvuntil("done.\r\n")
   oracle_process.sendline(str(CIN))
   return int(oracle_process.readline().strip())


UP = n
LOW = 0

cur_C = C
for i in range(n.bit_length()):
   cur_C = (cur_C * (2**e % n)) %n
   if oracle(cur_C) == 0:
        UP = (UP + LOW)/2
    else:
        LOW = (UP + LOW)/2

pt =  long_to_bytes(UP)

for x in range(256):
    for y in range(256):
        corr_pt = pt[:-2] + chr(x) + chr(y)
        if pow(bytes_to_long(corr_pt), e, n) == C:
            print corr_pt
            exit()
```

SharifCTF{65d7551577a6a613c99c2b4023039b0a}

Dec 18, 2016 · by creed