Flare-On 10 Challenge 11: over_the_rainbow

By Sam Kim

Solution

This challenge is a piece of ransomware written in C++. This ransomware uses 2 threads: a file collector thread that looks for short files with the extension `.3ncrypt_m3`, and an encryptor thread that encrypts collected files. The encryptor thread uses ChaCha20 and XOR to encrypt the file, and RSA with an embedded public key (decrypted at runtime) to encrypt the ChaCha20 state matrix and XOR key.

The encrypted private key is in the following format(where the last 64 bytes are the ChaCha20 state matrix):

```
[24 bytes random XOR key][48 random bytes][16 bytes: "extend 32-byte k"]
```

We'll use e to denote the RSA public exponent, `N` to denote the RSA modulus, `m` to denote the plaintext encrypted using RSA, and c to denote the ciphertext. The key bug that makes decrypting the file possible is that the RSA public exponent is small (e=3), the plaintext is relatively short, and the ransomware uses no padding for RSA.

Solution 1: The Elementary Way

These conditions are similar to what's required for the RSA cube-root attack, but unfortunately the plaintext is a bit too long (i.e. m^e > n). However, m^e isn't that much bigger than n, so it's possible to use a variant instead.
Let x be the known lower 16 bytes of the private key, and y be the unknown upper 72 bytes of the private key. We then have m = 2^128*y + x, so m^e = m^3 = 2^128*(...) + x^3, so the lower 128 bits of m^e are known!
Since c = m^e (mod N), we have m^e = c + kN for some integer k. We can filter down the candidates for k by ensuring that the LSBs of c+kN match the known LSBs of m^e. (There should be only 1 possible candidate for k given our key sizes).
Once we know k, we get m^e, so we can take the cube root to recover m.

Solution 2: The Fancy Way

Note that y is relatively small, so we can use Coppersmith's attack to recover it.
Here's a SageMath script that recovers y:

```
N = 254...169 # modulus
x = 134877382546472451071872875728390660203 # known lsbs of m
c = 242...316 # ciphertext
K = Zmod(N)
P.<y> = PolynomialRing(Zmod(N))
f = ((2^128*y + x)^3 - c).monic()
print(f.small_roots(epsilon=0.05))
```

Solve script (solution 1):

```
from Crypto.PublicKey import RSA

def icbrt(x):
    # find integer cube root
    if x == 1:
        return 1
    hi = 1
    while hi**3 < x:
        hi *= 2
    lo = hi//2
    while lo <= hi:
        mid = (lo+hi)//2
        if mid**3 < x:
            lo = mid+1
        elif mid**3 > x:
            hi = mid-1
        else:
            return mid
    print('failed')

def chacha_step(chacha_state):
    x = chacha_state[:]
    def ROTL(a,b):
        return ((a << b) | (a >> (32 - b))) & 0xFFFFFFFF
    def QR(a,b,c,d):
        x[a] = (x[a] + x[b]) & 0xFFFFFFFF
        x[d] ^= x[a]
        x[d] = ROTL(x[d], 16)
        x[c] = (x[c] + x[d]) & 0xFFFFFFFF
        x[b] ^= x[c]
        x[b] = ROTL(x[b], 12)
        x[a] = (x[a] + x[b]) & 0xFFFFFFFF
        x[d] ^= x[a]
        x[d] = ROTL(x[d], 8)
        x[c] = (x[c] + x[d]) & 0xFFFFFFFF
        x[b] ^= x[c]
        x[b] = ROTL(x[b], 7)
    for i in range(10):
        QR(0, 4, 8, 12)
        QR(1, 5, 9, 13)
        QR(2, 6, 10, 14)
        QR(3, 7, 11, 15)
        QR(0, 5, 10, 15)
        QR(1, 6, 11, 12)
        QR(2, 7, 8, 13)
        QR(3, 4, 9, 14)
    out = b''
    for i in range(16):
        out += ((chacha_state[i] + x[i]) & 0xFFFFFFFF).to_bytes(4, 'little')
    chacha_state[8] = (chacha_state[8] + 1) & 0xFFFFFFFF
    if chacha_state[8] == 0:
        chacha_state[9] = (chacha_state[9] + 1) & 0xFFFFFFFF
    return out

s = open('very_important_file.d3crypt_m3','rb').read()

pubkey = [86, 133, 248, 47, 2, 30, 204, 241, 170, 94, 29, 58, 194, 134, 189, 87,
8, 88, 238, 151, 166, 1, 116, 171, 158, 205, 7, 119, 46, 221, 131, 167,
```

```
95, 9, 59, 224, 168, 151, 88, 62, 232, 217, 154, 51, 64, 84, 211, 255,
186, 93, 3, 51, 222, 131, 186, 103, 2, 176, 236, 151, 78, 125, 35, 205,
250, 163, 86, 3, 62, 217, 136, 143, 64, 29, 10, 201, 184, 61, 94, 29,
195, 202, 221, 75, 116, 41, 227, 243, 185, 85, 41, 112, 247, 177, 82, 72,
31, 167, 216, 184, 87, 122, 93, 197, 246, 147, 71, 41, 50, 253, 189, 201,
83, 32, 239, 216, 234, 66, 115, 31, 162, 161, 186, 87, 114, 51, 239, 255,
172, 81, 45, 232, 253, 178, 49, 120, 32, 246, 214, 210, 90, 39, 5, 240,
228, 146, 98, 27, 60, 205, 224, 41, 74, 44, 193, 243, 237, 108, 56, 41,
233, 158, 189, 88, 36, 78, 155, 134, 97, 3, 114, 179, 211, 137, 102, 92,
34, 206, 240, 156, 38, 123, 20, 205, 131, 107, 96, 8, 178, 204, 149, 77,
14, 92, 163, 146, 221, 126, 37, 46, 244, 157, 190, 68, 24, 211, 214, 146,
125, 80, 40, 243, 206, 223, 95, 114, 26, 241, 154, 157, 80, 94, 203, 227,
163, 100, 4, 83, 213, 255, 155, 61, 0, 70, 235, 164, 149, 93, 60, 19,
207, 224, 51, 74, 8, 238, 140, 175, 102, 58, 39, 246, 241, 205, 106, 119,
46, 242, 186, 115, 22, 119, 200, 132, 153, 83, 8, 1, 214, 167, 191, 47,
29, 32, 252, 179, 152, 79, 9, 212, 131, 143, 125, 124, 5, 208, 203, 135,
46, 15, 68, 245, 158, 178, 75, 121, 244, 245, 228, 56, 80, 53, 164, 252,
191, 109, 32, 30, 233, 159, 128, 67, 3, 227, 247, 236, 83, 90, 12, 234,
130, 170, 76, 52, 40, 170, 144, 183, 101, 34, 82, 227, 160, 99, 100, 46,
193, 221, 151, 92, 115, 10, 175, 173, 176, 84, 32, 24, 245, 179, 79, 36,
109, 205, 129, 137, 57, 109, 80, 167, 147, 168, 115, 40, 52, 151, 152, 166,
89, 20, 221, 247, 189, 60, 9, 7, 227, 151, 155, 111, 17, 14, 153, 186,
169, 30, 4, 248, 201, 184, 57, 83, 40, 165, 231, 193, 36, 12, 48, 170,
228, 141, 86, 24, 59, 201, 161, 54, 88, 20, 225, 244, 139, 123, 44, 89,
145, 145, 164, 107, 13, 61, 222, 235, 62, 58, 112, 167, 154, 201, 60, 123,
37, 220, 229, 162, 74, 14, 53, 239, 144, 32, 102, 31, 222, 153, 204, 35,
22, 69, 149]

for i in range(len(pubkey)):
    pubkey[i] ^= (123+45*i) & 0xFF
pubkey = bytes(pubkey).decode('ascii').rstrip('\x00')
pubkey = RSA.import_key(pubkey)
n = pubkey.n
assert pubkey.e == 3

# recover xor key + chacha20 state
c = int.from_bytes(s[-256:], 'big')
lsb_mod = 1 << 128
known_lsb = (int.from_bytes(b'expand 32-byte k', 'big')**3) % lsb_mod
# c + k*n = known_lsb (mod lsb_mod)
k = ((known_lsb - c) * pow(n, -1, lsb_mod)) % lsb_mod
m3 = c + k*n
m = icbrt(m3)
m = m.to_bytes(256, 'big')

xorkey = m[-64-24:-64]
chacha_state = []
for i in range(16):
    chacha_state.append(int.from_bytes(m[-64+i*4:][:4], 'little'))

keystream = []
message = bytearray(s[:-256])
for i in range(len(message)):
    if i % 64 == 0:
        keystream = chacha_step(chacha_state)
    message[i] ^= keystream[i%64] ^ xorkey[i%24]
message = bytes(message).decode()
print(message)
```