Uselessness As A Service Writeup

December 21, 2017

1 Crapserver

The server allows us to do two things:

1. View the current state of the LCG
2. Encrypt the flag + next state of LCG

If we can figure out the initial state of the LCG, it will allow us to compute the whole sequence, which in turn can be used to find the value added to flag everytime it's encrypted. Let's dive into LCGs first!

2 Linear Congruential Generator

Linear congruential generators (LCGs) are often used to generate a sequence of pseudorandom numbers. LCGs are defined as following

$$s_{n+1} = ((s_n \times a) + b) \mod m, \tag{1}$$

where s_n is the current state and s_{n+1} is the next state. This can be seen in the supplied code:

```
def next(self):
        self.s = ((self.a * self.s) + self.b) % self.m
        return self.s
```

m is set to 325010924826975202081527196997. The initial state, a, and b are initialized to random, 80-bit integers. To recover the whole LCG sequence, we need to find a, b, and the initial state s_{init}.

$$s_1 \equiv a \times s_0 + b \mod m \tag{2}$$

$$s_2 \equiv a \times s_1 + b \mod m \tag{3}$$

$$s_3 \equiv a \times s_2 + b \mod m \tag{4}$$

We start by finding a.

Subtract equation 1 from 2 and 3:

$$s_2 - s_1 \equiv a \times (s_1 - s_0) \mod m \tag{5}$$

$$s_3 - s_1 \equiv a \times (s_2 - s_0) \mod m \tag{6}$$

$$a = \begin{cases} (s_2 - s_1) \times \text{modinv}((s_1 - s_0), m) & \text{if } \gcd((s_1 - s_0), m) = 1 \\ (s_3 - s_1) \times \text{modinv}((s_2 - s_0), m) & \text{if } \gcd((s_2 - s_0), m) = 1 \end{cases} \tag{7}$$

Then we can obtain b:

$$b = s_1 - (s_0 \times a) \mod m \tag{8}$$

And the initial state, s_{init}:

$$s_{init} = ((s_0 - b) \mod m) \times \text{modinv}(a, m) \mod m \tag{9}$$

3 Franklin-Reiter Related Message Attack

After recovering the sequence produced by the LCG, we can query the server to compute:

$$\begin{aligned} C_1 &\equiv (f + s_4)^e \mod n \quad \text{and,} \\ C_2 &\equiv (f + s_5)^e \mod n, \end{aligned} \tag{10}$$

where f is the flag, s_4 is the fifth state (s_0 is the first state we got from the server), and s_5 is the sixth state.

From [1] we have the following lemma and proof:

Lemma 1 *Set $e = 3$ and let $\langle N, e\rangle$ be an RSA public key. Let $M_1 \neq M_2 \in \mathbb{Z}_N^*$ satisfy $M_1 = f(M_2) \mod N$ for some linear polynomial $f = ax + b \in \mathbb{Z}_N[x]$ with $b \neq 0$. Then, given $\langle N, e, C_1, C_2, f\rangle$, Marvin can recover M_1, M_2 in time quadratic in $logN$.*

Proof 1 *Since $C_1 = M_1^e \mod N$, we know that M_2 is a root of the polynomial $g_1(x) = f(x)^e - C_1 \in \mathbb{Z}_N[x]$. Similarily, M_2 is a root of $g_2(x) = x^e - C_2 \in \mathbb{Z}_N[x]$. The linear factor $x - M_2$ divides both polynomials. Therefore, Marvin may use the Euclidean algorithm to compute the gcd of g_1 and g_2. If the gcd turns out to be linear, M_2 is found.*

We solve it almost the same way, except that we recover M_1 instead of M_2. To simplify things a bit, we first find the largest ciphertext, depending on whether s_5 or s_4 is the largest, and set this to C_2. Then we compute their difference $\Delta = s_{C_2} - s_{C_1}$, where s_{C_n} is the LCG state for ciphertext n. Let

$$\begin{aligned} g_1 =& x^3 - C_1 \quad \text{and,} \\ g_2 =& (x + \Delta)^3 - C_2 \end{aligned} \tag{11}$$

We have to compute the gcd of these two polynomials, leaving us with a polynomial on the form $x - m$. $-m$ is coefficient 0, which means that the only thing left to recover M_1 is to negate m. To recover the flag, we do

$$f = (-m) - s_{C_1} \tag{12}$$

Following is an implementation in sage:

```
# https://crypto.stackexchange.com/questions/30884/
# help-understanding-basic-franklin-reiter-related-message-attack
def my_gcd(a, b):
        return a.monic() if b == 0 else my_gcd(b, a % b)

R.<X> = Zmod(n)[]
g1 = X^3 - c1
g2 = (X + diff)^3 - c2

tmp = int(- my_gcd(g1, g2).coefficients()[0])
flag = tmp - flag_off
print "flag: {}".format(binascii.unhexlify(hex(flag)))
```

where `flag_off` is either s_4 or s_5.

References

[1] Dan Boneh et al. “Twenty years of attacks on the RSA cryptosystem”. In: *Notices of the AMS* 46.2 (1999), pp. 203–213.