Writeup: Going In Circles (Crypto) — Nullcon CTF

Descripción del reto

El reto nos da un script `chall.py` que:

1. Lee la flag desde `flag.txt` y la convierte en un entero (interpretando los bytes en big-endian).
2. Genera un polinomio aleatorio `f` de 32 bits (en realidad un entero de hasta 32 bits, que interpretamos como polinomio en **GF(2)[x]**).
3. Aplica una función `reduce(flag, f)` y nos devuelve el par `(reduce(flag, f), f)`.

```
def reduce(a, f):
    while (l := a.bit_length()) > BITS:
        a ^= f << (l - BITS)
    return a
```

Cada conexión al servicio nos da dos números: el “resto” de la flag módulo `f` y el propio `f`.

Interpretación en GF(2)[x]

En **GF(2)[x]** los coeficientes están en {0, 1} y la suma es XOR. Un entero se interpreta como polinomio: el bit en posición `i` es el coeficiente de `x^i`.

La función `reduce(a, f)` hace exactamente la **reducción de `a` módulo `f`** en GF(2)[x]:

- Mientras el grado de `a` (su longitud en bits menos 1) sea ≥ que el grado de `f` (32 bits ⇒ grado ≤ 31), se alinea el término líder de `f` con el de `a` y se hace `a ^= f << (l - BITS)`, es decir, se resta (XOR) un múltiplo de `f` para cancelar el bit más alto de `a`.
- El resultado es el **resto** de la división de `a` entre `f` en GF(2)[x].

Por tanto, en cada muestra tenemos:

- `r = flag mod f` (módulo polinomial en GF(2)[x])
- `f` = polinomio “módulo” (de grado ≤ 31)

Es decir: **flag ≡ r (mod f)** en el anillo GF(2)[x].

Idea de la solución: CRT en GF(2)[x]

El **Teorema Chino del Resto (CRT)** también vale en GF(2)[x]: si tenemos congruencias

- `x ≡ r1 (mod f1)`
- `x ≡ r2 (mod f2)`

y `f1` y `f2` son coprimos (su mcd en GF(2)[x] es 1), existe un único `x` módulo `f1·f2` (o módulo lcm(f1,f2)) que cumple ambas.

Cada conexión nos da un nuevo par `(r_i, f_i)` con `flag ≡ r_i (mod f_i)`. Si vamos combinando estas congruencias con CRT:

- Empezamos con `R = r1`, `M = f1`.
- Para cada nuevo `(r, f)` calculamos la solución común a `x ≡ R (mod M)` y `x ≡ r (mod f)`, obteniendo un nuevo resto `R'` y un nuevo módulo `M' = lcm(M, f)` (o un múltiplo de este). Actualizamos `R = R'`, `M = M'`.

Cuando el grado de `M` (o la longitud en bits de `M`) sea mayor o igual que la longitud en bits de la flag, la solución módulo `M` será única y coincidirá con la flag. Entonces podemos intentar decodificar `R` como bytes y buscar el formato de la flag (por ejemplo `ENO{...}`).

Implementación del solver

1. Aritmética en GF(2)[x]

Se implementan sobre enteros (polinomios en base 2):

- **Grado**: `poly_deg(a) = bit_length(a) - 1`
- **Multiplicación sin acarreo**: `poly_mul(a, b)` (XOR de desplazamientos de `a` según bits de `b`)
- **División con resto**: `poly_divmod(a, b)` → cociente y resto
- **Resto**: `poly_mod(a, m)`
- **MCD**: `poly_gcd(a, b)` (algoritmo de Euclides con la división polinomial)
- **MCD extendido**: `poly_egcd(a, b)` para obtener coeficientes en la identidad de Bézout y usarlos en el CRT

2. CRT en GF(2)[x]

La función `crt_poly(r1, m1, r2, m2)` resuelve:

- `x ≡ r1 (mod m1)`

- `x ≡ r2 (mod m2)`

Comprueba que `r1 ≡ r2 (mod gcd(m1, m2))` (consistencia). Luego, usando el MCD extendido entre `m1/g` y `m2/g`, construye una solución `x` y devuelve `(x, m)` con `m = lcm(m1, m2)` (normalizado como resto módulo `m`).

3. Bucle principal

1. Conectar al servicio y recibir `(r, f)`.
2. Si `f` es 0 o 1 se ignora (no dan información útil).
3. La primera muestra válida inicializa `R = r`, `M = f`.
4. Para las siguientes, se combina con CRT: `R, M = crt_poly(R, M, r, f)`.
5. Tras cada actualización se intenta decodificar `R` como bytes y buscar una cadena con el prefijo/sufijo de la flag (p. ej. `ENO{...}`). Si se encuentra, se imprime y se termina.

Así, con suficientes conexiones (y mientras los `f` no sean todos iguales o compartan factores grandes), el módulo combinado `M` crece hasta acotar unívocamente la flag y recuperarla.

Resumen

Concepto	Uso en el reto
reduce(a, f)	Resto de `a` módulo `f` en GF(2)[x]
Cada muestra	`flag ≡ r (mod f)` en GF(2)[x]
Solución	Combinar muchas muestras con CRT en GF(2)[x] hasta que el módulo combinado tenga grado ≥ longitud de la flag
Decodificación	Convertir el entero recuperado a bytes y buscar el formato de la flag

El nombre “Going In Circles” encaja con la idea de dar vueltas al servidor (muchas conexiones) y con la estructura algebraica de polinomios sobre GF(2) y congruencias (CRT).

Flag

```
ENO{CRC_is_just_some_modular_remainder}
```