Baby RSA - Writeup

Este desafío criptográfico del HFCTF nos presenta una implementación vulnerable del algoritmo RSA donde una filtración de información permite la recuperación de la clave privada.

Análisis inicial

El reto nos proporciona dos archivos:

- **chall.py**: El script que genera los parámetros RSA, cifra la flag y crea la salida.
- **output.txt**: Contiene los valores públicos `e`, `n`, el texto cifrado `ct`, y un valor adicional `leak`.

La configuración es típica de RSA, con el exponente público estándar `e = 65537`, pero con una particularidad: se filtra un valor adicional llamado `leak`.

Identificando la vulnerabilidad

La vulnerabilidad clave está en el valor filtrado `leak = p * phi`, donde:

- `p` es uno de los factores primos de `n`
- `phi = (p - 1) * (q - 1)` es la función totiente de Euler

Esto nos proporciona información crítica que normalmente debería mantenerse secreta en una implementación segura de RSA.

Enfoque matemático

Partiendo de la relación `leak = p * phi`, podemos desarrollar:

1. `leak = p * (p - 1) * (q - 1)`
2. Expresando `q` en términos de `n`: `q = n/p`
3. `leak = p * (p - 1) * (n/p - 1)`
4. Simplificando: `leak = (p - 1) * (n - p)`
5. Expandiendo: `leak = p*n - p² - n + p`
6. Reorganizando como ecuación cuadrática: `p² - (n + 1)*p + (n + leak) = 0`

Esta ecuación cuadrática nos permite calcular `p` directamente.

Desarrollo de la solución

Para resolver la ecuación cuadrática, calculamos:

1. El discriminante: `delta = (n - 1)² - 4 * leak`
2. La raíz cuadrada: `s = sqrt(delta)`
3. Dos posibles valores para `p`:
    - `p1 = (n + 1 + s) / 2`
    - `p2 = (n + 1 - s) / 2`
4. Verificando cuál divide a `n` perfectamente

Implementación

```
from Crypto.Util.number import long_to_bytes, inverse
import math

# Valores extraídos de output.txt
e = 65537
n = # número muy grande
ct = # número muy grande
leak = # número muy grande

# Calculamos el discriminante
delta = (n - 1)**2 - 4 * leak
s = math.isqrt(delta)

# Verificamos que delta sea un cuadrado perfecto
if s * s != delta:
    print("Error: Delta no es un cuadrado perfecto.")
    exit()
```

```
# Calculamos los posibles valores de p
p1 = (n + 1 + s) // 2
p2 = (n + 1 - s) // 2

# Encontramos el valor correcto de p
if n % p1 == 0:
    p = p1
    q = n // p1
elif n % p2 == 0:
    p = p2
    q = n // p2
else:
    print("Error: No se pudo factorizar n.")
    exit()

# Calculamos phi y la clave privada
phi = (p - 1) * (q - 1)
d = inverse(e, phi)

# Desciframos el mensaje
m = pow(ct, d, n)
flag = long_to_bytes(m).decode()

print(flag)  # Imprime: hfctf{ezZzZZ_Rs4_l3444k_cAlcul4ti0N5__!12}
```

Obteniendo la flag

Al ejecutar nuestro script con los valores correctos de `output.txt`, recuperamos la flag:

hfctf{ezZzZZ_Rs4_l3444k_cAlcul4ti0N5__!12}