Write-up: Twister

Me he encontrado en una página web esta versión del Twister, pero es un poco extraña. El objetivo no es poner el pie o la mano en el lugar indicado, sino predecir el futuro.

Análisis del reto

Al conectarnos al servidor (nc ctf.hackademics-forum.com 15364), se nos pide adivinar coordenadas (x,y) durante 100 rondas consecutivas para obtener la bandera.

Al examinar el código fuente proporcionado (server.py), encontramos estos puntos clave:

- El servidor utiliza `random.getrandbits(32)` para generar números aleatorios.
- Estos números se convierten a base 64 mediante una función `rebase(n)`.
- Las coordenadas se calculan a partir de valores extraídos de una lista `l`.
- Por cada acierto, se incrementa un contador de racha (streak).

Identificación de la vulnerabilidad

La vulnerabilidad principal radica en el uso del módulo `random` de Python, que implementa el algoritmo Mersenne Twister. Este PRNG (Generador de Números Pseudoaleatorios) no es criptográficamente seguro, ya que su estado interno puede ser reconstruido completamente si se observan suficientes salidas consecutivas.

El Mersenne Twister tiene un periodo de 2^19937-1, lo que lo hace estadísticamente robusto, pero su estado interno puede ser recuperado si se observan 624 valores de 32 bits consecutivos.

Estrategia de explotación

Nuestra estrategia consta de tres fases:

1. **Recolección de datos**: Jugar muchas rondas para observar las coordenadas reales.

2. **Reconstrucción del estado**: Utilizar los datos recolectados para inferir los valores originales generados por `getrandbits(32)`.
3. **Predicción**: Usar la biblioteca randcrack para predecir futuros valores y ganar el juego.

Implementación de la solución

Primero, necesitamos observar y reconstruir los valores originales:

```
from pwn import *
from randcrack import RandCrack

# Conectar al servidor
conn = remote('ctf.hackademics-forum.com', 15364)
rc = RandCrack()

# Lista para almacenar valores reconstruidos
reconstructed_values = []
current_sequence = []
bit_length = None

# Fase de recolección
for _ in range(4500):  # Necesitamos aproximadamente 4500 rondas para obtener 624 valores
    conn.sendline("0,0")  # Enviar una predicción cualquiera
    response = conn.recvline().decode()

    # Extraer coordenadas reales
    real_coords = response.split("[")[^1].split("]")[^0]
    real_x, real_y = map(int, real_coords.split(", "))

    # Calcular el valor original
    popped_n = real_x * 8 + real_y

    # Intentar reconstruir el valor original de getrandbits(32)
    if bit_length is None:
        bit_length = popped_n
        current_sequence = []
```

```
    else:
        current_sequence.append(popped_n)

        # Verificar si la secuencia está completa
        if len(current_sequence) >= bit_length // 6:  # Aproximadamente
            # Reconstruir el valor original
            n_candidate = 0
            for i, digit in enumerate(reversed(current_sequence)):
                n_candidate += digit * (64 ** i)

            # Validar el valor reconstruido
            if n_candidate.bit_length() == bit_length:
                reconstructed_values.append(n_candidate)
                rc.submit(n_candidate)

                # Reiniciar para la siguiente secuencia
                bit_length = None

                # Verificar si tenemos suficientes valores
                if len(reconstructed_values) >= 624:
                    break
```

Una vez que tenemos suficientes valores para predecir el estado del PRNG, podemos empezar a adivinar correctamente:

```
# Fase de predicción
l = []
streak = 0

while streak < 100:
    # Si nuestra lista está vacía, predecir el siguiente valor
    if not l:
        next_n = rc.predict_getrandbits(32)
        # Convertir a base 64 y añadir bit_length
        l = rebase(next_n) + [next_n.bit_length()]

    # Extraer el siguiente valor
    popped_n = l.pop()
```

```
    # Calcular coordenadas
    predict_x = popped_n // 8
    predict_y = popped_n % 8

    # Enviar predicción
    conn.sendline(f"{predict_x},{predict_y}")

    # Verificar resultado
    response = conn.recvline().decode()
    if "¡Correcto!" in response:
        streak += 1
    else:
        # En caso de error (no debería ocurrir)
        streak = 0

# Recibir la bandera
print(conn.recvall().decode())
```

Función rebase

La función `rebase` mencionada convierte un número a base 64:

```
def rebase(n):
    if n < 64:
        return [n]
    return [n % 64] + rebase(n // 64)
```

Obteniendo la bandera

Después de ejecutar este script, logramos predecir correctamente 100 coordenadas consecutivas, obteniendo la bandera:

```
¡Felicidades! Has conseguido predecir 100 posiciones consecutivas.
Aquí tienes tu bandera: hfctf{M3rsENn3_tw1ST3r_s0_bR0k333Nn...}
```