Writeup: Jumploit - Reversing Challenge

Dicen que en lo más alto encontrarás lo que buscas... pero, ¿y si el juego nunca te deja llegar? La obsesión por saltar una y otra vez te consume, como un jugador atrapado en un bucle sin fin.

Análisis Inicial

El archivo es un ejecutable ELF de 64 bits para Linux, no despojado (not stripped), lo que facilita nuestro análisis al conservar los nombres de las funciones. Usando Ghidra para el desensamblado, observamos que el programa utiliza extensivamente la biblioteca SDL2 para las funciones gráficas.

Dependencias identificadas:

- Funciones SDL (SDL_CreateWindow, SDL_CreateRenderer, SDL_PollEvent, etc.)
- Elementos de C++ (std::vector, constructores/destructores)

Análisis Detallado

Inicialización del Juego

Examinando las funciones de inicialización, descubrimos algo crucial:

1. Se llama a `rand()` **antes** de establecer una semilla con `srand(time(0))`
2. Esta llamada inicial a `rand()` genera un valor predecible: 1804289383
3. Se inicializa una variable global: `hidden_jump_count = (resultado_de_rand % 500) + 500`
4. Otra variable define la condición de victoria: `MAX_JUMP_COUNT = offset ^ hidden_jump_count`

Lógica Principal

La función `main` implementa un bucle de juego clásico:

- Procesa eventos SDL (detecta pulsaciones de teclas)

- Gestiona la física del juego (gravedad, colisiones con plataformas)
- Mantiene un contador de saltos que se incrementa cada vez que presionamos Espacio
- Comprueba la condición de victoria: `if (MAX_JUMP_COUNT <= contador_de_saltos)`

Sistema de Renderizado de la Flag

Al alcanzar la condición de victoria, el juego llama a una función que:

1. Crea una nueva ventana SDL
2. Copia 2976 bytes (372 qwords) desde una dirección fija en el binario
3. Interpreta estos datos como una matriz de enteros
4. Dibuja píxeles blancos donde el valor es exactamente 1

Solución

Para resolver el reto necesitamos:

1. **Calcular el número exacto de saltos requeridos:**
    - Valor del primer `rand()` (sin semilla): 1804289383
    - `hidden_jump_count = 1804289383 % 500 + 500 = 883`
    - `offset` (variable global no inicializada): 0
    - `MAX_JUMP_COUNT = 0 ^ 883 = 883`
2. **Obtener la flag:**
    - Método 1: Jugar el juego y hacer exactamente 883 saltos (tedioso)
    - Método 2: Extraer los datos del binario y renderizar la imagen

Extracción de la Flag (Método 2)

Localizamos los datos de la imagen en la dirección virtual 0x00103060 con un tamaño de 2976 bytes. Calculamos el offset correspondiente en el archivo y extraemos los datos:

```
dd if=Jumploit of=flag_data.bin bs=1 skip=12384 count=2976
```

Luego utilizamos un script para interpretar estos datos como una matriz de 124x6 píxeles y visualizar la flag:

```
import struct

IMG_WIDTH = 124
IMG_HEIGHT = 6
DATA_SIZE = 2976
INPUT_FILE = 'flag_data.bin'
PIXEL_ON = '#'
PIXEL_OFF = ' '

with open(INPUT_FILE, 'rb') as f:
    data = f.read()
integers = struct.unpack(f'<{IMG_WIDTH * IMG_HEIGHT}i', data)

print("-" * IMG_WIDTH)
idx = 0
for y in range(IMG_HEIGHT):
    line = ""
    for x in range(IMG_WIDTH):
        if integers[idx] == 1:
            line += PIXEL_ON
        else:
            line += PIXEL_OFF
        idx += 1
    print(line)
print("-" * IMG_WIDTH)
```

Al ejecutar el script, obtenemos la flag dibujada en ASCII por terminal:

`hfctf{h4ck5_t0_jUMP_th3_f14G}`