Instituto Tecnológico de San Juan del Río

Tópicos de Ciberseguridad

[Reconstruction_R003_R004_SourceRecovery]

P R E S E N T A:

[Oscar Alberto Leal Ramírez] [22590042]
[Isaac Castro Islas] [B19140346]
[Hortencia Sánchez Carlos] [B23590533]
[Ingeniería en Sistemas Computacionales]

PERIODO [Enero-Junio (2026)]

Herramientas Utilizadas:

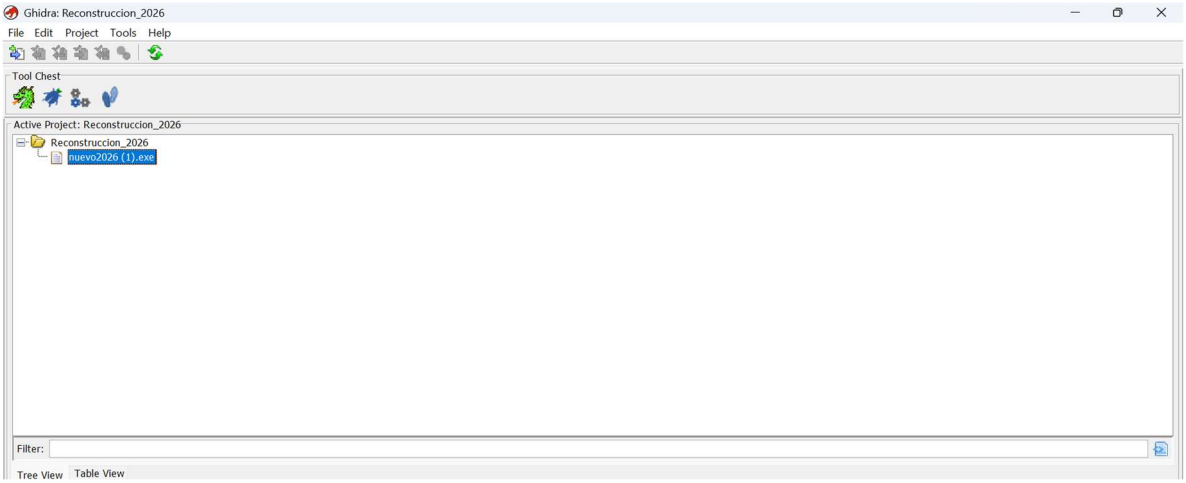
Ghidra: Se utilizó para realizar el análisis estático del ejecutable original. Con esta herramienta se importó el archivo .exe y se ejecutó el proceso de análisis automático (Auto-Analysis), lo que permitió identificar las funciones del programa, los segmentos de memoria y las referencias cruzadas. Posteriormente, se empleó el Decompiler para obtener una versión en lenguaje C a partir del código ensamblador generado por el compilador original, lo que facilitó la comprensión de la lógica del programa.

Java: Fue necesario para el funcionamiento de Ghidra, ya que esta herramienta está desarrollada sobre la plataforma Java y requiere su entorno de ejecución para poder operar correctamente.

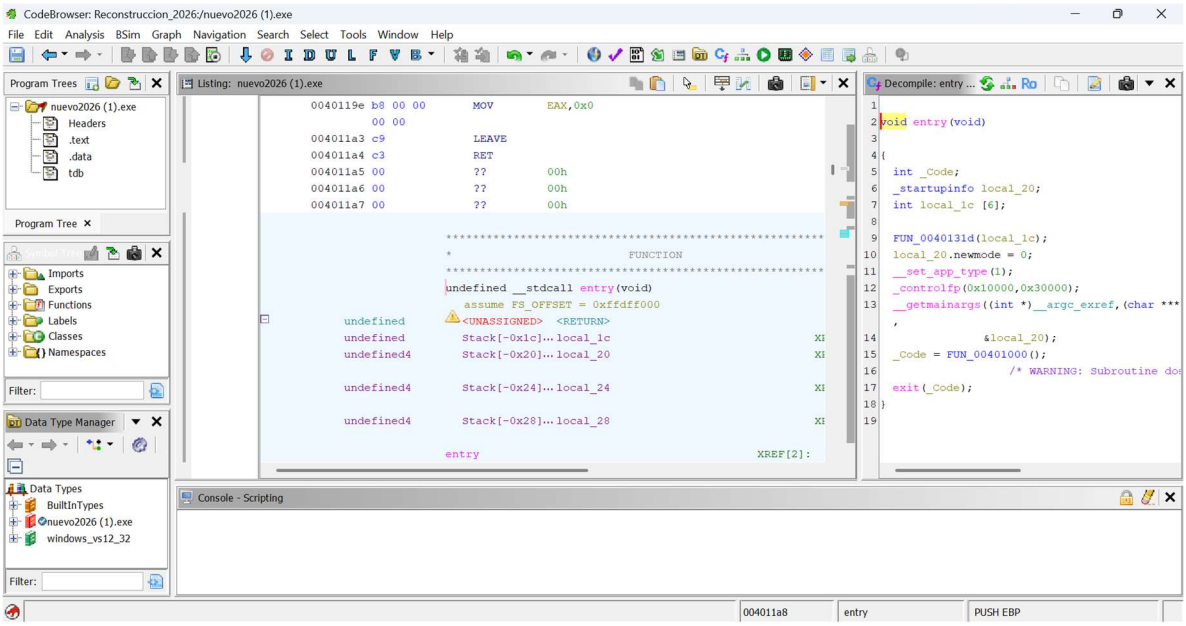
Símbolo del sistema (CMD de Windows): Se utilizó para trabajar en el entorno de compilación. A través del CMD se navegaron las carpetas del sistema, se ejecutaron los comandos de compilación y se verificó la correcta generación del ejecutable reconstruido y del archivo de imagen producido por el programa.

Tiny C Compiler: Se empleó para recompilar el código fuente reconstruido. Se eligió esta herramienta por su facilidad de uso y rapidez de compilación, lo que permitió validar de manera práctica si el nuevo ejecutable mantenía el mismo comportamiento que el original.

1. Creación del Proyecto y Configuración del Entorno:



Primero se creó un proyecto de tipo Non-Shared Project en Ghidra con el nombre Reconstruccion_2026. Después, se importó el ejecutable original en formato Portable Executable (PE).



Una vez importado el archivo, se ejecutó el proceso de Auto-Analysis. Este análisis permitió que Ghidra identificara automáticamente la arquitectura del programa, su punto de entrada (entry point), las funciones internas y las llamadas a librerías estándar como time(), srand(), rand(), fopen() y fwrite().

2. Análisis Estático del Binario

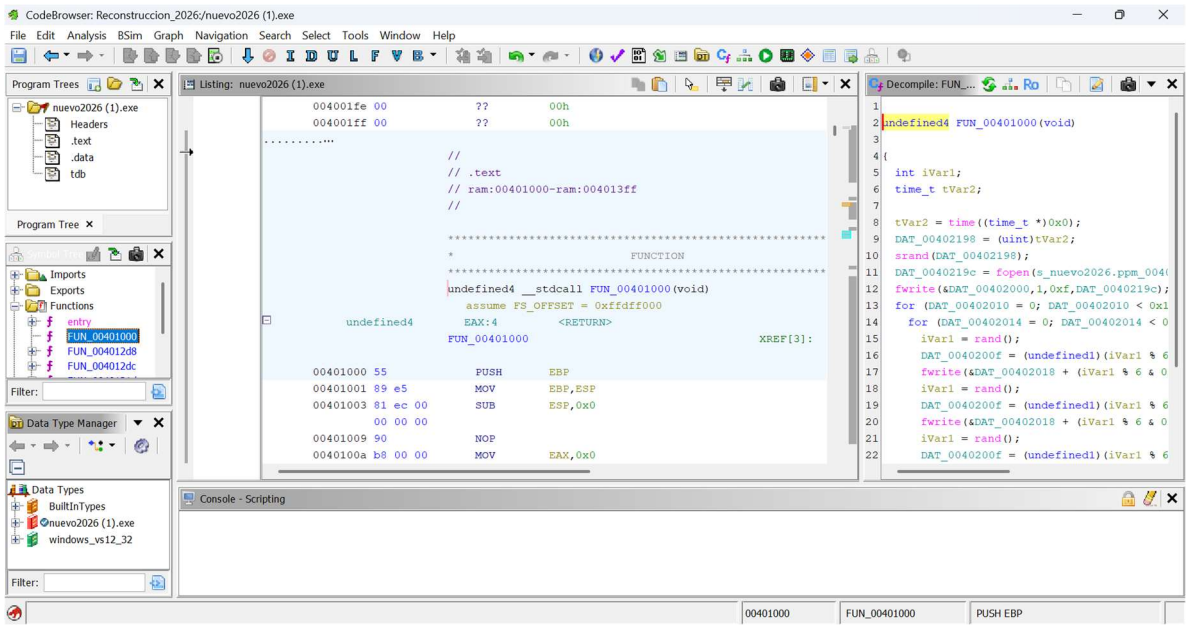
Durante el análisis en Ghidra, se localizó inicialmente la función de entrada (entry point), que representa el punto donde comienza la ejecución del programa. A partir de esta función se rastreó el flujo del programa hasta identificar la función principal, etiquetada por Ghidra como FUN_00401000, dentro del apartado Symbol Tree → Functions.

Mediante el uso del Decompiler se obtuvo una representación en lenguaje C del código ensamblador. Esta representación permitió comprender con mayor claridad la lógica del programa original.

Del análisis se determinó que el programa:

- Inicializa una semilla pseudoaleatoria utilizando las funciones time() y srand().

- Genera valores aleatorios mediante rand().
- Crea un archivo llamado nuevo2026.ppm usando fopen().
- Escribe el encabezado del formato PPM binario (P6) mediante fprintf().
- Escribe los valores RGB de cada píxel utilizando fwrite().
- Finaliza cerrando el archivo con fclose().



El encabezado identificado en el archivo corresponde a:

```
P6
640 480
255
```

Lo anterior indica que el programa genera una imagen en formato PPM binario con una resolución de 640x480 píxeles.

Asimismo, se identificó una doble estructura iterativa que recorre todos los píxeles de la imagen:

```
for (int i = 0; i < 480; i++)
    for (int j = 0; j < 640; j++)
```

En cada iteración se generan tres valores (R, G y B) calculados como rand() % 256, los cuales se escriben directamente en el archivo, produciendo una imagen de ruido aleatorio.

Este análisis permitió confirmar la lógica completa del programa antes de proceder con la reconstrucción manual del código.

3. Código Reconstruido

A partir del código decompilado en Ghidra y tras realizar un proceso de limpieza, reorganización y validación, se obtuvo un código funcional equivalente, listo para compilar con Tiny C Compiler (TCC).:

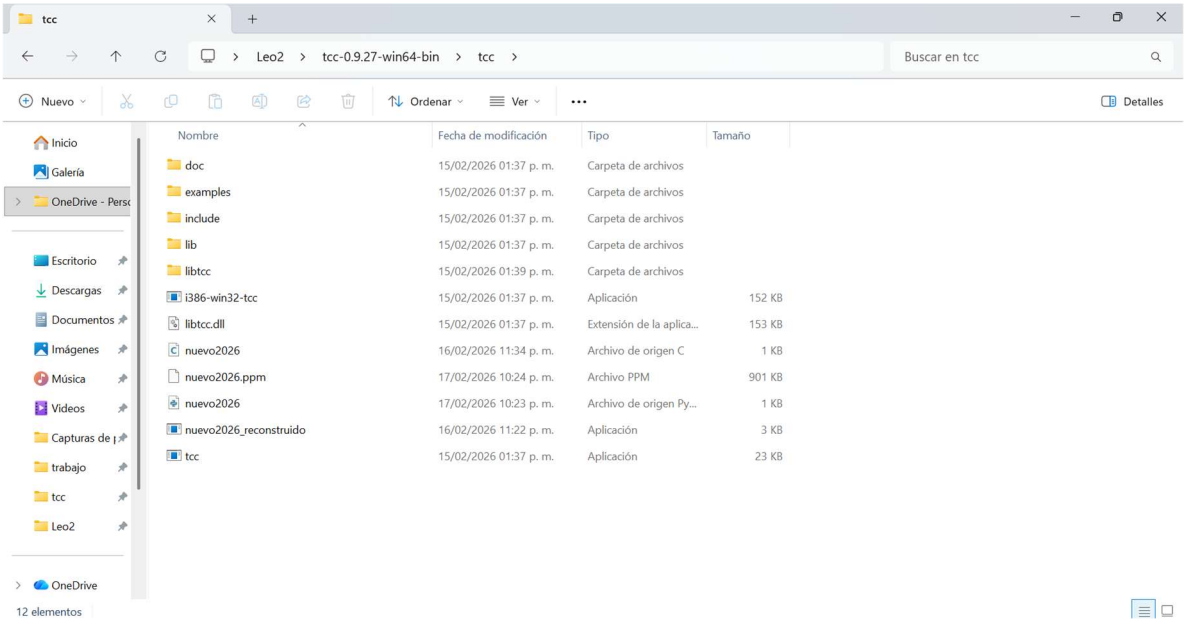
```python
import time
import random

def main():
    random.seed(int(time.time()))

    with open("nuevo2026.ppm", "wb") as f:
        f.write(b"P6\n640 480\n255\n")

        for i in range(480):
            for j in range(640):
                r = random.randint(0, 255)
                g = random.randint(0, 255)
                b = random.randint(0, 255)

                f.write(bytes([r, g, b]))

    print("Imagen generada correctamente: nuevo2026.ppm")

if __name__ == "__main__":
    main()
```

Durante la reconstrucción se realizaron los siguientes ajustes: Se renombraron variables para mejorar la claridad, se verificaron los límites correctos de los ciclos (480 filas y 640 columnas). Se agregó validación en la apertura del archivo para evitar errores en tiempo de ejecución. Se añadió un mensaje de confirmación al finalizar el proceso.

El programa reconstruido fue compilado correctamente y ejecutado sin errores, generando el archivo nuevo2026.ppm.

4. Compilación y Ejecución del Código Reconstruido

Una vez finalizada la reconstrucción del código fuente, el archivo nuevo2026.py fue guardado dentro del directorio donde se encuentra instalado Tiny C Compiler.

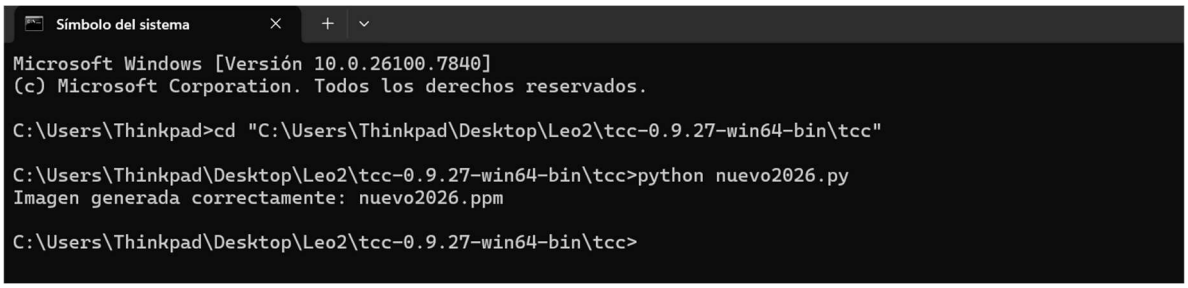


Posteriormente, se abrió el Símbolo del Sistema (CMD de Windows) y se accedió al directorio donde se encontraba el archivo Python mediante el siguiente comando:

cd "C:\Users\Thinkpad\Desktop\Leo2\tcc-0.9.27-win64-bin\tcc"

Una vez ubicado en la carpeta correcta, se procedió a la ejecución del programa con el siguiente comando:

python nuevo2026.py



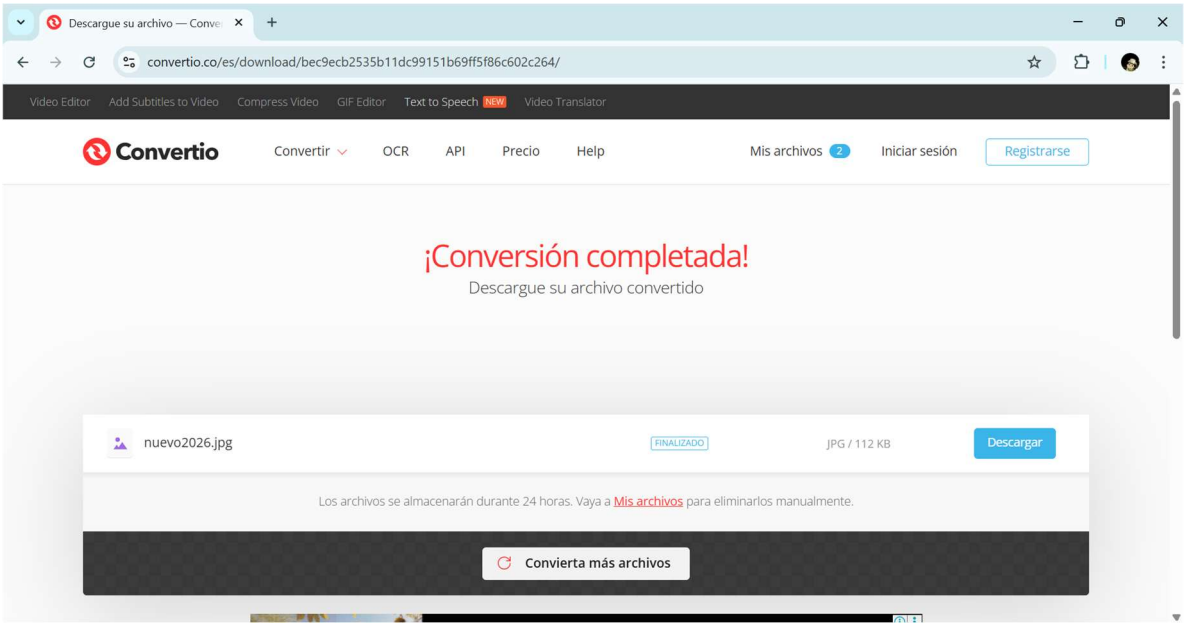
Si la terminal no muestra mensajes de error y aparece el mensaje de confirmación:

Imagen generada correctamente: nuevo2026.ppm

significa que la ejecución fue exitosa y se generó correctamente el archivo nuevo2026.ppm.

5. Verificación del Resultado

Para validar el funcionamiento del programa, se verificó la creación del archivo en el mismo directorio. Al abrirlo en un visor compatible o convertirlo a formato JPG, se confirmó que corresponde a una imagen en formato PPM binario (P6) con resolución de 640x480 píxeles, compuesta por valores RGB generados aleatoriamente.



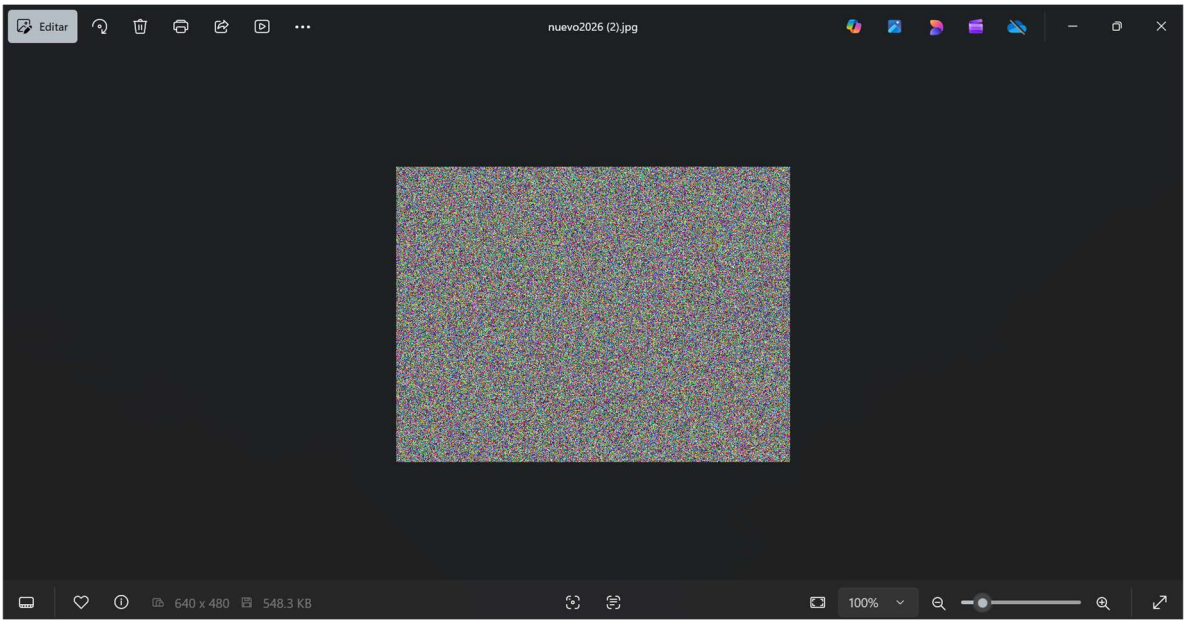
6. Comparación entre el Ejecutable Original y el Ejecutable Reconstruido:

Una vez ejecutado el código fuente reconstruido en Python, se realizó una comparación directa entre el ejecutable original y el programa reimplementado, con el objetivo de validar la equivalencia funcional del comportamiento del sistema. Aunque el lenguaje de implementación es diferente, se verificó que la lógica interna y el resultado generado son equivalentes.

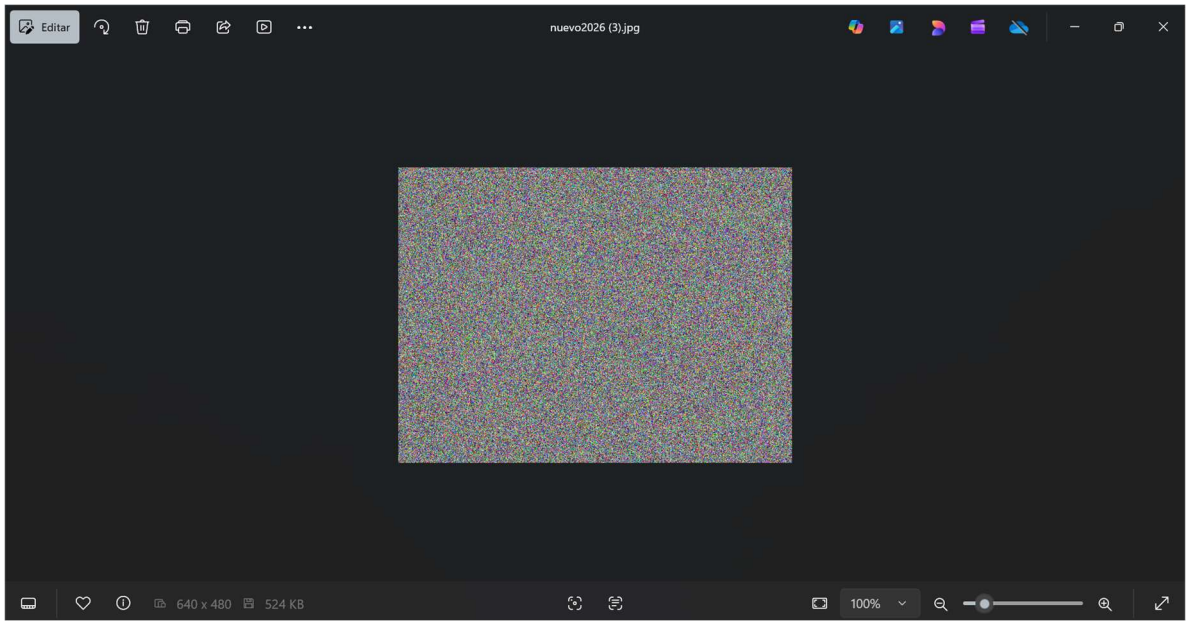
Comparación Técnica:

Parámetro	Ejecutable Original	Ejecutable Reconstruido
Tipo de archivo	Portable Executable (PE)	Script Python (.py)
Arquitectura	x86	Dependiente del intérprete Python
Formato de salida	PPM (P6)	PPM (P6)
Resolución de imagen	640 x 480 píxeles	640 x 480 píxeles
Generación de color	rand() % 256	random.randint(0,255)
Inicialización	time() + srand()	time() + random.seed()
Tipo de imagen generada	Ruido RGB aleatorio	Ruido RGB aleatorio

Original:



Nuevo:



La verificación se realizó revisando tanto la estructura del archivo PPM (encabezado y formato) como el funcionamiento del programa al ejecutarlo y visualizar la imagen generada. Esto permitió confirmar que el comportamiento del software reconstruido coincide con el del programa original.

Análisis de Resultados

La comparación demuestra que el programa reconstruido mantiene la misma lógica que el original. En ambos casos:

- Se inicializa la semilla con la hora del sistema.
- Se generan valores RGB aleatorios para cada píxel.
- Se produce una imagen en formato PPM (P6).
- Se utiliza una resolución de 640x480 píxeles.

Aunque el original era un ejecutable compilado y la versión final se implementó en Python, el resultado funcional es el mismo: una imagen de ruido aleatorio con las mismas características técnicas.

Conclusión

La ingeniería inversa permitió comprender y replicar correctamente la lógica del programa.Inicialmente, la reconstrucción que hicimos se realizó en C debido a las funciones identificadas en Ghidra, lo que indicaba que el ejecutable fue desarrollado en ese lenguaje. Posteriormente se adaptó a Python por indicación del docente. Este cambio no afectó la lógica ni el resultado final, por lo que se mantiene la equivalencia funcional del software original.