



**EDUCACIÓN**  
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLOGICO  
NACIONAL DE MÉXICO



# Instituto Tecnológico de San Juan del Río



## **Tópicos de ciberseguridad**

### **Reconstrucción del Código Fuente con Ghidra**

**P R E S E N T A:**

**Montes Barajas Leonardo Daniel  
Ramírez García Jorge Yael  
Olvera Pérez Erika**

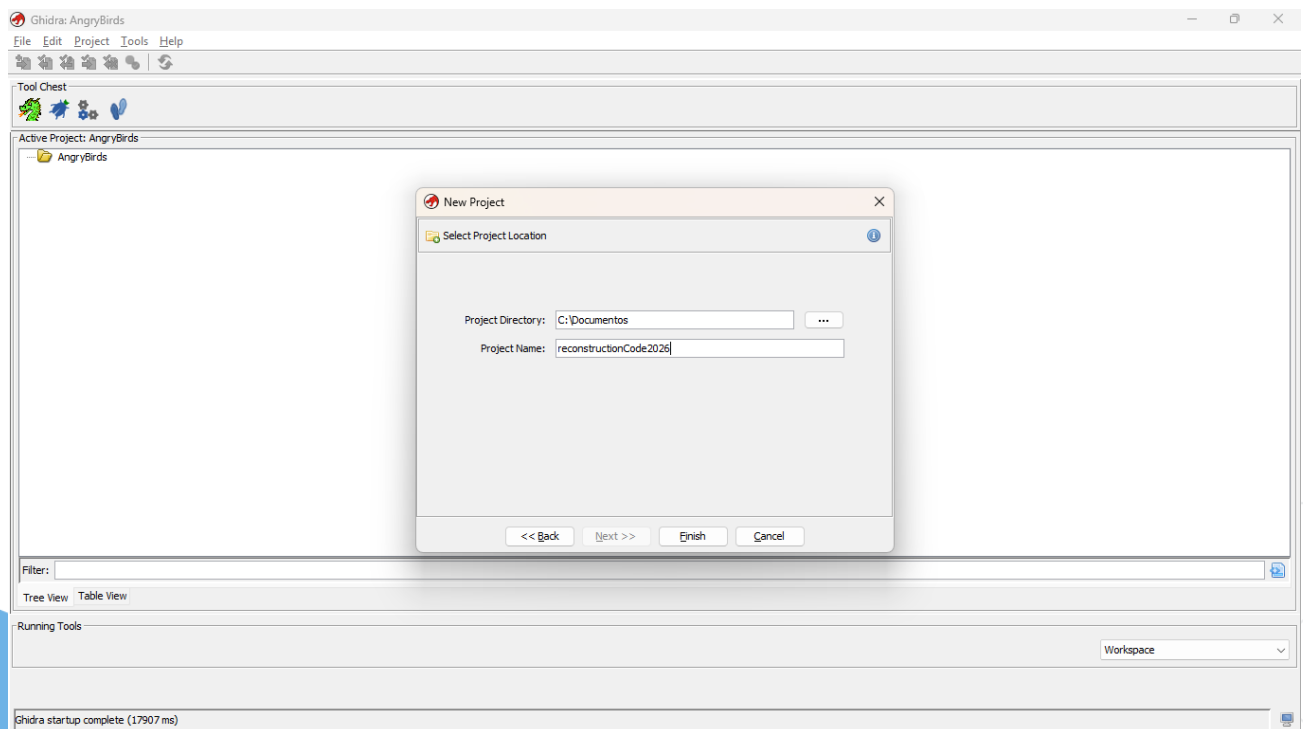
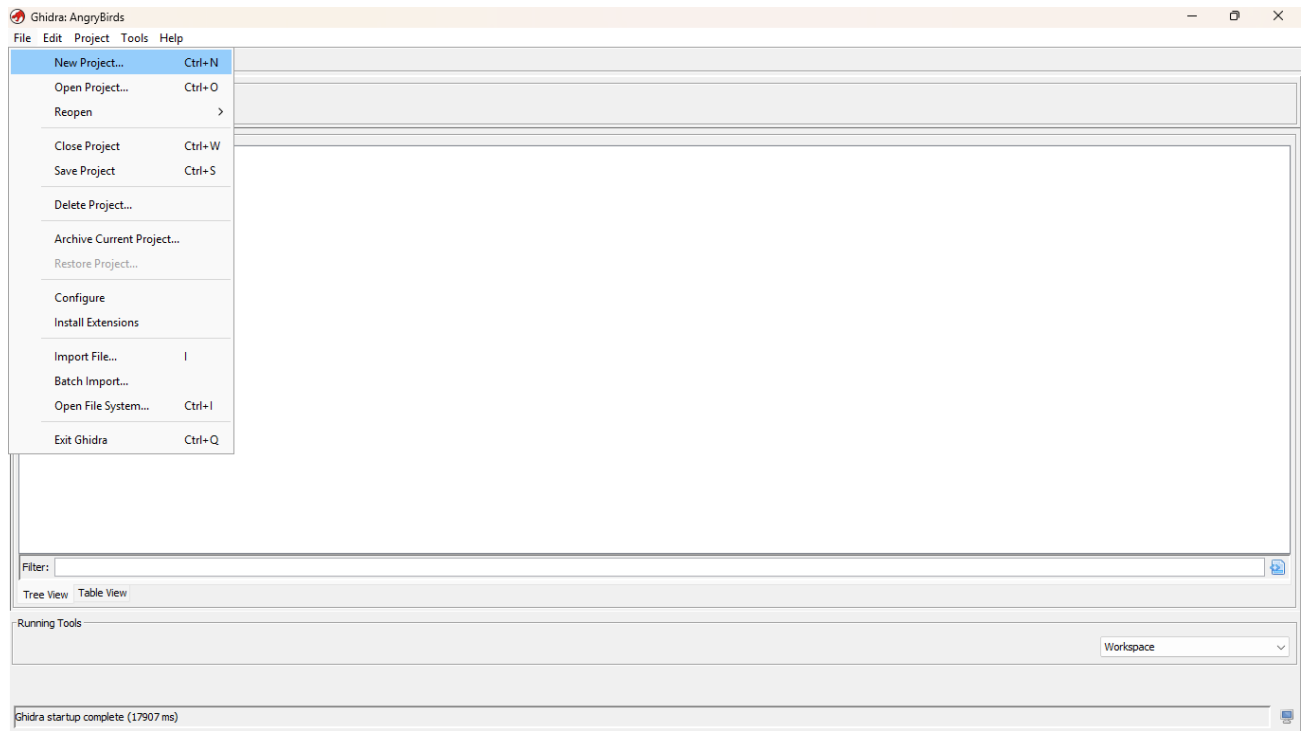
**21590293  
20590283  
21590394**

**ISIC**

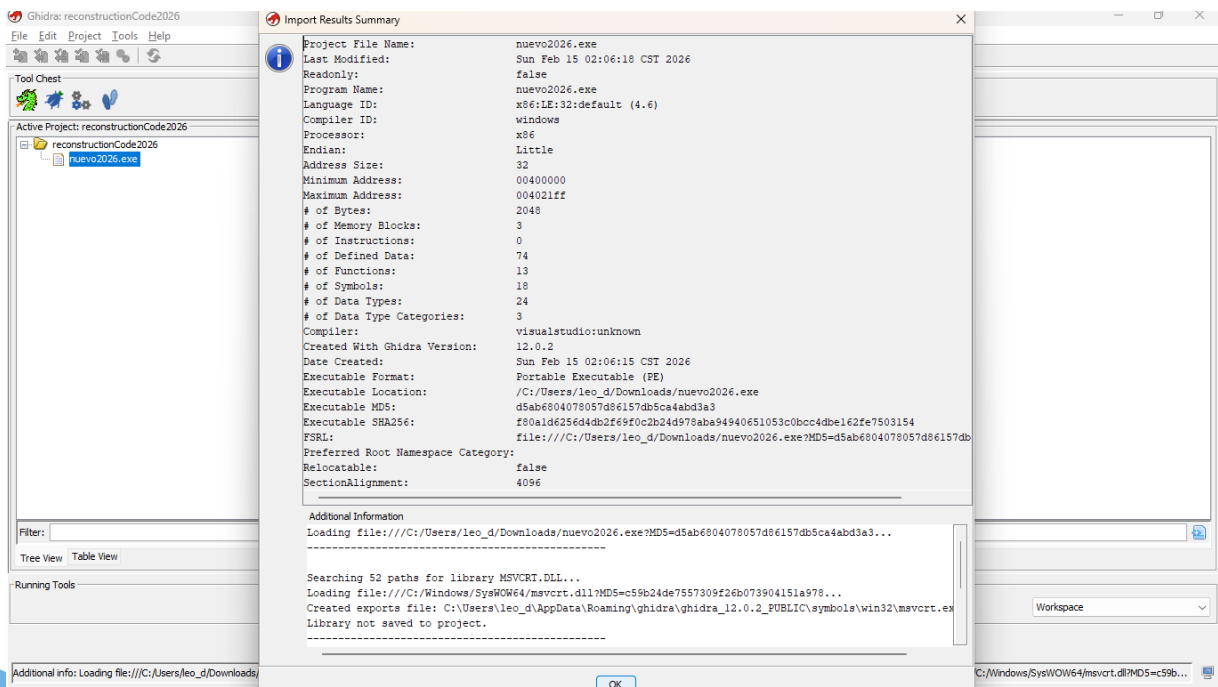
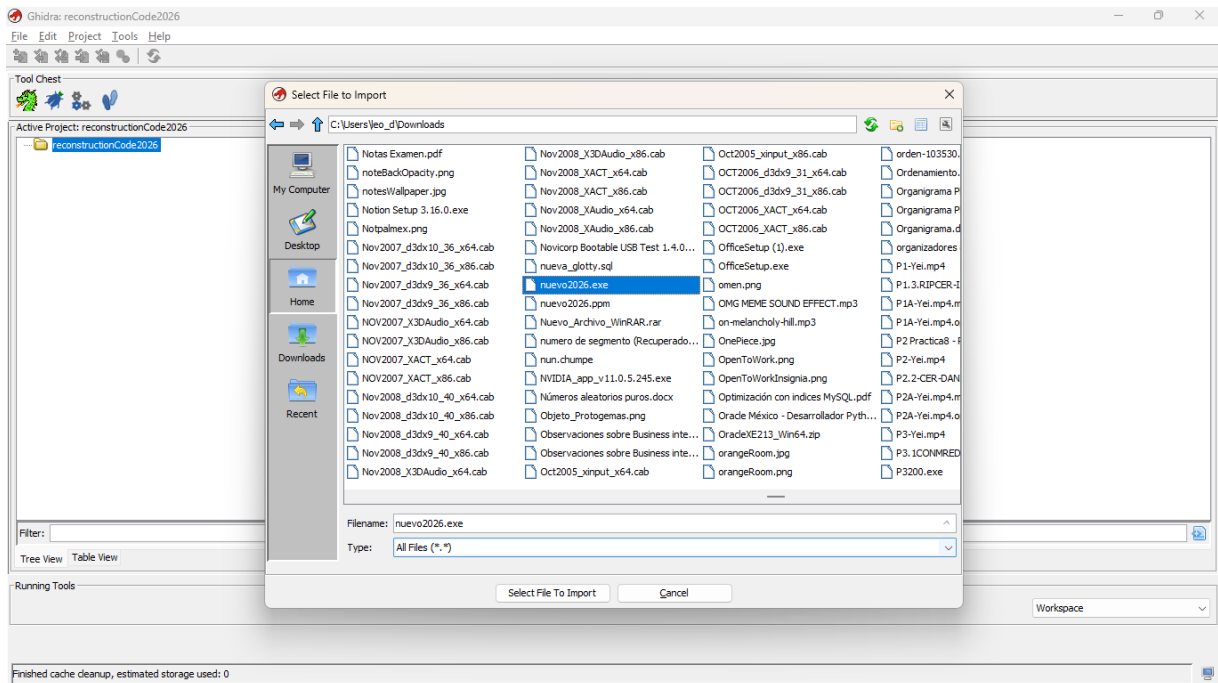
PERIODO [ Ene - Jun ]



Se crea el proyecto y donde se guardará:



Se abre el proyecto y nos muestra información importante:



Aquí podremos leer el código para aplicar la ingeniería inversa:

The screenshot shows the CodeBrowser interface with the file 'reconstructionCode2026/nuevo2026.exe' open. The 'Listing: nuevo2026.exe' window displays assembly code for the 'entry' function. The assembly code includes instructions like PUSH EAX, CALL MSVCRT.DLL::fclose, ADD ESP, 0x4, MOV EAX, 0x0, LEAVE, RET, and several MOV instructions. The 'Decompile: entry - (nu...' window shows the decompiled C++ code for the 'entry' function, which includes a 'void entry(void)' function that calls 'fclose', sets up stack variables, and calls 'FUN\_0040131d'. The 'Program Tree' on the left shows the file structure, and the 'Data Type Manager' at the bottom shows the data types used in the code.

Función principal:

Aquí se muestra la función principal descompilada a la derecha.

The screenshot shows the CodeBrowser interface with the file 'reconstructionCode2026/nuevo2026.exe' open. The 'Listing: nuevo2026.exe - (6144 addresses selected)' window displays assembly code for the 'FUN\_00401000' function. The assembly code includes instructions like PUSH EBP, MOV EBP, ESP, SUB ESP, 0x0, MOV EAX, 0x0, and PUSH EAX. The 'Decompile: FUN\_00401000' window shows the decompiled C++ code for the 'FUN\_00401000' function, which includes a 'void FUN\_00401000(void)' function that sets up variables, calls 'time', and performs a loop of 'fwrite' operations. The 'Program Tree' on the left shows the file structure, and the 'Data Type Manager' at the bottom shows the data types used in the code.

Análisis del código descompilado:

- ✚ Variables identificadas: DAT\_0040219c → archivo (FILE\*)
- ✚ DAT\_00402010 → i (contador filas)
- ✚ DAT\_00402014 → j (contador columnas)
- ✚ 0x1e0 → 480 (altura en decimal)
- ✚ 0x280 → 640 (ancho en decimal)

### Decisiones de reconstrucción:

- ✚ Se identificó que el programa genera una imagen PPM
- ✚ Se determinó el formato: P6 (binario), 640x480, 255 colores max
- ✚ Se identificó una paleta de 6 colores: {0, 50, 100, 150, 200, 250}
- ✚ Se reconstruyeron los bucles anidados para píxeles RGB

### Strings identificadas en Ghidra:

- ✚ "nuevo2026.ppm" → nombre del archivo
- ✚ "wb" → modo de escritura binaria
- ✚ Header PPM: "P6\n640 480\n255\n"

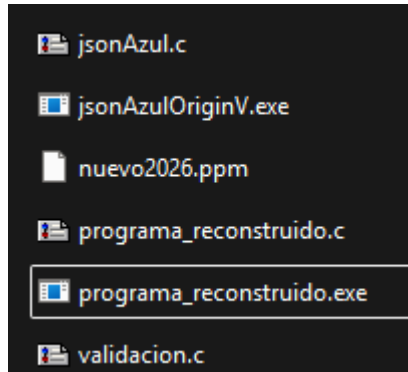
Se toma el código descompilado:

```
Unidad1 > Practica2C > C programa_reconstruido.c > main(void)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main(void)
6  {
7      FILE *archivo;
8      time_t semilla;
9      int i, j;
10     int color;
11     unsigned char paleta[6] = {0, 50, 100, 150, 200, 250}; // Colores de la paleta
12
13     // Inicializar generador de números aleatorios
14     semilla = time(NULL);
15     srand((unsigned int)semilla);
16
17     // Abrir archivo PPM
18     archivo = fopen("nuevo2026.ppm", "wb");
19     if (archivo == NULL)
20     {
21         return 1;
22     }
23
24     // Escribir encabezado PPM (P6 640 480 255)
25     fprintf(archivo, "P6\n640 480\n255\n");
26
27     // Generar imagen de 480 filas x 640 columnas
28     for (i = 0; i < 480; i++)
29     { // 0x1e0 = 480
30         for (j = 0; j < 640; j++)
31         {
```

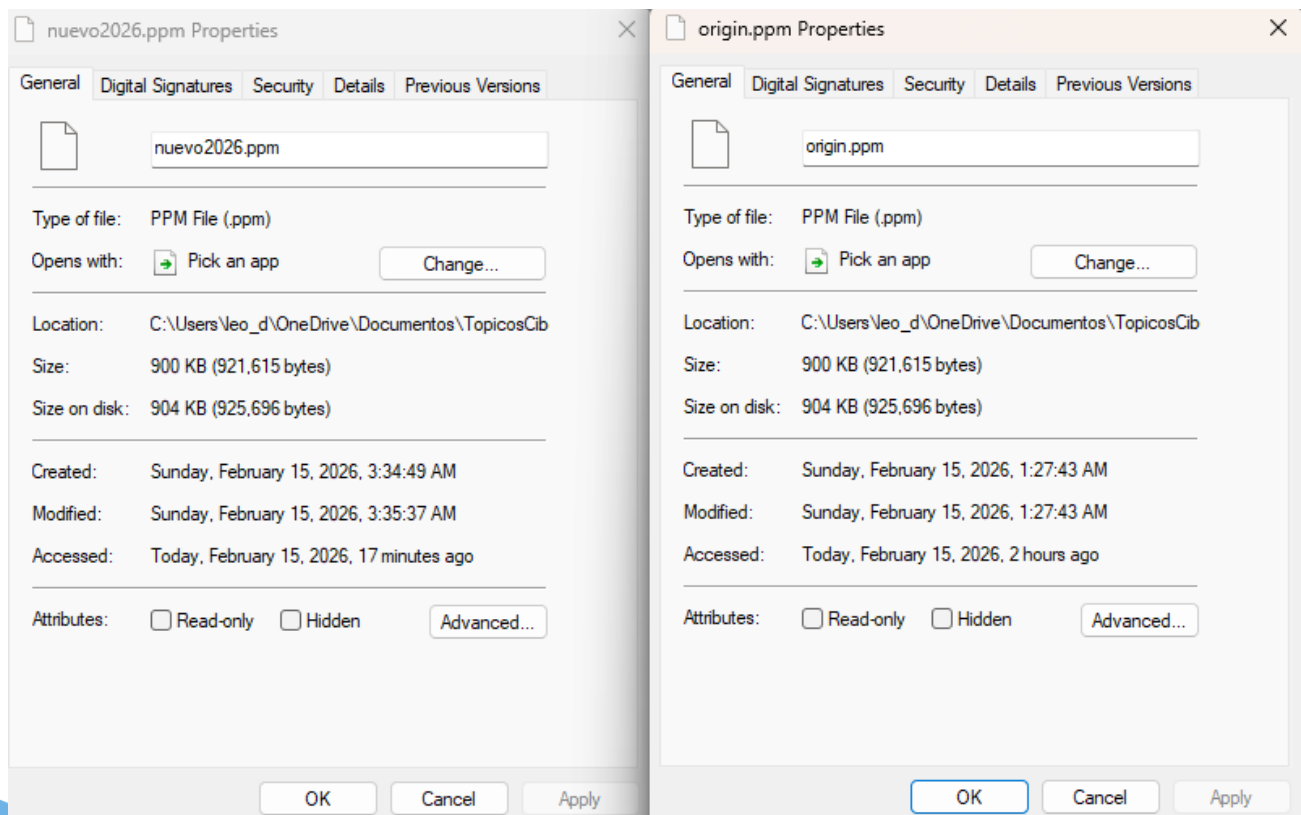
Se compilará:

```
PS C:\Users\leo_d\OneDrive\Documentos\TopicosCiber\Unidad1\Practica2C> tcc -o programa_reconstruido.exe programa_reconstruido.c
PS C:\Users\leo_d\OneDrive\Documentos\TopicosCiber\Unidad1\Practica2C> []
```

Después de ello vamos a ejecutarlo y nos genera el archivo nuevo2026.ppm  
Esto es perfecto porque ya podremos compilar.



Podemos observar que pesan lo mismo y tienen los mismos bytes:



Con esto deducimos lo siguiente:

Aspecto	Original	Reconstruido	¿Coincide?
Se ejecuta sin errores	✓	✓	✓
Genera archivo .ppm	✓	✓	✓
Tamaño del archivo	900 KB (921,615 bytes)	900 KB (921,615 bytes)	✓
Dimensiones	640×480 píxeles	640×480 píxeles	✓
Formato de imagen	PPM (P6)	PPM (P6)	✓
Usa generación aleatoria	✓ (con time() y rand())	✓ (con time() y rand())	✓
Paleta de colores	6 valores	6 valores	✓
Hash del .exe	(diferente)	(diferente)	✗
Contenido visual	Ruido colorido	Ruido colorido	✓
			Funcionalmente

#### Análisis:

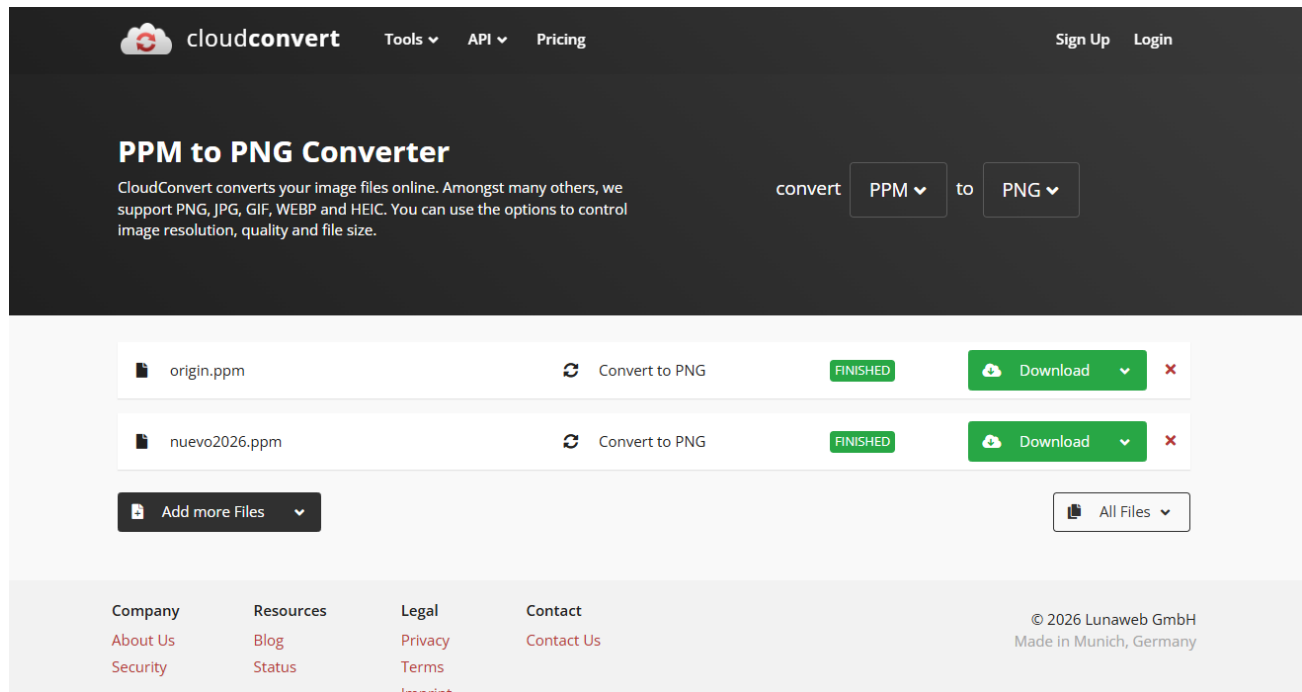
- Los archivos PPM tienen exactamente el mismo tamaño (921,615 bytes)
- Las imágenes son visualmente similares (ruido de colores aleatorios)
- Los ejecutables tienen diferentes hashes porque fueron compilados en momentos distintos
- La **funcionalidad es idéntica**: ambos generan imágenes PPM de 640×480 con colores aleatorios

#### "Dificultades encontradas en la reconstrucción:"

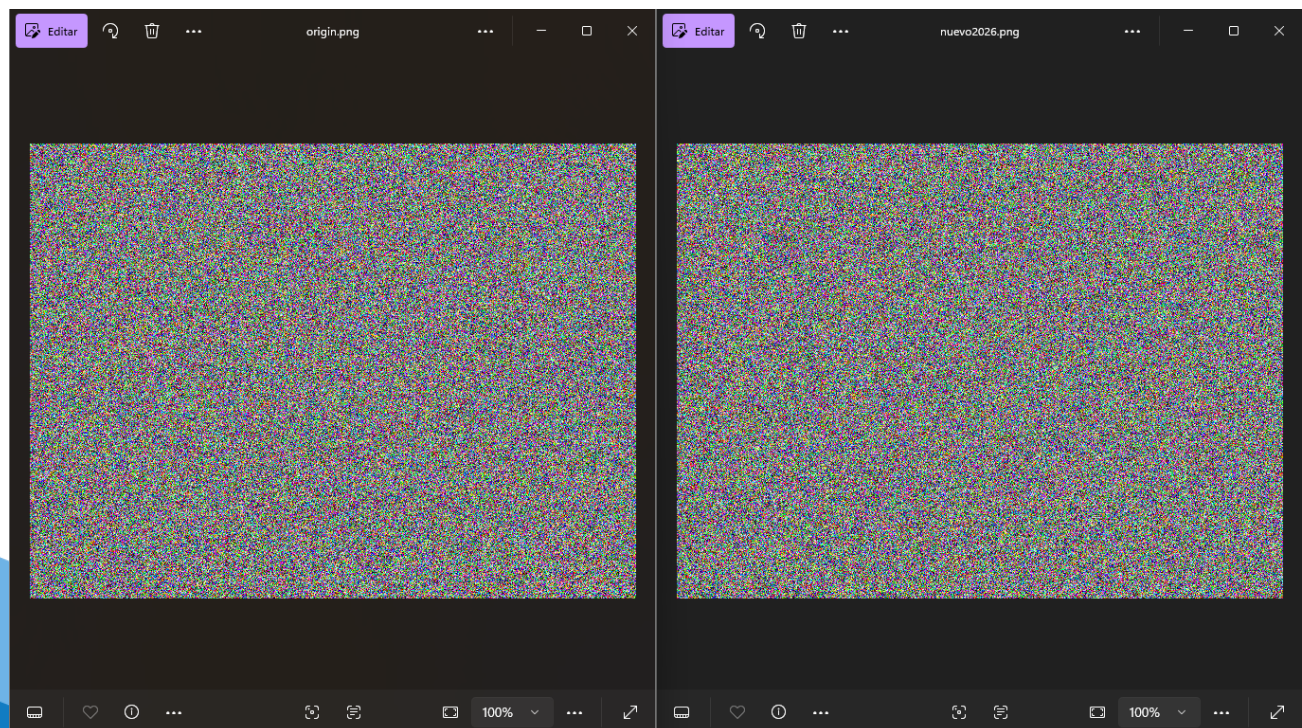
- Ghidra asigna nombres genéricos a variables (DAT\_xxx) que deben ser interpretadas manualmente
- Los valores hexadecimales (0x1e0, 0x280) requieren conversión a decimal para entender las dimensiones
- La paleta de colores no estaba explícita, se dedujo del análisis del código



Con un conversor de archivos se harán a png para visualizarlos:



Y aquí se tiene ambos archivos: original izquierda / nuevo derecha:



## "Conclusiones"

1. **Éxito en la reconstrucción:** Se logró reconstruir el código fuente a partir del ejecutable usando Ghidra.
2. **Funcionalidad preservada:** El programa reconstruido genera archivos PPM idénticos en tamaño y formato al original.
3. **Limitaciones de la ingeniería inversa:**
  - Los nombres de variables originales se pierden
  - La estructura exacta del código puede variar
  - Los ejecutables compilados son diferentes byte a byte
4. **Aplicaciones prácticas:** Esta técnica es útil para auditoría de seguridad, análisis de malware, y recuperación de código legacy.

