



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO®



Instituto Tecnológico de San Juan del Río

Ingeniería en Sistemas Computacionales

Tópicos de Ciberseguridad

Reconstrucción de un Código Fuente usando Ghidra

P R E S E N T A:

Edgar Alfredo Torres Trujillo - 21590398

Jose Luis Velazquez Trejo - 21590299

López Arteaga Giovanni - 21590287

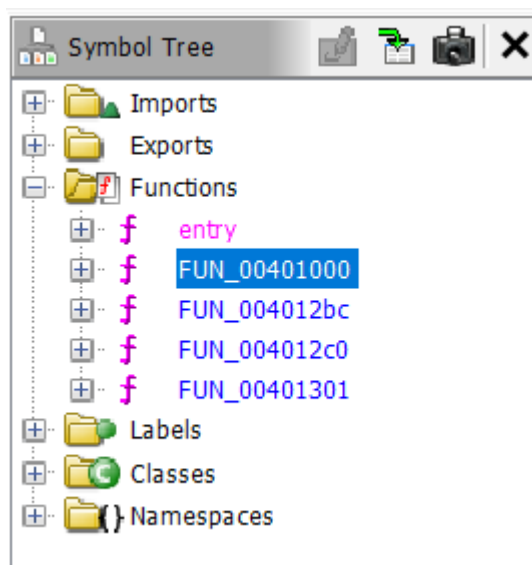
San Juan del Río, Querétaro a 11 de Septiembre 2025

Análisis del Ejecutable con Ghidra

El proceso en Ghidra siguió estos pasos generales:

1. **Importar el Ejecutable:** Se cargó el archivo `color.exe` en Ghidra.
2. **Análisis Automático:** Ghidra analizó el ejecutable, identificando funciones, datos, strings, y llamadas al sistema.
3. **Navegación de Símbolos:** En la ventana **"Symbol Tree"** (Árbol de Símbolos), se identificaron las funciones dentro del programa. La función de interés fue `FUN_00401000`, que Ghidra identificó como la función principal (o entry point) del programa.
 - **FUN_00401000:** Función principal del programa.
 - **FUN_004012bc, FUN_004012c0, FUN_00401301:** Otras funciones identificadas, aunque el foco estuvo en la principal.
4. **Vista de Desensamblado (Assembly):** Ghidra mostró el código en lenguaje ensamblador (x86) de la función `FUN_00401000`. Este código es de muy bajo nivel y es directamente lo que el procesador ejecuta.

Desensamblado del ejecutable:



```
Listing: color.exe

*****
* FUNCTION
*****
undefined4 __stdcall FUN_00401000(void)
    assume FS_OFFSET = 0xfffff000
    EAX:4 <RETURN>
    FUN_00401000 XREF[3]: 004000ac(*), 00400184(*),
    entry:0040121b(c)

00401000 55 PUSH EBP
00401001 89 e5 MOV EBP,ESP
00401003 81 ec 00 SUB ESP,0x0
00 00 00
00401009 90 NOP
0040100a b8 00 00 MOV EAX,0x0
00 00
0040100f 50 PUSH EAX
00401010 e8 2b 03 CALL MSVCRT.DLL::time time_t time(time_t * _Time)
00 00
00401015 83 c4 04 ADD ESP,0x4
00401018 89 05 98 MOV dword ptr [DAT_00402198],EAX
21 40 00
0040101e 8b 05 98 MOV EAX,dword ptr [DAT_00402198]
21 40 00
00401024 50 PUSH EAX
00401025 e8 1e 03 CALL MSVCRT.DLL::srand void srand(uint _Seed)
00 00
0040102a 83 c4 04 ADD ESP,0x4
0040102d b8 23 20 MOV EAX,DAT_00402023 = 77h w
```

Decompilado de la función principal

```
Decompile: FUN_00401000 - (color.exe)

1
2 undefined4 FUN_00401000(void)
3
4 {
5     int iVar1;
6     time_t tVar2;
7
8     tVar2 = time((time_t *)0x0);
9     DAT_00402198 = (uint)tVar2;
10    srand(DAT_00402198);
11    DAT_0040219c = fopen(s_sample.ppm_00402018,&DAT_00402023);
12    fwrite(&DAT_00402000,1,0xf,DAT_0040219c);
13    for (DAT_00402010 = 0; DAT_00402010 < 0x1e0; DAT_00402010 = DAT_00402010 + 1) {
14        for (DAT_00402014 = 0; DAT_00402014 < 0x280; DAT_00402014 = DAT_00402014 + 1) {
15            iVar1 = rand();
16            DAT_0040200f = (undefined1)(iVar1 % 0x100);
17            fwrite(&DAT_0040200f,1,1,DAT_0040219c);
18            iVar1 = rand();
19            DAT_0040200f = (undefined1)(iVar1 % 0x100);
20            fwrite(&DAT_0040200f,1,1,DAT_0040219c);
21            iVar1 = rand();
22            DAT_0040200f = (undefined1)(iVar1 % 0x100);
23            fwrite(&DAT_0040200f,1,1,DAT_0040219c);
24        }
25    }
26    fclose(DAT_0040219c);
27    return 0;
28 }
```

El decompilado de FUN_00401000 reveló la siguiente lógica:


- **Líneas 8-10: Inicialización del Generador de Números Aleatorios**
 - `time((time_t *)0x0);` Obtiene la hora actual (en segundos desde una fecha de referencia, llamada "epoch"). Esto se usa como **semilla** (seed).
 - `srand(DAT_00402198);` Inicializa el generador de números aleatorios (`rand()`) con esa semilla. Esto asegura que la secuencia de números aleatorios sea diferente cada vez que se ejecuta el programa.
- **Línea 11: Apertura de Archivo**
 - `fopen(s_sample.ppm_00402018, &DAT_00402023);` Abre un archivo llamado `sample.ppm` para escritura. El segundo argumento ("`w`") indica el modo "write" (escribir).
- **Línea 12: Escritura de la Cabecera del Archivo**
 - `fwrite(&DAT_00402000, 1, 0xf, DAT_0040219c);` Escribe los primeros 15 bytes (`0xf`) del programa en el archivo. Ghidra mostró que `DAT_00402000` contenía la string "`P6\n640 480\n255\n`", que es la **cabecera de un archivo PPM en formato binario (P6)**. Esta cabecera define:
 1. P6: Tipo de archivo (PPM binario).
 2. 640 480: Dimensiones de la imagen (ancho x alto).
 3. 255: Valor máximo de color.
- **Líneas 13-25: Generación de los Datos de la Imagen (Bucles)**
 - Dos bucles `for` anidados iteran sobre cada píxel de la imagen (480 filas x 640 columnas).
 - **Para cada píxel:**
 1. Se genera un número aleatorio (`rand()`).
 2. Se toma el módulo 256 (`% 0x100`) para obtener un valor entre 0 y 255.
 3. Este valor se escribe en el archivo como el componente **Rojo** del píxel.
 4. Se repiten los pasos 1-3 para el componente **Verde**.
 5. Se repiten los pasos 1-3 para el componente **Azul**.
 - **Conclusión:** El programa genera una imagen de **640x480 píxels** donde el color de cada píxel es **completamente aleatorio (ruido estático)**.
- **Línea 26: Cierre del Archivo**
 - `fclose(DAT_0040219c);` Cierra el archivo correctamente.

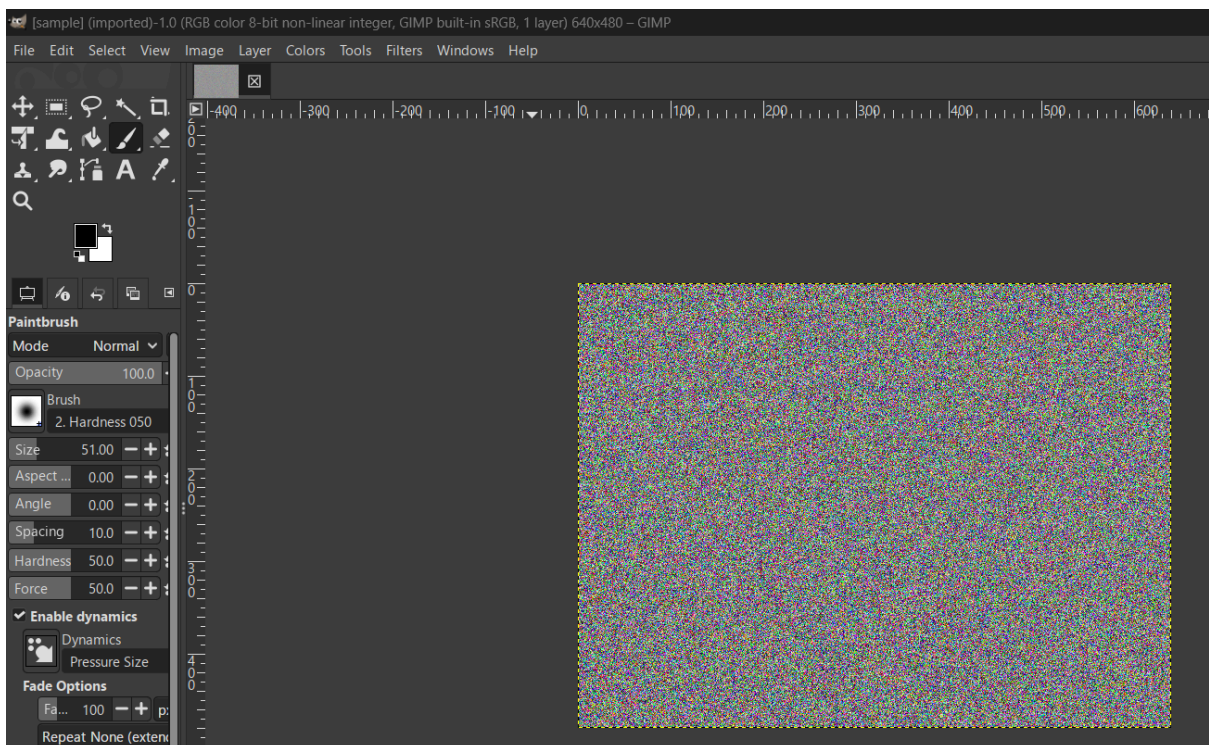
Invocación del ejecutable:

```
C:\Windows\System32\cmd.e  X  +  v

C:\Users\edgar\OneDrive\Nueva carpeta\Documentos\Noveno semestre\Topicos de ciberseguridad\tcc-0.9.27-win32-bin\tcc>color.exe

C:\Users\edgar\OneDrive\Nueva carpeta\Documentos\Noveno semestre\Topicos de ciberseguridad\tcc-0.9.27-win32-bin\tcc>
```

Name	Date modified	Type	Size
 sample	9/15/2025 10:21 PM	PPM File	901 KB



Ejecución del Programa Original

- Se ejecutó el programa original color.exe desde la línea de comandos.
- El resultado fue la creación de un archivo llamado sample.ppm de **901 KB**, lo cual es coherente con el tamaño esperado para una imagen de 640x480 con 3 canales de color ($640 * 480 * 3 = 921,600$ bytes \approx 900 KB).

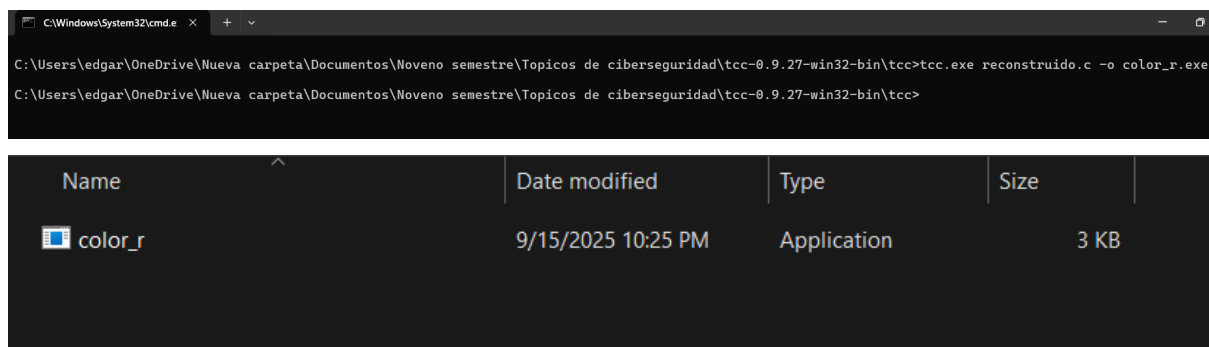
Conversión a código C:

```
reconstruido.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main(void)
6  {
7      FILE *file;
8      int i, j;
9      int random_value;
10     unsigned char pixel_component;
11
12     // Cabecera PPM (P6 indica formato binario, 640x480, máximo valor 255)
13     const char *ppm_header = "P6\n640 480\n255\n";
14
15     // Inicializar generador de números aleatorios
16     srand((unsigned int)time(NULL));
17
18     // Abrir archivo para escritura binaria
19     file = fopen("sample_reconstructed.ppm", "wb");
20     if (file == NULL) {
21         printf("Error: No se pudo crear el archivo\n");
22         return 1;
23     }
24
25     // Escribir cabecera PPM (15 bytes)
26     fwrite(ppm_header, 1, 15, file);
27
28     // Generar imagen de 480x640 píxeles (alto x ancho)
29     for (i = 0; i < 480; i++) {          // 0x1e0 = 480 decimal
30         for (j = 0; j < 640; j++) {      // 0x280 = 640 decimal
31             // Componente Rojo (0-255)
32             random_value = rand();
33             pixel_component = (unsigned char)(random_value % 256);
34             fwrite(&pixel_component, 1, 1, file);
35
36             // Componente Verde (0-255)
37             random_value = rand();
38             pixel_component = (unsigned char)(random_value % 256);
39             fwrite(&pixel_component, 1, 1, file);
40
41             // Componente Azul (0-255)
42             random_value = rand();
43             pixel_component = (unsigned char)(random_value % 256);
44             fwrite(&pixel_component, 1, 1, file);
45         }
46     }
47
48     fclose(file);
49     printf("Archivo sample_reconstructed.ppm generado exitosamente\n");
50     printf("Dimensiones: 640x480 pixels (formato PPM P6)\n");
51
52     return 0;
53 }
```

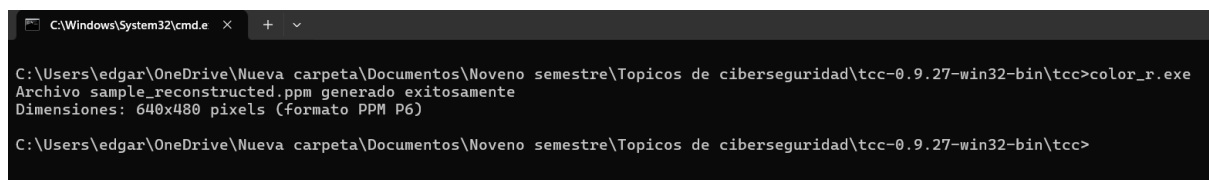
Basándose **exclusivamente en el análisis del decompilado**, se escribió un nuevo programa en C que replica la funcionalidad exacta del original. El código reconstruido es claro y legible:

- **Inclusión de Librerías:** Se incluyen `<stdio.h>`, `<stdlib.h>`, y `<time.h>` para las funciones de archivo, números aleatorios y tiempo, respectivamente.
- **Variables y Cabecera:** Se declaran variables con nombres claros (`i`, `j`, `random_value`, `pixel_component`) y se define explícitamente la cabecera PPM.
- **Lógica Replicada:**
 1. Inicializar `srand()` con `time(NULL)`.
 2. Abrir el archivo `"sample_reconstructed.ppm"` en modo escritura binaria (`"wb"`).
 3. Escribir la cabecera PPM.
 4. Usar dos bucles `for` para generar 3 valores aleatorios (R, G, B) por cada píxel y escribirlos en el archivo.
 5. Cerrar el archivo e imprimir un mensaje de confirmación.

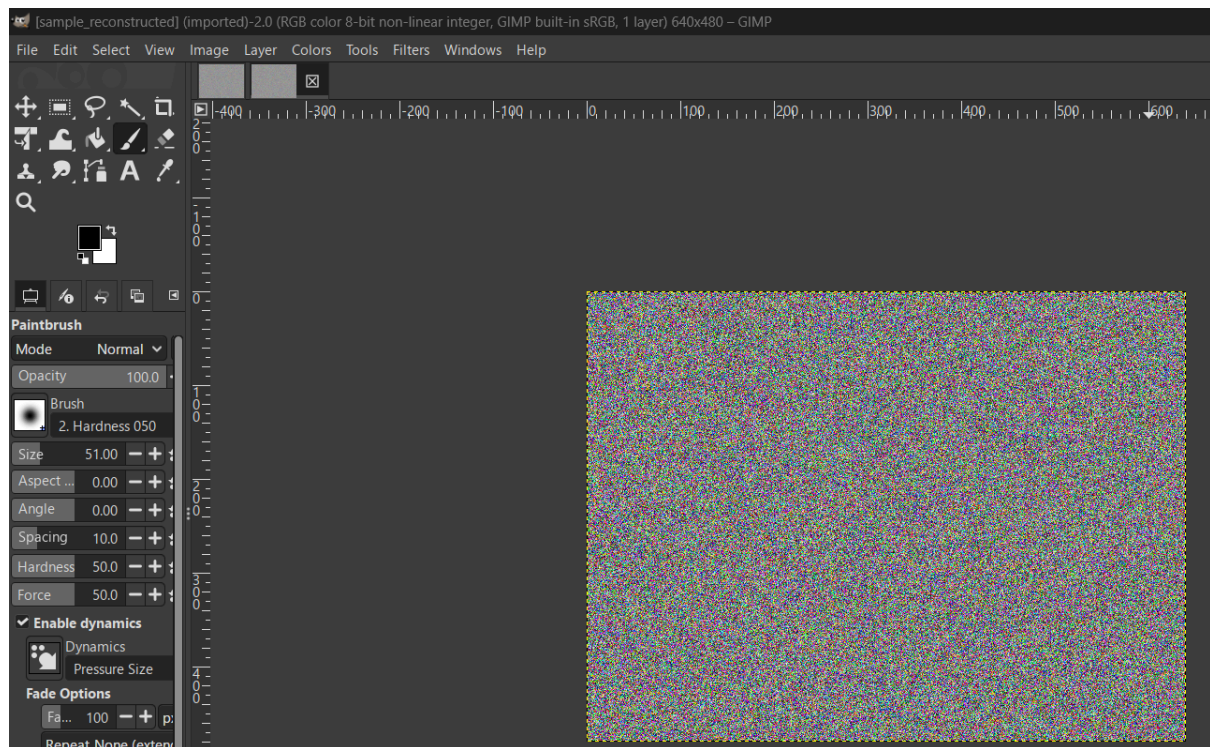
Compilación del código reconstruido:



Invocación al nuevo ejecutable que realiza la misma función:



Resultado final:



Compilación y Verificación del Nuevo Código

- El código reconstruido (reconstruido.c) se compiló usando Tiny C Compiler para generar un nuevo ejecutable (color_r.exe).
- Al ejecutar color_r.exe, se creó exitosamente el archivo sample_reconstructed.ppm con las mismas dimensiones (640x480) y formato (PPM P6) que el original, confirmando que la reconstrucción fue un **éxito total**.