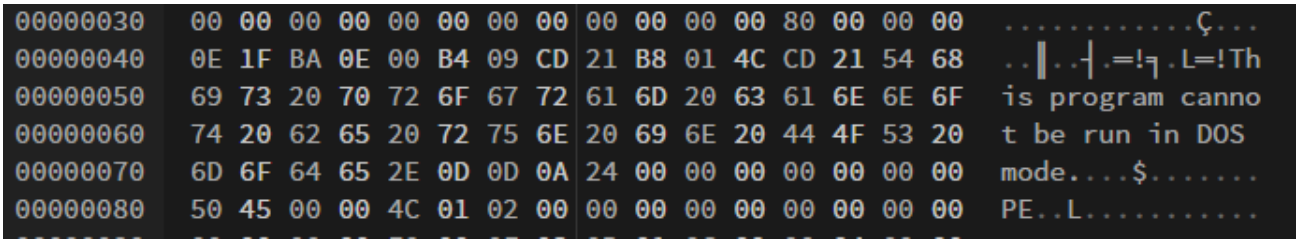
Instituto Tecnológico de San Juan del Río

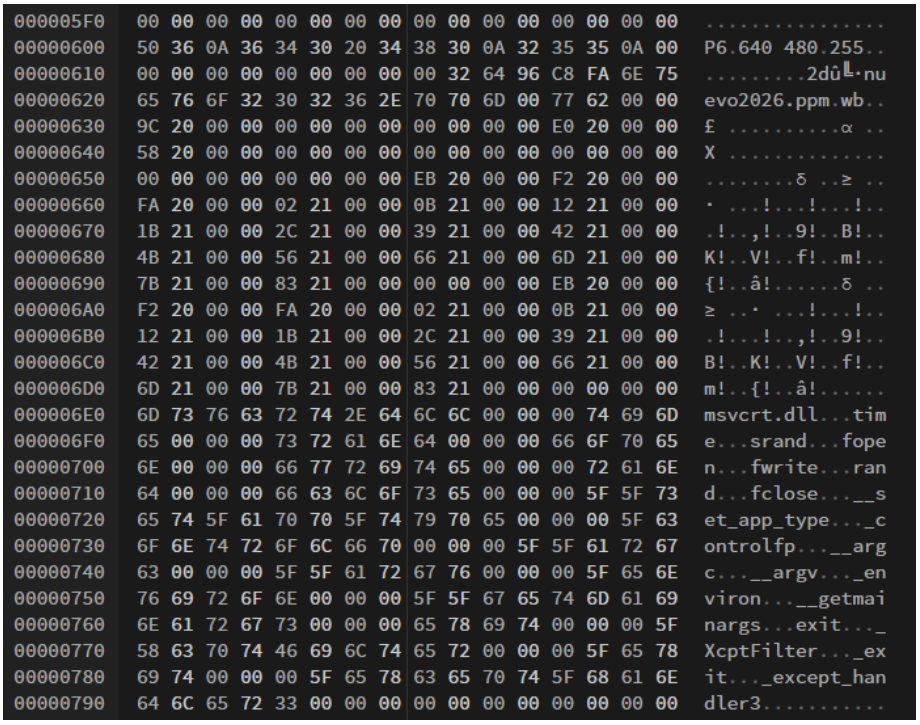
Andres Martin Sinecio
Colunga Ochoa Pablo
Javier Hernandez Mendoza

PERIODO [Enero-Junio 2026]



```
00000030   00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00   ............Ç...
00000040   0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ..▌..┤.=!╕.L=!Th
00000050   69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F   is program canno
00000060   74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20   t be run in DOS
00000070   6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00   mode....$.......
00000080   50 45 00 00 4C 01 02 00 00 00 00 00 00 00 00 00   PE..L...........
```

Aquí confirmamos que el archivo es un ejecutable de Windows. Se ve el mensaje típico del stub DOS y, más importante, la firma PE\0\0, que identifica el formato Portable Executable. Esto valida que Ghidra lo trate como un binario PE y pueda analizar sus secciones y funciones.”



```
000005F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00000600   50 36 0A 36 34 30 20 34 38 30 0A 32 35 35 0A 00   P6.640 480.255..
00000610   00 00 00 00 00 00 00 00 00 32 64 96 C8 FA 6E 75   .........2dû╚·nu
00000620   65 76 6F 32 30 32 36 2E 70 70 6D 00 77 62 00 00   evo2026.ppm.wb..
00000630   9C 20 00 00 00 00 00 00 00 00 00 00 E0 20 00 00   £ ..........α ..
00000640   58 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00   X ..............
00000650   00 00 00 00 00 00 00 00 EB 20 00 00 F2 20 00 00   ........δ ..≥ ..
00000660   FA 20 00 00 02 21 00 00 0B 21 00 00 12 21 00 00   · ...!...!...!..
00000670   1B 21 00 00 2C 21 00 00 39 21 00 00 42 21 00 00   .!..,!..9!..B!..
00000680   4B 21 00 00 56 21 00 00 66 21 00 00 6D 21 00 00   K!..V!..f!..m!..
00000690   7B 21 00 00 83 21 00 00 00 00 00 00 EB 20 00 00   {!..â!......δ ..
000006A0   F2 20 00 00 FA 20 00 00 02 21 00 00 0B 21 00 00   ≥ ..· ...!...!..
000006B0   12 21 00 00 1B 21 00 00 2C 21 00 00 39 21 00 00   .!...!..,!..9!..
000006C0   42 21 00 00 4B 21 00 00 56 21 00 00 66 21 00 00   B!..K!..V!..f!..
000006D0   6D 21 00 00 7B 21 00 00 83 21 00 00 00 00 00 00   m!..{!..â!......
000006E0   6D 73 76 63 72 74 2E 64 6C 6C 00 00 00 74 69 6D   msvcrt.dll...tim
000006F0   65 00 00 00 73 72 61 6E 64 00 00 00 66 6F 70 65   e...srand...fope
00000700   6E 00 00 00 66 77 72 69 74 65 00 00 00 72 61 6E   n...fwrite...ran
00000710   64 00 00 00 66 63 6C 6F 73 65 00 00 00 5F 5F 73   d...fclose..._s
00000720   65 74 5F 61 70 70 5F 74 79 70 65 00 00 00 5F 63   et_app_type..._c
00000730   6F 6E 74 72 6F 6C 66 70 00 00 00 5F 5F 61 72 67   ontrolfp..._arg
00000740   63 00 00 00 5F 5F 61 72 67 76 00 00 00 5F 65 6E   c..__argv..._en
00000750   76 69 72 6F 6E 00 00 00 5F 5F 67 65 74 6D 61 69   viron..__getmai
00000760   6E 61 72 67 73 00 00 00 65 78 69 74 00 00 00 5F   nargs...exit..._
00000770   58 63 70 74 46 69 6C 74 65 72 00 00 00 5F 65 78   XcptFilter..._ex
00000780   69 74 00 00 00 5F 65 78 63 65 70 74 5F 68 61 6E   it..._except_han
00000790   64 6C 65 72 33 00 00 00 00 00 00 00 00 00 00 00   dler3...........
```

“Aquí está la evidencia más clara: el header P6 640 480 255 demuestra que el programa genera una imagen PPM binaria de 640×480; nuevo2026.ppm y wb indican el archivo y el modo de escritura; y los imports time/srand/rand/fwrite confirman que los pixeles se

generan con aleatoriedad y se escriben byte por byte.”

Decompile: FUN_00401000 - (nuevo2026.exe)

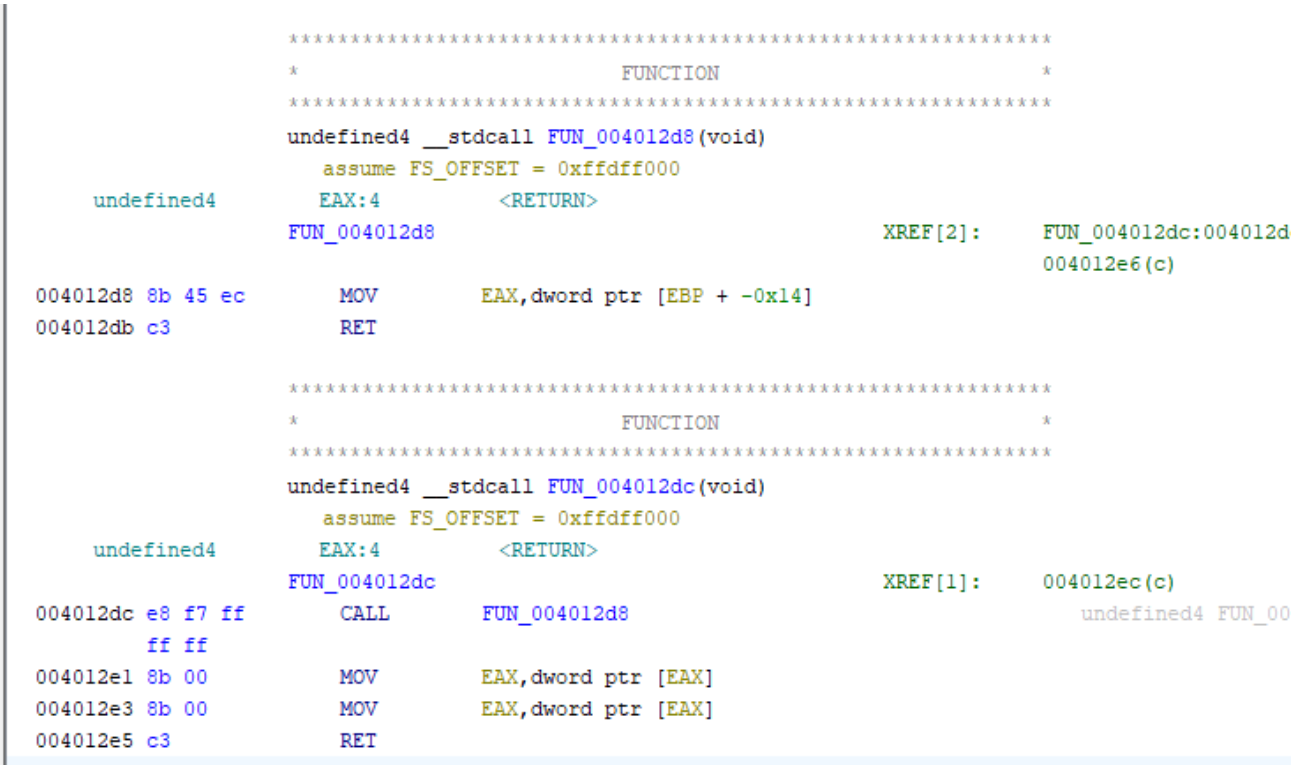
```
1
2  undefined4 FUN_00401000(void)
3
4  {
5    int iVar1;
6    time_t tVar2;
7
8    tVar2 = time((time_t *)0x0);
9    DAT_00402198 = (uint)tVar2;
10   srand(DAT_00402198);
11   DAT_0040219c = fopen(s_nuevo2026.ppm_0040201e,&DAT_0040202c);
12   fwrite(&DAT_00402000,1,0xf,DAT_0040219c);
13   for (DAT_00402010 = 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 % 6);
17       fwrite(&DAT_00402018 + (iVar1 % 6 & 0xff),1,1,DAT_0040219c);
18       iVar1 = rand();
19       DAT_0040200f = (undefined1)(iVar1 % 6);
20       fwrite(&DAT_00402018 + (iVar1 % 6 & 0xff),1,1,DAT_0040219c);
21       iVar1 = rand();
22       DAT_0040200f = (undefined1)(iVar1 % 6);
23       fwrite(&DAT_00402018 + (iVar1 % 6 & 0xff),1,1,DAT_0040219c);
24     }
25   }
26   fclose(DAT_0040219c);
27   return 0;
28 }
```

“Esta es la función principal del programa. Primero toma la hora actual y la usa como semilla para rand(). Luego abre nuevo2026.ppm en modo binario, escribe el encabezado P6 640 480 255 y recorre 480×640 pixeles. Para cada pixel escribe tres bytes RGB; cada byte se elige con rand()%6 desde una tabla de seis intensidades. Al final cierra el archivo y retorna 0.”

Decompile: entry - (nuevo2026.exe)

```
1
2  void entry(void)
3
4  {
5    int _Code;
6    _startupinfo local_20;
7    int local_1c [6];
8
9    FUN_0040131d(local_1c);
10   local_20.newmode = 0;
11   __set_app_type(1);
12   _controlfp(0x10000,0x30000);
13   __getmainargs((int *)__argc_exref,(char ***)__argv_exref,(char ***)_environ_exref,DAT_00402la0,
14                 &local_20);
15   _Code = FUN_00401000();
16                     /* WARNING: Subroutine does not return */
17   exit(_Code);
18 }
19
```

“Esta función entry() es el punto de entrada del ejecutable. No contiene la lógica del programa; es el runtime de C preparando excepciones, argumentos y el entorno. La parte clave es que después de inicializar todo, llama a FUN_00401000(). Esa llamada nos indica cuál es la función que realmente implementa el comportamiento del ejecutable, y al terminar pasa el código de retorno a exit().”



```
                    ****************************************************************
                    *                          FUNCTION                            *
                    ****************************************************************
                    undefined4 __stdcall FUN_004012d8(void)
                      assume FS_OFFSET = 0xffdff000
    undefined4        EAX:4          <RETURN>
                    FUN_004012d8                                    XREF[2]:     FUN_004012dc:004012d
                                                                                 004012e6(c)
004012d8 8b 45 ec       MOV        EAX,dword ptr [EBP + -0x14]
004012db c3             RET

                    ****************************************************************
                    *                          FUNCTION                            *
                    ****************************************************************
                    undefined4 __stdcall FUN_004012dc(void)
                      assume FS_OFFSET = 0xffdff000
    undefined4        EAX:4          <RETURN>
                    FUN_004012dc                                    XREF[1]:     004012ec(c)
004012dc e8 f7 ff       CALL       FUN_004012d8                                     undefined4 FUN_00
         ff ff
004012e1 8b 00          MOV        EAX,dword ptr [EAX]
004012e3 8b 00          MOV        EAX,dword ptr [EAX]
004012e5 c3             RET
```

“Aquí vemos dos funciones muy pequeñas generadas alrededor de accesos a memoria. La primera solo regresa un valor guardado en la pila ([EBP-0x14]). La segunda llama a la primera y luego hace doble desreferencia ([EAX] dos veces), lo que indica una cadena de punteros, típico de estructuras del runtime como argv o environ. Esto confirma que parte del código que vemos no es el algoritmo principal, sino helpers del CRT y manejo de punteros.”

Decompile: UndefinedFunction_004012e6 - (nuevo2026.exe)

```
1
2  void UndefinedFunction_004012e6(void)
3
4  {
5    _EXCEPTION_POINTERS *_ExceptionPtr;
6    ulong _ExceptionNum;
7
8    _ExceptionPtr = (_EXCEPTION_POINTERS *)FUN_004012d8();
9    _ExceptionNum = FUN_004012dc();
10   _XcptFilter(_ExceptionNum,_ExceptionPtr);
11   return;
12 }
13
```

“esta función existe para procesar excepciones”.

```
                    DAT_0040130c                                    XREF[2]:     FUN_0040131d:0040133
                                                                                 FUN_0040131d:0040134
  0040130c ff              ??         FFh
  0040130d ff              ??         FFh
  0040130e ff              ??         FFh
  0040130f ff              ??         FFh
  00401310 e6 12 40 00     addr       LAB_004012e6
  00401314 fb 12 40 00     addr       DAT_004012fb                       = 8Bh

                    LAB_00401318                                    XREF[2]:     FUN_0040131d:0040133
                                                                                 FUN_0040131d:0040133
  00401318 e9 9b 00        JMP        MSVCRT.DLL::except_handler3        undefined except_
           00 00
                    -- Flow Override: CALL_RETURN (CALL_TERMINATOR)
```

Aquí se ve la prueba de cómo el ejecutable instala su manejo de excepciones. FUN_0040131d referencia esta tabla DAT_0040130c y el handler LAB_00401318. El handler no es del programa: hace un JMP directo a MSVCRT::except_handler3. Y la tabla apunta a LAB_004012e6, que es la función que llama a XcptFilter usando _EXCEPTION_POINTERS. Esto confirma que estas partes pertenecen al runtime y no al algoritmo que genera la imagen.”

Decompile: FUN_0040131d - (nuevo2026.exe)

```
1
2 void __cdecl FUN_0040131d(int *param_1)
3
4 {
5   *param_1 = (int)&stack0x00000008;
6   param_1[1] = 0;
7   param_1[2] = (int)ExceptionList;
8   param_1[3] = (int)&LAB_00401318;
9   param_1[4] = (int)&DAT_0040130c;
10  param_1[5] = 0;
11  ExceptionList = param_1 + 2;
12  return;
13 }
14
```

“Esta función instala el manejo de excepciones del programa. Recibe un arreglo en la pila y lo llena como un registro SEH: guarda el handler anterior (ExceptionList), asigna el handler (LAB_00401318, que salta a except_handler3) y apunta a una tabla de control (DAT_0040130c). Al final actualiza ExceptionList para que este frame sea el primero en la cadena. Esto es código del runtime, no del algoritmo que genera la imagen.”

Decompile: time - (nuevo2026.exe)

```
1
2 time_t __cdecl time(time_t *_Time)
3
4 {
5   time_t tVar1;
6
7                     /* WARNING: Could not
8                     /* WARNING: Treating
9   tVar1 = time(_Time);
10  return tVar1;
11 }
12
```

“Aquí Ghidra muestra time() como si se llamara a sí misma, pero esto no es recursión real. Es un ‘thunk’ o stub de importación: un puente que redirige a la función real time() del runtime msvcrt.dll. Esto confirma que el programa depende del CRT para obtener la hora y usarla como semilla para generar la imagen.”