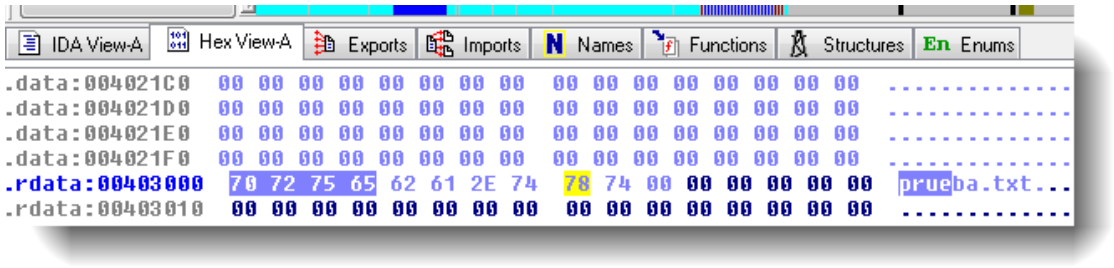
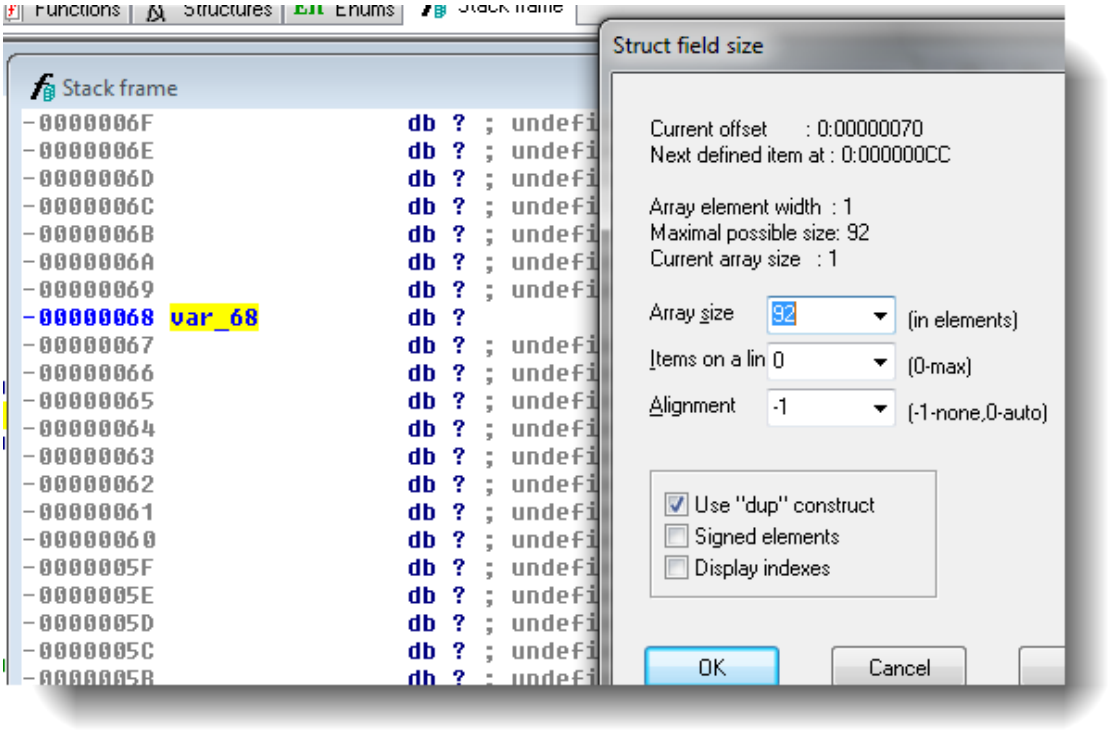
Vemos el comienzo de la función que es igual al comentado en la parte 10:

```
var_D8= dword ptr -0D8h
var_D4= dword ptr -0D4h
var_D0= dword ptr -0D0h
var_C8= byte ptr -0C8h
var_68= dword ptr -68h
var_64= dword ptr -64h
var_60= word ptr -60h
var_5E= byte ptr -5Eh
var_5D= byte ptr -5Dh
var_C= dword ptr -0Ch

push    ebp
mov     ebp, esp
sub     esp, 0D8h
mov     eax, ds:dword_403000
mov     [ebp+var_68], eax
mov     eax, ds:dword_403004
mov     [ebp+var_64], eax
movzx   eax, ds:word_403008
mov     [ebp+var_60], ax
movzx   eax, ds:byte_40300A
mov     [ebp+var_5E], al
lea     edx, [ebp+var_5D]
```

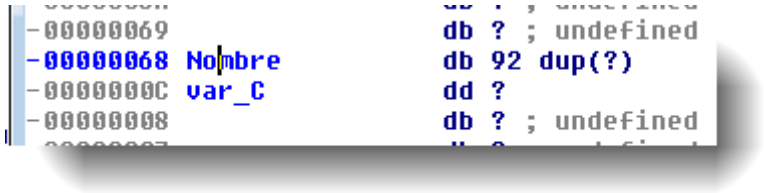
Clickamos en la dirección seleccionada y vemos que cadena se está guardando.



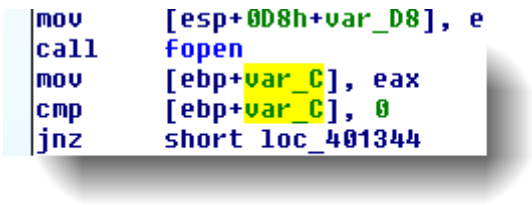
Es el archivo “prueba.txt”, así que renombraremos la variable var_68 (undefiniendo las **var_64, var_60, var_5E, var_5D**, para que ocupe todo el buffer)



Hay 92 bytes libres, ocuparemos todo el buffer, y le ponemos y lo renombraremos a 'nombre' por guardar el nombre del archivo.

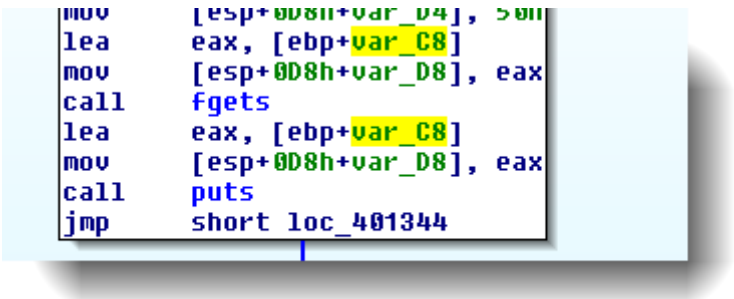


Var_C

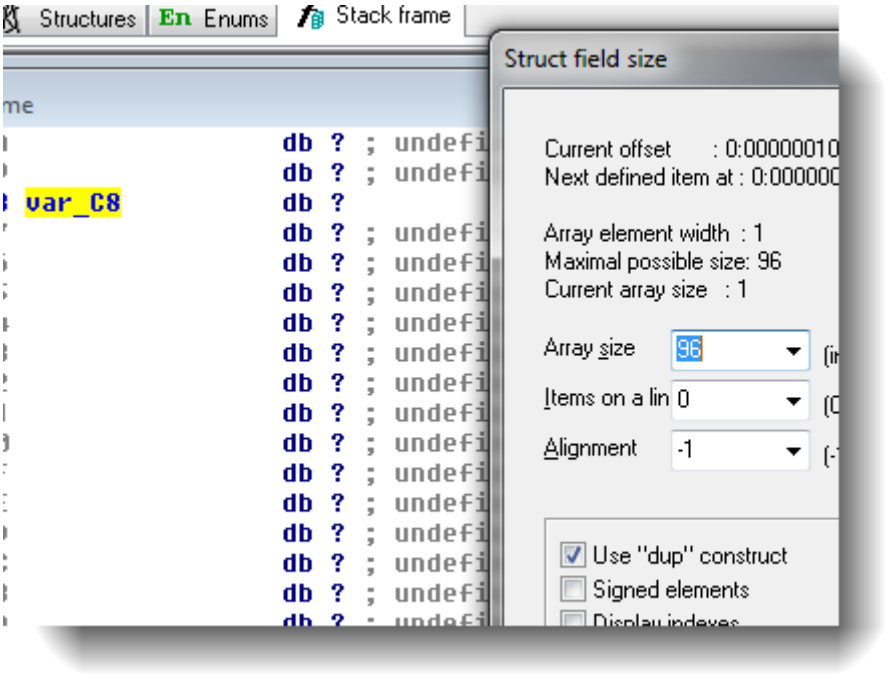


Se guarda el valor devuelto por fopen, por lo que aquí se guarda el handle del archivo. Lo renombraremos como tal.

Var_C8

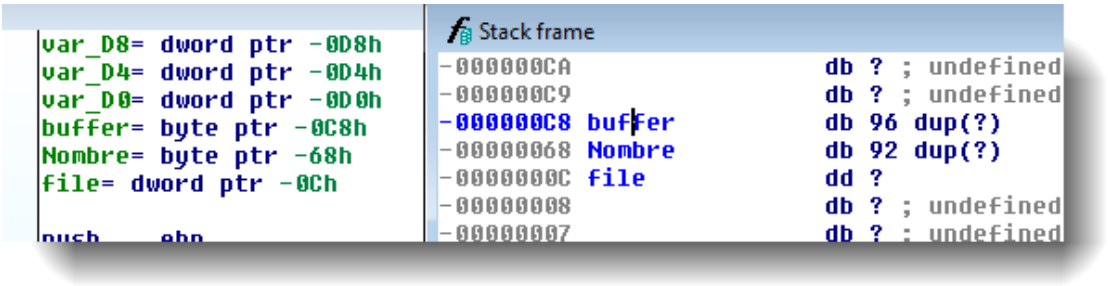


Como hemos visto en los ejemplos anteriores, el ultimo parámetro de fgets es el buffer donde dejaremos lo leído en el archivo (que también será parámetro de entrada en el puts), Este será nuestro buffer.



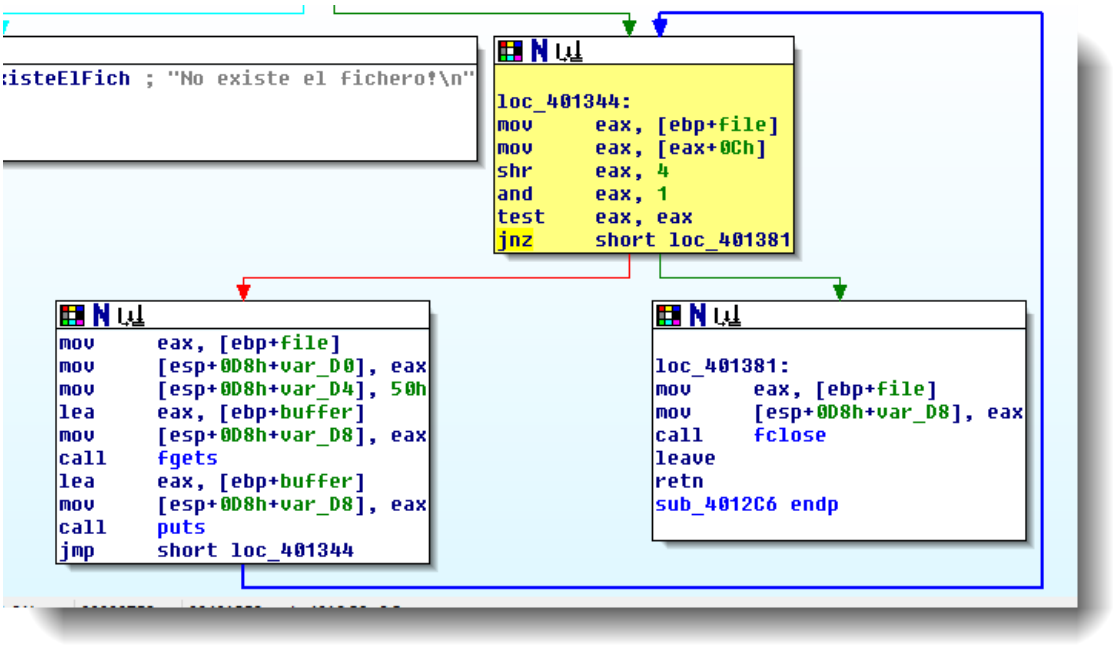
El buffer tiene sitio para 96 bytes. Lo ocupamos todo.

Nos quedará asi la lista de parámetros:

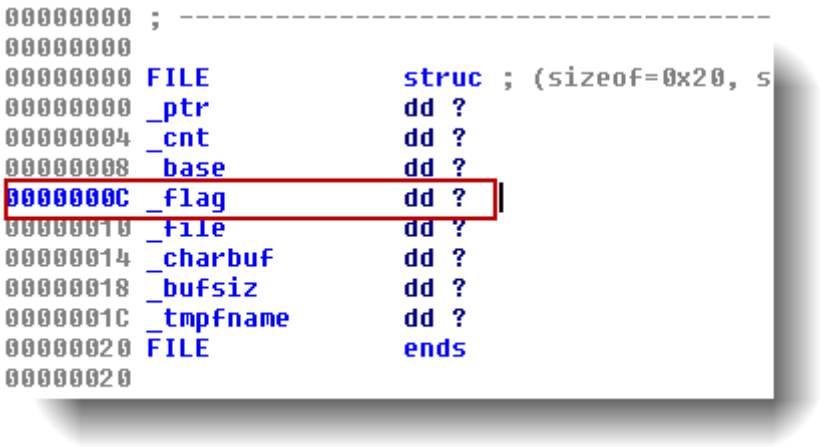


Buscando la diferencia

Hasta ahora el código es similar al visto en la parte 10 del curso del Ricardo, pero si miramos el código vemos una notable diferencia.



El bloque coloreado es el añadido en este ejemplo. Si seguimos el flujo del programa vemos que se van leyendo líneas del archivo hasta que file+0Ch tenga un valor determinado. Como nos explica Ricardo en la parte 10, file es un puntero a la estructura FILE, en el IDA la vemos y marcamos el elemento en 0Ch:



El elemento que nos interesa es el _flag. No hace falta mucha imaginación para intuir que el código anterior, leerá el fichero hasta el final, ahí interviene este flag indicando que hemos sobrepasado el eof del archivo.

En C para saber cuando se llega al final del archivo se utiliza la función feof()

Función feof ANSI C

```
int feof(FILE *stream);
```

La función *feof* comprueba el indicador de final de fichero para el stream apuntado por **stream**.

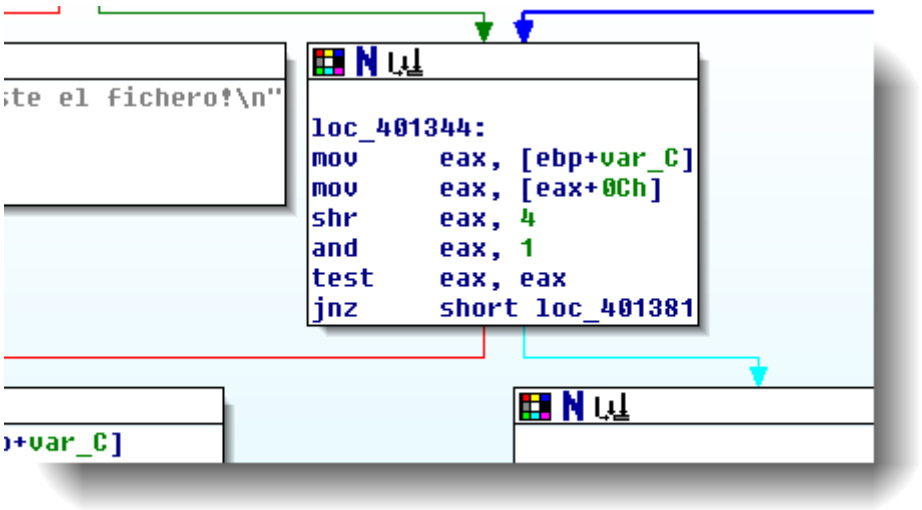
Valor de retorno:

La función *feof* retorna un valor distinto a cero si y sólo si el indicador de final de fichero está activado para **stream**.

Hacemos una prueba con el siguiente código para ver el resultado:

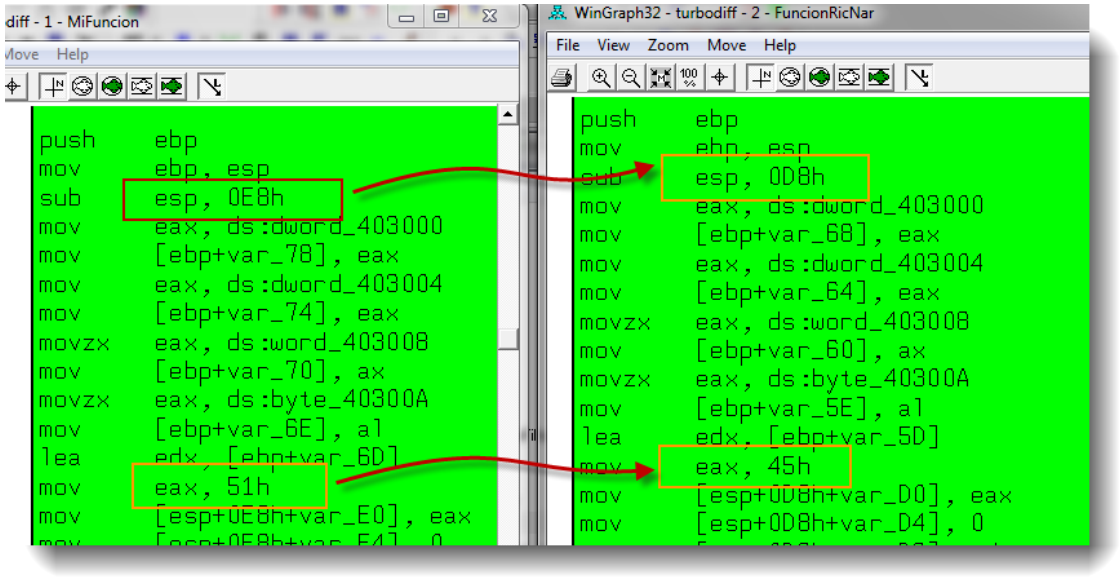
```
funcion()
{

    FILE* fichero;
    char nombre[92] = "prueba.txt";
    char buffer[96];

    fichero = fopen(nombre, "rt");

    if (fichero == NULL)
    {
      printf("No existe el fichero!\n");
      exit(1);
    }
    while(!feof (fichero)){
        fgets(buffer, 80, fichero);
        puts(buffer);
        }
    fclose(fichero);

}
```

El resultado:



Es exacta. Hemos dado en el clavo.

Para asegurar el resultado probamos con e turbodiff, para ver si esta todo igual, pero los colores nos indican que hay diferencias:



Vemos que en mi función se utilizan para rellenar el buffer con ceros 51h bytes en lugar de los 45h bytes del programa original. Esto quiere decir que hemos utilizado 12 bytes (51h – 45h) de más en el array donde guardamos el nombre. Lo habíamos declarado como:

char nombre[92] = "prueba.txt";

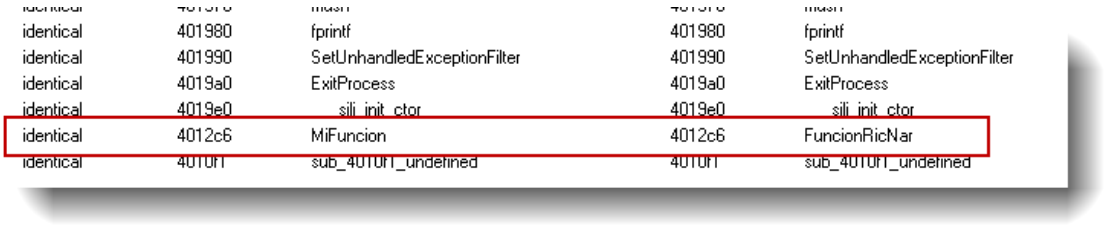
Corregimos los 12 bytes de más:

char nombre[80] = "prueba.txt";

Y volvemos a compilar

```
funcion()
{

    FILE* fichero;
    char nombre[80] = "prueba.txt";
    char buffer[96];

    fichero = fopen(nombre, "rt");

    if (fichero == NULL)
    {
      printf("No existe el fichero!\n");
      exit(1);
    }
    while(!feof (fichero)){
        fgets(buffer, 80, fichero);
        puts(buffer);
        }
    fclose(fichero);
}
```

y comparar con el turbodiff:



identical	401980	fprintf	401980	fprintf
identical	401990	SetUnhandledExceptionFilter	401990	SetUnhandledExceptionFilter
identical	4019a0	ExitProcess	4019a0	ExitProcess
identical	4019e0	sili_init_ctor	4019e0	sili_init_ctor
identical	4012c6	MiFuncion	4012c6	FuncionRicNar
identical	4010f1	sub_4010f1_undefined	4010f1	sub_4010f1_undefined

Identico.

Saludos y hasta la próxima.