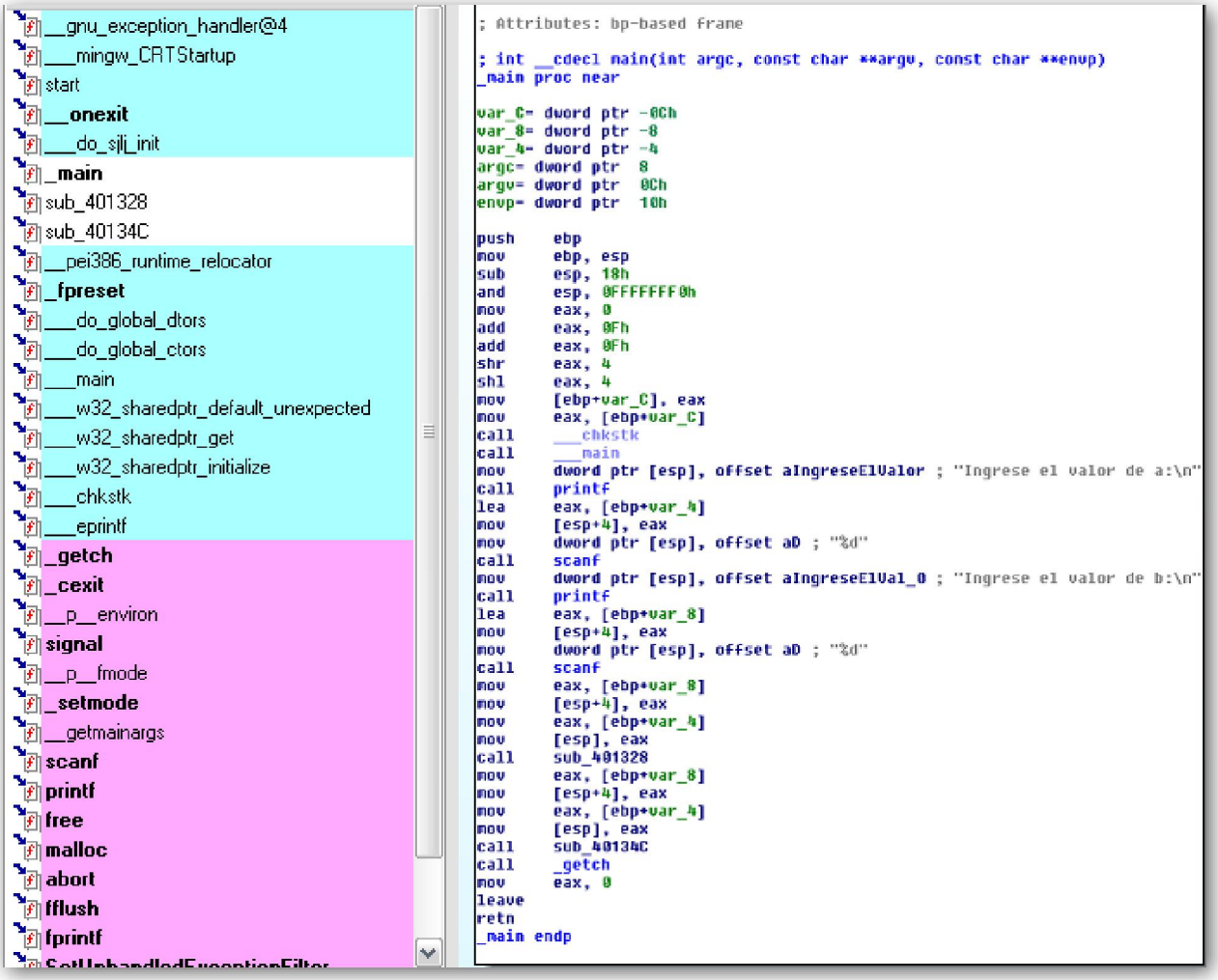
EjemploRic

Como siempre ejecutamos el binario y vemos lo que nos muestra en pantalla para poder tener una ligera idea a lo que nos vamos a enfrentar. Después de ir ejecutándolo vemos lo siguiente:

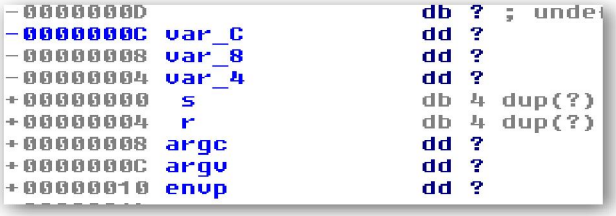
```
Ingrese el valor de a:
4
Ingrese el valor de b:
3
El valor de la suma es 7:

El valor de a es mayor que el de b
```

Como dice el dicho “una imagen vale más que mil palabras”, de ésta podemos deducir una serie de conceptos ; es un programa que nos pide introducir dos valores, una vez introducidos nos muestra su suma y una comparación de valor entre ellos.

Bien, en principio parece que será una binario de mínima dificultad, ahora para determinar lo que suponemos y estudiarlo a fondo lo “destriparemos” con nuestra amiga IDA.



Ahí tenemos a nuestra querida función “**main**”. Echemos un vistazo al “**stack**”, vemos a nuestras tres variables por encima de “**sr**” y los argumentos de compilación por debajo de “**sr**”. Sigamos observando más adelante, vemos que nos pide introducir un primer valor el cual será colocado en “**var_4**” con lo cual podemos



renombrarla como “**valor_a**”; a continuación nos pide un segundo valor que será colocado en la variable “**var_0**”, también la renombramos como “**valor_b**”. Además renombraremos también la función “**main**” como “**funcionChachi**”.

A continuación vemos como se realizan dos llamadas a distintas funciones, la primera a **sub_401328** y la segunda a **sub_40134C**, y también como antes de cada llamada a cada función se la pasan al “**stack**” dos argumentos para que cada función los tome como valores. Realizando todo lo dicho hasta aquí la función “**main**” tendría este aspecto:

```
; int __cdecl funcionChachi(int argc, const char **argv, const char **envp)
funcionChachi proc near

var_C= dword ptr -0Ch
valor_b= dword ptr -8
valor_a= dword ptr -4
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

push    ebp
mov     ebp, esp
sub     esp, 18h
and     esp, 0FFFFFFF0h
mov     eax, 0
add     eax, 0Fh
add     eax, 0Fh
shr     eax, 4
shl     eax, 4
mov     [ebp+var_C], eax
mov     eax, [ebp+var_C]
call    ___chkstk
call    ___main
mov     dword ptr [esp], offset aIngreseElValor ; "Ingrese el valor de a:\n"
call    printf
lea     eax, [ebp+valor_a]
mov     [esp+4], eax
mov     dword ptr [esp], offset aD ; "%d"
call    scanf
mov     dword ptr [esp], offset aIngreseElVal_0 ; "Ingrese el valor de b:\n"
call    printf
lea     eax, [ebp+valor_b]
mov     [esp+4], eax
mov     dword ptr [esp], offset aD ; "%d"
call    scanf
mov     eax, [ebp+valor_b]
mov     [esp+4], eax    ; "push" al stack del segundo arg a sub_401328
mov     eax, [ebp+valor_a]
mov     [esp], eax
call    sub_401328      ; "push" al stack del primer arg. a sub_401328
mov     eax, [ebp+valor_b]
mov     [esp+4], eax    ; "push" al stack del segundo arg. a sub_40134C
mov     eax, [ebp+valor_a]
mov     [esp], eax      ; "push" al stack del primer arg. a sub_40134C
call    sub_40134C
call    _getch
mov     eax, 0
leave
retn
funcionChachi endp
```

Hasta este punto tenemos ya un poco clarificado qué realiza nuestro “**main**”. Recordando nuestra primera vista al binario, nos proporcionaba una suma de valores y también una comparación de dichos valores. Por lo tanto dos funciones distintas no sería ninguna “boludez” pensar que las dos llamadas a funciones en el “**main**” pudieran corresponder a cada una de dichas funciones, veamos.

Llamada a la función sub_401328:

```
var_4= dword ptr -4
arg_0= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 18h
mov     eax, [ebp+arg_4]
add     eax, [ebp+arg_0]
mov     [ebp+var_4], eax
mov     eax, [ebp+var_4]
mov     [esp+4], eax
mov     dword ptr [esp], offset aElValorDeLaSum ; "El valor de la suma es %d:\n\n"
call    printf
leave
retn
sub_401328 endp
```

Como podemos observar, de momento no vamos mal con nuestras conjuras. Podemos ver como esta función lo que realiza es la suma de dos valores que son pasados como argumentos, **valor_a y valor_b**, y que su resultado será colocado en "**var_4**", la cual renombraremos como "**resultado**". Veamos el **stack** de la función:

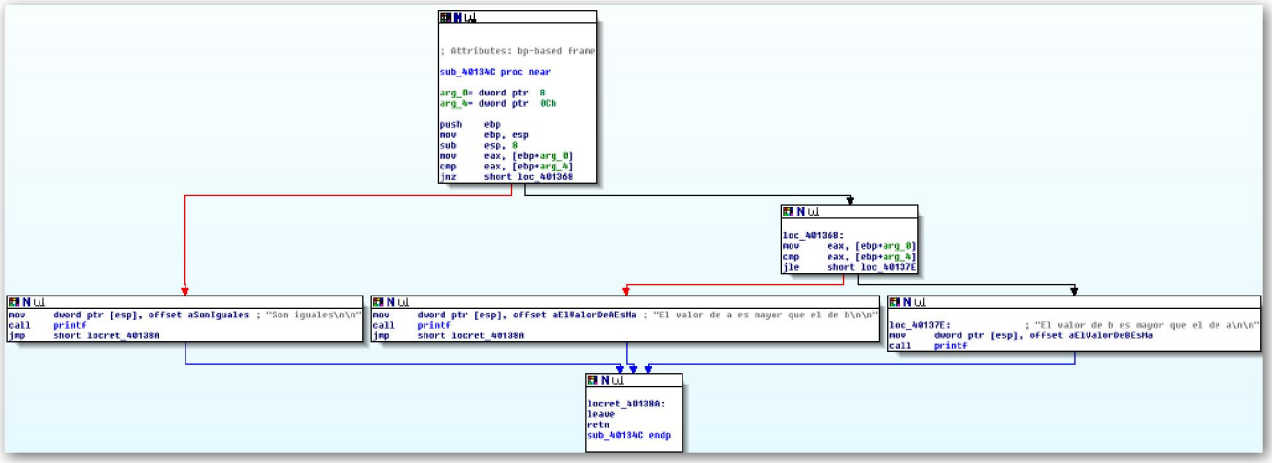
```
-00000004 var_4           dd ?
+00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)
+00000008 arg_0           dd ?
+0000000C arg_4           dd ?
```

Como ya sabemos variables por encima de "**sr**" y argumentos, en este caso dos, por debajo de "**sr**".
Como sabemos que dicha función suma los dos valores introducidos en "**main**" y su resultado se coloca en "**var_4**", renombremoslos e identifiquemos el tipo de la función, también renombraremos a dicha función con el nombre de **funcionSuma**. Una vez realizado nos quedará de dicha forma:

```
; int __cdecl funcionSuma(int valor_b, int valor_a)
funcionSuma proc near

resultado= dword ptr -4
valor_b= dword ptr  8
valor_a= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 18h
mov     eax, [ebp+valor_a]
add     eax, [ebp+valor_b]
mov     [ebp+resultado], eax
mov     eax, [ebp+resultado]
mov     [esp+4], eax
mov     dword ptr [esp], offset aElValorDeLaSum ; "El valor de la suma es %d:\n\n"
call    printf
leave
retn
funcionSuma endp
```

Una vez clarificada la primera llamada a una función vayamos a por la siguiente.

Llamada a sub_4013AC:



En un principio habíamos supuesto que era de comparación y a la vista está de que es así, debido a las distintas desviaciones posibles de flujo de ejecución, causadas en este caso por instrucciones del tipo "**if**" y del tipo "**else**". Vayamos módulo por módulo:

```
; Attributes: bp-based frame

sub_40134C proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 8
mov     eax, [ebp+arg_0]
cmp     eax, [ebp+arg_4]
jnz     short loc_401368
```

En esta ocasión vemos que dicha función solo utilizará para su cometido los dos argumentos pasados a la función desde "**main**", que no son más que **valor_a** y **valor_b**. Renombremos los dos argumentos con sus nombres. Al final del primer módulo se realiza una comparación la cual dice; Si "**if**" comparación igual a cero desvía a

```
mov     dword ptr [esp], offset aSonIguales ; "Son iguales\n\n"
call    printf
jmp     short locret_40138A
```

quiere decir que **valor_a == valor_b**.
Sino "**else**"desvía a

```
loc_401368:
mov     eax, [ebp+valor_a]
cmp     eax, [ebp+valor_b]
jle     short loc_40137E
```

Al llegar a este módulo se realiza otra comparación del mismo tipo que la anterior la cual desviará el flujo de ejecución de la siguiente forma.
Si "**if**" **valor_a > que valor_b** desviará hacia

```
mov     dword ptr [esp], offset aElValorDeAEsMa ; "El valor de a es mayor que el de b\n\n"
call    printf
jmp     short locret_40138A
```

Sino "**else**" **valor_b > que valor_a**

```
loc_40137E:             ; "El valor de b es mayor que el de a\n\n"
mov     dword ptr [esp], offset aElValorDeBEsMa
call    printf
```

Para finalizar veamos como queda nuestra función, la cual hemos renombrado y tipificado como **funcionCompara:**

```
; int __cdecl funcionCompara(int valor_a, int valor_b)
funcionCompara proc near

valor_a= dword ptr  8
valor_b= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 8
mov     eax, [ebp+valor_a]
cmp     eax, [ebp+valor_b]
jnz     short loc_401368
```

Las tres desviaciones posibles de la función terminan todas con este módulo final el cual finaliza la función en cuestión.

```
locret_40138A:
leave
retn
funcionCompara endp
```

Ahora llega el final del estudio lo cual es pasar toda nuestra información a código fuente para que con ello podamos realizar las mil modificaciones que queramos a dicho binario.

Codigo fuente de ejemploRic

```
#include <stdio.h>

funcionSuma (valor_a, valor_b )
{
        int resultado;

        resultado = valor_a + valor_b;
        printf ("El valor de la suma es %d:\n\n", resultado);

        getchar ( );
}

funcionCompara (valor_a, valor_b)
{

          if (valor_a == valor_b)
          {
           printf ("Son iguales\n\n");
          }
          else
          {
             if (valor_a <= valor_b)
             printf ("El valor de b es mayor que el de a\n\n");
             else
```

```
            printf ("El valor de a es mayor que el de b\n\n");
        }

        getchar ( );
}

main ( )
{
    int valor_a;
    int valor_b;

    printf ("Ingrese el valor de a:\n");
    scanf ("%d", &valor_a);

    printf ("Ingrese el valor de b:\n");
    scanf ("%d", &valor_b);

    funcionSuma (valor_a, valor_b );
    funcionCompara (valor_a, valor_b );

    getchar ( );
}
```

Saludos
Bigundill@