C Y REVERSING (parte 4) por Ricnar

Seguiremos adelante con un ejemplo de arrays, también llamado tabla, vector, arreglo o matriz, el cual es un conjunto de elementos en el cual todos son del mismo tipo, por ejemplo:

int ejemplo[4];

Define un **array** de cuatro elementos en los cuales cada uno es un **int**, si se necesita acceder a uno de los elementos se hará mediante un subindice entre corchetes comenzando con el valor cero para el primer elemento y así sucesivamente, por ejemplo los elementos del array ejemplo son **ejemplo[0], ejemplo[1], ejemplo[2], ejemplo[3].**

```
# include <stdio.h>

main(){
 funcion();
 getchar();
}

funcion(){


   int numero[5];         /* Un array de 5 números enteros */
   int suma;             /* Un entero que será la suma */

   numero[0] = 200;      /* Les damos valores */
   numero[1] = 150;
   numero[2] = 100;
   numero[3] = -50;
   numero[4] = 300;
   suma = numero[0] +    /* Y hallamos la suma */
      numero[1] + numero[2] + numero[3] + numero[4];
   printf("Su suma es %d", suma);

         }
```

Vemos que declara un array de 5 números enteros y a continuación lo inicializa con sus correspondientes valores, luego halla la suma de todos y lo imprime, veamos como se ve en el IDA todo esto.

```
.text:004012C6
.text:004012C6 funcion_investigada proc near           ; CODE XREF: _main+2A↑p
.text:004012C6
.text:004012C6 var_2C          = dword ptr -2Ch
.text:004012C6 var_28          = dword ptr -28h
.text:004012C6 var_24          = dword ptr -24h
.text:004012C6 var_20          = dword ptr -20h
.text:004012C6 var_1C          = dword ptr -1Ch
.text:004012C6 var_18          = dword ptr -18h
.text:004012C6
.text:004012C6                 push    ebp
.text:004012C7                 mov     ebp, esp
.text:004012C9                 sub     esp, 48h
.text:004012CC                 mov     [ebp+var_28], 0C8h
.text:004012D3                 mov     [ebp+var_24], 96h
.text:004012DA                 mov     [ebp+var_20], 64h
.text:004012E1                 mov     [ebp+var_1C], 0FFFFFFCEh
.text:004012E8                 mov     [ebp+var_18], 12Ch
```

Tenemos allí las variables que vamos a tratar de mostrar como **array**, es de mencionar que un array como tiene todos los tipos de variable iguales, puede ser perfectamente declarado si uno reversea este código, como cinco variables **int** independientes ya que el IDA normalmente las trata así, y para la funcionalidad, es similar escribir el código como lo hemos hecho antes o escribirlo así. (aunque en este caso como ya veremos no podremos usar un for para iterar por los campos del array y habrá que sumarlos uno a uno)

```
# include <stdio.h>

main(){
 funcion();
 getchar();
}

funcion(){



   int suma;              /* Un entero que será la suma */

   int numero0 = 200;      /* Les damos valores */
   int numero1 = 150;
   int numero2 = 100;
   int numero3 = -50;
   int numero4 = 300;
   suma = numero0 +    /* Y hallamos la suma */
      numero1 + numero2 + numero3 + numero4;
   printf("Su suma es %d", suma);

         }
```

Vemos si lo abrimos en IDA.

```
sub_4012C6 proc near

var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 28h
mov     [ebp+var_8], 0C8h
mov     [ebp+var_C], 96h
mov     [ebp+var_10], 64h
mov     [ebp+var_14], 0FFFFFFCEh
mov     [ebp+var_18], 12Ch
mov     eax, [ebp+var_C]
add     eax, [ebp+var_8]
add     eax, [ebp+var_10]
add     eax, [ebp+var_14]
add     eax, [ebp+var_18]
```

No hay ninguna diferencia así que se podría interpretar de ambas formas, volvamos al ejemplo original con el array.

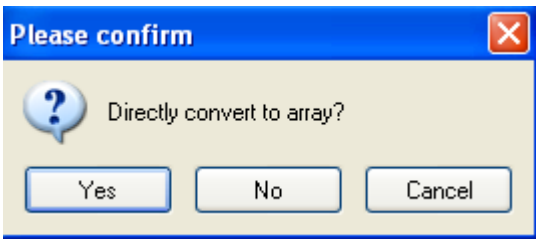
Volvamos al código original, veamoslo en el IDA.

```
.text:004012C6
.text:004012C6 var_2C          = dword ptr -2Ch
.text:004012C6 var_28          = dword ptr -28h
.text:004012C6 var_24          = dword ptr -24h
.text:004012C6 var_20          = dword ptr -20h
.text:004012C6 var_1C          = dword ptr -1Ch
.text:004012C6 var_18          = dword ptr -18h
.text:004012C6
.text:004012C6                 push    ebp
.text:004012C7                 mov     ebp, esp
.text:004012C9                 sub     esp, 48h
.text:004012CC                 mov     [ebp+var_28], 0C8h
.text:004012D3                 mov     [ebp+var_24], 96h
.text:004012DA                 mov     [ebp+var_20], 64h
.text:004012E1                 mov     [ebp+var_1C], 0FFFFFFCEh
.text:004012E8                 mov     [ebp+var_18], 12Ch
.text:004012EF                 mov     eax, [ebp+var_24]
.text:004012F2                 add     eax, [ebp+var_28]
.text:004012F5                 add     eax, [ebp+var_20]
.text:004012F8                 add     eax, [ebp+var_1C]
.text:004012FB                 add     eax, [ebp+var_18]
.text:004012FE                 mov     [ebp+var_2C], eax
```

Renombraremos la variable **var_2c** como **suma** ya que vemos que guarda el resultado de la misma y no es parte del array que esta pintado en verde, luego vamos a la tabla de variables, haciendo doble click en cualquiera de ellas.

```
-0000002E                db ? ; undefined
-0000002D                db ? ; undefined
-0000002C suma           dd ?
-00000028 var_28         dd ?
-00000024 var_24         dd ?
-00000020 var_20         dd ?
-0000001C var_1C         dd ?
-00000018 var_18         dd ?
-00000014                db ? ; undefined
-00000013                db ? ; undefined
-00000012                db ? ; undefined
-00000011                db ? ; undefined
```

Ahora marcamos la **var_28** que es la primera variable del array y apretamos asterisco, vemos que en la ventana nos sale que el largo de cada elemento del array que lo toma de la variable actual donde hicimos doble click o sea **var_28** es de 4 bytes, y el máximo largo posible sin pisar las siguientes variables es 1 ya que abajo esta la **var_24** pero como nosotros queremos que nuestro array abarque 5 dwords aunque pise las variables siguientes, en tamaño o ARRAY SIZE ponemos 5 y damos OK y luego YES.



Vemos que quedo así

```
-0000002C suma           dd ?
-00000028 var_28         dd 5 dup(?)
-00000014                db ? ; undefined
-00000013                db ? ; undefined
-00000012                db ? ; undefined
-00000011                db ? ; undefined
-00000010                db ? ; undefined
-0000000F                db ? ; undefined
-0000000E                db ? ; undefined
```

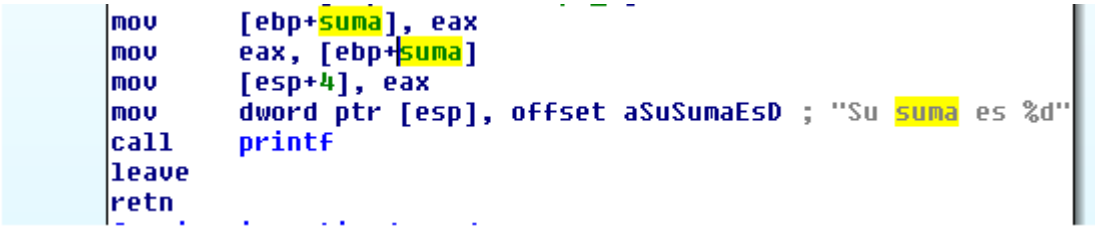
Ahora si vemos en el listado.

```
sub_4012C6 proc near

suma= dword ptr -2Ch
var_28= dword ptr -28h

push    ebp
mov     ebp, esp
sub     esp, 48h
mov     [ebp+var_28], 0C8h
mov     [ebp+var_28+4], 96h
mov     [ebp+var_28+8], 64h
mov     [ebp+var_28+0Ch], 0FFFFFFCEh
mov     [ebp+var_28+10h], 12Ch
mov     eax, [ebp+var_28+4]
add     eax, [ebp+var_28]
add     eax, [ebp+var_28+8]
add     eax, [ebp+var_28+0Ch]
add     eax, [ebp+var_28+10h]
```

Vemos que hay una sola variable **var_28** que la renombraremos a **numero** como en el código original y es un array, los campos subsiguientes del array se ven como **ebp+numero**, luego **ebp+numero+4**, y así sucesivamente.

```
funcion_investigada proc near

suma= dword ptr -2Ch
numero= dword ptr -28h

push    ebp
mov     ebp, esp
sub     esp, 48h
mov     [ebp+numero], 0C8h
mov     [ebp+numero+4], 96h
mov     [ebp+numero+8], 64h
mov     [ebp+numero+0Ch], 0FFFFFFCEh
mov     [ebp+numero+10h], 12Ch
mov     eax, [ebp+numero+4]
add     eax, [ebp+numero]
add     eax, [ebp+numero+8]
add     eax, [ebp+numero+0Ch]
add     eax, [ebp+numero+10h]
mov     [ebp+suma], eax
mov     eax, [ebp+suma]
mov     [esp+4], eax
mov     dword ptr [esp], offset aSuSumaEsD ; "Su suma es %d"
call    printf
leave
```

Bueno no hay mucho mas que decir de este código, ya tenemos nuestro array, con su nombre correcto, y sus campos creados en forma correcta, vemos que los suma y los guarda en la variable **suma** la cual imprime mediante **printf** usando **%d** para mostrar su valor numérico.



Los arrays se pueden inicializar al mismo tiempo que se declaran como cualquier variable el código en ese caso seria así y el código en el IDA es similar al visto anteriormente.

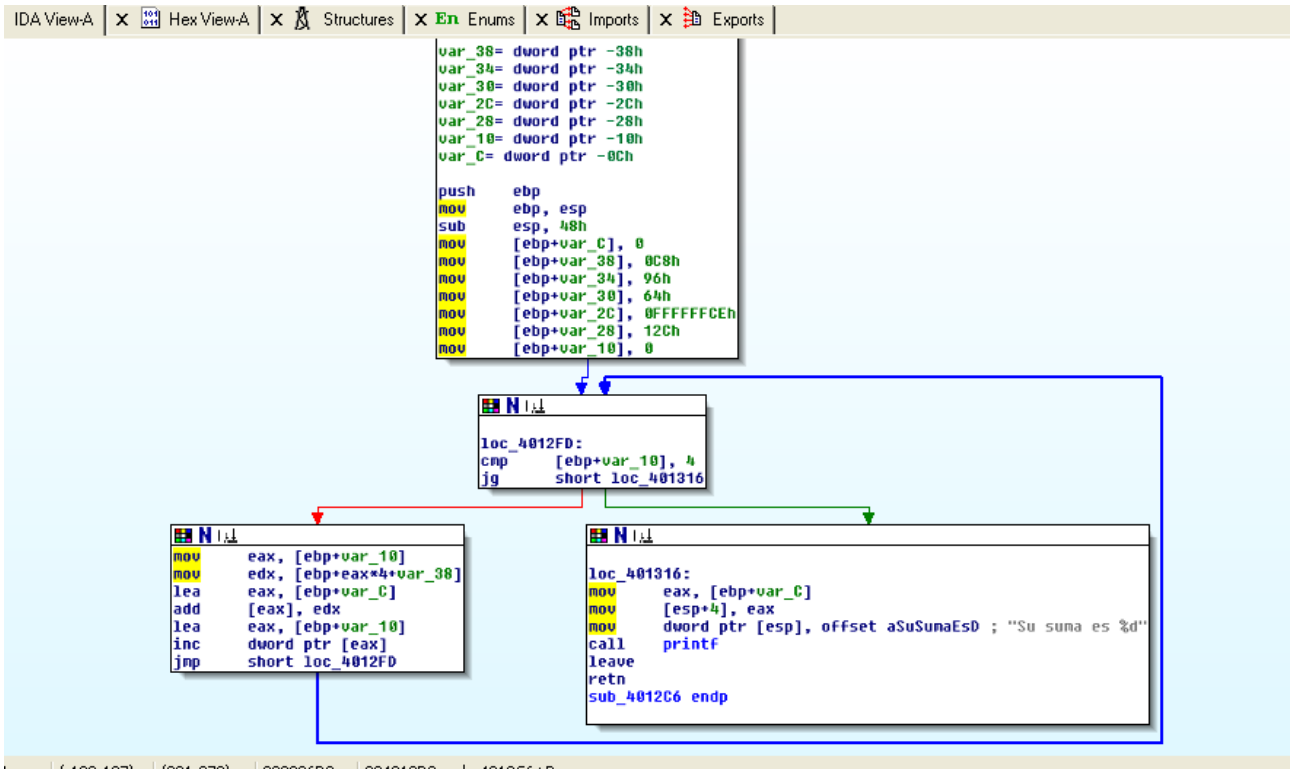
```
# include <stdio.h>

main(){
 funcion();
 getchar();
}

funcion(){



   int suma;              /* Un entero que será la suma */

   int numero[5] =          /* Un array de 5 números enteros */
     {200, 150, 100, -50, 300};
   suma = numero0 +    /* Y hallamos la suma */
      numero1 + numero2 + numero3 + numero4;
   printf("Su suma es %d", suma);

         }
```

Si utilizamos un for para sumar los miembros del array es mucho mas conveniente sobre todo si es un array muy largo.

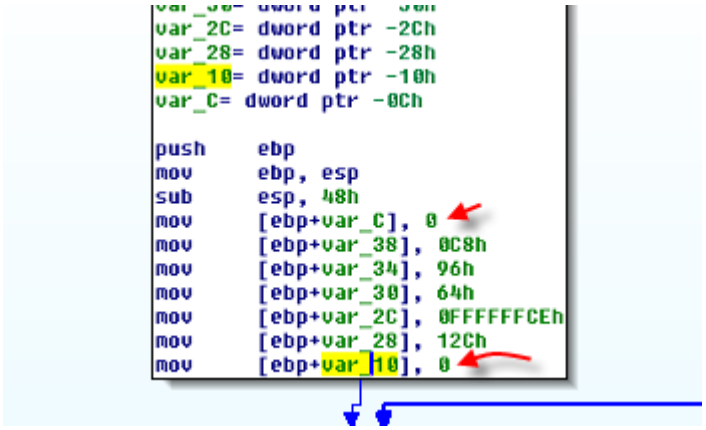
```
# include <stdio.h>

main(){
 funcion();
 getchar();
}

funcion(){

   int suma=0;              /* Un entero que será la suma */
   int i;

   int numero[5] ={200, 150, 100, -50, 300};
```

```
        for(i=0;i<=4;i++) suma += numero[i];
    printf("Su suma es %d", suma);
        }
```

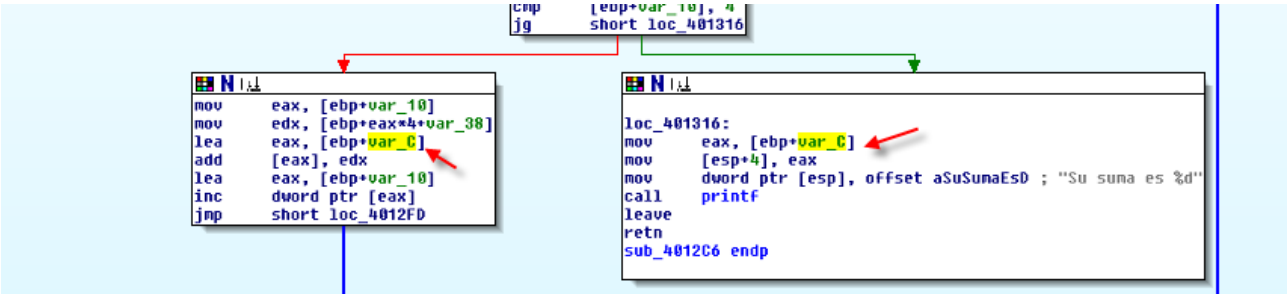
Vemos que tenemos que inicializar en este caso la variable **suma** con 0, pues si no dará error dentro del for cuando quiera sumar la primera vez y no tenga valor, también debemos declarar la variable **i** como **int** que sera el contador en el for tomando los valores de 0 a 4 para usarse como subindice de **numero[i]** y sumar todos los miembros, luego al terminar el for imprimirá la suma, veamos este ejemplo en el IDA.



Bueno iremos renombrando las variables vemos que hay dos variable que se inicializan a cero, la **var_C** y **var_10**.

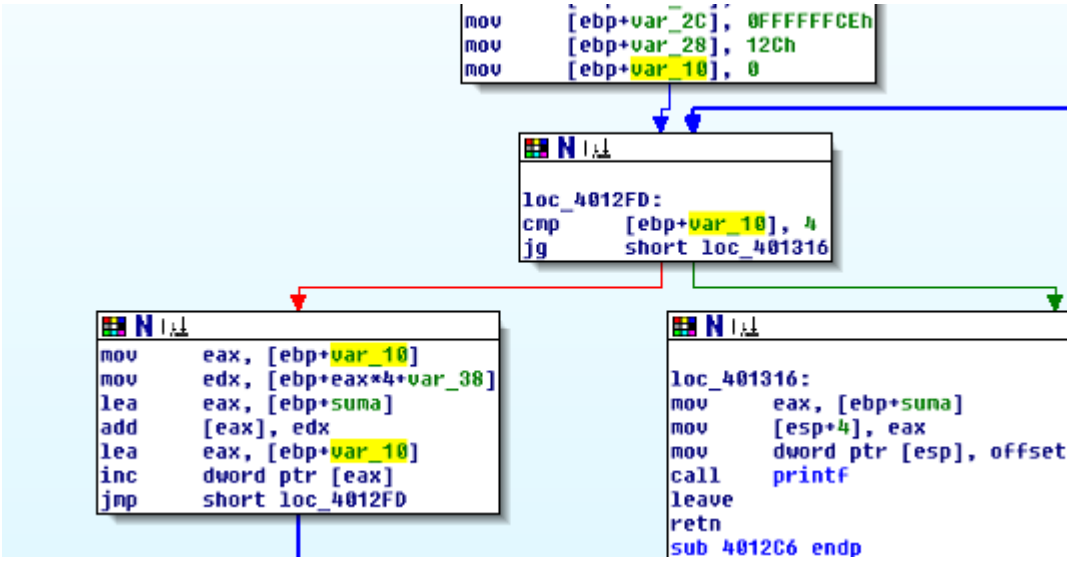


Una rápida inspección de lo que hace **var_C** podemos marcarla y ver donde se usa o apretar X.



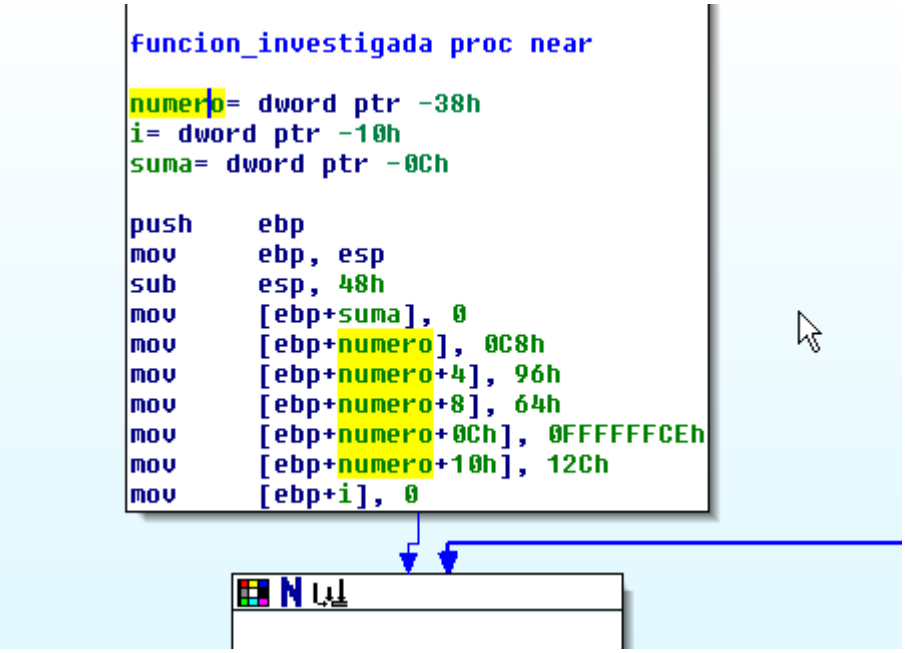
Vemos que mediante un **lea** dentro del loop obtiene la dirección de la misma y luego le va sumando cosas con un **add**, ya veremos que, pero podemos colegir que es la variable **suma** ya que al salir del loop se utiliza para imprimir su valor mediante **printf** con el mensaje **La suma es....**

La renombramos como **suma**.

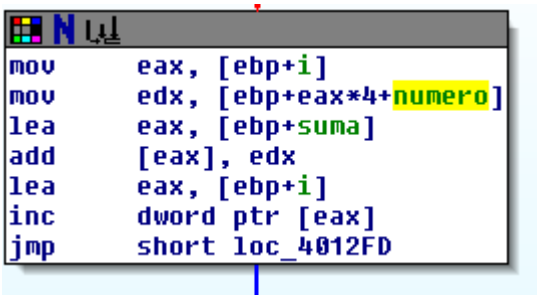


La variable **var_10** es el contador del loop la llamamos **i** como en el código fuente, podríamos haberla llamado contador, se ve cuando se inicializa a cero y que el máximo valor sera cuatro, y cuando sea mas grande que 4, saldrá del loop a imprimir.

Definimos el array, realizamos los pasos como en el ejemplo anterior y quedara así.



Ahora si vemos el array con sus campos, veamos que hace dentro del loop que es lo que nos faltaba.



Vemos que EAX tomara el valor de la variable **i** que empieza en 0 y termina en 4 ya que es el contador del loop.

mov edx, [ebp+eax*4+numero]

En la primera vez que loopea como EAX vale 0 se tendrá en EDX

mov edx, [ebp + numero]

Cuando EAX valga 1

mov edx, [ebp+1*4+numero]

que es igual a

mov edx, [ebp + numero + 4]

y que siempre termina siendo EDX el campo actual ya que salta de cuatro en cuatro

Así que EDX tendrá los valores de los campos en dada interación y a continuación los suma aquí obteniendo la dirección de la variable **suma** y sumándole EDX a su contenido, en cada loop se le sumara el campo siguiente, al salir del loop la variable **suma** tendrá la sumatoria de todos los campos y se imprimirá como en el ejemplo original.

```
lea     eax, [ebp+suma]
add     [eax], edx
```

CADENAS DE CARACTERES

Las cadenas de texto se crean como arrays de caracteres, lo cual veremos como manejar en el IDA.

Las misma están formadas por una sucesión de caracteres **terminada con un carácter nulo (\0)**, de modo que tendremos que reservar una letra más de las que necesitamos. Por ejemplo, para guardar el texto “Hola” usaríamos “char saludo[5]”.

```
# include <stdio.h>

main(){
 funcion();
 getchar();
 getchar();
}

funcion(){


   char texto[40];         /* Para guardar hasta 39 letras */

   printf("Introduce tu nombre: ");
   scanf("%s", &texto);
   printf("Hola, %s\n", texto);


        }
```

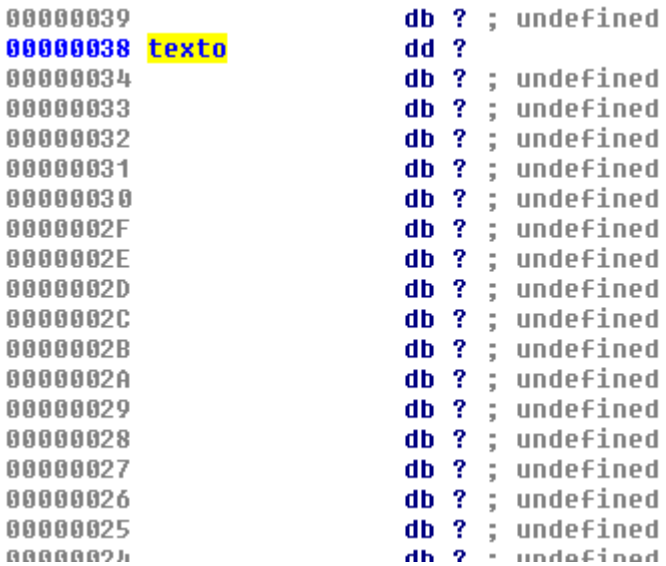
Allí vemos que creamos un array sin inicializar que puede guardar hasta 39 caracteres ya que debe poner el cero final, luego mediante **printf** se imprime el mensaje **"Introduce tu nombre: ");** y luego a scanf se le pasa la dirección de la variable texto usando el **&** para que llene la misma, luego se imprime lo que el usuario tipeo en la variable texto precedido de un Hola.
Lindo código para un stack overflow jeje, así que compilemos y veamos el código en IDA.

```
sub_4012CB proc near

var_38= dword ptr -38h

push    ebp
mov     ebp, esp
sub     esp, 48h
mov     dword ptr [esp], offset aIntroduce
call    printf
lea     eax, [ebp+var_38]
mov     [esp+4], eax
mov     dword ptr [esp], offset aS ; "%s"
call    scanf
lea     eax, [ebp+var_38]
mov     [esp+4], eax
mov     dword ptr [esp], offset aHolaS ; "H
call    printf
leave
retn
sub_4012CB endp
```

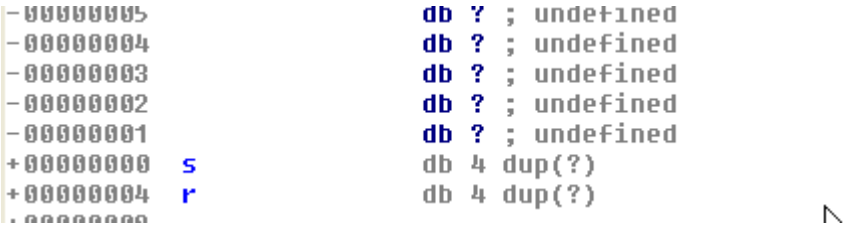
Vemos que hay una sola variable que es nuestro array llamado **texto** lo renombramos.

```
; Attributes: bp-based frame

funcion_investigada proc near

texto= dword ptr -38h

push    ebp
mov     ebp, esp
sub     esp, 48h
mov     dword ptr [esp], offset aIntroduceTuNom ; "Introduce tu
call    printf
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aS ; "%s"
call    scanf
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aHolaS ; "Hola, %s\n"
call    printf
leave
retn
funcion_investigada endp
```

Si vemos las variables haciendo doble click en texto.



Vemos que mas abajo están el stored ebp y el return address.



No hay que ser un genio para darse cuenta de que si el usuario tipea demasiados caracteres, como no hay ningún chequeo ni nada, terminara pisando ambos y al continuar la ejecución del programa y llegar el **retn** y al salir de mi funcion habré redirigido el programa a la dirección que halla quedado allí en el return address pisado por mi, según con que caracteres lo haya hecho, pero bueno que esto no es un curso de exploits, aunque es bueno remarcarlo je.

En el caso de los arrays volveremos a usar el comando de IDA llamado ARRAY , apretando asterisco cobre la variable texto.

Allí IDA se da cuenta que podría crear un ARRAY de hasta 56 bytes antes de pisar el stored ebp y el return address, como nosotros sabemos que nuestro array es menor elegimos 40 de largo igual el procesador reserva un poco mas de lugar del necesario, si eligiéramos 56 funcionaria de la misma forma.

```
-00000039                db ? ; undefined
-00000038 texto          db 40 dup(?)
-00000010                db ? ; undefined
-0000000F                db ? ; undefined
-0000000E                db ? ; undefined
-0000000D                db ? ; undefined
-0000000C                db ? ; undefined
-0000000B                db ? ; undefined
-0000000A                db ? ; undefined
-00000009                db ? ; undefined
-00000008                db ? ; undefined
-00000007                db ? ; undefined
-00000006                db ? ; undefined
-00000005                db ? ; undefined
-00000004                db ? ; undefined
-00000003                db ? ; undefined
-00000002                db ? ; undefined
-00000001                db ? ; undefined
+00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)
+00000008
```

De esta forma quedo el código así, no se aprecian camios pues no existe el usa de caracteres intermedios del array, si fuera ese el caso si se accediera a un carácter especifico por posición usando los corchetes como subindices, el código habría cambiado, lo cual veremos en el siguiente ejemplo.

```
; Attributes: bp-based frame

funcion_investigada proc near

texto= byte ptr -38h

push    ebp
mov     ebp, esp
sub     esp, 48h
mov     dword ptr [esp], offset aIntroduceTuNom ; "Introduce tu nombre:
call    printf
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aS ; "%s"
call    scanf
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aHolaS ; "Hola, %s\n"
call    printf
leave
retn
funcion_investigada endp
```

El código es el siguiente:

```
# include <stdio.h>

main(){
 funcion();
 getchar();
  getchar();
}
```

funcion(){

char texto[40];

printf("Introduce tu nombre: ");
scanf("%s", texto);
printf("Hola, %s. Tu inicial es %c\n", texto, texto[0]);
}

Vemos que luego de crear el array de 40 caracteres inclusive el cero final, luego hace un printf usando un doble format string, primero con **%s** y el texto tipeado completo como string, y luego **%c** con solo el primer carácter que obtiene usando el subindice cero (**texto[0]**)

Si vemos en el IDA el código y renombramos convenientemente.

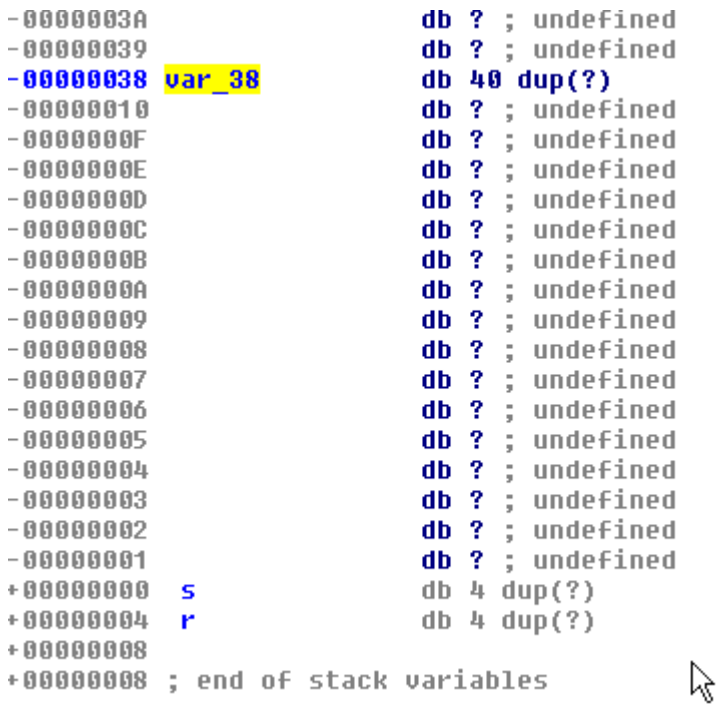
```
funcion_investigada proc near

texto= byte ptr -38h

push    ebp
mov     ebp, esp
sub     esp, 48h
mov     dword ptr [esp], offset aIntroduceTuNom ; "Introduce tu nombre: "
call    printf
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aS ; "%s"
call    scanf
movsx   eax, [ebp+texto]
mov     [esp+8], eax
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aHolaS_TuInicia ; "Hola, %s. Tu inicial es %c\n"
call    printf
leave
retn
funcion_investigada endp
```

Vemos que le pasa la dirección de la variable texto mediante el LEA para qiue el usuario lo llene usando **scanf.**

```
call    scanf
movsx   eax, [ebp+texto]
mov     [esp+8], eax
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aHolaS_TuInicia ; "Hola, %s. Tu inicial es %c\n"
call    printf
leave
```

Vemos que con movsx mueve el primer byte de lo tipeado y lo pasa como argumento al stack para el format string **%c**, luego pasa la dirección a la string completa con lea, para hacer el otro format string **%s**.

Definiendo el array.

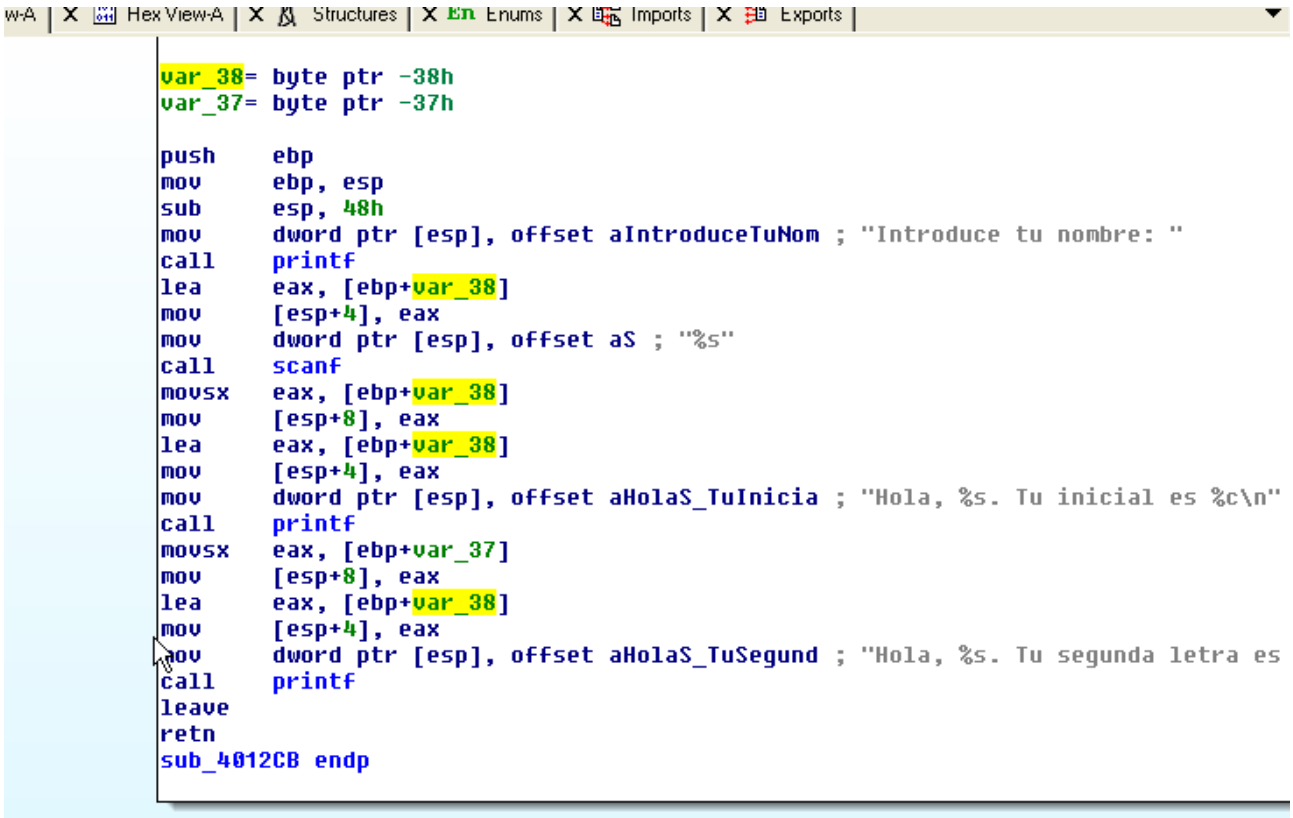


Al ver el código vemos que no hubo variación ya que usa solo el primer carácter si cambiamos el código para que use el segundo.

```
# include <stdio.h>

main(){
 funcion();
 getchar();
  getchar();
}

funcion(){
```

```
char texto[40];

printf("Introduce tu nombre: ");
scanf("%s", texto);
printf("Hola, %s. Tu inicial es %c\n", texto, texto[0]);
printf("Hola, %s. Tu segunda letra es %c\n", texto, texto[1]);
    }
```

Vemos en el IDA que ahora tuvo que crear una variable para el segundo carácter, así que creo dos variables de un byte lo cual no es lo mas aproximado al código nuestro ya que nosotros usamos los dos primeros caracteres de un array y no dos variables char sueltas.



Si marcamos la variable superior y definimos el array de 40 caracteres de largo.

```
-0000003A                db ? ; undefined
-00000039                db ? ; undefined
-00000038 var_38         db 40 dup(?)
-00000010                db ? ; undefined
-0000000F                db ? ; undefined
-0000000E                db ? ; undefined
-0000000D                db ? ; undefined
-0000000C                db ? ; undefined
-0000000B                db ? ; undefined
-0000000A                db ? ; undefined
-00000009                db ? ; undefined
-00000008                db ? ; undefined
-00000007                db ? ; undefined
-00000006                db ? ; undefined
-00000005                db ? ; undefined
-00000004                db ? ; undefined
-00000003                db ? ; undefined
-00000002                db ? ; undefined
-00000001                db ? ; undefined
+00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)
+00000008
+00000008 ; end of stack variables
```

Vemos que ahora si tenemos una sola variable array y si vemos el código, este cambio.

```
; Attributes: bp-based frame

sub_4012CB proc near

texto= byte ptr -38h

push    ebp
mov     ebp, esp
sub     esp, 48h
mov     dword ptr [esp], offset aIntroduceTuNom ; "Intro
call    printf
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aS ; "%s"
call    scanf
movsx   eax, [ebp+texto]
mov     [esp+8], eax
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aHolaS_TuInicia ; "Hola,
call    printf
movsx   eax, [ebp+texto+1]
mov     [esp+8], eax
lea     eax, [ebp+texto]
mov     [esp+4], eax
mov     dword ptr [esp], offset aHolaS_TuSegund ; "Hola,
call    printf
leave
retn
sub_4012CB endp
```

Vemos las dos flechas rojas leen el primer carácter y el segundo carácter, pero esta vez como primer y segundo campo de un array y no como variables separadas el resto es similar al ejemplo anterior.
Bueno en la parte 5 seguimos con mas arrays y estructuras.
Hasta la parte 5
Ricardo Narvaja