

REVERSING WITH IDA PRO FROM SCRATCH

PART 41

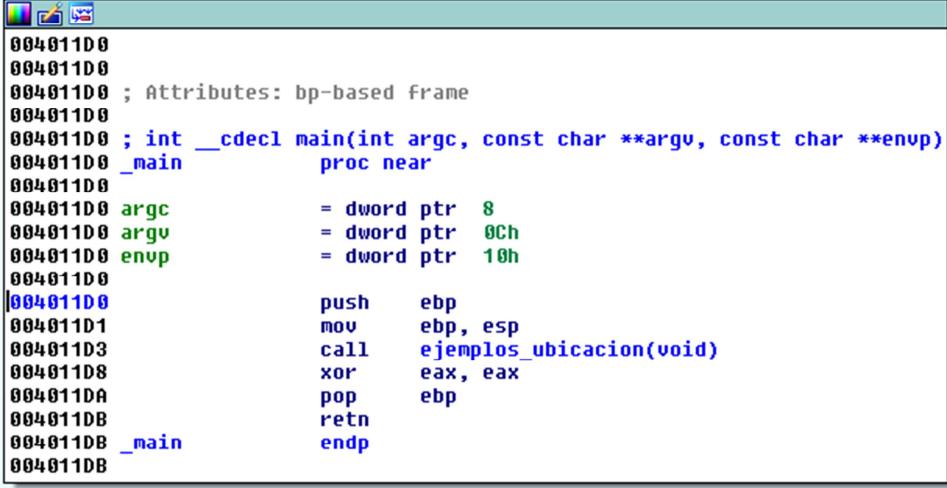
We'll continue practicing and seeing examples, in this case, it is a code that has different ways of handling and locating strings.

```
L 1
2
3  #include "stdafx.h"
4  #include <iostream>
5
6
7  char string[] = "Donde se ubicara?";
8  char string2[20];
9
10
11 void ejemplos_ubicacion()
12 {
13
14     char mensaje_en_stack[]="hola reverser en stack";
15     char mensaje_en_stack_sin_inicializar[100];
16     char *mensaje_en_data="hola reverser en data";
17     char *mensaje_en_heap;
18
19
20     mensaje_en_heap = (char *)malloc(strlen(mensaje_en_data)+1);
21
22     strcpy(mensaje_en_stack_sin_inicializar, "hola reverser en stack sin inicializar");
23
24     strcpy(mensaje_en_heap,mensaje_en_data);
25
26     memcpy(mensaje_en_heap+strlen("hola reverser en "), "heap",4);
27
28     strcpy(string2,"Donde se ubicara?");
29
30
31     printf("direccion mensaje_en_stack = 0x%08x\n", mensaje_en_stack);
32     printf("direccion mensaje_en_stack_sin_inicializar = 0x%08x\n", mensaje_en_stack_sin_inicializar);
33     printf("direccion mensaje_en_data = 0x%08x\n", mensaje_en_data);
34     printf("direccion mensaje_en_heap = 0x%08x\n", mensaje_en_heap);
35     printf("direccion string = 0x%08x\n", string);
36     printf("direccion string2 = 0x%08x\n", string2);
37
38     getchar();
39
40 }
```

There are some characters and then, it prints the addresses of each one to see where it is located.

Here, we aren't looking for some vulnerability. We are just seeing addresses.

Let's open the executable in the LOADER. We make it load the symbols. We'll see the MAIN function.



```
004011D0
004011D0
004011D0 ; Attributes: bp-based frame
004011D0
004011D0 ; int __cdecl main(int argc, const char **argv, const char **envp)
004011D0 _main           proc near
004011D0
004011D0 argc            = dword ptr  8
004011D0 argv            = dword ptr  0Ch
004011D0 envp            = dword ptr  10h
004011D0
004011D0                 push   ebp
004011D0                 mov    ebp, esp
004011D0                 call   ejemplos_ubicacion(void)
004011D0                 xor    eax, eax
004011D0                 pop    ebp
004011D0                 retn
004011D0 _main           endp
004011D0
```

There, we have the MAIN. It only has a CALL to the **ejemplos_ubicacion** function. Nothing else. Let's activate the DEMANGLE NAMES –NAMES.

```
void ejemplos_ubicacion()
{
    char mensaje_en_stack[]="hola reverser en stack";
    char mensaje_en_stack_sin_inicializar[100];
    char *mensaje_en_data="hola reverser en data";
    char *mensaje_en_heap;

    mensaje_en_heap = (char *)malloc(strlen(mensaje_en_data)+1);
    strcpy(mensaje_en_stack_sin_inicializar, "hola reverser en stack sin inicializar");
    strcpy(mensaje_en_heap,mensaje_en_data);
    memcpy(mensaje_en_heap+strlen("hola reverser en "), "heap", 4);
    strcpy(string2,"Donde se ubicara?");

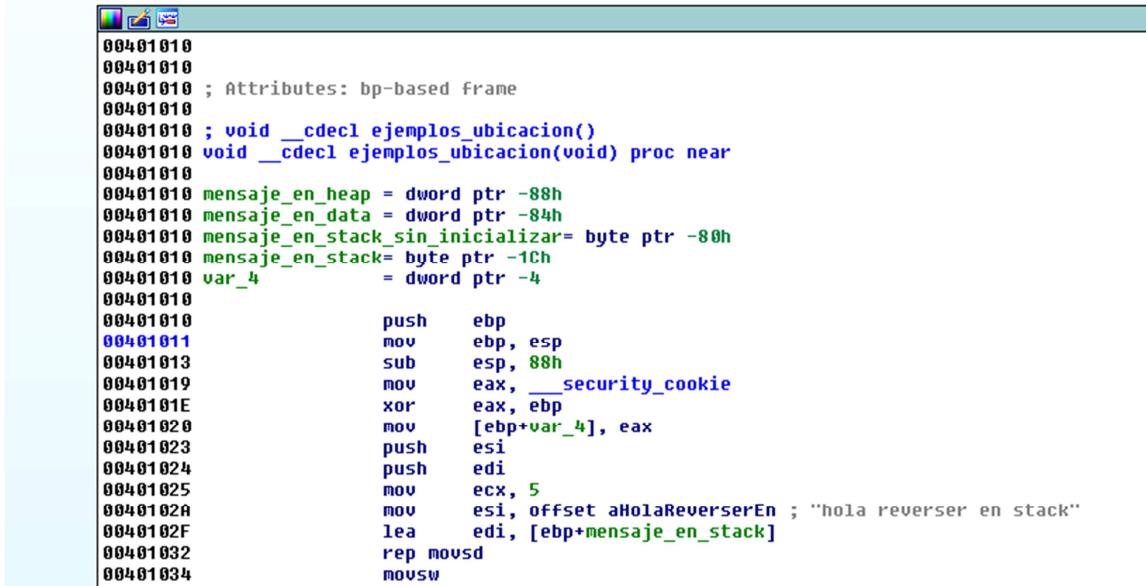
    printf("direccion mensaje_en_stack = 0x%llx\n", mensaje_en_stack);
    printf("direccion mensaje_en_stack_sin_inicializar = 0x%llx\n", mensaje_en_stack_sin_inicializar);
    printf("direccion mensaje_en_data = 0x%llx\n", mensaje_en_data);
    printf("direccion mensaje_en_heap = 0x%llx\n", mensaje_en_heap);
    printf("direccion string = 0x%llx\n", string);
    printf("direccion string2 = 0x%llx\n", string2);

    getch();
}

int _tmain(int argc, _TCHAR* argv[])
{
    ejemplos_ubicacion();
    return 0;
}
```

There, it is. Let's see what happens inside the function. We see it doesn't have args, only variables.

```
void __cdecl ejemplos_ubicacion()
```



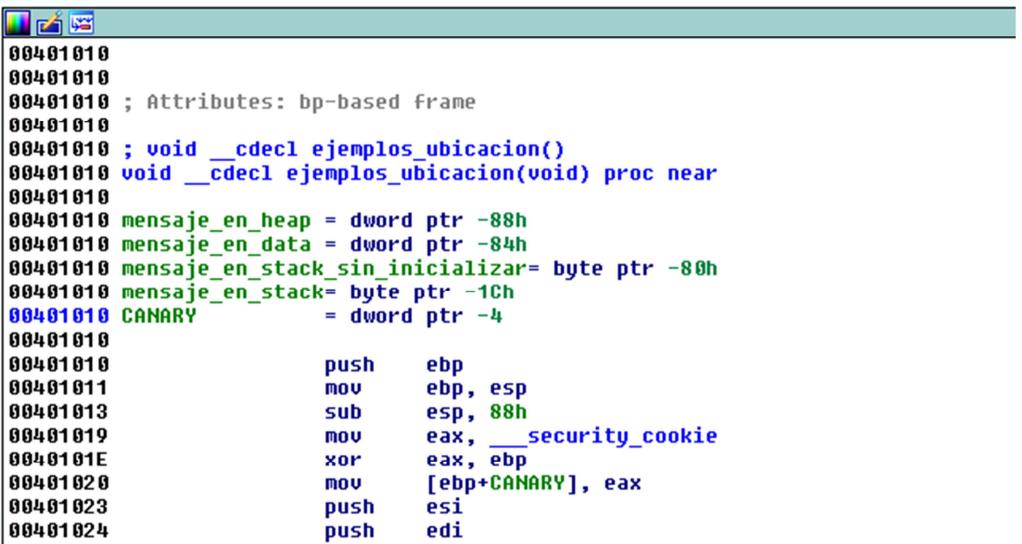
```
00401010
00401010 ; Attributes: bp-based frame
00401010
00401010 ; void __cdecl ejemplos_ubicacion()
00401010 void __cdecl ejemplos_ubicacion(void) proc near
00401010
00401010     mensaje_en_heap = dword ptr -88h
00401010     mensaje_en_data = dword ptr -84h
00401010     mensaje_en_stack_sin_inicializar= byte ptr -80h
00401010     mensaje_en_stack= byte ptr -1ch
00401010     var_4           = dword ptr -4
00401010
00401010             push    ebp
00401011             mov     ebp, esp
00401013             sub     esp, 88h
00401019             mov     eax, __security_cookie
0040101E             xor     eax, ebp
00401020             mov     [ebp+var_4], eax
00401023             push    esi
00401024             push    edi
00401025             mov     ecx, 5
0040102A             mov     esi, offset aHolaReverserEn ; "hola reverser en stack"
0040102F             lea     edi, [ebp+mensaje_en_stack]
00401032             rep    movsd
00401034             movsw
```

If it had args, they would be in the parenthesis. Besides, seeing the stack representation...

```
-0000008F          db ? ; undefined
-0000008E          db ? ; undefined
-0000008D          db ? ; undefined
-0000008C          db ? ; undefined
-0000008B          db ? ; undefined
-0000008A          db ? ; undefined
-00000089          db ? ; undefined
-00000088  mensaje_en_heap dd ? ; offset
-00000084  mensaje_en_data dd ? ; offset
-00000080  mensaje_en_stack_sin_inicializar db 100 dup(?)
-0000001C  mensaje_en_stack db 23 dup(?)
-00000005          db ? ; undefined
-00000004  var_4      dd ?
+00000000  s          db 4 dup(?)
+00000004  r          db 4 dup(?)
+00000008
+00000008 ; end of stack variables
```

If it had args, they should be below the Return Address (r) but there is nothing, only variables.

There, I see the CANARY saved in var_4 to start reversing.



```

00401010
00401010
00401010 ; Attributes: bp-based frame
00401010
00401010 ; void __cdecl ejemplos_ubicacion()
00401010 void __cdecl ejemplos_ubicacion(void) proc near
00401010
00401010 mensaje_en_heap = dword ptr -88h
00401010 mensaje_en_data = dword ptr -84h
00401010 mensaje_en_stack_sin_inicializar= byte ptr -80h
00401010 mensaje_en_stack= byte ptr -1Ch
00401010 CANARY           = dword ptr -4
00401010
00401010             push    ebp
00401011             mov     ebp, esp
00401013             sub     esp, 88h
00401019             mov     eax, __security_cookie
0040101E             xor     eax, ebp
00401020             mov     [ebp+CANARY], eax
00401023             push    esi
00401024             push    edi

```

Let's go back to the stack representation.

```

-0000008B          db ? ; undefined
-0000008A          db ? ; undefined
-00000089          db ? ; undefined
-00000088 mensaje_en_heap dd ? ; offset
-00000084 mensaje_en_data dd ? ; offset
-00000080 mensaje_en_stack_sin_inicializar db 100 dup(?)
-0000001C mensaje_en_stack db 23 dup(?)           
-00000005             db ? ; undefined
-00000004 CANARY      dd ?
+00000000 s          db 4 dup(?)
+00000004 r          db 4 dup(?)
+00000008
+00000008 ; end of stack variables

```

We see that as **mensaje_en_stack** as **mensaje_en_stack_sin_inicializar** are stack buffers. One of 100 bytes and the other of 23 bytes.

There, we see the two cases.

- 1) **char mensaje_en_stack[]="hola reverser en stack"; #initialized**
- 2) **char mensaje_en_stack_sin_inicializar[100]; #not initialized**

In case 2, it reserves 100 bytes because it doesn't know what will be saved there. It could be something entered by the user and not constant. The other reserves the space for the “**hola reverser en stack**” string that has a determined length.

```
Output window
Python>len("hola reverser en stack")
22
Python
```

It is 22 bytes long + the terminating null. 23 totally.

There, we see when it copies the string in the stack and initializes the variable. First, it gets the string address with OFFSET and moves it to ESI. Then, it will copy it to EDI that has the buffer address in the stack.

```
00401024      push    edi
00401025      mov     ecx, 5
0040102A      mov     esi, offset aHolaReverserEn ; "hola reverser en stack"
0040102F      lea     edi, [ebp+mensaje_en_stack]
00401032      rep     movsd
00401034      movsw
00401034      movch
```

rdata

Declares an initialized data section that is readable but not writable. Microsoft compilers use this section to place constants in it.

In the **rdata** section, Visual Studio, when compiling, saves the constant data and the string is saved there. As the section is not writable, it won't change. Then, it copies it to the stack.

```
.rdata:00402104      align 10h
.rdata:00402110 aHolaReverserEn db 'hola reverser en stack',0 ; DATA XREF: ejemplos_ubicacion(void)+1Afo
.rdata:00402110
.rdata:00402127      align 4
.rdata:00402128      aHolaReverser_0 db 'hola reverser en data',0 ; DATA XREF: ejemplos_ubicacion(void)+27fo
.rdata:00402128
.rdata:0040213E      align 10h
.rdata:00402140 ; char aHolaReverser_1[]
.rdata:00402140 aHolaReverser_1 db 'hola reverser en stack sin inicializar'.0
```

The LEA moves the buffer address in the stack and copies the string there with reps movs, initializing the variable.

In case 1, the variable was initialized while in case 2, it wasn't.

Logically, in case 2, this non-initialized buffer is there for something and the program will use and fill it some time.

```
00401063      push    offset aHolaReverser_1 ; "hola reverser en stack sin inicializar"
00401068      lea     ecx, [ebp+mensaje_en_stack_sin_inicializar]
0040106B      push    ecx ; char *
0040106C      call    _strcpy
```

It will do it there, it looks similar to case 1, but now, the program uses a Windows API to copy, it takes the OFFSET of the “**hola reverser sin inicializar**” string from the **rdata** section and copies it with strcpy.

strcpy(mensaje_en_stack_sin_inicializar, "**hola reverser en stack sin inicializar**");

Obviously, the compiler as in the stack variable initialization as in case 1, it won’t use Windows APIs. It will use instructions like reps movs while in case 2, it is program code that can use APIs.

It is a coincidence that this case 2 has a determined string of a constant length, but it could be a string entered by the user that its length can change. There, we’ll have to check that it doesn’t overflow the buffer with a longer string.

Obviously, from these two strings we’ll get their addresses with LEA and they’ll be printed.

```
00401007      add    esp, 8
0040100B      lea     ecx, [ebp+mensaje_en_stack]
0040100F      push   ecx
00401013      push   offset _Format ; "direccion mensaje_en_stack = 0x%x\n"
00401017      call   _printf
0040101B      add    esp, 8
0040101F      lea     edx, [ebp+mensaje_en_stack_sin_inicializar]
00401023      push   edx
00401027      push   offset aDireccionMen_0 ; "direccion mensaje_en_stack_sin_iniciali...
0040102B      call   _printf
0040102F      add    esp, 8
```

```
004010DE      add    esp, 8
004010E1      mov    eax, [ebp+mensaje_en_data]
004010E7      push   eax
004010E8      push   offset aDireccionMen_1 ; "direccion mensaje_en_data = 0x%x\n"
004010ED      call   _printf
004010F2      add    esp, 8
004010F5      mov    ecx, [ebp+mensaje_en_heap]
004010FB      push   ecx
004010FC      push   offset aDireccionMen_2 ; "direccion mensaje_en_heap = 0x%x\n"
00401101      call   _printf
00401106      add    esp, 8
00401109      push   offset char * string ; "Donde se ubicara?"
0040110E      push   offset abireccionStrin ; "direccion string = 0x%x\n"
00401113      call   _printf
00401118      add    esp, 8
0040111B      push   offset char * string2
00401120      push   offset aDireccionStr_0 ; "direccion string2 = 0x%x\n"
00401125      call   _printf
0040112A      add    esp, 8
0040112D      call   ds: _imp_getchar
00401133      pop    edi
00401134      pop    esi
```

If I set a BP there, I choose the local debugger and run the program in debugger mode.

The screenshot shows a debugger interface with two main windows. The top window displays assembly code for a C++ program. The bottom window shows a memory dump (Stack view) with several memory blocks. A red arrow points to one of the memory blocks in the dump window, highlighting the address 0010F800.

```

012710C2 push    ecx
012710C3 push    offset _Format ; "direccion mensaje_en_stack = %d\n"
012710C8 call    _printf
012710CD add     esp, 8
012710D0 lea    edx, [ebp+mensaje_en_stack_sin_inicializar]
012710D3 push    edx
012710D4 push    offset abireccionMen_0 ; "direccion mensaje_en_st
012710D9 call    _printf
012710E0 add     esp, 8
012710E5 mov    eax, [ebp+mensaje_en_data]
012710E7 push    eax
012710E8 push    offset abireccionMen_1 ; "direccion mensaje_en_da
012710ED call    _printf
012710F2 add     esp, 8
012710F5 mov    ecx, [ebp+mensaje_en_heap]
012710FB push    ecx
012710FC push    offset abireccionMen_2 ; "direccion mensaje_en_he
0127110B call    _printf
0127110C add     esp, 8
0127110E push    offset char * string ; "Donde se ubicara?"
0127110F push    offset abireccionStrin ; "direccion string = 0x2A
01271110 add     esp, 8
01271118 push    offset char * string2
01271120 push    offset abireccionStr_0 ; "direccion string2 = 0x2A
01271125 call    _printf
0127112A add     esp, 8
0127112B call    ds: _IO_getchar
01271133 pop    edi
01271134 pop    esi
01271135 mov    ecx, [ebp+CANARY]
01271138 xor    ecx, ebp ; cookie
0127113E call    __security_check_cookie(x)
0127113F mov    esp, ebp
01271141 pop    ebp
01271142 retn

12D: ejemplos_ubicacion(void)+11D (Synchronized with EIP)

08 F0 21 27 01 E8 9E 00  .Ia|---Phf*:Px.
FF FF FF F1 68 41 22 27  .N---X-x---0h..h8
08 68 00 30 27 01 68 38  .N---X-x---0h..h8
33 Cb 08 68 38 30 27 01  ...bx...bx...h0...
00 00 83 C4 08 FF 15 88  hT***.bf...3...@.
33 CD E8 90 00 00 88  .*.^IM3-b...Y
00 00 00 00 00 00 00 00  .0000000000000000

```

Address	Value	Content
0010F800	000772E4	ucrtbase.dll.00A772E4
0010F800	400772E0	ucrtbase.dll.00A772E0
0010F800	000304F0	debug018:004304F0
0010F800	01272128	.rdata:aholaReverser_0
0010F800	61606F68	
0010F800	76657228	
0010F800	65737765	

The printed addresses belong to the stack. I can see the strings in those addresses.

```

Stack[000023E0]:001BF907 db 72h ; u
Stack[000023E0]:001BF90A db 9Dh ; 0
Stack[000023E0]:001BF90B db 6Bh ; k
EIP Stack[000023E0]:001BF90C db 68h ; h (highlighted)
Stack[000023E0]:001BF90D db 6Fh ; 0
Stack[000023E0]:001BF90E db 6Ch ; l
Stack[000023E0]:001BF90F db 61h ; a
Stack[000023E0]:001BF910 db 20h
Stack[000023E0]:001BF911 db 72h ; r
Stack[000023E0]:001BF912 db 65h ; e
Stack[000023E0]:001BF913 db 76h ; v
Stack[000023E0]:001BF914 db 65h ; e
Stack[000023E0]:001BF915 db 72h ; r
Stack[000023E0]:001BF916 db 73h ; s
Stack[000023E0]:001BF917 db 65h ; e
Stack[000023E0]:001BF918 db 72h ; r
Stack[000023E0]:001BF919 db 20h
Stack[000023E0]:001BF91A db 65h ; e
Stack[000023E0]:001BF91B db 6Eh ; n
Stack[000023E0]:001BF91C db 20h
Stack[000023E0]:001BF91D db 73h ; s
Stack[000023E0]:001BF91E db 74h ; t
Stack[000023E0]:001BF91F db 61h ; a
Stack[000023E0]:001BF920 db 63h ; c
Stack[000023E0]:001BF921 db 6Bh ; k
Stack[000023E0]:001BF922 db 0
Stack[000023E0]:001BF923 db 0
Stack[000023E0]:001BF924 db 63h ; c
Stack[000023E0]:001BF925 db 0F9h ; "
Stack[000023E0]:001BF926 db 0EAh ; Ü
Stack[000023E0]:001BF927 db 77h ; w
Stack[000023E0]:001BF928 db 30h ; 0
Stack[000023E0]:001BF929 db 0F0h ; "

```

If I press A to convert it in ASCII string...

```

Stack[000023E0]:001BF909 db 75h ; u
Stack[000023E0]:001BF90A db 9Dh ; 0
Stack[000023E0]:001BF90B db 6Bh ; k
Stack[000023E0]:001BF90C aHolaReverser_3 db 'hola reverser en stack',0
Stack[000023E0]:001BF923 db 0
Stack[000023E0]:001BF924 db 63h ; c
Stack[000023E0]:001BF925 db 0F9h ; "
Stack[000023E0]:001BF926 db 0EAh ; Ü
Stack[000023E0]:001BF927 db 77h ; w
Stack[000023E0]:001BF928 db 30h ; 0
Stack[000023E0]:001BF929 db 0F0h ; "

```

And the other one.

```

IDA View-EIP
d:\drive\curso c++\clase 3\punteros_y_referencias\punteros
Stack[000023E0]:001BF8A4 db 28h ; (
Stack[000023E0]:001BF8A5 db 21h ; !
Stack[000023E0]:001BF8A6 db 27h ; .
Stack[000023E0]:001BF8A7 db 1
Stack[000023E0]:001BF8A8 aHolaReverser_4 db 'hola reverser en stack sin inicializar',0
Stack[000023E0]:001BF8CF db 0D7h ; Í
Stack[000023E0]:001BF8D0 db 1
Stack[000023E0]:001BF8D1 db 0
Stack[000023E0]:001BF8D2 db 0
Stack[000023E0]:001BF8D3 db 0
Stack[000023E0]:001BF8D4 db 0

```

We are OK until now. Let's see the other strings. Let's stop the debugger and go back to the loader.

```
00000000 ; 
00000000 ; 
00000000 db ? ; undefined
0000000F db ? ; undefined
0000000E db ? ; undefined
0000000D db ? ; undefined
0000000C db ? ; undefined
0000000B db ? ; undefined
0000000A db ? ; undefined
00000009 db ? ; undefined
00000008 mensaje_en_heap dd ?
00000004 mensaje_en_data dd ? ; offset
00000000 mensaje_en_stack_sin_inicializar db 100 dup(?) ; offset
0000001C mensaje_en_stack db 23 dup(?)
00000005 db ? ; undefined
00000004 CANARY dd ?
00000000 s db 4 dup(?)
00000004 r db 4 dup(?)
00000008 ; end of stack variables
```

The other two variables are pointers (offset) and just need 4 bytes each one (dd)

Let's see the **mensaje_en_data** pointer first.

```
01271034 movsw
01271036 movsb
01271037 mov [ebp+mensaje_en_data], offset aHolaReverser_0 ; "hola reverser en data"
01271041 mov eax, [ebp+mensaje_en_data]
01271047 push eax ; char *
01271048 call _strlen
0127104D add esp, 4
01271050 add eax, 1
01271053 push eax ; Size
```

In this case, the variable is a pointer that saves the address it gets with OFFSET in the **mensaje_en_data** stack variable

The string is located in **rdata** and saved there. What we handle in the stack is the pointer to it while in the previous ones the full string was copied in the stack filling a buffer in itself.

```
012710DE add esp, 8
012710E1 mov eax, [ebp+mensaje_en_data]
012710E7 push eax
012710E8 push offset aDireccionMen_1 ; "direccion mensaje_en_data = 0x%x\n"
012710ED call _printf
```

Now, it doesn't need a LEA to find the address because the variable value is a pointer and it is the address that is moved to EAX and pushed to print its value. Let's see if we debug it as before.

```

013810C2 push    ecx
013810C3 push    offset _printf ; "direccion mensaje_en_stack = 0x%0x\n"
013810C8 call    _printf
013810CD add     esp, 8
013810D0 lea     edx, [ebp+mensaje_en_stack_sin_inicializar]
013810D3 push    edx
013810D4 push    offset abireccionMen_0 ; "direccion mensaje_en_stack_sin_inicializar = 0x16f970"
013810D9 call    _printf
013810DE add     esp, 8
013810E1 mov     eax, [ebp+mensaje_en_data]
013810E7 push    eax
013810E8 push    offset abireccionMen_1 ; "direccion mensaje_en_data = 0x%0x\n"
013810ED call    _printf
013810F2 add     esp, 8
013810F5 mov     eax, [ebp+mensaje_en_heap]
013810FB push    eax
013810FC push    offset abireccionMen_2 ; "direccion mensaje_en_heap = 0x%0x\n"
013810B1 call    _printf
013810B6 add     esp, 8
013810B9 push    offset char * string ; "Donde se ubicara?"
013810BE push    offset abireccionStrin ; "direccion string = 0x%0x\n"
01381113 call    _printf
01381118 add     esp, 8
0138111B push    offset char * string2
01381120 push    offset abireccionStr_0 ; "direccion string2 = 0x%0x\n"
01381125 call    _printf
0138112A add     esp, 8
0138112D call    ds: _imp_getchar
01381133 pop    edi

```

If I go to that rdata address, the string will be there.

```

.rdata:01382118 align 4 ; DATA XREF: ejemplos_ubicacion(void)+1Afo
.rdata:01382127 align 4 ; DATA XREF: ejemplos_ubicacion(void)+1Afo
.rdata:01382128 aHolaReverser_0 db 'hola reverser en data',0 ; DATA XREF: Stack[00003A24]:0016F908fo
.rdata:01382128 ; DATA XREF: Stack[00003A24]:0016F908fo
.rdata:01382128 ; ejemplos_ubicacion(void)+27fo
.rdata:0138213E align 10h ; DATA XREF: ejemplos_ubicacion(void)+53fo
.rdata:01382140 ; char aHolaReverser_1[]
.rdata:01382140 aHolaReverser_1 db 'hola reverser en stack sin inicializar',0 ; DATA XREF: ejemplos_ubicacion(void)+53fo
.rdata:01382140

```

And, in the stack variable, that address (OFFSET) is saved.

```

Stack[00003A24]:0016F904 db 0F0h ; 
Stack[00003A24]:0016F905 db 0D4h ; E
Stack[00003A24]:0016F906 db 41h ; A
Stack[00003A24]:0016F907 db 0
Stack[00003A24]:0016F908 dd offset aHolaReverser_0 ; "hola reverser en data"
Stack[00003A24]:0016F90C db 68h ; h
Stack[00003A24]:0016F90D db 6Fh ; o
Stack[00003A24]:0016F90E db 6Ch ; l
Stack[00003A24]:0016F90F db 61h ; a
Stack[00003A24]:0016F910 db 20h
Stack[00003A24]:0016F911 db 72h ; r

```

There, the OFFSET is. If I want to see the numeric value, I press D.

```

Stack[00003A24]:0016F907 db 0
Stack[00003A24]:0016F908 db 28h
Stack[00003A24]:0016F909 db 21h
Stack[00003A24]:0016F90A db 38h
Stack[00003A24]:0016F90B db 1

```

If I press D some times when it becomes a DWORD, it detects that it points to the string and changes to its OFFSET.

IDA View-EIP

```

Stack[00003A24]:0016F904 db 0F0h ; 
Stack[00003A24]:0016F905 db 0D4h ; E
Stack[00003A24]:0016F906 db 41h ; A
Stack[00003A24]:0016F907 db 0
Stack[00003A24]:0016F908 dd offset aHolaReverser_0 ; "hola reverser en data"
Stack[00003A24]:0016F90C db 68h ; h
Stack[00003A24]:0016F90D db 6Fh ; o
Stack[00003A24]:0016F90E db 6Ch ; l
Stack[00003A24]:0016F90F db 61h ; a
Stack[00003A24]:0016F910 db 20h
013810D4 push offset aDireccionMen_0 ; "direccion mensaje_en_stack_sin_iniciali...
013810D9 call _printf
013810DE add esp, 8
013810E1 mov eax, [ebp+mensaje_en_data]
013810E7 push eax
013810E8 push offset aDireccionMen_1 ; "direccion mensaje_en_data_=8u%u\n"
013810ED call _printf
013810F2 add esp, 8
013810F5 mov ecx, [ebp+mensaje_en_heap]

```

Hovering the mouse on there, we see that variable has the string address (OFFSET) saved because it is a pointer variable that saves addresses.

Let's leave the pending stack variable for the end and see the global variables. For those who don't know, the global variables are located in the source code outside the all functions, generally above all.

```

1
2
3 #include "stdafx.h"
4 #include <iostream>
5
6
7 char string[] = "Donde se ubicara?";
8 char string2[20];
9
10
11 void ejemplos ubicacion()
12 {
13
14     char mensaje_en_stack[]="hola reverser en stack";
15     char mensaje_en_stack_sin_inicializar[100];
16     char *mensaje_en_data="hola reverser en data";
17     char *mensaje_en_heap;
18
19
20     mensaie en head = (char *)malloc(strlen(mensaie en data)+1);

```

First, let's see the string variable that is initialized and it is global.

If I see the code, I realize that string is used here.

```
01381101    call  _printf
01381106    add   esp, 8
01381109    push  offset char * string ; "Donde se ubicara?"
0138110E    push  offset abDireccionString ; "direccion string = 0x%x\n"
01381113    call  _printf
01381118    add   esp, 8
```

It pushes the string offset.

We see something confusing. We are in the data section that is used for the global variables.

```
.data:01383000 _data          segment para public 'DATA' use32
.data:01383000                         assume cs:_data
.data:01383000                         org 1383000h
.data:01383000 ;\char string[18]
.data:01383000 char * string db 'Donde se ubicara?',0
.data:01383000                         ; DATA XREF: ejemplos_ubicacion(void)+F9↑o
.data:01383012                         align 4
.data:01383014 ; unsigned int _security_cookie_complement
.data:01383014     security cookie complement dd 44BF19B1h
```

We have the **char string [18]** buffer definition taken from the symbols. It knows that it is a string of that length that will be there. There, it is the “**Donde se ubicara**” saved string. If we press D, we see the bytes. If we press A, we go back as it was before.

```
; Segment permissions: Read/Write
.data:01383000 ; Segment permissions: Read/Write
.data:01383000 _data          segment para public 'DATA' use32
.data:01383000                         assume cs:_data
.data:01383000                         org 1383000h
.data:01383000 ;\char string[18]
.data:01383000 char * string db 44h           ; DATA XREF: ejemplos_ubicacion(void)+F9↑o
.data:01383001 db 6Fh ; o
.data:01383002 db 6Eh ; n
.data:01383003 db 64h ; d
.data:01383004 db 65h ; e
.data:01383005 db 20h
.data:01383006 db 73h ; s
.data:01383007 db 65h ; e
.data:01383008 db 20h
.data:01383009 db 75h ; u
.data:0138300A db 62h ; b
.data:0138300B db 69h ; i
.data:0138300C db 63h ; c
.data:0138300D db 61h ; a
.data:0138300E db 72h ; r
.data:0138300F db 61h ; a
.data:01383010 db 3Fh ; ?
.data:01383011 db 0
.data:01383012                         align 4
.data:01383012 ; unsigned int _security_cookie_complement
```

Normally, when there is a second definition is because there is some reference to some API. From that API, it knows what kind of variable it needs and places it as definition too.

```

.data:01383000 ; segment permissions: Read/Write
.data:01383000 _data          segment para public 'DATA' use32
.data:01383000         assume cs:_data
.data:01383000 ;org 1383000h
.data:01383000 ; char string[18]
.data:01383000 char * string  db 'Donde se ubicara?',0
.data:01383000         ; DATA XREF: ejemplos ubicacion(void)+F9 to l
.data:01383012 align 4
.data:01383012         ; unsigned int security_cookie_complement

```

We see that there is a reference there.

01381115	cdll	_printf
01381118	add	esp, 8
0138111B	push	offset char * string2
01381120	push	offset aDireccionStr_0 ; "direccion string2 = 0x%x\n"
01381125	call	_printf
0138112A	add	esp, 8
0138112D	call	ds: imp_getchar

It is an arg to printf whose second arg is an address. If we right click, we'll see that it is the address the string is.

013810FB	mov	ecx, _strupvar_00J
013810FC	push	ecx
01381101	push	offset aDireccionMen_2 ; "direccion mensaje_en_heap = 0x%
01381106	call	_printf
01381109	push	1383000h
0138110E	push	offset aDireccionString ; "direccion string = 0x%x\n"
01381113	call	_printf
01381118	add	esp, 8
0138111B	push	1383038h
01381120	push	offset aDireccionStr_0 ; "direccion string2 = 0x%x\n"
01381125	call	_printf
0138112A	add	esp, 8
0138112D	call	ds: imp_getchar
01381133	pop	edi
01381134	pop	esi

There, when right clicking, I directly replaced the offsets to the strings by the address it will print. In both, they are in the data section.

```

data:01383000 ; char string2[20]           ; DATA XREF: ejemplos ubicacion(void)+A2 to l
data:01383000 char * string2 dw 0           ; DATA XREF: ejemplos ubicacion(void)+A2 to l
data:0138303A db 0
data:0138303B db 0
data:0138303C db 0
data:0138303D db 0
data:0138303E db 0
data:0138303F db 0
data:01383040 db 0
data:01383041 db 0
data:01383042 db 0
data:01383043 db 0
data:01383044 db 0
data:01383045 db 0
data:01383046 db 0
data:01383047 db 0
data:01383048 db 0
data:01383049 db 0
data:0138304A db 0
data:0138304B db 0
data:0138304C align 10h
data:01383050 ; unsigned __int64 __local_stdio_printf_options'::`2'::__OptionsStorage
data:01383050 unsigned __int64 __local_stdio_printf_options'::`2'::__OptionsStorage dq 0

```

The only difference is that String2 is not initialized so with the A of ASCII, it won't revert to the original state because it is full of 0s, and it is empty. So, right click- ARRAY will work.

```
.data:01383030 ; const int __scrt_ucrt_dll_is_in_use ; __isa_available_init+2C1w ...
.data:01383034 __scrt_ucrt_dll_is_in_use dd 1 ; DATA XREF: __scrt_is_ucrt_dll_in_use+
.data:01383038 ; char string2[20]
.data:01383038 char * string2 dw 0Ah dup(0) ; DATA XREF: ejemplos ubicacion(void)+A2
.data:01383040 align 10h
.data:01383050 ; unsigned __int64 __local_stdio_printf_options'::'2'::_OptionsStorage
.data:01383050 unsigned __int64 __local_stdio_printf_options'::'2'::_OptionsStorage dq 0
.data:01383050 ; DATA XREF: __local_stdio_printf_optio
.data:01383058 ; _EXCEPTION_RECORD GS_ExceptionRecord
.data:01383058 _ExceptionRecord exception_record cs = DATA XREF: __local_stdio_printf_optio
```

char * string2 dw 0Ah dup(0)

dup(0)

It means that the 0 is repeated dw(2 bytes) 0xA times or 20 decimal.

```
.data:01383034 __scrt_ucrt_dll_is_in_use dd 1 ; DATA XREF
.data:01383038 ; char string2[20]
.data:01383038 char * string2 db 14h dup(0) ; DATA XREF
.data:01383040 align 10h
.data:01383050 ; unsigned __int64 __local_stdio_printf_options'::
.data:01383050 unsigned __int64 __local_stdio_printf_options'::'2
.data:01383050 ; DATA XREF
```

If I change it to bytes, it is more direct. It will be 0x14 or 20 byte full of 0s.

Let's see the differences where it is filled.

The screenshot shows a debugger interface with assembly code on the left and a cross-reference dialog on the right. The assembly code includes calls to `_printf` and `ejemplos ubicacion`. The cross-reference dialog lists two entries under "xrefs to char * string2": one pointing up to `ejemplos ubicacion` and another pointing down to the same function. The dialog has buttons for OK, Cancel, Search, and Help.

```
01381109 push offset char * string ; "Donde se ubicara?"
0138110E push offset aDireccionStrin ; "direccion string = 0x%x\n"
01381113 call _printf
01381118 add esp, 8
0138111B push offset char * string2
01381120 push offset aDireccionStr_0 ; "direccion string2 = 0x%x\n"
01381125 call _printf
0138112A add esp, 8
0138112D call ejemplos ubicacion
01381133 pop
01381134 pop
01381135 mov
01381138 xor
0138113A cal
0138113F mov
01381141 pop
01381142 ret
01381142 void __cdecl ejemplos ubicacion()
```

xrefs to char * string2

Direction	Type	Address	Text
Up	o	ejemplos ubicacion(void)+A2	push offset char * string2; char *
Down	o	ejemplos ubicacion(void)+...	push offset char * string2

Line 1 of 2

I converted the OFFSET that is passed as an arg to the original representation, with right click choosing **offset * char** and then with X, I see the references. The first one is where the buffer will be filled.

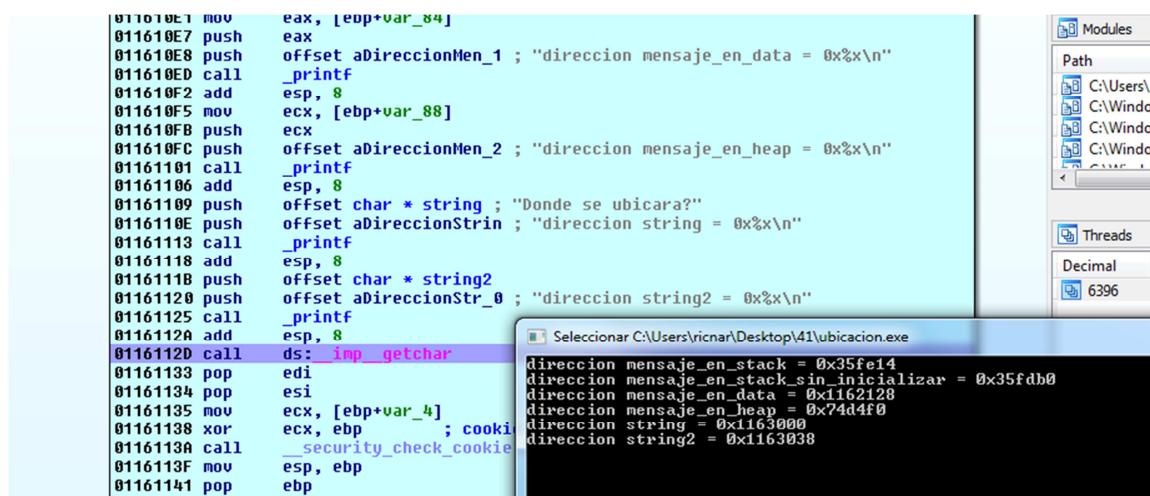
```

013810H4      push    eax ; void *
013810A5      call    _memcpy
013810AA      add     esp, 0Ch
013810AD      push    offset aDondeSeUbicara ; "Donde se ubicara?"*
013810B2      push    offset char * string2 ; char *
013810B7      call    _strcpy
..
```

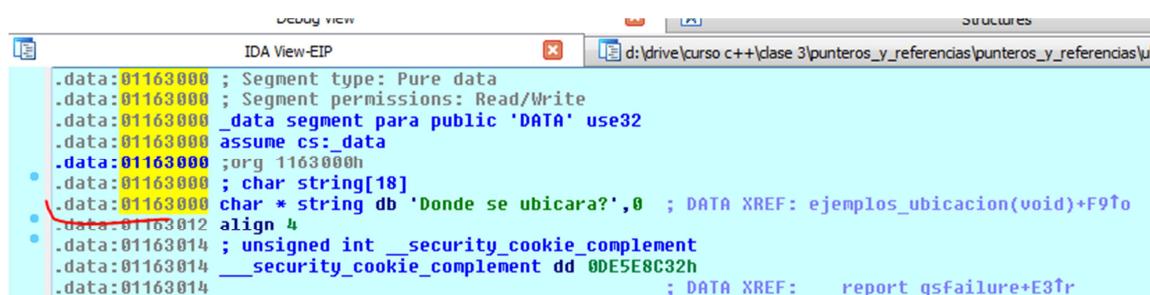
The “**Donde se ubicara?**” string that comes from **rdata** is copied there to the buffer in **data** that is writable and can be modified.

Obviously, in the data section, as it is a writable section that can have buffers, there can be overflows too, if the string length is not well calculated, stepping the global variables there affecting the program.

If we try it, we'll see when debugging...



The string and string2 addresses that correspond to the **data** section.



```

.data:01163030 __isa_enabled dd 7          ; DATA XREF: __isa_available_init+111w
.data:01163030
.data:01163034 ; const int __scrt_ucrt_dll_is_in_use      ; DATA XREF: __isa_available_init+2C1w ...
.data:01163034 __scrt_ucrt_dll_is_in_use dd 1    ; DATA XREF: __scrt_is_ucrt_dll_in_use+21r
.data:01163038 ; char string2[20]                ; DATA XREF: ejemplos_ubicacion(void)+A21o
.data:01163038 char * string2 db 44h, 6Fh, 6Eh, 64h, 65h, 20h, 73h, 65h, 20h, 75h, 62h, 69h, 63h, 61h
.data:01163038                                         ; DATA XREF: ejemplos_ubicacion(void)+10B1o
.data:01163038 db 72h, 61h, 3Fh, 0, 0, 0

```

We need to see the one in the heap. It was a pointer in the stack that saved the string address located in the heap.

```

00401036     movsb
00401037     mov    [ebp+mensaje_en_data], offset aHolaReverser_0 ; "hola reverser en data"
00401041     mov    eax, [ebp+mensaje_en_data]
00401047     push   eax ; char *
00401048     call   _strlen

```

There, it saves the OFFSET or the “hola reverser en data” string address and then, it passes the address to strlen to get its length.

It add 1 to the result as in the source code.

```

00401047     push   eax ; cnar *
00401048     call   _strlen
0040104D     add    esp, 4
00401050     add    eax, 1
00401053     push   eax ; Size
00401054     call   ds:_imp__malloc

```

```

mensaje_en_heap = (char *)malloc(strlen(mensaje_en_data)+1);
-----+-----+-----+-----+-----+-----+-----+-----+

```

And it passes that size to malloc to reserve, in the heap, a dynamic buffer of the string length + 1.

The returned address will vary, but it will be a reserved zone with read and write permission where it will copy later.

```

00401010 ; Attributes: bp-based frame
00401010
00401010 ; void __cdecl ejemplos_ubicacion()
00401010 void __cdecl ejemplos_ubicacion(void) proc near
00401010
00401010     mensaje_en_heap = dword ptr -88h
00401010     mensaje_en_data = dword ptr -84h
00401010     mensaje_en_stack_sin_inicializar= byte ptr -80h
00401010     mensaje_en_stack= byte ptr -1Ch
00401010     var_4           = dword ptr -4
00401010
00401010             push    ebp
00401011             mov     ebp, esp
00401013             sub     esp, 88h
00401019             mov     eax, __security_cookie
0040101E             xor     eax, ebp
00401020             mov     [ebp+var_4], eax
00401023             push    esi
00401024             push    edi
00401025             mov     ecx, 5
0040102A             mov     esi, offset aHolaReverserEn ; "hi"
0040102F             lea     edi, [ebp+mensaje_en_stack]
00401032             rep    movsd
00401034             movsw
00401036             movsb
00401037             mov     [ebp+mensaje_en_data], offset aHi
00401041             mov     eax, [ebp+mensaje_en_data]
00401047             push    eax ; char *
00401048             call    _strlen
0040104D             add    esp, 4
00401050             add    eax, 1
00401053             push    eax ; Size
00401054             call    ds:_imp__malloc
0040105A             add    esp, 4
0040105D             mov     [ebp+mensaje_en_heap], eax

```

There, it saves that heap address that points to that buffer, in the stack pointer variable called **mensaje_en_heap**.

```

0040106C             call    _strcpy
00401071             add    esp, 8
00401074             mov     edx, [ebp+mensaje_en_data]
0040107A             push    edx ; char *
0040107B             mov     eax, [ebp+mensaje_en_heap]
00401081             push    eax ; char *
00401082             call    _strcpy

```

Then, it copies the string pointed by **mensaje_en_data** to the buffer created in the heap.

```

0040108A             push    4 ; size_t
0040108C             push    offset aHeap ; "heap"
00401091             push    offset aHolaReverser_2 ; "hola reverser en "
00401096             call    _strlen
0040109B             add    esp, 4
0040109E             add    eax, [ebp+mensaje_en_heap]
004010A4             push    eax ; void *
004010A5             call    _memcpy

```

Then, it gets the length of the “**hola reverser en**” string and adds it to the buffer start address in the heap. It points to replace the word **data** for **heap** because it does a 4-byte memcpy passing “**heap**” as **source** and the pointer to the word **data** as **destination** that is in the heap buffer.

In the source code we see this:

```
memcpy(mensaje_en_heap+strlen("hola reverser en "),"heap",4);
```

The **destination** is the **mensaje_en_heap** address + “**hola reverser en**” string length. That will point to the word data which will be stepped with 4 bytes being the source the word “**heap**”.

Let's debug to see if that is true.

```
00401025    push   ecx, 5
0040102A    mov    esi, offset aHolaReverserEn ; "hola reverser en stack"
0040102F    lea    edi, [ebp+mensaje_en_stack]
00401032    rep    movsd
00401034    movsw
00401036    movsb
00401037    mov    [ebp+mensaje_en_data], offset aHolaReverser_0 ; "hola reverser en data"
00401041    mov    eax, [ebp+mensaje_en_data]
00401047    push   eax ; char *
00401048    call   _strlen
0040104D    add    esp, 4
00401050    add    eax, 1
00401053    push   eax ; Size
00401054    call   ds:_imp__malloc
0040105A    add    esp, 4
0040105D    mov    [ebp+mensaje_en_heap], eax
00401063    push   offset aHolaReverser_1 ; "hola reverser en stack sin inicializar"
00401068    lea    ecx, [ebp+mensaje_en_stack_sin_inicializar]
0040106B    push   ecx ; char *
0040106C    call   _strcpy
```

```
001F1013    sub    esp, 88h
001F1019    mov    eax, __security_cookie
001F101E    xor    eax, ebp
001F1020    mov    [ebp+var_4], eax
001F1023    push   esi
001F1024    push   edi
001F1025    mov    ecx, 5
001F102A    mov    esi, offset aHolaReverserEn ; "hola reverser en stack"
001F102F    lea    edi, [ebp+mensaje_en_stack]
001F1032    rep    movsd
001F1034    mousw
001F1036    mousb
001F1037    mov    [ebp+mensaje_en_data], offset aHolaReverser_0 ; "hola reverser en data"
001F1041    mov    eax, [ebp+mensaje_en_data]
001F1047    push   eax ; char *
001F1048    call   _strlen
001F104D    add    esp, 4
001F1050    add    eax, 1
001F1053    push   eax ; Size
001F1054    call   ds:_imp__malloc
```

There, it got the “**hola reverser en data**” string length. It is 15 with the terminated null.

External symbol

```

001F1013 sub esp, 88h
001F1019 mov eax, __security_cookie
001F101E xor eax, ebp
001F1020 mov [ebp+var_4], eax
001F1023 push esi
001F1024 push edi
001F1025 mov ecx, 5
001F102A mov esi, offset aHolaReverserEn ; "hola reverser en stack"
001F102F lea edi, [ebp+mensaje_en_stack]
001F1032 rep movsd
001F1034 movsbw
001F1036 movsb
001F1037 mov [ebp+mensaje_en_data], offset aHolaReverser_0 ; "hola reverser en data"
001F1041 mov eax, [ebp+mensaje_en_data]
001F1047 push eax ; char *
001F1048 call strlen
001F104D add esp, 4
001F1050 add eax, 1
001F1053 push eax ; Size
001F1054 call ds:_imp_malloc
001F105A add esp, 4
001F105D mov [ebp+mensaje_en_heap], eax
001F1063 push offset aHolaReverser_1 ; "hola reverser en stack sin inicializar"
001F1068 lea ecx, [ebp+mensaje_en_stack_sin_inicializar]

```

There, it adds 1 and it passed it to malloc to reserve 0x16 bytes.

```

001F1013 sub esp, 88h
001F1019 mov eax, __security_cookie
001F101E xor eax, ebp
001F1020 mov [ebp+var_4], eax
001F1023 push esi
001F1024 push edi
001F1025 mov ecx, 5
001F102A mov esi, offset aHolaReverserEn ; "hola reverser en stack"
001F102F lea edi, [ebp+mensaje_en_stack]
001F1032 rep movsd
001F1034 movsbw
001F1036 movsb
001F1037 mov [ebp+mensaje_en_data], offset aHolaReverser_0 ; "hola reverser en data"
001F1041 mov eax, [ebp+mensaje_en_data]
001F1047 push eax ; char *
001F1048 call strlen
001F104D add esp, 4
001F1050 add eax, 1
001F1053 push eax ; Size
001F1054 call ds:_imp_malloc
001F105A add esp, 4
001F105D mov [ebp+mensaje_en_heap], eax
001F1063 push offset aHolaReverser_1 ; "hola reverser en stack sin inicializar"
001F1068 lea ecx, [ebp+mensaje_en_stack_sin_inicializar]
001F106B push ecx ; char *

```

There, we have the buffer start in the heap. In my case, it is 0x67d4f0.

```
debug018:0067D4ED db 3Ah ; :  
debug018:0067D4EE db 0 ;:  
debug018:0067D4EF db 1Ah ;:  
debug018:0067D4F0 db 0Dh ;:  
debug018:0067D4F1 db 0F0h ;:  
debug018:0067D4F2 db 0ADh ;:  
debug018:0067D4F3 db 0BAh ;:  
debug018:0067D4F4 db 0Dh ;:  
P debug018:0067D4F5 db 0F0h ;:  
debug018:0067D4F6 db 0ADh ;:  
debug018:0067D4F7 db 0BAh ;:  
debug018:0067D4F8 db 0Dh ;:  
debug018:0067D4F9 db 0F0h ;:  
debug018:0067D4FA db 0ADh ;:  
debug018:0067D4FB db 0BAh ;:  
debug018:0067D4FC db 0Dh ;:  
debug018:0067D4FD db 0F0h ;:  
debug018:0067D4FE db 0ADh ;:  
debug018:0067D4FF db 0BAh ;:  
debug018:0067D500 db 0Dh ;:  
debug018:0067D501 db 0F0h ;:  
debug018:0067D502 db 0ADh ;:  
debug018:0067D503 db 0BAh ;:  
debug018:0067D504 db 0EEh ;:  
debug018:0067D505 db 0FEh ;:  
debug018:0067D506 db 0ABh ;:  
debug018:0067D507 db 0ABh ;:  
debug018:0067D508 db 0ABh ;:  
debug018:0067D509 db 0ABh ;:  
debug018:0067D50A db 0ABh ;:  
debug018:0067D50B db 0ABh ;:  
debug018:0067D50C db 0ABh ;:  
debug018:0067D50D db 0ABh ;:  
debug018:0067D50E db 0EEh ;:  
debug018:0067D50F db 0FEh ;
```

As the heap is in debug mode, when running from the debugger the 0x16 looks clear. It is 22 decimal if we convert it in dwords from the start.

```
debug018:0067D4E0 dd 00000000  
debug018:0067D4F0 dd 0BAADF00Dh  
debug018:0067D4F4 dd 0BAADF00Dh  
debug018:0067D4F8 dd 0BAADF00Dh  
debug018:0067D4FC dd 0BAADF00Dh  
debug018:0067D500 dd 0BAADF00Dh  
debug018:0067D504 dw 0FEEEh  
debug018:0067D506 db 0ABh ; %
```

As it is in debug mode (if it isn't like this, it won't work) and there is nothing written yet, we see BAAD FOOD. ☺

They are 5 dwds or a 20-byte buffer + the 2 final bytes: 0xFFFF. I asked it 0x16 or 22 decimal.

Python> 0x16

22

We'll see more stuff about the heap. By now, that's the buffer full of **BAD FOOD**.
😊

Then, it saves the address in the stack pointer variable.

```

new d:\drive\curso c++\clase 3\punteros_y_referencias\punteros_y_referencias\ubicacion.cpp
001F102A mov    esi, offset aHolaReverserEn ; "hola reverser en stack"
001F102F lea    edi, [ebp+mensaje_en_stack]
001F1032 rep    mowsd
001F1036 movsb
001F1037 mov    [ebp+mensaje_en_data], offset aHolaReverser_0 ; "hola reverser en data"
001F1041 mov    eax, [ebp+mensaje_en_data]
001F1047 push   eax, [ebp+mensaje_en_data] ; char *
001F1048 call   _strtol
001F104D add    esp, 4
001F1050 add    eax, 1
001F1053 push   eax, [ebp+mensaje_en_data] ; Size
001F1054 call   ds:_imp_malloc
001F105A add    esp, 4
001F105D mov    [ebp+mensaje_en_heap], eax
001F1063 push   offset aHolaReverser_1 ; "hola reverser en stack sin inicializar"
001F1068 lea    ecx, [ebp+mensaje_en_stack_sin_inicializar]
001F106B push   ecx, [ebp+mensaje_en_data] ; char *
001F106C call   _strcpy
001F1071 add    esp, 8
001F1074 mov    edx, [ebp+mensaje_en_data]
001F107A push   edx, [ebp+mensaje_en_data] ; char *
001F107B mov    eax, [ebp+mensaje_en_h][ebp+mensaje_en_data]=[Stack[000038C8]:0038FAA4]
001F1081 push   eax, [ebp+mensaje_en_data] ; char
001F1082 call   _strcpy
001F1087 add    esp, 8
001F108A push   4, [ebp+mensaje_en_data] ; size_t
001F108C push   4, offset aHeap ; "heap"
001F1091 push   offset aHolaReverser_2 ; "hola reverser en "

```

General registers
EAX 0067D4F0 ↗ debug018:0067D4F0
EBX 7EFDE000 ↗ debug012:7EFDE000
ECX 77843516 ↗ ntdll.dll:ntdll.Rt
EDX 0067D4EB ↗ debug018:0067D4EB
ESI 001F2127 ↗ .rdata:001F2127
EDI 0038FB23 ↗ Stack[000038C8]:00
EBP 0038FB28 ↗ Stack[000038C8]:00
ESP 0038FA98 ↗ Stack[000038C8]:00
EIP 001F105D ↗ ejemplos_ubicacion
EFL 00000202

Then, in that strcpy, it will copy the **hola reverser en data** string in **data** in my baad food. 😊

```

001F1050 add    eax, 1
001F1053 push   eax, [ebp+mensaje_en_data] ; Size
001F1054 call   ds:_imp_malloc
001F105A add    esp, 4
001F105D mov    [ebp+mensaje_en_heap], eax
001F1063 push   offset aHolaReverser_1 ; "hola reverser en stack sin inicializar"
001F1068 lea    ecx, [ebp+mensaje_en_stack_sin_inicializar]
001F106B push   ecx, [ebp+mensaje_en_data] ; char *
001F106C call   _strcpy
001F1071 add    esp, 8
001F1074 mov    edx, [ebp+mensaje_en_data]
001F107A push   edx, [ebp+mensaje_en_data] ; char *
001F107B mov    eax, [ebp+mensaje_en_h][ebp+mensaje_en_data]=[Stack[000038C8]:0038FAA4]
001F1081 push   eax, [ebp+mensaje_en_data] ; char
001F1082 call   _strcpy
001F1087 add    esp, 8
001F108A push   4, [ebp+mensaje_en_data] ; size_t
001F108C push   4, offset aHeap ; "heap"
001F1091 push   offset aHolaReverser_2 ; "hola reverser en "

```

General registers
EBX 7EFDE000 ↗ debug012:7EFDE000
ECX 001F2168 ↗ .rdata:aHeap
EDX 001F2128 ↗ .rdata:aHolaRevers
ESI 001F2127 ↗ .rdata:001F2127
EDI 0038FB23 ↗ Stack[000038C8]:00
EBP 0038FB28 ↗ Stack[000038C8]:00
ESP 0038FA90 ↗ Stack[000038C8]:00
EIP 001F1082 ↗ ejemplos_ubicacion
EFL 00000200

If I trace it with F8...

```

debug018:0067D4EC dd 1A003AE8h
debug018:0067D4F0 aHolaReverser_3 db 'hola reverser en data',0 ; DATA XREF: Stack[000038C8]:0038FAA80
debug018:0067D506 db 0ABh ; %
debug018:0067D507 db 0ABh ; %
debug018:0067D508 db 0ABh ; %
debug018:0067D509 db 0ABh ; %
debug018:0067D50A db 0ABh ; %
debug018:0067D50B db 0ABh ; %

```

I see the copied bytes in the heap buffer. By pressing the letter A, I convert them into ASCII.

```

001F1087 add    esp, 8
001F108A push   4          ; size_t
001F108C push   offset aHeap    ; "heap"
001F1091 push   offset aHolaReverser_2 ; "hola reverser en "
001F1096 call   _strlen
001F109B add    esp, 4
001F109E add    eax, [ebp+mensaje_en_heap]

```

There, it will get the “hola reverser en” length.

General register
EAX 00000011
EBX 7EFDE000
ECX 001F2170
EDX 7EFFFF1F
ESI 001F2127
EDI 0038FB23
EBP 0038FB28
ESP 0038FA8C
EIP 001F109B
EFL 00000204

```

001F104D add    esp, 4
001F1050 add    eax, 1
001F1053 push   eax          ; Size
001F1054 call   ds:_imp__malloc
001F105A add    esp, 4
001F105D mov    [ebp+mensaje_en_heap], eax
001F1063 push   offset aHolaReverser_1 ; "hola reverser en stack sin inicializar"
001F1068 lea    ecx, [ebp+mensaje_en_stack_sin_inicializar]
001F106B push   ecx,          ; char *
001F106C call   _strcpy
001F1071 add    esp, 8
001F1074 mov    edx, [ebp+mensaje_en_data]
001F107A push   edx,          ; char *
001F107B mov    eax, [ebp+mensaje_en_heap]
001F1081 push   eax,          ; char *
001F1082 call   _strcpy
001F1087 add    esp, 8
001F108A push   4          ; size_t
001F108C push   offset aHeap    ; "heap"
001F1091 push   offset aHolaReverser_2 ; "hola reverser en "
001F1096 call   _strlen
001F109B add    esp, 4
001F109E add    eax, [ebp+mensaje_en_heap]
001F1094 push   eax,          ; void *
001F10A5 call   memenu

```

It gives 11, then it adds the address to the buffer start in the heap.

General registers
EAX 0067D501 ↗ debug012:aHolaReverser_3+11
EBX 7EFDE000 ↗ debug012:7EFDE000
ECX 001F2170 ↗ .rdata:aHolaReverser_2
EDX 7EFFFF1F ↗
ESI 001F2127 ↗ .rdata:001F2127
EDI 0038FB23 ↗ Stack[000038C8]:0038FB23
EBP 0038FB28 ↗ Stack[000038C8]:0038FB28
ESP 0038FA90 ↗ Stack[000038C8]:0038FA90
EIP 001F1094 ↗ ejemplos_ubicacion(void)+94
EFL 00000202

```

001F1060 call   _strcpy
001F1071 add    esp, 8
001F1074 mov    edx, [ebp+mensaje_en_data]
001F107A push   edx,          ; char *
001F107B mov    eax, [ebp+mensaje_en_heap]
001F1081 push   eax,          ; char *
001F1082 call   _strcpy
001F1087 add    esp, 8
001F108A push   4          ; size_t
001F108C push   offset aHeap    ; "heap"
001F1091 push   offset aHolaReverser_2 ; "hola reverser en "
001F1096 call   _strlen
001F1097 add    esp, 4
001F1098 add    eax, [ebp+mensaje_en_heap]
001F1099 push   eax,          ; void *
001F109C call   _memcpy
001F109D add    esp, 8Ch
001F10AD push   offset aDondeSeUbicara ; "Donde se ubicara?"

```

If I decompose the original string in bytes...

IDA View-EIP Structures Enums

```

IDA View-EIP      x  d:\drive\curso c++\clase 3\punteros_y_referencias\punteros_y_referencias\ubicacion.cpp
:0067D4E6 db 0
:0067D4E7 db 0
:0067D4E8 dd 6DF3C789h
:0067D4EC dd 1A003AE8h
:0067D4F0 byte_67D4F0 db 68h ; DATA XREF: Stack[000038C8]:0038FAA0$0
:0067D4F1 db 6Fh ; o
:0067D4F2 db 6Ch ; l
:0067D4F3 db 61h ; a
:0067D4F4 db 20h
:0067D4F5 db 72h ; r
:0067D4F6 db 65h ; e
:0067D4F7 db 76h ; v
:0067D4F8 db 65h ; e
:0067D4F9 db 72h ; r
:0067D4FA db 73h ; s
:0067D4FB db 65h ; e
:0067D4FC db 72h ; r
:0067D4FD db 20h
:0067D4FE db 65h ; e
:0067D4FF db 6Eh ; n
:0067D500 db 20h
:0067D501 db 64h ; d
:0067D502 db 61h ; a
:0067D503 db 74h ; t
:0067D504 db 61h ; a
:0067D505 db 0
:0067D506 db 0ABh ; %
:0067D507 db 0ABh ; %
:0067D508 db 0ABh ; %

General registers
EAX 0067D501 ↳ debug018:aHolaRever
EBX 7EFDE800 ↳ debug012:7EFDE800
ECX 001F2170 ↳ .rdata:aHolaRever
EDX 7EEFF1F ↳
ESI 001F2127 ↳ .rdata:001F2127
EDI 0038FB23 ↳ Stack[000038C8]:!
EBP 0038FB28 ↳ Stack[000038C8]:!
ESP 0038FA90 ↳ Stack[000038C8]:!
EIP 001F1004 ↳ ejemplos_ubicacion
EFL 00000202

```

Then, I press A there.

```

ebug018:0067D4E6 db 0
ebug018:0067D4E7 db 0
ebug018:0067D4E8 dd 6DF3C789h
ebug018:0067D4EC dd 1A003AE8h
ebug018:0067D4F0 byte_67D4F0 db 68h
ebug018:0067D4F1 db 6Fh ; o
ebug018:0067D4F2 db 6Ch ; l
ebug018:0067D4F3 db 61h ; a
ebug018:0067D4F4 db 20h
ebug018:0067D4F5 db 72h ; r
ebug018:0067D4F6 db 65h ; e
ebug018:0067D4F7 db 76h ; v
ebug018:0067D4F8 db 65h ; e
ebug018:0067D4F9 db 72h ; r
ebug018:0067D4FA db 73h ; s
ebug018:0067D4FB db 65h ; e
ebug018:0067D4FC db 72h ; r
ebug018:0067D4FD db 20h
ebug018:0067D4FE db 65h ; e
ebug018:0067D4FF db 6Eh ; n
ebug018:0067D500 db 20h
ebug018:0067D501 aData db 'data',0
ebug018:0067D506 db 0ABh ; %
ebug018:0067D507 db 0ABh ; %
ebug018:0067D508 db 0ABh ; %

```

```

001F107B push    eax, [ebp+mensaje_en_heap]
001F1081 push    eax           ; char *
001F1082 call    _strcpy
001F1087 add     esp, 8
001F108A push    4             ; size_t
001F108C push    offset aHeap  ; "heap"
001F1091 push    offset aHolaReverser_2 ; "hola reverser en "
001F1096 call    _strlen
001F109B add     esp, 4
001F109E add     eax, [ebp+mensaje_en_heap]
001F10A4 push    eax           ; void *
001F10A5 call    _memcpy
001F10AA add     esp, eax=debug018:aData
001F10AD push    offset aData  db 'data',0 se ubicara?
001F10B2 push    offset char * string2 ; char *
001F10B7 call    _strcpy

```

It will write 4 bytes stepping the word “data” with “heap”.

```

debug018:0067D4E5 db    0
debug018:0067D4E6 db    0
debug018:0067D4E7 db    0
debug018:0067D4E8 dd 6DF3C789h
debug018:0067D4EC dd 1A003AE8h
debug018:0067D4F0 byte_67D4F0 db 68h
debug018:0067D4F1 db 6Fh ; o
debug018:0067D4F2 db 6Ch ; l
debug018:0067D4F3 db 61h ; a
debug018:0067D4F4 db 20h
debug018:0067D4F5 db 72h ; r
debug018:0067D4F6 db 65h ; e
debug018:0067D4F7 db 76h ; v
debug018:0067D4F8 db 65h ; e
debug018:0067D4F9 db 72h ; r
debug018:0067D4FA db 73h ; s
debug018:0067D4FB db 65h ; e
debug018:0067D4FC db 72h ; r
debug018:0067D4FD db 20h
debug018:0067D4FE db 65h ; e
debug018:0067D4FF db 6Eh ; n
debug018:0067D500 db 20h
debug018:0067D501 aData db 'heap',0
debug018:0067D506 db 0ABh ; %
debug018:0067D507 db 00Bh ; %

```

So, I can decompose the word **heap** and create the whole string.

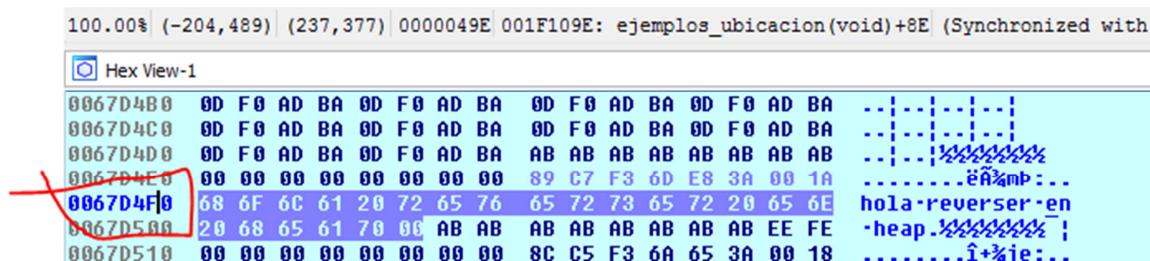
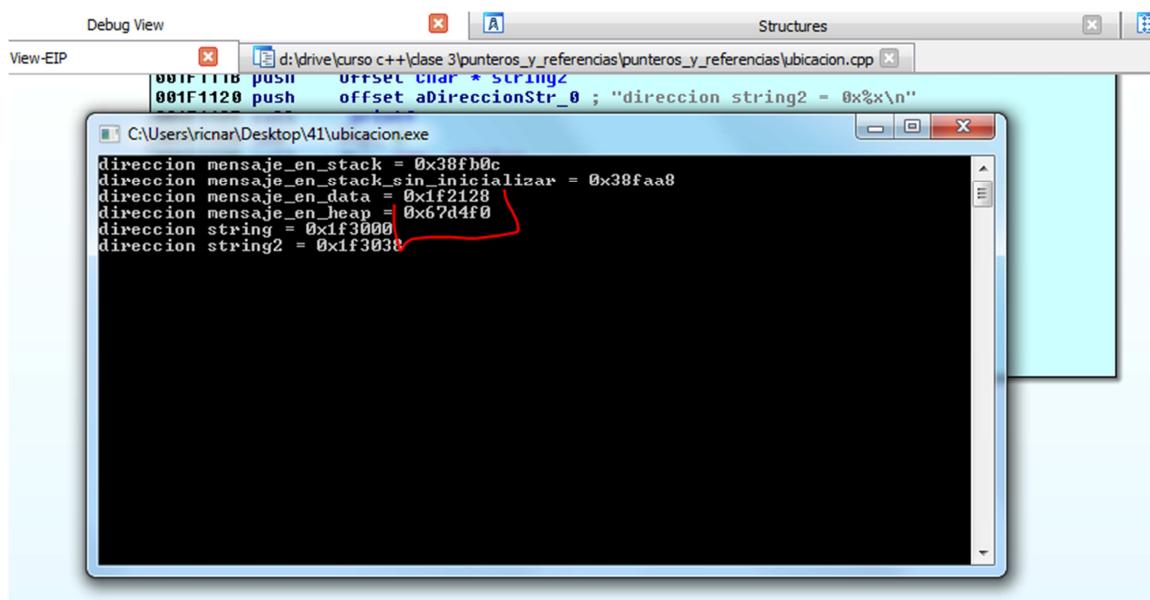
IDA View-Elf

```

debug018:0067D4E5 db 0
debug018:0067D4E6 db 0
debug018:0067D4E7 db 0
debug018:0067D4E8 dd 6DF3C789h
debug018:0067D4EC dd 1A003AE8h
debug018:0067D4F0 aHolaReverser_3 db 'hola reverser en heap',0
debug018:0067D4F0 ; DATA XREF: Start
debug018:0067D506 db 0ABh ; %
debug018:0067D507 db 0ABh ; %
debug018:0067D508 db 0ABh ; %
debug018:0067D509 db 0ABh ; %
debug018:0067D50A db 0ABh ; %
debug018:0067D50B db 0ABh ; %
debug018:0067D50C db 0ABh ; %
debug018:0067D50D db 0ABh ; %
debug018:0067D50E db 0EEh ;
debug018:0067D50F db 0FEh ; :
debug018:0067D510 db 0
debug018:0067D511 db 0
debug018:0067D512 db 0

```

The string is already created in the heap. It only remains to print the address.



That's all folks. Try to practice, see the strings well and handle them comfortably to get used to it slowly.

Ricardo Narvaja

Translated by: @IvinsonCLS