

REVERSING WITH IDA PRO FROM SCRATCH

PART 27

We will solve the exercise that we left to solve in the previous part called IDA_STRUCT.7z. If you did not do it or you do not have it, download it from here.

http://ricardo.crver.net/WEB/INTRODUCCION%20AL%20REVERSING%20CON%20IDA%20PRO%20DESDE%20CERO/EJERCICIOS/IDA_STRUCT_RESOLVER%20DESPUES%20DE%20LA%20PARTE%2026.7Z

The executable is called ConsoleApplication4.exe and the symbols are in the same folder.

ConsoleApplication4.pdb.

When it starts and tells you that it will try to find the symbols, you can point to that file and load them with which it clears up a bit, but I will delete or rename the pdb file to load it without symbols which is what we will normally find, though more complex, it is more real.

Unexplored ■ Instruction ■ External symbol

DA View-A Hex View-1 Structures Enums Imports Exports

004014DE
004014DE
004014DE ; Attributes: library function
004014DE
004014DE public start
004014DE start proc near
004014DE
004014DE ; FUNCTION CHUNK AT 0040136F SIZE 00000012C BYTES
004014DE ; FUNCTION CHUNK AT 004014D8 SIZE 00000006 BYTES
004014DE
004014DE call sub_401873
004014E3 jmp loc_40136F
004014E3 Start endp ; sp-analysis Failed
004014E3

0040136F ; START OF FUNCTION CHUNK FOR start
0040136F
0040136F loc_40136F:
0040136F push 14h
00401371 push offset unk_402558
00401376 call sub_401B90

0% (297,-10) | (497,388) | 000008DE 004014DE: start | (Synchronized with Hex View-1)

Obviously, having no symbols, this will not detect the main. We can get to it as it is a console application looking for the argv or argc or if not, more generic, looking at the strings.

If we ever run it, we see that the first thing it does is ask us to enter a number. There, you see the strings that appear to be those of good boy and bad boy.

Address	Length	Type	String
.rdata:00CD2100	00000027	C	\nPlease Enter Your Number of Choice:\n
.rdata:00CD212C	00000019	C	Genious you are the man\n
.rdata:00CD2148	00000011	C	Not not and not\n
.rdata:00CD22DC	00000005	C	GCTL
.rdata:00CD22E8	00000009	C	.text\$mn
.rdata:00CD22FC	00000009	C	.idata\$5

Click on the string “Please enter your number....”

```

.rdata:004020F0          db 0Ah ; DATA XREF: sub_401080+4↑o
.rdata:004020FE          db 0
.rdata:004020FF          db 0
.rdata:00402100 aPleaseEnterYou db 0Ah ; DATA XREF: sub_401080+4↑o
.rdata:00402100          db 'Please Enter Your Number of Choice: ',0Ah,0
.rdata:00402127          align 4
.rdata:00402128 aD         db '%d',0 ; DATA XREF: sub_401080+18↑o
.rdata:00402128          align 4
.rdata:0040212C aGeniousYouAreT db 'Genious you are the man',0Ah,0
.rdata:0040212C          ; DATA XREF: sub_401150+9C↑o
.rdata:00402145          align 4

```

There, we see that it has references by passing the mouse by the arrow, or by pressing X in the direction.

Address	Length	Type	String
.rdata:004020FC	db 0		
.rdata:004020FD	db 0		
.rdata:004020FE	db 0		
.rdata:004020FF	db 0		
.rdata:00402100 aPleaseEnterYou	db 0Ah ; DATA XREF: sub_401080+4↑o		
.rdata:00402100	db 'Please Enter Your Number of Choice: ',0Ah,0		
	; sub_401150+6A↓p		
var_4	= dword ptr -4		
arg_0	= dword ptr 8		
	push ebp		
	mov ebp, esp		
	push ecx		
	push offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice: \\"...		
	call sub_401220		
			loc_4011FB↑o
.rdata:0040215C ExceptionInfo _EXCEPTION_POINTERS <offset dword_403028, offset dword_403078>			
.rdata:0040215C	; DATA XREF: sub_401510+ED↑o		

xrefs to aPleaseEnterYou		
Direction	Ty	Address
Up	o	sub_401080+4
Text		
push offset aPleaseEnterYou; "\nPlease Enter Your Number of Choice: \"..."		
OK Cancel Search Help		

Line 1 of 1

Let's go there.

```
00401080
00401080
00401080 ; Attributes: bp-based frame
00401080
00401080 sub_401080 proc near
00401080
00401080 var_4= dword ptr -4
00401080 arg_0= dword ptr 8
00401080
00401080 push ebp
00401081 mov ebp, esp
00401083 push ecx
00401084 push offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice: \"...
00401089 call sub_401220
0040108E add esp, 4
00401091 mov eax, [ebp+arg_0]
00401094 add eax, 10h
00401097 push eax
00401098 push offset aD      ; "%d"
0040109D call sub_401260
004010A2 add esp, 8
```

6,7) (89,81) 00000484 00401084: sub_401080+4 (Synchronized with Hex View-1)

We see that we are in a function, we see the string out there and the call to print it, as we have no symbols, it does not tell us that it is 0x401220 is printf, if we look inside it.

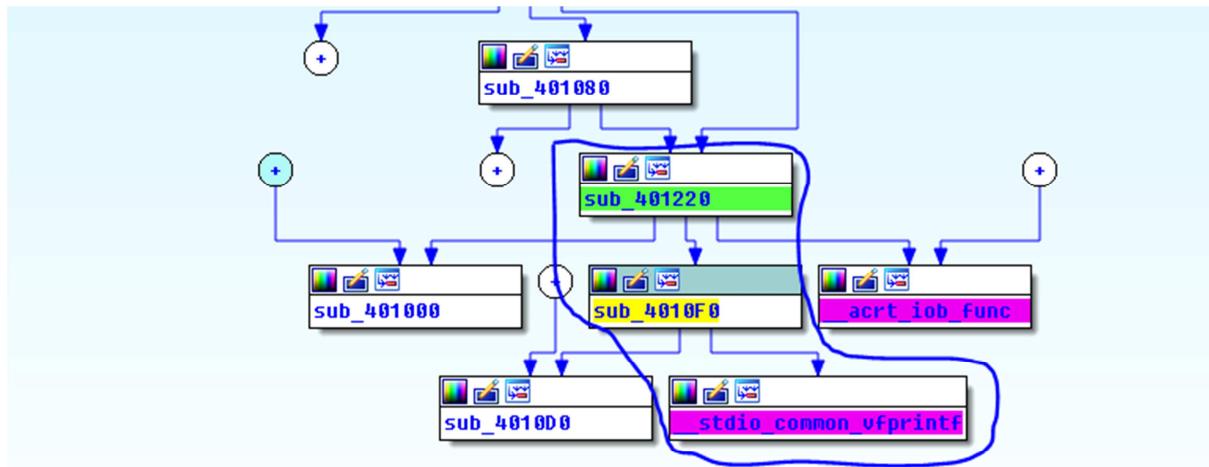
```
00401080
00401080 var_4= dword ptr -4
00401080 arg_0= dword ptr 8
00401080
00401080 push ebp
00401081 mov ebp, esp
00401083 push ecx
00401084 push offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice: \"...
00401089 call sub_401220
0040108E add esp, 4
```

You can look inside and see that there are several functions.

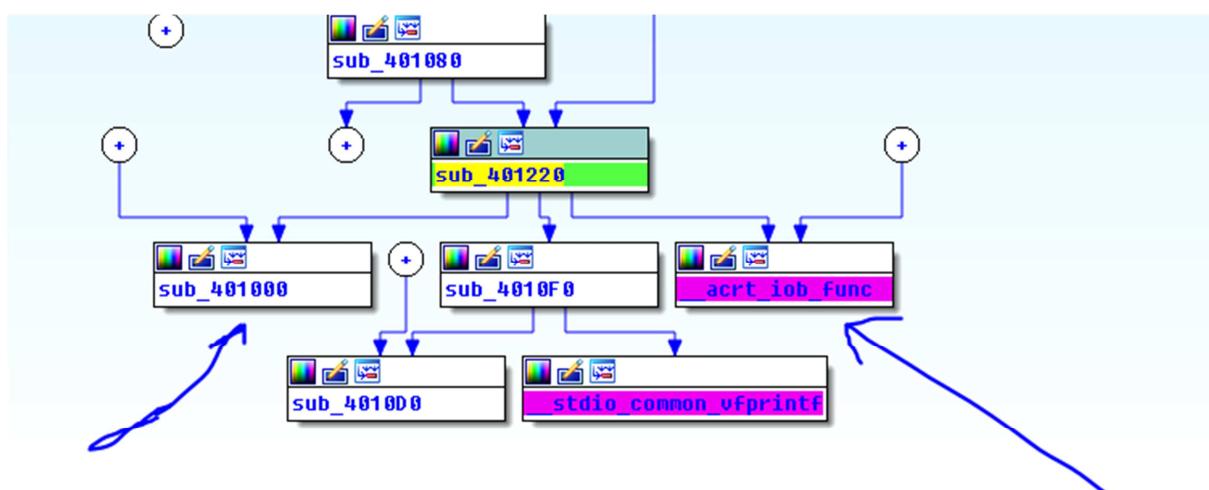
```

00401220
00401221 push    ebp
00401222 mov     ebp, esp
00401223 sub    esp, 8
00401226 call    sub_401000
00401228 lea    eax, [ebp+arg_4]
0040122E mov     [ebp+var_4], eax
00401231 mov     ecx, [ebp+var_4]
00401234 push    ecx
00401235 push    0
00401237 mov     edx, [ebp+arg_0]
0040123A push    edx
0040123B push    1
0040123D call    ds:_acrt_iob_func
00401243 add    esp, 4
00401246 push    eax
00401247 call    sub_4010F0
0040124C add    esp, 10h
0040124F mov     [ebp+var_8], eax
00401252 mov     [ebp+var_4], 0
00401259 mov     eax, [ebp+var_8]
0040125C mov     esp, ebp
D1247: sub_401220+27

```



In the proximity view that is entered by pressing the - key and exiting by pressing +, we see that 0x401220 calls those same three functions, but both 0x401000 and acrt_iob_func are functions that do something and return, do not follow to other child functions.



There, where the arrows that I added are, it is seen that it does not follow to other functions, the only one that follows is 0x4010f0 that calls to two functions and one is vfprintf, and from there it returns, there is no more down.

That can be seen also in the disassembly if I look inside each function I will see the same thing. We see that 0x401000 does not follow. It just makes a nuisance and comes back.

```
00401000
00401000
00401000 ; Attributes: bp-based frame
00401000
00401000 sub_401000 proc near
00401000 push    ebp
00401001 mov     ebp, esp
00401003 pop    ebp
00401004 retn
00401004 sub_401000 endp
00401004
```

```
0040123A push    edx
0040123B push    1
0040123D call    ds:_acrt_iob_func
00401243 add    esp, 4
00401246 push    eax
00401247 ...
```

And **_acrt_iob_func** is an API, so it will not continue. It will only initialize stdout and then print.

In visual studio 2015, stdin, stderr, stdout are defined as follow :

```
#define stdin  (_acrt_iob_func(0))
#define stdout (_acrt_iob_func(1)) ↗
#define stderr (_acrt_iob_func(2))
```

In other words, passing argument 1 as in our case will initialize stdout.

```
00401237 mov    edx, [ebp+arg_0]
0040123A push    edx
0040123B push    1
0040123D call    ds:_acrt_iob_func
00401243 add    esp, 4
00401247 ...
```

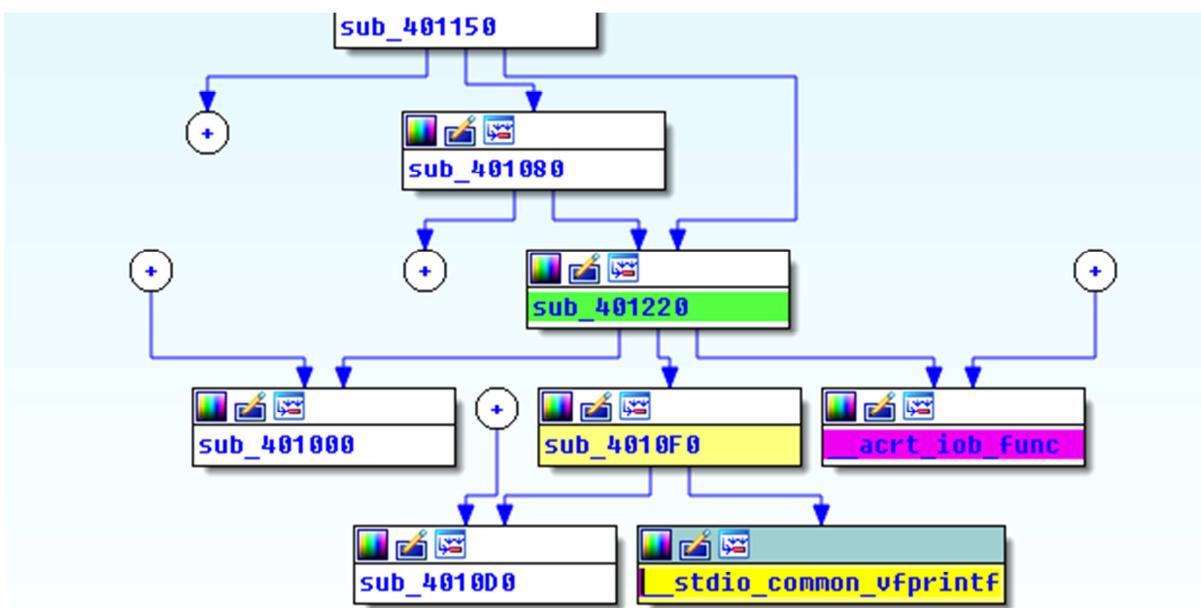
And the third function that calls.

```

00401000  arg_0= dword ptr  10h
004010F0  arg_C= dword ptr  14h
004010F0
004010F0 push    ebp
004010F1 mov     ebp, esp
004010F3 mov     eax, [ebp+arg_C]
004010F6 push    eax
004010F7 mov     ecx, [ebp+arg_8]
004010FA push    ecx
004010FB mov     edx, [ebp+arg_4]
004010FE push    edx
004010FF mov     eax, [ebp+arg_0]
00401102 push    eax
00401103 call    sub_4010D0
00401108 mov     ecx, [eax+4]
0040110B push    ecx
0040110C mov     edx, [eax]
0040110E push    edx
0040110F call    ds:_stdio_common_vfprintf
00401115 add    esp, 10h
00401118 pop    ebp
00401119 retn
0040111A sub    4010E8 ends

```

It ends by calling vfprintf, or we end up seeing the same as in the proximity view but it takes longer.



So, we rename 0x401220 as printf.

```

00401220
00401220
00401220 ; Attributes: bp-based frame
00401220
00401220 printf proc near
00401220
00401220 var_8= dword ptr -8
00401220 var_4= dword ptr -4
00401220 arg_0= dword ptr 8
00401220 arg_4= byte ptr 0Ch
00401220
00401220 push    ebp
00401221 mov     ebp, esp
00401223 sub    esp, 8
00401226 call   sub_401000
00401228 lea    eax, [ebp+arg_4]
0040122E mov    [ebp+var_4], eax
00401231 mov    ecx, [ebp+var_4]
00401234 push   ecx
00401235 push   0
00401237 mov    edx, [ebp+arg_0]
0040123A push   edx
0040123B push   1
0040123D call   ds:_acrt_iob_func

```

0000620 00401220: printf

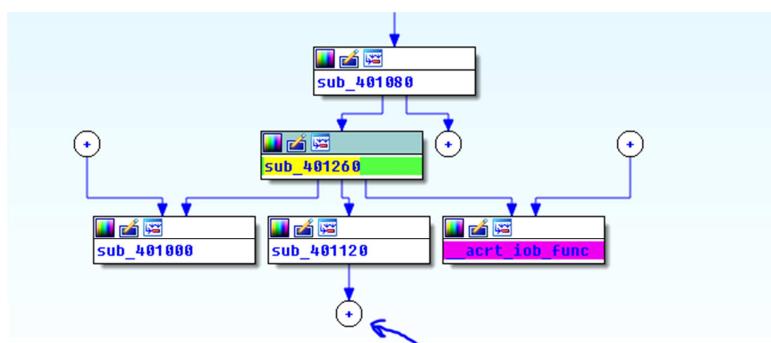
I paint celestial ones that end up being an API like in this case printf. Everyone will do it to their liking.

```

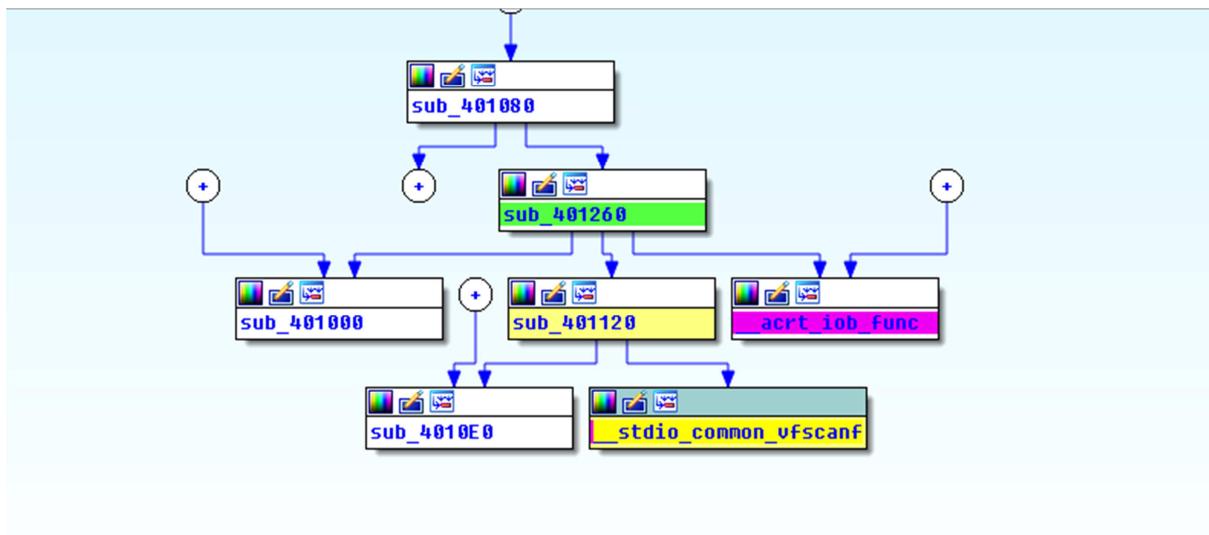
mov    esp, ebp
push   ecx
push   offset aPleaseEnterYou ; "\nPle
call   printf
add    esp, 4
mov    eax, [ebp+_struct]
add    eax, 10h
push   eax
push   offset aD ; "%d"
call   scanf
add    esp, 8

```

The next function of 0x40109D is surely scanf if we enter and we see the proximity view.



There, it follows. I click to make it unfold.



We see it is scanf.

```
00401277 mov     edx, [ebp+arg_0]
0040127A push    edx
0040127B push    0
0040127D call    ds:_acrt_iob_func
00401283 add     esp, 4
00401286 push    eax
0040128F ---
```

And in this case the function _acrt_iob_func with the argument 0 initializes stdin.

```
#define stdin  (_acrt_iob_func(0))
#define stdout (_acrt_iob_func(1))
#define stderr (_acrt_iob_func(2))
```

So, we rename scanf.

```
00401260
00401260
00401260 ; Attributes: bp-based frame
00401260
00401260 scanf proc near
00401260
00401260 var_8= dword ptr -8
00401260 var_4= dword ptr -4
00401260 arg_0= dword ptr  8
00401260 arg_4= byte ptr  0Ch
00401260
00401260 push    ebp
00401261 mov     ebp, esp
00401263 sub    esp, 8
00401266 call    sub_401000
0040126E lea    eax, [ebp+arg_4]
0040126F mov    [ebp+var_4], eax
00401271 mov    ecx, [ebp+var_4]
00401274 push    ecx
00401275 push    0
00401277 mov    edx, [ebp+arg_0]
0040127A push    edx
0040127B push    0
0040127D call    ds:_acrt_iob_func
```

The screenshot shows the assembly view of IDA Pro. At the top, the assembly code for `sub_401080` is displayed:

```

00401080 sub_401080 proc near
00401080 var_4= dword ptr -4
00401080 arg_0= dword ptr 8
00401080 push    ebp
00401081 mov     ebp, esp
00401083 push    ecx
00401084 push    offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice: \..."
00401089 call    printf
0040108E add    esp, 4
00401091 mov    eax, [ebp+arg_0]
00401094 add    eax, 10h
00401097 push    eax
00401098 push    offset aD ; "%d"
0040109D call    scanf
004010A2 add    esp, 8

```

Below it, two call sites are shown:

- `004010A5 loc_4010A5:` This block contains code to read a character from the console and store it in `arg_0`. It includes `call ds:getchar`, `mov [ebp+var_4], eax`, `mov [ecx+arg_0], eax`, `mov edx, [ebp+var_4]`, `mov [ecx+14h], edx`, `cmp [eax+var_4], 0Ah`, and `jz short loc_4010C8`.
- `004010BD mov eax, [ebp+arg_0]` and `004010C0 cmp dword ptr [eax+14h], 0FFFFFFFh`: These instructions are part of another function's logic, likely comparing the user input with a specific value.

We see something that is possibly a structure because when you pass an address as an argument and then retrieve and add offsets to access the fields in each place that is used, it is possibly the address of a structure.

Let's look at the references of this function.

The screenshot shows the "xrefs to sub_401080" dialog in IDA Pro. It lists two references:

Direction	Ty	Address	Text
Down	p	sub 401150+46	call sub 401080
Down	p	sub 401150+6A	call sub 401080

At the bottom, there are buttons for OK, Cancel, Search, and Help.

We see that there are two places if I look there.

```

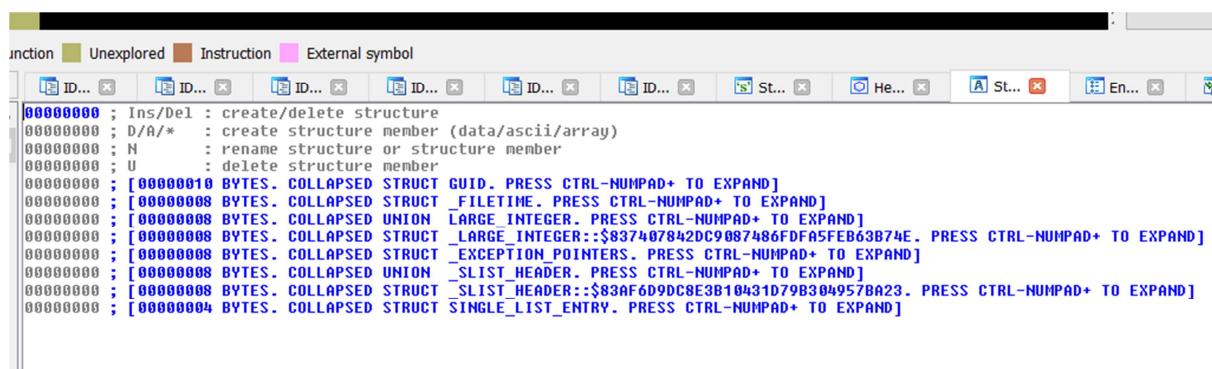
0040118E mov    [ebp+var_28], 0
00401192 lea    eax, [ebp+Buf]
00401195 push   eax
00401196 call   sub_401080
0040119B add    esp, 4
0040119E lea    ecx, [ebp+Buf]
004011A1 push   ecx ; Buf
004011A2 call   sub_401010
004011A7 add    esp, 4
004011AA lea    edx, [ebp+Buf]
004011AD push   edx
004011AE call   sub_401060
004011B3 add    esp, 4
004011B6 lea    eax, [ebp+var_44]
004011B9 push   eax
004011BA call   sub_401080
004011BF add    esp, 4

```

I see that the argument in both cases is a direction, which gives the idea of structures.

As two different directions give the impression that they were two structures of the same type, we will start creating a single one, without knowing the size, without knowing the fields or anything. We will reverse them little by little.

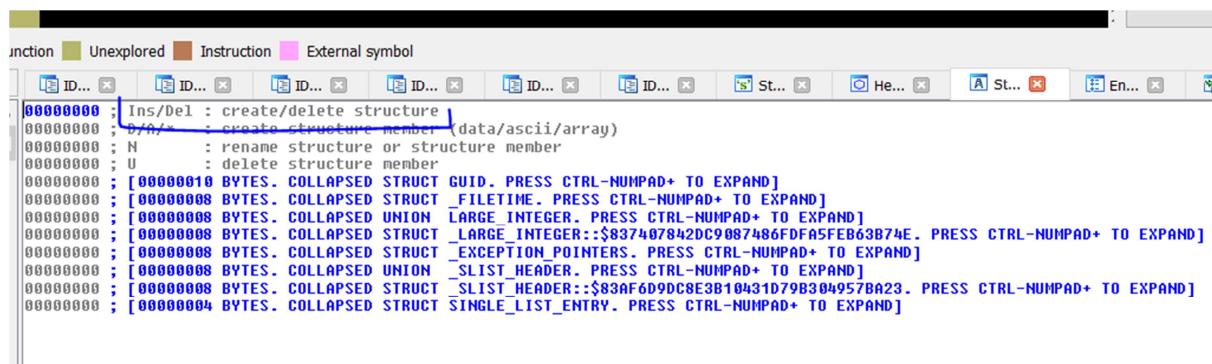
We see that the maximum offset that I find so far is 0x14 so I will create a structure of that length, if it becomes bigger I will enlarge it.



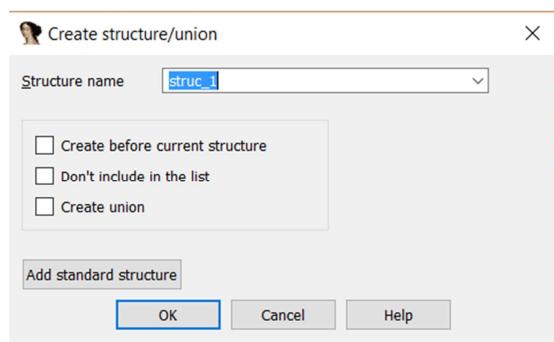
I go to the **structures** tab. It is one of the options to create it. The other would be to go to LOCAL TYPES and create it as code in C. We will do it for now here.

It's a little annoying and not really intuitive, but when we are in the place where it is defined to do CREATE STRUCT FROM SELECTION, we will usually create it here without knowing anything.

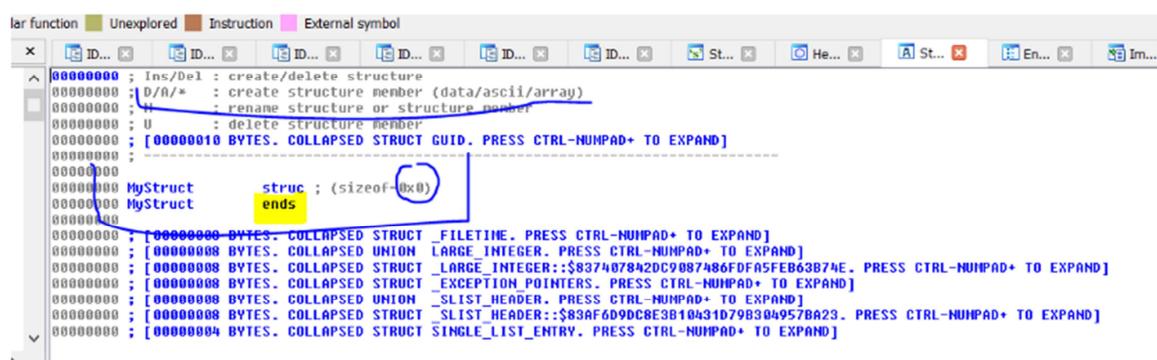
Obviously, if I analyze the representation of the stack of main, I could use CREATE STRUCT FROM SELECTION and it would make my life easier, but let's take the worst case, that we are in a function of a very large program and that we are very far from where it was defined and we have to fix them as we can.



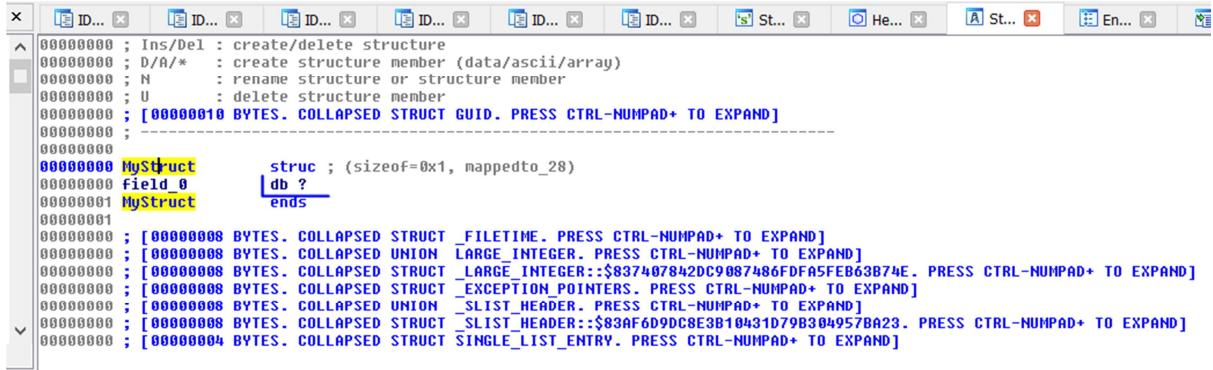
There, we see that to create a structure we have to press the INS key, we do it.



I can rename it as I want. I'll call it MyStruct.



There, it was created with size 0. Now, I will do a trick for when I still do not know the fields or anything and I want to give a size, first press D on the word **ends** to add a single field.



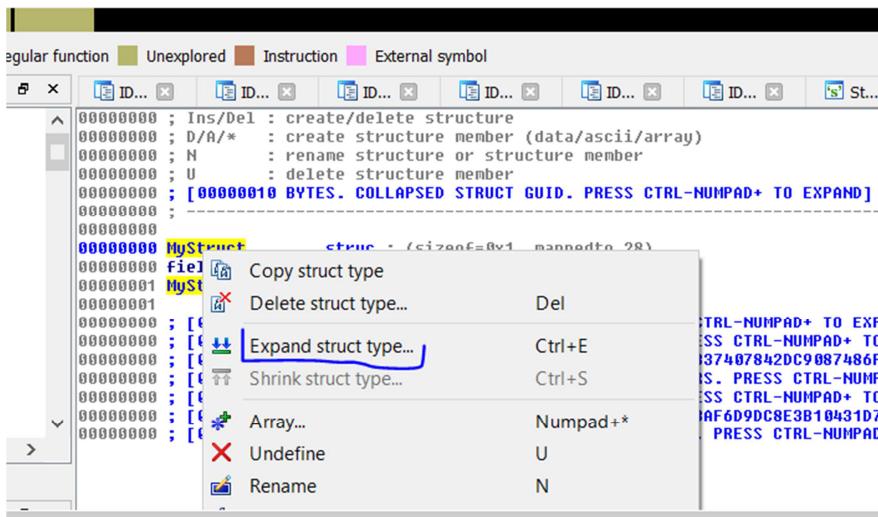
```

00000000 ; Ins/Del : create/delete structure
00000000 ; D/A/*   : create structure member (data/ascii/array)
00000000 ; N       : rename structure or structure member
00000000 ; U       : delete structure member
00000000 ; [ 00000010 BYTES. COLLAPSED STRUCT GUID. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ;
00000000 ;
00000000 MyStruct      struc ; (sizeof=0x1, mappedto_28)
00000000 Field_0     db ?
00000001 MyStruct      ends
00000001
00000000 ; [ 00000008 BYTES. COLLAPSED STRUCT _FILETIME. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [ 00000008 BYTES. COLLAPSED UNION _LARGE_INTEGER. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [ 00000008 BYTES. COLLAPSED STRUCT _LARGE_INTEGER:::$837407842DC9087486FDFA5FE863874E. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [ 00000008 BYTES. COLLAPSED STRUCT _EXCEPTION_POINTERS. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [ 00000008 BYTES. COLLAPSED UNION _SLIST_HEADER. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [ 00000008 BYTES. COLLAPSED STRUCT _SLIST_HEADER:::$83AF6D9DC8E3B10431D79B304957BA23. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [ 00000004 BYTES. COLLAPSED STRUCT SINGLE_LIST_ENTRY. PRESS CTRL-NUMPAD+ TO EXPAND]

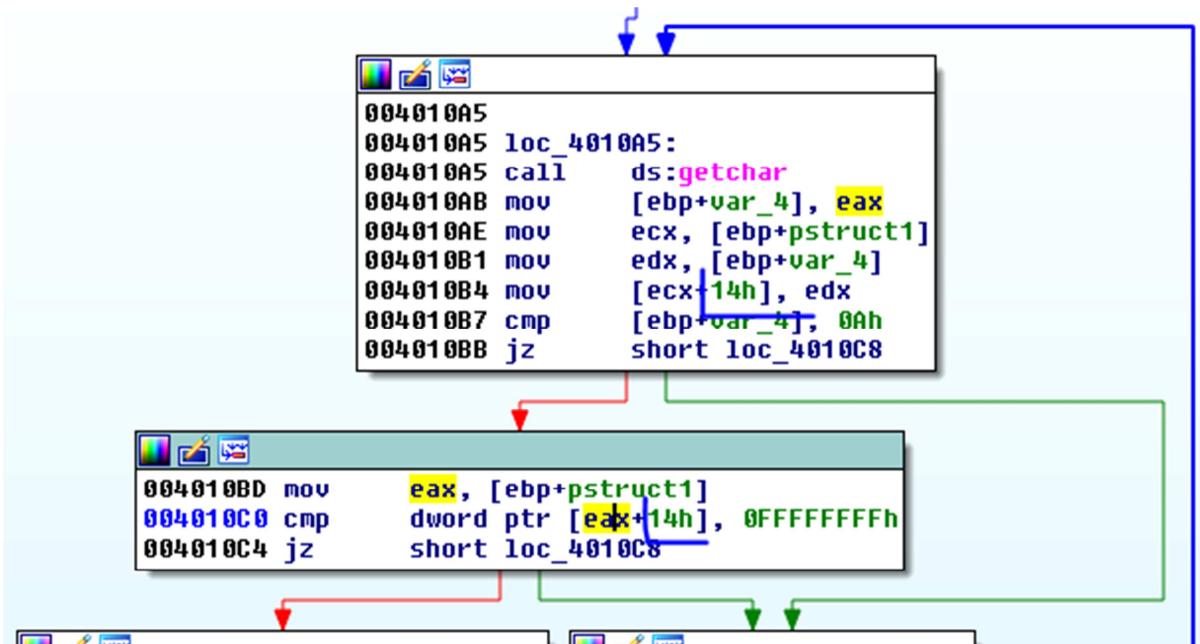
```

There, I add a field of 1 byte long DB. If I would press D again I would switch to word DW and then to DWORD DD.

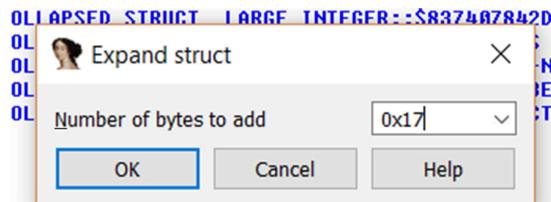
But here as we do not know, we leave it like this and right click on the structure.



Since I've seen a field in 0x14.



So, as to fill that field with a dword, it needs 4 more bytes, I will create it with 0x18. I will add 0x17 to the byte it had.



```
00000000  MyStruct      struc ; (sizeof=0x18, mappedto_29)
00000000  db ? ; undefined
00000001  db ? ; undefined
00000002  db ? ; undefined
00000003  db ? ; undefined
00000004  db ? ; undefined
00000005  db ? ; undefined
00000006  db ? ; undefined
00000007  db ? ; undefined
00000008  db ? ; undefined
00000009  db ? ; undefined
0000000A  db ? ; undefined
0000000B  db ? ; undefined
0000000C  db ? ; undefined
0000000D  db ? ; undefined
0000000E  db ? ; undefined
0000000F  db ? ; undefined
00000010  db ? ; undefined
00000011  db ? ; undefined
00000012  db ? ; undefined
00000013  db ? ; undefined
00000014  db ? ; undefined
00000015  db ? ; undefined
00000016  db ? ; undefined
00000017  Field_0
00000018  MyStruct      ends
```

I see that I was left with size 0x18 for now we will leave it like this. We need to enlarge it.

As this function is called twice, the first with the address of a first MyStruct type structure that we will call arbitrarily **pepe** and the second with the address of a second structure of the same MyStruct type that we will call **juan**, within the function we will put a generic name that serves both.

In the source code, this looks like this, to clarify two variables of MyStruct type one called **pepe** and another called **juan**, both are passed their address as argument to the functions.

```
APPLICATION4
MyStruct pepe;
MyStruct juan;
```

```
pepe.cookie = 0;
pepe.size = 0;
pepe.c = 0;
pepe.flag = false;

juan.cookie = 0;
juan.size = 0;
juan.c = 0;
juan.flag = false;
```

```
enter(&pepe);
check(&pepe);
desicion(&pepe);
```

```
enter(&juan);
check(&juan);
desicion2(&juan);
```

```
00401080
00401080
00401080 ; Attributes: bp-based frame
00401080
00401080 sub_401080 proc near
00401080
00401080 var_4= dword ptr -4
00401080 _struct= dword ptr 8
00401080
00401080 push    ebp
00401081 mov     ebp, esp
00401083 push    ecx
00401084 push    offset aPleaseEnterYou ; "\nPlease Enter Your Ni
00401089 call    printf
0040108E add     esp, 4
00401091 mov     eax, [ebp+_struct]
00401094 add     eax, 10h
00401097 push    eax
00401098 push    offset ad          ; "%d"
0040109D call    scanf
004010A2 add     esp, 8

.00% (-145,-85) (38,53) 00000491 00401091: sub_401080+11
```

Since the function will first have the address of the first structure or `pepe` in `arg0` and the second time it is called it will have the address of the `juan` structure, I will give it a generic name for both, for example `_struct`. If I decompile the function with F5 I see that it is not right.

```
1 int __cdecl sub_401080(int _struct)
2 {
3     char v1; // cl@0
4     int result; // eax@2
5
6     printf("\nPlease Enter Your Number of Choice: \n", v1);
7     scanf("%d", _struct + 16);
8     do
9     {
10         result = getchar();
11         *(_DWORD *)(_struct + 20) = result;
12         if ( result == 10 )
13             break;
14         result = _struct;
15     }
16     while ( *(_DWORD *)(_struct + 20) != -1 );
17     return result;
18 }
```

I see that the definition of the variable is a simple `int` and not as in the original code as the address of a structure. I can fix it here.

function Unexplored Instruction External symbol

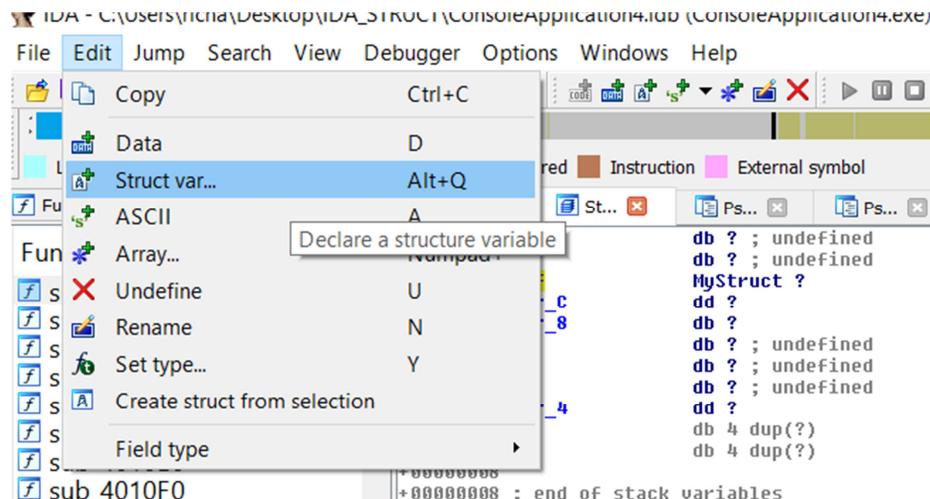
```
1 int __cdecl sub_401080(int _struct)
2 {
3     char v1; // cl@0
4     int result; // eax@2
5
6     printf("\nPlease Enter Your N
7     scanf("%d", _struct + 16);
8     do
9     {
10         result = getchar();
11         *(_DWORD *)(_struct + 20) =
12         if ( result == 10 )
13             break;
14         result = _struct;
15     }
16     while ( *(_DWORD *)(_struct + 20) != -1 );
17     return result;
18 }
```

Rename lvar N
Set lvar type Y
Convert to struct *
Create new struct type
Jump to xref... X
Edit func comment /
Mark as decompiled
Copy to assembly
Hide casts \

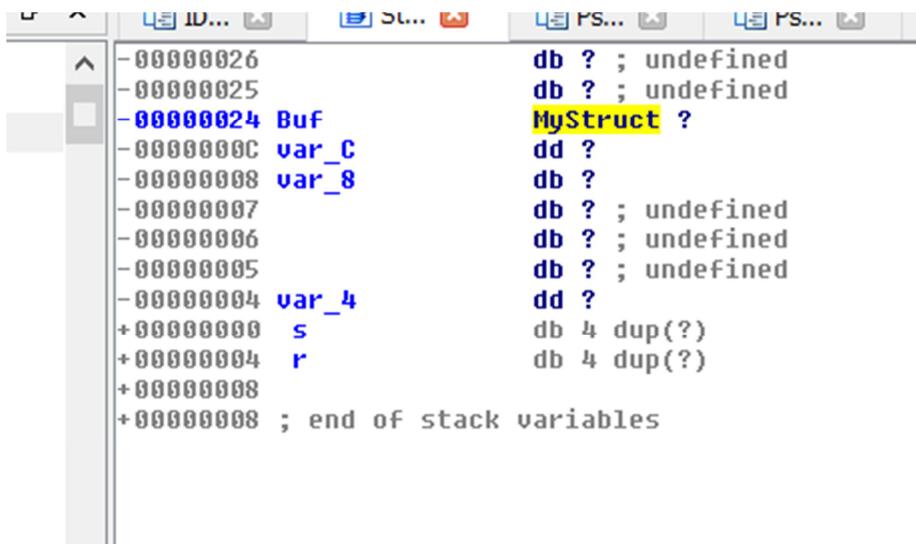
This shows to choose the address of which structure it is and here we will choose of MyStruct type.

105) 00000596 00401196: sub_401150+46

Obviously, Buf is pepe and there it gets its address and passes it as an argument, let's see Buf in the representation of the stack. As the structure is not necessary to create it because it already exists. I just have to tell you that Buf is MyStruct type. For that, press ALT + Q in Buf.



And it will allocate to Buf the MyStruct type if we put the size down, some fields will be left out, but after that it will be possible to enlarge MyStruct and it will be corrected only here (if it does not break, hehe)



```

-00000026 db ? ; undefined
-00000025 db ? ; undefined
-00000024 Buf MyStruct ?
-0000000C var_C dd ?
-00000008 var_8 db ?
-00000007 db ? ; undefined
-00000006 db ? ; undefined
-00000005 db ? ; undefined
-00000004 var_4 dd ?
+00000000 s db 4 dup(?)
+00000004 r db 4 dup(?)
+00000008
+00000008 ; end of stack variables

```

We rename Buf as pepe.

We see that, there, the address of pepe is passed and in the second call the address of var_44 is passed that will also be the other juan variable of the MyStruct type, so we go to the representation of the stack and in var_44 we also press ALT + Q.

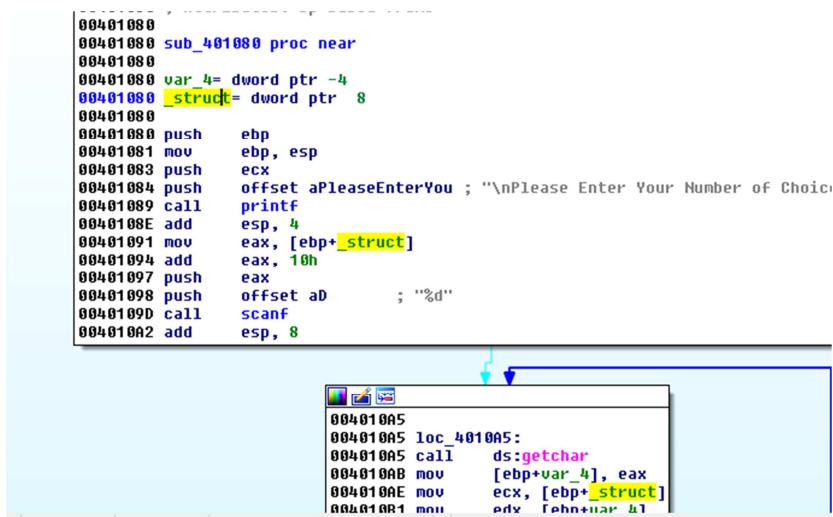


```

-00000044 ,
-00000044
-00000044 var_44 MyStruct ?
-0000002C var_2C dd ?
-00000028 var_28 db ?
-00000027 db ? ; undefined
-00000026 db ? ; undefined
-00000025 db ? ; undefined

```

We already have the two structures of MyStruct type. I return to the function.



```

00401080
00401080 sub_401080 proc near
00401080
00401080 var_4= dword ptr -4
00401080 _struct= dword ptr 8
00401080
00401080 push ebp
00401081 mov ebp, esp
00401083 push ecx
00401084 push offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice"
00401089 call printf
0040108E add esp, 4
00401091 mov eax, [ebp+_struct]
00401094 add eax, 10h
00401097 push eax
00401098 push offset aD ; "%d"
0040109D call scanf
004010A2 add esp, 8

```



```

004010A5
004010A5 loc_4010A5:
004010A5 call ds:getchar
004010A8 mov [ebp+var_4], eax
004010AE mov ecx, [ebp+_struct]
004010B1 mnu edx [ebp+var_4]

```

```

00401080 ; int __cdecl sub_401080(MyStruct *_struct)
00401080 sub_401080 proc near
00401080
00401080 var_4= dword ptr -4
00401080 _struct= dword ptr 8
00401080
00401080 push    ebp
00401081 mov     ebp, esp
00401083 push    ecx
00401084 push    offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice"
00401089 call    printf
0040108E add     esp, 4
00401091 mov     eax, [ebp+_struct]
00401094 add     eax, 10h
00401097 push    eax
00401098 push    offset aD           ; "%d"
0040109D call    scanf
004010A2 add     esp, 8

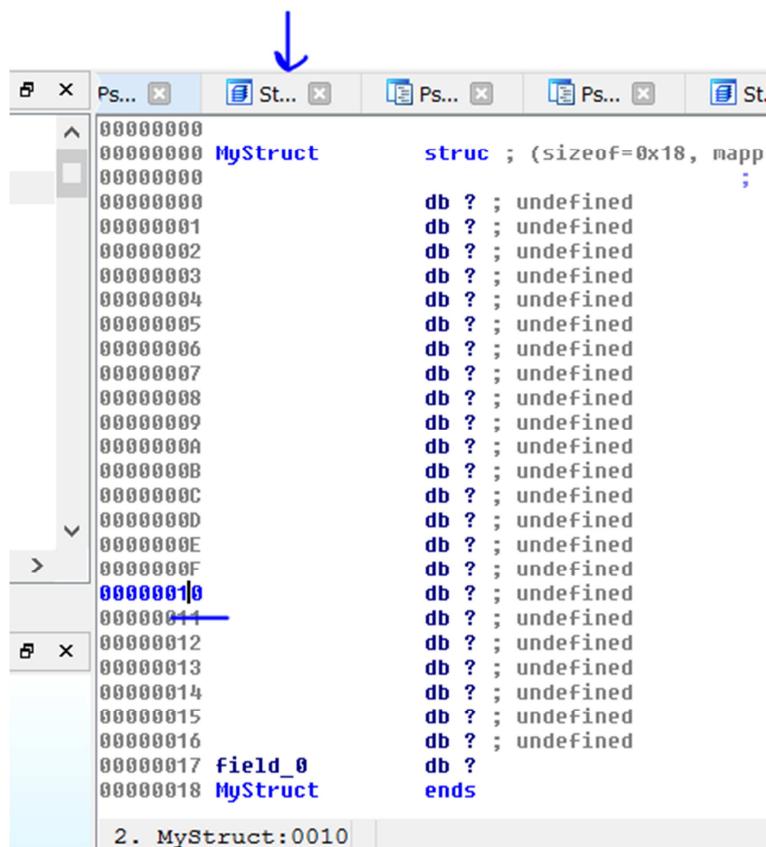
```

```

004010A5
004010A5 loc_4010A5:
004010A5 call    ds:getchar
004010AB mov     [ebp+var_4], eax
004010AF mou    ecx [ebp+_struct]

```

We see that the field in 0x10 is a dword where it receives the value of scanf, so we go to MyStruct and in 0x10 we press the D until it is of DWORD DD type.



```

00000000 MyStruct      struc ; (sizeof=0x18, mappedto_29) ; XREF: sub_401150/r
00000000
00000001
00000002
00000003
00000004
00000005
00000006
00000007
00000008
00000009
0000000A
0000000B
0000000C
0000000D
0000000E
0000000F
00000010 Field_18
00000014
00000015
00000016
00000017 Field_0
00000018 MyStruct      ends
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _FILETIME. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED UNION _LARGE_INTEGER. PRESS CTRL-NUMPAD+ TO EXPAND]

```

2. MvStruct:0010

I will rename it as numero.

```

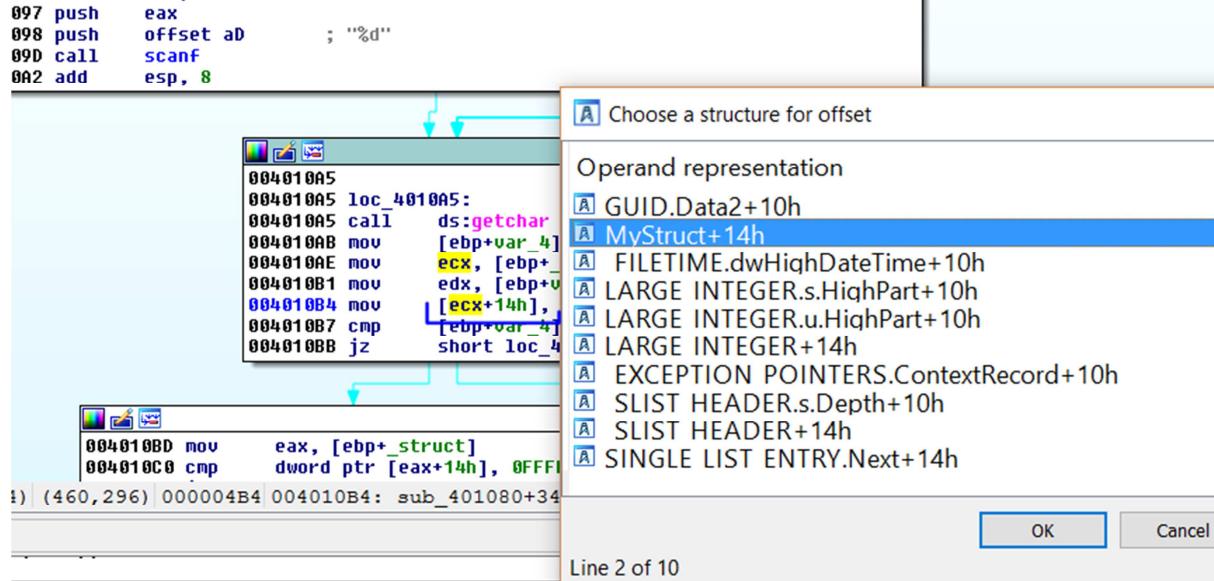
004010A5
004010A5 loc_4010A5:
004010A5 call    ds:getchar
004010AB mov     [ebp+var_4], eax
004010AE mov     ecx, [ebp+_struct]
004010B1 mov     edx, [ebp+var_4]
004010B4 mov     [ecx+14h], edx
004010B7 cmp     [ebp+var_4], 0Ah
004010BB jz      short loc_4010C8

004010BD mov     eax, [ebp+_struct]
004010C0 cmp     dword ptr [eax+14h], 0FFFFFFFh

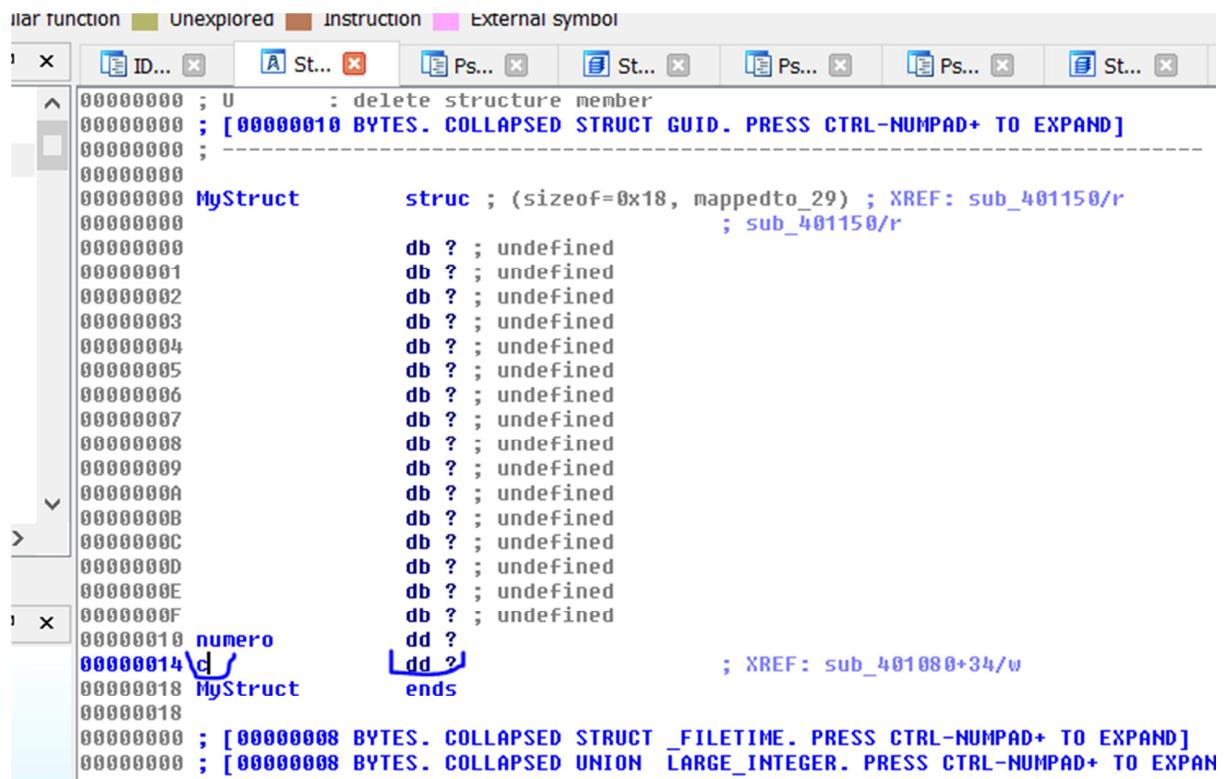
```

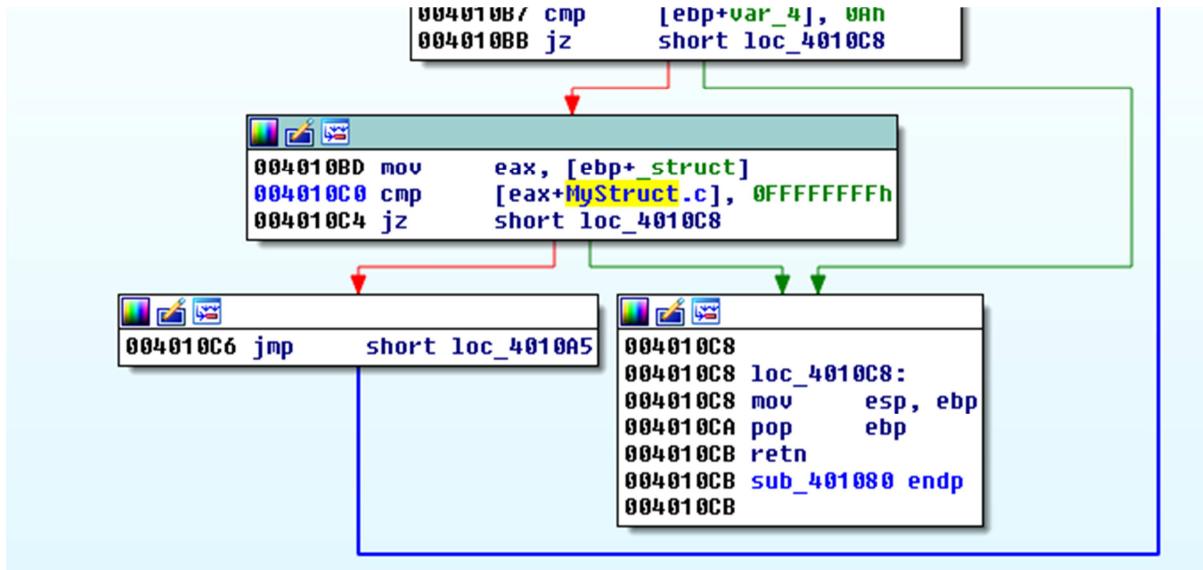
(33, 204) (383, 259) 000004AE 004010AE: sub 401080+2E

The other input is the 0x14 field that is used in the loop to remove the 0A. I will rename it c.



Let's go to MyStruct's 0x14 and press D until it's a DWORD and name it C.





There, we press T and it is already.

Finally, rename the function as **enter**.

00401180	mov [ebp+juan.numero], 0
00401187	mov [ebp+juan.c], 0
0040118E	mov [ebp+juan.flag], 0
00401192	lea eax, [ebp+pepe]
00401195	push eax ; _struct
00401196	call enter
0040119B	add esp, 4
0040119E	lea ecx, [ebp+pepe]

We see that the first three functions call them passing it pepe and the next three passes it juan.

Let's look at the next function.

ion Unexplored Instruction External symbol

ID... St... Ps... St... Ps... St... Ps... St... ID...

```

00401010
00401010 ; Attributes: bp-based frame
00401010
00401010 ; int _cdecl sub_401010(char *_struct)
00401010 sub_401010 proc near
00401010
00401010 _struct= dword ptr  8
00401010
00401010 push    ebp
00401011 mov     ebp, esp
00401013 mov     eax, [ebp+_struct]
00401016 cmp     dword ptr [eax+10h], 10h
0040101A jle     short loc_401024

```

```

0040101C push    1           ; Code
0040101E call    ds:_imp_exit

```

```

00401024
00401024 loc_401024:
00401024     mov     ecx, [ebp+_struct]
00401027     mov     edx, [ecx+10h]
0040102A     push    edx          ; Size
0040102B     mov     eax, [ebp+_struct]
0040102E     push    eax          ; Buf
0040102F     call    ds:gets_s
00401035     add    esp, 8
00401038     pop    ebp
00401039     retn

```

00.00% (-67, 31) (647, 426) 00000410 00401010: sub_401010

Also, this is called by both structures so you can do as in the previous one by pressing F5.

ID... Ps... St... Ps... St... Ps...

```

1 char * __cdecl sub_401010(char *_struct)
2 {
3     if ( *((_DWORD *) _struct + 4) > 16 )
4         exit(1);
5     return gets_s(_struct, *((_DWORD *) _struct + 4));
6 }

```

There, in the _struct variable, right click CONVERT TO STRUCT *.

ID... Ps... St... Ps... St... Ps...

```

1 char * __cdecl sub_401010(MyStruct *_struct)
2 {
3     if ( _struct->numero > 16 )
4         exit(1);
5     return gets_s(_struct->gap0, _struct->numero);
6 }

```

Now, this is the address of a MyStruct structure and just as before we will see the fields, pressing T where it corresponds.

```

00401010 ; Attributes: bp-based frame
00401010 ; char * __cdecl sub_401010(MyStruct *_struct)
00401010 sub_401010 proc near
00401010 _struct= dword ptr 8
00401010
00401010 push    ebp
00401011 mov     ebp, esp
00401013 mov     eax, [ebp+_struct]
00401016 cmp     [eax+MyStruct.numero], 10h
0040101A jle     short loc_401024

```


0040101C push : Code		
0040101E call ds:_imp_exit		

00401024	00401024 loc_401024:	
00401024 mov ecx, [ebp+_struct]		
00401027 mov edx, [ecx+MyStruct.numero]		
0040102A push edx		; Size
0040102B mov eax, [ebp+_struct]		
0040102E push eax		; Buf
0040102F call ds:gets_s		
00401035 add esp, 8		
00401038 pop ebp		
00401039 retn		

)% (-67, 22) (75, 43) 00000416 00401016: sub_401010+6

There, we see that it compares the number we passed with 0x10 and because the comparison is signed, any negative number can pass it as for example 0xFFFFFFFF which is -1 which is less than 0x10.

```

; Code
00401024 loc_401024:
00401024 mov     ecx, [ebp+_struct]
00401027 mov     edx, [ecx+MyStruct.numero]
0040102A push    edx
0040102B mov     eax, [ebp+_struct]
0040102E push    eax
0040102F call    ds:gets_s
00401035 add    esp, 8
00401038 pop    ebp
00401039 retn
00401039 sub_401010 endp
00401039

```

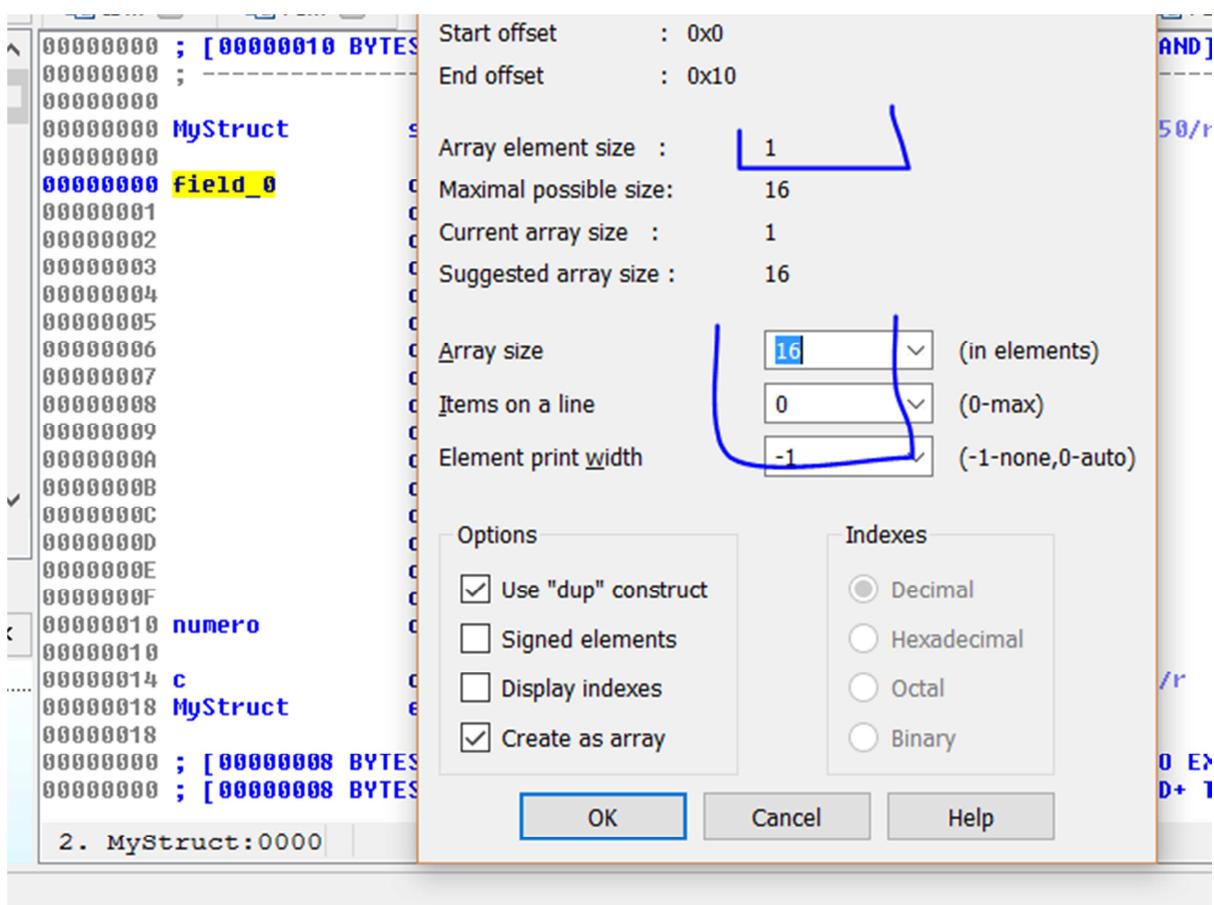
Then, it uses the number that we passed as the size of gets_s, and the other argument, it must be a buffer that is at the beginning of the structure because it uses its address. I go to MyStruct and in 0x0 press D once to create a single byte field.

```

00000000 ; [00000010 BYTES. COLLAPSED STRUCT GUID. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ;
00000000
00000000 MyStruct      struc ; (sizeof=0x18, mappedto_29) ; XREF: sub_401150/r
00000000 ; sub_401150/r
00000000 Field_0        db ?
00000001 db ? ; undefined
00000002 db ? ; undefined
00000003 db ? ; undefined
00000004 db ? ; undefined
00000005 db ? ; undefined
00000006 db ? ; undefined
00000007 db ? ; undefined
00000008 db ? ; undefined
00000009 db ? ; undefined
0000000A db ? ; undefined
0000000B db ? ; undefined
0000000C db ? ; undefined
0000000D db ? ; undefined
0000000E db ? ; undefined
0000000F db ? ; undefined
00000010 numero        dd ? ; XREF: sub_401010+6/r
00000010 ; sub_401010+17/r
00000014 c              dd ? ; XREF: enter+34/w enter+40/r
00000018 MyStruct      ends
00000018
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _FILETIME. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED UNION  LARGE_INTEGER. PRESS CTRL-NUMPAD+ TO EXPAND]

```

There, right click - array.



The length of the buffer will be 16. I accept it.

And I rename it as **Buffer**.

```

00000000 ; [00000010 BYTES. COLLAPSED STRUCT GUID. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ;
00000000 MyStruct     struc ; (sizeof=0x18, mappedto_29) ; XREF: sub_401150/r
00000000     Buffer      db 16 dup(?)           ; sub_401150/r
00000000     numero     dd ?                  ; XREF: sub_401010+6/r
00000000     c          dd ?                  ; sub_401010+17/r
00000000 MyStruct     ends
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _FILETIME. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED UNION _LARGE_INTEGER. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _LARGE_INTEGER::$837407842DC9087486FDFASFEB63B74E. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _EXCEPTION_POINTERS. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED UNION _SLIST_HEADER. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _SLIST_HEADER::$83AF6D9DC8E3B10431D79B304957BA23. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000004 BYTES. COLLAPSED STRUCT _SINGLE_LIST_ENTRY. PRESS CTRL-NUMPAD+ TO EXPAND]

```

Size = 16 decimal.

Let's continue reversing.

```

00401024
00401024 loc_401024:
00401024 mov    ecx, [ebp+_struct]
00401027 mov    edx, [ecx+MyStruct.numero]
0040102A push   edx           ; Size
0040102B mov    eax, [ebp+_struct]
0040102E push   eax           ; Buf
0040102F call   ds:gets_s
00401035 add    esp, 8
00401038 pop    ebp
00401039 retn
00401039 sub_401010 endp
00401039

```

02E: sub 401010+1E

The question is that the gets_s can be overflowed, since the check passes negative values when used as size. They will be taken as unsigned values and will be large,

If, for example, we pass 0xFFFFFFFF in the comparison, it will be -1 because it is signed and it will be less than 0x10, but using it as size, it will be the positive value 0xFFFFFFFF which allows us to pass the number of characters we want in the gets_s to the buffer and overflow it .

So we could rename the function as check or get whatever we want it to be representative of what the function does. We will check it to match.

```

00401010 ; Attributes: bp-based frame
00401010 ; char *__cdecl check(MyStruct *_struct)
00401010 check proc near
00401010 _struct= dword ptr 8
00401010
00401010 push    ebp
00401011 mov     ebp, esp
00401013 mov     eax, [ebp+_struct]
00401016 cmp     [eax+MyStruct.numero], 10h
0040101A jle     short loc_401024

0040101C push    1           ; Code
0040101E call    ds:_imp_exit
00401024 loc_401024:
00401024 mov     ecx, [ebp+_struct]
00401027 mov     edx, [ecx+MyStruct.numero]
0040102A push    edx         ; Size
0040102B mov     eax, [ebp+_struct]
0040102E push    eax         ; Buf
0040102F call    ds:gets_s
00401035 add    esp, 8
00401038 pop    ebp
00401039 retn

```

I have the third function. The argument is the same so I repeat the procedure, press F5 and change the argument type.

```

1 MyStruct *__cdecl sub_401060(MyStruct *_struct)
2 {
3     MyStruct *result; // eax@1
4
5     result = _struct;
6     if ( *_DWORD *)&_struct[1].Buffer[0] == -1718052970 )
7         _struct[1].Buffer[4] = 1;
8     return result;
9 }

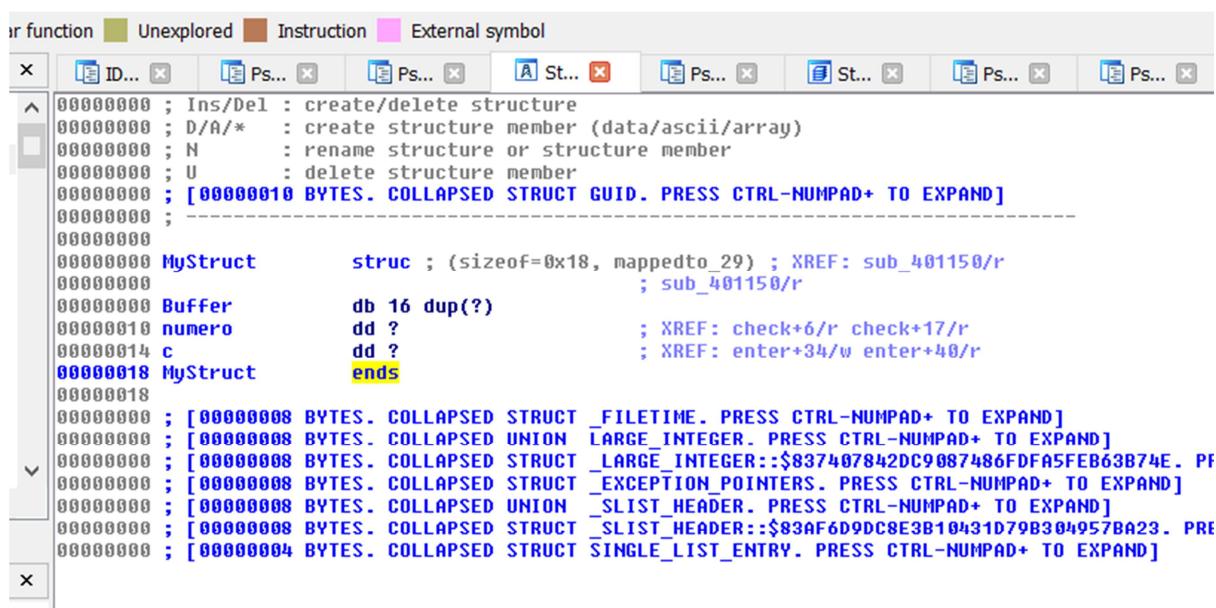
```

I keep working.

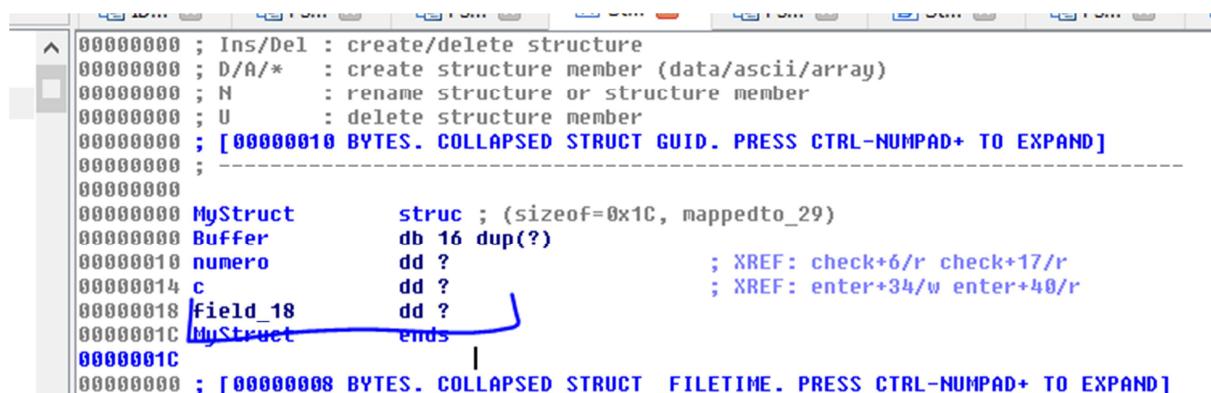
```
00401060 ; Attributes: bp-based frame
00401060
00401060 ; int __cdecl sub_401060(MyStruct *_struct)
00401060 sub_401060 proc near
00401060
00401060 _struct= dword ptr  8
00401060
00401060 push    ebp
00401061 mov     ebp, esp
00401063 mov     eax, [ebp+_struct]
00401066 cmp     dword ptr [eax+18h], 99989796h
0040106D jnz     short loc 401076

0040106F mov     ecx, [ebp+_struct]
00401072 mov     byte ptr [ecx+1Ch], 1
```

We see that there is one more field since it is trying to compare the [EAX + 0x18], which we have not defined since the last field of our MyStruct is 0x14. We will add it.



On the word **ends**, press D until a new DD DWORD field is created.



Rename it as cookie.

```
00000000 ; Ins/Del : create/delete structure
00000000 ; D/A/*   : create structure member (data/ascii/array)
00000000 ; N      : rename structure or structure member
00000000 ; U      : delete structure member
00000000 ; [ 00000010 BYTES. COLLAPSED STRUCT GUID. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ;
00000000
00000000 MyStruct      struc ; (sizeof=0x1C, mappedto_29)
00000000 Buffer        db 16 dup(?)
00000010 numero       dd ?
00000014 c             dd ?           ; XREF: check+6/r check+17/r
00000018 cookie        dd ?           ; XREF: enter+34/w enter+40/r
0000001C MyStruct      ends
0000001C
00000000 ; [ 00000008 BYTES. COLLAPSED STRUCT _FILETIME. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [ 00000008 BYTES. COLLAPSED UNION LARGE_INTEGER. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [ 00000008 BYTES. COLLAPSED STRUCT LARGE_INTEGER. PRESS CTRL-NUMPAD+ TO EXPAND]
```

I return to the function and press T.

```
00401060 ; int __cdecl sub_401060(MyStruct * _struct)
00401060 sub_401060 proc near
00401060 _struct= dword ptr 8
00401060
00401060 push    ebp
00401061 mov     ebp, esp
00401063 mov     eax, [ebp+_struct]
00401066 cmp     dword ptr [eax+18h], 99989796h
0040106D jnz    short
A Choose a structure for offset
```

Operand representation

- A GUID.Data4+10h
- A MyStruct.cookie
- A FILETIME.dwLowDateTime+18h
- A Large Integer

We see that there is another field. This is a single byte.

The screenshot shows a debugger interface with assembly code and a context menu. The assembly code is:

```
0040106F mov     ecx, [ebp+_struct]
00401072 mov     byte ptr [ecx+1Ch], 1
```

A context menu is open at the end of the second instruction, showing options like "Choose a structure for offset", "Operand representation", and a list of memory locations. One item in the list is highlighted: "MyStruct.cookie".

So, we go back to MyStruct and over ends we press D once and we will have a one byte field.

```

^ 00000000 ; u - DELETE STRUCTURE MEMBER
00000000 ; [00000010 BYTES. COLLAPSED STRUCT GUID. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ;
00000000 MyStruct      struc ; (sizeof=0x10, mappedto_29)
00000000 Buffer        db 16 dup(?)
00000010 numero       dd ? ; XREF: check+6/r check+17/r
00000014 c             dd ? ; XREF: enter+34/w enter+40/r
00000018 cookie        dd ? ; XREF: sub_401060+6/r
0000001C Flag |       db ?
0000001D MyStruct      ends
0000001D

00000000 ; [00000008 BYTES. COLLAPSED STRUCT _FILETIME. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED UNION _LARGE_INTEGER. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _LARGE_INTEGER::$837407842DC9087486FDFA5FEB63B74E. PRESS CTF]
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _EXCEPTION_POINTERS. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED UNION _SLIST_HEADER. PRESS CTRL-NUMPAD+ TO EXPAND]
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _SLIST_HEADER::$83AF6D9DC8E3B10431D79B304957BA23. PRESS CTRL]
00000000 ; [00000004 BYTES. COLLAPSED STRUCT SINGLE_LIST_ENTRY. PRESS CTRL-NUMPAD+ TO EXPAND]

```

I rename it as **flag** to match.

Returning to the function.

```

00401063 mov     eax, [ebp+_struct]
00401066 cmp     [eax+MyStruct.cookie], 99989796h
0040106D jnz     short loc_401076

```

```

0040106F mov     ecx, [ebp+_struct]
00401072 mov     [ecx+MyStruct.flag], 1

```

If cookie is equal to 0x99989796 then it will set the structure flag to 1.

```

00401060 ; Attributes: bp-based frame
00401060
00401060 ; int __cdecl desicion(MyStruct *_struct)
00401060 desicion proc near
00401060
00401060 _struct= dword ptr  8
00401060
00401060 push    ebp
00401061 mov     ebp, esp
00401063 mov     eax, [ebp+_struct]
00401066 cmp     [eax+MyStruct.cookie], 99989796h
0040106D jnz     short loc_401076

```

```

0040106F mov     ecx, [ebp+_struct]
00401072 mov     [ecx+MyStruct.flag], 1

```

```

00401076
00401076 loc_401076:
00401076 pop    ebp
00401077 retn
00401077 desicion endp
00401077

```

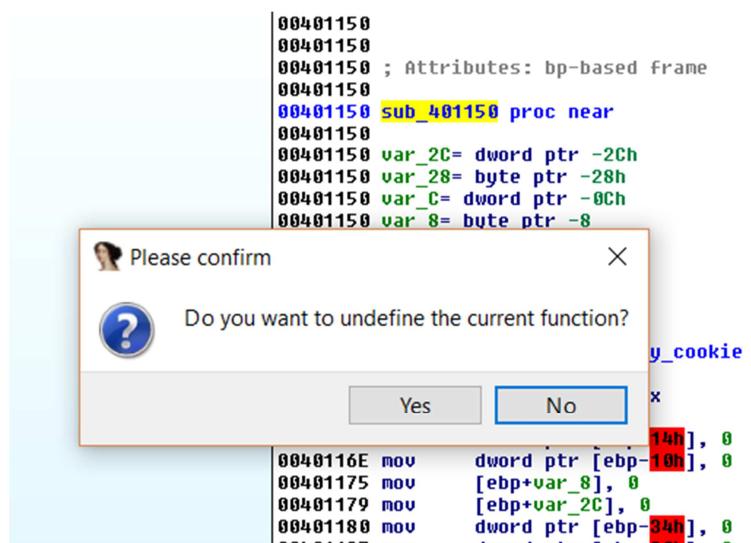
If for some change that did not propagate well, the main variables break and we did not snapshot as it happened to me.

```

00401150
00401150 sub_401150 proc near
00401150
00401150 var_2C= dword ptr -2Ch
00401150 var_28= byte ptr -28h
00401150 var_C= dword ptr -0Ch
00401150 var_8= byte ptr -8
00401150 var_4= dword ptr -4
00401150
00401150 push    ebp
00401151 mov     ebp, esp
00401153 sub    esp, 44h
00401156 mov     eax, __security_cookie
00401158 xor     eax, ebp
0040115D mov     [ebp+var_4], eax
00401160 mov     [ebp+var_C], 0
00401167 mov     dword ptr [ebp-14h], 0
0040116E mov     dword ptr [ebp-10h], 0
00401175 mov     [ebp+var_8], 0
00401179 mov     [ebp+var_2C], 0
00401180 mov     dword ptr [ebp-34h], 0
00401187 mov     dword ptr [ebp-30h], 0
0040118E mov     [ebp+var_28], 0
00401192 lea     eax, [ebp-24h]
00401195 push    eax      ; struct

```

I go to the beginning of the broken function, and press U.



I accept and it will remain so.

```

00401148 add    esp, 18h
0040114B pop    ebp
0040114C retn
0040114C endp
0040114C ;
00401150 al1on 10h
00401150 db    55h ; U
00401150 db    00h ; X
00401150 db    0Ech ; 8
00401150 db    83h ; 3
00401150 db    0ECh ; 8
00401150 db    4h ; D
00401150 db    0A1h ; I
00401150 db    4 ; OFF32 SEGDEF [_data,403004]
00401150 db    30h ; 0
00401150 db    40h ; @
00401150 db    0
00401150 db    33h ; 3
00401150 db    0C5h ; +
00401150 db    89h ; E
00401150 db    45h ; E
00401150 db    0FCh ; n
00401150 db    0C7h ; :
00000550 00401150: .text:unk_401150

```

Then, at the same start, I press C.

```
.text:00401140 ; -----  
.text:0040114C align 10h  
.text:00401140  
.text:00401150  
.text:00401150 loc_401150: ; CODE XREF: start-7B↑p  
.text:00401150 push ebp  
.text:00401151 mov ebp, esp  
.text:00401153 sub esp, 44h  
.text:00401156 mov eax, __security_cookie  
.text:00401158 xor eax, ebp  
.text:0040115D mov [ebp-4], eax  
.text:00401160 mov dword ptr [ebp-0Ch], 0  
.text:00401167 mov dword ptr [ebp-14h], 0  
.text:0040116E mov dword ptr [ebp-10h], 0  
.text:00401175 mov byte ptr [ebp-8], 0  
.text:00401179 mov dword ptr [ebp-2Ch], 0  
.text:00401188 mov dword ptr [ebp-34h], 0  
.text:00401187 mov dword ptr [ebp-30h], 0  
.text:0040118E mov byte ptr [ebp-28h], 0  
.text:00401192 lea eax, [ebp-24h]  
.text:00401195 push eax  
.text:00401196 call enter  
.text:00401196 add esp, 44h
```

And then, right click - CREATE FUNCTION.

```
00401150
00401150
00401150 ; Attributes: bp-based frame
00401150
00401150 sub_401150 proc near
00401150
00401150 var_44= MyStruct ptr -44h
00401150 _struct= MyStruct ptr -24h
00401150 var_4= dword ptr -4
00401150
00401150 push    ebp
00401151 mov     ebp, esp
00401153 sub    esp, 44h
00401156 mov     eax, __security_cookie
0040115B xor     eax, ebp
0040115D mov     [ebp+var_4], eax
00401160 mov     [ebp+_struct.cookie], 0
00401167 mov     [ebp+_struct.numero], 0
0040116E mov     [ebp+_struct.c], 0
00401175 mov     [ebp+_struct.flag], 0
00401179 mov     [ebp+var_44.cookie], 0
00401180 mov     [ebp+var_44.numero], 0
00401187 mov     [ebp+var_44.c], 0
0040118E mov     [ebp+var_44.flag], 0
00401192 lea     eax, [ebp+_struct]
00401195 push    eax ; _struct
```

Now, it's okay.

Seeing the representation of the stack.

```

000000044 , frame size= 44, saved regs= 4, purge= 0
-000000044 ;
-000000044
-000000044 var_44      MyStruct ?
-000000027 db ? ; undefined
-000000026 db ? ; undefined
-000000025 db ? ; undefined
-000000024 _struct
-000000007 MyStruct ?
-000000006 db ? ; undefined
-000000005 db ? ; undefined
-000000004 var_4      db ? ; undefined
+000000000 s          db 4 dup(?)
+000000004 r          db 4 dup(?)
+000000008
+000000008 ; end of stack variables

```

We see that after each structure there are three empty bytes because the last field was a single byte and there is nothing else.

Only the names of the structures went wrong, but I will change them as I did in the source code with Pepe and Juan.

```

int main(int argc, char *argv[])
{
    MyStruct pepe;
    MyStruct juan;

    pepe.cookie = 0;
    pepe.size = 0;
}

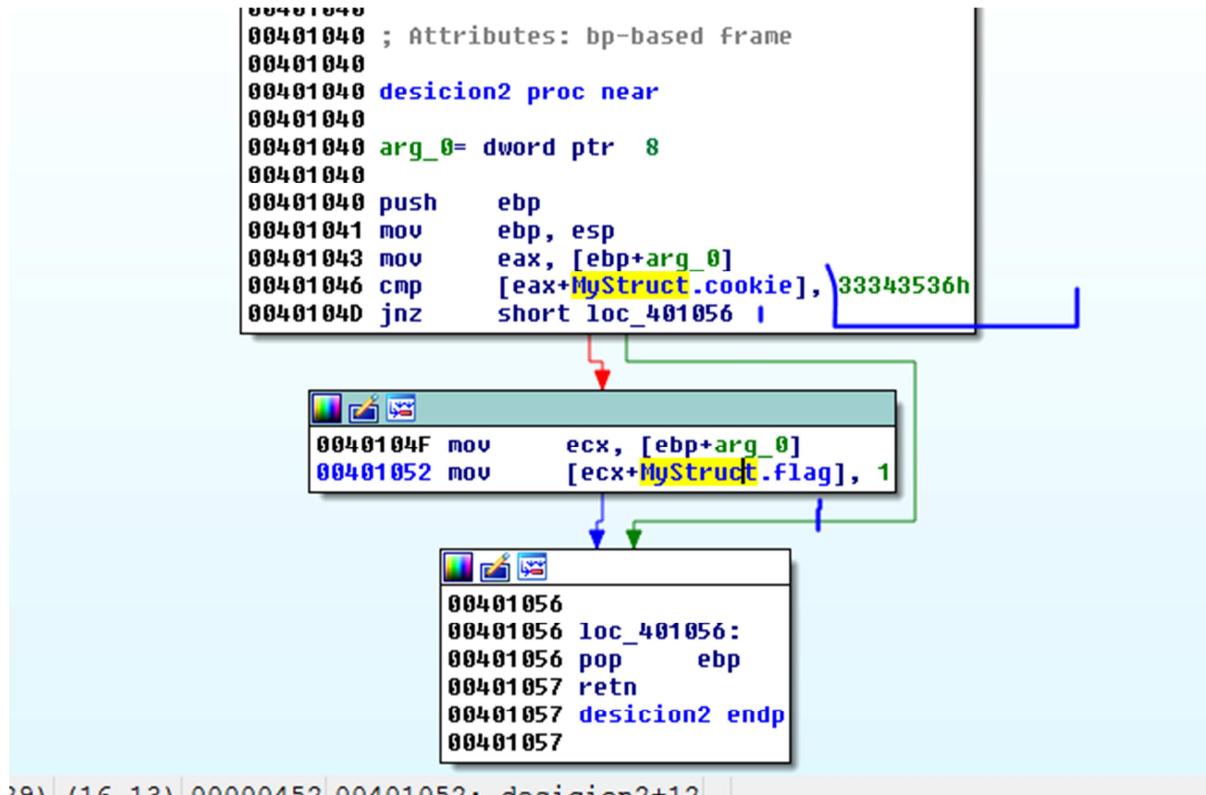
```

```

00401175 mov    [ebp+pepe.flag], 0
00401179 mov    [ebp+juan.cookie], 0
00401180 mov    [ebp+juan.numero], 0
00401187 mov    [ebp+juan.c], 0
0040118E mov    [ebp+juan.flag], 0
00401192 lea    eax, [ebp+pepe]
00401195 push   eax
00401196 call   enter
00401198 add    esp, 4
0040119E lea    ecx, [ebp+pepe]
004011A1 push   ecx
004011A2 call   ; _struct
004011A7 add    esp, 4
004011AA lea    edx, [ebp+pepe]
004011AD push   edx
004011AE call   ; _struct
004011B3 add    esp, 4
004011B6 lea    eax, [ebp+juan]
004011B9 push   eax
004011B9A call   ; _struct
004011BF add    esp, 4
004011C2 lea    ecx, [ebp+juan]
004011C5 push   ecx
004011C6 call   ; _struct
004011CB add    esp, 4
004011CE lea    edx, [ebp+juan]
004011D1 push   edx
004011D2 call   sub_401040
004011D7 add    esp, 4

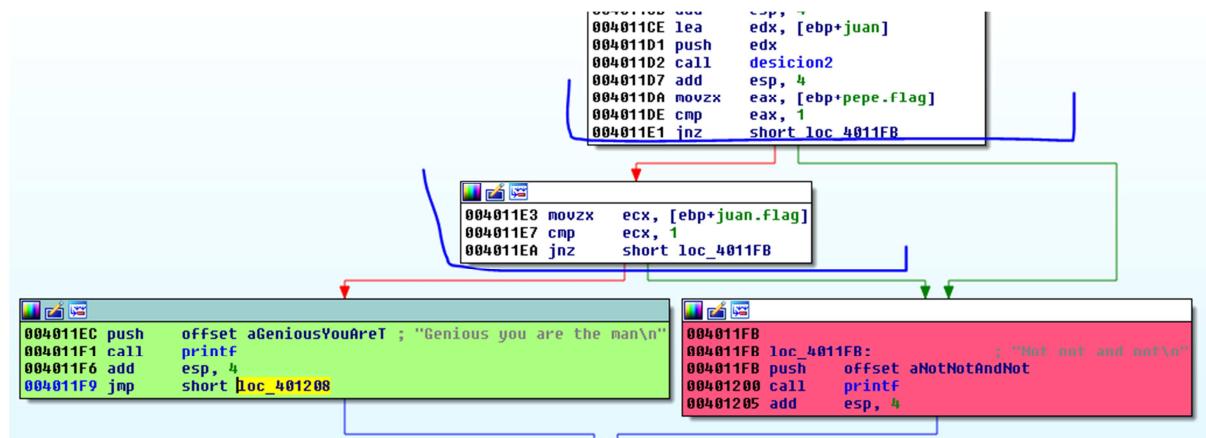
```

We see that with juan, it will do the same as it did with pepe on **enter** and **check**, but it has a third different function. Let's see what it does.



By pressing T in the fields, we see that it is similar to the "decision" function, except that the constant with which the cookie compares to juan is different, in this case, 0x33343536.

So the pepe cookie should be worth 0x99989796 and juan cookie should be worth 0x33343536 with that both flags of each structure will be 1.



We see that for this to arrive at the good boy both flags must be 1.

This exercise has many solutions because as the juan structure is higher in the stack...

```
-00000044 ; Use data definition commands to creat
-00000044 ; Two special fields " r" and " s" repr
-00000044 ; Frame size: 44; Saved regs: 4; Purge:
-00000044 ;
-00000044
-00000044 juan      MyStruct ?
-00000027      db ? ; undefined
-00000026      db ? ; undefined
-00000025      db ? ; undefined
-00000024 pepe      MyStruct ?
-00000007      db ? ; undefined
-00000006      db ? ; undefined
-00000005      db ? ; undefined
-00000004 var_4      dd ?
+00000000      s      db 4 dup(?)
+00000004      r      db 4 dup(?)
+00000008
+00000008 ; end of stack variables
```

When it does its gets_s we could overwrite all the flags so that they remain to 1 and that way, with a single overflow, to be able to pass the checks, also this can be done individually, overwriting in each gets_s the flag and it is not necessary to overwrite The cookie with the correct value, because we directly overwrote the flag without waiting to compare the cookie to change the same.

To overwrite the flag I have 16 decimal bytes, plus 3 dwds or 12,

This would be: $16 + 12 = 28$ bytes.

```
10000000 MyStruct      struc ; (sizeof=0x1D, mappedto_29) ; XREF: sub_401150/r
10000000
10000000 Buffer        db 16 dup(?) ; sub_401150/r
10000010 numero        dd ? ; XREF: check+6/r check+17/r
10000014 c              dd ? ; XREF: enter+34/w enter+40/r
10000018 cookie         dd ? ; XREF: desicion2+6/r
10000018
1000001C flag           db ? ; desicion+6/r
1000001C
1000001D MyStruct      ends ; desicion+12/w
1000001D
```

Like this:

fruta= 28* “A” + “\x01”

A screenshot of a Python IDE showing a script named 'pepe.py'. The code uses the subprocess module to run 'ConsoleApplication4.exe' in a pipe. It prints a message and reads input. It then writes '-1\n' to standard input, followed by 28 'A' characters, a byte value '\x01', and another '\n'. Finally, it reads the result from the process. A tooltip 'Platform and Plugin Updater' is visible in the bottom right.

```
1  from subprocess import *
2  import struct
3  p = Popen([r'ConsoleApplication4.exe', 'f'], stdout=PIPE, stdin=PIPE, stderr=STDOUT)
4
5  print "ATACHEA EL DEBUGGER Y APRETA ENTER\n"
6  raw_input()
7
8  primera="-1\n"
9  p.stdin.write(primer)
10
11 fruta= fruta= 28* "A" + "\x01" +"\n"
12 p.stdin.write(fruta)
13
14 primera="-1\n"
15 p.stdin.write(primer)
16
17 fruta= fruta= 28* "A" + "\x01" +"\n"
18 p.stdin.write(fruta)
19
20 testresult = p.communicate()[0]
21
```

Of course, to each of the structures you have to pass the -1 or the negative value that it passes the check against 0x10, and then the fruit.

A screenshot of a debugger interface showing the 'pepe.py' script and its output window. The script is identical to the one above. The output window shows two instances of the application asking for input, both receiving '-1\n'. The final output line is 'Genious you are the man', with a blue bracket highlighting it. The status bar at the bottom says 'Process finished with exit code 0'.

```
1  from subprocess import *
2  import struct
3  p = Popen([r'ConsoleApplication4.exe', 'f'], stdout=PIPE, stdin=PIPE, stderr=STDOUT)
4
5  print "ATACHEA EL DEBUGGER Y APRETA ENTER\n"
6  raw_input()
7
8  primera="-1\n"
9  p.stdin.write(primer)
10
11 fruta= fruta= 28* "A" + "\x01" +"\n"
12 p.stdin.write(fruta)
13
```

Please Enter Your Number of Choice:
Please Enter Your Number of Choice:
Genious you are the man
Process finished with exit code 0

If we debug a little, we see the -1 that enters the field number of pepe.

```

00CD1010 ; char *__cdecl check(MyStruct *_struct)
00CD1010 check proc near
00CD1010
00CD1010 _struct= dword ptr  8
00CD1010
00CD1010 push    ebp
00CD1011 mov     ebp, esp
00CD1013 mov     eax, [ebp+_struct]
00CD1016 cmp     [eax+MyStruct.numero], 10h
00CD101A jle     short loc_CD1024

```

[eax+MyStruct.numero]=[Stack[0000322C]:0039FD38]
dd 0xFFFFFFFF

572,128| 00000416| 00CD1016: check+6 (Synchronized with EIP)

```

00CD1024
00CD1024 loc_CD1024:
00CD1024 mov     ecx, [ebp+_struct]
00CD1027 mov     edx, [ecx+MyStruct.numero]
00CD102A push    edx          ; Size
00CD102B mov     eax, [ebp+_struct]
00CD102E push    eax          ; Buf
00CD102F call    ds:gets_s
00CD1035 add     esp, 8
00CD1038 pop     ebp
00CD1039 retn

```

[F 00CD102F: check+1F (Synchronized with EIP)

It passes the check and arrives at gets_s and there, it reads the fruit.

```

00CD1024 mov     ecx, [ebp+_struct]
00CD1027 mov     edx, [ecx+MyStruct.numero]
00CD102A push    edx          ; Size
00CD102B mov     eax, [ebp+_struct]
00CD102E push    eax          ; Buf
00CD102F call    ds:gets_s
00CD1035 add     esp, 8
00CD1038 pop     ebp

```

[ebp+_struct]=[Stack[0000322C]:0039FD04]
dd offset unk_89FD028

There, I see the address of pepe in my machine after going through the gets_s. If I go there...

```

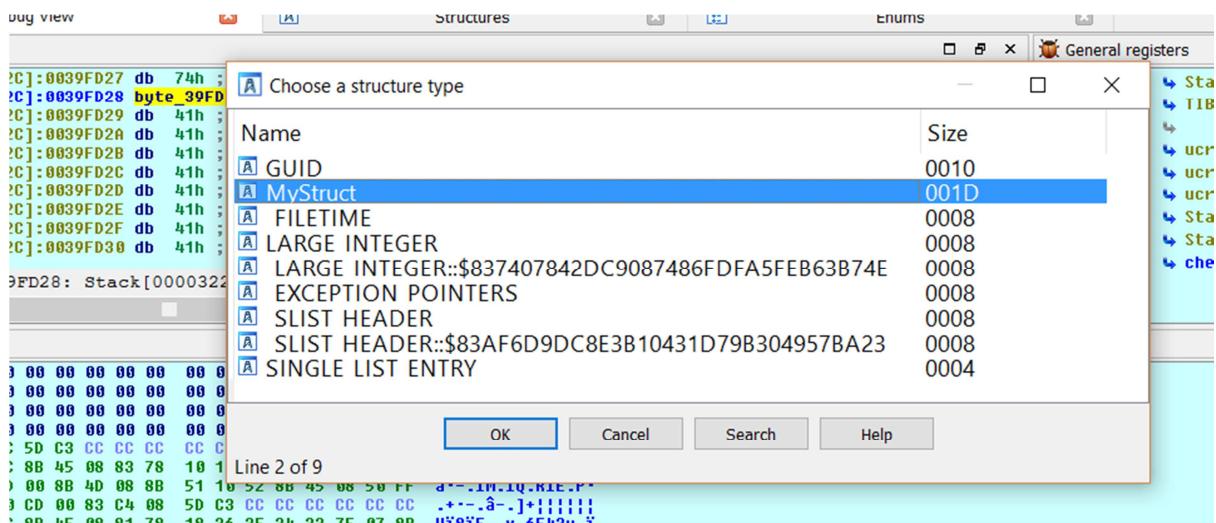
Stack[0000322C]:0039FD26 db 0C2h ; -
Stack[0000322C]:0039FD27 db 74h ; t
Stack[0000322C]:0039FD28 unk_39FD28 db 41h ; A
Stack[0000322C]:0039FD29 db 41h ; A
Stack[0000322C]:0039FD2A db 41h ; A
Stack[0000322C]:0039FD2B db 41h ; A
Stack[0000322C]:0039FD2C db 41h ; A
Stack[0000322C]:0039FD2D db 41h ; A
Stack[0000322C]:0039FD2E db 41h ; A
Stack[0000322C]:0039FD2F db 41h ; A
; DATA XREF: Stack[0000322C]:unk_39FD28

UNKNOWN 0039FD28: Stack[0000322C]:unk_39FD28 (Synchronized with EIP)

```

x View-1

There, I see the fruit that I sent it. If I want to turn it into a structure, I press ALT + Q.



I select MyStruct.

```

?FD27 db 74h ; t
?FD28 stru_39FD28 db 41h, 41h; Buffer
?FD28 ; DATA XREF: Stack[0000322C]:0039FD04To
?FD28 db 41h, 41h ; Buffer
?FD28 dd 41414141h ; numero
?FD28 dd 41414141h ; c
?FD28 dd 41414141h ; cookie
?FD28 db 1 ; flag
?FD45 db 0
?FD46 db 0C2h ,

```

I see the fields and that flag is already overwritten to 1 by the overflow, without going through the other functions, I continue tracing.

```

00CD1060 desicion proc near
00CD1060 _struct= dword ptr 8
00CD1060 push    ebp
00CD1061 mov     ebp, esp
00CD1063 mov     eax, [ebp+_struct]
00CD1066 cmp     [eax+MyStruct.cookie], 99989796h
00CD106D jnz     short loc_CD1076

```

[eax+MyStruct.cookie]=[Stack[0000322C]:stru_39FD28+18]

```

00466 0039FD28: Stack[0000322C]:st
db 41h, 41h ; Buffer
dd 41414141h ; numero
dd 41414141h ; c
dd 41414141h ; cookie
00 00 00 00 00 00 00 00 db 4 ; flag
00 00 00 00 00 00 00 00 db 8
00 00 00 00 00 00 00 00 db 0C2h ; -
00 00 00 00 00 00 00 00 db 74h ; t
CC CC CC CC CC CC CC UI8]+-----+
10 10 7E 08 6A 01 FF 15 UI8IE.ax...1...

```

We see that this compares cookie with 0x99989796 and since it is not the same, it does not modify the flag but it is already 1, so I do not care.

The same process will be repeated and we get to **decision2**.

```

00CD11C2 lea      ecx, [ebp+juan]
00CD11C5 push    ecx          ; _struct
00CD11C6 call    check
00CD11CB add    esp, 4
00CD11CE lea      edx, [ebp+juan]
00CD11D1 push    edx
00CD11D2 call    desicion2
00CD11D7 add    esp, 4
00CD11DA movzx  eax, [ebp+pepe.flag]
00CD11DE cmp     eax, 1
00CD11E1 jnz     short loc_CD11FB

```

) 000005D2 00CD11D2: sub_CD1150+82 (Synchronized with EIP)

I enter it with F7.

```

00CD1040
00CD1040 desicion2 proc near
00CD1040 arg_0= dword ptr 8
00CD1040 push    ebp
00CD1041 mov     ebp, esp
00CD1043 mov     eax, [ebp+arg_0]
00CD1046 cmp     [eax+MyStruct.cookie], 33343536h
00CD104D jnz     short loc_CD1056

```

178) 00000443 00CD1043: desicion2+3 (Synchronized with EIP)

General registers
EAX 010FFC6C Stack[00000000]
EBX 00EA7000 TIB[00000000]
ECX 00000000
EDX 010FFC6C Stack[00000000]
ESI 74CC6314 ucrtbase.dll!74CC6314
EDI 74CC6308 ucrtbase.dll!74CC6308
EBP 010FFC60 Stack[00000000]
ESP 010FFC60 Stack[00000000]
EIP 00CD1046 desicion2
EFL 00000206

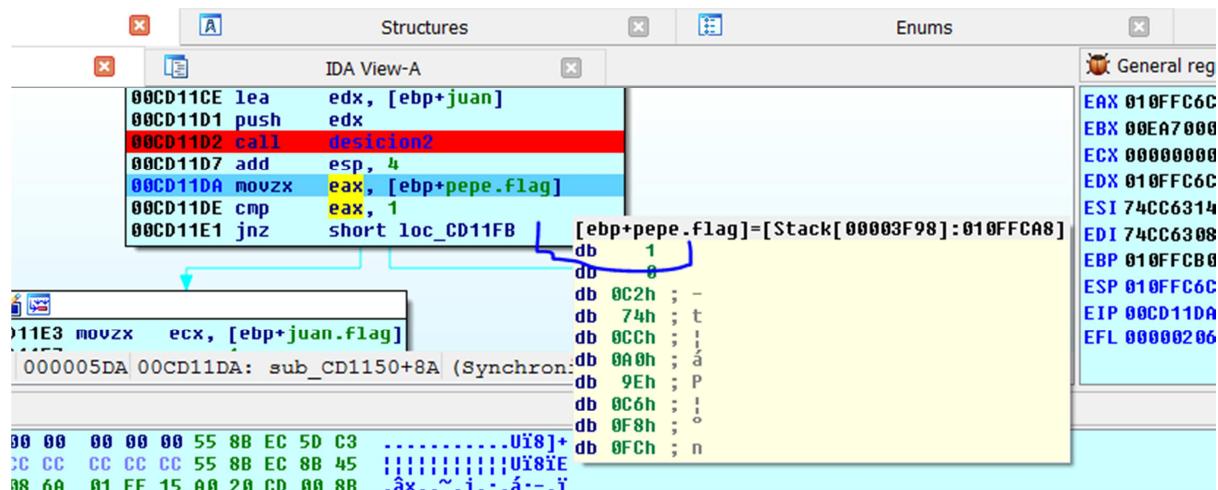
EAX has the beginning of juan structure. Let's see it.

```
ack[ 00003F98]:010FFC6A db 0Fh
ack[ 00003F98]:010FFC6B db 1
ack[ 00003F98]:010FFC6C db 41h ; A
ack[ 00003F98]:010FFC6D db 41h ; A
ack[ 00003F98]:010FFC6E db 41h ; A
ack[ 00003F98]:010FFC6F db 41h ; A
ack[ 00003F98]:010FFC70 db 41h ; A
ack[ 00003F98]:010FFC71 db 41h ; A
ack[ 00003F98]:010FFC72 db 41h ; A
ack[ 00003F98]:010FFC73 db 41h ; A
UNKNOWN 010FFC6C: Stack[00003F98]:010FFC6C
```

There, I also do ALT + Q to change it to structure and I choose MyStruct.

```
0003F98]:010FFC68 db 6Ch ; l  
0003F98]:010FFC69 db 0FCh ; n  
0003F98]:010FFC6A db 0Fh  
0003F98]:010FFC6B db 1  
0003F98]:010FFC6C db 41h, 41h  
0003F98]:010FFC6C db 41h, 41h ; Buffer  
0003F98]:010FFC6C dd 41414141h ; numero  
0003F98]:010FFC6C dd 41414141h ; c  
0003F98]:010FFC6C dd 41414141h ; cookie  
0003F98]:010FFC6D db 1 ; flag  
0003F98]:010FFC69 db 0
```

The same as before the flag will be 1. The comparison with the cookie will not be the same but it continued because the flag is already well.



pepe.flag is worth 1. It's OK. Let's continue.

Juan.flag is also correct so I get to the good boy msg.

The screenshot shows a debugger interface with two main panes. The top pane displays assembly code:

```
00CD11EC push    offset aGeniousYouAreT ; "Genious you are the man\n"
00CD11F1 call    printf
00CD11F6 add     esp, 4
00CD11F9 jmp     short loc_CD1208
```

A red arrow points from the assembly code area down to the memory dump pane below. The bottom pane shows memory dump data:

855	(233, 65)	000005EC	00CD11EC: sub_CD1150+9C (Synchronized with EIP)
-----	-----------	----------	---

And ready. This is already solved.

Ricardo Narvaja

Translated by: @IvinsonCLS