

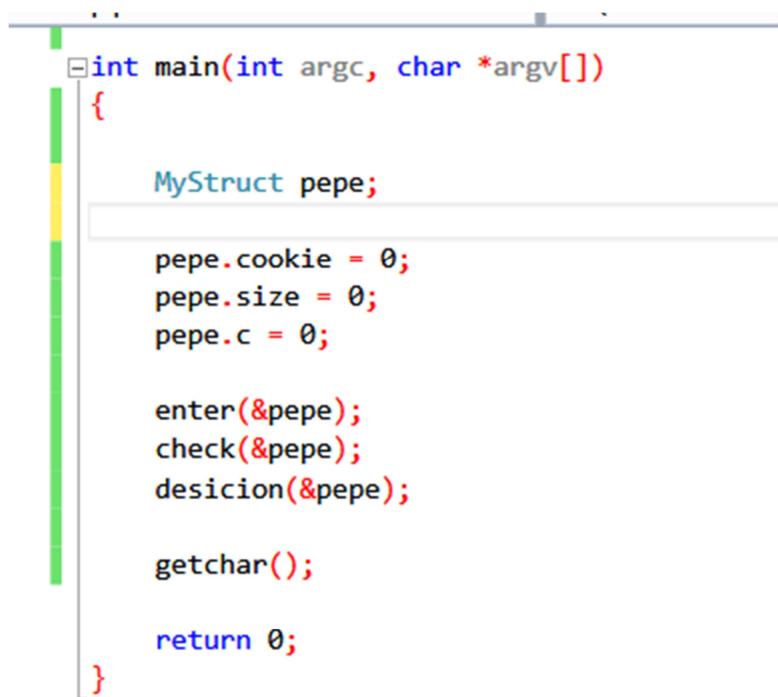
REVERSING WITH IDA PRO FROM SCRATCH

PART 26

The idea of programming, having most of what we need grouped within one or more structures, makes a lot of sense.

If I, for example, create a program and it uses many different types of data, some in certain or several parts of the program, are modified, are used, be passing as an argument between functions so many different values of different type, the work is Hard, while using structures we group them and always just passing the address where the structure begins, I can read or modify anywhere in the program, any field of it.

Let's see this example here we see only the main one of a program.



```
int main(int argc, char *argv[])
{
    MyStruct pepe;
    pepe.cookie = 0;
    pepe.size = 0;
    pepe.c = 0;

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

    getchar();

    return 0;
}
```

We see that, in this example, there is a `MyStruct` structure but in this case, instead of being declared locally in the stack it is declared as global which is also possible and gives me the possibility to handle it better between functions but it will be valid only In the main in this case, although it is not the only possibility, we will see later the heap and how structures and variables are handled there.

```
1 // ConsoleApplication4.cpp : Defines the entry point for the application
2 //
3
4 #include "stdafx.h"
5 #include <windows.h>
6
7 struct MyStruct
8 {
9     char buf[0x10];
10    int size;
11    int c;
12    int cookie;
13}
14
15
16 void check(MyStruct * _pepe) {
17     if (_pepe->size > 0x10) { exit(1); }
18     gets_s(_pepe->buf, _pepe->size);
19 }
20
21 void desicion(MyStruct * _pepe) {
22     if (_pepe->cookie == 0x45934215) {
```

We see that the definition of the structure is outside the main and any function and that makes it global, however the `pepe` variable that is of type `MyStruct` is local and is declared in the stack.

```
1 int main(int argc, char *argv[])
2 {
3     MyStruct pepe;
4
5     pepe.cookie = 0;
6     pepe.size = 0;
7     pepe.c = 0;
```

We understand the program's idea of passing a pointer to the `pepe` structure and thus being able to read or change values in it within the three functions: **enter**, **check** and **decision**.

There, I compile it with symbols, it does not give me much more info because being the global definition it is not easy for IDA to detect that the `pepe` variable is of structure type.

Hex View-1 Structures Enums Imports Exports

```

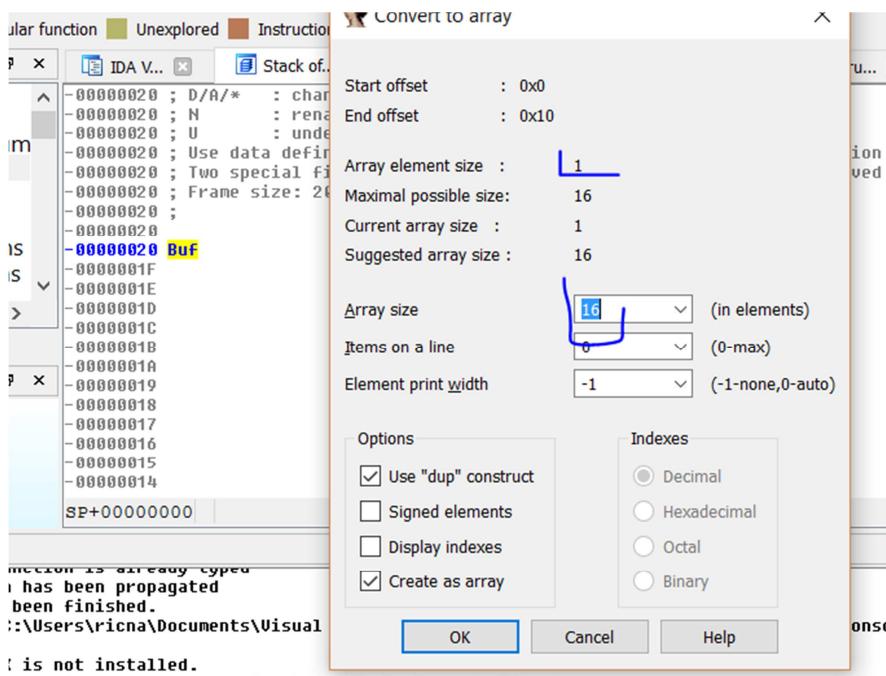
00401143 sub    esp, 20h
00401146 mov    eax, __security_cookie
00401148 xor    eax, ebp
0040114D mov    [ebp+var_4], eax
00401150 mov    [ebp+var_8], 0
00401157 mov    [ebp+var_10], 0
0040115E mov    [ebp+var_C], 0
00401165 lea    eax, [ebp+Buf]
00401168 push   eax
00401169 call   enter ←
0040116E add    esp, 4
00401171 lea    ecx, [ebp+Buf]
00401174 push   ecx, [ebp+Buf]; Buf
00401175 call   check ←
0040117A add    esp, 4
0040117D lea    edx, [ebp+Buf]
00401180 push   edx
00401181 call   desicion ←
00401186 add    esp, 4
00401189 call   ds:_imp_getchar
0040118F xor    eax, eax
00401191 mov    ecx, [ebp+var_4]
00401194 xor    ecx, ebp
00401196 call   __security_check_cookie
0040119B mov    esp, ebp
0040119D pop    ebp
0040119E retn
0040119F main endp
004011D0

```

75,44 00000569 00401169: main+29 (Synchronized with Hex View-1)

We see that in the main there is a BUFFER.

Let's look at the length of this buffer according to IDA.



We see that the length of the buffer is 16 per 1 of each element or 16 decimal, so far if we do not know the code, this would be another buffer and I have no idea what a field structure is.

I accept the length.

```
-00000020 ; Use data definition commands to create local vari;
-00000020 ; Two special fields "r" and "s" represent return
-00000020 ; Frame size: 20; Saved regs: 4; Purge: 0
-00000020 ;
-00000020
-00000020 Buf db 16 dup(?)
-00000010 var_10 dd ?
-0000000C var_C dd ?
-00000008 var_8 dd ?
-00000004 var_4 dd ?
+00000000 s db 4 dup(?)
+00000004 r db 4 dup(?)
+00000008 argc dd ?
+0000000C argv dd ? ; offset
+00000010 envp dd ? ; offset
+00000014
+00000014 ; end of stack variables

SP+00000000
```

We see the other local variables if we press X to see where they are used.

```
00401140 argc= dword ptr 8
00401140 argv= dword ptr 0Ch
00401140 envp= dword ptr 10h
00401140
00401140 push ebp
00401141 mov esp, ebp
00401143 sub esp, 20h
00401146 mov eax, __security_cookie
0040114B xor eax, ebp
0040114D mov [ebp+var_4], eax
00401150 mov [ebp+var_8], 0
00401157 mov [ebp+var_10], 0
0040115E mov [ebp+var_C], 0
00401165 lea eax, [ehn+RUE]
```

xrefs to var_10

Direction	Type	Address	Text
w		main+17	mov [ebp+var_10], 0

OK Cancel Search Help

We see that all are initialized to zero and are not reused, which is suspect in a local variable, so that it is created and initialized if not used, here some alarms start to light up.

And then there is the CANARY.

```
00401140 BuF= byte ptr -20h
00401140 var_10= dword ptr -10h
00401140 var_C= dword ptr -0Ch
00401140 var_8= dword ptr -8
00401140 CANARY= dword ptr -4
00401140 argc= dword ptr 8
00401140 argv= dword ptr 0Ch
00401140 envp= dword ptr 10h
00401140
00401140 push    ebp
00401141 mov     ebp, esp
00401143 sub     esp, 20h
00401146 mov     eax, __security_cookie
00401148 xor     eax, ebp
0040114D mov     [ebp+CANARY], eax
00401150 mov     [ebp+var_8], 0
00401157 mov     [ebp+var_10], 0
0040115E mov     [ebp+var_C], 0
00401165 lea     eax, [ebp+Buf]
00401168 push    eax
00401169 call    enter
0040116E add    esp, 4
```

There is no more variable, we cannot rename the three local variables that are initialized to zero because they are not used within the function. It does not make sense to name them. I do not know what name I would put it if I do not see anything special in them.

Let's go to the first function.

```
00401158 mov     [ebp+var_8], 0
00401157 mov     [ebp+var_10], 0
0040115E mov     [ebp+var_C], 0
00401165 lea     eax, [ebp+Buf]
00401168 push    eax
00401169 call    enter
0040116E add    esp, 4
```

We see that it finds the address of the buffer and passes it as an argument, which is perfectly possible, being able to fill the buffer inside one of these functions.

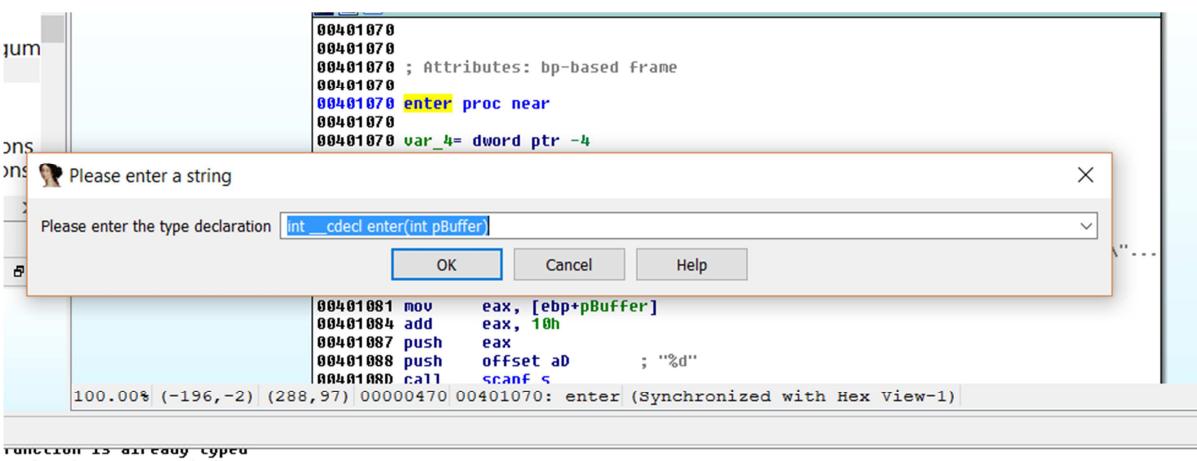
Let's enter **check**.

```
00401070
00401070 ; Attributes: bp-based frame
00401070 enter proc near
00401070 var_4= dword ptr -4
00401070 pBuffer= dword ptr 8
00401070
00401070 push    ebp
00401071 mov     ebp, esp
00401073 push    ecx
00401074 push    offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice: \..."
00401079 call    printf
0040107E add     esp, 4
00401081 mov     eax, [ebp+pBuffer]
00401084 add     eax, 10h
00401087 push    eax
00401088 push    offset aD          ; "%d"
0040108D call    scanf
```

2) (138,39) 00000470 00401070: enter (Synchronized with Hex View-1)

I rename it as **pBuffer** since it is a pointer or address, because I use LEA and I do not pass the value but the address where the buffer is.

If I do **Set Type** or I press **Y** to propagate.



```
00401070
00401070 ; Attributes: bp-based frame
00401070 enter proc near
00401070 ; int __cdecl enter(int pBuffer)
00401070 var_4= dword ptr -4
00401070 pBuffer= dword ptr 8
00401070
00401070 push    ebp
00401071 mov     ebp, esp
00401073 push    ecx
00401074 push    offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice: \..."
00401079 call    printf
0040107E add     esp, 4
00401081 mov     eax, [ebp+pBuffer]
```

Let's see if in reference it was good.

```

00401150 mov    [ebp+var_8], 0
00401157 mov    [ebp+var_10], 0
0040115E mov    [ebp+var_C], 0
00401165 lea    eax, [ebp+Buf]
00401168 push   eax
00401169 call   enter
0040116F add    esp, 4

```

We see that this is correct since EAX has the Buffer address that was obtained with LEA.

```

00401070
00401078 var_4= dword ptr -4
00401078 pBuffer= dword ptr 8
00401078
00401078 push    ebp
00401071 mov     ebp, esp
00401073 push    ecx
00401074 push    offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice: \\"...
00401079 call    printf
0040107E add    esp, 4
00401081 mov    eax, [ebp+pBuffer]
00401084 add    eax, 10h
00401087 push   eax
00401088 push   offset aD      ; "%d"
0040108D call   scanf_s
00401092 add    esp, 8

```

Here, another alarm sounds if the buffer is 16 in length or 0x10, we see that the Buffer address adds 0x10 and it is not writing in it but then just below.

This already makes me think, if you pass a pointer to the buffer, then you are typing out, it is the typical behavior of the structures, you pass the initial address to a function and then you can access any field, in this case something that is below a first BUFFER field.

And there was just below BUFFER in the main stack.

```

-00000020 ; Two special fields " r" and " s" represent return address
-00000020 ; Frame size: 20; Saved regs: 4; Purge: 0
-00000020 ;
-00000020
-00000020 Buf           db 16 dup(?)
-00000010 var_10        dd ?
-00000008 var_c          dd ?
-00000008 var_8          dd ?
-00000004 CANARY         dd ?
+00000000 s              db 4 dup(?)
+00000004 r              db 4 dup(?)
+00000008 argc           dd ?
+0000000C argv           dd ?                                ; offset
+00000010 envp           dd ?                                ; offset
+00000014
+00000014 ; end of stack variables

SP+00000010

```

We see that it is writing in var_10, so that can only happen if at least as much Buffer as var_10 are fields of a structure.

Many people wonder why I have to look at the main stack instead of the check function?

The point is that when passing the Buffer address, that address where it starts and the same Buffer are in the main, and if I add 0x10 to that address, I will reach the var_10 of the main, which as a local variable would not make sense inside **check**, but it does make sense if it belongs to a structure.

Well, for now, we see that...

It asks me for a number and saves it in that field of the structure called var_10, using scanf_s, so rename var_10 to number, but before as I know there is a structure and the three fields are surely accessed in the functions, I will create it.

```

-00000020 ; N      : rename
-00000020 ; U      : undefine
-00000020 ; Use data definition commands to create local variables and function
-00000020 ; Two special fields "r" and "s" represent return address and saved
-00000020 ; Frame size: 20; Saved regs: 4; Purge: 0
-00000020 ;
-00000020
arqum
ptions
ptions >
□ □ X
[...]
SP+00000018

```

I go back to the main and select until the CANARY and I go to EDIT-CREATE STRUCTURE FROM SELECTION.

```

-00000020 ; Two special fields "r" and "s" represent return :
-00000020 ; Frame size: 20; Saved regs: 4; Purge: 0
-00000020 ;
-00000020
-00000020 Buf      struct_0 ?
-00000004 CANARY   dd ?
+00000000 s        db 4 dup(?)
+00000004 r        db 4 dup(?)
+00000008 argc    dd ?
+0000000C argv    dd ?          ; offset
+00000010 envp    dd ?          ; offset
+00000014
+00000014 ; end of stack variables

```

There, I create the structure but I call it Buf, I will change the name to pepe.

```

00000000 ; [ 00000008 BYTES. COLLAPSED STRUCT _EXCEPTION_POINTERS. PRESS CTRL-NUMPAD+ ]
00000000 ;
00000000
00000000 struct_0      struc ; (sizeof=0x1C, mappedto_35) ; XREF: main/r
00000000 Buf       db 16 dup(?)
00000010 var_10    dd ?
00000014 var_C     dd ?
00000018 var_8     dd ?
0000001C struct_0  ends
0000001C

```

I will rename the structure as MyStruct.

```
00000000 ; [00000004 BYTES. COLLAPSED STRUCT SINGLE_LIST_ENTRY. PRESS CTRL-NUMPAD+ TO
00000000 ; [00000008 BYTES. COLLAPSED STRUCT _EXCEPTION_POINTERS. PRESS CTRL-NUMPAD+ 1
00000000 ;
00000000
00000000 MyStruct      struc ; (sizeof=0x1C, mappedto_35) ; XREF: main/r
00000000 Buf           db 16 dup(?)
00000010 numero        dd ?
00000014 var_C          dd ?
00000018 var_8          dd ?
0000001C MyStruct      ends
0000001C
```

Also, I rename **numero** or **number** in English.

```
00401141 mov    ebp, esp
00401143 sub    esp, 20h
00401146 mov    eax, __security_cookie
0040114B xor    eax, ebp
0040114D mov    [ebp+CANARY], eax
00401150 mov    [ebp+pepe.var_8], 0
00401157 mov    [ebp+pepe.numero], 0
0040115E mov    [ebp+pepe.var_C], 0
00401165 lea    eax, [ebp+pepe]
00401168 push   eax          ; pBuffer
00401169 call   enter
0040116E add    esp, 4
00401171 lea    ecx, [ebp+pepe]
00401174 push   ecx          ; Buf
00401175 call   check
0040117A add    esp, 4
0040117D lea    edx, [ebp+pepe]
00401180 push   edx
00401181 call   desicion
00401186 add    esp, 4
00401189 call   ds:_imp_getchar
.20) 00000546 00401146: main+6 (Synchronized with Hex View-
```

And in the main where everything is declared, I see that everything is fine.

Let's go back to the enter function.

```
00401070
00401070 ; Attributes: bp-based frame
00401070
00401070 ; int __cdecl enter(int pBuffer)
00401070 enter proc near
00401070
00401070 var_4= dword ptr -4
00401070 ppepe= dword ptr  8
00401070
00401070 push    ebp
00401071 mov     ebp, esp
00401073 push    ecx
00401074 push    offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice: \\"...
00401079 call    printf
0040107E add    esp, 4
00401081 mov    eax, [ebp+ppepe]
00401084 add    eax, 10h
00401087 push   eax
```

I change the name to **ppepe** for being a pointer to **pepe** and press "Y". I delete the statement, accept it and press "Y" again for the new declaration.

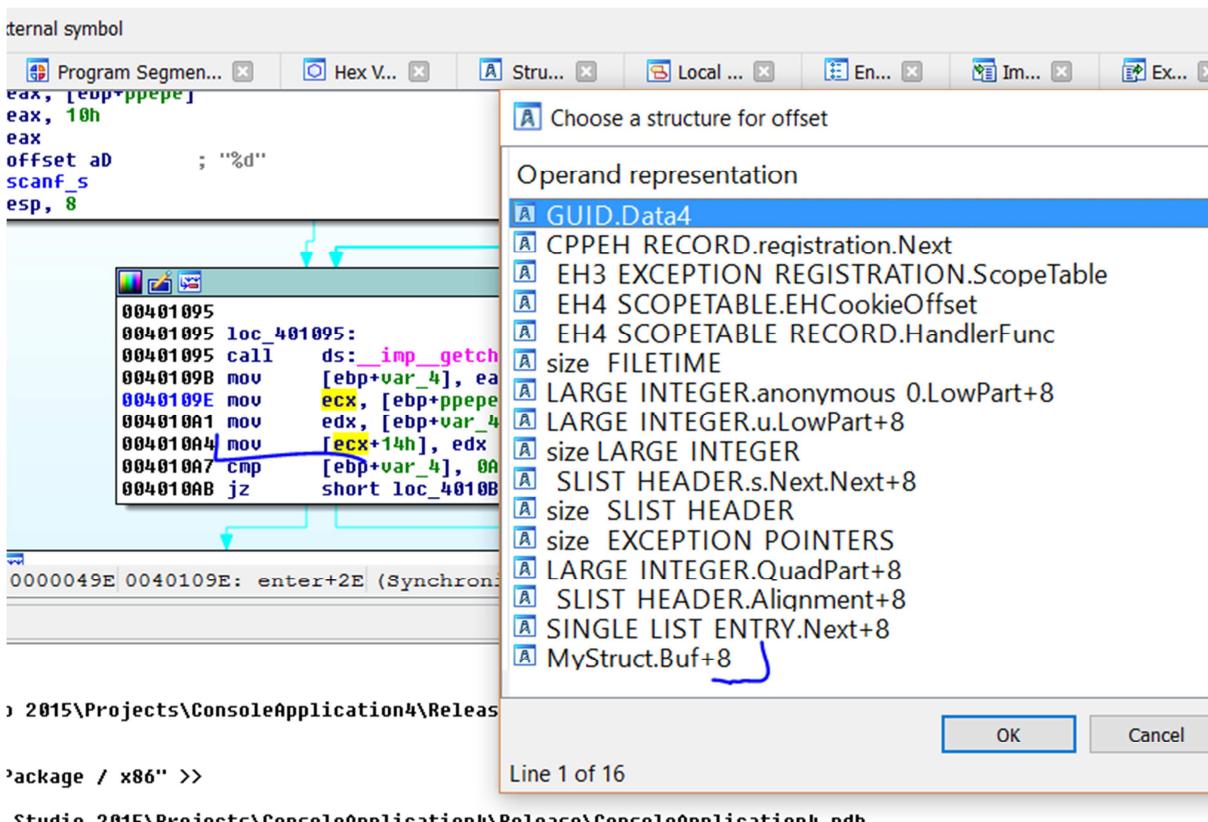
```
00401070 ; Attributes: bp-based Frame
00401070 ; int __cdecl enter(int ppepe)
00401070 enter proc near
00401070 var_4= dword ptr -4
00401070 ppepe= dword ptr 8
00401070
00401070 push    ebp
00401071 mov     ebp, esp
00401073 push    ecx
00401074 push    offset aPleaseEnterYou ; "\nPlease Enter Your Number of Choice: \\"...
00401079 call    printf
0040107E add     esp, 4
00401081 mov     eax, [ebp+ppepe]
00401084 add     eax, 10h
00401087 push    eax
```

Let's continue reversing.

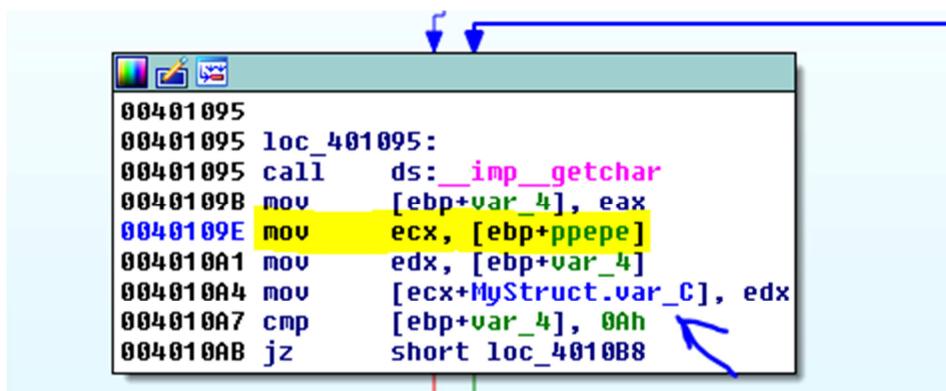
```
00401095
00401095 loc_401095:
00401095 call    ds:_imp_getchar
00401098 mov     [ebp+var_4], eax
0040109E mov     ecx, [ebp+ppepe]
004010A1 mov     edx, [ebp+var_4]
004010A4 mov     [ecx+14h], edx
004010A7 cmp     [ebp+var_4], 0Ah
004010AB jz      short loc_4010B8
```

(1) 0000049E 0040109E: enter+2E (Synchronized with Hex View-1)

Here, it is using a field of the structure since it moves the initial direction to ECX and soon adds 0x14 to him to save it in a field of the same. Here, we do not need to do accounts. I press "T" on that instruction.



I choose to what structure it corresponds and I see in the list that MyStruct is there.



We see that only by detecting that it is a field of the structure, pressing T and choosing the correct one, IDA accommodates its field name, obviously there are many structures and IDA as it is not defined here, it cannot know which structure it belongs to but we indicate it and it puts us the right field.

We see that it is the loop that is placed to filter the 0a after a scanf is an auxiliary field in the code. It is called c but it can have any letter or name.

```
00401095
00401095 loc_401095:
00401095 call ds:_imp_getchar
00401098 mov [ebp+temp], eax
0040109E mov ecx, [ebp+ppepe]
004010A1 mov edx, [ebp+temp]
004010A4 mov [ecx+MyStruct.c], edx
004010A7 cmp [ebp+temp], 0Ah
004010AB jz short loc_4010B8
```

Each time the pointer is passed to the structure to a record and then an offset is added.

```
004010AD mov eax, [ebp+ppepe]
004010B0 cmp dword ptr [eax+14h], 0FFFFFFFh
004010B4 jz short loc_4010B8
```

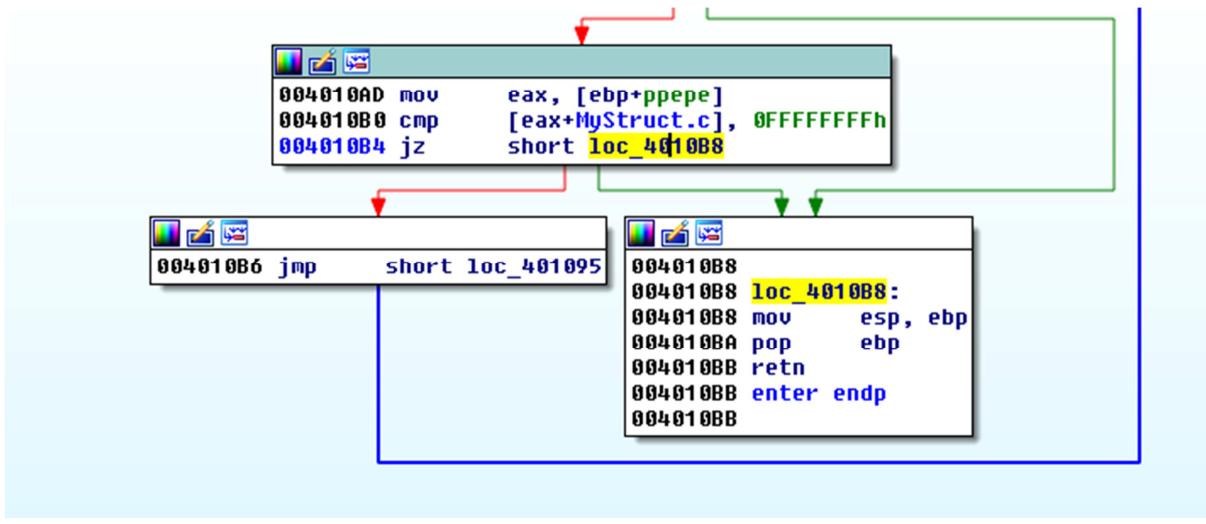
This will be accessing another field. I press T there.

```
004010A4 mov [ecx+MyStruct.c], edx
004010A7 cmp [ebp+temp], 0Ah
004010AB jz short loc_4010B8
```



```
004010AD mov eax, [ebp+ppepe]
004010B0 cmp [eax+MyStruct.c], 0FFFFFFFh
004010B4 jz short loc_4010B8
```

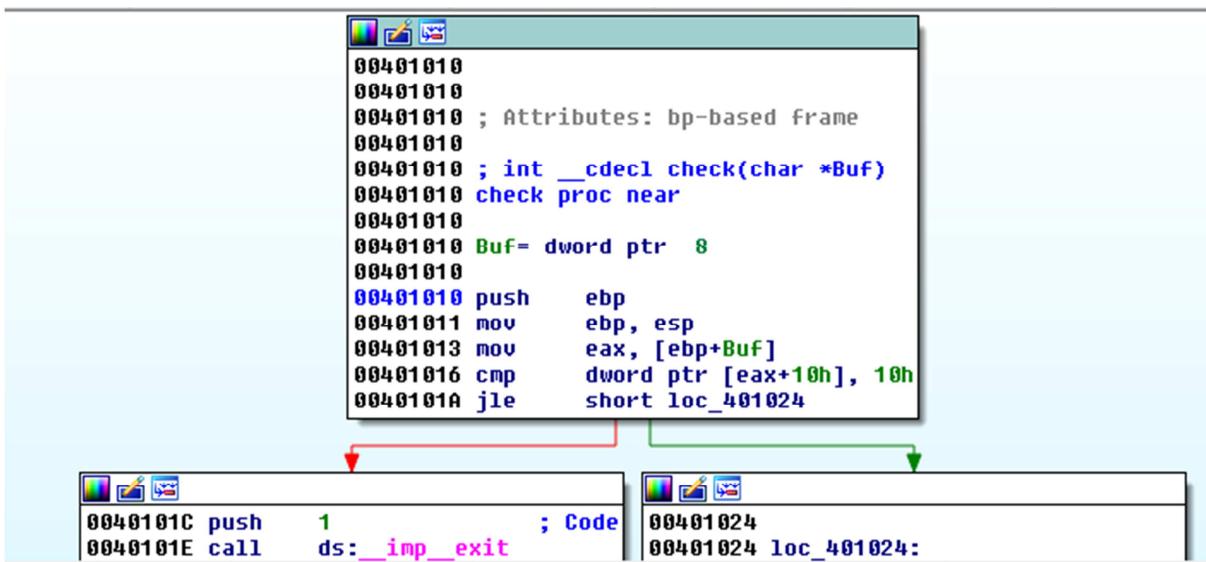
It's the end of that checkup. I go ahead.



It does nothing else and does not return anything in EAX.

So the first function was just to read the number we typed and save it in the field number.

The next check function.



You have to rename Buff to ppepe.

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

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

0,-2) (171,16) 00000410 00401010: check (Synchronized with Hex View-1)

I press Y for Set Type.

There, I see that it compares a field with 0x10. I press T on that instruction.

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

This compares the number entered with 0x10 considering the sign. This is dangerous if it is used as size later.

We see that when we manage as a structure, the field reads it in a function, it checks it in another function and then it will use it possibly in a third and if we follow the pointer to the structure we can always determine which field it is without having to debug, doing so as loose variables it is complicated to determine that it is always the same value.

```
[eax+MyStruct.numero], 10h  
short loc_401024  
00401024 loc_401024:  
00401024 mov    ecx, [ebp+ppepe]  
00401027 mov    edx, [ecx+10h]  
0040102A push   edx      ; Size  
0040102B mov    eax, [ebp+ppepe]  
0040102E push   eax      ; Buf  
0040102F call   ds:_imp_gets_s  
00401035 add    esp, 8  
00401038 pop    ebp  
00401039 retn  
00401039 check endp  
13 00401013: check+3 (Synchronized with Hex View-1)
```

There, there is another field. I press T.

```
eax, [ebp+ppepe]  
[eax+MyStruct.numero], 10h  
short loc_401024  
00401024 loc_401024:  
00401024 mov    ecx, [ebp+ppepe]  
00401027 mov    edx, [ecx+MyStruct.numero]  
0040102A push   edx      ; Size  
0040102B mov    eax, [ebp+ppepe]  
0040102E push   eax      ; Buf  
0040102F call   ds:_imp_gets_s  
00401035 add    esp, 8  
00401038 pop    ebp  
00401039 retn  
00401039 check endp  
2A 0040102A: check+1A (Synchronized with Hex View-)
```

I see that the field number is used as a size of a **gets_s** but that number could be 0xffffffff because the previous check was signed and in that case 0xFFFFFFFF is -1 and it is less than 0x10 and it passes perfectly.

We see that **gets_s** is passed the **ppepe** address, as the first field is the Buffer, it will write there.

We already see that there will be overflow.

```

00401040 desicion proc near
00401040 arg_0= dword ptr 8
00401040 push    ebp
00401041 mov     ebp, esp
00401043 mov     eax, [ebp+arg_0]
00401046 cmp     dword ptr [eax+18h], 45934215h
0040104D jnz     short loc_40105C

0040104F push    offset aYouAreAWinnerM ; "You are a winner man"
00401054 call    printf
00401059 add    esp, 4

0000440 00401040: desicion (Synchronized with Hex View-1)

```

The last function is also passed to pepe. Let's rename and fix everything.

```

00401040
00401040 ; Attributes: bp-based frame
00401040 ; int __cdecl desicion(int ppepe)
00401040 desicion proc near
00401040 ppepe= dword ptr 8
00401040 push    ebp
00401041 mov     ebp, esp
00401043 mov     eax, [ebp+ppepe]
00401046 cmp     [eax+MyStruct.cookie], 45934215h
0040104D jnz     short loc_401066

AreAWinnerM ; "You are a winner man"
xit : Code
00401064 jmp     short loc_401073
00401066 loc_401066:           ; "You are a loser"
00401066 push    offset aYouAreALoserMa
00401068 call    printf
00401070 add    esp, 4

100.00% (222,12) | (67,46) | 00000440 00401040: desicion (Synchronized with Hex View-1)

```

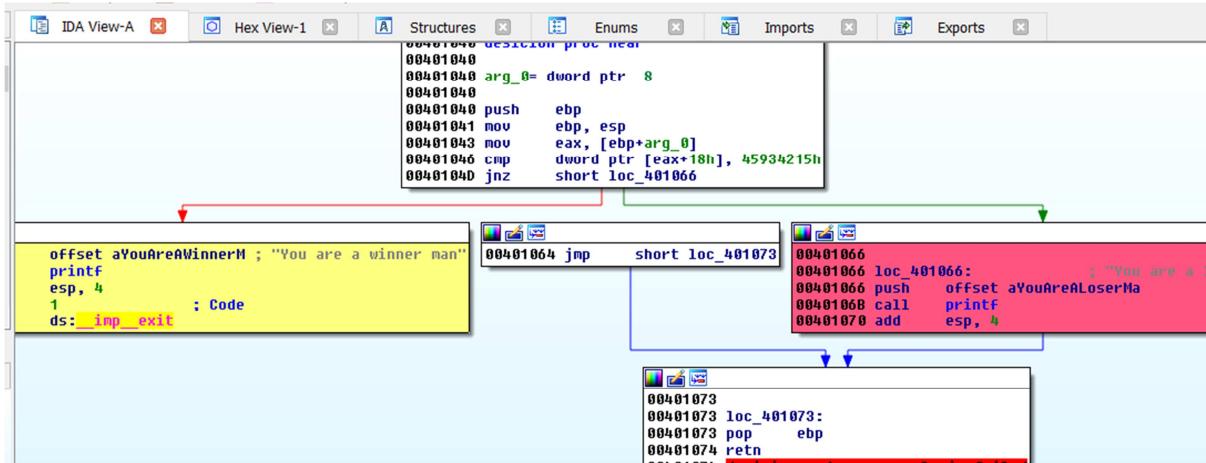
There, it reads the address of the structure in EAX. Then there is another field, when it adds an offset.

```

Hex View-1 Structures Enums Imports Exp
00401040 desicion_proc_near
00401040 arg_0= dword ptr 8
00401040 push    ebp
00401041 mov     ebp, esp
00401043 mov     eax, [ebp+arg_0]
00401046 cmp     dword ptr [eax+18h], 45934215h
0040104D jnz     short loc_401066

```

That var_8 is the variable that checks. He never used it or used it any more. I rename it as a cookie but if I do not know the name any will do.



We see that the decision is made here if cookie is equal to 0x45934215, it will tell us that we win if not goodbye. So, we already know what we must pass. Let's look at the distribution of the main stack.

```

000020 ; Two special fields " r" and " s" represent return ad
000020 ; Frame size: 20; Saved regs: 4; Purge: 0
000020 ;
000020
000020 pepe      MyStruct ?
000004 CANARY    dd ?
000000 s          db 4 dup(?)
000004 r          db 4 dup(?)
000008 argc       dd ?
00000C argv       dd ?           ; offset
000010 envp       dd ?           ; offset
000014
000014 ; end of stack variables

```

Obviously, everything is inside pepe. The buffer and the cookie. Let's go to structures to see the sizes of each.

```

00000000 MyStruct   struc ; (sizeof=0x1C, mappedto_35) ; XREF: desicion+6/o
00000000               ; enter+34/o ...
00000000 Buf        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: desicion+6/r
0000001C MyStruct   ends
0000001C

```

I have to fill the buffer with 16 Aes then two more dwords and then this cookie would be something like this.

```
fruta= "A" * 16 + numero + c + cookie
```

The script is similar to the previous ones.

```
from subprocess import *
import struct
p      = Popen([r'C:\Users\ricna\Documents\Visual Studio 2015\Projects\ConsoleApplication4\Release\ConsoleApplication4.exe', 'f'], stdout=PIPE, stdin=PIPE, stderr=STDOUT)

print "ATACHEA EL DEBUGGER Y APRETA ENTER\n"
raw_input()

primera="-1\n"
p.stdin.write(primera)

numero=struct.pack("<L",0x1c)
c=struct.pack("<L",0x90909090)
cookie=struct.pack("<L",0x45934215)

fruta= "A" * 16 + numero + c + cookie + "\n"
p.stdin.write(fruta)

testresult = p.communicate()[0]

print(testresult)
```

We see that it passes -1 as number to pass the check when it compares **signed** to 0x10 and then the fruit of 16 bytes to fill the buffer, then the number to which I pass a correct value of 0x1C to overflow it. If not, I will change it, then c which can be any value, then the 0x45934215 cookie.

The screenshot shows the PyCharm IDE interface. On the left is the project tree with a single file 'pepe.py'. The main editor window contains the following Python code:

```
from subprocess import *
import struct
p = Popen([r'C:\Users\ricna\Documents\Visual Studio 2015\Projects\Console1\Console1\Debug\pepe'])
print "ATACHEA EL DEBUGGER Y APRETA ENTER\n"
raw_input()
primera="-1\n"
p.stdin.write(primera)

numero=struct.pack("<L",0xic)
#struct.pack("L",0xffffffff)
```

Below the code, the terminal output shows:

```
Please Enter Your Number of Choice:
You are a winner man
Process finished with exit code 0
```

A message box in the bottom right corner says "Platform and I" and "PyCharm Community".

Well, with this we end part 26

I attached an exercise called IDA_STRUCT.7z see if it is vulnerable and what can be done. ☺

Ricardo Narvaja

Translated by: @IvinsonCLS