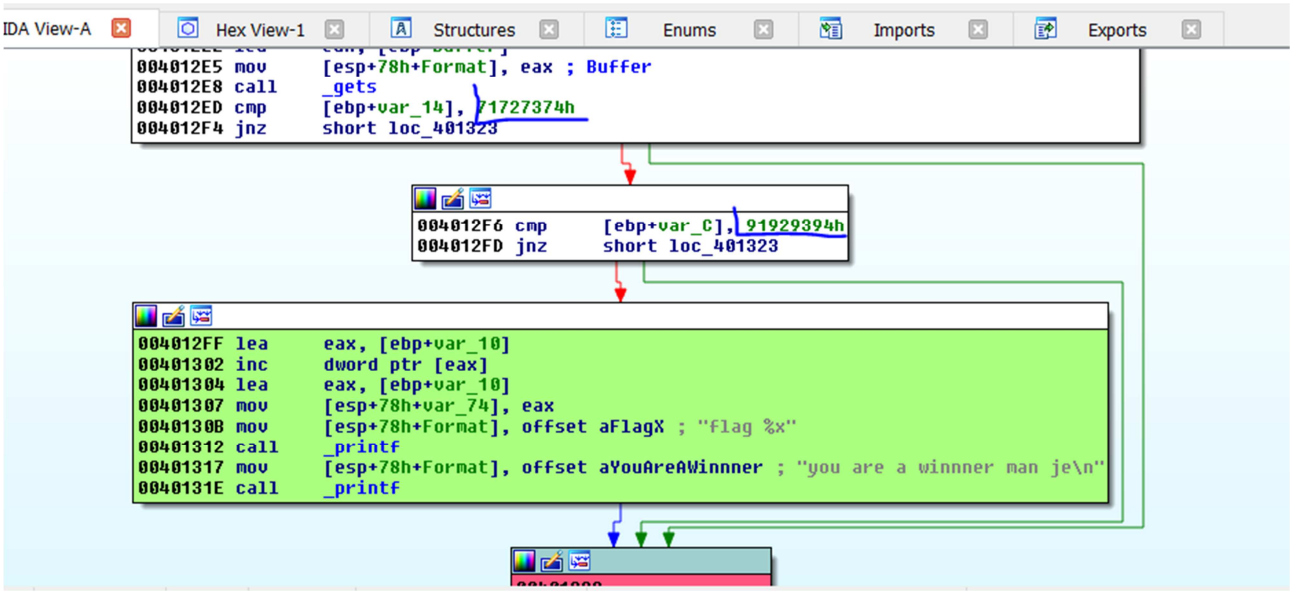
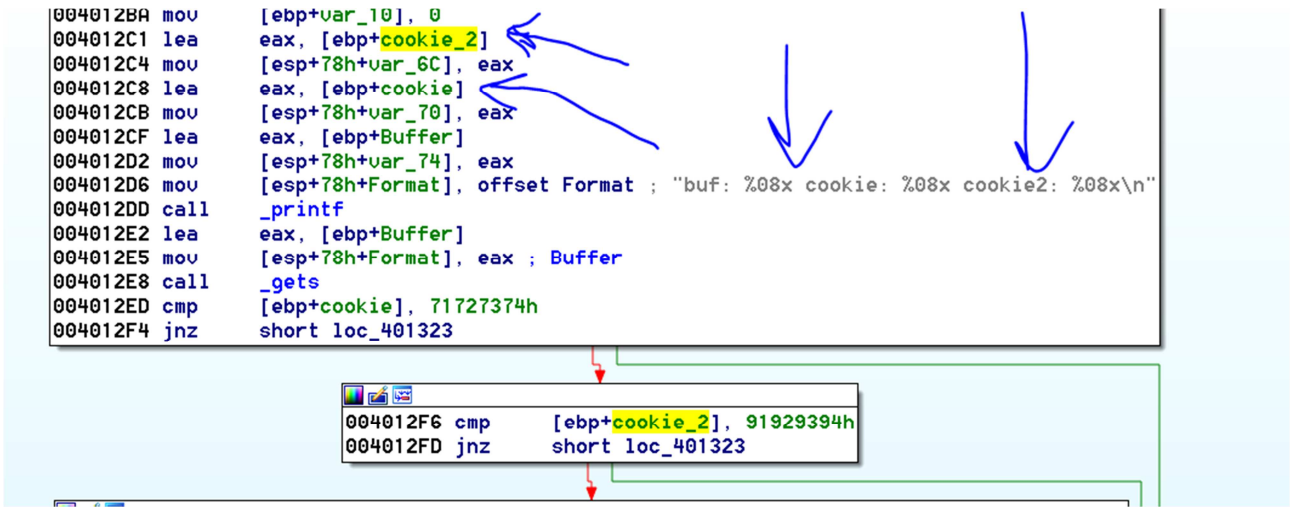
PART 24

The solution to IDA2.exe is quite similar to the previous one only that here are two variables or we can call them cookies, to compare instead of one to take you to the good boy zone.

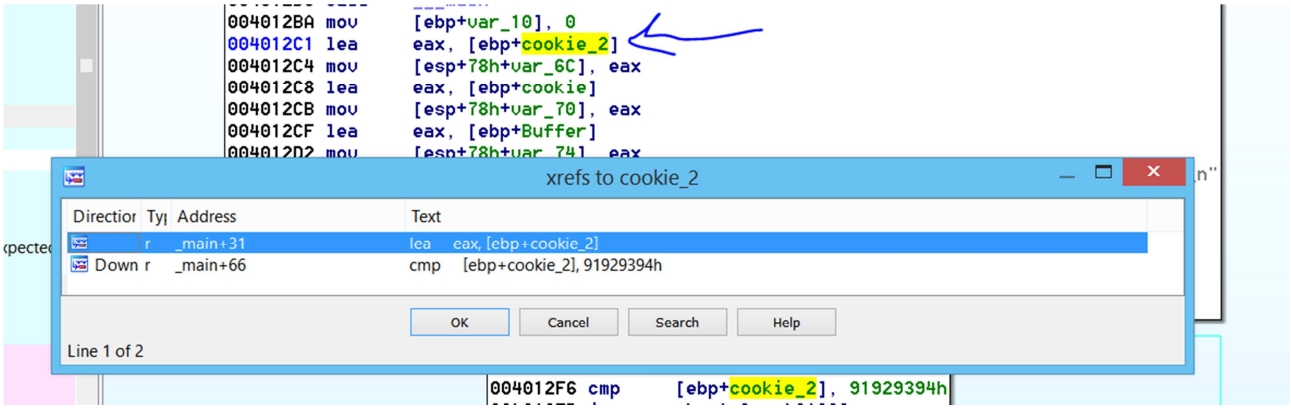


Let's see if there is any place where you can change those variables. We rename them to cookie and cookie 2 since the same program calls them that way.



We see that the only places where there is access to these variables is when

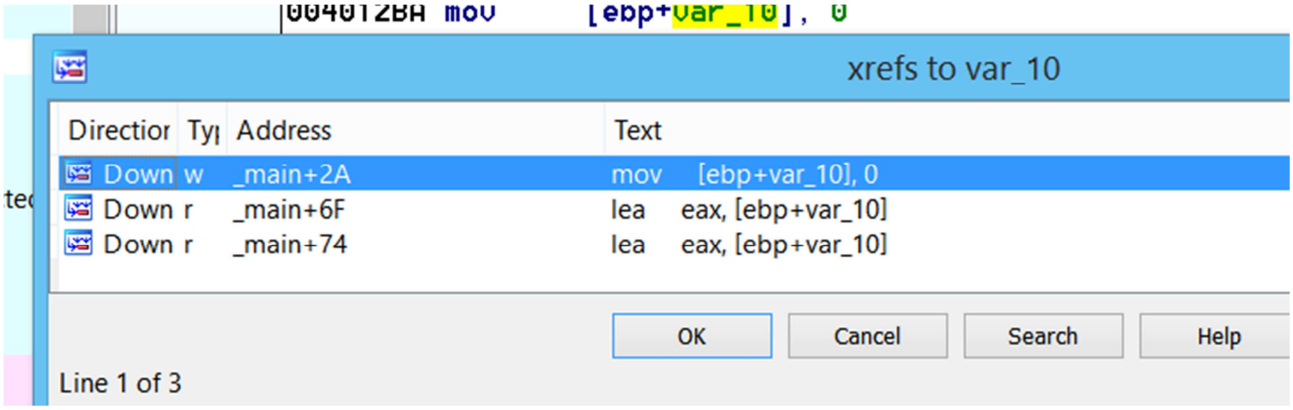
in **printf**, through LEA, it obtains the addresses of the same to print them but cannot alter its value there.



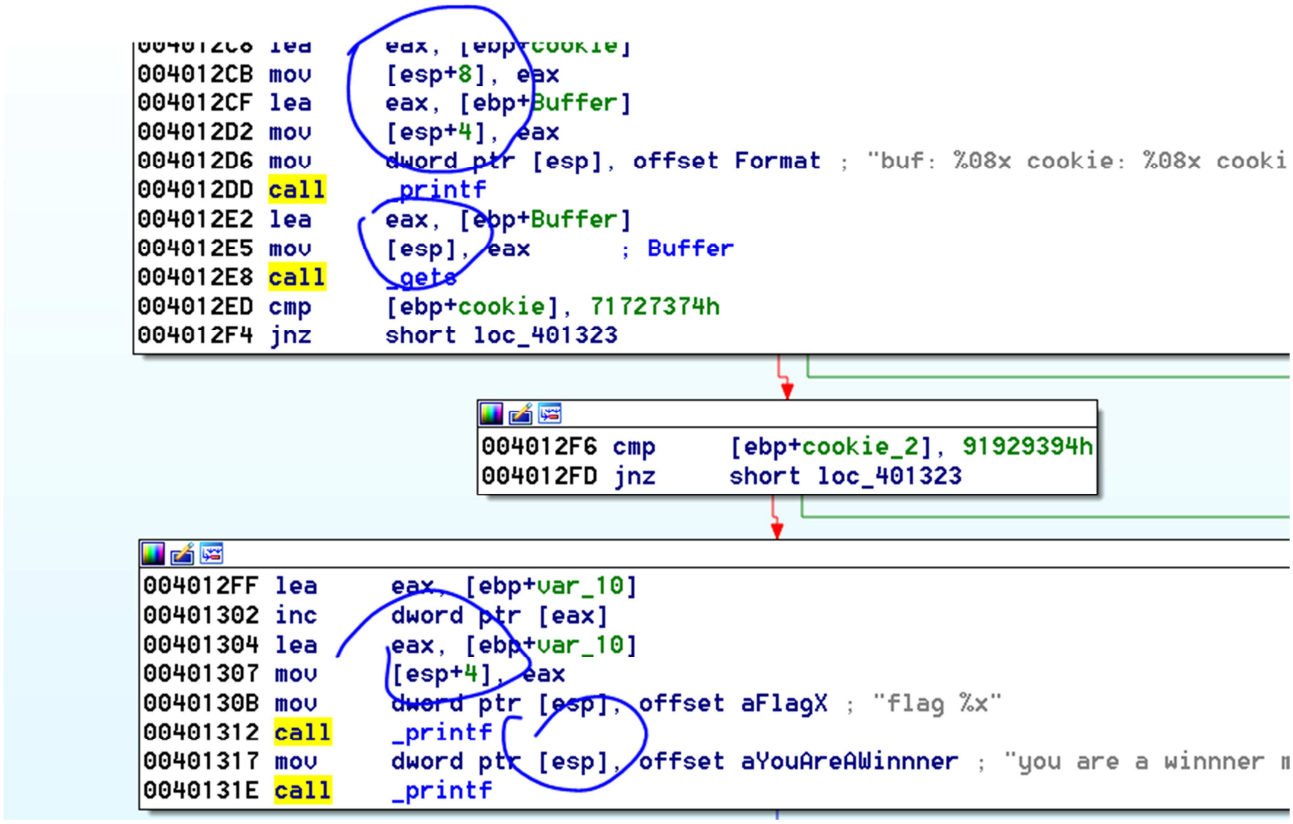

```
00401290 cookie= dword ptr -14h
00401290 var_10= dword ptr -10h
00401290 cookie_2= dword ptr -0Ch
00401290 argc= dword ptr  8
00401290 argv= dword ptr  0Ch
00401290 envp= dword ptr  10h
00401290
00401290 push    ebp
00401291 mov     ebp, esp
00401293 sub     esp, 78h
00401296 and     esp, 0FFFFFFF0h
00401299 mov     eax, 0
0040129E add     eax, 0Fh
004012A1 add     eax, 0Fh
004012A4 shr     eax, 4
004012A7 shl     eax, 4
004012AA mov     [ebp+var_5C], eax
004012AD mov     eax, [ebp+var_5C]
004012B0 call    __alloca
004012B5 call    ___main
004012BA mov     [ebp+var_10], 0
004012C1 lea     eax, [ebp+cookie_2]
```

If we press x on either variable, we confirm that only one is accessed twice to find the address before passing it to printf and the second when it already compares the value.So if you cannot change the value of these variables, how can we get to the msg of good guy to get to it if both variables must have a specific value? The other possibility of filling these variables would be that there is an overflow in some buffer that can overwrite its value.

There we see a var_10 variable. Let's see what it is, there we see that it initializes it to zero. Let's look at the other places where you use it with x.

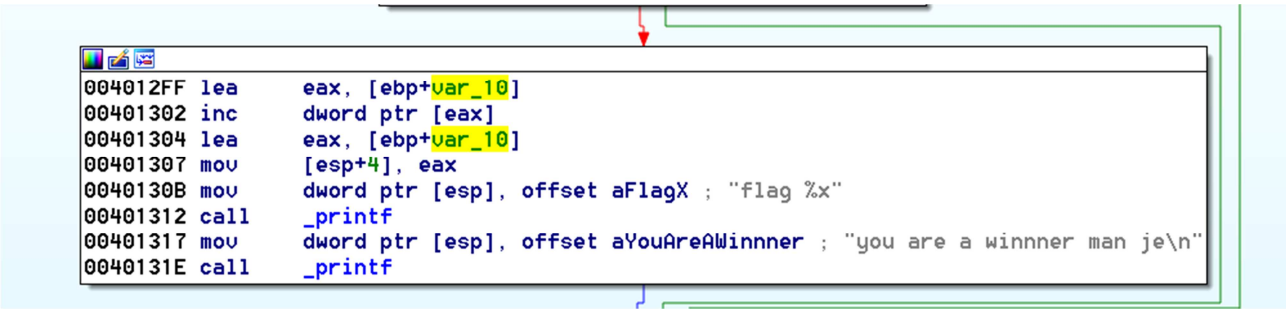


There, we see the three places where the first one is used when it initializes it to zero. Let's see the other two, but before we accommodate the arguments that are passed to the APIs. So that, it is not so ugly using the technique that we saw in my IDA tutorial 1, by right clicking on them and changing by an alternative representation that IDA shows us there.



Now it looks better and it's clear how it saves the arguments in the stack to pass them to the APIs.

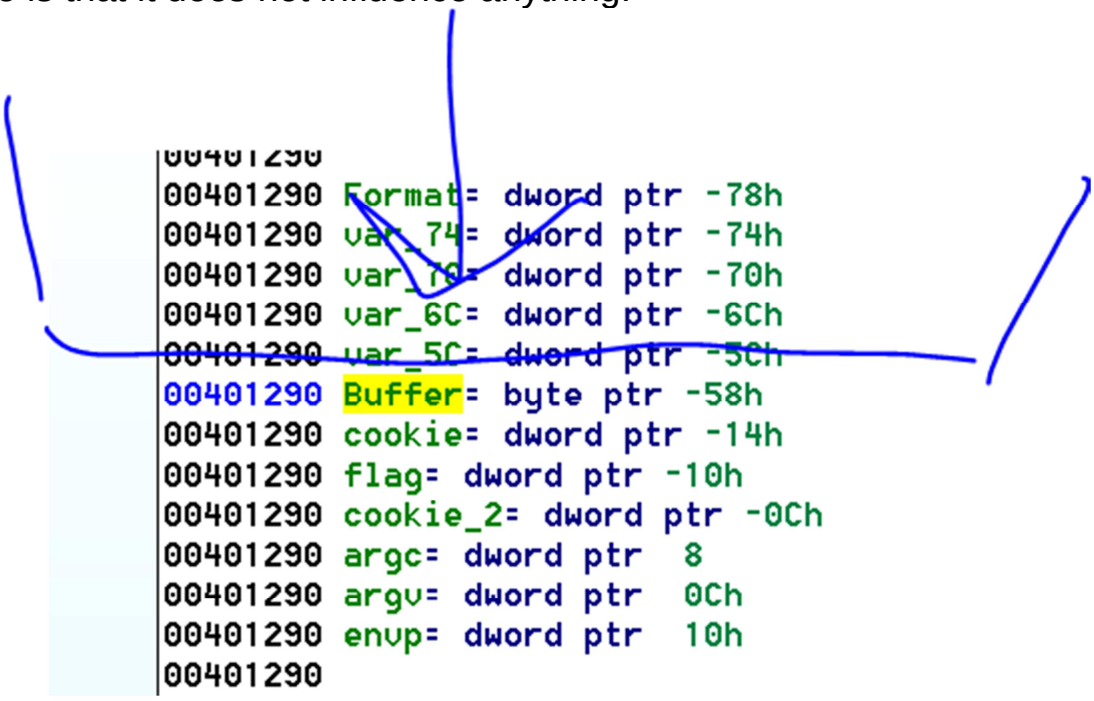
Now let's see where it uses the variable.



We see that there with the LEA gets the address of the variable and then increases its content or increases the value of it.

Then it gets the address of the variable again and passes it as an argument to **printf**, or it prints the address of the variable, not the value, if we get to the good zone.

As the message says **flag** we can rename it with that same name, what we do see is that it does not influence anything.



We only have to study the buffer since the variables that are above are variables created by the processor and temporary not the program itself.

Looking at the static representation of the stack.

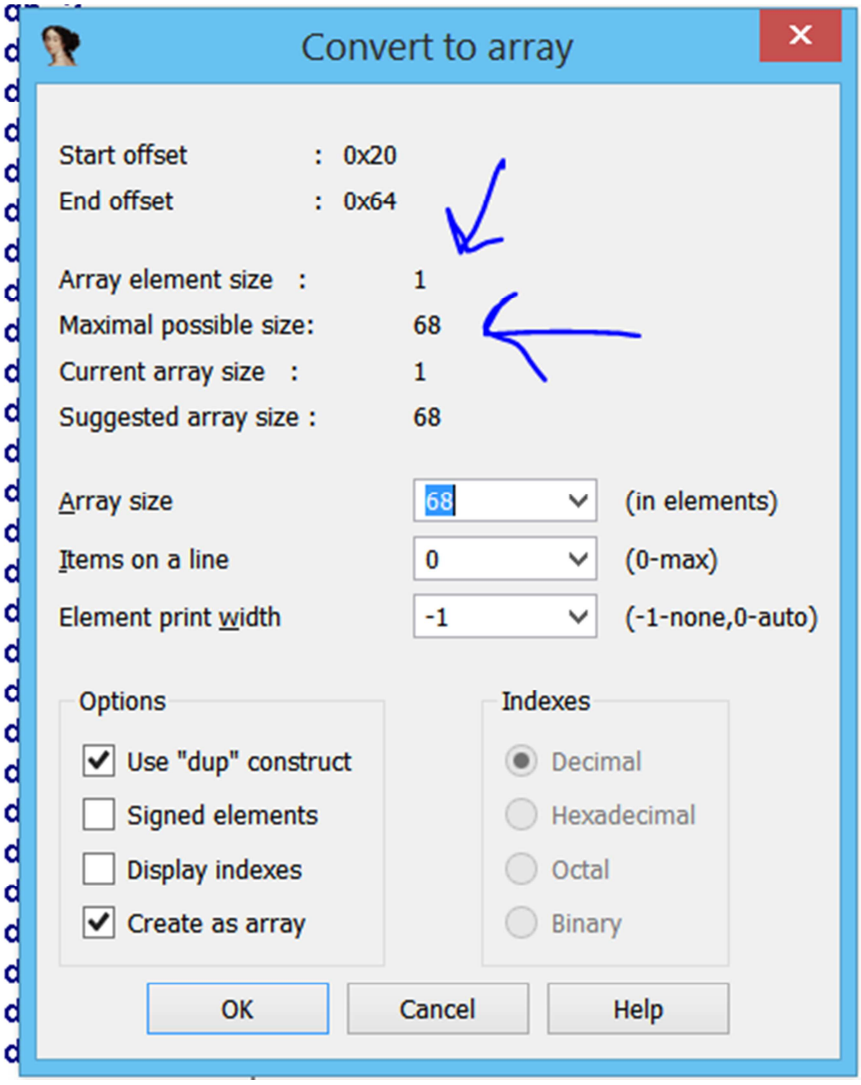
```
-0000005C var_5C          dd ?
-00000058 Buffer          db ?
-00000057                 db ? ; undefined
-00000056                 db ? ; undefined
-00000055                 db ? ; undefined
-00000054                 db ? ; undefined
-00000053                 db ? ; undefined
-00000052                 db ? ; undefined
-00000051                 db ? ; undefined
-00000050                 db ? ; undefined
-0000004F                 db ? ; undefined
-0000004E                 db ? ; undefined
-0000004D                 db ? ; undefined
-0000004C                 db ? ; undefined
-0000004B                 db ? ; undefined
-0000004A                 db ? ; undefined
-00000049                 db ? ; undefined
-00000048                 db ? ; undefined
-00000047                 db ? ; undefined
-00000046                 db ? ; undefined
-00000045                 db ? ; undefined
-00000044                 db ? ; undefined
-00000043                 db ? ; undefined
-00000042                 db ? ; undefined
-00000041                 db ? ; undefined
```

Double-clicking the buffer shows the static representation of the IDA stack. Obviously, **buffer** is the place reserved in memory to save what is typed in console since it is passed the address as argument to the **gets** function.

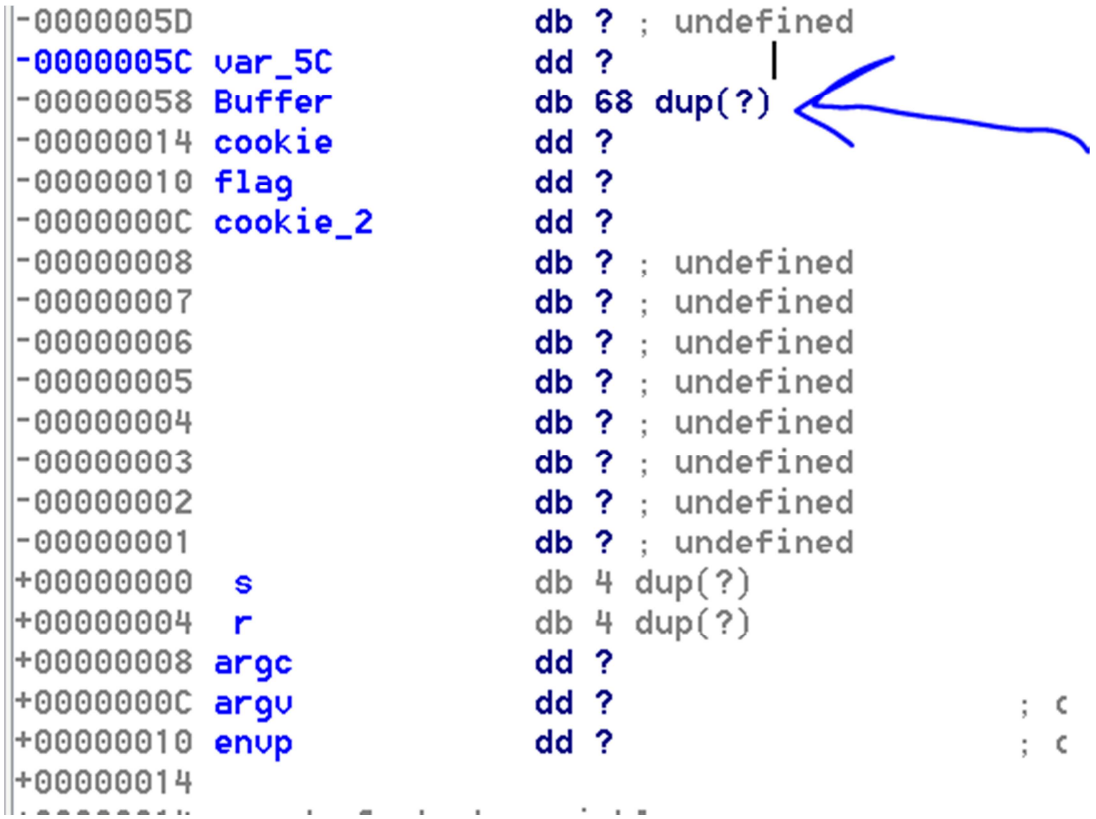
```
004012C8 lea     eax, [ebp+cookie]
004012CB mov     [esp+8], eax
004012CF lea     eax, [ebp+Buffer]
004012D2 mov     [esp+4], eax
004012D6 mov     dword ptr [esp], offset Format ; "buf: %08
004012DD call    _printf
004012E2 lea     eax, [ebp+Buffer]
004012E5 mov     [esp], eax      ; Buffer
004012E8 call    _gets
004012ED cmp     [ebp+cookie], 71727374h
```

The first access to Buffer gets the address and this is passed printf to print it and the second passes the address to gets so that it receives what we typed.

To see the size of that buffer in the representation of the stack we right click - array and see the size that IDA suggests.



It suggests to us 68 decimal long as each element is one byte long it will be 68 decimal. We accept.



There, we see the buffer and also seeing this representation we know that filling the buffer with 68 characters will be just about to overflow and how "gets" has no restrictions we can overflow the buffer and overwrite it with four bytes plus the cookie variable that is next. Then, with four other bytes I overwrite **flag** which is a dword (dd) and with another four, I overwrite **cookie 2**.

With that, we could build the script. What we would send to it would be.

fruta= 68 *"A"+ cookie + flag + cookie2

```
from subprocess import *
import struct
p = Popen([r'C:\Users\ricna\Desktop\23-INTRODUCCION AL REVERSING CON IDA PRO DESDE CERO PARTE 23\IDA2.exe', 'f'], stdout=PIPE, stdin=PIPE, stderr=STDOUT)
```

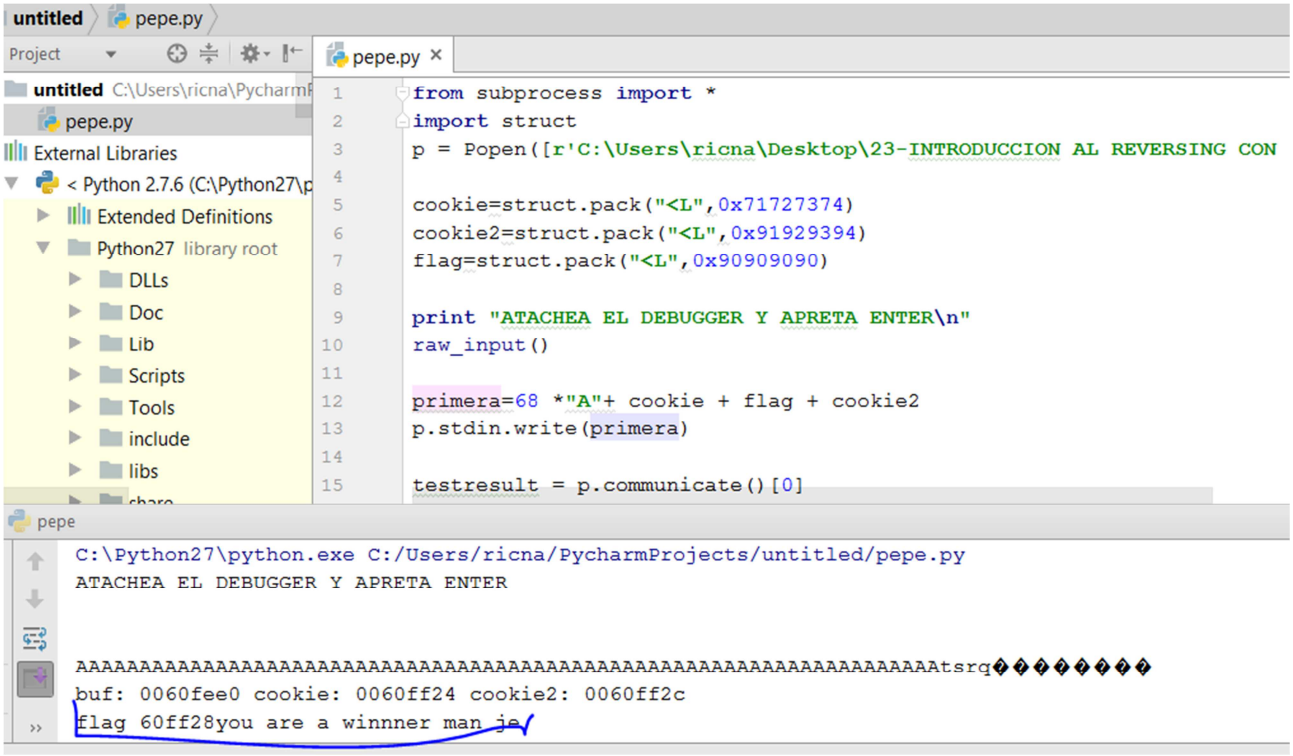
```
cookie=struct.pack("<L",0x71727374)
cookie2=struct.pack("<L",0x91929394)
flag=struct.pack("<L",0x90909090)

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

primera=68 *"A"+ cookie + flag + cookie2
p.stdin.write(primera)

testresult = p.communicate()[0]

print primera
print(testresult)
```



The chicken is ready. We set the corresponding values to **cookies** and 90909090 to **flag** or whatever that does not disturb. And there it is. Until the next part. To practice, you have the IDA3.exe and the IDA4.exe that are attached.

Ricardo Narvaja

Translated by: @IvinsonCLS