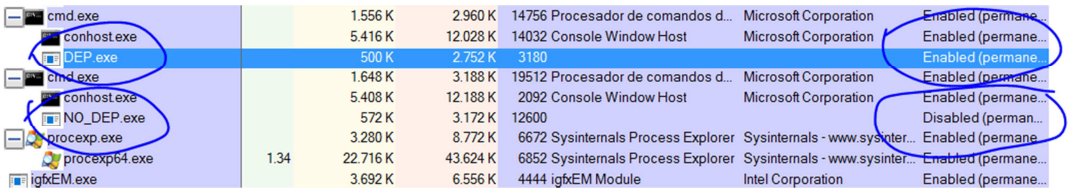
PART 35

I have compiled two programs, one with DEP and the other without it. The code is the same, but in this case, instead of changing it in the compilation, I call the SetProcessDEPPolicy API. One with the 0 argument (DEP off) and the other with the 1 argument (DEP on)

If I run both and see them in Process Explorer, both stopped in the gets_s waiting for data and passed by SetProcessDEPPolicy. So, the DEP is set in both with the API. In one, it is on and the other off.

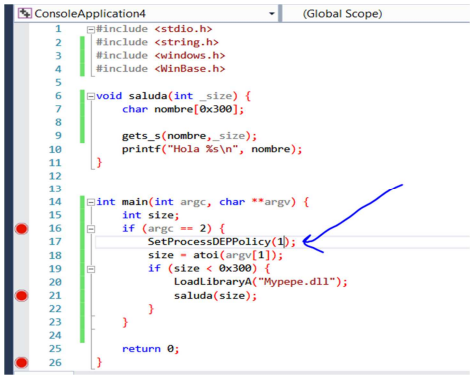


These examples are better than the previous one because the code is similar and the previous one didn't have enough space to do ROP.

The other plugin we need to install is **idasploiter**.

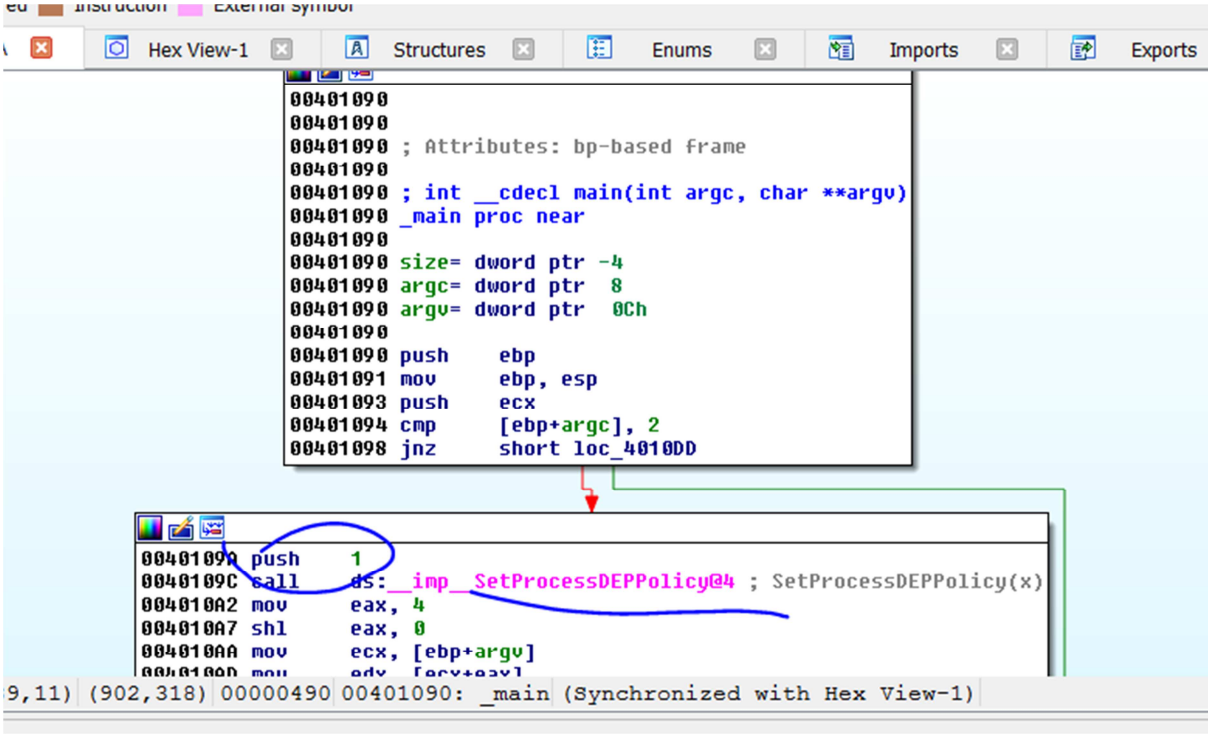
https://github.com/iphelix/ida-sploiter

It is a .py we download from the above link by pressing CLONE OR DOWNLOAD and we copy the .py to IDA plugin folder.

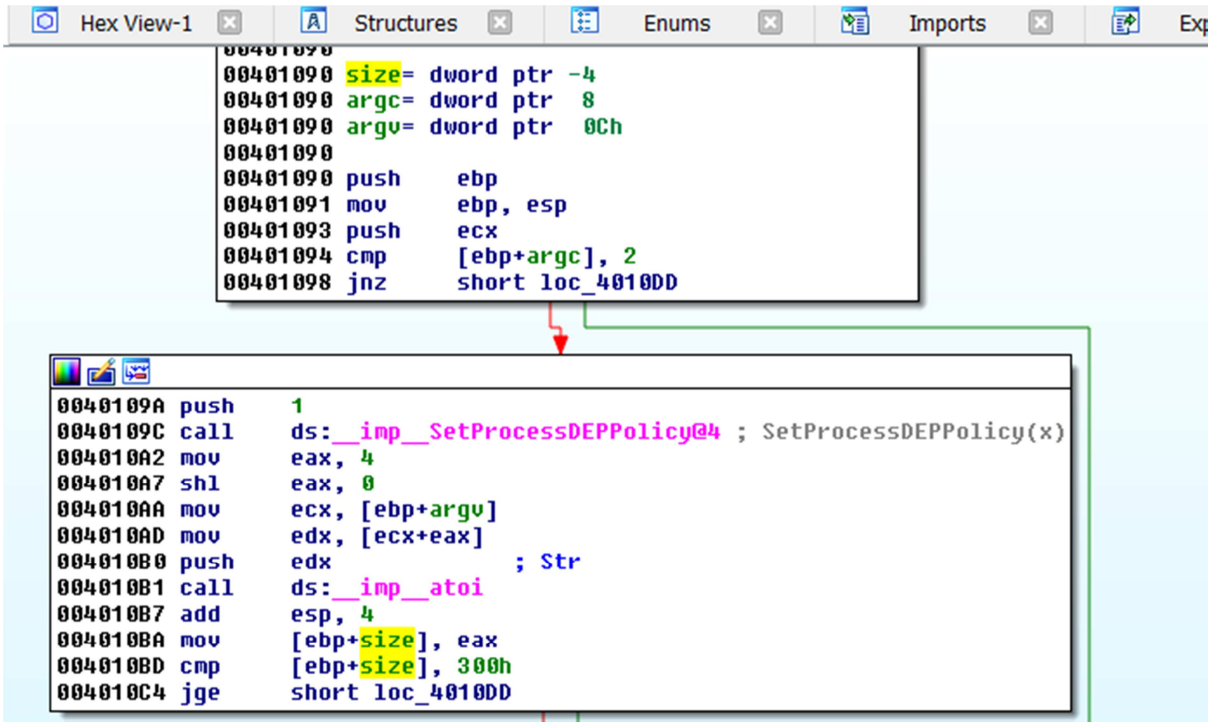


The code is similar in both. The only difference is the 0 and 1 argument of the function.

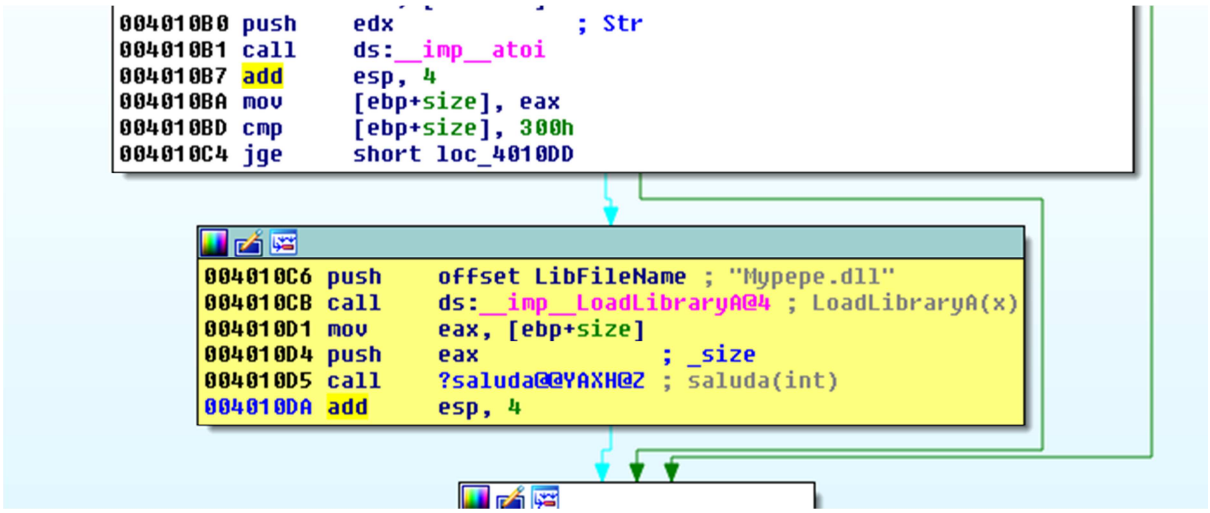
As it is the first one, it will be easier to analyze just once because the reversing will be similar since they have the same code.



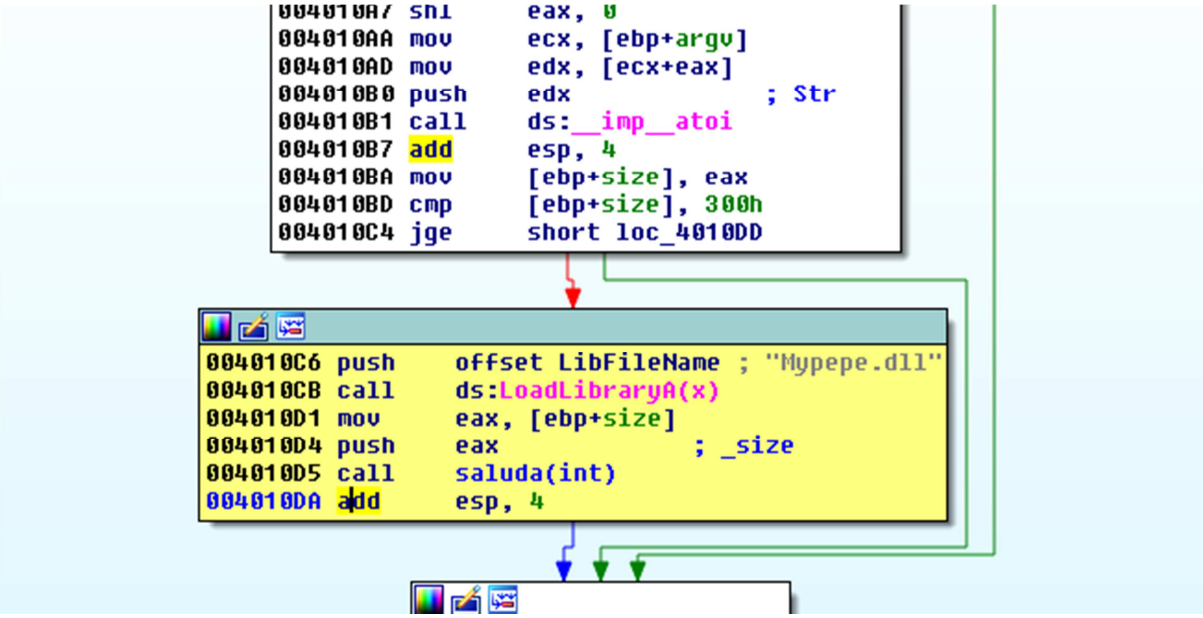
I will work with the one without DEP. Anyways, the analysis will work for both.



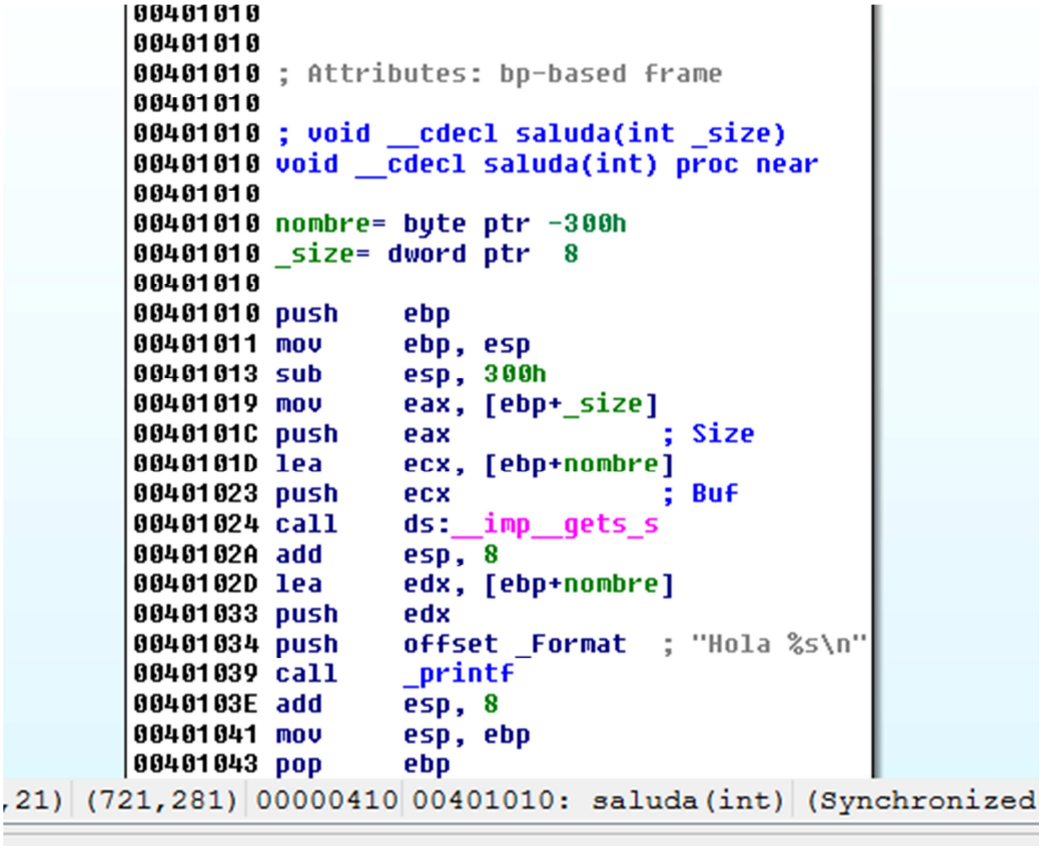
It uses **atoi** to convert the number we enter to integer and use it as an argument and it saves it in the size variable that is signed. Below, it compares it with **JG** that is a signed comparison. So, we can enter negative numbers below 0x300 which is used as an argument of the **saluda** function and inside, it is used as a **gets_s** size that is taken as unsigned triggering a possible overflow because it will let us enter more than 0x300 bytes in the buffer of that size.



It loads Mypepe.dll using LoadLibrary. We can use **Demangle names** to see it better.



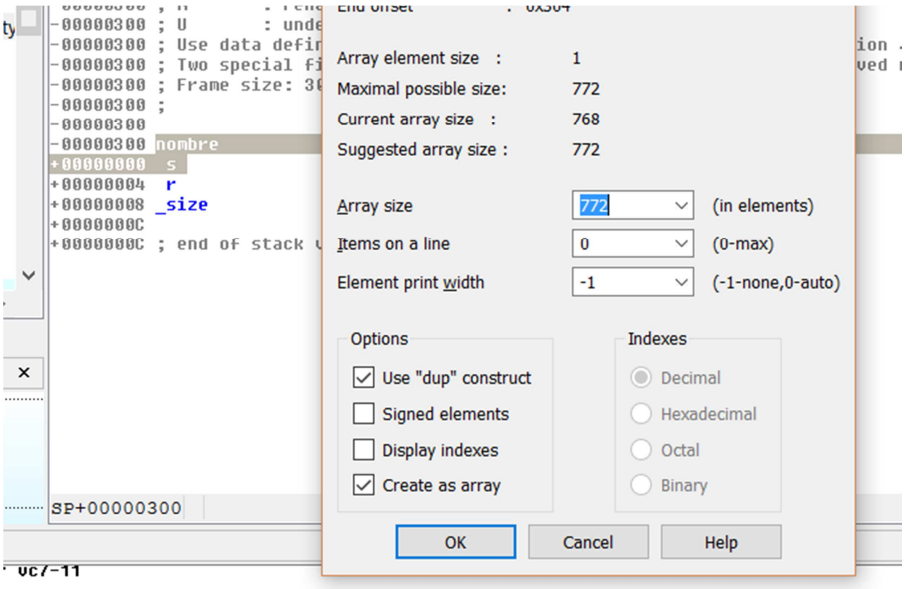
Now, it looks better. Let's see the **saluda** function.



As I compiled it with symbols, it detects the **nombre** buffer and passes the address to **get_s**, besides the size that is this function argument.

Let's see the stack representation.

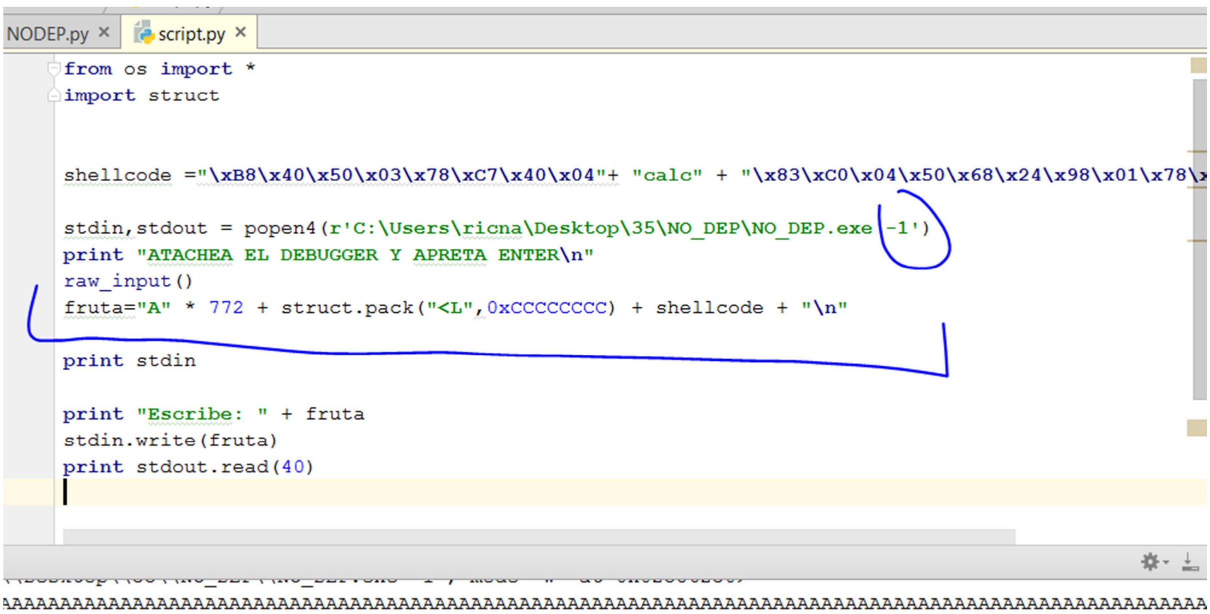
To overflow the buffer and overwrite just before the stack, I need 722 bytes.



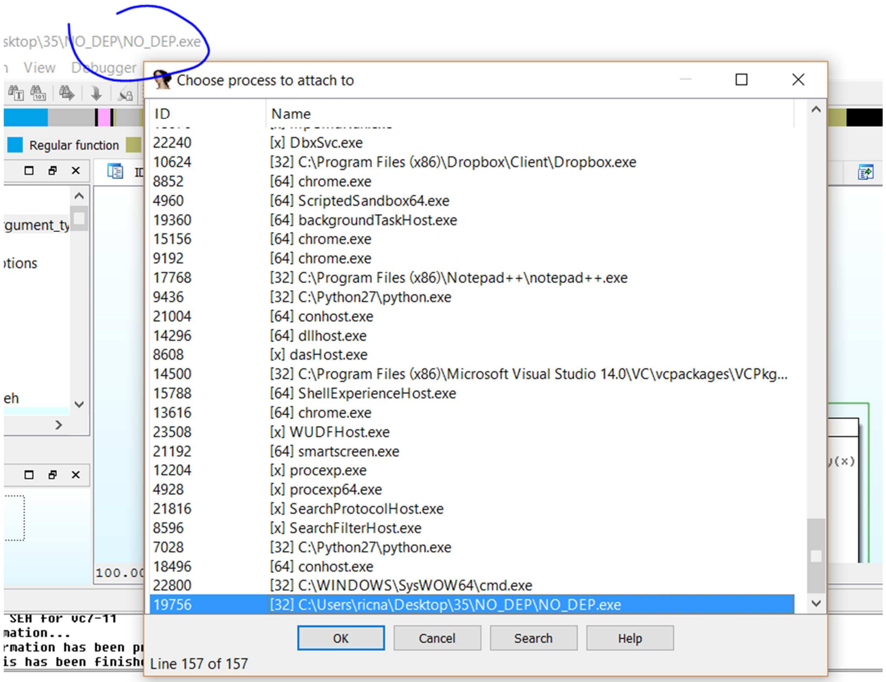
So, I should enter something like this in gets:

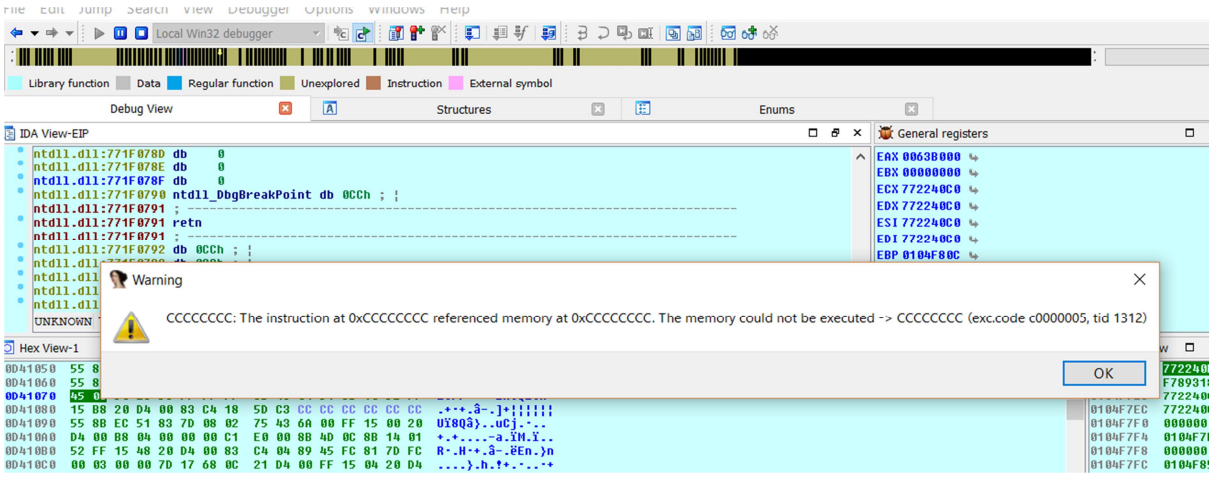
fruta="A" * 772 + struct.pack("<L",0xCCCCCCCC) + shellcode

Let's create the script. It must enter the negative size as an argument to trigger the overflow.

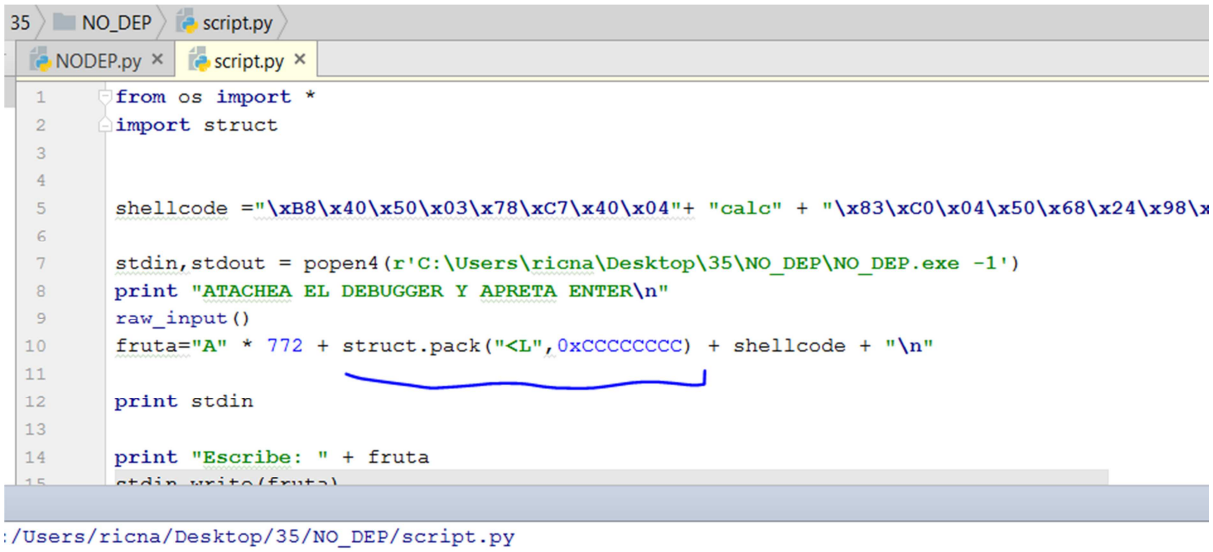


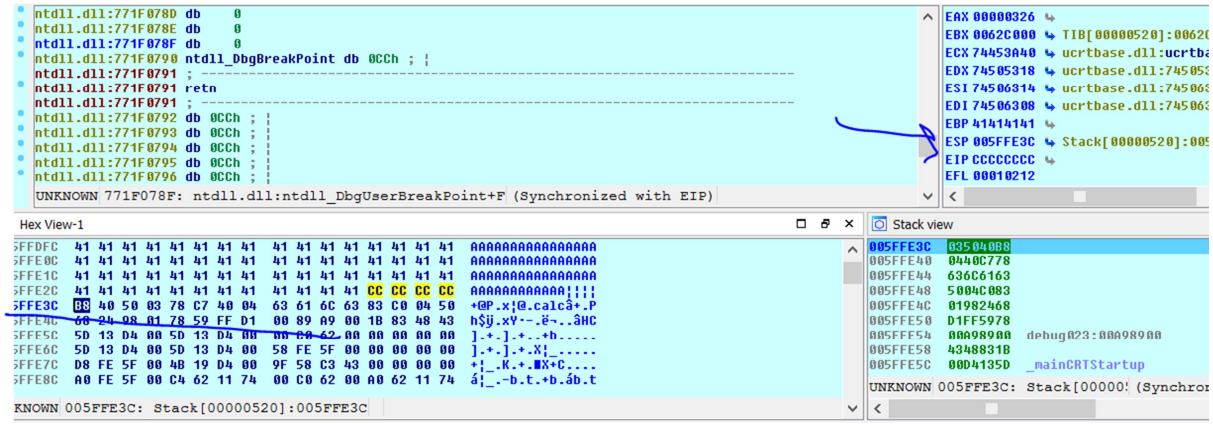
If I run it and attach it with IDA that has the analysis of the NO_DEP.exe





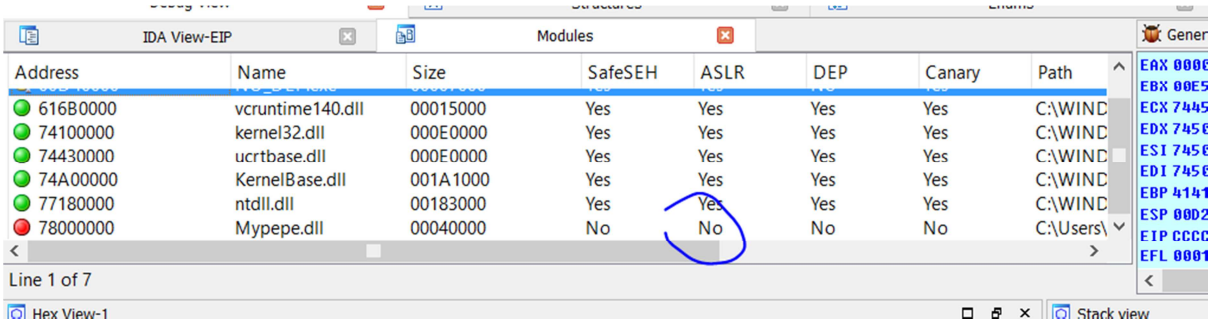
I see that all is well calculated. It jumps to 0xCCCCCCCC as I programmed it in my script.





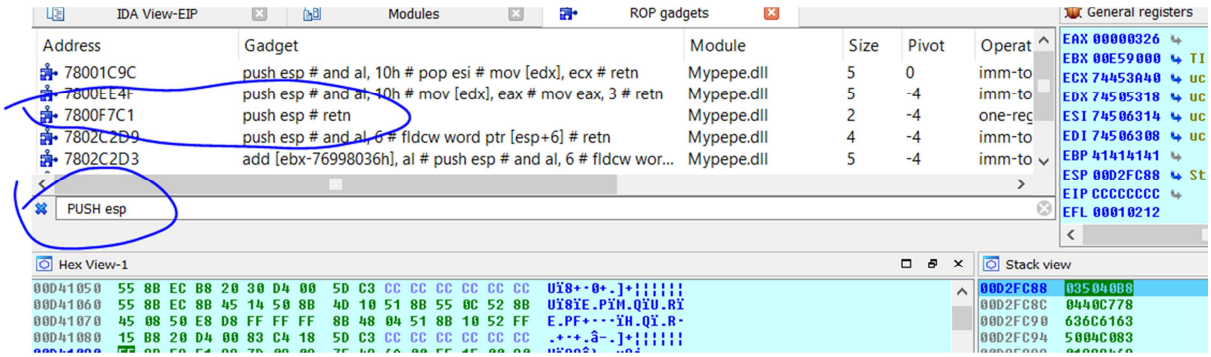
After accepting, ESP points to my shellcode in the stack as there is no DEP, if instead of jumping to 0xCCCCCCCC, it jumped to JMP ESP, CALL ESP or PUSH ESP-RET in some module without randomization for it not to change, it would be ready.

Courtesy of IDA SPLOITER, it will show other module list that is in VIEW-OPEN SUBVIEW-MODULES or SHIFT+F6.

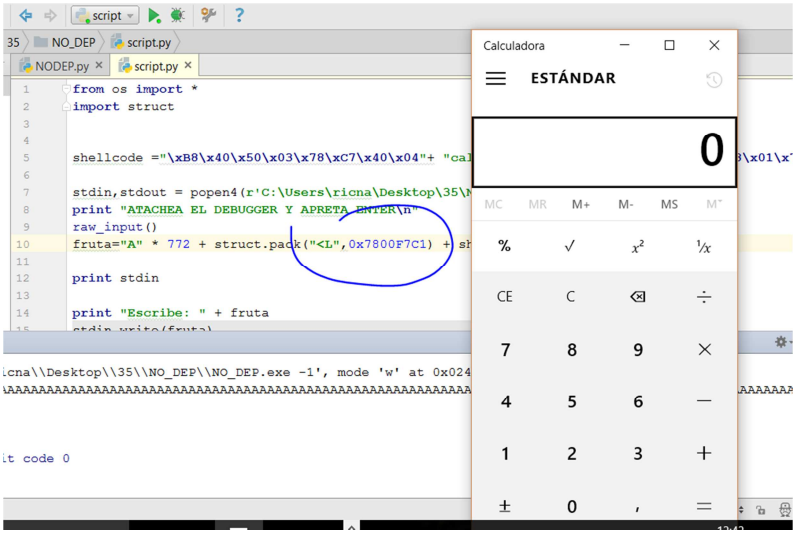


We see that Mypepe.dll has no ASLR (randomization). So, it is a good candidate to look for the JMP ESP there.

If we right click, it has the SEARCH GADGETS option to look for code chunks that end in a RET. Once I make it list all the gadgets, I can press CTRL+F and look for PUSH ESP.



So, I could use that address here. No problem with the 0's because gets_s accepts them.



Ready, the shellcode was made for mypepe. So, it works like the last time.

I would like you to see if the program with DEP can execute the calculator without ROP just rearranging the stack a little and without a shellcode.

In the next part, we will do the DEP.exe with ROP.

Ricardo Narvaja

Translated by: @IvinsonCLS