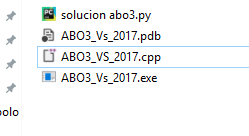
EXPLOITING Y REVERSING

USANDO HERRAMIENTAS

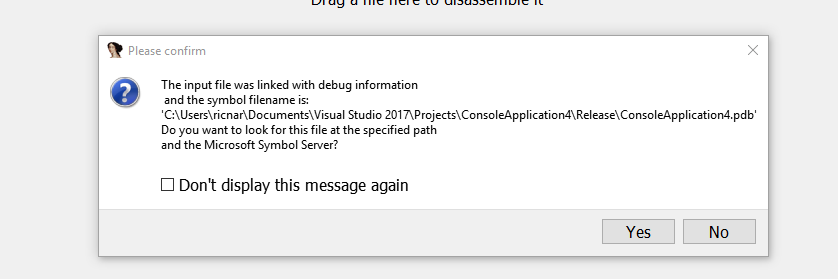
GRATUITAS (PARTE 8)

Hemos solucionado el ABO2 en GHIDRA y ahora nos toca usar IDA FREE para solucionar el ABO3.

Como en todos los ABOS la idea es ejecutar la calculadora o algún ejecutable que querramos.



Abrimos el ABO3 en el IDA FREE para analizarlo.



Busco el pdb con los símbolos podría mostrar el siguiente error.



# MS DIA SDK ERROR

En caso de que les salga este error necesitamos instalar el Visual C++ 2008 redistributable para 64 bits.

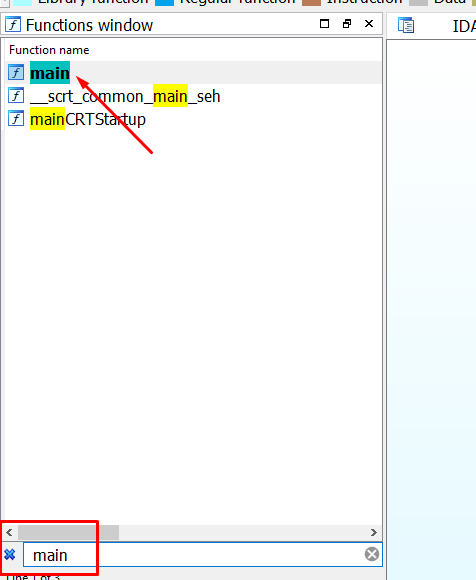
# INSTALANDO VISUAL C++ 2008 REDISTRIBUTABLE PARA 64 BITS.

El paquete se puede bajar desde aqui.

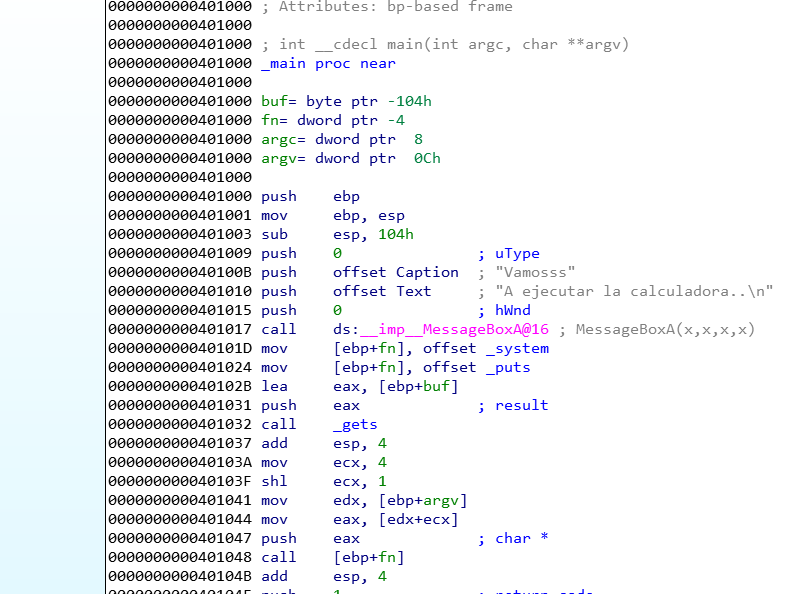
<https://www.microsoft.com/en-us/Download/confirmation.aspx?id=15336>

Una vez instalado y reiniciado IDA cargamos el ABO3.

Cuando nos pida los símbolos los abrimos y luego en la ventana de funciones apretamos CTRL + F y tipeamos **main** para buscar.

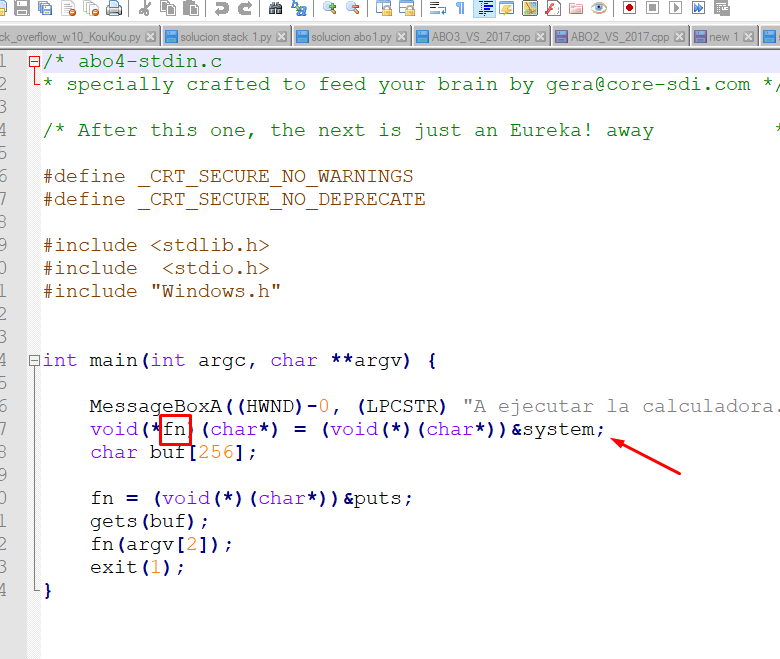


Vamos allí con doble click en el resultado hallado.



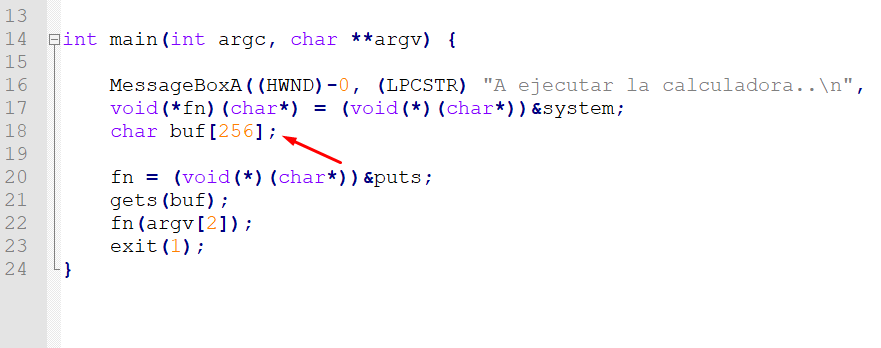
En IDA FREE no tenemos decompilador, solo la versión PRO lo tiene, igual está el código fuente en el archivo cpp y si no tenemos el cpp, podemos decompilarlo con el x64dbg.

# CÓDIGO FUENTE



Vemos una variable **fn** en el stack del tipo puntero a función, se inicializa con la dirección de la función **system()**, que sirve para ejecutar un comando de consola, si le pasamos el nombre del un ejecutable, si el mismo existe y lo puede encontrar lo ejecutará.

Hay un buffer de 256 bytes llamado **buf**.



Al iniciar el programa, la variable **fn** cambia a guardar la dirección de la función **puts()**, que sirve para imprimir en la consola un texto.

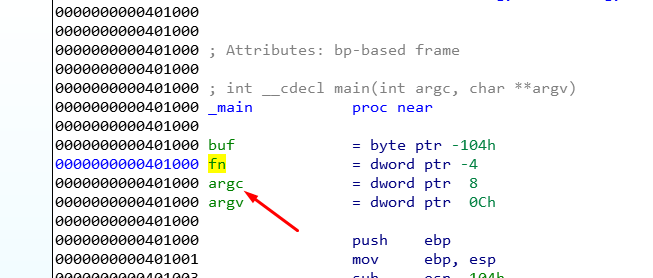
Luego llama a **gets()** para llenar el buffer llamado **buf** y finalmente llama a ejecutar la funcion cuya dirección está guardada en **fn**, la cual si no hay overflow será **puts()**, se imprimirá el texto que ingresamos como argumento 2 (**argv[2]**) y saldría del main sin llegar nunca al return address.

Bueno la idea es que como el ingreso de datos es mediante **gets()** podemos desbordar **buf** y pisar el valor de la variable **fn,** con lo cual al ejecutar en **fn(argv[2])**, podremos saltar donde queremos.

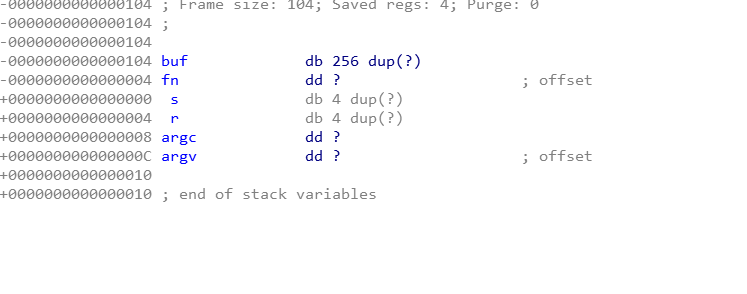
Ahora hagamos el reversing estático en IDA como si no hubiéramos visto el código fuente.

# ANÁLISIS ESTÁTICO DEL ABO3

Haciendo doble click en alguna variable en la definición de la funcion



Vemos la **representación estática del stack.**



Alli vemos **buf,** que es una variable tipo array de bytes (db= BYTES) cuyo largo es 256 bytes y debajo está **fn,** que como es una variable del tipo puntero en 32 bits su largo es 4 bytes o (dd= DWORD)



Alli vemos todos los accesos a la variable **fn**, primerose guarda la dirección de la función **system**, en la siguiente línea se guarda la dirección de **puts** y finalmente se salta a ejecutar la funcion cuyo valor está guardado en **fn**, si no hay overflow saltara a **puts**.

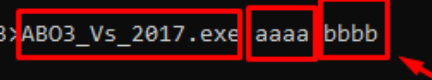
Y cuál será el argumento de **puts**?

**argv** es un array de punteros, cada uno de ellos apunta a un argumento de consola.

**argv[0]** apunta al nombre del ejecutable.

**argv[1]** apunta al primer argumento.

**argv[2]** apunta al segundo argumento etc.



Si lo ejecuto desde una consola con esos argumentos.

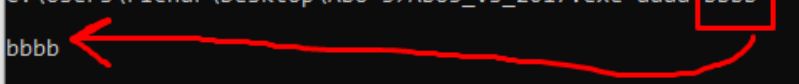
**ABO3\_Vs\_2017.exe aaaa bbbb**

**argv[0]** apunta al nombre del ejecutable =ABO3\_Vs\_2017.exe

**argv[1]** apunta al primer argumento = aaaa

**argv[2]** apunta al segundo argumento= **bbbb**

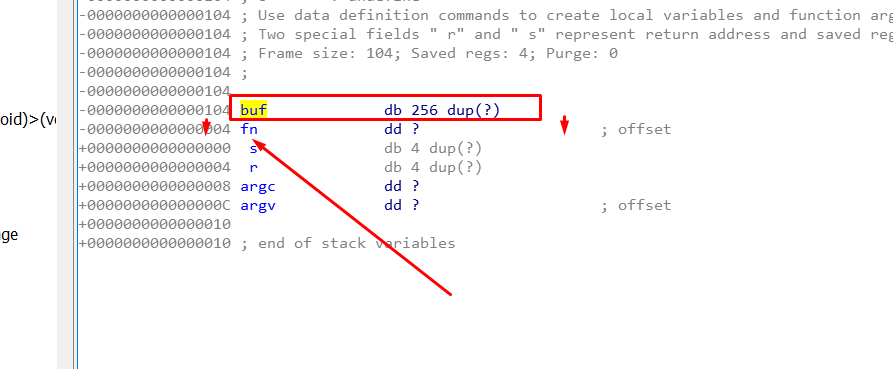
Por lo tanto el argumento que se le pasa a la funcion **puts** está controlado por nosotros, vemos que finalmente después de apretar ENTER, para pasar la pausa del programa, imprime **bbbb** o sea el **argv[2].**



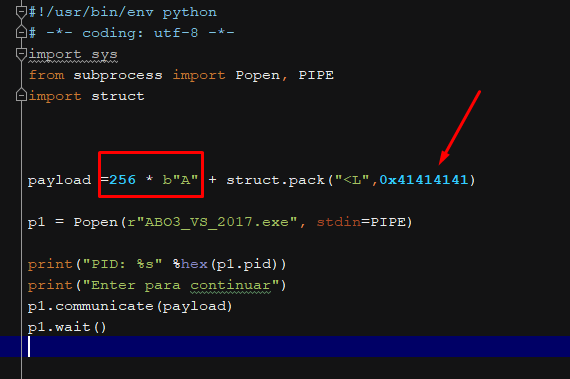
Asi que el argumento sea cuál sea la funcion que este guardado en **fn**, será el **argv[2]** y será controlado por nosotros.

Si hiciéramos un script.

Como la idea es llenar **buf** y desbordarlo para pisar **fn**.

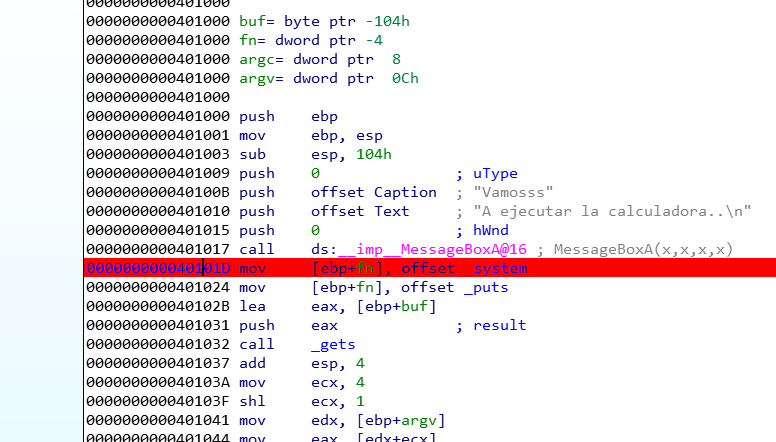


Habrá que enviar 256 bytes para llenar **buf**  y luego los 4 bytes que pisaran **fn**.

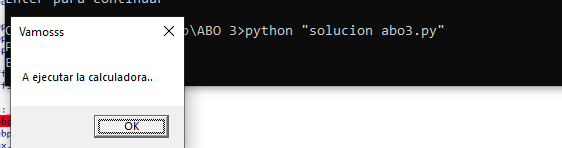


Eso saltaría a ejecutar a 0x41414141 probemos.

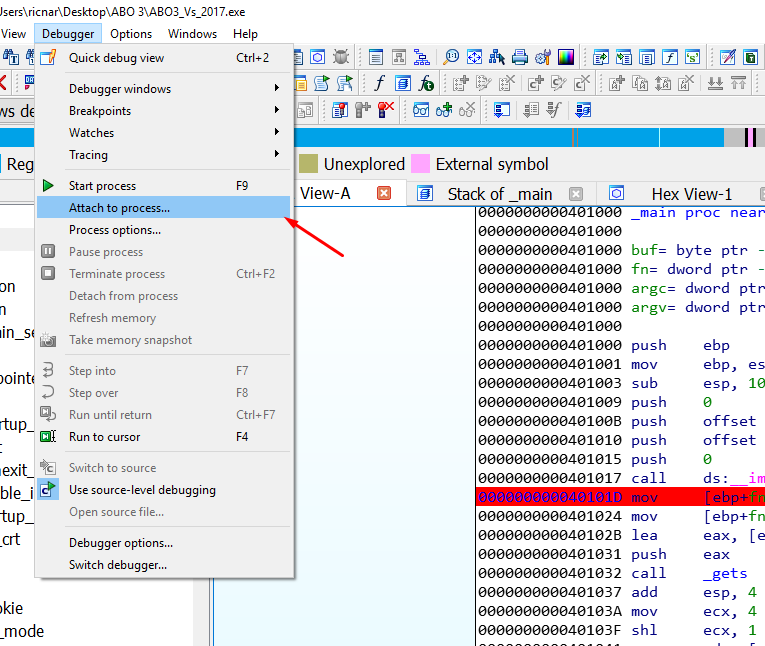
Pongamos un breakpoint al volver del MessageBoxA

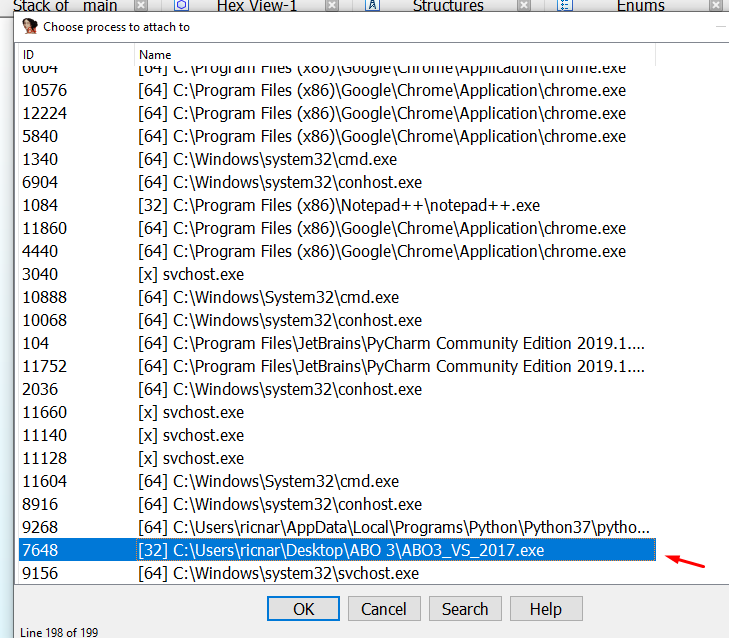


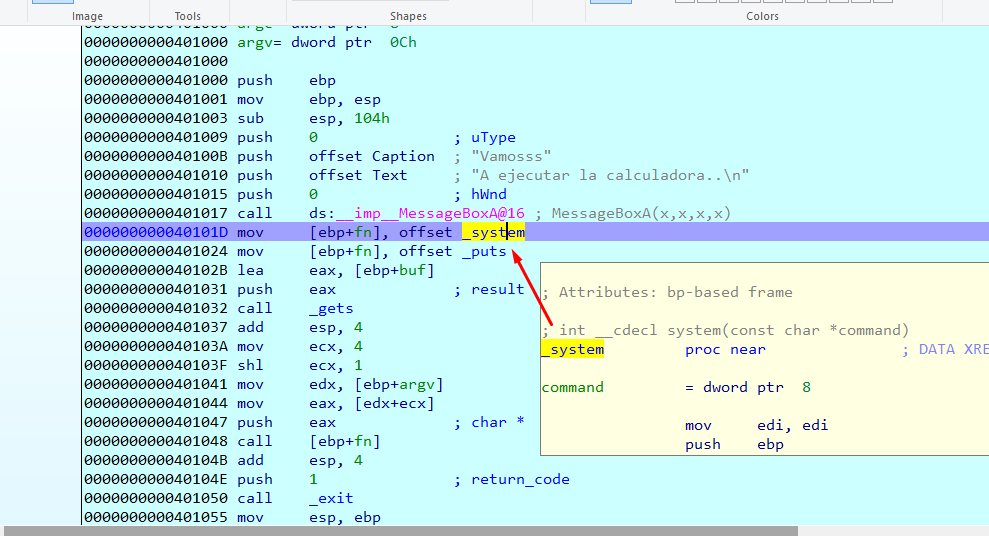
Corramos el script desde una consola.



Atacheemos el IDA.

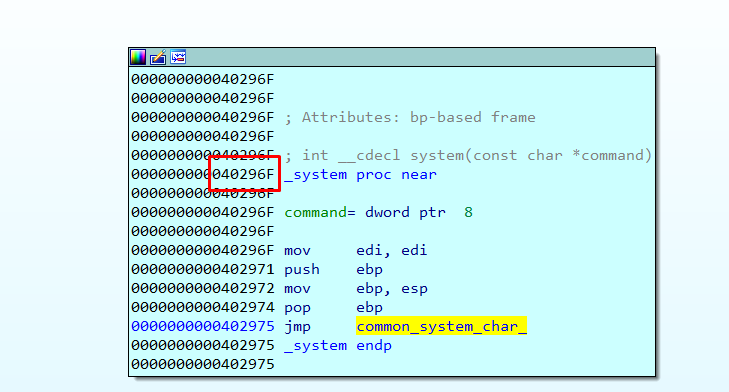




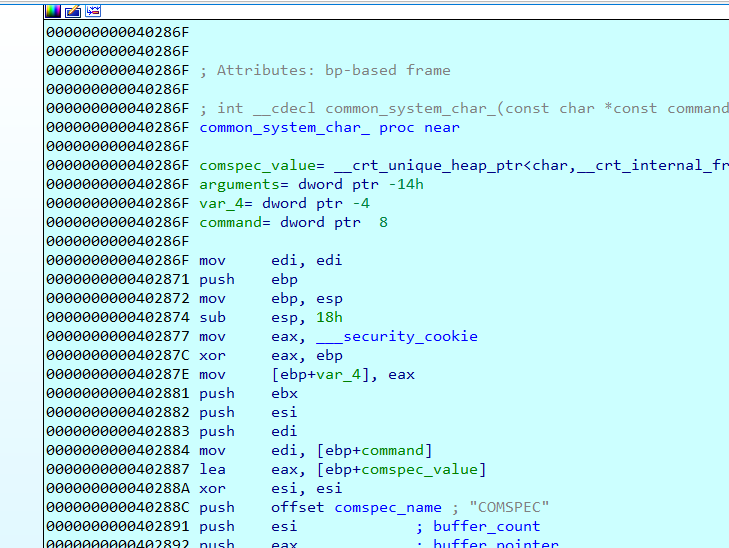


Una vez aceptado el mensaje se detiene y vemos como va a guardar la dirección de **system** en **fn**.

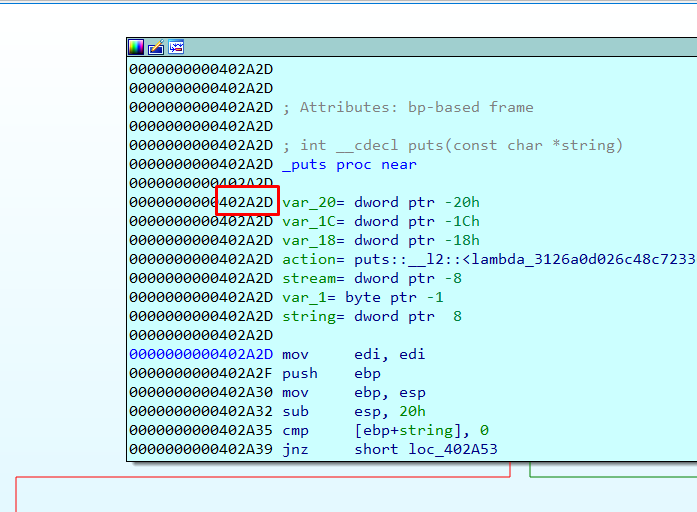
Si hacemos doble click en **system** nos llevará a la dirección de la función.



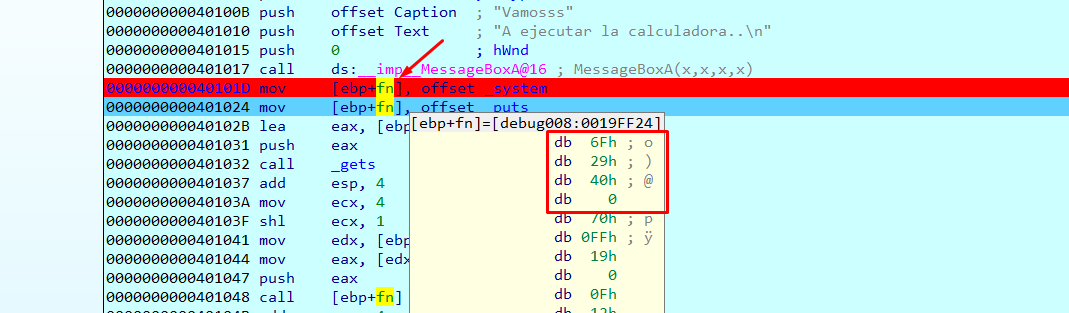
Como el ejecutable está compilado con las dll embebidas, **system** está dentro del mismo ejecutable, en este caso no pertenece a una dll.



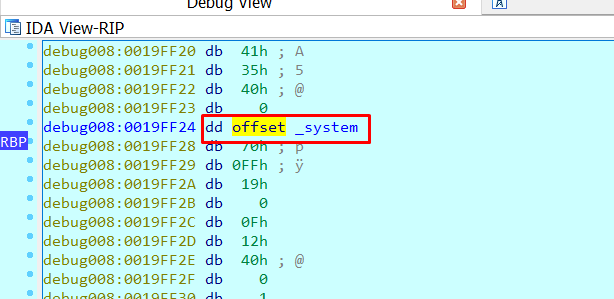
O sea que al compilar guardo una copia de la api **system** en el mismo ejecutable, lo mismo pasa con **puts**.



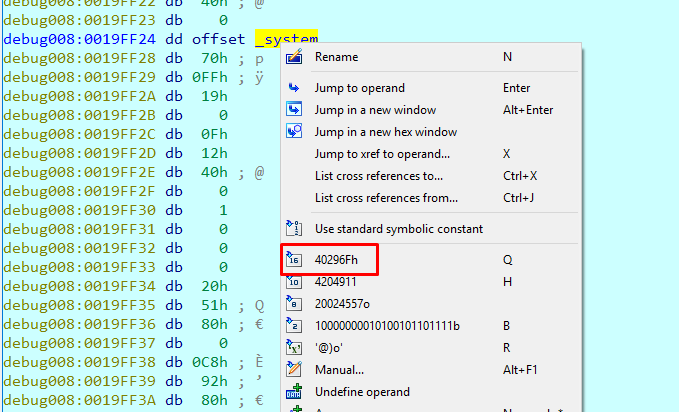
Si apretamos f8 y paso el mouse encima de **fn,** veo que guardo la dirección de **system**.



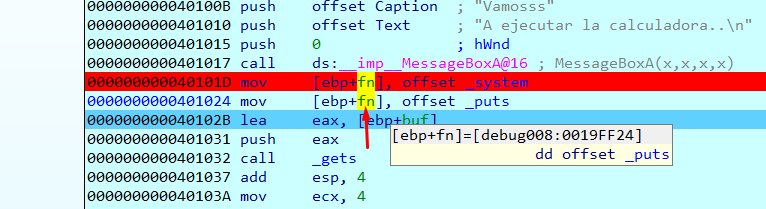
Si hago doble click en **fn,** lo puedo acomodar para que sea un DWORD, apreto la tecla D hasta transformarlo en un DWORD y mostrará **system**.



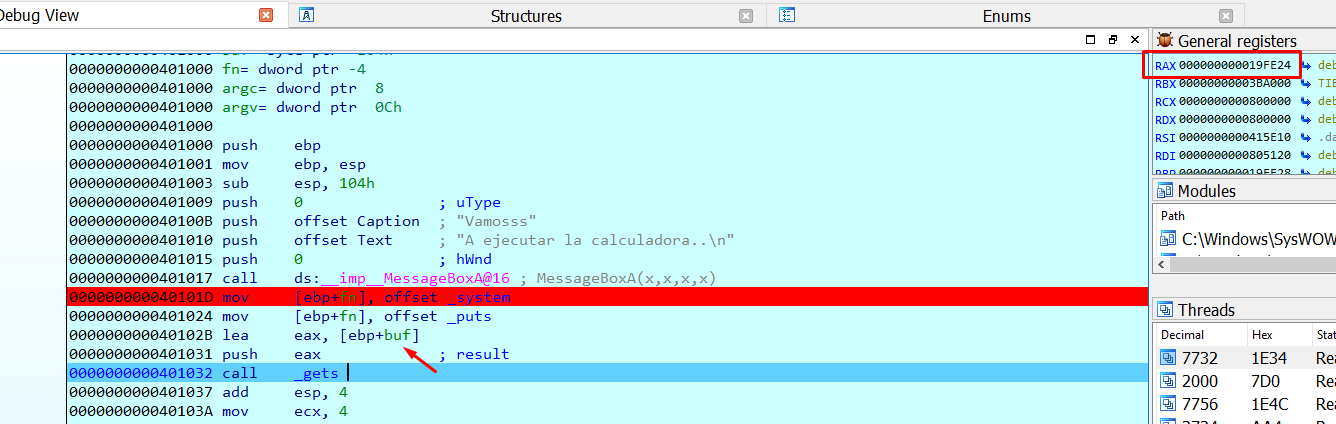
Alli vemos la dirección de system con click derecho.



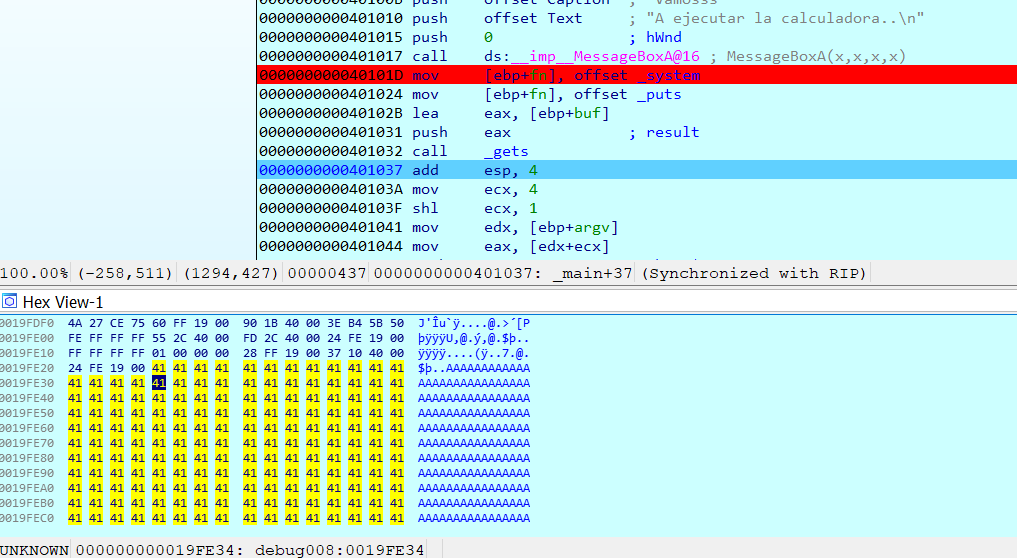
Apreto f8 de nuevo y guarda la dirección de **puts** en **fn**, pasando el mouse por encima la veré como DWORD, pues ya quedó cambiada dicha posición de memoria donde está **fn**.



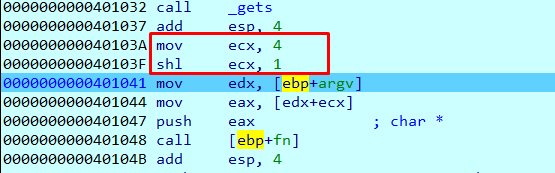
Si sigo traceando llegare el gets().



EAX tendrá la dirección de **buf** y llenará el mismo al pasar por encima del **gets()**.



Sigo traceando.

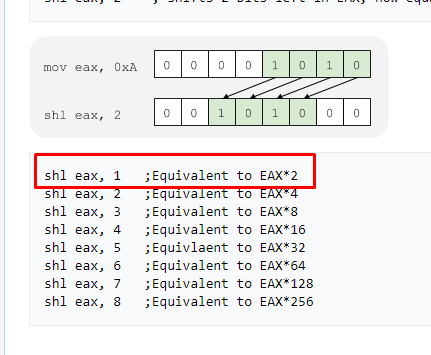


Mueve 4 a ECX y luego

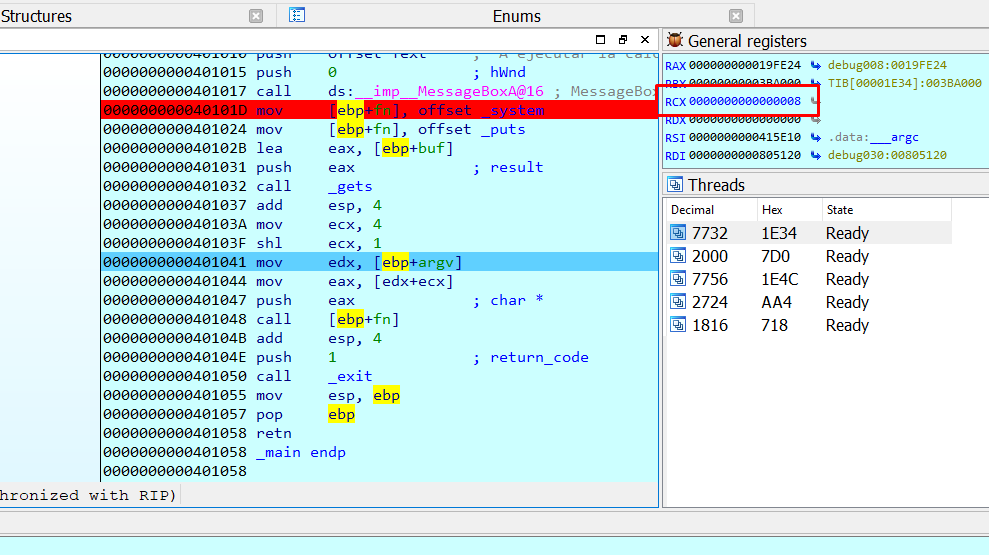
**SHL ECX, 1**

<https://www.aldeid.com/wiki/X86-assembly/Instructions/shl>

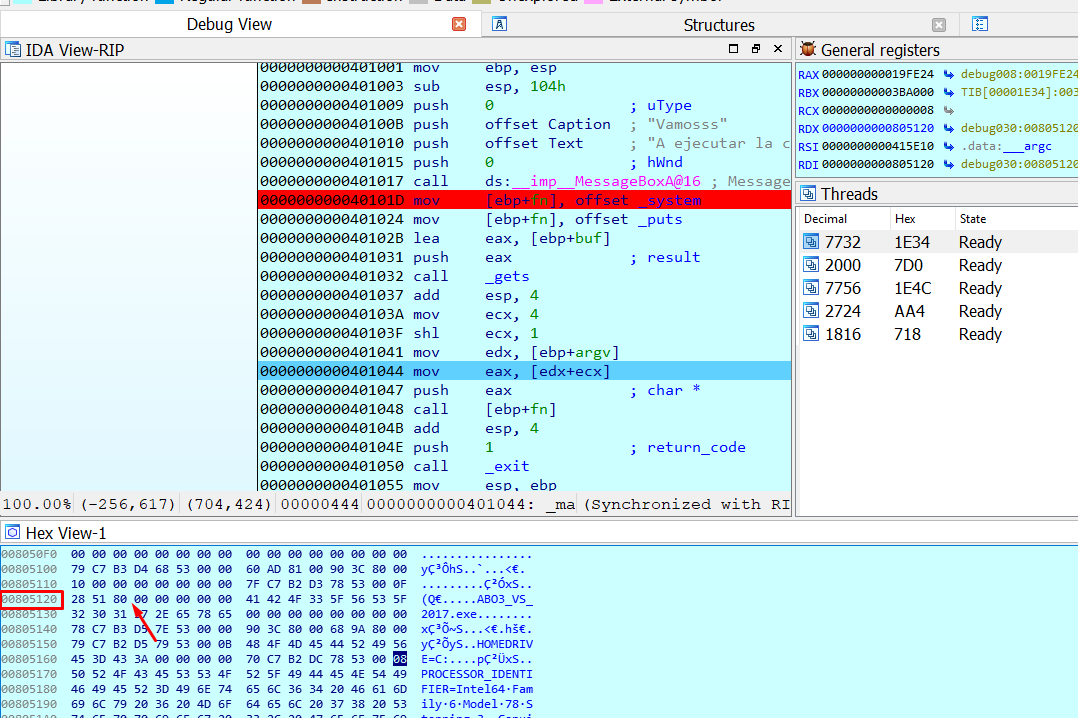
Que es equivalente a multiplicar ECX por 2.



Así que ECX valdrá 8 después de la multiplicación por 2.

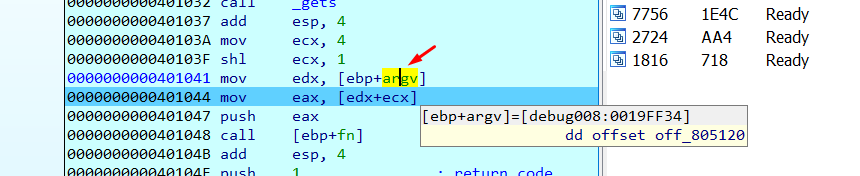


A EDX se mueve **argv,** ya vimos que es un array de punteros por lo tanto cada campo será de 4 bytes.



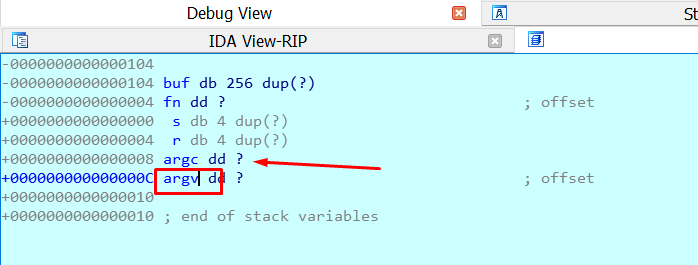
Como no pasamos argumentos **argv** solo tiene un campo, que es puntero al nombre del ejecutable.

La cantidad de campos de **argv** coincide con el valor de **argc,** que es la cantidad de argumentos pasados.

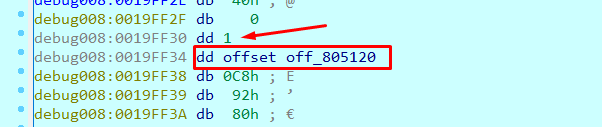


Como sabemos ambos también son argumentos de la funcion **main, argc** está arriba de **argv** por el orden en que se pasaron los argumentos en el stack al llamar a **main**

Si hacemos doble click en argv en la definición de la funcion, iremos a la r**epresentación estática del stack** y veremos arriba de **argv**, la variable **argc**.

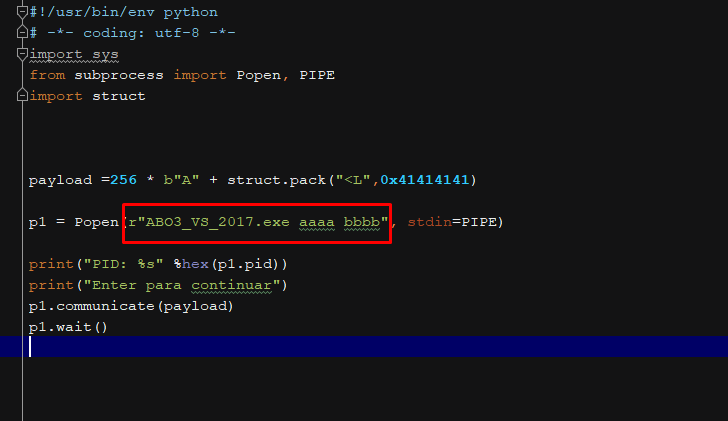


Si hago doble click en la variable **argv**, pero en el código me lleva a la dirección de memoria donde está la misma cuando estoy debuggeando.



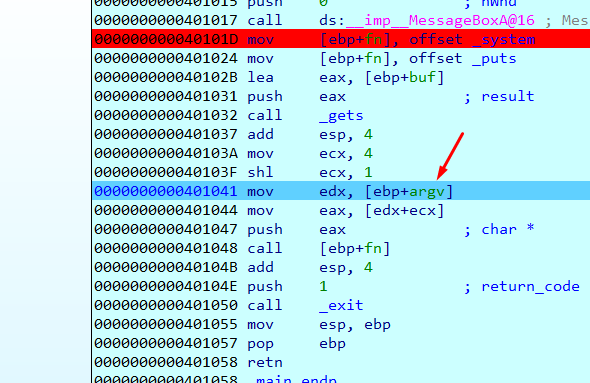
y arriba estará **argc** que es 1.

Asi que debere pasarle los argumentos en el script y lanzarlo nuevamente.

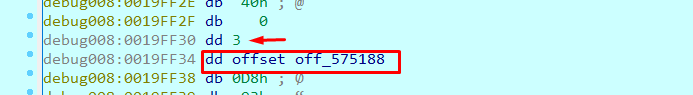


Los argumentos de consola se pasan a continuación del nombre del ejecutable, separados por espacios.

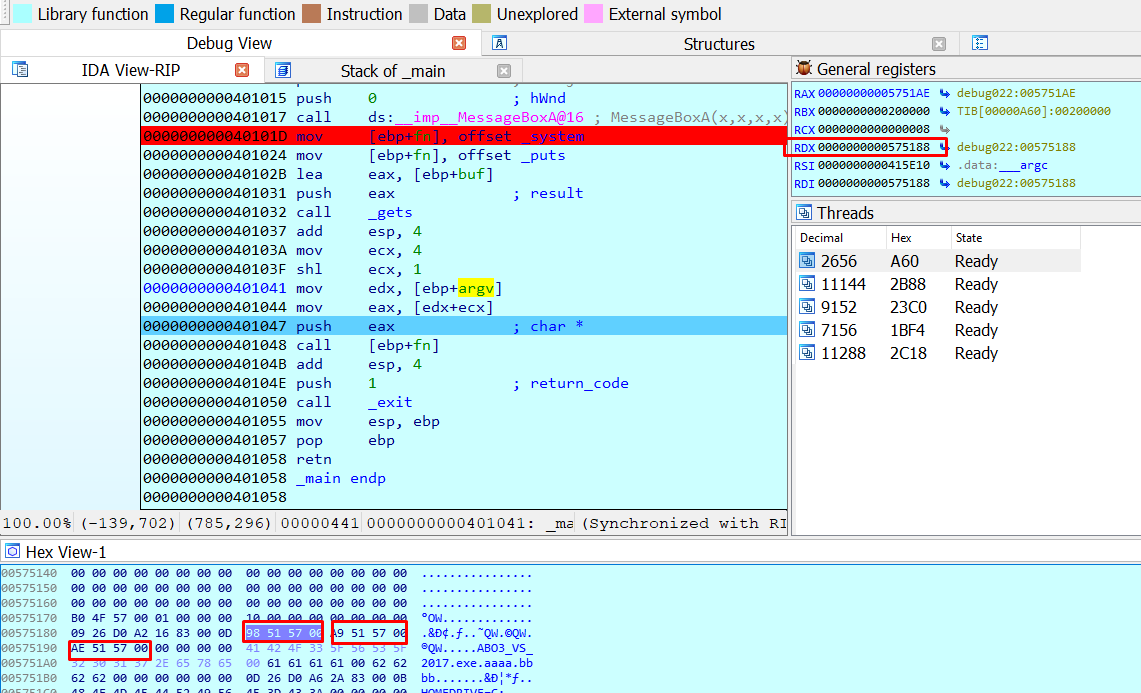
Ejecutemos de nuevo y lleguemos al mismo punto donde estábamos.



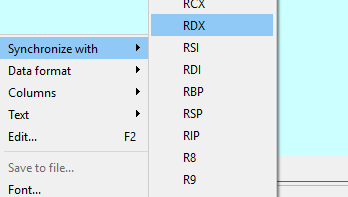
Hago doble click en **argv** en el código.



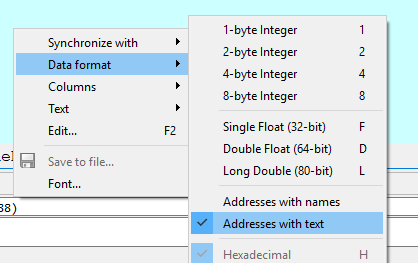
Ya los transforme a DWORD apretando D varias veces y veo que **argc** es 3 porque pase tres argumentos, así que ahora **argv** es un array de tres punteros a cada string argumento.



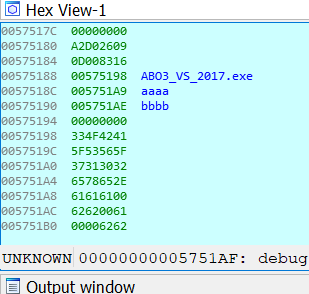
Si en el HEX VIEW hago CLICK DERECHO - SYNCRONIZE WITH RDX.



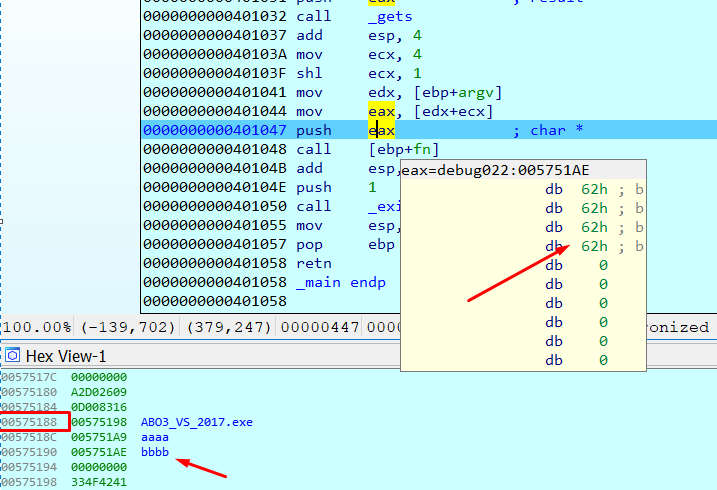
Y luego lo cambio a DATA FORMAT -ADDRESSES WITH TXT.



Puedo ver el array de punteros a strings, cada puntero apunta a un argumento, siendo el 0, el nombre del ejecutable, el 1 el primer argumento **aaaa** y el 2 el segundo argumento **bbbb**,

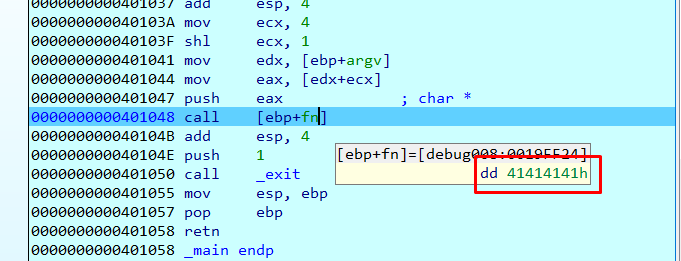


El tema es que al sumarle 8 a argv moverá a EAX el puntero al segundo argumento **bbbb**.

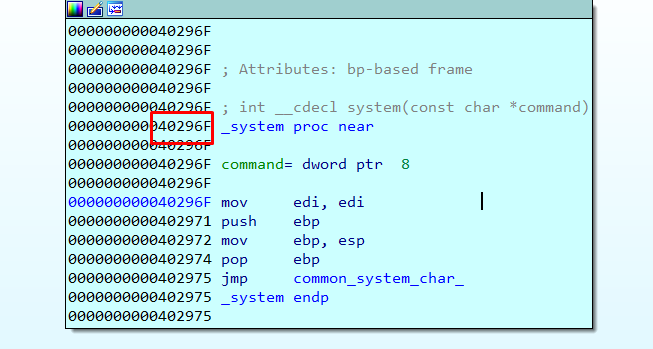


Y ese **bbbb** será el argumento de la funcion a la que saltaremos.

Cuya dirección será 0x41414141 si lo paso a DWORD.

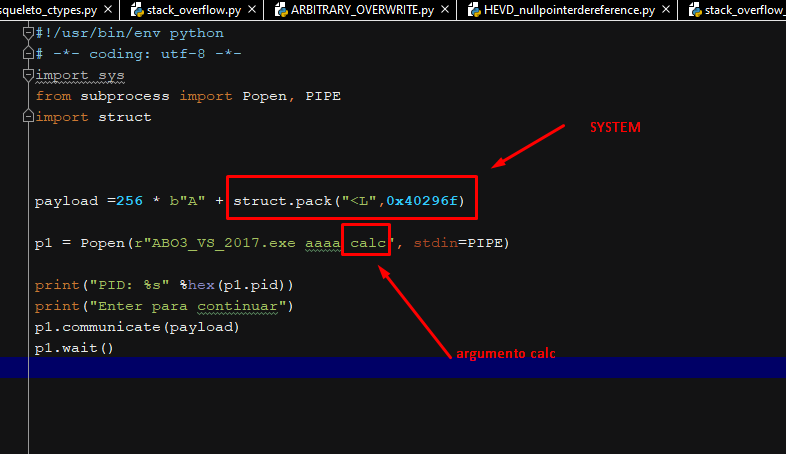


O sea que salto donde yo quiero y controlo el argumento, si en vez de pisar **fn** con 0x41414141, lo piso con la dirección de system que es fija y conocida pues como vimos está embebida en el ejecutable.

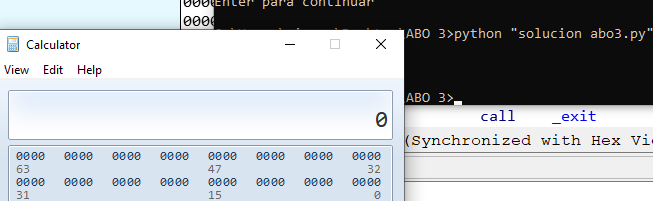


SCRIPT SOLUCION

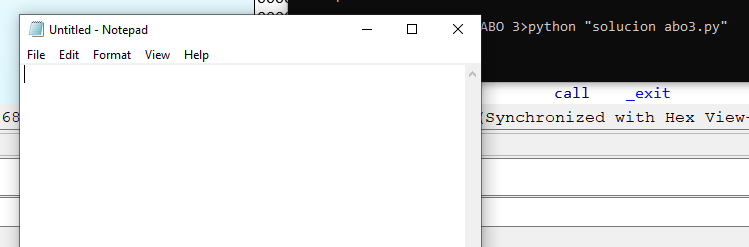
Y en vez del argumento **bbbb** le pongo la string **calc** para que ejecute la calculadora.



Si ejecuto el script se ejecutara la calculadora

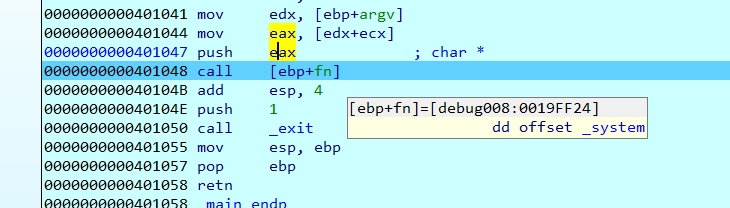


También puedo ejecutar otra archivo ejecutable, por ejemplo notepad.

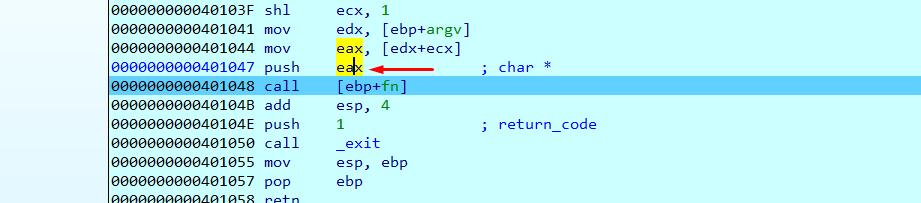


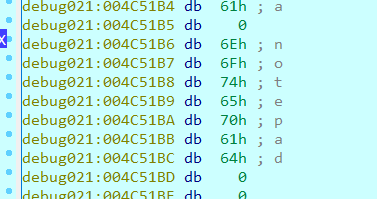
Si atacheo y paro en el salto a ejecutar.

Una vez que lo paso a DWORD veo que ejecuta system



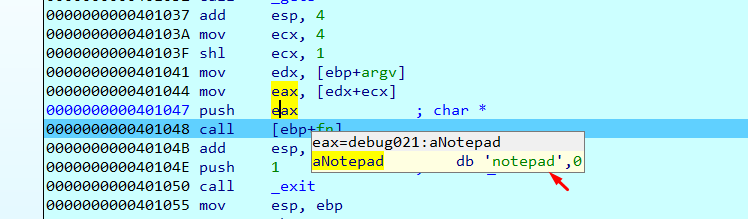
Y si hago doble click alli en EAX





Puedo transformarlo en string con la tecla A.





Bueno con eso hemos solucionado el ABO3 en IDA FREE la próxima parte el ABO4 en radare.

Hasta la parte 9

Ricardo Narvaja

31/1/2020