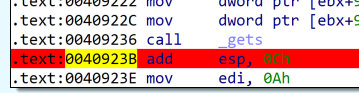
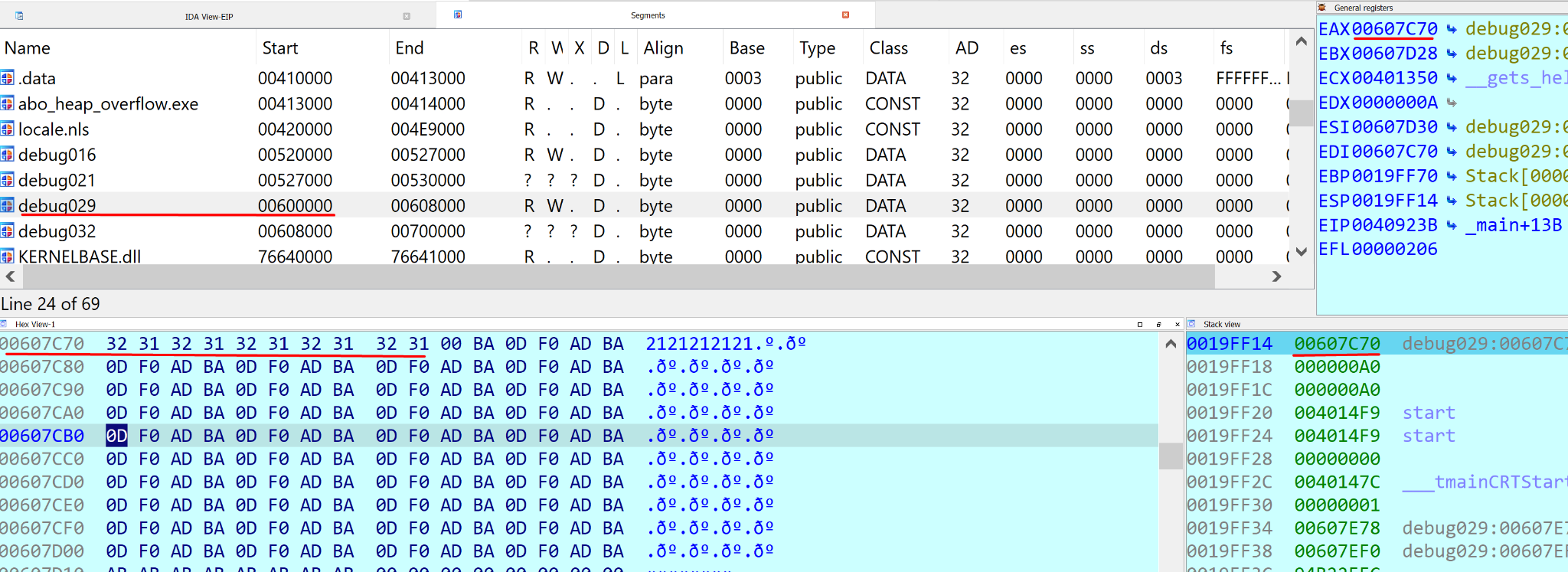
Vamos a abrir este **ABO** con el **IDA** y pondremos un **BP** justo despues de la función **get.**

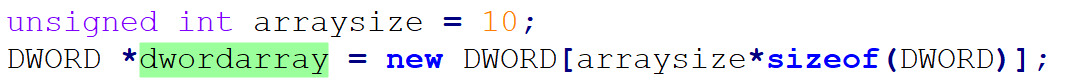
****

Lo ponemos a correr e ingresamos “2121212121”, damos un enter y caemos sobre el breakpoint.

Lo que ingresamos queda apuntado en **eax,** y al tope de la pila

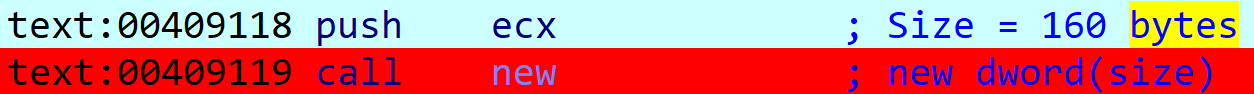


Se almacena en un espacio de memoria que fue reservado dinámicamente.



**Sizeof** retorna el número de **bytes** de una estructura, variable o tipo de datos. En este caso vemos un **sizeof(DWORD)** que devolvería **4 bytes**, a esto lo multiplica por **arraysize** que vale **10**, entonces el resultado final serían **40 bytes.**

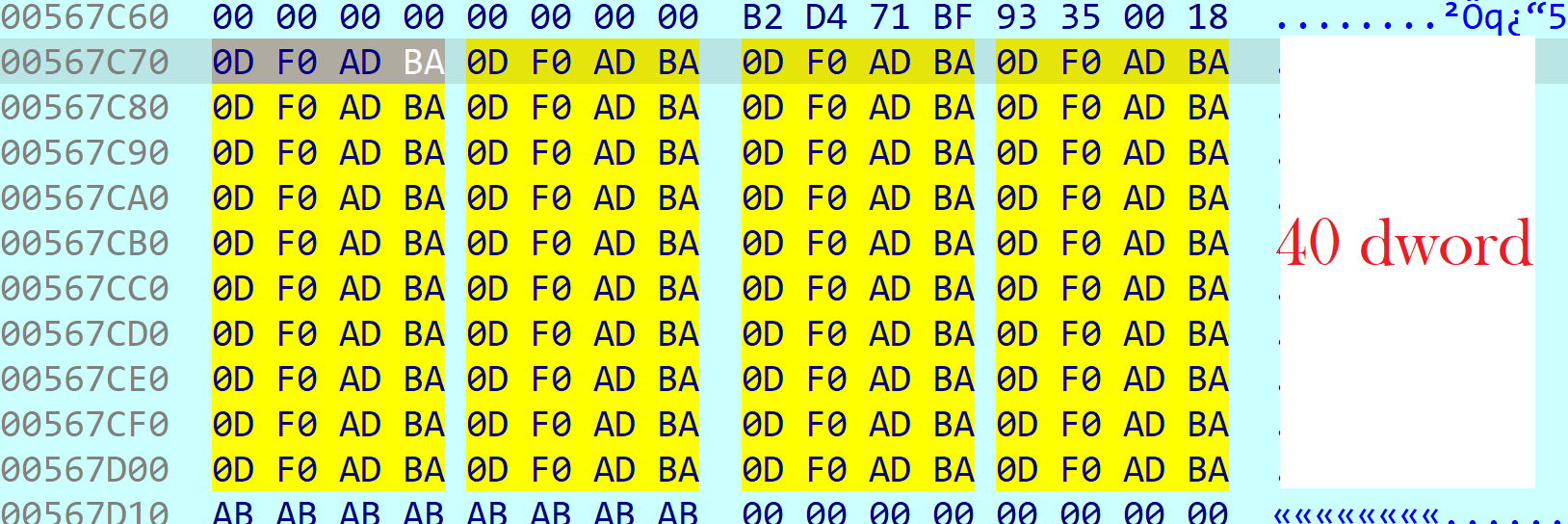
Estariamos reservando memoria para **40 DWORD,**  es decir **40 \* 4 bytes de un dword** = **160 bytes.**



Al ejecutar el **new** en **eax** nos va a quedar el puntero al espacio de memoria que se reservó. Como tenemos el codigo sabemos que se almacena en **\*dwordarray.**

El operador **new** sirve para reservar memoria dinámica, similar a la función **malloc**. Este operador permite crear un objeto de cualquier tipo, y devuelve un puntero al objeto creado.

La memoria reservada con **new** será válida hasta que se libere con **delete** o hasta el fin del programa, aunque es aconsejable liberar siempre la memoria reservada con **new** usando delete. Se considera una práctica muy sospechosa no hacerlo.



Luego se pide mas memoria

| somestruct \*structarray = new somestruct[arraysize]; |
| --- |



Ahí vemos que pide memoria para **160 bytes**, producto del resultado de multiplicar el **arraysize** con el total de bytes que tiene la estructura **someestruc**.

| typedef struct somestruct {  DWORD size;  DWORD location;  DWORD callback;  DWORD next; } somestruct; |
| --- |

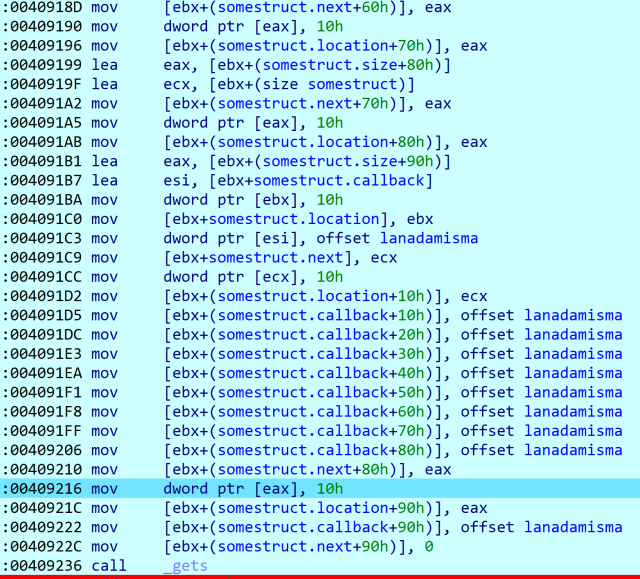
Hay **4 dwords**, osea son **16** bytes. **arraysize** vale **10** y al multiplicarlo por **16,** da como resultado **160.**

En esta segunda ocasión sabemos que almacena el puntero en la variable **\*structarray.**

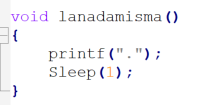
****

He creado esta estructura en **IDA**, no voy a explicar cómo crearlas pero una buena lectura donde lo explican es en el curso de Ricardo, C Y REVERSING (parte 7) por Ricnar.doc

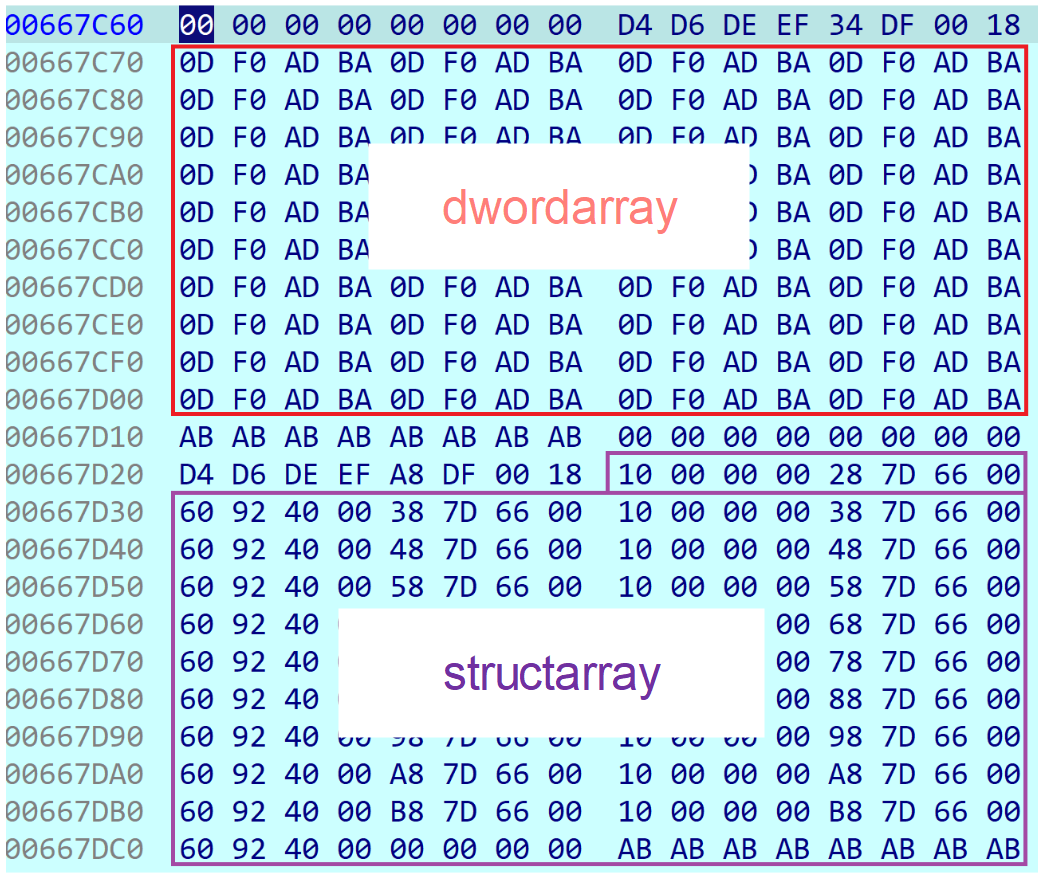
Luego sabemos que en **ebx** me quedo apuntando al comienzo de esta estructura, entonces nos paramos encima de por ejemplo **[ebx+4]** y presionamos la tecla **T**, elegimos nuestra estructura y le damos enter, y así con todas. Nos va a quedar algo asi:



Vemos que en **callback** recibe el puntero de una función llamada **lanadamisma**



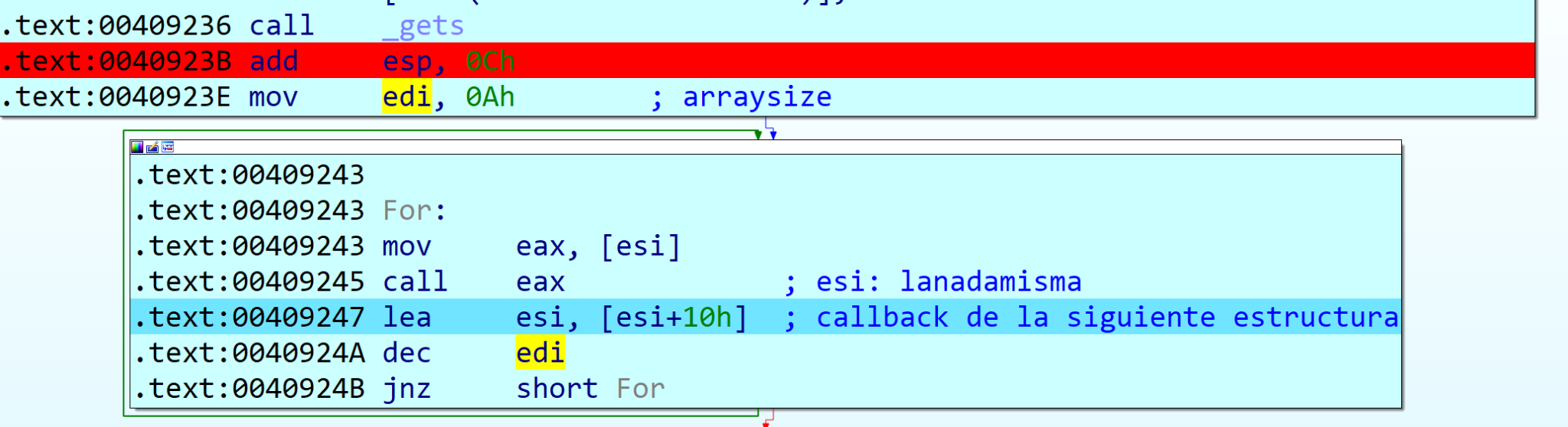
La estructura **structarray** es rellenada con datos como vemos en la imagen de arriba. Esta estructura se encuentra justo por debajo del array de dwords **dwordarray.**



El **dwordarray** se llenará con el **gets** con toda la fruta que le pasemos por teclado.

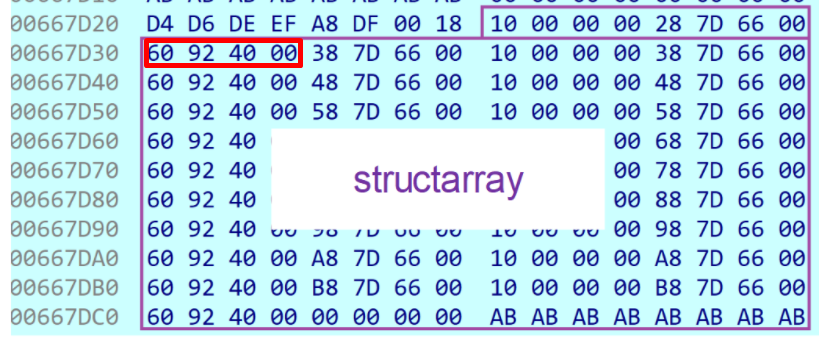
| gets((char \*)dwordarray); |
| --- |

Aquí abajo vemos que después obtiene la dirección de la función **lanadamisma** a través del **callback** que previamente se guardó en **ESI.**

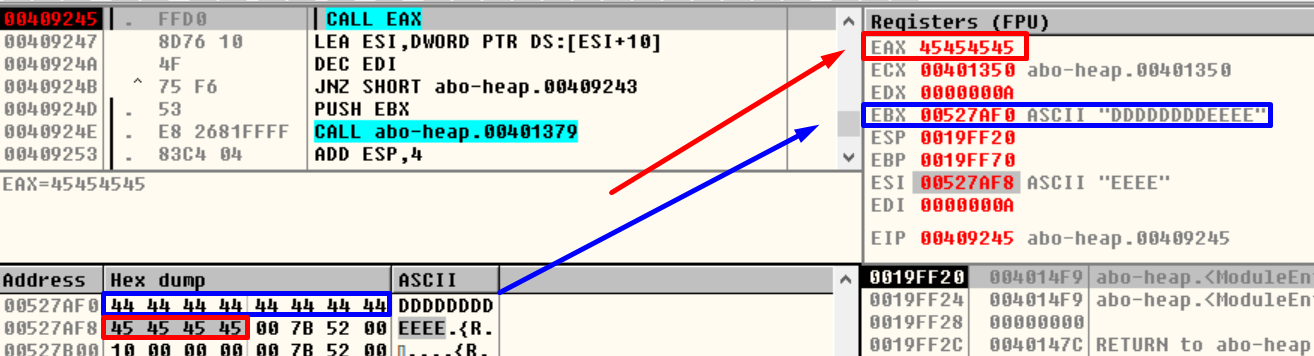


Okey, nuestro objetivo será pisar el contenido del **callback** y poner una dirección que apunte a nuestro **shellcode** para ejecutar la calculadora.

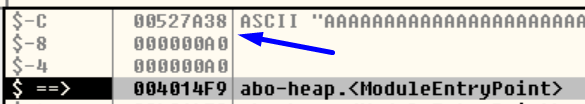
Entonces, sabemos que **dwordarray** tiene un tamaño de **160** **bytes**, si vemos la imagen arriba nos daremos cuenta que entre medio del **dwordarray** y **structarray** hay **24 bytes basuras,** y hasta el **callback** de la primera estructura en el array hay **8 bytes mas.**



Vamos a probar, ingresemos 160 + 24 + 8 dígitos por teclado después del **gets.**

****

Estando parado sobre el **call eax** si vemos los registros en **eax** tendremos el puntero hacia donde queremos saltar, en **ebx** vemos que quedó apuntando a la **shellcode** pero solo tenemos **8 bytes.**

****

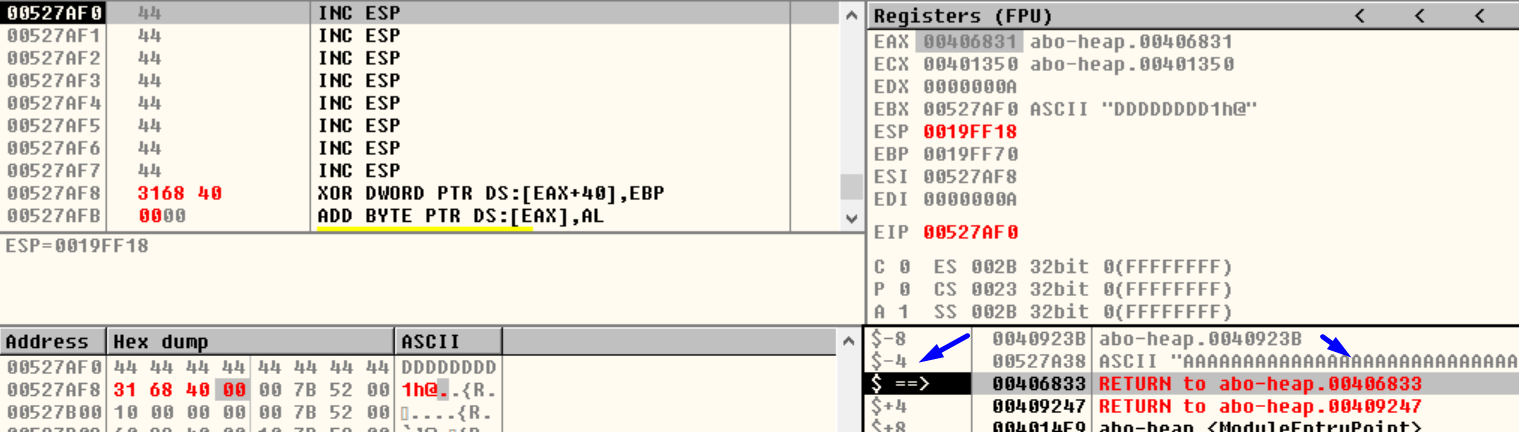
Vemos que en **ESP - 0xC**  está el puntero al comienzo de nuestra **shellcode.**

Bueno, lo primero que necesitamos hacer es saltar a **ebx**, para ello podemos usar un **call ebx.** Por ejemplo aquí tenemos uno en **406831**

****

Ahora necesitamos saltar a este **call ebx**, para ello usaremos el **call eax** en donde estamos parados, y haremos que **eax** apunte al **call ebx,** entonces en el final del **shellcode** tendremos que poner este puntero.

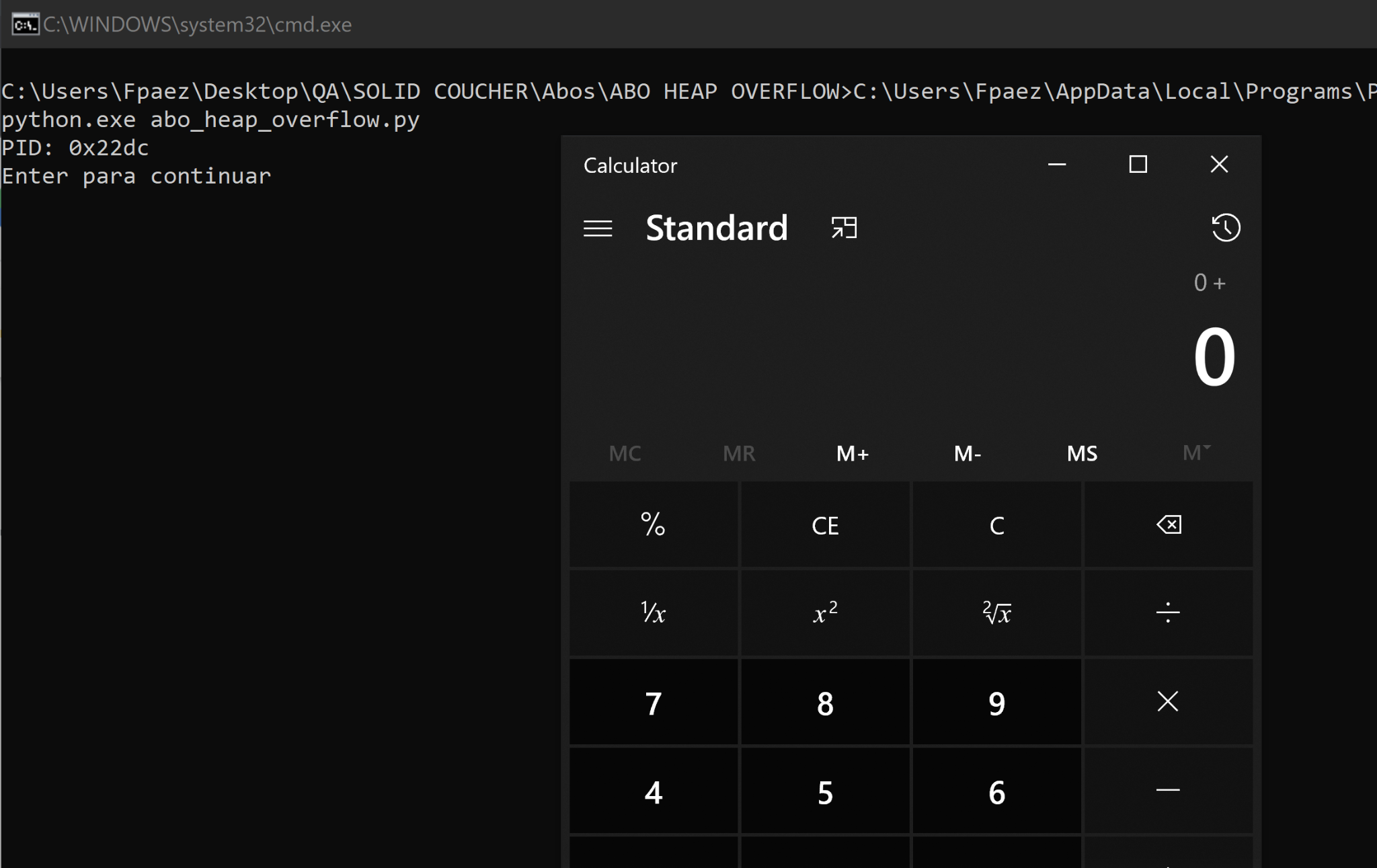
Una vez que entramos al **call ebx** llegaremos a nuestra **shellcode**. En la pila podemos ver en este instante que en **esp-4** quedó apuntando al comienzo de nuestra **shellcode.**



Podemos usar estos **8 bytes** que tenemos disponibles y escribir un

| sub esp,4 retn |
| --- |

Con la primera línea quedamos parados justo sobre la **shellcode** en la pila, y con el **retn** saltamos a ella.



Referencias:

* [Función sizeof](http://www.it.uc3m.es/pbasanta/asng/course_notes/ch06s04.html)
* [Función sizeof(dword)](http://download.mikroe.com/documents/compilers/mikropascal/pic/help/sizeof_operator.htm)
* [Función new](https://www.zator.com/Cpp/E4_9_20.htm)
* [Función new](http://conclase.net/c/curso/cap13)
* C Y REVERSING (parte 7) por Ricnar.doc