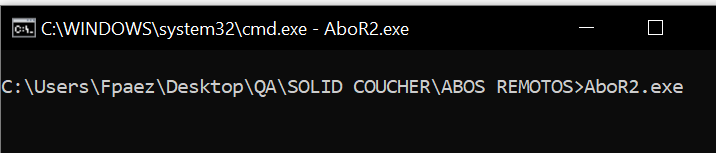
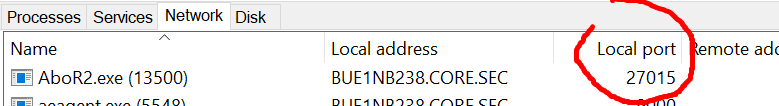
Llegamos al segundo nivel de los abos remotos, y ahora estamos con el **Abor2**

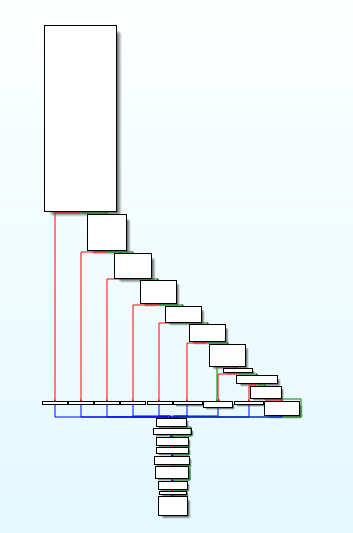
Si lo ponemos a correr, lo primero que noto a diferencia con el primer nivel es que en este no nos imprime ningún mensaje en la consola.



Si nos fijamos con **Proccess Hacker** vemos que escucha en el **puerto 27015**

****

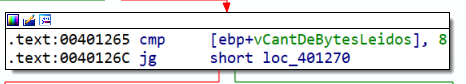
Ahora miremos con el **IDA,** este es todo el arbolito



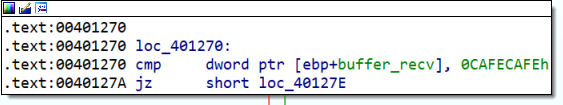
Aquí lee con **recv** los datos que le mandemos al puerto



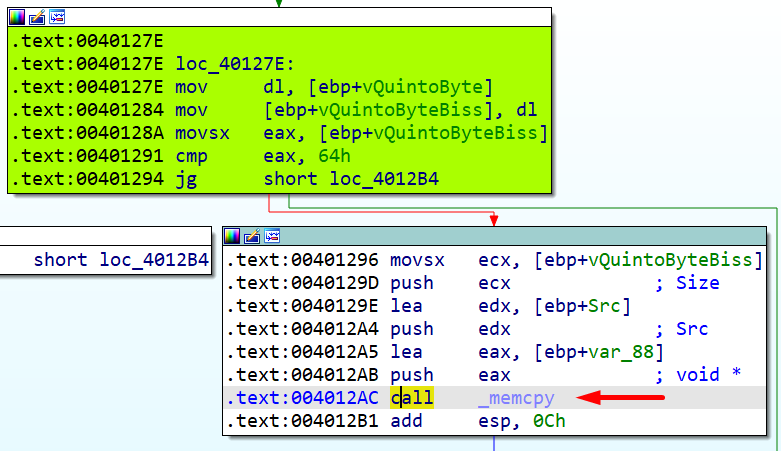
La cantidad de bytes leídos tiene que ser mayor a **8**

****

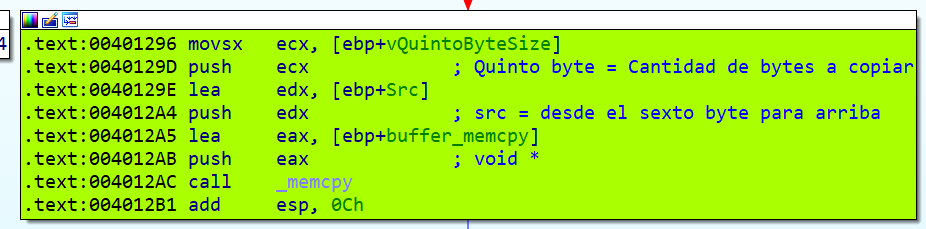
Los primeros 4 bytes tienen que ser **0xCAFECAFE**

****

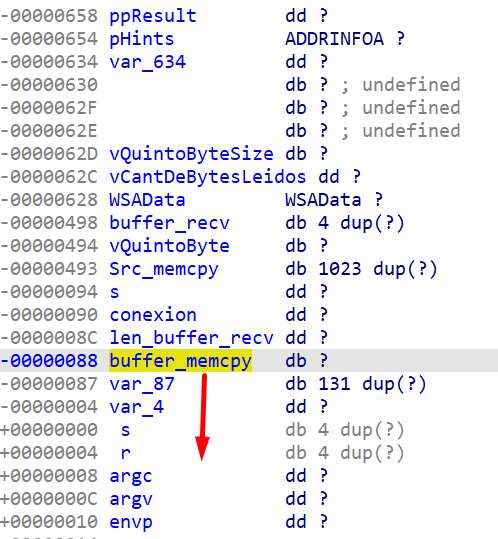
El quinto byte debe ser menor a **0x64,** de esta forma saltaremos al bloque que tiene la función **memcpy**

****

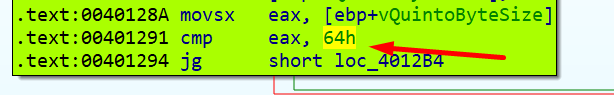
Observamos que al **quinto byte** se lo usa como el **size** en la función **memcpy.** Copiara a partir del **sexto byte**.



Vamos a pisar todas las variables hasta el **return address**

****

Tenemos un problema, desde el **buffer\_memcpy** hasta el **return address** hay **144 bytes.** El problema nace en la comparación del quinto byte, que debe ser menor a **100 bytes**

****

Así que solo se copiaran **100 bytes** y no los **144 bytes** que necesitamos para llegar y pisar el **return address.**

Como dijimos este **quinto byte** se usa como **size** en la función **memcpy**

| void \*memcpy(void \*dest, const void \* src, size\_t n) |
| --- |

ahí vemos que **n**  es de tipo **size\_t**, la cual es un tipo **entero sin signo.**

Entonces Si **n** es un valor **negativo**, el signo no se tendrá en cuenta. Por ejemplo, si argumentamos con **-1**, el valor operativo sería **255 (0xFF)**.

La comparación que se hace con este valor, la cual tiene que ser < **0x64**, sí se hace

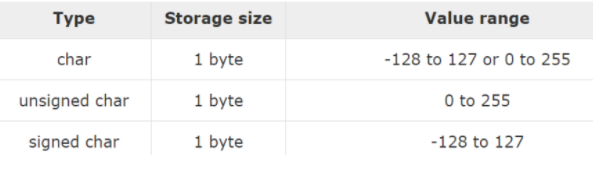
teniendo en cuenta el signo. Por lo que si el valor es -1, pasaremos la

comprobación, pues **-1** < **100**.

Concluyendo. Usando **valores negativos**, pasaremos la comprobación del tamaño y

además nos permitiría asignar a **'Size'** valores superiores a **0x64**, lo cual es justo lo que necesitamos para provocar el desbordamiento que nos permita pisar y controlar el **'ret'**

Repasemos un poco lo de **unsigned**, en el código que tenemos se compara un solo byte, el quinto.

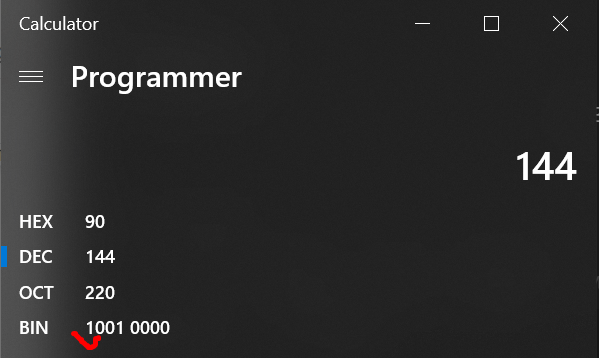


Para representar un número **signed** consiste en asignar:

**Un bit** para **representar el signo**. Ese **bit** a menudo es el **bit más significativo** y, por convención: **un 0 denota un número positivo**, y un **1 denota un número negativo**.

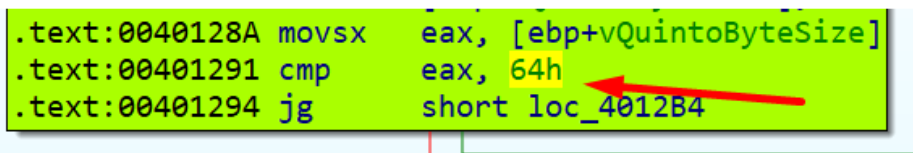
Por lo tanto, pasando como **size** con un valor de **0x90** sería más que suficiente,

pues esto nos daría para sobrescribir **144 bytes**.



Como se ve en binario al comienzo tiene un 1 (uno) lo que significa que es un número negativo.

Entonces al llegar a la comparación lo tomará como un **número negativo**, que será menor a **0x64,**  por lo tanto conseguiremos que no se produzca el salto.

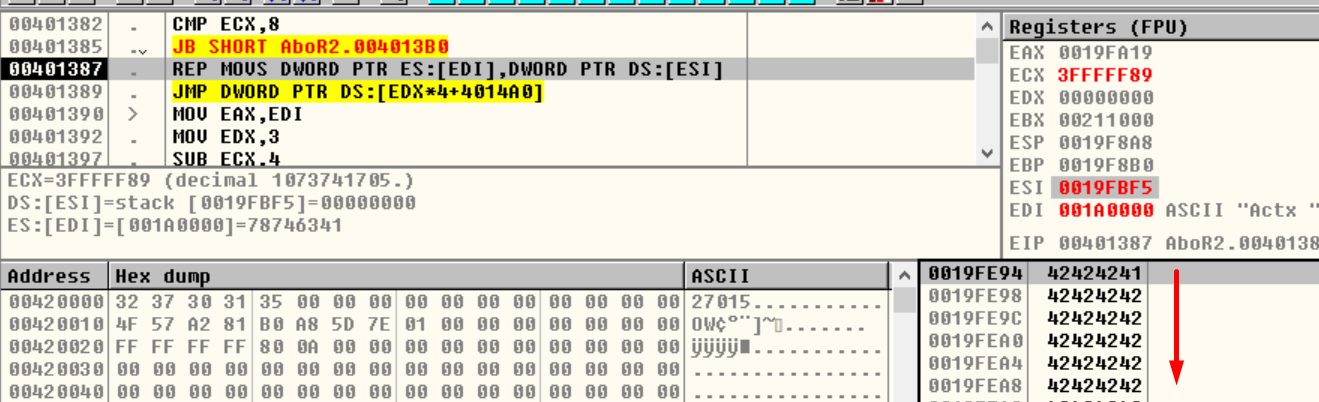


Después al llegar al **memcpy** lo tomara como un **número positivo alto**, y copiara toda esa cantidad de bytes al buffer.

Una **longitud negativa** se convierte en un tipo **size\_t** para la llamada a **memcpy**, que como se sabe, se convierte en un **valor grande sin signo**.

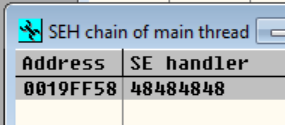
Yo le pase **0x90**, que serían **144 bytes**, como el primer bit de este número es un **1,** entonces es un **número negativo**, pero a la función **memcpy** no le importa esto y lo toma como **positivo.**

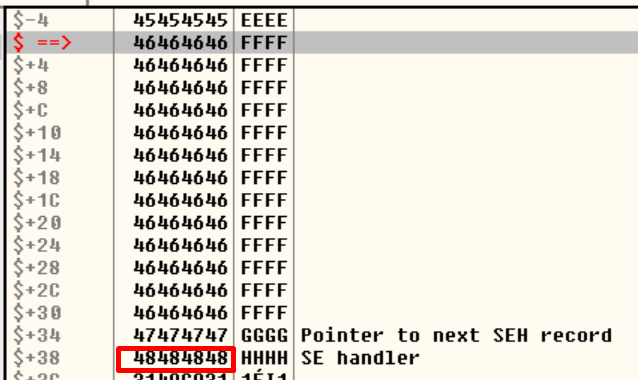
Aquí vemos que ya estamos dentro del **memcpy** justo en el momento donde hace la copia, y se produce un **“access violation”** porque el valor que le pasamos, que era un **negativo**, se ha convertido en un **número extremadamente muy alto**, y al copiar rebasa la pila, provocando el acceso invalido.





Lo que podemos hacer ahora es pisar el **SEH** con una dirección a la que saltara a ejecutar.



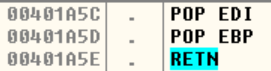


Una vez que saltemos a **0x48484848** vemos que en **ESP+8** se encuentra la dirección del stack en donde se encuentra el **Pointer to next seh record.**

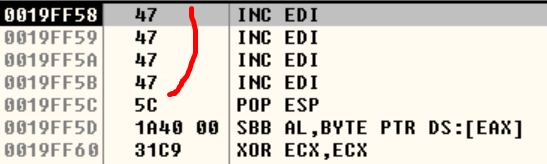


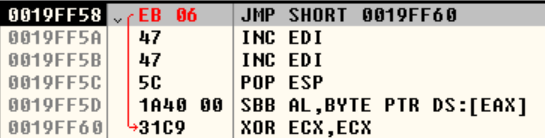


Y como se ve ahí hay **4 bytes** que nosotros controlamos, por lo tanto podemos sacar los dos primeros valores con dos **pop** y luego saltar a esos **4 bytes** con el **ret.**

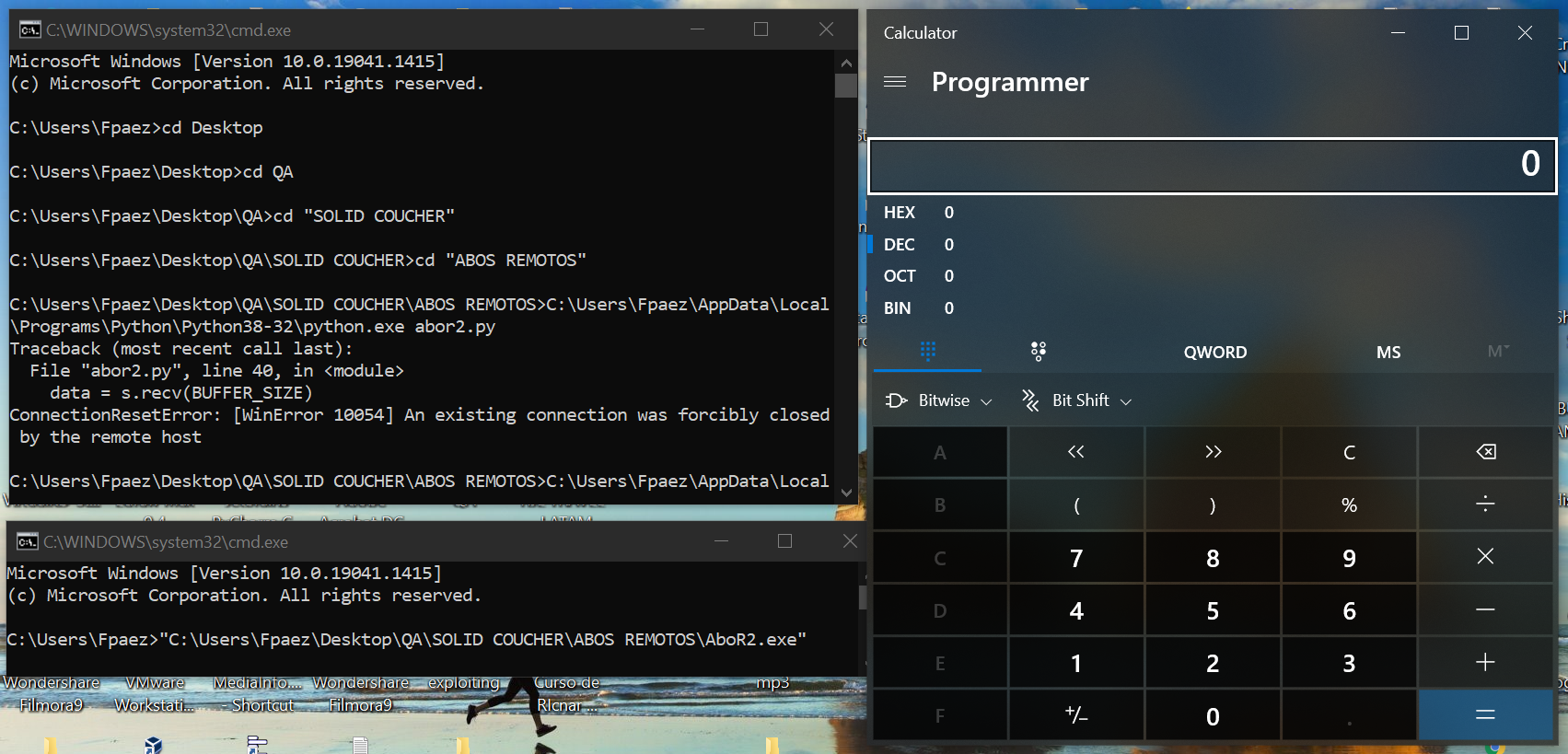


Aquí ya estamos sobre esos **4 bytes**, y los siguientes **4 bytes** corresponden al **seh**, por lo que debemos evitar pisarlo, así que ensamblamos un **JMP** para que salte justo después de estos otros 4 bytes.

****

****

y ya después de este **jmp** tenemos el **payload** que ejecuta la calculadora. Después de este le agregue un **ExitProcess** ya que se ejecutaba varias veces la calculadora.

****

Referencias:

* <https://www.tutorialspoint.com/c_standard_library/c_function_memcpy.htm>
* <https://en.cppreference.com/w/c/types/size_t>
* [JG (signed)](http://unixwiz.net/techtips/x86-jumps.html)
* [size\_t](https://www.informit.com/articles/article.aspx?p=686170&seqNum=6)
* <https://blog.feabhas.com/2014/10/vulnerabilities-in-c-when-integers-go-bad/>