INTRODUCCION AL CRACKING CON OLLYDBG PARTE 40

Antes de continuar con la tabla IAT del pelock, y los antidumps, en esta parte veremos dos temas, uno que quedo en el tintero y otro que es una gran ayuda para continuar con el tpelock o cualquier otro programa similar que anula o detecta los HBP ya veremos.

El primer tema es que en la pate 39 si descargaron el tutorial apenas salio, no se dieron cuenta, pero luego fue modificado y aparecia un mensaje de texto que decia.

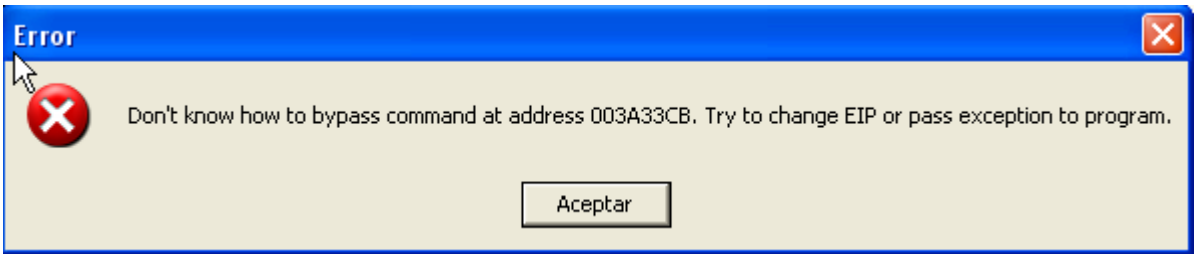
Para realizar este tute se debe utilizar el ollydbg parcheado 4 que esta dentro del rar actual, es un OLLYDBG parcheado que se debe colocar en la misma carpeta que el OLLYDBG.exe, porque el OLLYDBG normal tiene un bug al manejar las ILLEGAL INSTRUCTION, que hace que salga un cartel si o si cuando le pones la tilde en debugging options-exceptions para saltearla, con el parcheado 4, el bug esta reparado, al momento de hacer el tute lo habia olvidado, pero Solid me recordo al no poderlo hacerlo al tute, el problema y al pasarle el parcheado 4 para que ponga en la carpeta del OLLYDBG y usarlo, no tiene ningun problema.

Ricardo Narvaja

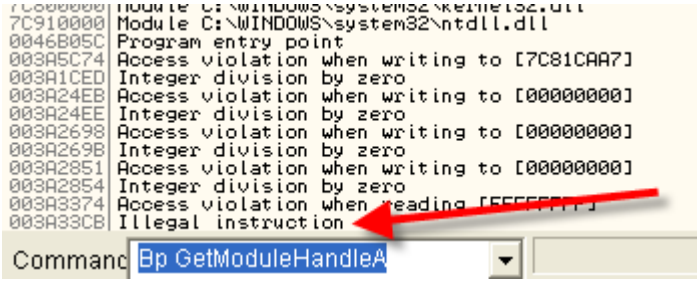
Y traia un OLLYDBG modificado llamado PARCHEADO 4 que se coloca en la misma carpeta del OLLYDBG.exe y usando este ultimo sirve mejor para hacer el tutorial de PELOCK.

Lo que queria explicar como primer punto que tiene de especial este parcheado 4.

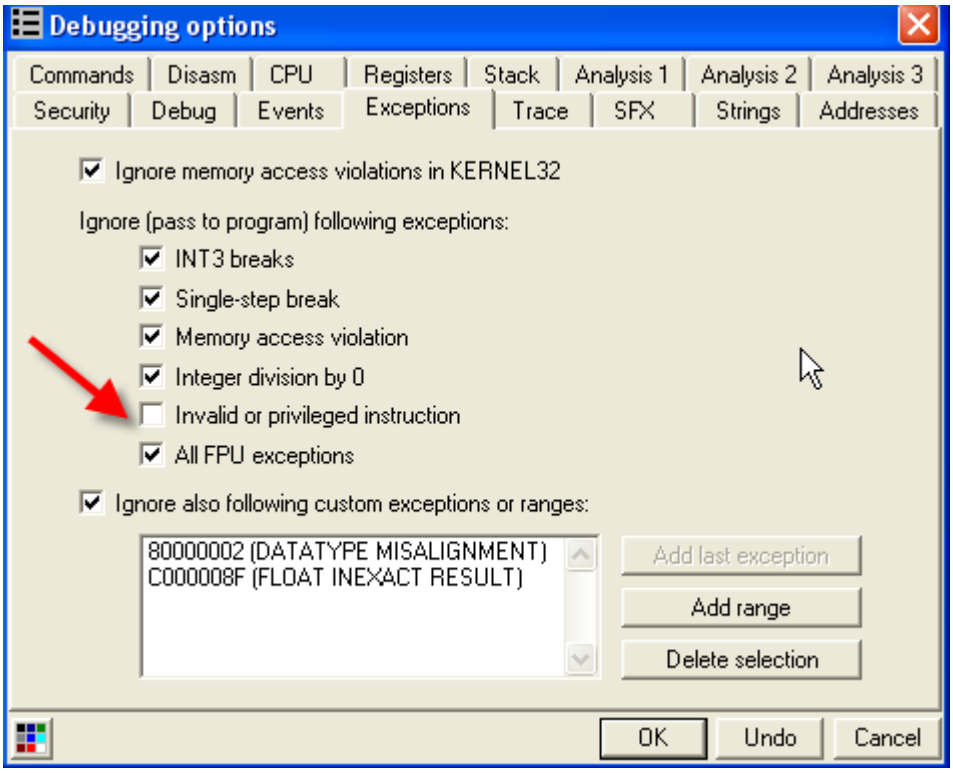
Bueno el OLLYDBG posee un bug al manejar las excepciones ILLEGAL INSTRUCTION;
Si en un OLLYDBG normal (sea renombrado o no) con los plugins para ocultarlo, marcamos todas las tildes en DEBUGGING OPTIONS-EXCEPTIONS y damos RUN.



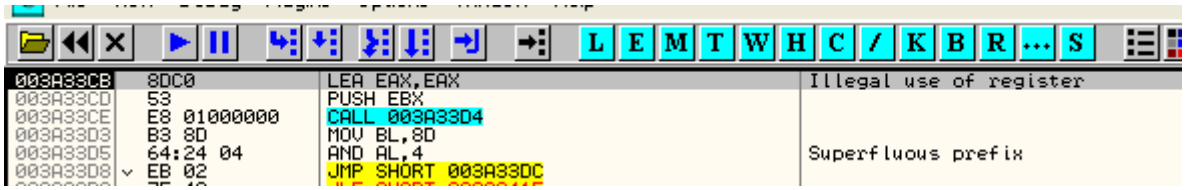
Obtenemos este error, mientras que en el parcheado 4, corre perfectamente que ocurre aquí veamos en el momento que aparece el cartel de error, lo aceptamos y miramos el LOG del OLLYDBG.exe normal.



Vemos que la ultima excepcion antes de que salga el cartel es una ILLEGAL INSTRUCTION, ahora vayamos a debugging options- exceptions y quitemos la tilde de ese tipo de excepcion.



Esa es la tilde que corresponde a esa excepcion, la quitamos y arrancamos el pelock y damos RUN.

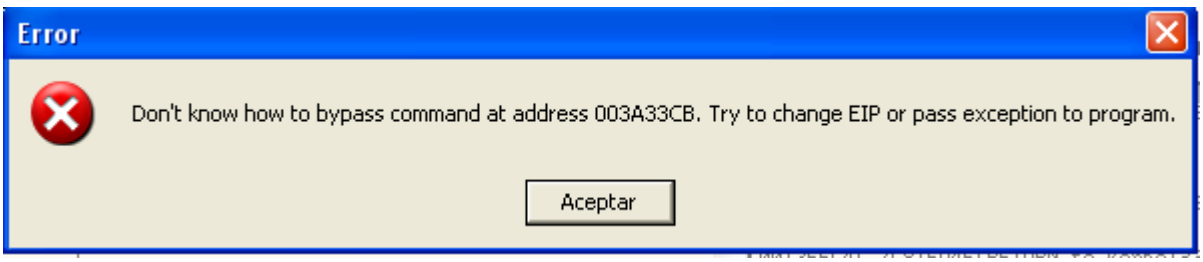


Para en la excepcion y si la pasamos con SHIFT +f9, no hay problema para en dos oportunidades mas en excepciones de este tipo y arranca el pelock.

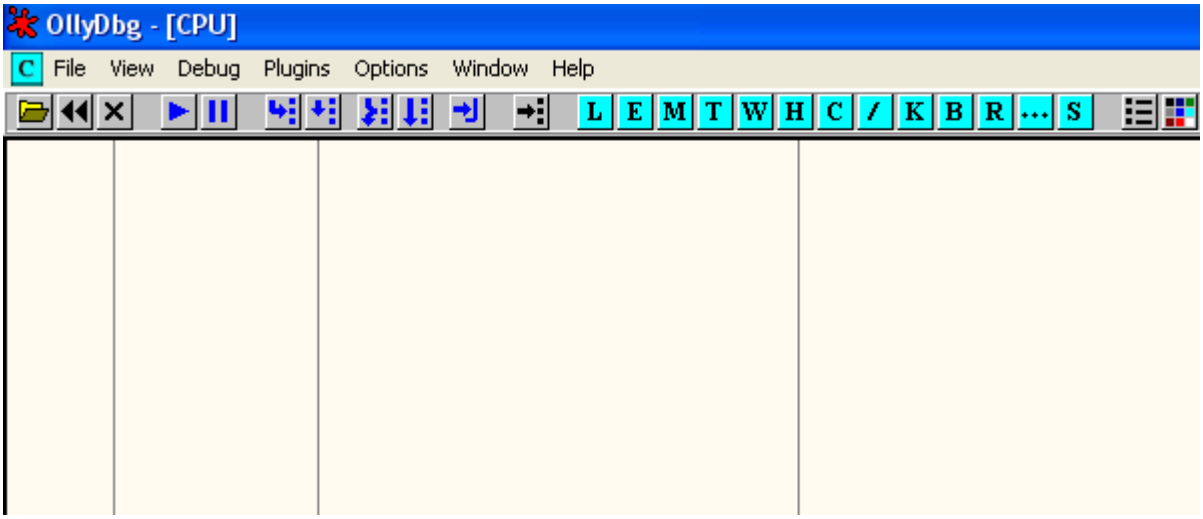
Eso quiere decir que OLLYDBG tiene un problema cuando le pones la tilde para que pase esa excepcion automaticamente.

Si solo fuera apretar SHIFT + f9 y todo se arregla no habra problema, pero alertados de este bug muchos programadores generaban 200 o 300 excepciones de este tipo las cuales habia que pasar todas a mano con SHIFT + f9 lo cual era pesado al menos.

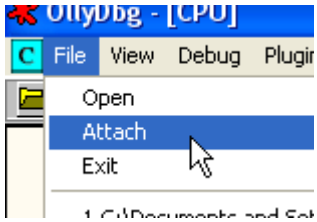
Ahora volvamos a colocarle la tilde y hagamos que aparezca el cartle de error nuevamente.

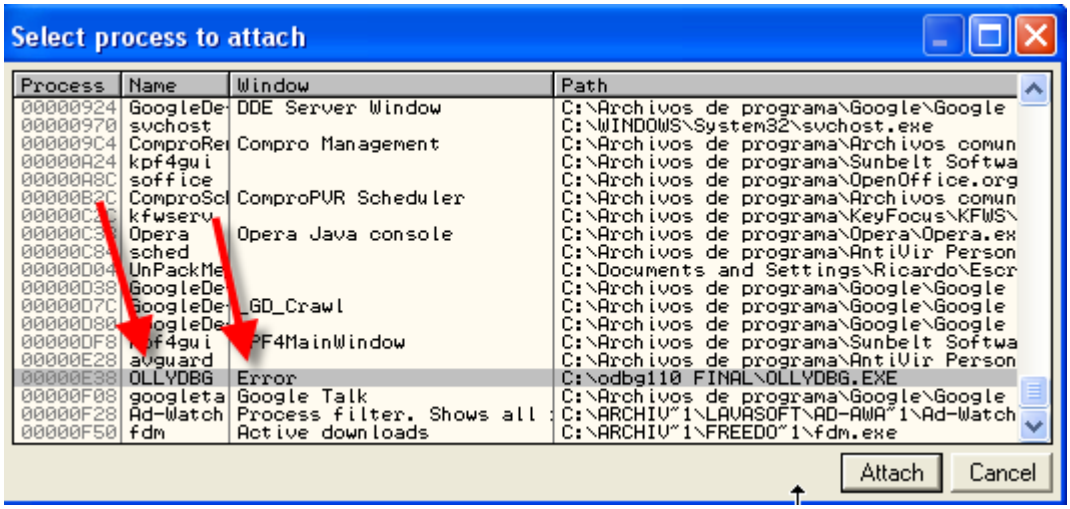


Al mejor estilo cracker, lo que haremos sera crackear el OLLYDBG para arreglarle el bug, para lo cual abrimos un segundo OLLYDBG vacio, con el cual crackearemos al que esta detenido en el cartel de error.

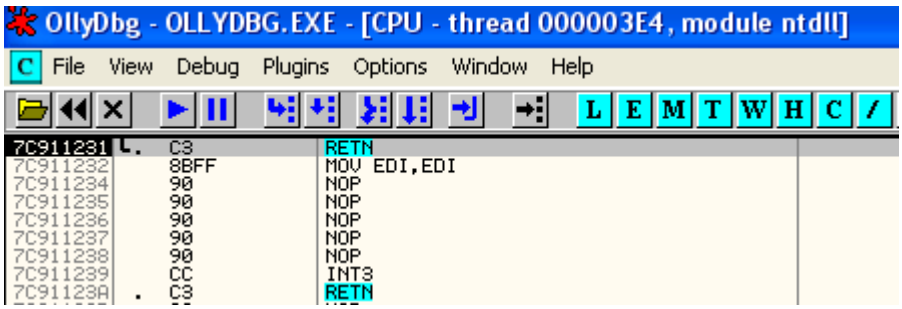


En este OLLYDBG vacio iremos a FILE -ATTACH para atachear y debuggear al otro OLLYDBG que este detenido en el cartel de error.



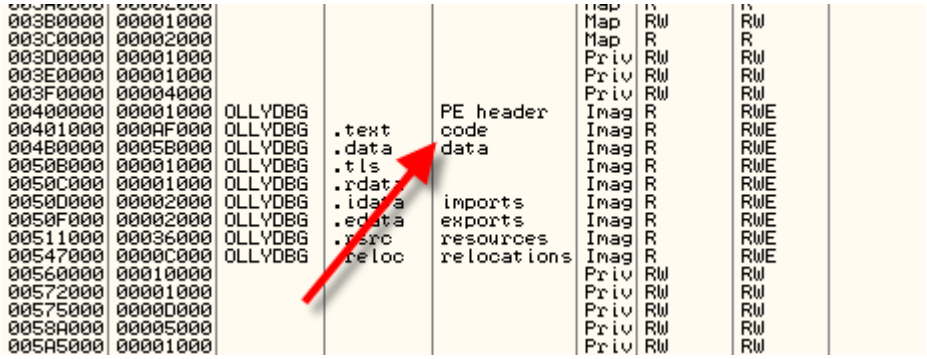


En la ventana que se abre buscamos el proceso OLLYDBG.exe cuya ventana o WINDOW tiene el eror como muestra alli, y apretamos attach.

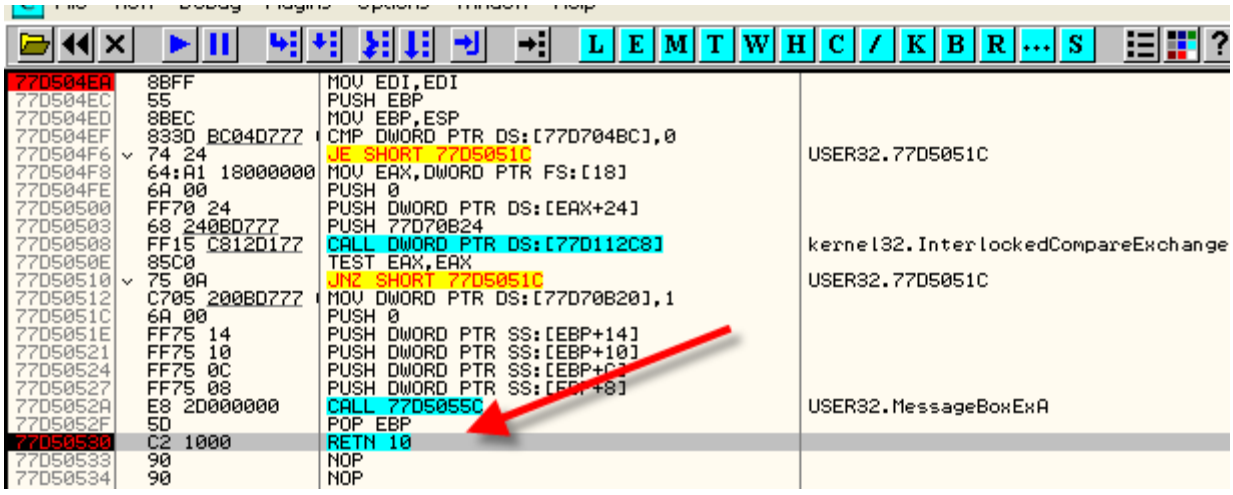


Alli paro ahora damos RUN

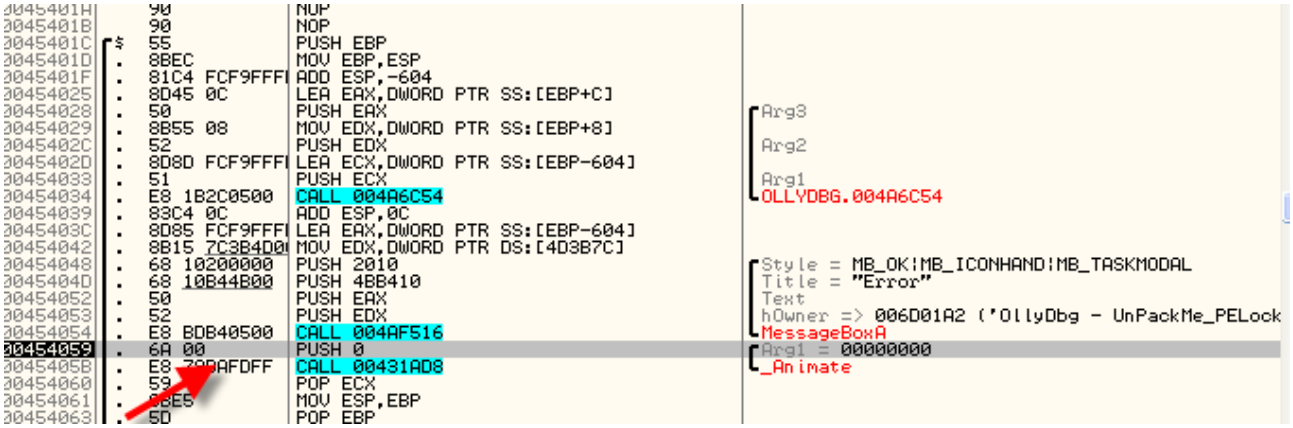
Ahora miraremos el mapa de memoria.



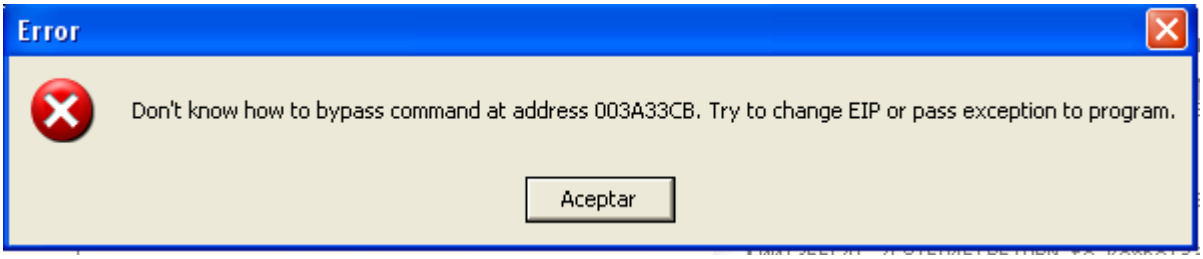
Por supuesto la victima es el otro OLLYDBG cuyas secciones vemos alli, lo que haremos sera colocar un Bp MessageBoxA



Pero ese no es tan importante solo nos sirve como guia para poner el BP en el RET de la api MessageBoxA, y al aceptar el error, parara en el RET al volver de dicho cartel.

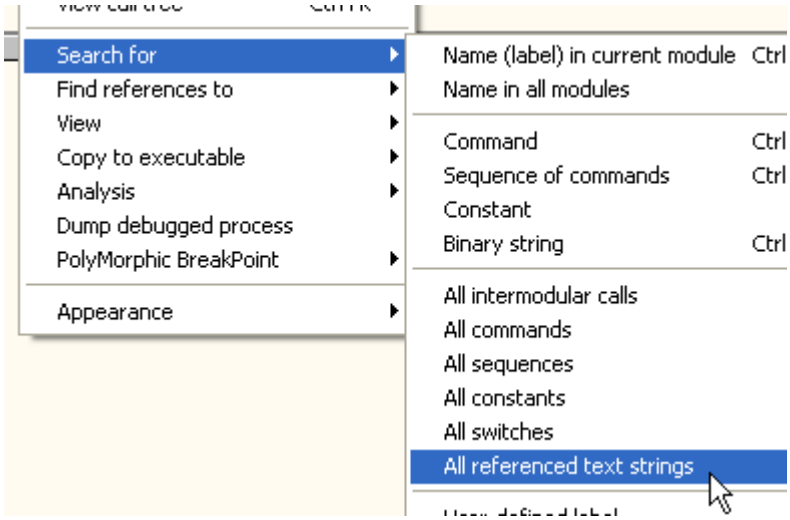


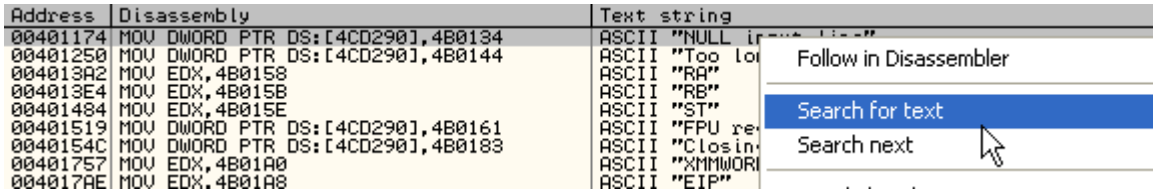
Alli vemos cuando retornamos al programa de la api, al apretar f7.



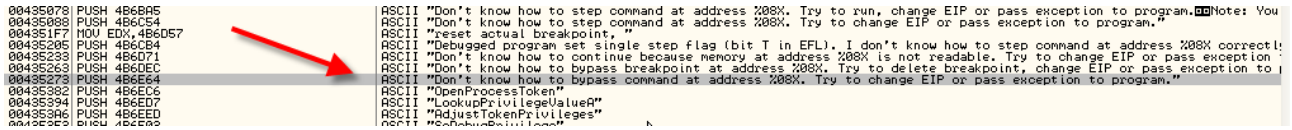
Si recordamos el texto del cartel decia lo que vemos arriba.

Si buscamos entre las strings del OLLYDBG

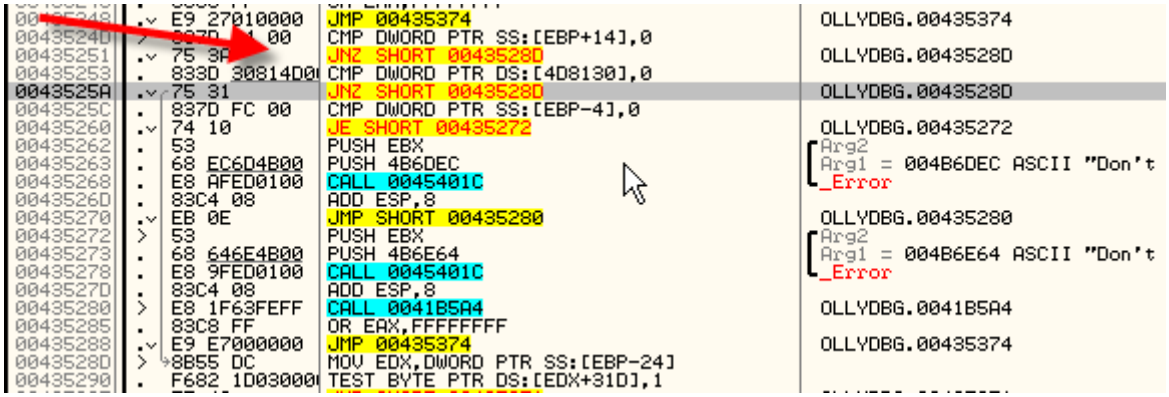




y alli buscamos la palabra bypass que esta en el mensaje.

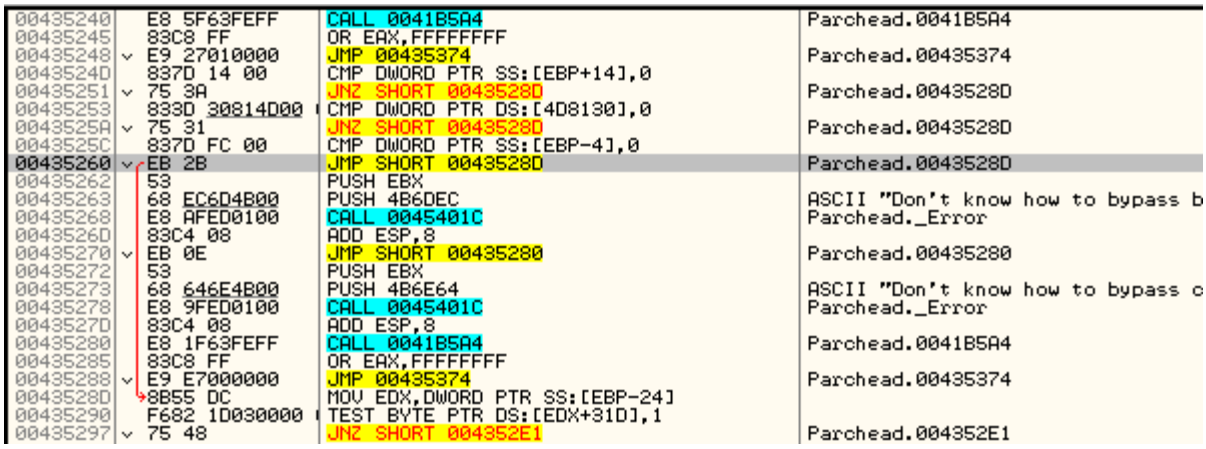


Vemos que esa es la string correcta, hagamos doble click para ir al lugar donde esta la misma en el listado.



Vemos que cualquiera de esos JNZ evita que salga el cartel de error podriamos cambialo por JMP y tendriamos nuestro programa parcheado, la guardar los cambios definituvos en memoria con COPY TO EXECUTABLE y SAVE FILE.

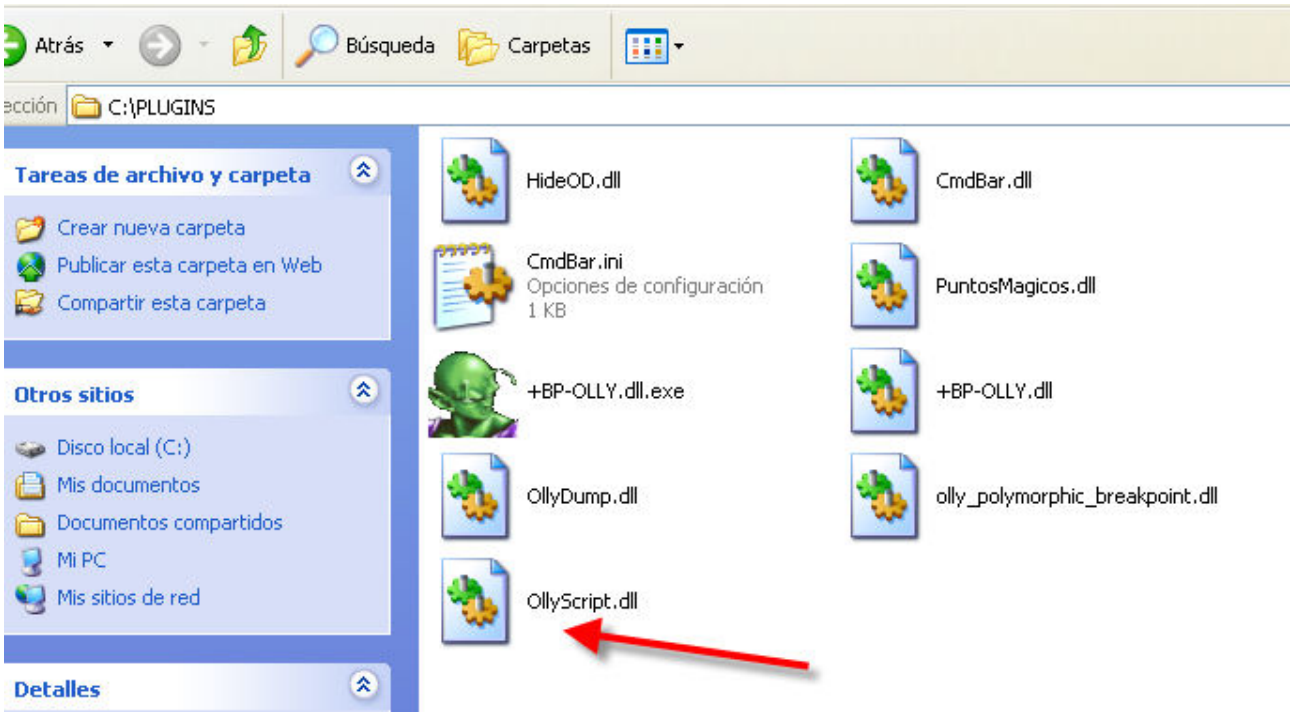
Abramos el parcheado 4 en OLLYDBG para mirarlo y veamos esta misma zona a ver como es.



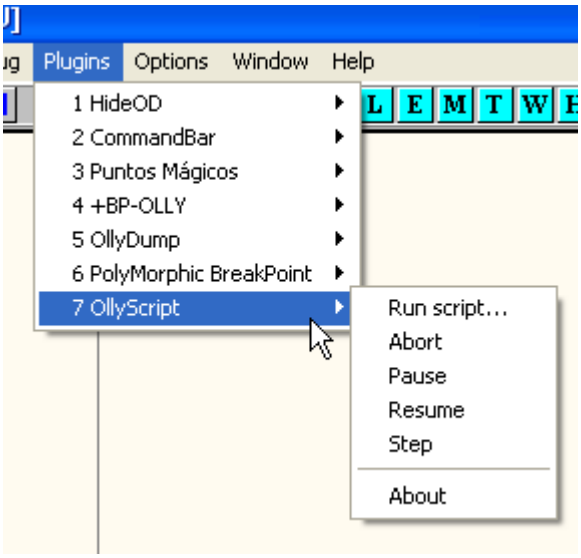
Vemos que en este directamente en vez de parchear los saltos que de cualquier manera nos llevan a saltear el cartel, lo que hice en su momento, fue cambiar el ultimo salto justo antes del cartel y cambiarlo a JMP 43528d, por si acaso salte de algun otro lugar siempre evite el cartel por cualquier camino,

Bueno esto es todo el secreto del parcheado 4 y porque no tiene el bug de ILLEGAL INSTRUCTION como el OLLYDBG.exe normal, asi que pueden parchearlo por ustedes mismos o usar el que esta dentro de la leccion 39, en la correcion fue agregado dentro tambien el PARCHEADO 4.

La segunda parte de este tutorial sera hacer un simple script, para lo cual usaremos el OLLYSCRIPT 0.92 que adjunto a este tutorial, y pondremos la dll en la carpeta plugins del OLLYDBG.



Ahora abrimos el Parcheado 4

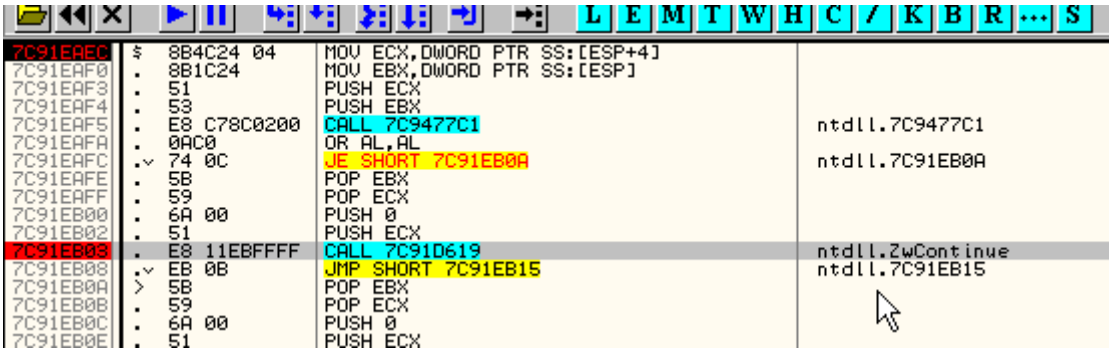


Vemos que aparece el PLUGIN

Ahora abro el pelock que estabamos estudiando, el tema sera hacer un minisript para que funcionen los hardware bpx y que no sean detectados por el packer.

Sera un sencillo script veremos los comandos que por supuesto los consultamos en el README del plugin OLLYSCRIPT.

La idea es la siguiente como vimos en la parte anterior cada vez que para en KiUserExceptionDispatcher estamos justo antes de que el sistema operativo llegue al manejador de excepciones donde sabemos que pueden ser detectador los HBP y borrados, asi que la idea es hacer un script que cada vez que el programa pare en el Bp KiUserExceptionDispatcher, borre el HBP y cuando llegue al CALL a ZwContinue lo restaure.



Como es nuestro primer script y para no complicarlo demasiado, los BP en estas direcciones tanto en KiUserExceptionDispatcher y en el call que va a ZwContinue los colocaremos a mano pues ya sabemos ubicarlos, facilmente y logicamente se pueden colocar sin problemas con el mismo script, pero eso ya lo veremos mas adelante no quiero complicarlo de mas, asi que la idea es ir y colocar a mano un BP
KiUserExceptionDispatcher y en el CALL que va a ZwContinue como vemos en la imagen.

Ahora que ya pusimos a mano empezaremos el script el cual normalmente se comienza con un etiqueta arbitraria en mi caso use inicio:

inicio:

esto se coloca para que si necesitamos saltar al principio del script facilmente con un Jmp inicio esta solucionado, luego de inicio viene el Hardware Breakpoint que queremos colocar por ejemplo.

bphws 12ffc4, "r"

bphws es el comando que coloca un hardware breakpoint en la direccion en este caso 12ffc4, y entre comillas la r significa on access o read que es lo mismo, si fuera w seria on write y si fuera x seria on execution.

inicio:

bphws 12ffc4, "r"

trabajo:

Luego de la colocacion del o los hardware bpx que quiero usar, pongo otra etiqueta llamada trabajo por si quiero retornar luego de la colocacion de los HBP.

inicio:

bphws 12ffc4, "r"

trabajo:

eob pirulo
run

vemos alli el comando eob, lo que hace el mismo es que cada vez que el OLLYDBG detecte una excepcion o breakpoint, pasa la ejecucion a la etiqueta que esta al lado del comando en este caso eob pirulo, significa que cuando el programa pare en una excepcion o breakpoint seguira corriendo desde la etiqueta pirulo, y luego doy RUN con el comando llamado run.

Este es la parte principal, alli cuando ejecute el script el OLLYDBG comenzara a correr la victima, debemos preparar que vamos a hacer cuando detecte una excepcion o BREAKPOINT , para ello escribiremos la etiqueta pirulo:

pirulo:
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
jmp final

en mi maquina 7c91eaec es la direccion de KiUserExceptionDispatcher o sea cuando pare en ese BP, el eip de mi maquina sera 7c91eaec, por lo tanto compruebo que estoy realmente alli y salto a la etiqueta quitar, para borrar los HBP.
Luego hay otra comparacion para cuando pare por el otro BP que esta en CONTINUE y cuando detecte que el programa paro por ese BP, vaya a la etiqueta restaurar, para poner nuevamente los HBP.

Luego solo me queda

quitar:
bphwc 12ffc4
jmp trabajo

Cuando salto a quitar para borrar los HBP el comando bphwc sirve para borrar los mismos y luego de borrarlos salto a trabajo, para que vuelva a ejecutarse todo y a correr el programa pero sin colocar los HBP.

La otra etiqueta es restaurar que s eactiva en el Call a ZwContinue

restaurar:
jmp inicio

que directamente salta al inicio donde se vuelven a colocar los HBP y se reinicia el ciclo.

Por supuesto debemos tener una posibilidad de parar el programa cuando OLLYDBG pare por nuestro HBP, en ese caso es esta parte final.

Como cuando ejecuta eob para tambien por cualquier breakpoint o excepcion, cuando para por nuestro HBP, tambien ira a pirulo y como no es ninguno de los BP que colocamos a mano seguira hasta la etiqueta final:

pirulo:
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
jmp final

alli en final:

final:
MSGYN "Continuar?"
cmp $RESULT,1
je inicio
ret

MSGYN es que hacemos aparecer un messagebox con la opcion continuar, para que veamos si nos gusta el lugar donde paro o queremos que siga corriendo hasta la proxima vez que pare por el HBP. Si apretamos YES en la variable reservada $RESULT habra un 1, y si apretamos no un 0, por lo cual cuando queremos que continue, al apretar YES el script compara $RESULT con 1 y vuelve al inicio a continuar trabajando, si no es 1 va al ret donde termina y para el script y queda el OLLYDBG parado donde nosotros queriamos.
El script completo es (recordar poner manualmente los BP en Bp KiUserExceptionDispatcher y el CALL que va a ZwContinue)

inicio:
bphws 12ffc4, "r"
trabajo:
eob pirulo
run

pirulo:
cmp eip, 7c91eaec
je quitar
cmp eip, 7c91eb03
je restaurar
jmp final

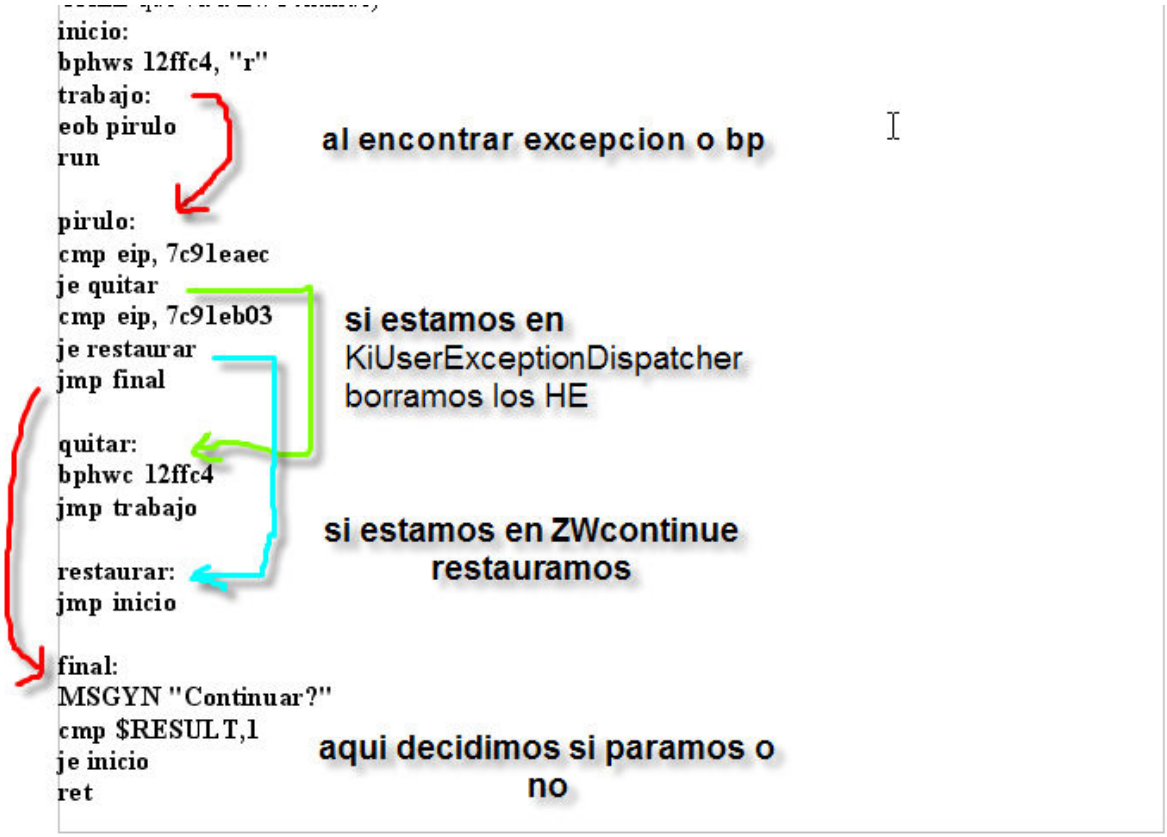
quitar:
bphwc 12ffc4
jmp trabajo

restaurar:
jmp inicio

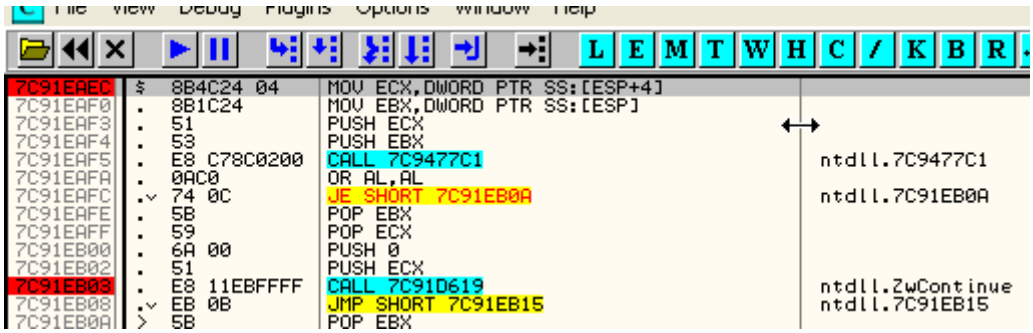
final:

MSGYN "Continuar?"
cmp $RESULT,1
je inicio
ret

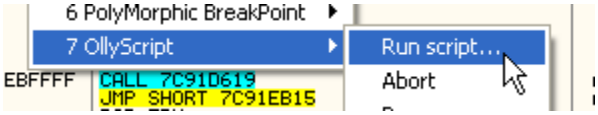
Abajo lo vemos mejor con unas flechas explicativas, al inicio ponemos los HB y corremos el programa dejando la trampa lista que cuando encuente BP o excepcion salte a pirulo.
A pirulo solo saltara cuando hay BP o excepcion y en ese momento tenemos que testear las distintas posibildades, de que sea por los BP que colocamos a mano, o por el HE o por cualquier excepcion.
Testeando el eip, podemos determinar si se disparo el salto a pirulo, por el BP de KiUserExceptionDispatcher con lo cual si se verifica, salta a quitar los HBP para que no sean detectados en el manejador de excepciones, y retorna a trabajo donde vuelve a correr el programa sin HE.
Luego en el BP del CALL a ZwContinue saltara nuevamente a pirulo y ahora comparando eip, nos llevara a restaurar ya que en este momento ya paso el manejador de excepciones y los puedo volover a colocar, y luego para cualquier otro BP o excepcion o si para por el HE, salta a final donde decido si parar o continuar, si paro va al ret si continuo va a inicio.

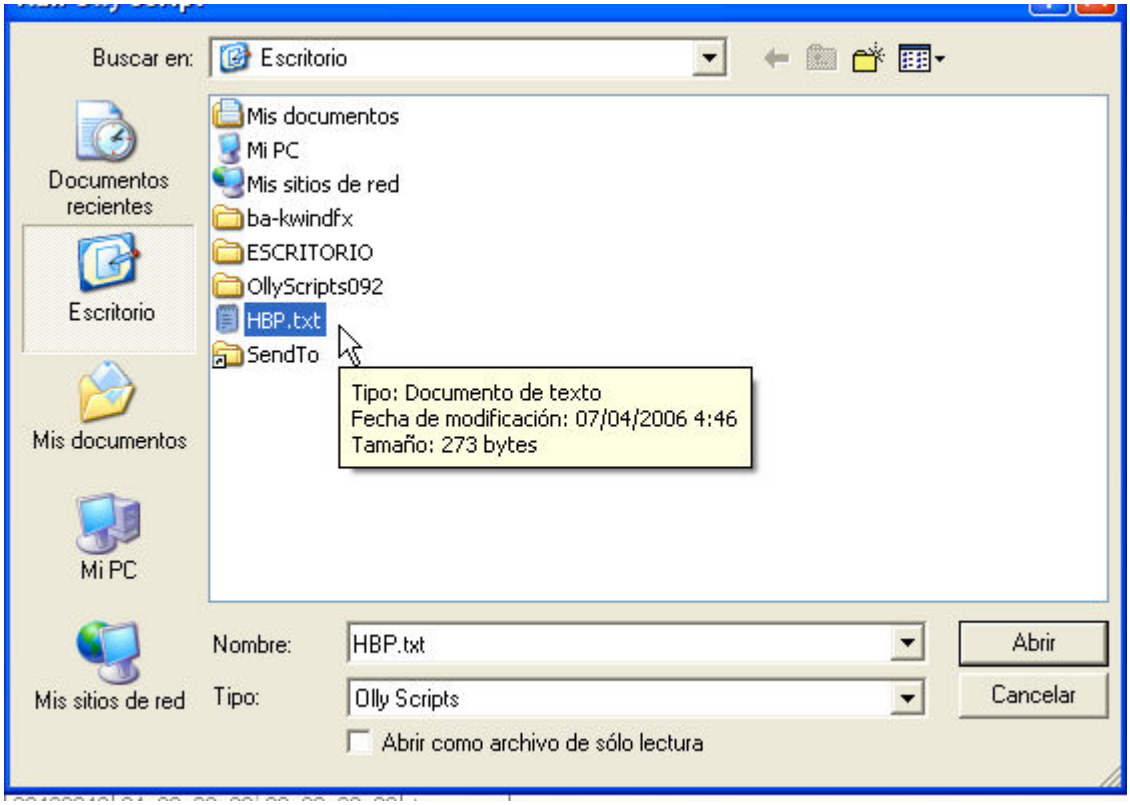


Probaremos el script lo guardo en una fila de texto y arranco el pelock, luego pongo manualmente los 2 BP y voy al menu del plugin

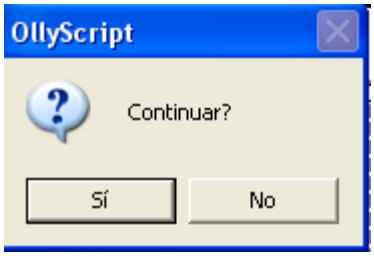


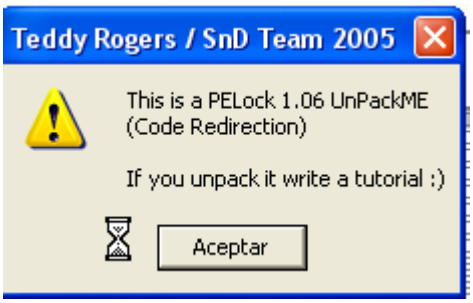
.



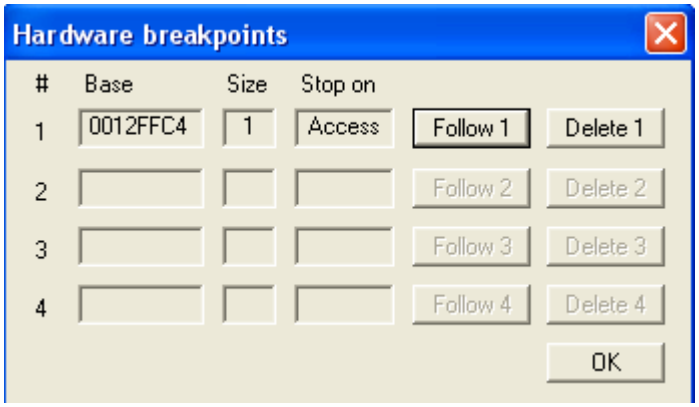


Veo que para por el HBP, igual apreto YES para que siga





Veo que el programa corrio perfectamente y veo en la ventana dehardware breakpoints



El mismo se encuentra colocado por lo cual el programa corre con HBP y para en los mismos sin ningun problema, que era lo que queriamos lograr, jeje.

En la parte 41 y despues de esta introduccion que nos ayudara seguiremos haciendo scripts y repararemos el pelock completamente.

Hasta la parte 41
Ricardo Narvaja
07/04/06