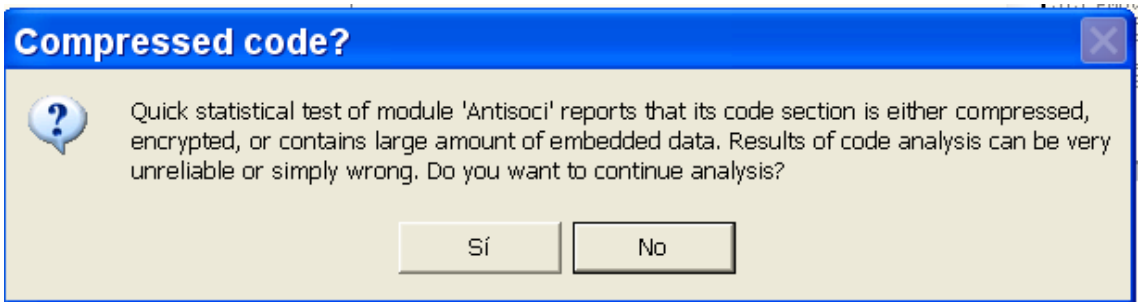
INTRODUCCION AL CRACKING CON OLLYDBG PARTE 24

Me pidieron que antes de empezar con la parte que versa sobre excepciones, muestre como se hace correr en OLLYDBG el antisocial que deje como ejemplo, la vez anterior.

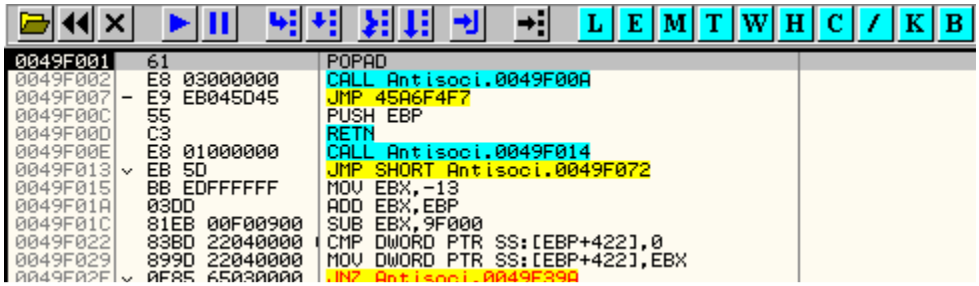
Antes que nada sabemos que es un programa empacado, y eso aun no se ha enseñado por lo cual lo haremos correr en OLLYDBG haciendo los cambios en la memoria solamente sin guardarlos, cuando ya sepamos desempacar pues podremos guardar los cambios definitivos.

Primero lo haremos correr en un OLLYDBG renombrado con todos los plugins puestos para ver porque no corre y tratar de arreglarlo.

Arrancamos el antisocial en mi OLLYDBG renombrado, parcheado y con todos los plugins habilitados.

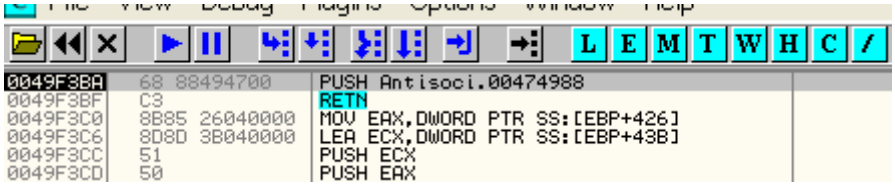


Esto ya nos da una idea que puede estar empacado, llegamos el ENTRY POINT

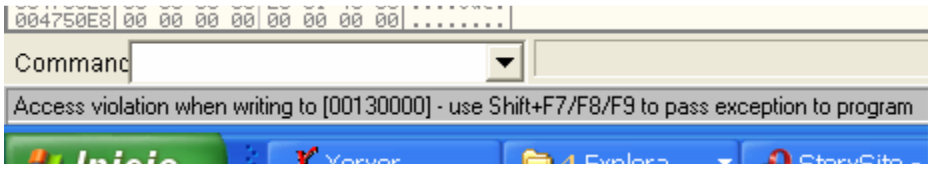


Allí ya vemos algo extraño, pues POPAD es la sentencia que se usa para recuperar del stack, los valores que guardo previamente un PUSHAD, y aquí no hay ningún PUSHAD antes, por lo tanto hay algo sospechoso allí.

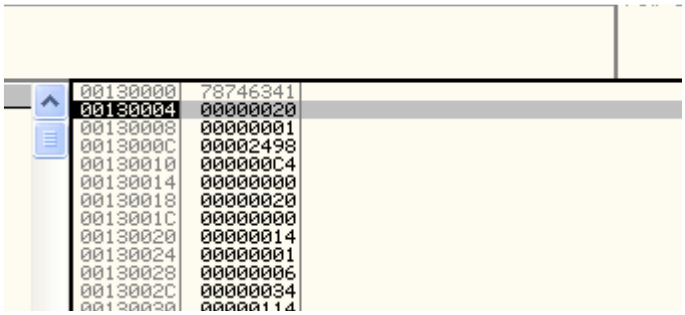
Lo corremos igual a ver que pasa



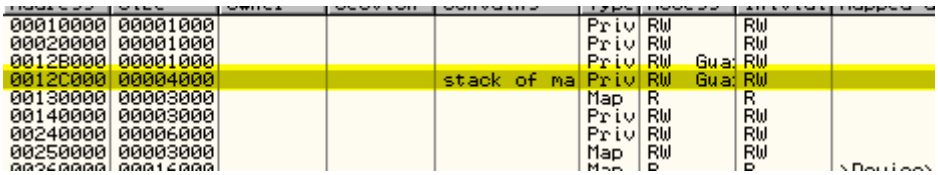
Para aquí



O sea da error cuando quiere hacer el PUSH como si el stack no tuviera permiso para escribir alli, pero el stack siempre tiene permiso de escritura que ocurrió aquí, veamos el stack

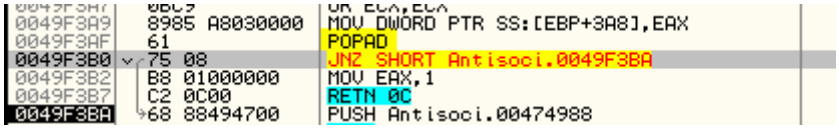


O sea el valor superior del stack es 130000, y si reiniciamos el programa

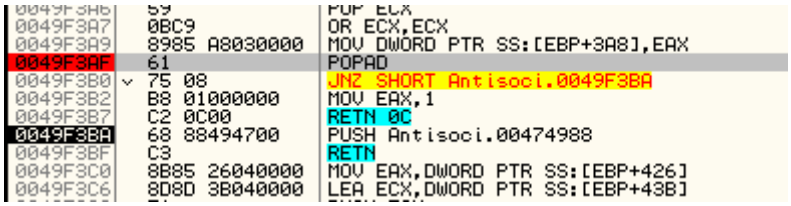


Y vemos las secciones, vemos que el stack va en mi maquina desde 12c000 hasta 12ffff, el error fue que el stack se salio de esa sección, y ahora apunta a la sección siguiente que empieza en 130000 y que como no es el stack, no tiene permiso de escritura y da error.

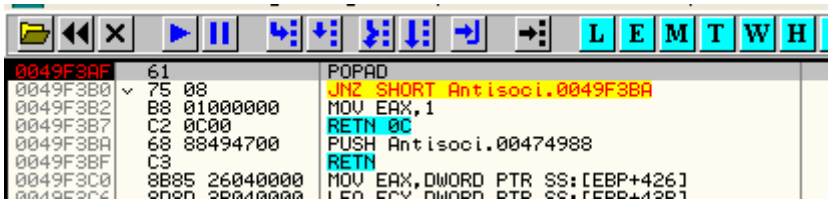
Lleguemos nuevamente a la zona del error



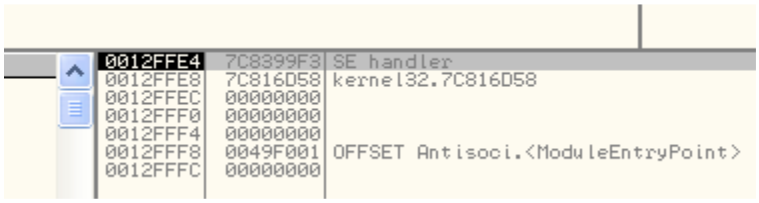
Vemos que el programa ejecuta otro popad y hace un salto JNZ al PUSH que da error veamos, pongamos un BPX en ese popad antes del error.



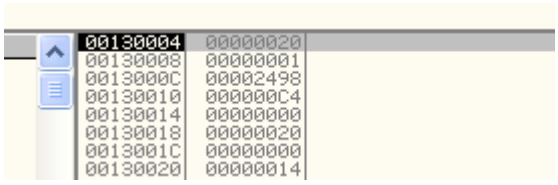
Ahora reiniciemos el programa y demos RUN para que pare allí.



Veamos el stack

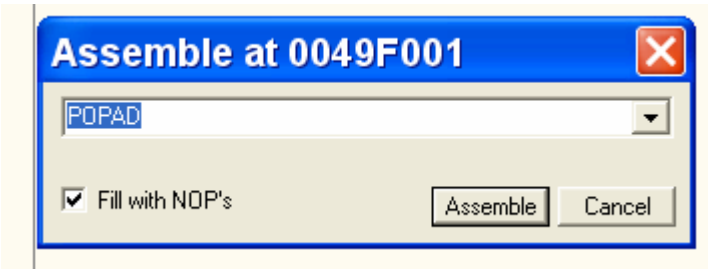


Esta aun en la sección correcta ejecutemos el popad

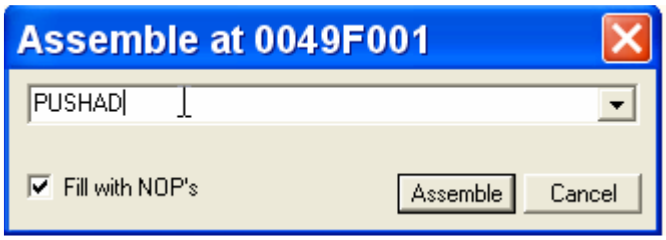


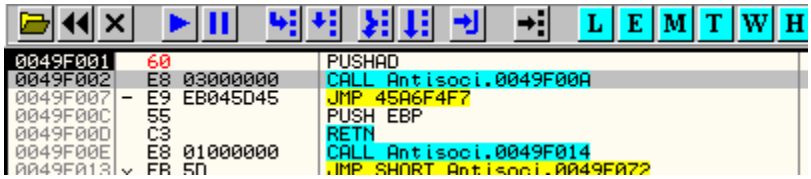
Allí ya se salio de sección, así que el problema se puede resolver nopeando este popad, pero lo correcto es que el popad inicial debe ser un pushad, que guarda los valores de los registros iniciales en el stack, y este popad es la sentencia opuesta los recupera, veamos que pasa si reemplazamos el popad inicial por un PUSHAD, reiniciemos.

Apreto la barra espaciadora

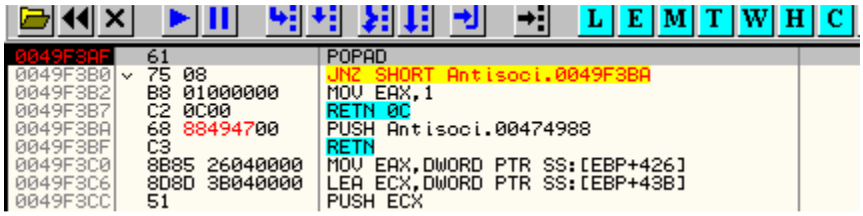


Y escribo PUSHAD

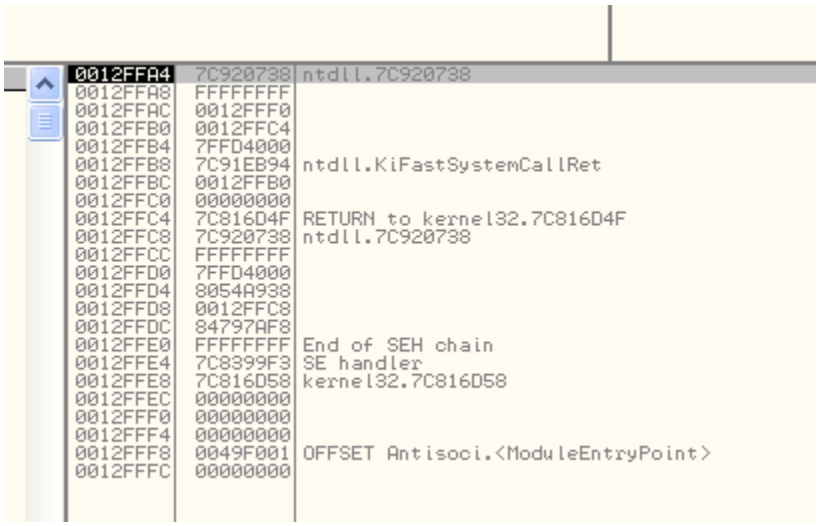




Ahora doy RUN y para en el segundo POPAD

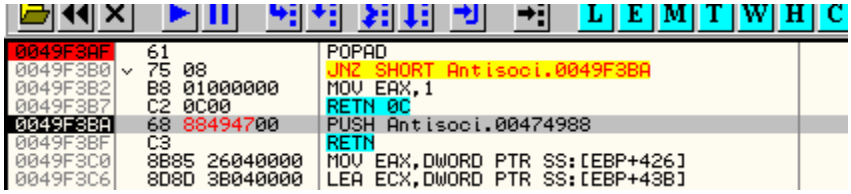


Pero esta vez el stack esta mas arriba

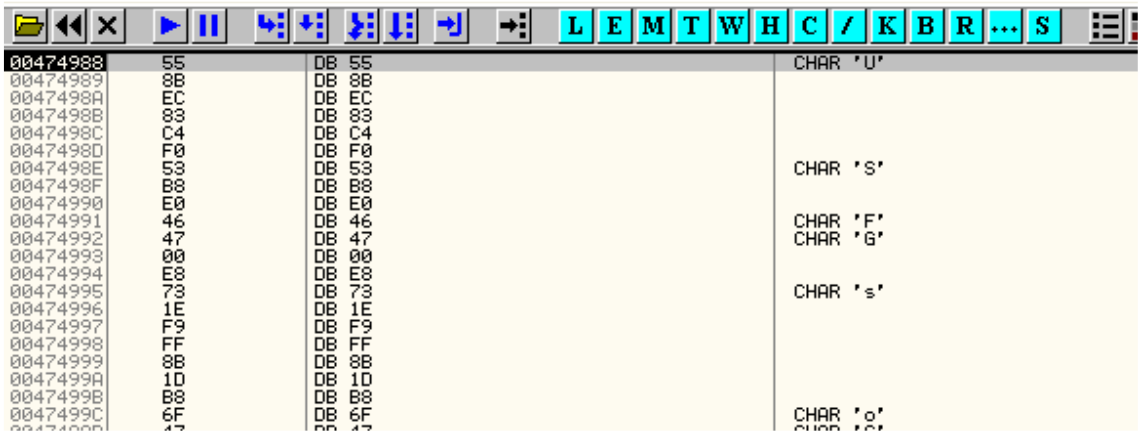


Por lo cual hacer un POPAD no lo sacara de sección, veamos pasémoslo con F8

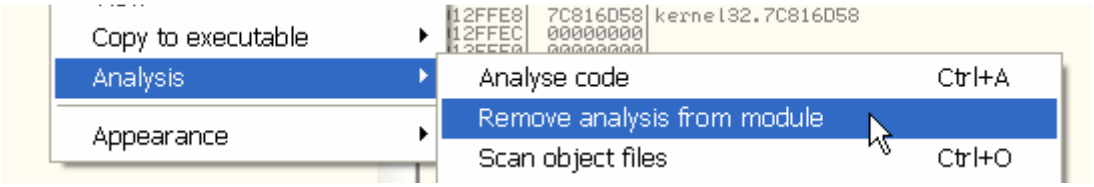
Vemos que se mantiene todo correcto

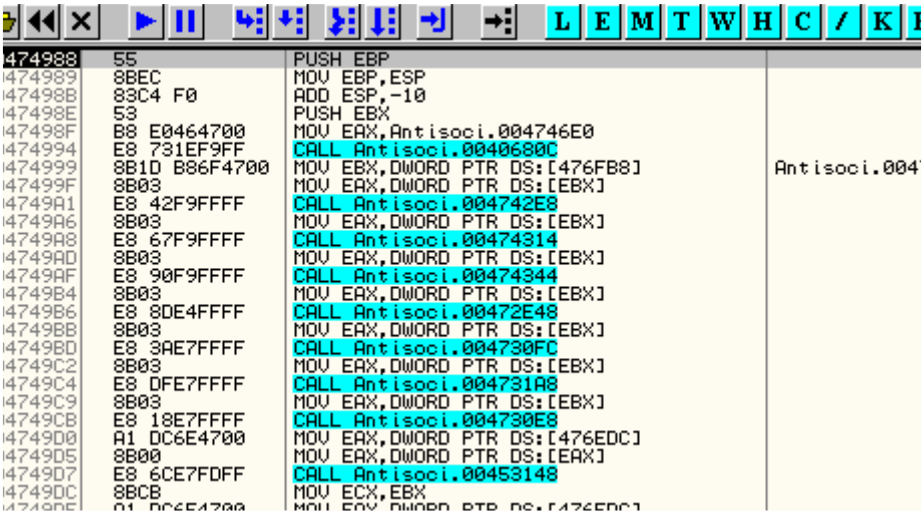


Llegamos al PUSH y al RET, traceando, y con f8 pasamos el RET



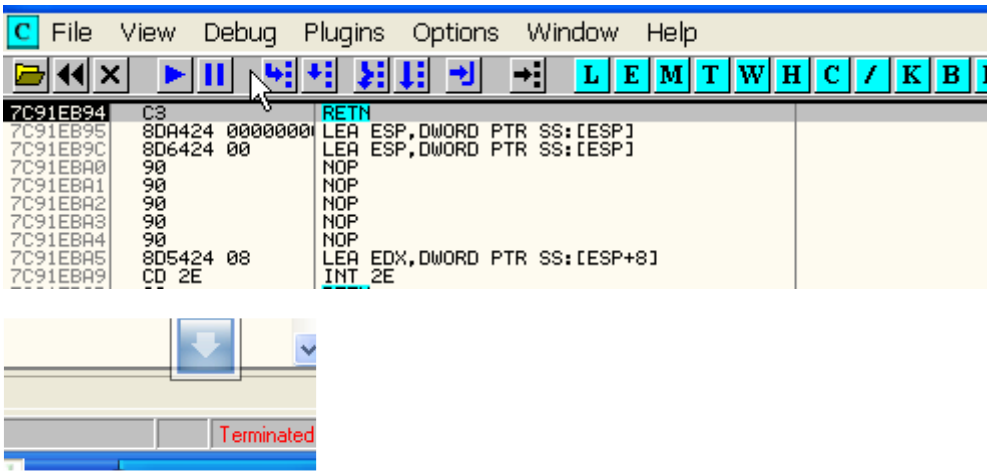
Bueno aquí hay un problema de análisis



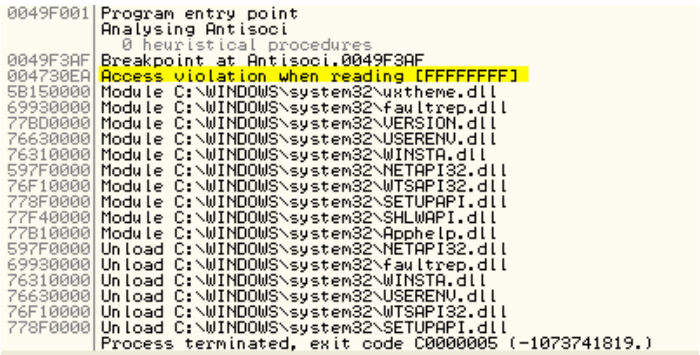


Ahora si se ve bien

Veamos que ocurre si damos RUN

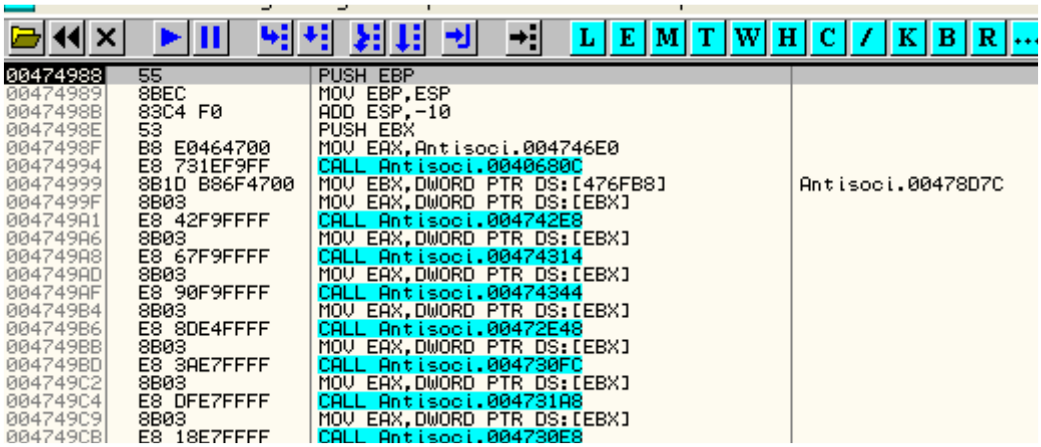


Veamos en el LOG del OLLYDBG si vemos algo ya que el programa se termina

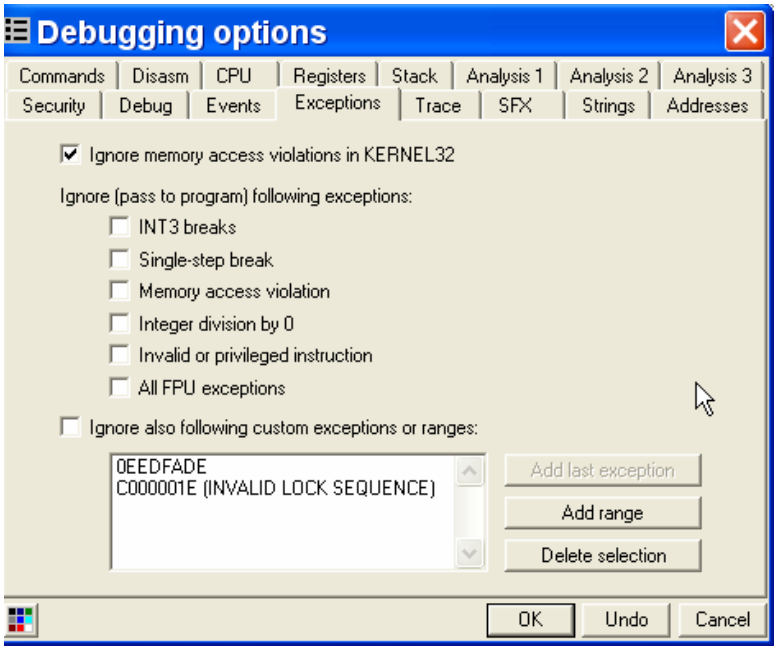


Vemos que hay una excepción, luego de parar en el BPX del popad

Reiniciemos y repitamos los pasos para llegar nuevamente adonde estábamos.

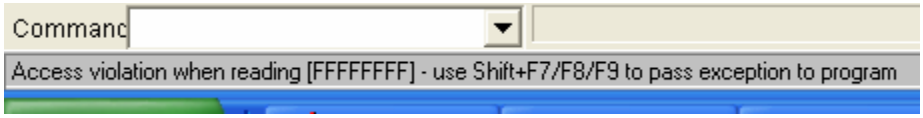


Quitemos todas las tildes, menos la primera para que pare en las excepciones

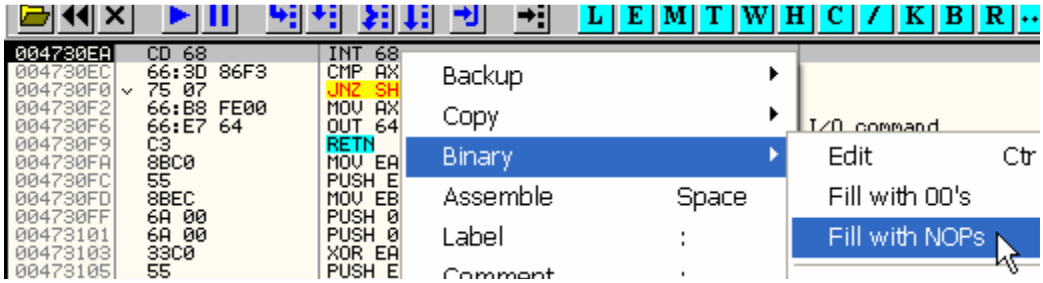


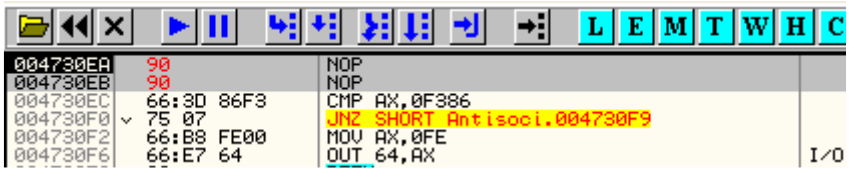
Demos RUN

Para en la excepción que vimos en el LOG

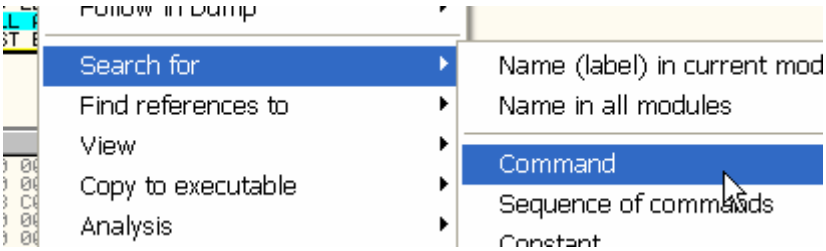


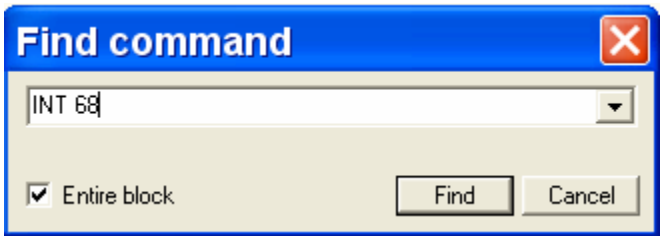
INT68 es una de las pocas excepciones que el OLLYDBG no puede manejar, podemos pasarla nopeandola.

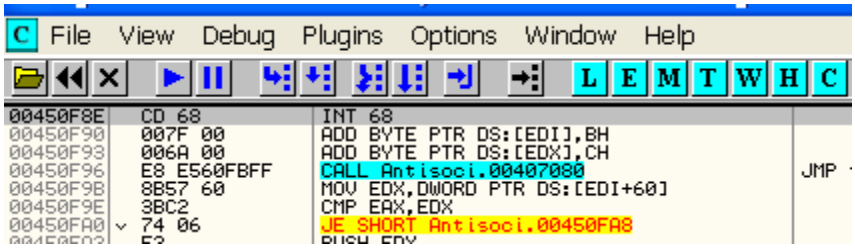




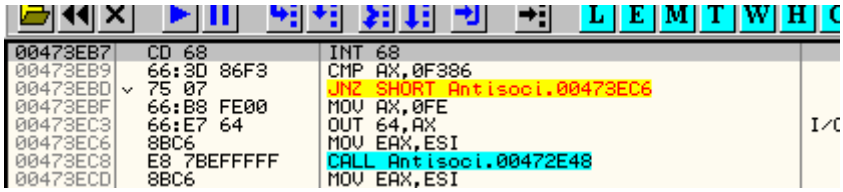
Por otro lado sabemos que puede haber mas INT68 que molesten busquemos a ver si halla alguna mas, así lo nopeamos directamente.







Vemos que halla otro lo nopeamos



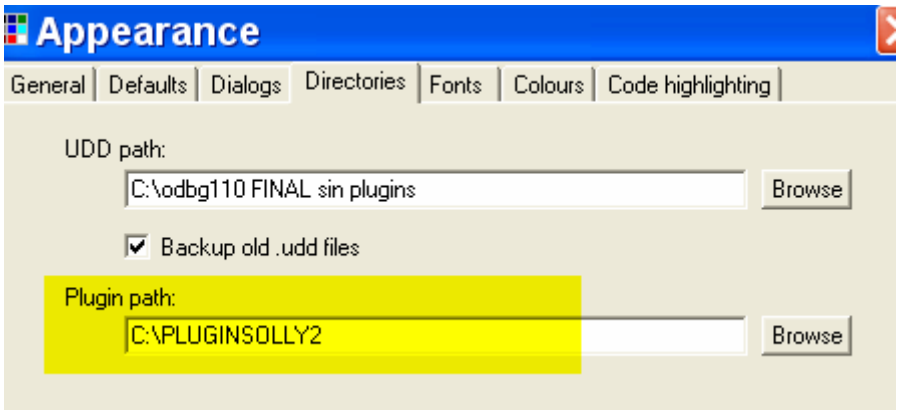
Buscamos nuevamente hay otro mas, lo nopeamos

Si hacemos CTRL + L continua buscando a partir del ultimo que hallo

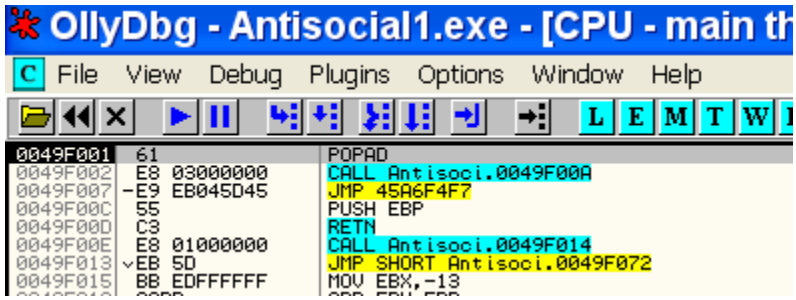
Ya no halla mas ahora si damos RUN

Como ven todo eso hay que hacer para que corra el OLLYDBG con los plugins, ahora tratemos de correrlo en un OLLYDBG sin plugins, y sin renombrar ni parchear.

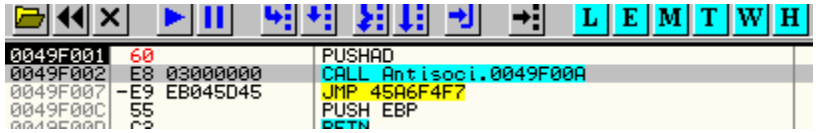
Allí descomprimí un OLLYDBG sin plugins, bah el único que tiene es el command bar, ya que ese no protege de nada es solo por comodidad, a este OLLYDBG, le hice una carpeta diferente de plugins y el path a los plugins lo apunte allí.

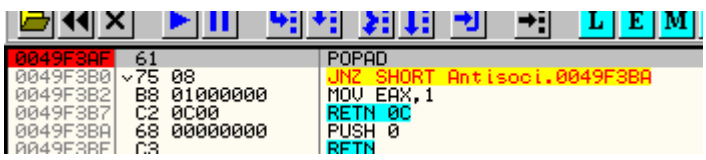


Como vemos allí hay solo dos plugins, lo arranco en ese OLLYDBG

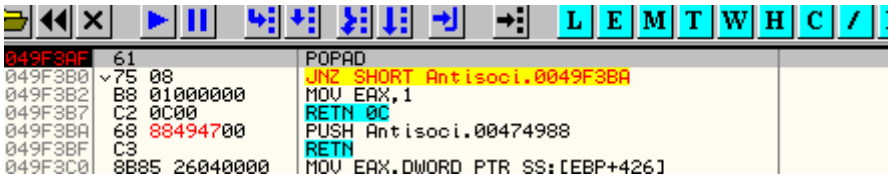


Repito las operaciones de cambiar el popad

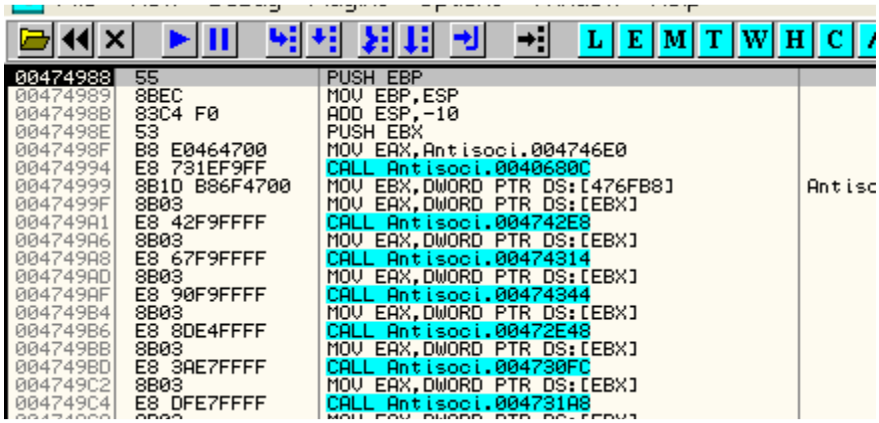




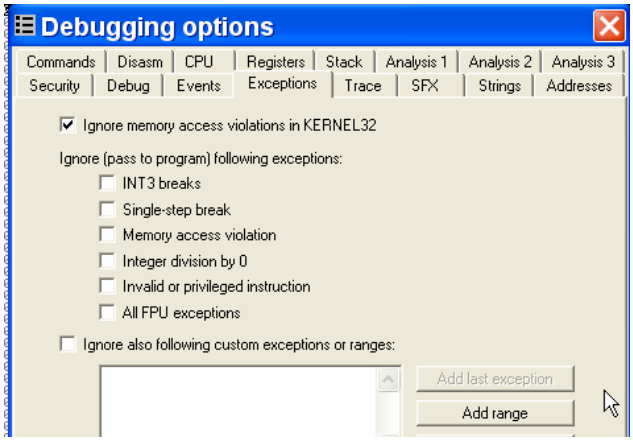
Poner el BPX en el POPAD veamos si llega alli demos RUN



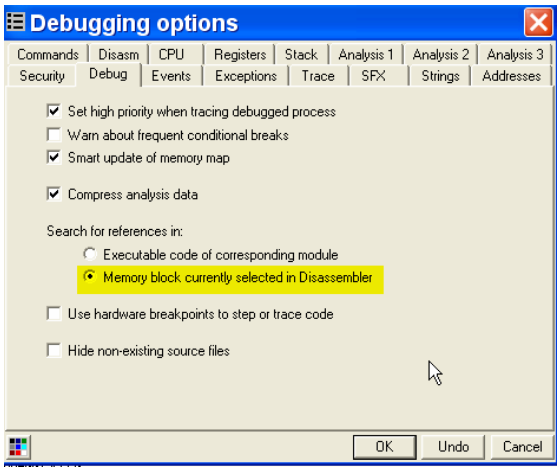
Lleguemos hasta el código del programa desempacado.



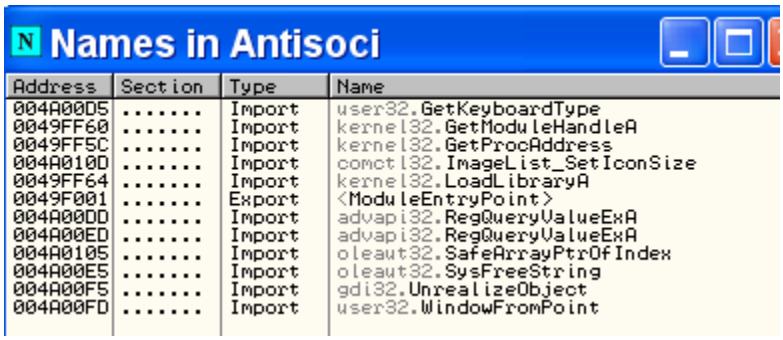
Podemos nopearle los INT68 o quitar las tildes en exceptions y cada vez que pare si es un INT68 nopearlo y si no pasarlo con SHIFT +f9 como a cualquier excepción.



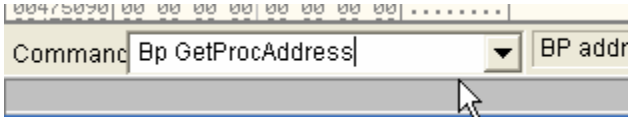
Veamos que apis usa el programa para esto debemos recordar que la tilde en



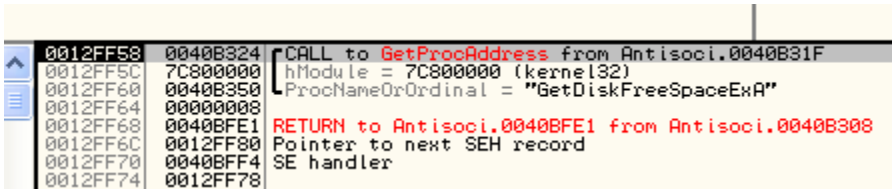
Debe estar colocada alli para que muestre la información de la seccion en que estamos.



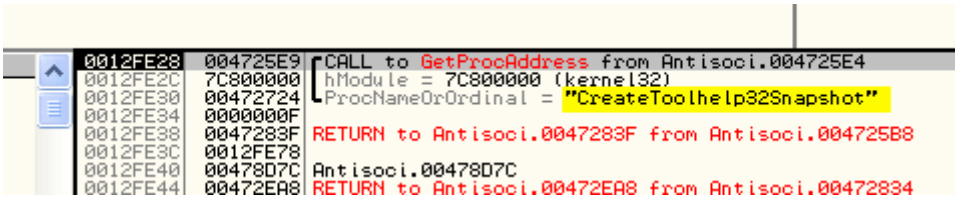
Bueno vemos que no hay apis de las sospechosas, pero usa muy pocas apis y esta alli GetProcAddress para cargar mas apis nuevas, asi que pongamos un BPX en la api GetProcAddress.



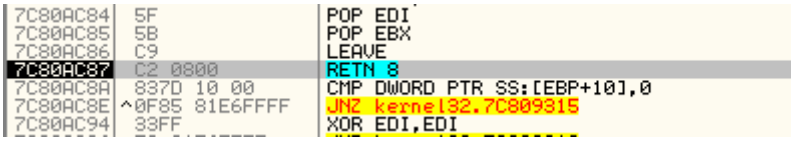
Demos RUN

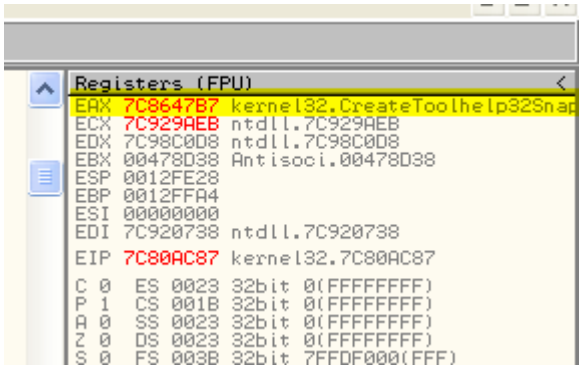


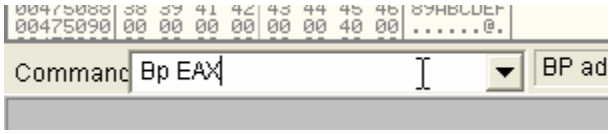
No es sospechosa sigamos dando RUN hasta que pare en alguna sospechosa

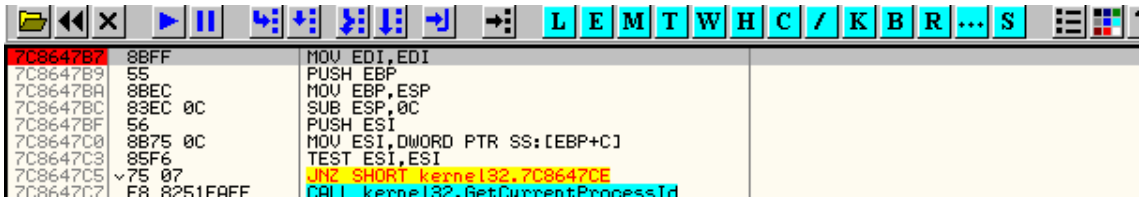


Aquí vemos la primera sospechosa la api que nos hace la foto de todos los procesos que están corriendo en nuestra maquina, lleguemos al RET y pongamos un BP EAX ya que en EAX estará la dirección de la api en nuestra maquina.

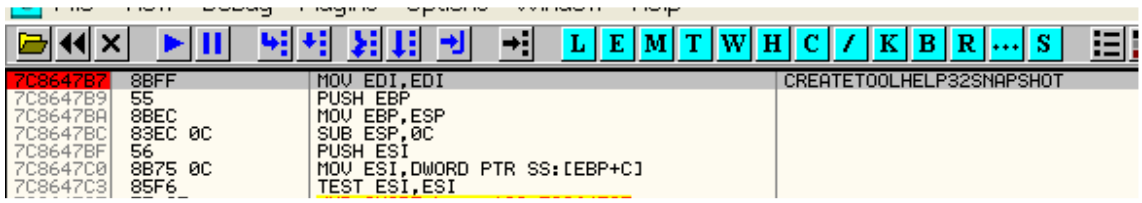






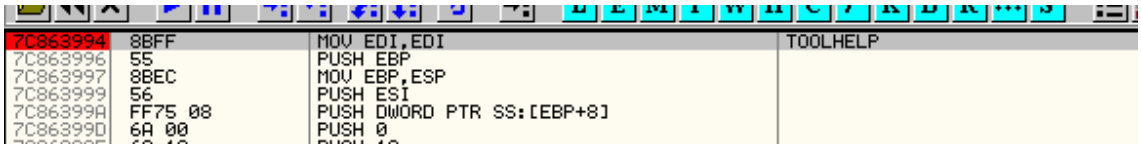


Allí esta el BP le pondré el comentario para saber que api es

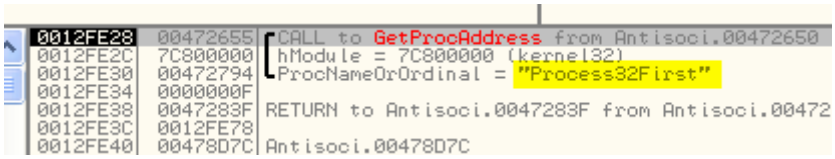


Listo continuemos

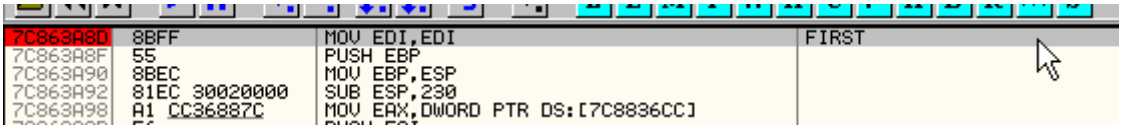
Hmm otra parecida a la anterior por si acaso pongámosle un BP también

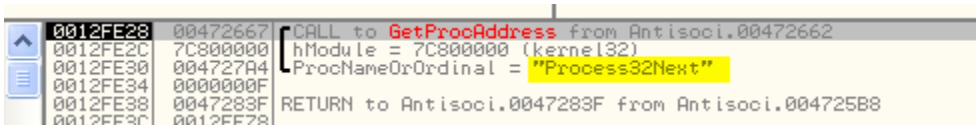


Demos RUN

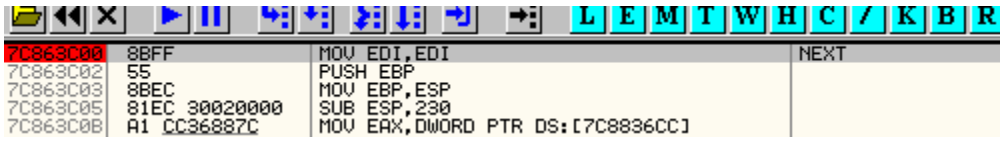


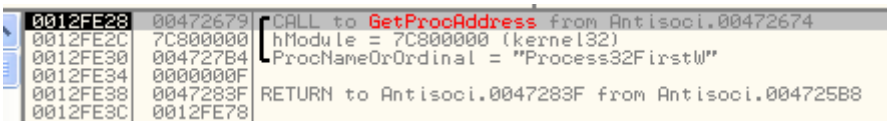
Bueno esta es conocida BP en ella

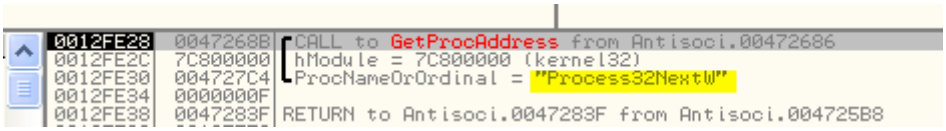


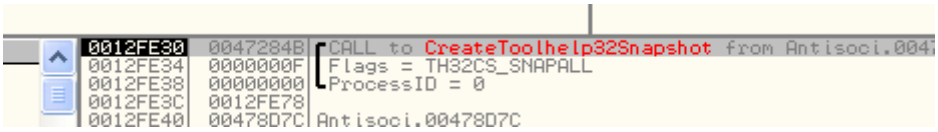


Lo mismo

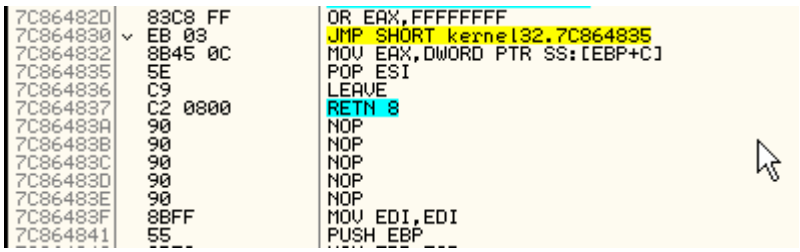




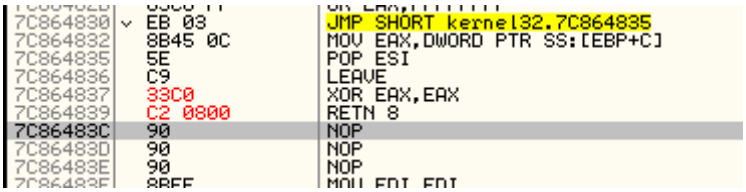




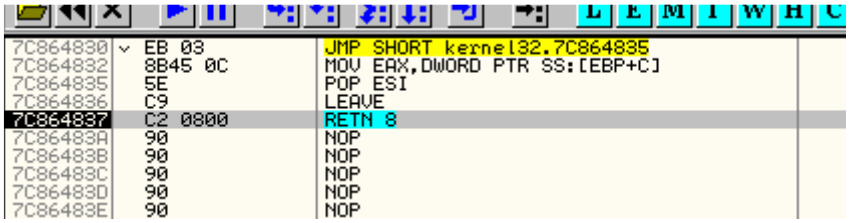
Allí para en la api que saca foto y como toda la protección depende de la foto, desde la cual de donde trabaja con la lista de procesos, threads, y no se cuantas cosas mas testea todo de la bendita foto, podemos intentar parchear esta api de la foto para que no devuelva el handle de la misma sino cero, para ellos vamos al ret de la api.



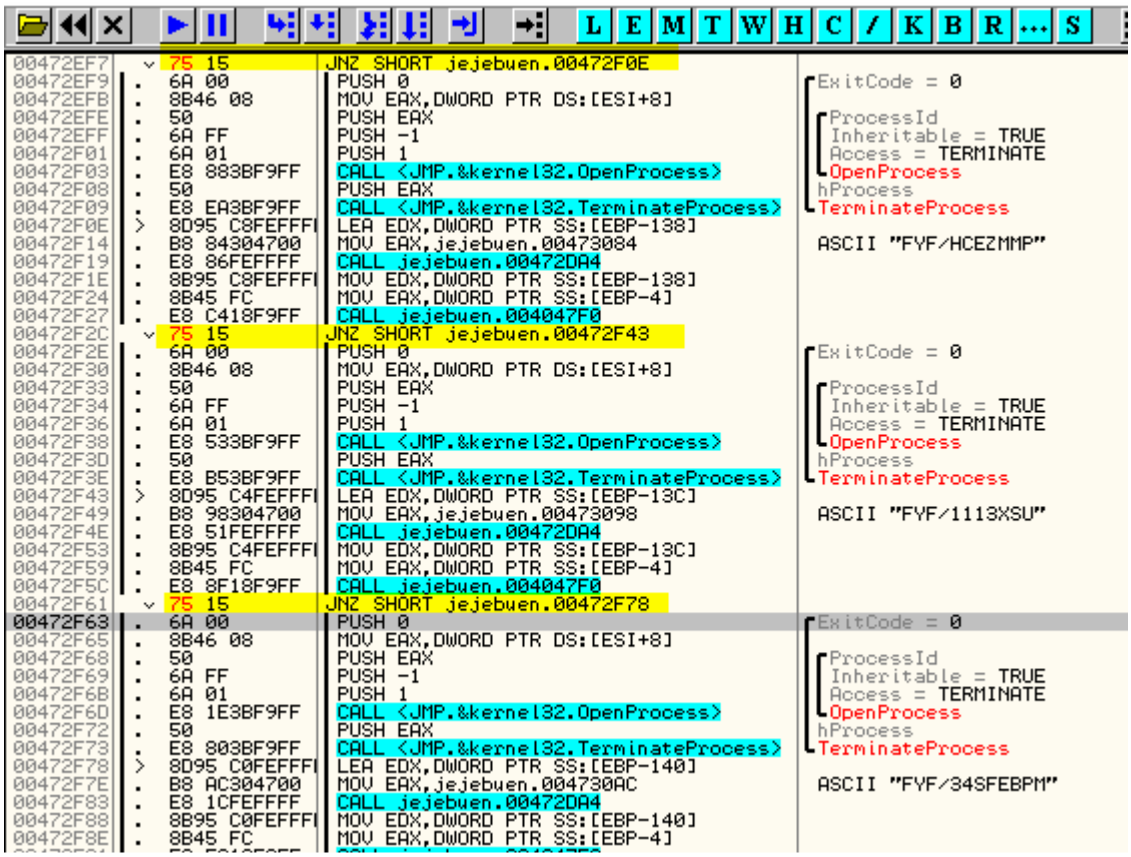
Vemos que alli hay un espacio vacio asi que podemos hacer EAX igual a cero antes de volver



Con eso devolverá cero y el programa no tendrá handle para poder manejar las fotos ni averiguar sobre ellas, esa es una posibilidad la otra cambiando en el codigo del programa llegamos al ret dela api la cual no modificamos.

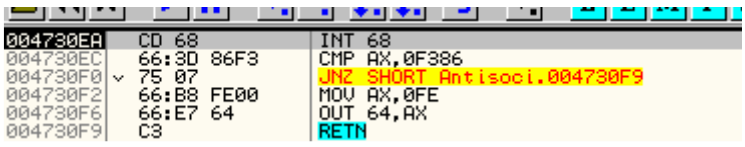


Alli vemos un poco mas abajo unos saltos JNZ y justo abajo los call a TerminateProcess que cerrarian el OLLYDBG con la llamada anterior a OpenProcess para obtener el handle, y vemos que hay como cinco de estas partes justo una debajo de otra



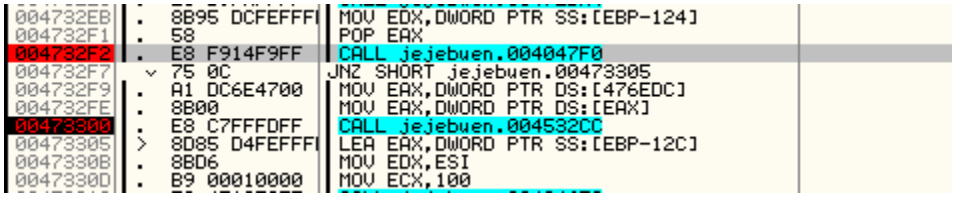
Por supuesto cambiando todos los saltos JNZ por JMP, evitamos que el programa llegue a TerminateProcess evitamos y que cierre el OLLYDBG.

Esta primera parte esta solucionada ahora sigamos con la segunda parte de la protección, demos RUN

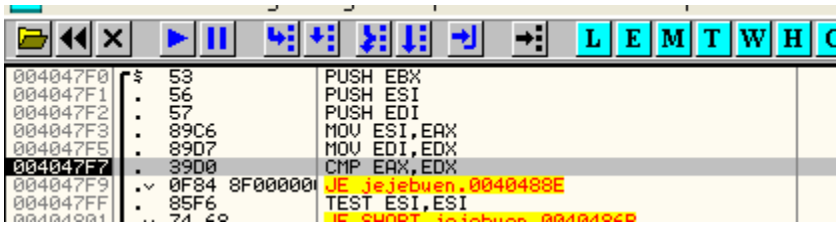


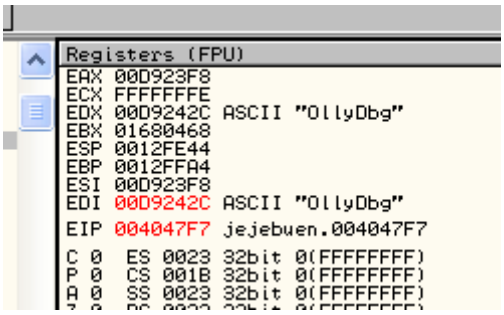
Nopeamos y damos RUN y corre pero se cierra el programa, si le pongo solo el plugin HideDebugger 1.23f y lo protejo contra FindWindows/EnumWindows corre bien con los pasos anteriores que vimos.

El tema del lugar de donde se cierra es aquí

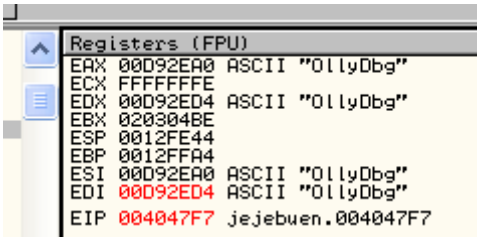


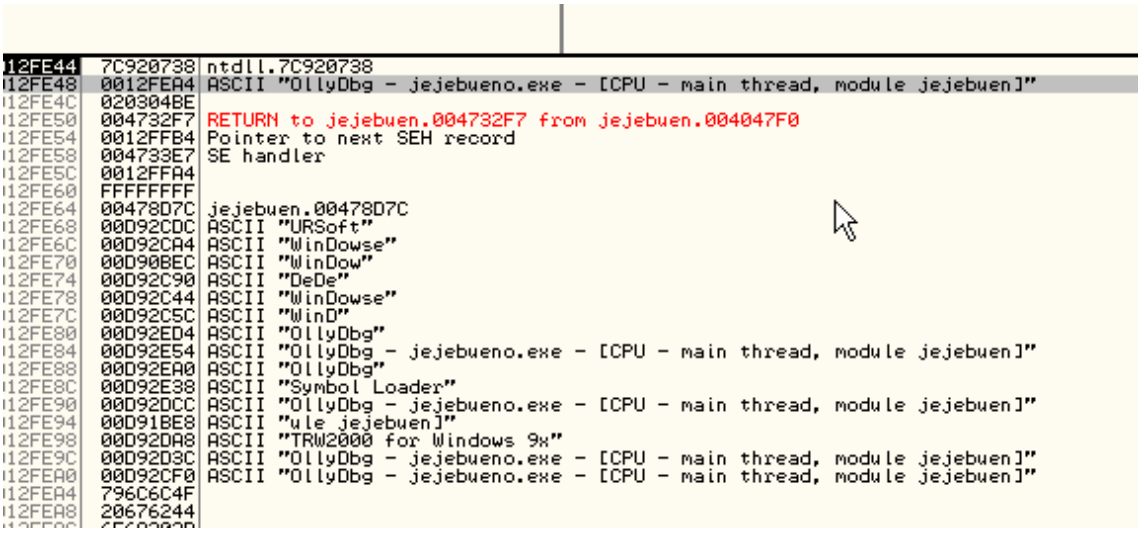
Cuando corremos el programa cuando para alli en el call superior.





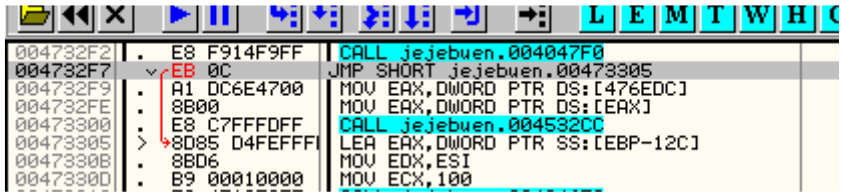
Vemos que en una de esas tantas comparaciones, que para alli.



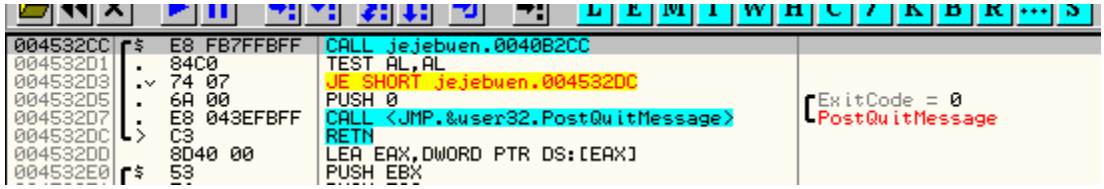


La cuestión, es que aquí compara si hay cosas malas, jeje y que todo eso se evita con el jnz que esta a la salida del primer CALL si lo cambio por JMP

Ese es el salto clave si lo cambio por JMP



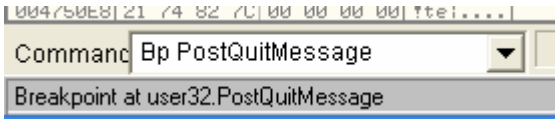
Evita el segundo call que si lo vemos dentro

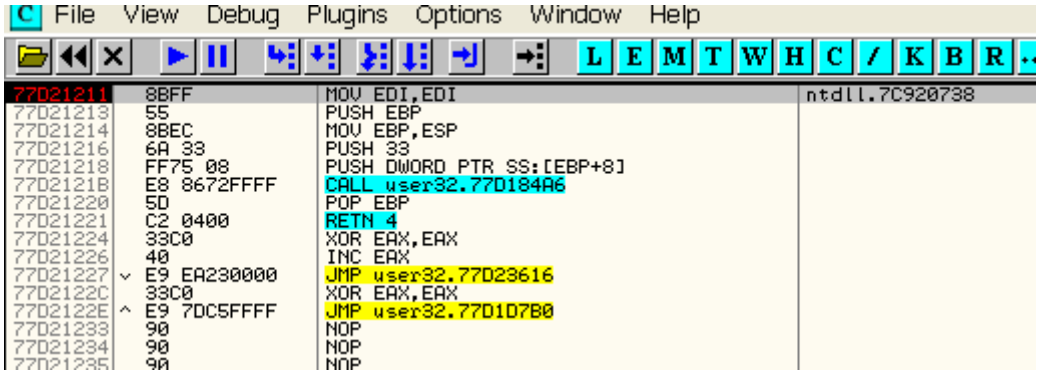


Es el que te lleva a PostQuitMessage y va cerrando la ventana pues esta por terminar el programa lo cual lo hace mas adelante, pero aquí ya la decisión esta tomada, el PostQuitMessage es que se va a cerrar el loop de mensajes y no queda otra que el crackme se cierre ya que no tiene mas ventanas.

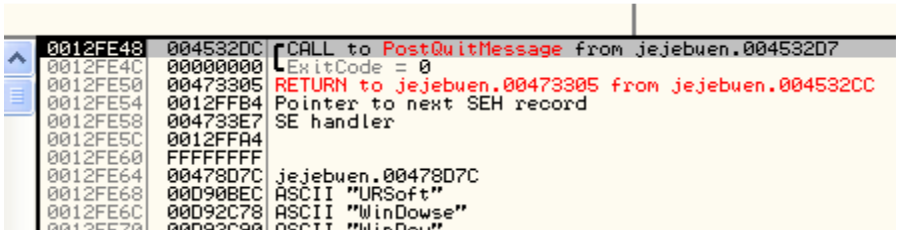
Por supuesto si me dicen como halle ese JNZ pues es fácil.

Una vez que paso la primera parte de la protección, pongo un BPX en PostQuitMessage.

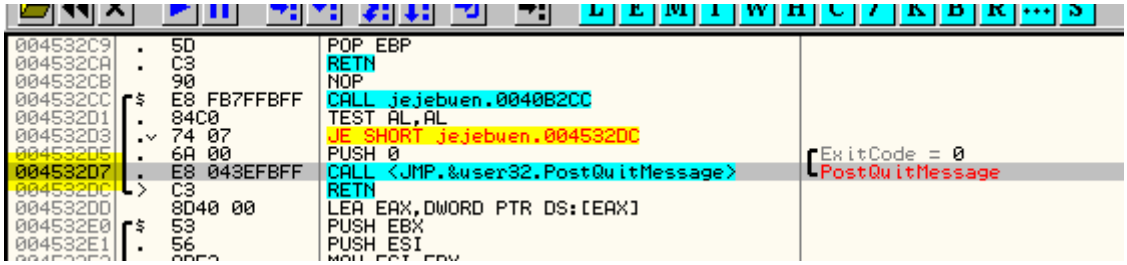




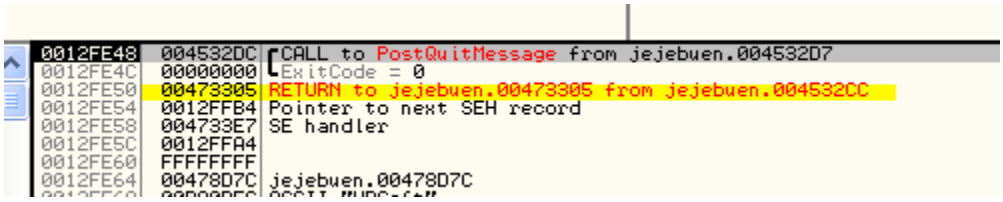
Para en la api miro el stack a ver de donde fue llamado



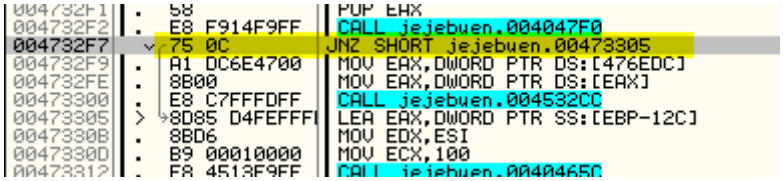
Voy a 4532d7 ya que el stack me dice que fue llamado de alli



Por supuesto probé ese JE que esta antes del PostQuitMessage y no evitaba que se cierre al invertirlo, así que fui al segundo RETURN TO que encontré en el stack.



Este viene de 4532CC



Como vemos este call se evita con el JNZ anterior con lo cual no entra y no va a PostQuitMessage y al cambiarlo por JMP veo que corre perfecto sin ningún plugin.

Adjunto el crackme desempacado que corre en cualquier OLLYDBG por supuesto ustedes aun no aprendieron a desempacar en este curso, por lo cual aun eso no se pide, pero si quieren verlo por curiosidad, allí esta adjunto pueden probar que corre en cualquier OLLYDBG se llame como se llame y sin ningún plugin, ya que además de desempacarlo, le hice los cambios permanentes que vimos en este tute y con eso quedo limbito como culito de bebe.

Hasta la parte 25 ahora si empezamos con las excepciones
Ricardo Narvaja
06 de enero de 2006