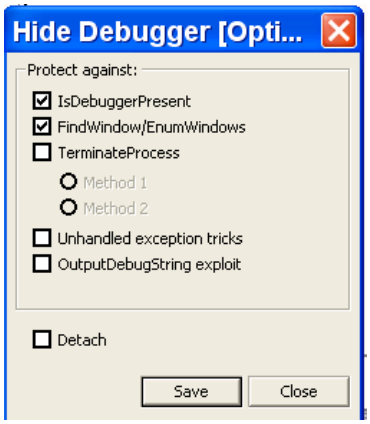
INTRODUCCION AL CRACKING CON OLLYDBG parte 22

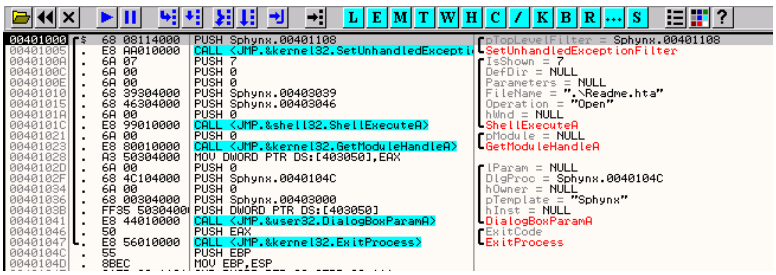
Bueno seguiremos con mas trucos antidebugging en esta parte veremos dos trucos antidebugging simultáneamente, ya que uno trabaja con el otro, por lo tanto estudiaremos los dos.
Es de mencionar que siempre estamos dando por sabido todo lo visto anteriormente y que iniciaremos este tute con el OLLYDBG que parcheamos en la parte 21, que se llama NVP11, con el plugin HideDebugger 1.23f por ahora configurado así.



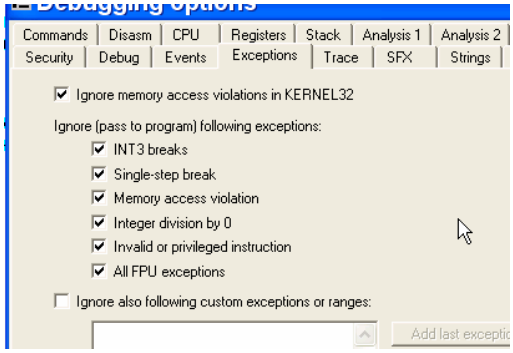
Allí vemos que hay una tilde llamada UNHANDLED EXCEPTION TRICKS, bueno estudiaremos este truco, a su vez que trabaja conjuntamente con otro truco que podemos encontrar también usado en forma separada, es el llamado a la api ZwQueryInformationProcess para detectar el debugger.

Usaremos el crackme adjunto llamado sphynx.zip, ya sabemos desde aquí que si le ponemos la tilde en UNHANDLED EXCEPTION TRICKS corre perfectamente, pero veremos un poco como trabajan ambos métodos.

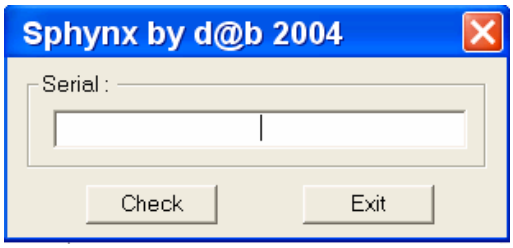
Otra cosa de aclarar que estos crackmes que no estamos solucionando, los veremos mas adelante, por ejemplo este necesita para solucionarse usar el método de fuerza bruta para hallar el serial correcto, y es posible que lo usemos como ejemplo cuando veamos ese método de ataque, por ahora solo lo haremos correr en OLLYDBG y explicaremos su método de protección.



Cargo el crackme en el Nvp11 (OLLYDBG parcheado) y verifico que las opciones del plugin HideDebugger 1.23f, estén como en la primera imagen y además dejo por ahora las excepciones todas marcadas.

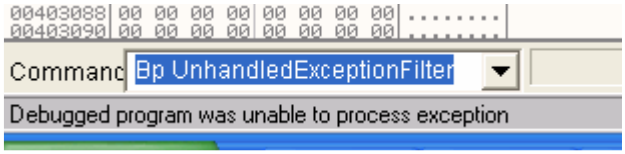


Doy RUN



Y el crackme llega a la ventana principal pero la protección esta después.

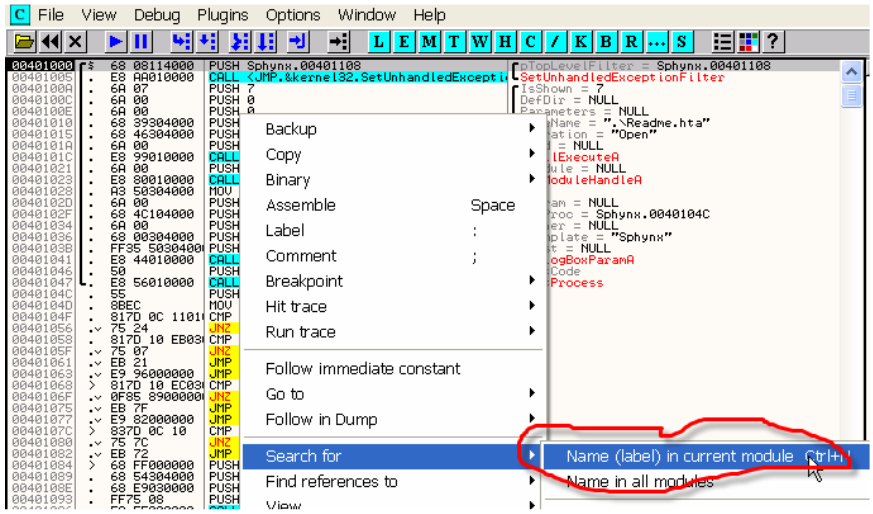
Coloco un serial falso y apreto check

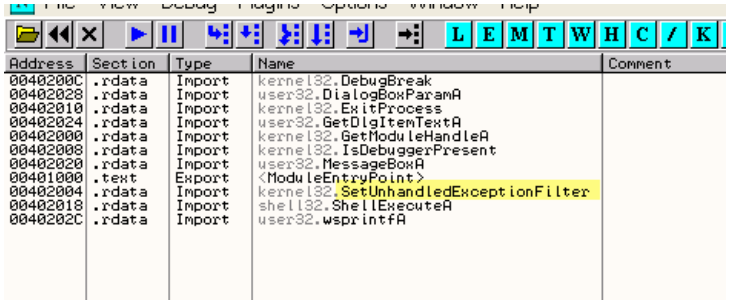


Me dice que no es posible procesar la excepción y se cierra si doy RUN.

Bueno me doy cuenta de que si lo pruebo fuera de OLLYDBG, el crackme no se cierra, continua funcionando y dándonos la posibilidad de seguir intentando con diferentes seriales.

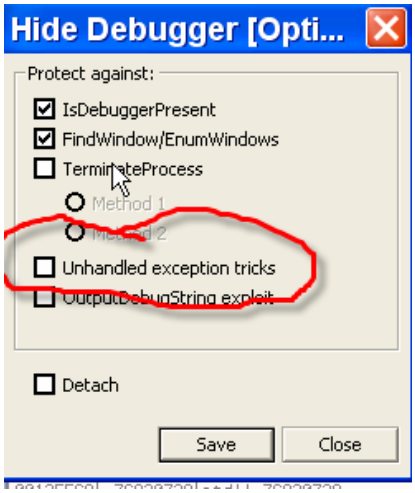
Bueno reiniciémoslo y veamos las apis que utiliza





Address	Section	Type	Name	Comment
0040200C	.rdata	Import	kernel32.DebugBreak	
00402028	.rdata	Import	user32.DialogBoxParamA	
00402010	.rdata	Import	kernel32.ExitProcess	
00402024	.rdata	Import	user32.GetDlgItemTextA	
00402000	.rdata	Import	kernel32.GetModuleHandleA	
00402008	.rdata	Import	kernel32.IsDebuggerPresent	
00402020	.rdata	Import	user32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
00402004	.rdata	Import	kernel32.SetUnhandledExceptionFilter	
00402018	.rdata	Import	shell32.ShellExecuteA	
0040202C	.rdata	Import	user32.wsprintfA	

Recordamos que el plugin HideDebugger tenía la opción para protegernos de los trucos de esta api



Pero antes de poner la tilde aprenderemos a pasarlo a mano y a ver como trabaja

Veamos la definición de la api SetUnhandledExceptionFilter en el WINAPI32

SetUnhandledExceptionFilter Quick Info

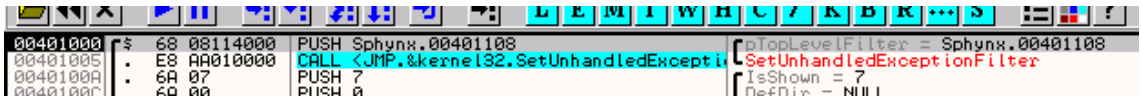
The **SetUnhandledExceptionFilter** function lets an application supersede the top-level exception handler that Win32 places at the top of each thread and process.

After calling this function, if an exception occurs in a process that is not being debugged, and the exception makes it to the Win32 unhandled exception filter, that filter will call the exception filter function specified by the *lpTopLevelExceptionFilter* parameter.

```
LPTOP_LEVEL_EXCEPTION_FILTER SetUnhandledExceptionFilter(
  LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter          // exception filter function
);
```

Bueno vemos que uno de los parámetros de la api, es la dirección de donde continuara ejecutándose, cuando el programa encuentre una excepción, siempre que el programa NO ESTE SIENDO DEBUGGEADO jeje, allí lo dice y para comprobar si esta siendo debuggeado o no utiliza un llamado a ZwQueryInformationProcess.

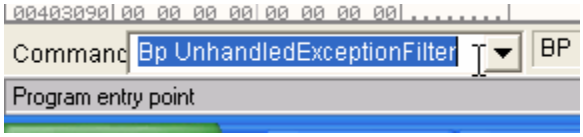
Bueno miremos el programa, allí mismo en el inicio ya llama a esta api

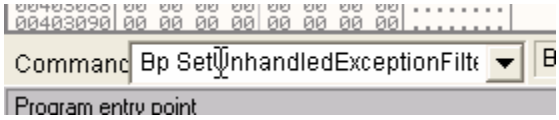


Como vemos el único parámetro, es la dirección adonde continuara ejecutándose el programa cuando halle una excepción, si es que no hay debugger jeje, ira a 401108 si hay debugger ira al tacho de basura.

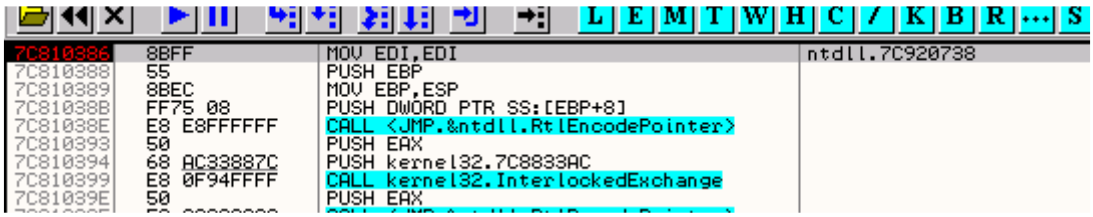
O sea ya veremos que un programa coloca manejadores de excepciones, para cuando se produce una de ellas, el programa siga corriendo a partir de la dirección que indica el manejador, pues esta api hace eso, fuerza al programa a que cuando encuentre una excepción, continúe a partir de la dirección que nos indica el parámetro, pero eso si siempre y cuando no haya un debugger.

Bueno pongamos un BP en las dos apis que trabajan juntas estas son SetUnhandledExceptionFilter y UnhandledExceptionFilter las hermanas jeje.

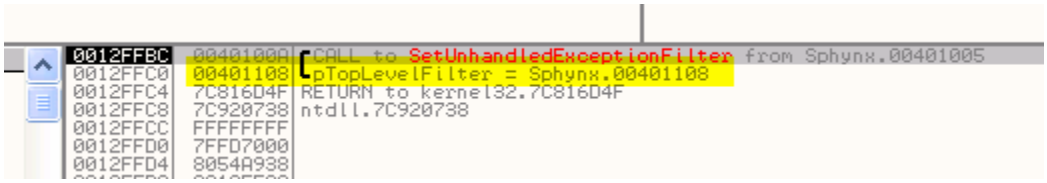




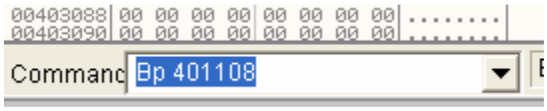
Ahora demos RUN



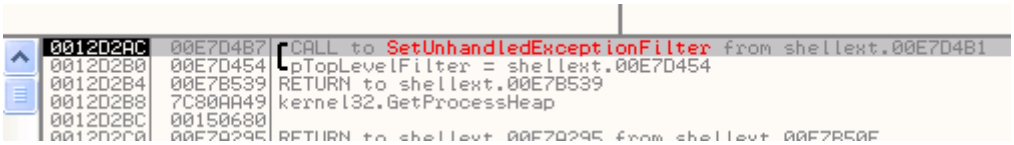
Para al inicio del programa cuando instala la dirección donde continuara cuando halle una excepción.



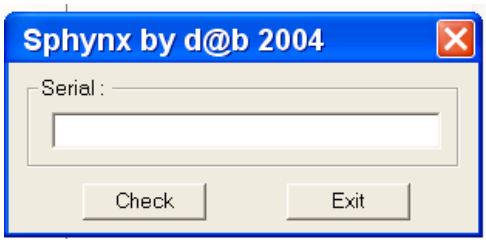
Como vemos la dirección es 401108 pongamos un BP alli



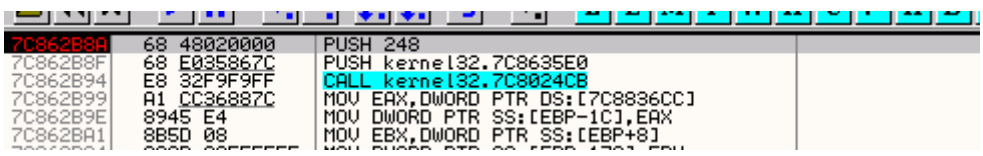
Ahora demos RUN y vemos que para en SetUnhandledExceptionFilter llamado desde dlls

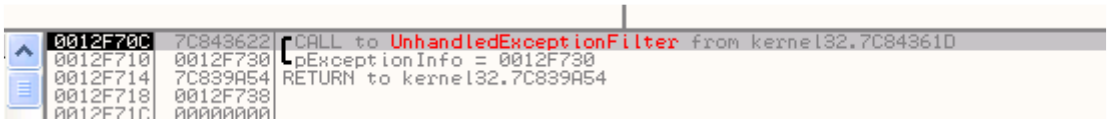


Como estas no nos interesan y ya esta colocado el manejador del programa le quito el bp a esta api.

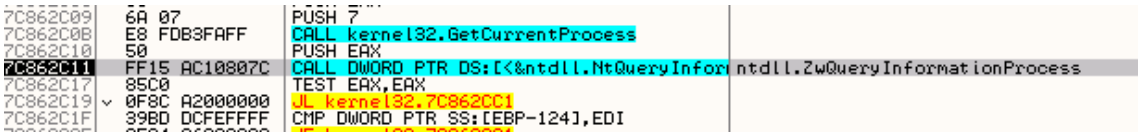


Ahora tipeo un serial falso y apreto CHECK y para en la otra api en UnhandledExceptionFilter, porque ha hallado una excepción que seguramente ha sido generada por el programa intencionalmente para llegar aquí y que esta api decida si esta siendo debuggeado o no y de esta forma ir a la zona caliente en 401108 o al tacho de residuos jeje.

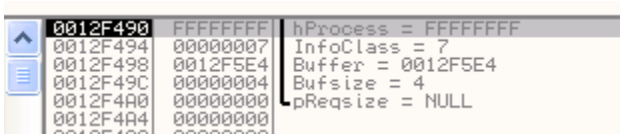




Esta api verificara si el proceso no esta siendo debuggeado y hará el salto a 401108, traceemos con f8 la api a ver como detecta el debugger.

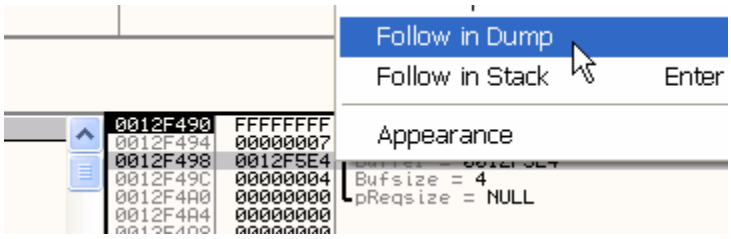


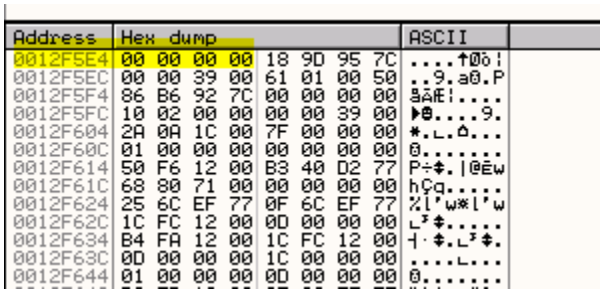
Llegamos, aquí este el segundo truco antidebugger que veremos hoy, es una forma de detectar el debugger que usa esta api y que puede usar un programa separadamente llamando a la api directamente ZwQueryInformationProcess con el parámetro INFO CLASS=7



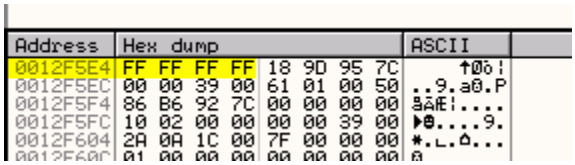
Esta api devuelve información del proceso que esta corriendo y con el parámetro INFOCLASS=7 devuelve la información en el buffer de si el proceso esta siendo debuggeado o no.

Veamos el buffer en el dump

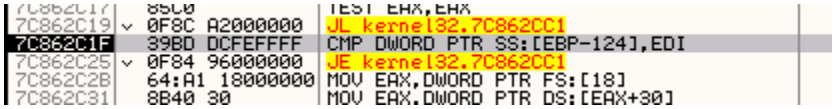




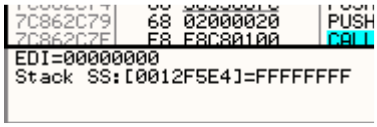
Ese es el buffer vimos que su tamaño es 4 bytes, y allí la api devolverá FFFFFFFF si esta siendo debuggeado el proceso o cero si no lo esta, pasemos con f8 la api a ver que guarda allí.



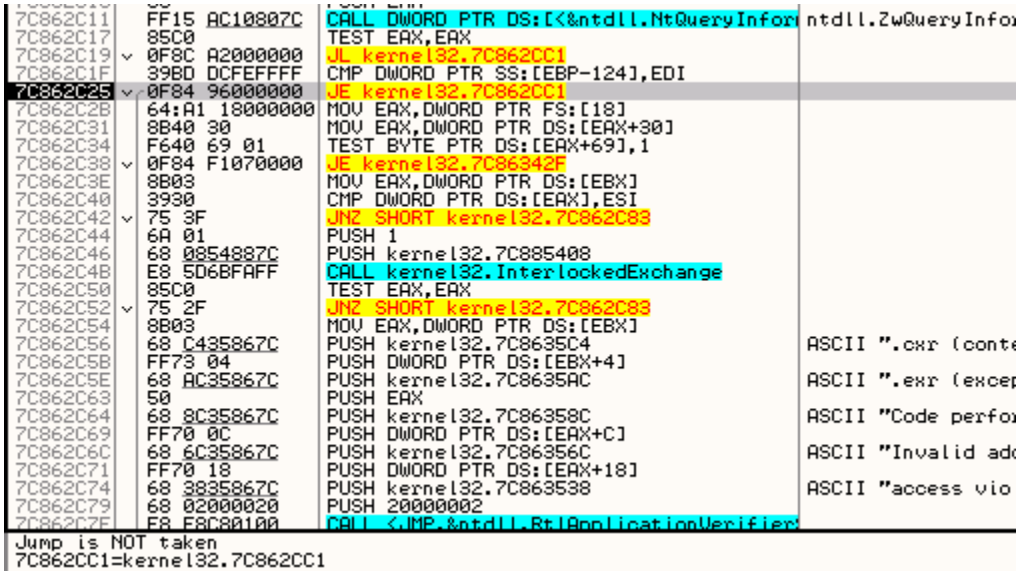
Allí vemos que devuelve FFFFFFFF o sea que hay debugger mi amigo jeje



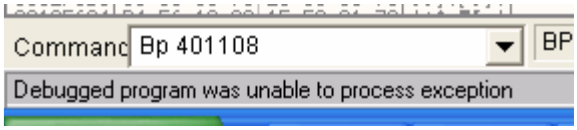
Vemos que luego viene una comparación que ve si el valor ese es cero o no, ya que EDI vale cero.



Y allí no salta



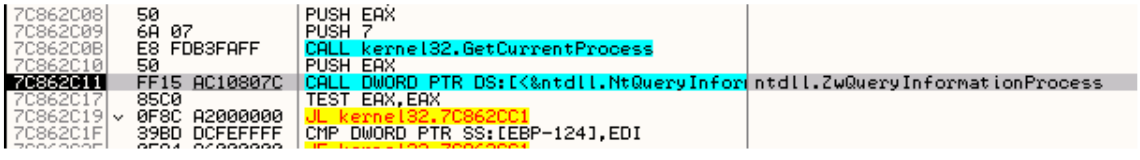
Al no ser igual a cero el buffer, el salto decisivo no salta y vamos al muere.



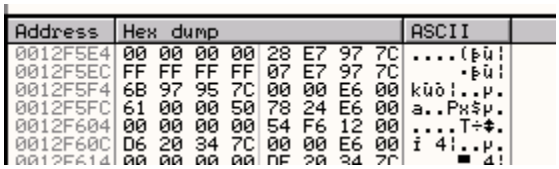
Vemos que se va al error y se cierra.

Ahora reiniciare y repetiré el proceso pero esta vez modificare el resultado de la api ZwQueryInformationProcess.

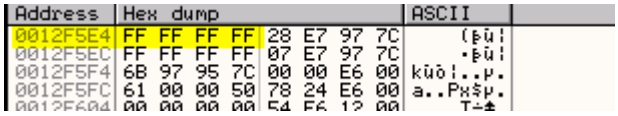
Reiniciemos y cuando llegamos a



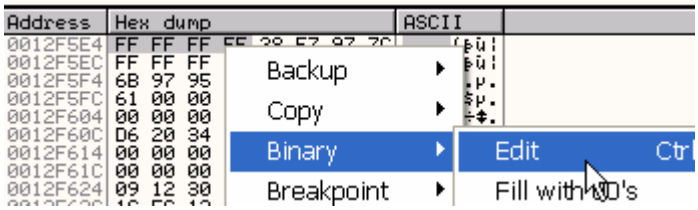
Vemos el buffer en el dump



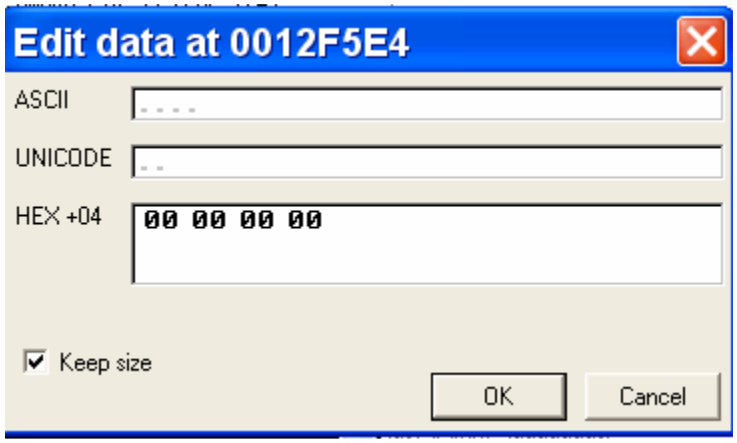
Pasamos la api con F8

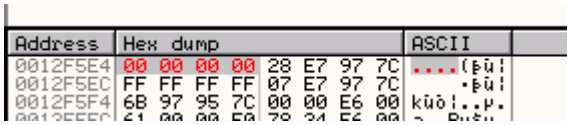


Y allí guardo al igual que antes el FFFFFFFF de que hay debugger lo cambio a cero

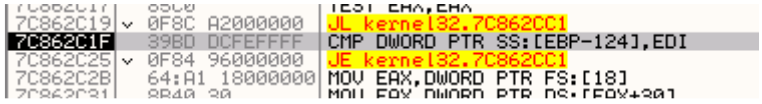


Cambio el contenido a todos ceros



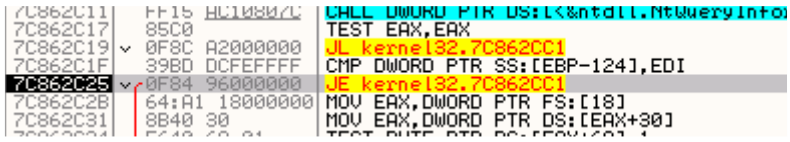


Ahora llego a la comparación

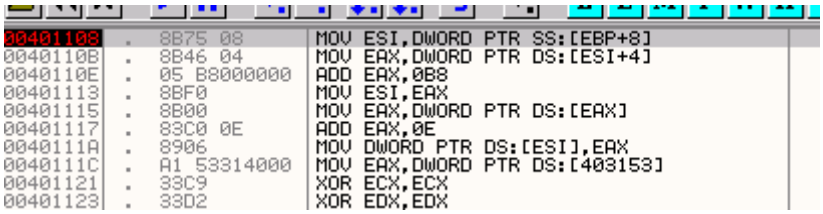


Como ambos miembros son cero

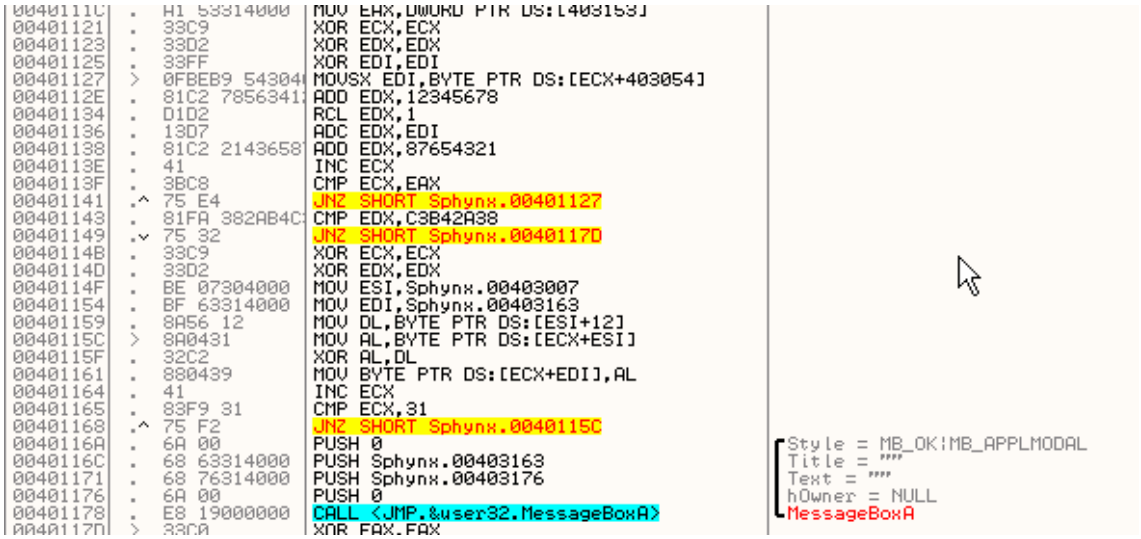
Ahora si saltara



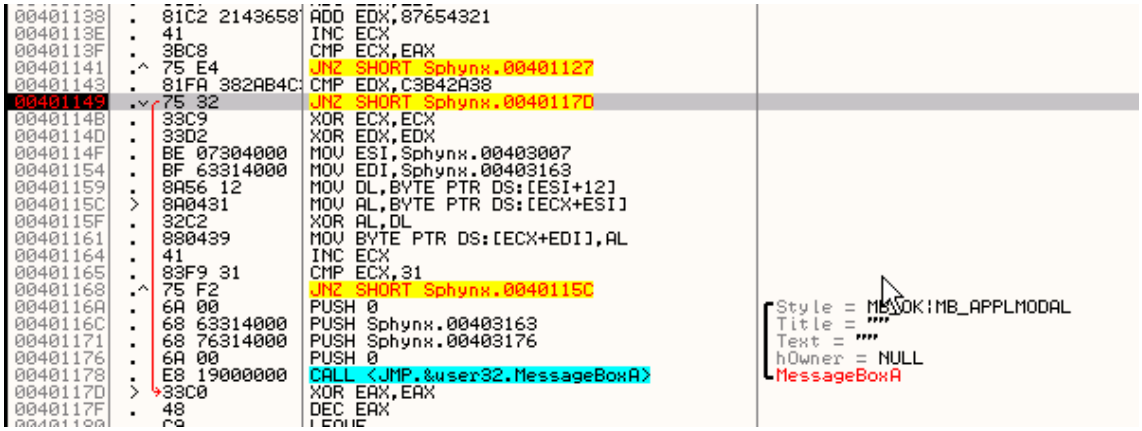
Y si doy RUN



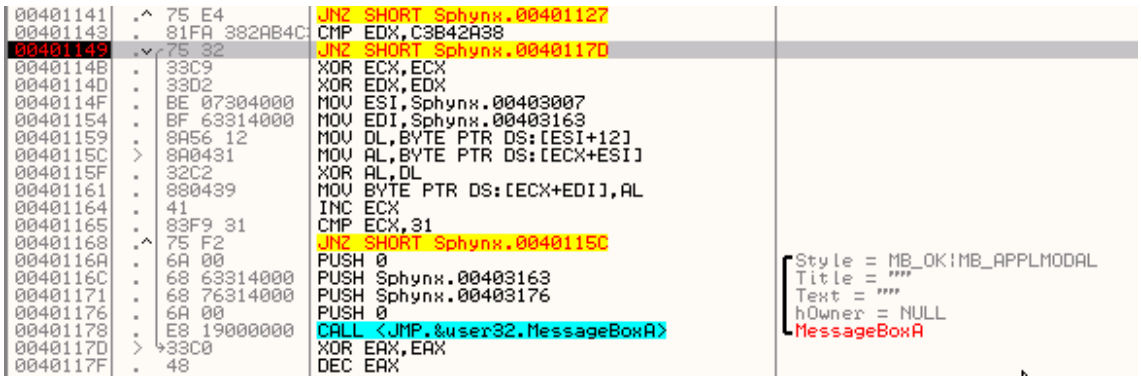
Vemos que ahora si llega a 401108 y mas abajo veo la parte caliente



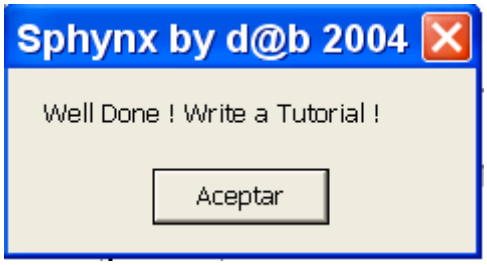
Y el messageboxa de correcto, pueden probar que



Al invertir este salto y forzarlo a que no salte, mostrara el mensaje correcto de que el crackme esta solucionado.



Ahora doy RUN



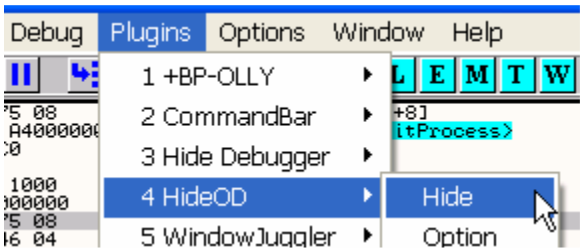
Bueno como he dicho todo esto puede ser evitado si ponemos la tilde en el plugin HideDebugger en UnhandledExceptionTricks y cierro y vuelvo a abrir el OLLYDBG veo que correrá perfectamente.

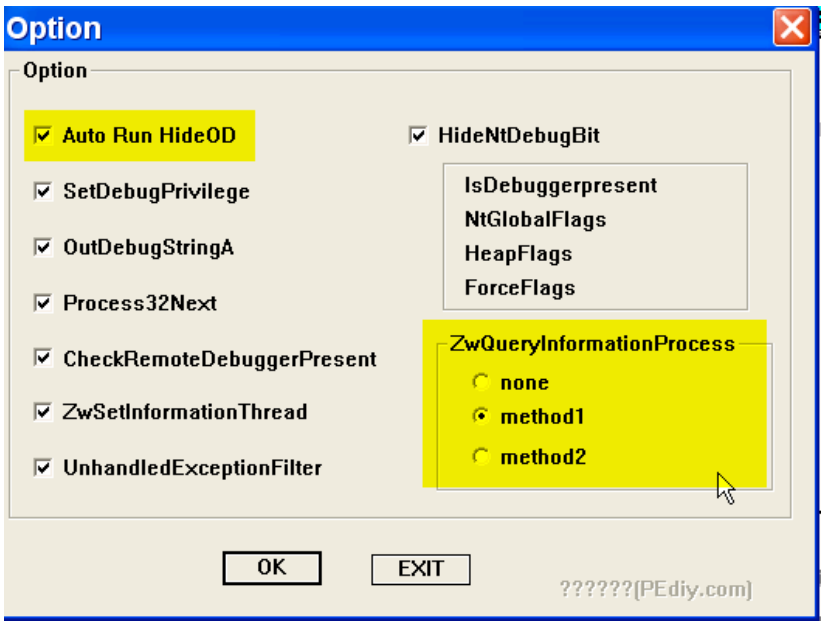
El único tema que queda por ver es como evitar la detección por la api ZwQueryInformationProcess, cuando esta es llamada en forma directa y no dentro de la api UnhandledExceptionFilter.

Por supuesto ya vimos como pasarlo a mano cambiando a cero el valor de retorno del buffer, pero existe algún plugin que lo haga automáticamente, así nos olvidamos de esto?

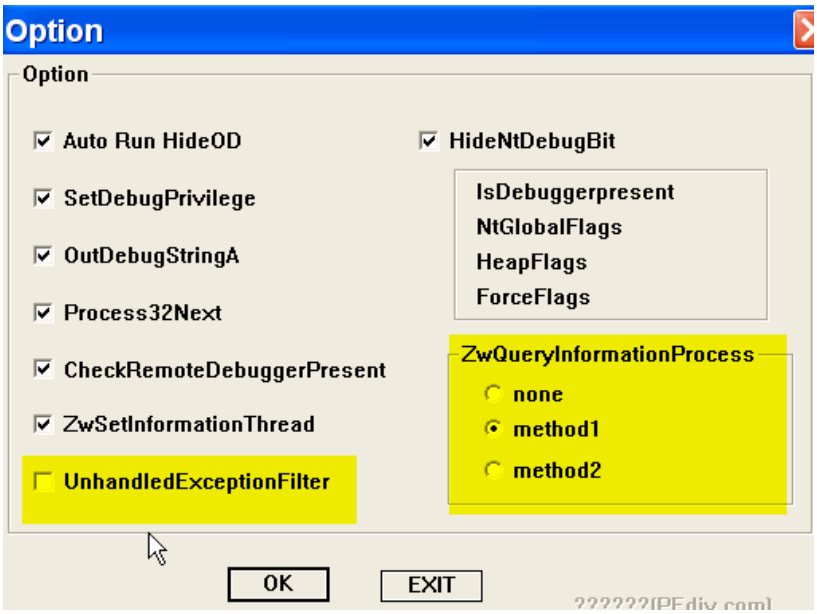
http://www.pediy.com/tools/Debuggers/ollydbg/plugin/hideOD/hideod.rar

Este plugin permite que el programa sin tener que cambiarlo a mano nos proteja de la deteccion de esta api.





Vemos que tiene muchas mas opciones que el otro Hidedebugger, así que puede complementarse, es importante si se activa cualquier opción, siempre poner la tilde también en AUTORUNHIDEOD para que proteja automáticamente y no tener que apretar HIDE cada vez que arrancamos OLLYDBG.



Vemos que aun quitando la protección contra UnhandledExceptionFilter, como esta detecta el debugger por medio de ZwQueryInformationProcess igual corre ya que el plugin nos protege también de esta última que es la usada para detectar si hay debugger o no como vimos.

Bueno ya vimos como protegernos de estos dos trucos manualmente y por medio de plugins, nuestro OLLYDBG parcheado y con ambos plugins es casi una fortaleza jeje igual seguiremos viendo los trucos antidebugging que faltan, que aun hay mas.

Hasta la parte 23
Ricardo Narvaja
31 de diciembre de 2005