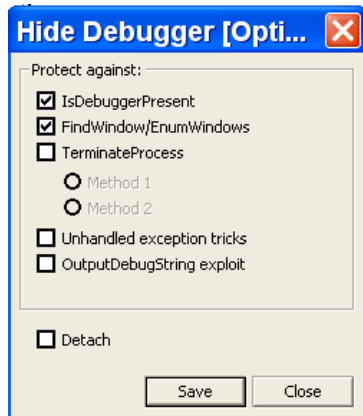


## INTRODUCCION AL CRACKING CON OLLYDBG parte 22

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.

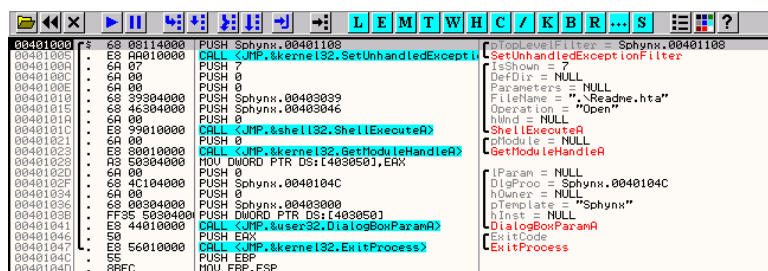
Es de mencionar que siempre estamos dando por sabido todo lo visto anteriormente y que iniciaremos este tute con el OLLYDBG que parcheamos en la parte 21, que se llama NVP11, con el plugin HideDebugger 1.23f por ahora configurado así.



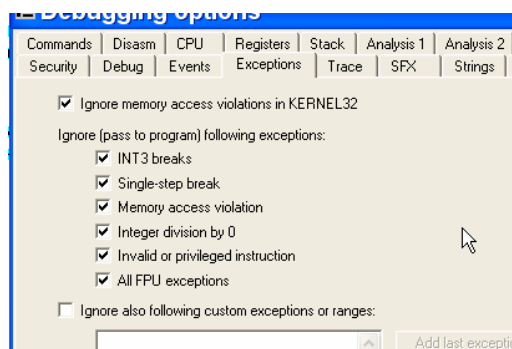
Allí vemos que hay una tilde llamada UNHANDLED EXCEPTION TRICKS, bueno estudiaremos este truco, a su vez que trabaja conjuntamente con otro truco que podemos encontrar también usado en forma separada, es el llamado a la api ZwQueryInformationProcess para detectar el debugger.

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.

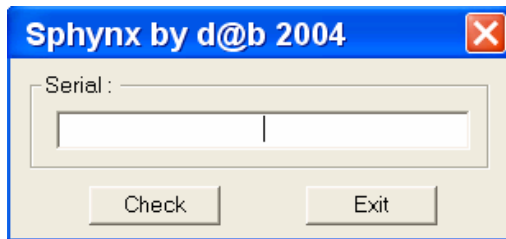
Otra cosa de aclarar que estos crackmes que no estamos solucionando, los veremos mas adelante, por ejemplo este necesita para solucionarse usar el método de fuerza bruta para hallar el serial correcto, y es posible que lo usemos como ejemplo cuando veamos ese método de ataque, por ahora solo lo haremos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

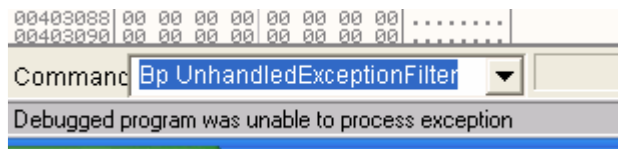


Doy RUN



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

Coloco un serial falso y apreto check

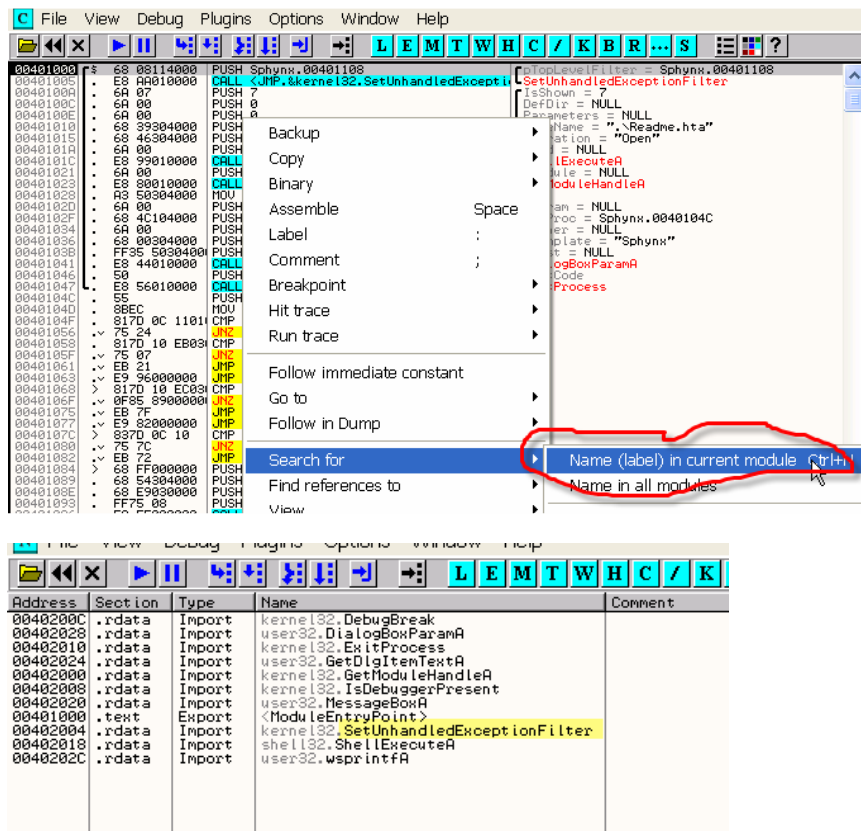


Me dice que no es posible procesar la excepción y se cierra si doy RUN.

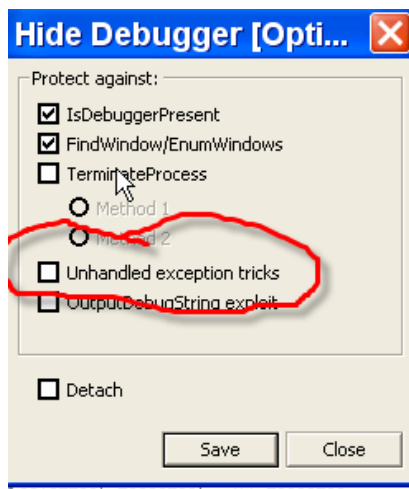


Bueno me doy cuenta de que si lo pruebo fuera de OLLYDBG, el crackme no se cierra, continua funcionando y dándonos la posibilidad de seguir intentando con diferentes seriales.

Bueno reiniciémoslo y veamos las apis que utiliza



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

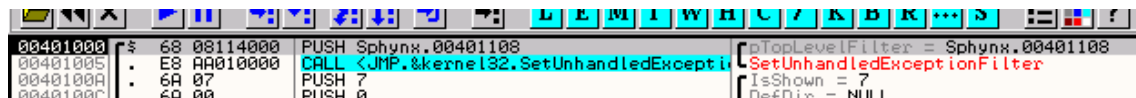
The **SetUnhandledExceptionFilter** function lets an application supersede the top-level exception handler that Win32 places at the top of each thread and process.

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.

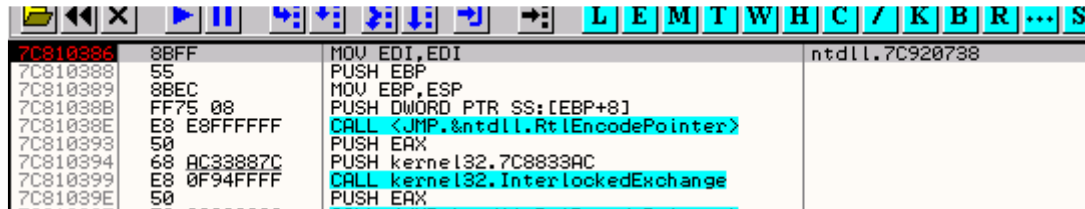
O sea ya veremos que un programa coloca manejadores de excepciones, para cuando se produce una de ellas, el programa siga corriendo a partir de la dirección que indica el manejador, pues esta api hace eso, fuerza al programa a que cuando encuentre una excepción, continúe a partir de la dirección que 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.

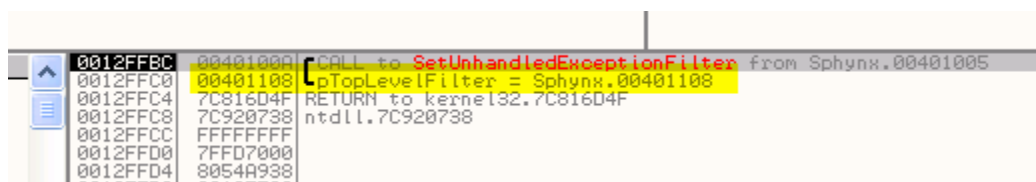




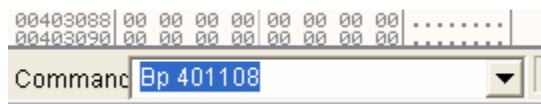
Ahora demos RUN



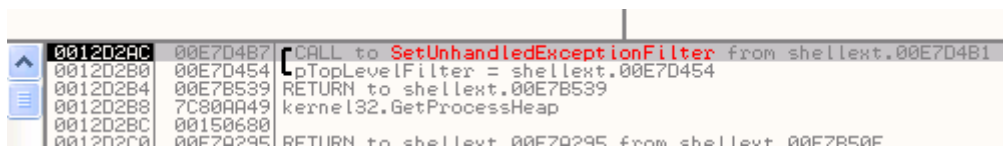
Para al inicio del programa cuando instala la dirección donde continuara cuando halle una excepción.



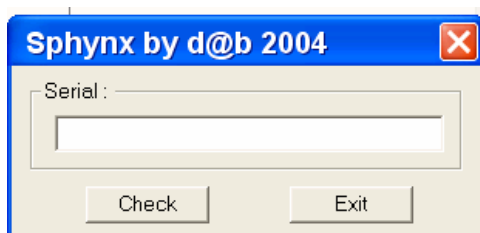
Como vemos la dirección es 401108 pongamos un BP allí



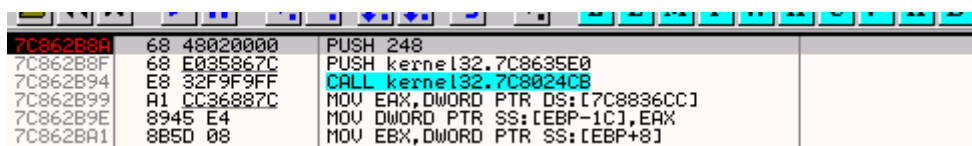
Ahora demos RUN y vemos que para en SetUnhandledExceptionFilter llamado desde dlls



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.



0012F70C	7C843622	CALL to UnhandledExceptionFilter from kernel32.7C84361D
0012F710	0012F730	pExceptionInfo = 0012F730
0012F714	7C839A54	RETURN to kernel32.7C839A54
0012F718	0012F738	
0012F71C	00000000	

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.

7C862C09	6A 07	PUSH 7
7C862C0B	E8 FDB3FAFF	CALL kernel32.GetCurrentProcess
7C862C0D	50	PUSH EAX
7C862C11	FF15 AC10807C	CALL DWORD PTR DS:[<&ntdll.NtQueryInformationProcess
7C862C17	85C0	TEST EAX,EAX
7C862C19	0F8C A2000000	JL kernel32.7C862CC1
7C862C1F	39BD DCFEFFFF	CMP DWORD PTR SS:[EBP-124],EDI

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

0012F490	FFFFFFFF	hProcess = FFFFFFFF
0012F494	00000007	InfoClass = 7
0012F498	0012F5E4	Buffer = 0012F5E4
0012F49C	00000004	Bufsize = 4
0012F4A0	00000000	pReqsize = NULL
0012F4A4	00000000	
0012F4A8	00000000	

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

0012F490	FFFFFFFF	hProcess = FFFFFFFF
0012F494	00000007	InfoClass = 7
0012F498	0012F5E4	Buffer = 0012F5E4
0012F49C	00000004	Bufsize = 4
0012F4A0	00000000	pReqsize = NULL
0012F4A4	00000000	
0012F4A8	00000000	

Address	Hex dump	ASCII
0012F5E4	00 00 00 00	....
0012F5EC	00 00 39 00	..9.a0.P
0012F5F4	86 B6 92 7C	3AE!....
0012F5FC	10 02 00 00	00 00 39 00
0012F604	2A 0A 1C 00	*.L.A...
0012F60C	01 00 00 00	00 00 00 00
0012F614	50 F6 12 00	P=*. 0Ew
0012F61C	68 80 71 00	hCq.....
0012F624	25 6C EF 77	%l'w*!w
0012F62C	1C FC 12 00	L?*.L?*
0012F634	B4 FA 12 00	+.L?*
0012F63C	00 00 00 00	....L...
0012F644	01 00 00 00	00 00 00 00

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.

Address	Hex dump	ASCII
0012F5E4	FF FF FF FF	....
0012F5EC	00 00 39 00	..9.a0.P
0012F5F4	86 B6 92 7C	3AE!....
0012F5FC	10 02 00 00	00 00 39 00
0012F604	2A 0A 1C 00	*.L.A...
0012F60C	01 00 00 00	00 00 00 00

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

7C862C17	85C0	TEST EAX,EAX	
7C862C19	0F8C A2000000	JL kernel32.7C862CC1	
7C862C1F	39BD DCFEFFFF	CMP DWORD PTR SS:[EBP-124],EDI	
7C862C25	0F84 96000000	JE kernel32.7C862CC1	
7C862C2B	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]	
7C862C31	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]	

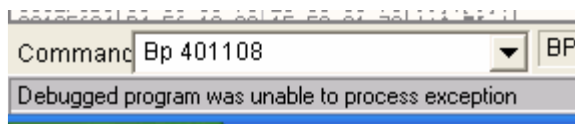
Vemos que luego viene una comparación que ve si el valor ese es cero o no, ya que EDI vale cero.

7C862C79	68 02000020	PUSH	
7C862C7E	F8 F8C8A100	CALL	
EDI=00000000			
Stack SS:[0012F5E4]=FFFFFFFF			

Y allí no salta

7C862C11	FF15 AC10807C	CALL DWORD PTR DS:[<&ntdll.NtQueryInformationProcess]	ntdll.ZwQueryInfo
7C862C17	85C0	TEST EAX,EAX	
7C862C19	0F8C A2000000	JL kernel32.7C862CC1	
7C862C1F	39BD DCFEFFFF	CMP DWORD PTR SS:[EBP-124],EDI	
7C862C25	0F84 96000000	JE kernel32.7C862CC1	
7C862C2B	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]	
7C862C31	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]	
7C862C34	F640 69 01	TEST BYTE PTR DS:[EAX+69],1	
7C862C38	0F84 F1070000	JE kernel32.7C86342F	
7C862C3E	8B03	MOV EAX,DWORD PTR DS:[EBX]	
7C862C40	3930	CMP DWORD PTR DS:[EAX],ESI	
7C862C42	75 3F	JNZ SHORT kernel32.7C862C83	
7C862C44	6A 01	PUSH 1	
7C862C46	68 0854887C	PUSH kernel32.7C885408	
7C862C4B	E8 5D6BFAFF	CALL kernel32.InterlockedExchange	
7C862C50	85C0	TEST EAX,EAX	
7C862C52	75 2F	JNZ SHORT kernel32.7C862C83	
7C862C54	8B03	MOV EAX,DWORD PTR DS:[EBX]	
7C862C56	68 C435867C	PUSH kernel32.7C8635C4	ASCII ".chr (cont
7C862C5B	FF73 04	PUSH DWORD PTR DS:[EBX+4]	
7C862C5E	68 AC35867C	PUSH kernel32.7C8635AC	ASCII ".exr (excep
7C862C63	50	PUSH EAX	
7C862C64	68 8C35867C	PUSH kernel32.7C86358C	ASCII "Code perfo
7C862C69	FF70 0C	PUSH DWORD PTR DS:[EAX+C]	
7C862C6C	68 6C35867C	PUSH kernel32.7C86356C	ASCII "Invalid ad
7C862C71	FF70 18	PUSH DWORD PTR DS:[EAX+18]	
7C862C74	68 3835867C	PUSH kernel32.7C863538	ASCII "access vio
7C862C79	68 02000020	PUSH 20000002	
7C862C7E	F8 F8C8A100	CALL <IMP.&ntdll.RtlApplicationVerifier	
Jump is NOT taken			
7C862CC1=kernel32.7C862CC1			

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

7C862C08	50	PUSH EAX	
7C862C09	6A 07	PUSH 7	
7C862C0B	E8 FDB3FAFF	CALL kernel32.GetCurrentProcess	
7C862C10	50	PUSH EAX	
7C862C11	FF15 AC10807C	CALL DWORD PTR DS:[<&ntdll.NtQueryInformationProcess]	ntdll.ZwQueryInformationProcess
7C862C17	85C0	TEST EAX,EAX	
7C862C19	0F8C A2000000	JL kernel32.7C862CC1	
7C862C1F	39BD DCFEFFFF	CMP DWORD PTR SS:[EBP-124],EDI	

Vemos el buffer en el dump

Address	Hex dump	ASCII
0012F5E4	00 00 00 00 28 E7 97 7C	....(pü!
0012F5EC	FF FF FF FF 07 E7 97 7C	..ü!
0012F5F4	68 97 95 7C 00 00 E6 00	küö!..µ.
0012F5FC	61 00 00 50 78 24 E6 00	a..Px\$µ.
0012F604	00 00 00 00 54 F6 12 00	....T=\$.
0012F60C	D6 20 34 7C 00 00 E6 00	i 4!..µ.
0012F614	00 00 00 00 nF 20 34 7C	.. 4!

Pasamos la api con F8

Address	Hex dump	ASCII
0012F5E4	FF FF FF FF	28 E7 97 7C (pü!
0012F5EC	FF FF FF FF	07 E7 97 7C .pü!
0012F5F4	6B 97 95 7C 00 00 E6 00	küò!.,p.
0012F5FC	61 00 00 50 78 24 E6 00	a..Pü\$µ.
0012F604	00 00 00 00 E4 F4 12 00	T=2

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

Address	Hex dump	ASCII
0012F5E4	FF FF FF FF	28 E7 97 7C (pü!
0012F5EC	FF FF FF FF	07 E7 97 7C .pü!
0012F5F4	6B 97 95 7C 00 00 E6 00	küò!.,p.
0012F5FC	61 00 00 50 78 24 E6 00	a..Pü\$µ.
0012F604	00 00 00 00 E4 F4 12 00	T=2
0012F60C	06 20 34	
0012F614	00 00 00	
0012F61C	00 00 00	
0012F624	09 12 30	
0012F62C	1C 7C 12	

Cambio el contenido a todos ceros

### Edit data at 0012F5E4

ASCII

UNICODE

HEX +04

☒ Keep size

Address	Hex dump	ASCII
0012F5E4	00 00 00 00	28 E7 97 7C ....(pü!
0012F5EC	FF FF FF FF	07 E7 97 7C .pü!
0012F5F4	6B 97 95 7C 00 00 E6 00	küò!.,p.
0012F5FC	61 00 00 50 78 24 E6 00	a..Pü\$µ.

Ahora llego a la comparación

7C862C17	0F8C A2000000	JL kernel32.7C862CC1
7C862C19	39BD DCFEFFFF	CMP DWORD PTR SS:[EBP-124],EDI
7C862C1F	0F84 96000000	JE kernel32.7C862CC1
7C862C25	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
7C862C2B	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]

Como ambos miembros son cero

EDI=00000000  
Stack SS:[0012F5E4]=00000000

Ahora si saltara

7C862C17	0F8C A2000000	JL kernel32.7C862CC1
7C862C19	39BD DCFEFFFF	CMP DWORD PTR SS:[EBP-124],EDI
7C862C1F	0F84 96000000	JE kernel32.7C862CC1
7C862C25	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
7C862C2B	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]

Y si doy RUN

00401108	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
0040110B	8B46 04	MOV EAX,DWORD PTR DS:[ESI+4]
0040110E	05 B8000000	ADD EAX,0B8
00401113	8BF0	MOV ESI,EAX
00401115	8B00	MOV EAX,DWORD PTR DS:[EAX]
00401117	83C0 0E	ADD EAX,0E
0040111A	8906	MOV DWORD PTR DS:[ESI],EAX
0040111C	A1 53314000	MOV EAX,DWORD PTR DS:[403153]
00401121	33C9	XOR ECX,ECX
00401123	33D2	XOR EDX,EDX

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

0040111C	A1 53314000	MOV EAX,DWORD PTR DS:[403153]
00401121	33C9	XOR ECX,ECX
00401123	33D2	XOR EDX,EDX
00401125	33FF	XOR EDI,EDI
00401127	0FBEB9 54304	MOVSX EDI,BYTE PTR DS:[ECX+403054]
0040112E	81C2 7856341	ADD EDX,12345678
00401134	D1D2	RCL EDX,1
00401136	13D7	ADC EDX,EDI
00401138	81C2 2143658	ADD EDX,87654321
0040113E	41	INC ECX
0040113F	3BC8	CMP ECX,EAX
00401141	75 E4	JNZ SHORT Sphynx.00401127
00401143	81FA 382AB4C	CMP EDX,C3B42A38
00401149	75 32	JNZ SHORT Sphynx.00401170
0040114B	33C9	XOR ECX,ECX
0040114D	33D2	XOR EDX,EDX
0040114F	BE 07304000	MOV ESI,Sphynx.00403007
00401154	BF 63314000	MOV EDI,Sphynx.00403163
00401159	8A56 12	MOV DL,BYTE PTR DS:[ESI+12]
0040115C	8A0431	MOV AL,BYTE PTR DS:[ECX+ESI]
0040115F	32C2	XOR AL,DL
00401161	880439	MOV BYTE PTR DS:[ECX+EDI],AL
00401164	41	INC ECX
00401165	83F9 31	CMP ECX,31
00401168	75 F2	JNZ SHORT Sphynx.0040115C
0040116A	6A 00	PUSH 0
0040116C	68 63314000	PUSH Sphynx.00403163
00401171	68 76314000	PUSH Sphynx.00403176
00401176	6A 00	PUSH 0
00401178	E8 19000000	CALL <JMP.&user32.MessageBoxA>
0040117D	33C9	XOR EAX,EAX

Style = MB\_OK|MB\_APPLMODAL  
 Title = ""  
 Text = ""  
 hOwner = NULL  
 MessageBoxA

Y el messageboxa de correcto, pueden probar que

00401138	81C2 2143658	ADD EDX,87654321
0040113E	41	INC ECX
0040113F	3BC8	CMP ECX,EAX
00401141	75 E4	JNZ SHORT Sphynx.00401127
00401143	81FA 382AB4C	CMP EDX,C3B42A38
00401149	75 32	JNZ SHORT Sphynx.00401170
0040114B	33C9	XOR ECX,ECX
0040114D	33D2	XOR EDX,EDX
0040114F	BE 07304000	MOV ESI,Sphynx.00403007
00401154	BF 63314000	MOV EDI,Sphynx.00403163
00401159	8A56 12	MOV DL,BYTE PTR DS:[ESI+12]
0040115C	8A0431	MOV AL,BYTE PTR DS:[ECX+ESI]
0040115F	32C2	XOR AL,DL
00401161	880439	MOV BYTE PTR DS:[ECX+EDI],AL
00401164	41	INC ECX
00401165	83F9 31	CMP ECX,31
00401168	75 F2	JNZ SHORT Sphynx.0040115C
0040116A	6A 00	PUSH 0
0040116C	68 63314000	PUSH Sphynx.00403163
00401171	68 76314000	PUSH Sphynx.00403176
00401176	6A 00	PUSH 0
00401178	E8 19000000	CALL <JMP.&user32.MessageBoxA>
0040117D	33C9	XOR EAX,EAX
0040117F	48	DEC EAX

Style = MB\_OK|MB\_APPLMODAL  
 Title = ""  
 Text = ""  
 hOwner = NULL  
 MessageBoxA

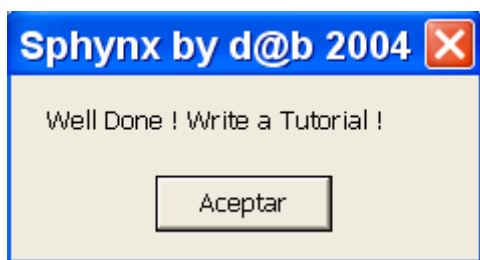
Al invertir este salto y forzarlo a que no salte, mostrara el mensaje correcto de que el crackme esta solucionado.

00401141	75 E4	JNZ SHORT Sphynx.00401127
00401143	81FA 382AB4C	CMP EDX,C3B42A38
00401149	75 32	JNZ SHORT Sphynx.00401170
0040114B	33C9	XOR ECX,ECX
0040114D	33D2	XOR EDX,EDX
0040114F	BE 07304000	MOV ESI,Sphynx.00403007
00401154	BF 63314000	MOV EDI,Sphynx.00403163
00401159	8A56 12	MOV DL,BYTE PTR DS:[ESI+12]
0040115C	8A0431	MOV AL,BYTE PTR DS:[ECX+ESI]
0040115F	32C2	XOR AL,DL
00401161	880439	MOV BYTE PTR DS:[ECX+EDI],AL
00401164	41	INC ECX
00401165	83F9 31	CMP ECX,31
00401168	75 F2	JNZ SHORT Sphynx.0040115C
0040116A	6A 00	PUSH 0
0040116C	68 63314000	PUSH Sphynx.00403163
00401171	68 76314000	PUSH Sphynx.00403176
00401176	6A 00	PUSH 0
00401178	E8 19000000	CALL <JMP.&user32.MessageBoxA>
0040117D	33C9	XOR EAX,EAX
0040117F	48	DEC EAX

Style = MB\_OK|MB\_APPLMODAL  
 Title = ""  
 Text = ""  
 hOwner = NULL  
 MessageBoxA



Ahora doy RUN



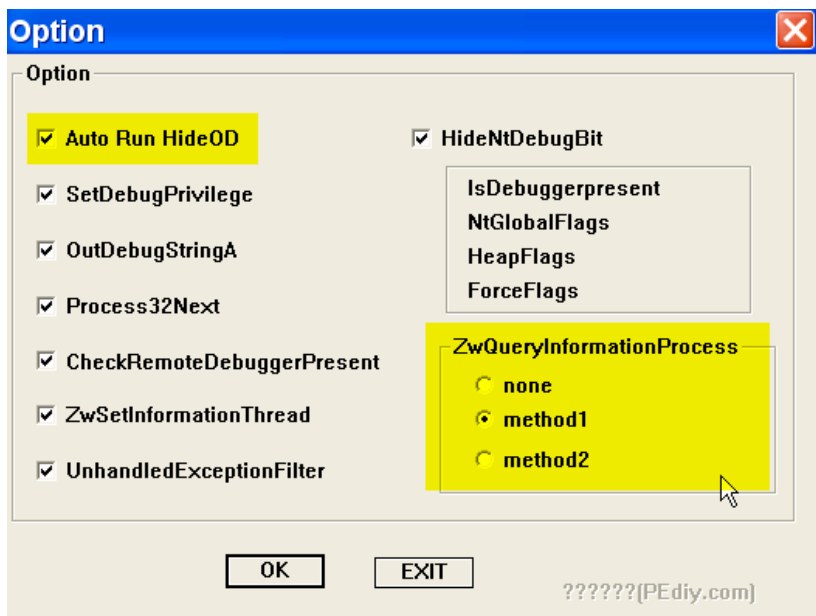
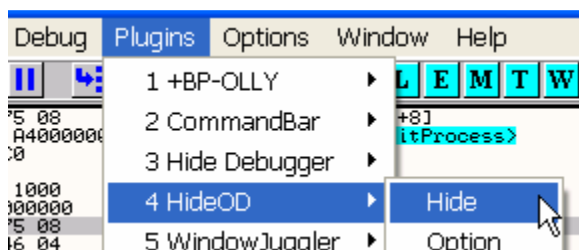
Bueno como he dicho todo esto puede ser evitado si ponemos la tilde en el plugin HideDebugger en UnhandledExceptionTricks y cierro y vuelvo a abrir el OLLYDBG veo que correrá perfectamente.

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.

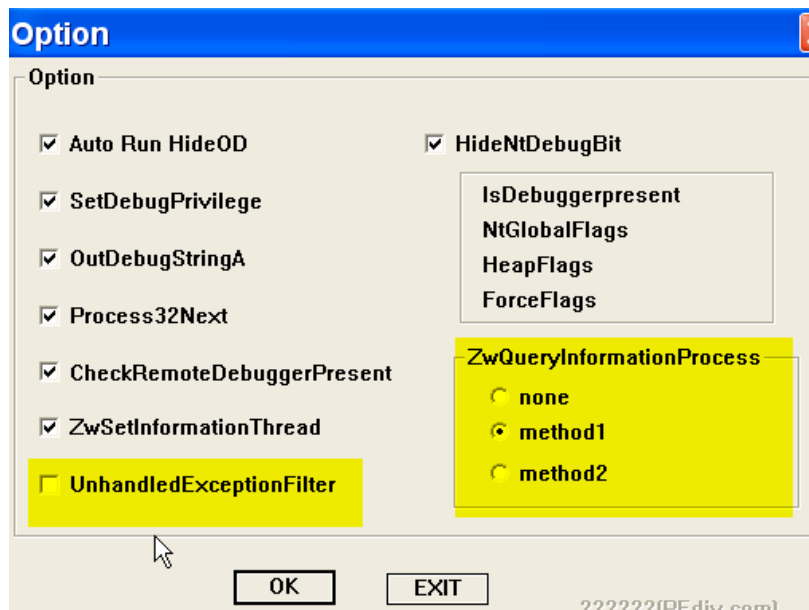
Por supuesto ya vimos como pasarlo a mano cambiando a cero el valor de retorno del buffer, pero existe algún plugin que lo haga automáticamente, así nos olvidamos de esto?

<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