

# CracksLatinoS! 2009



## CracksLatinoS! 2010

### -\* Eliminando la Shell de Sentinel \*-

Programa	Ferrocad
Download	<a href="http://www.iproin.com/Descargas/Instalacion%20FerroCad.zip">http://www.iproin.com/Descargas/Instalacion%20FerroCad.zip</a>
Descripción	
Herramientas	OllyDbg v2.0, IDA, PUPE, Import Reconstructor.
Dificultad	Media
Compresor/Compilador	
Protección	Sentinel Shell
Objetivos	Hacer correr el programa sin mochila
Cracker	Lionel <a href="mailto:lionelgomezdu@yahoo.es">lionelgomezdu@yahoo.es</a>
Fecha:	26/01/10
Tutorial n°	

### .\*. Introduccion \*.\*.

Hacia mucho tiempo que no escribía nada sobre todo por falta de tiempo, y aunque sigo con el problema del tiempo creo que ya iba siendo hora de devolver parte de lo que la lista me ha dado durante tantos años.

En esta ocasión el tutorial va a tratar sobre una protección con mochila, pero un tanto particular. Viene a ser algo similar a lo que es el HASP Envelope, pero en este caso aplicado a la mochila Sentinel. Es lo que llaman SENTINEL SHELL. Tengo que decir que sólo llegar a la conclusión de que el tipo de protección era ésta me llevó mucho tiempo, ya que la información que hay circulando por la red es mínima, y desde luego, no he encontrado nada en español.

Una vez que pude averiguar a qué me estaba enfrentando me vino muy bien un tutorial escrito en el año 2001 por un tal CyberHeg llamado Breaking the Shell, aunque la versión del Shell a la que nos vamos a enfrentar tiene algún truco más que la analizada en ese caso.

Vamos pues a comenzar el trabajo. Lo único que quiero aclarar es que voy a utilizar la nueva versión de Olly, la 2.0, que tiene como inconveniente el que no tiene plugins, pero así será más divertido y tendremos que enfrentarnos a las trampas a pecho descubierto.

### .\*. Explorando el objetivo \*.\*.

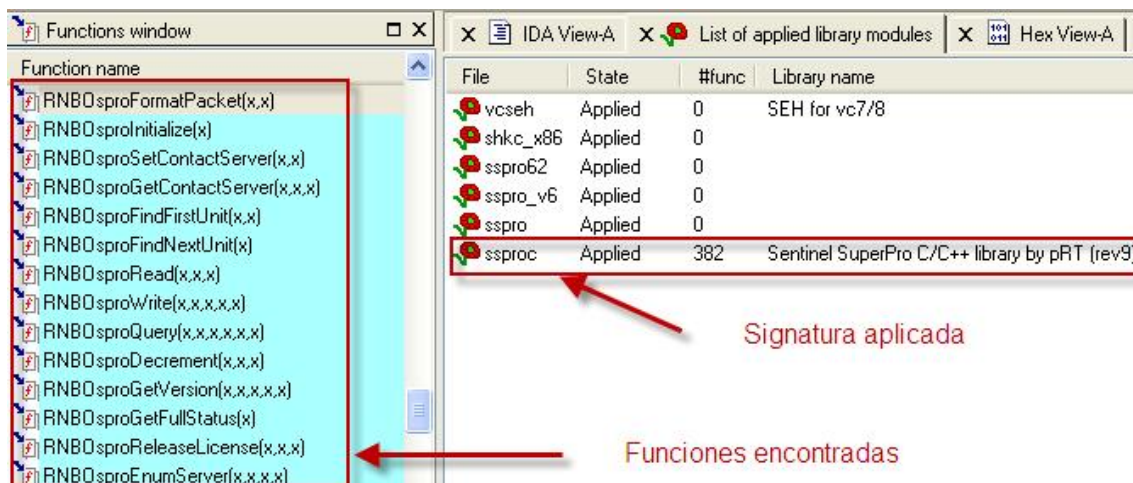
Tras instalar el programa lo ejecutamos normalmente y vemos que nos aparece la siguiente ventana:



Sabemos que utiliza una protección con mochila Sentinel porque en la página de descarga del programa aparece un enlace para bajarnos los drivers correspondientes. De todas maneras vamos a pasar el ejecutable por el RDG Packer Detector para ver si nos da alguna pista más, pero parece que no:



No nos queda más remedio que agarrarnos a lo único que sabemos e intentar exprimirlo al máximo. Lo que haremos antes de cargarlo en Olly es abrirlo con IDA, cargar un archivo de firmas de Sentinel y que reconozca las funciones de la mochila para así poder analizarlo con Olly:



Ahí vemos todas las funciones que ha encontrado. Ahora podemos crear un MAP file para cargarlo con algún plugin de Olly, o bien como haré yo (ya que con este Olly no hay plugins), apuntar a mano las direcciones y nombre de cada función y en Olly añadir una etiqueta a mano en cada una y ponerle un BP a cada una:

Names in FerroCad			
Address	Section	Type	Name
00FF4800	00000006	Label	RNBOFormatPacket(x,x)
00FF4840	00000006	Label	RNBOInitialize(x)
00FF4880	00000006	Label	RNBOSetContactServer(x,x)
00FF4900	00000006	Label	RNBGetContactServer(x,x,x)
00FF4940	00000006	Label	RNBFindFirstUnit(x,x)
00FF4980	00000006	Label	RNBFindNextUnit(x)
00FF4A00	00000006	Label	RNBRead(x,x,x)
00FF4A40	00000006	Label	RNBWrite(x,x,x,x,x)
00FF4A80	00000006	Label	RNBQuery(x,x,x,x,x,x)
00FF4B00	00000006	Label	RNBDecrement(x,x,x)
00FF4B40	00000006	Label	RNBGetVersion(x,x,x,x,x)
00FF4B80	00000006	Label	RNBGetFullStatus(x)
00FF4C00	00000006	Label	RNBReleaseLicense(x,x,x)
00FF4C40	00000006	Label	RNBEnumServer(x,x,x,x)

Y ahora comienza lo que para mí fue lo realmente complicado. Al cargar el programa enseguida empieza a ejecutar un gran bucle que empieza en la dirección FF2861 y termina en FF2E0D. En cada pasada ECX toma un valor diferente, y dependiendo de ese valor ejecuta unos comandos u otros de ese bucle.

```

00FF2861 > A1 7857FE00 MOV EAX, DWORD PTR DS:[0FE5778]
00FF2862 > 8B20 7057FE00 MOV EBP, DWORD PTR DS:[0FE577C]
00FF286C > 66:8B40 00 MOV CX, WORD PTR SS:[EBP]
00FF2870 > 83C5 08 ADD EBP, 8
00FF2873 > 66:8900 50FD MOV WORD PTR DS:[0FEFD50], CX
00FF287A > 66:8B75 FA MOV SI, WORD PTR SS:[EBP-6]
00FF287E > 8B00 50FDFE0 MOV ECX, DWORD PTR DS:[0FEFD50], SI
00FF2884 > 66:8935 52FD MOV WORD PTR DS:[0FEFD52], SI
00FF2888 > 8B55 FC MOV ECX, DWORD PTR SS:[EBP-4]
00FF2893 > 01F1 FFF00000 AND ECX, 0000FFFF
00FF2894 > 81F9 10590000 CMP ECX, 5910
00FF289A > 8915 48FDFE0 MOV DWORD PTR DS:[0FEFD48], EDX
00FF28A0 > 892D 7C57FE0 MOV DWORD PTR DS:[0FE577C], EBP
00FF28A6 > 0F8F 3D020000 JG 00FF2AE9
00FF28AC > 0F84 6D040000 JE 00FF2D1F
00FF28B2 > 81F9 92190000 CMP ECX, 1992
00FF28B8 > 0F8F 5E010000 JG 00FF2A1C
00FF28BE > 0F84 39010000 JE 00FF23FD
00FF28C4 > 81F9 04070000 CMP ECX, 704
00FF28CA > 0F8F D5000000 JG 00FF29A5
00FF28D0 > 0F84 AB000000 JE 00FF2981
00FF28D6 > 81F9 E4000000 CMP ECX, 0E4
00FF28DC > 74 6F JE SHORT 00FF294D
00FF28DE > 81F9 42030000 CMP ECX, 842
00FF28E4 > 74 5A JE SHORT 00FF294D

```

Figura 1: Inicio del bucle

```

00FF2DAE > 50 PUSH EAX
00FF2DAF > 51 PUSH ECX
00FF2DB0 > 52 PUSH EDX
00FF2DB1 > FF15 A04BFE0 CALL DWORD PTR DS:[0FE4BA0]
00FF2DB7 > C2 0C00 SETN 0
00FF2DBA > 6A 01 PUSH 1
00FF2DBC > FF15 3412FF0 CALL DWORD PTR DS:[<&MSUCRT.exit>]
00FF2DC2 > C705 84F8FE0 MOV DWORD PTR DS:[0FEF884], 2
00FF2DCC > 66:8125 0420 AND WORD PTR DS:[0FE2004], 0FFF
00FF2DD5 > 6A 00 PUSH 0
00FF2DD7 > 68 00 PUSH 0
00FF2DD9 > 68 BCF8FE0 PUSH OFFSET FerroCad.00FEF88C
00FF2DDE > E8 5D230000 CALL RNBORReleaseLicense(x,x,x)
00FF2DE3 > 68 10200000 PUSH 2010
00FF2DE8 > 00 00 CALL 00FF2060
00FF2DED > 50 PUSH EAX
00FF2DEE > E8 FD0C0000 CALL 00FF3AF0
00FF2DF3 > 00 00 MOV AL, BYTE PTR DS:[0FE2004]
00FF2DF8 > 83C4 08 ADD ESP, 8
00FF2DFB > A8 01 TEST AL, 01
00FF2DFD > 75 0C JNE SHORT 00FF2E0B
00FF2DFF > A1 84F8FE0 MOV EAX, DWORD PTR DS:[0FEF884]
00FF2E04 > 50 PUSH EAX
00FF2E05 > FF15 3810FF0 CALL DWORD PTR DS:[<&KERNEL32.ExitProcess>]
00FF2E08 > 3300 XOR EBX, EBX

```

```

[Arg3 => [ARG.3]
Arg2 => [ARG.2]
Arg1 => [ARG.1]
FerroCad.00FF2E80

[status = 1, case B of switch FerroCa
[MSUCRT.exit
Default case of switch FerroCad.00FF2D
Case 2 of switch FerroCad.00FF2D88
Arg3 = 0, cases 1, 3, 4, 5, 6, 7, 8,
Arg2 = 0
Arg1 = FerroCad.00FEF88C
FerroCad.RNBORReleaseLicense(x,x,x)

[Arg1
FerroCad.00FF3AF0

[ExitCode => [0FEF884] = 0
[KERNEL32.ExitProcess

```

Figura 2: Final del bucle

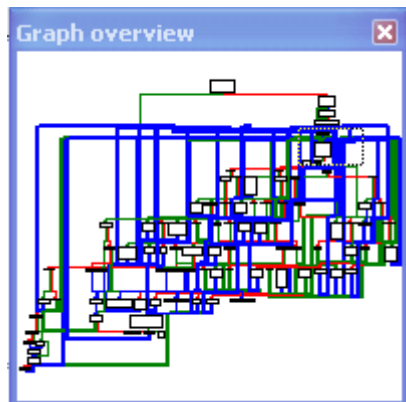


Figura 3: Vista del bucle en IDA

En la zona central pueden verse hasta 7 CALL EDX que en cada pasada del bucle llega con un valor distinto de EDX y por tanto nos lleva a una subrutina distinta, en las distintas pasadas del bucle una vez se pasará por alguno de esos CALL EDX y otras veces no. Esos CALL se encuentran en FF2962, FF29C9, FF2A78, FF2C59, FF2CA7, FF2CC1 y FF2CED

```

00FF2C59 > FFD2 CALL EDX
00FF2C5B > 8B00 7857FE0 MOV ECX, DWORD PTR DS:[0FE5778]
00FF2C61 > 8941 10 MOV DWORD PTR DS:[ECX+10], EAX
00FF2C64 > A1 7857FE0 MOV EAX, DWORD PTR DS:[0FE5778]
00FF2C69 > 83C0 10 ADD EAX, 10
00FF2C6C > A3 7857FE0 MOV DWORD PTR DS:[0FE5778], EAX
00FF2C71 > E9 F0BFFFFF JMP 00FF2866
00FF2C76 > 81F9 078E0000 CMP ECX, 08ED
00FF2C7C > 0F8F A8000000 JG 00FF2D2A
00FF2C82 > 0F84 82000000 JE 00FF2D00
00FF2C88 > 81F9 61AA0000 CMP ECX, 0AA6
00FF2C8E > 74 4E JE SHORT 00FF2CDE
00FF2C90 > 81F9 23B30000 CMP ECX, 0B323
00FF2C96 > 74 1E JE SHORT 00FF2CB6
00FF2C98 > 81F9 518A0000 CMP ECX, 0B8A5
00FF2C9E > 0F85 C2BFFFFF JNE 00FF2866
00FF2CA4 > 8B00 MOV EAX, DWORD PTR DS:[EAX]
00FF2CA6 > 50 PUSH EAX
00FF2CA7 > FFD2 CALL EDX
00FF2CA9 > 8B00 7857FE0 MOV ECX, DWORD PTR DS:[0FE5778]
00FF2CAF > 8901 MOV DWORD PTR DS:[ECX], EAX
00FF2CB1 > E9 ABFFFFF JMP 00FF2861
00FF2CB6 > 8B08 MOV ECX, DWORD PTR DS:[EAX]
00FF2CB8 > 51 PUSH ECX
00FF2CB9 > 8B48 04 MOV ECX, DWORD PTR DS:[EAX+4]
00FF2CBC > 8B40 08 MOV EAX, DWORD PTR DS:[EAX+8]
00FF2CBF > 51 PUSH ECX
00FF2CC0 > 50 PUSH EAX
00FF2CC1 > FFD2 CALL EDX

```

Figura 4: Algunos CALL EDX

También pude observar que había varios saltos condicionales que se tomaban o no dependiendo de la respuesta que recibiera de la emulación que pudiéramos hacer de la pastilla. Si la repuesta que hacíamos dar con nuestra emulación a la pastilla era correcta, el programa tomaba el salto adecuado y seguía la ejecución normal del programa. En caso contrario nos llevaba a la parte final del bucle donde nos sacaba del programa. Estos saltos son 4 y se encuentran en FF292F, FF2B44, FF2BB7 y FF2C10:

```

00FF2BB2 | A3 7857FE00 | MOV DWORD PTR DS:[0FES778],EAX
00FF2BB7 | ^ 0F85 A9FCFFF | JNE 00FF2866
00FF2BB8 | 8915 7C57FE00 | MOV DWORD PTR DS:[0FES77C],EDX
00FF2BC3 | ^ E9 9EFCFFF | JMP 00FF2866
00FF2BC8 | > 81F9 BB74000 | CMP ECX,74BB
00FF2BCE | ^ 74 5F | JE SHORT 00FF2C2F
00FF2BD0 | ^ 81F9 0981000 | CMP ECX,0981
00FF2BD6 | ^ 74 27 | JE SHORT 00FF2BFF
00FF2BD8 | ^ 81F9 0491000 | CMP ECX,0491
00FF2BDE | ^ 0F85 82FCFFF | JNE 00FF2866
00FF2BE4 | 56 | PUSH ESI
00FF2BE5 | 52 | PUSH EDI
00FF2BE6 | E8 55FBFFFF | CALL 00FF2740
00FF2BEB | 8B00 | MOV EDI,EDI
00FF2BED | A1 7857FE00 | MOV EAX,DWORD PTR DS:[0FES778]
00FF2BF2 | 83C4 08 | ADD ESP,8
00FF2BF5 | 0FAF10 | IMUL EDI,DWORD PTR DS:[EAX]
00FF2BF8 | ^ 8910 | MOV DWORD PTR DS:[EAX],EDI
00FF2BFH | ^ E9 62FCFFF | JMP 00FF2866
00FF2BF7 | > 8B10 | MOV EDI,DWORD PTR DS:[EAX]
00FF2C01 | ^ 33C9 | XOR ECX,ECX
00FF2C03 | 66:8BCE | MOV CX,SI
00FF2C06 | 83C0 04 | ADD EAX,4
00FF2C09 | 3BD1 | CMP EDI,ECX
00FF2C0B | ^ A3 7857FE00 | MOV DWORD PTR DS:[0FES778],EAX
00FF2C10 | ^ 0F85 50FCFFF | JNE 00FF2866
00FF2C16 | ^ 83E8 04 | SUB EAX,4

```

Figura 5: Algunos Saltos condicionales

Bueno pues así estuve dando palos de ciego y sin conseguir nada claro durante unos días. Puse BP en todas las funciones de la mochila, todas las llamadas CALL EDX y los 4 saltos condicionales que he mencionado. Al llegar a las funciones de Sentinel las emulaba como creía pero siempre llegaba a un punto en el que el programa me tiraba fuera o bien repetía las mismas instrucciones una y otra vez.

## \*\*-\*\* Empezando a comprender \*\*-\*\*

Como parecía que la clave estaba en los valores que iba tomando ECX empecé a buscar un patrón. Fue de casualidad que ví que en la primera pasada del bucle el primer valor que coge para ECX lo coge de FE5680:

Address	Hex dump	ASCII
00FE5680	03 CD 05 03 BC F8 FE 00	...
00FE5688	51 BA 01 00 40 48 FF 00	...
00FE5690	B7 22 05 03 00 00 00 00	...
00FE5698	11 06 8E 02 18 4C FE 00	...
00FE56A0	44 1A 00 00 F0 20 FF 00	...
00FE56A8	03 CD 05 03 04 20 FE 00	...
00FE56B0	03 CD 05 03 30 31 FE 00	...

Además en la línea FF288B pone en EDX lo que hay en FE5684 que vale FEF8BC

CPU - main thread, module FerroCad

```

00FF2861 | > A1 7857FE00 | MOV EAX,DWORD PTR DS:[0FES778]
00FF2866 | > 8B2D 7C57FE00 | MOV EBP,DWORD PTR DS:[0FES77C]
00FF286C | 66:8B4D 00 | MOV CX,DWORD PTR SS:[EBP]
00FF2870 | 83C5 08 | ADD EBP,8
00FF2873 | 66:890D 50FD | MOV WORD PTR DS:[0FEFD50],CX
00FF287A | 66:8B75 FA | MOV SI,DWORD PTR SS:[EBP-6]
00FF287E | 8B0D 50FDFE0 | MOV ECX,DWORD PTR DS:[0FEFD50]
00FF2884 | 66:8935 52FD | MOV WORD PTR DS:[0FEFD52],SI
00FF288B | 8B55 FC | MOV EDI,DWORD PTR SS:[EBP-4]
00FF2892 | 81F1 FFFF000 | AND ECX,0000FFFF

```

Registers (FPU)

```

EAX 00FEFD48 FerroCad.00FEFD48
ECX 00000000
EDX 00FEF8BC FerroCad.00FEF8BC
EBX 7FFDF000
ESP 0012FFBC
EBP 00FE5688 FerroCad.00FE5688
ESI 7FFD03D5
EDI 7C924388 ntdll.7C924388

```

Cuando traceando llegué a FF2CA7 CALL EDX, ví que en la pila estaba precisamente el valor FEF8BC:

0012FFB8	00FEF8BC	Arg1 = FerroCad.0FEF8BC
0012FFBC	7FFDF000	
0012FFC0	0012FFFF	
0012FFC4	7C817077	RETURN to kernel32.7C817
0012FFC8	7C924388	RETURN from ntdll.7C91FE

# CracksLatinoS! 2009

Así pude ver que cada vez que empezaba el bucle con un valor de ECX igual a CD03, esto hacía que cuando llegaba al siguiente CALL EDX el valor que había a continuación aparecía en la pila como argumento de esa subrutina. ¿Puede que estuviera ante una especie de máquina virtual? ¿Una especie de PCode casero?.

Seguí traceando y ví que cuando en ECX había el valor BA51 siempre se llegaba a FF2CA7 CALL EDX en el que EDX valía precisamente el valor almacenado 4 bits más adelante que ese BA51. En este caso:

Address	Hex dump		ASCII	
00F05680	03	CD 05 03	BC F8 FE 00	00 00 00 00
00F05688	51	BA 01 00	40 48 FF 00	00 00 00 00
00F05690	B7	22 05 03	00 00 00 00	00 00 00 00
00F05698	11	06 8E 02	18 4C FE 00	00 00 00 00
00F056A0	44	1A 00 00	F0 20 FF 00	00 00 00 00
00F056A8	03	CD 05 03	04 20 FE 00	00 00 00 00
00F056B0	03	CD 05 03	30 31 FE 00	00 00 00 00



Parece que había encontrado el patrón que buscaba que me hiciera comprender el funcionamiento. Así que me puse a rebuscar en la red y encontré esto:

courtesy of ketan:

```

ControlStruc Struc
val 1          dw      ?          ; func id
val 2          dw      ?          ; element id
val 3          dd      ?          ; element
ControlStruc   Ends

; element id
0x068          *b
0x28e          *w
0x234          *d
0x3d5          d
0x0d4          d (index) into cElement (* dword array)

; func definitions
0x1a44, n/a, *func ; cElement = func()
0xba51, n/a, *func ; cElement = func(cElement)
0x1109, n/a, *func ; cElement = func(cElement, 04, cElement)
0xb323, n/a, *func ; cElement = func(cElement, 08, cElement, 04, cElement)
0xaa61, n/a, *func ; cElement = func(cElement, 0C, cElement, 08, cElement, 04, cElement)
0x9b73, n/a, *func ; cElement = func(cElement, 10, cElement, 0C, cElement, 08, cElement, 04, cElement)
0x00e4, n/a, *func ; cElement = func(cElement, 14, cElement, 10, cElement, 0C, cElement, 08, cElement, 04, cElement)

0xcd03, element id, element ; cElement = element
0x1092, element id, element ; cElement += element
0x22b7, element id, element ; cElement -= element
0x9104, element id, element ; cElement *= element
0xf652, element id, element ; cElement /= element
0x3345, element id, element ; cElement &= element
0x654d, element id, element ; cElement |= element
0x74bb, element id, element ; cElement ^= element
0x698a, element id, element ; element = 0
0x1992, element id, element ; element = cElement
0x5211, element id, element ; element = cElement
0x0704, element id, element ; element++
0x3dc8, element id, element ; element--
0x5910, n/a, lpControlStruc ; clpControlStruc = lpControlStruc
0x7123, element id, lpControlStruc ; if (cElement == 0) then clpControlStruc = lpControlStruc
0x0611, element id, lpControlStruc ; if (cElement != 0) then clpControlStruc = lpControlStruc
0x2245, index, n/a ; clpControlStruc = cElement, cElement += index*4
0x0342, index, value ; cElement[index*4] = value
0xbcd7, n/a, n/a ; cElement = clpControlStruc
0x8109, id, n/a ; if (cElement == id) then cElement = clpControlStruc
0x65ab, id, n/a ; if (cElement != id) then cElement = clpControlStruc
0x096f, exitcode, n/a ; if (exitcode == 0) then SUCCESS, else display error message & exit

```

Que no es ni más ni menos que lo que estábamos buscando.

Resumiendo lo que tenemos básicamente es una máquina virtual que mediante el bucle que hemos descrito antes va leyendo las instrucciones que están codificadas en memoria entre las posiciones FE4BC8 y FE5780. Estas instrucciones se leen en bloques de 8 bits, en los que los 2 primeros indican en comando a ejecutar (ya vimos que por ejemplo CD03 significa PUSH), los 2 siguientes indican el tipo de argumento (byte, word, dword, etc) y los 4 últimos son el argumento de la función. De esta manera se va llamando a varias subrutinas, entre ellas



algunas funciones de la mochila Sentinel, de la que se obtendrán los datos necesarios para descryptar parte del código del ejecutable. Una vez se haya descryptado, este bucle nos hará saltar al OEP, desde el que podremos dumppear, y así tendremos el programa original que podrá ejecutarse sin mochila, ya que las llamadas a la mochila se hacen desde el bucle inicial que no volverá a llamarse.

### \*\*-\*\* Primera función a emular: RNBOsproInitialize\*\*-\*\*

Ahora ya estamos en condiciones de saber cómo debemos trabajar. Para hacerlo reiniciamos Olly y ponemos BP en todas las funciones de Sentinel, y en todos los CALL EDX que vimos anteriormente. De esta manera tendremos controladas todas las llamadas a rutinas fuera del bucle principal.

B INT3 breakpoints				
Address	Module	Status	Disassembly	Comment
00FF2364	FerroCad	Active	CALL EDX	
00FF23C9	FerroCad	Active	CALL EDX	
00FF2A76	FerroCad	Active	CALL EDX	
00FF2C59	FerroCad	Active	CALL EDX	
00FF2CA7	FerroCad	Active	CALL EDX	
00FF2C11	FerroCad	Active	CALL EDX	
00FF2CED	FerroCad	Active	CALL EDX	
00FF4800	FerroCad	Active	PUSH ESI	
00FF4840	FerroCad	Active	CMP WORD PTR DS:[0FED4E0],0	FerroCad.RNBOFormatPacket(x,x)(guessed Arg1,Arg2)
00FF4840	FerroCad	Active	PUSH EDI	FerroCad.RNBOInitialize(x)(guessed Arg1)
00FF4860	FerroCad	Active	PUSH ESI	FerroCad.RNBOSetContactServer(x,x)(guessed Arg1,Arg2)
00FF49F0	FerroCad	Active	SUB ESP,48	FerroCad.RNBOSetContactServer(x,x,x)(guessed Arg1,Arg2,Arg3)
00FF4CB0	FerroCad	Active	PUSH ESI	
00FF4D20	FerroCad	Active	PUSH ESI	FerroCad.RNBORead(x,x,x)(guessed Arg1,Arg2,Arg3)
00FF4D80	FerroCad	Active	PUSH ESI	FerroCad.RNBOWrite(x,x,x,x,x)(guessed Arg1,Arg2,Arg3,Arg4,Arg5)
00FF4E60	FerroCad	Active	PUSH ESI	FerroCad.RNBOWery(x,x,x,x,x)(guessed Arg1,Arg2,Arg3,Arg4,Arg5,Arg6)
00FF4F20	FerroCad	Active	PUSH ESI	FerroCad.RNBODecrement(x,x,x)(guessed Arg1,Arg2,Arg3)
00FF4FB0	FerroCad	Active	SUB ESP,444	FerroCad.RNBORGetVersion(x,x,x,x,x)(guessed Arg1,Arg2,Arg3,Arg4,Arg5)
00FF5100	FerroCad	Active	PUSH ESI	FerroCad.RNBORGetFullStatus(x)(guessed Arg1)
00FF5140	FerroCad	Active	PUSH ESI	FerroCad.RNBOReleaseLicense(x,x,x)(guessed Arg1,Arg2,Arg3)
00FF51D0	FerroCad	Active	MOV EAX,DWORD PTR SS:[ESP+10]	FerroCad.RNBORUnServer(x,x,x)(guessed Arg1,Arg2,Arg3,Arg4)

En principio con eso bastaría. Ahora pulsamos F9 y enseguida llegamos a FF2CA7:

CPU - main thread, module FerroCad				
Address	Disassembly	Comment	Registers (FPU)	
00FF2CA4	MOV EAX,DWORD PTR DS:[EAX]		EAX 00FEF8BC FerroCad.00FEF8BC	
00FF2CA6	PUSH EAX		ECX 0000BA51	
00FF2CA7	CALL EDX		EDX 00FF4840 FerroCad.RNBOInitialize(x)	
00FF2CA9	MOV ECX,DWORD PTR DS:[0FE5778]		EBX 7FFD9000	
00FF2CAF	MOV DWORD PTR DS:[ECX],EAX		ESP 0012FFB8	
00FF2CB1	JMP 00FF2861		EBP 00FE5690 FerroCad.00FE5690	
00FF2CB6	JMP 00FF2861		ESI 7FFD0001	
00FF2CB8	PUSH ECX		EDI 7C924388 ntdll.7C924388	
00FF2CB9	MOV ECX,DWORD PTR DS:[EAX+4]			
00FF2CBC	MOV EAX,DWORD PTR DS:[EAX+8]			

Vemos que EDX vale en esta ocasión FF4840, que es donde está situada la función RNBOInitialize(x). Si pulsamos F9 de nuevo apareceremos al principio de dicha función:

Address	Disassembly	Comment
00FF4840	RNBOInitialize(x)	
00FF4848	CMP WORD PTR DS:[0FED4E0],0	
00FF4849	PUSH ESI	
00FF4849	JNE SHORT 00FF4850	
00FF484B	CALL 00FF4750	
00FF484B	MOV ESI,DWORD PTR SS:[ARG.1]	
00FF484D	TEST ESI,ESI	
00FF484D	JNE SHORT 00FF4860	
00FF4850	MOV AX,10	
00FF4850	POP ESI	
00FF4850	RET 4	
00FF4850	PUSH ESI	
00FF4850	CALL 00FF7970	

Según el Sentinel Developer Guide:

## Format

```
unsigned short int RNBOsproInitialize(
RB_SPRO_APIPACKET packet
);
```

## Parameters

packet A pointer to the RB\_SPRO\_APIPACKET record.

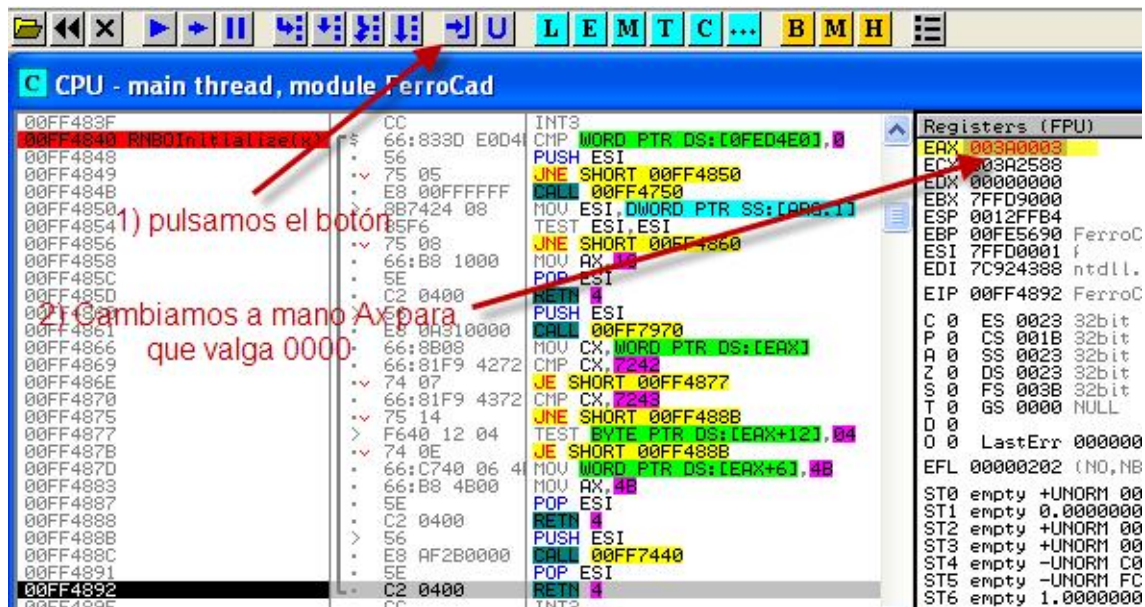
## Return Values

If successful, the function returns SP\_SUCCESS (0)

Vemos que si la función tiene éxito, debe devolver 0. Eso equivale a poner AX=0. No vamos a hacer una emulación propiamente dicha modificando el código del programa, porque nuestro

objetivo es quitar esta protección al programa original, así que si tenemos éxito, el programa no volverá a pasar por este punto.

Así pulsamos Ctrl+F9 para ejecutar hasta el RET, y una vez allí situados modificamos manualmente AX para que valga 0:



Después de cambiar AX:

Registers (FP)
EAX 003A0000
ECX 003A2588
EDX 00000000
EBX 7FFD9000
ESP 0012FFB4
EBP 00FE5690
ESI 7FFD0001
EDI 7C924388
EIP 00FF4892

Ya tenemos emulada la primera función.

## .-\*\* Segunda función: RNBOsproSetContactServer \*\*.-

Ahora tendremos que pulsar varias veces F9 porque irá parando en algunos CALL EDX que de momento no nos interesan. Finalmente llegamos a:

00FF48A0 RNBOSetContactServer(x,x)	53	PUSH EBX
00FF48A1	56	PUSH ESI
00FF48A2	8B5C24 0C	MOV EBX, DWORD PTR SS:[ARG.1]
00FF48A6	57	PUSH EDI
00FF48A7	66:33FF	XOR DI, DI
00FF48AA	85DB	TEST EBX, EBX
00FF48AC	75 0A	JNE SHORT 00FF48B8
00FF48AE	66:B8 1000	MOV AX, 10
00FF48B2	5F	POP EDI
00FF48B3	5E	POP ESI
00FF48B4	5B	POP EBX
00FF48B5	C2 0800	RET 8
00FF48B8	53	PUSH EBX
00FF48B9	E8 B2300000	CALL 00FF7970

Según el Sentinel Developer Guide:

### Format

```
unsigned short int RNBOsproSetContactServer(
RBP_SPRO_APIPACKET packet,
char *serverName
);
```

### Parameters

*packet* A pointer to the RB\_SPRO\_APIPACKET record.  
*serverName* The name to which the contact server is set.

## Return Values

If successful, the function returns **SP\_SUCCESS(0)**.

Vemos que si hacemos Ctrl+F9 para ir al RET de la función y ponemos AX=0 la tendremos emulada:

00FF4948	> 8B4424 14	MOV EAX, DWORD PTR SS:[ARG.2]	Registers (FF) EAX 00000000 ECX 003A2588 EDX 00000000 EBX 7FFD9000 ESP 0012FF68 EBP 00FE5708 ESI 7FFD0001 EDI 7C924388
00FF494C	• 50	PUSH EAX	
00FF494D	• 53	PUSH EBX	
00FF494E	• E8 0D2C0000	CALL 00FF7630	
00FF4953	> 5F	POP EDI	
00FF4954	• 5E	POP ESI	
00FF4955	• 5B	POP EBX	
00FF4956	• C2 0800	RET 8	
00FF4959	• CC	INT3	
00FF495D	• 77	TNT3	

Después de editar EAX:

Registers (FF)	
EAX	00000000
ECX	003A2588
EDX	00000000
EBX	7FFD9000
ESP	0012FF68
EBP	00FE5708
ESI	7FFD0001
EDI	7C924388

Ya tenemos la 2ª función emulada, esto va bien.

## .-\*\* 3ª función: RNBOsproFindFirstUnit \*\*.-

Pulsamos F9 para continuar y llegamos a:

00FF29B9	• 83E9 77	SUB ECX, 77	Registers (FPU) EAX 00FEF8BC FerroCad.00FEF8BC ECX 00001429 EDX 00FF49F0 FerroCad.RNBOFindFirstUnit(x,x) EBX 7FFD9000 ESP 0012FFB4 EBP 00FE5730 FerroCad.00FE5730 ESI 7FFD0000 EDI 7C924388 ntdll.7C924388 EIP 00FF29C9 FerroCad.00FF29C9
00FF29BC	• ^ 0F85 A4FEFF	JNE 00FF2866	
00FF29C2	• 8B08	MOV ECX, DWORD PTR DS:[EAX]	
00FF29C4	• 8B40 04	MOV EAX, DWORD PTR DS:[EAX+4]	
00FF29C7	• 51	PUSH ECX	
00FF29C8	• 50	PUSH EAX	
00FF29C9	• FFD2	CALL EDX	
00FF29CB	• 8B0D 7857FE00	MOV ECX, DWORD PTR DS:[0FE5778]	
00FF29D1	• 8941 04	MOV DWORD PTR DS:[ECX+4], EAX	
00FF29D4	• A1 7857FE00	MOV EAX, DWORD PTR DS:[0FE5778]	
00FF29D9	• 83C0 04	ADD EAX, 4	

Vemos que en EDX está la dirección de la función RNBOFindFirstUnit. Pulsamos F9 una vez más y llegamos a:

00FF49EF	• CC	INT3	Registers (FPU) EAX 00FEF8BC FerroCad.00FEF8BC ECX 00001429 EDX 00FF49F0 FerroCad.RNBOFindFirstUnit(x,x) EBX 7FFD9000 ESP 0012FFB0 EBP 00FE5730 FerroCad.00FE5730 ESI 7FFD0000 EDI 7C924388 ntdll.7C924388 EIP 00FF49F0 FerroCad.RNBOFindFirstUnit(x,x)  C 0 ES 0023 32bit 0(FFFFFFFF) P 1 CS 001B 32bit 0(FFFFFFFF) A 0 SS 0023 32bit 0(FFFFFFFF) Z 1 DS 0023 32bit 0(FFFFFFFF) S 0 FS 003B 32bit 7FFDF000(FFF) T 0 GS 0000 NULL D 0 O 0 LastErr 000000CB ERROR_ENVVAR_NOT_FOUND EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE) ST0 endtv +UNORM 0002 00000031 00000000
00FF49F0	• 83EC 48	SUB ESP, 48	
00FF49F3	• 66:C74424 02	MOV WORD PTR SS:[LOCAL.17+2], 0	
00FF49FA	• 53	PUSH EBX	
00FF49FB	• 56	PUSH ESI	
00FF49FC	• 8B5C24 54	MOV EBX, DWORD PTR SS:[ARG.1]	
00FF4A00	• 57	PUSH EDI	
00FF4A01	• 55	PUSH EBP	
00FF4A02	• 850B	TEST EBX, EBX	
00FF4A04	• 0F84 8B020000	JE 00FF4C95	
00FF4A0A	• 66:8B6C24 60	MOV BP, WORD PTR SS:[ARG.2]	
00FF4A0F	• 66:85ED	TEST BP, BP	
00FF4A12	• 0F84 70020000	JE 00FF4C95	
00FF4A18	• 66:31FD FFFF	CMPL BP, 0FFFF	
00FF4A1D	• 75 0E	JNE SHORT 00FF4A2D	
00FF4A1F	• 66:58 0300	MOV AX, 0	
00FF4A23	• 5D	POP EBP	
00FF4A24	• 5F	POP EDI	
00FF4A25	• 5E	POP ESI	
00FF4A26	• 5B	POP EBX	
00FF4A27	• 83C4 48	ADD ESP, 48	
00FF4A2A	• C2 0800	RET 8	

Sabemos que:

## Format

```
unsigned short int RNBOsproFindFirstUnit(
RB_SPRO_APIPACKET packet,
unsigned short int developerID
);
```

## Parameters



*packet* A pointer to the RB\_SPRO\_APIPACKET record.

*developerID* This is assigned to you by Rainbow Technologies or your distributor. It identifies the SentinelSuperPro device to search for.

## Return Values

If successful, the function returns **SP\_SUCCESS (0)**.

Por lo que de nuevo lo que tenemos que hacer es hacer que AX valga 0 cuando estemos en el RET, pero esta vez sí vamos a modificar el ejecutable ya que de lo contrario comienza con llamadas anidadas a funciones de Sentinel y es un poco lioso salir de ahí sin perderse. Las modificaciones a hacer son las siguientes (recordar que EIP sigue estando al principio de la función):

```

00FF49F0 83EC 48      SUB ESP, 48
00FF49F3 66:C74424 02 MOV WORD PTR SS:[LOCAL.17+2], 0
00FF49FA 53          PUSH EBX
00FF49FB 56          PUSH ESI
00FF49FC 8B5C24 54    MOV EBX, DWORD PTR SS:[ARG.1]
00FF4A00 57          PUSH EDI
00FF4A01 55          PUSH EBP
00FF4A02 85DB        TEST EBX, EBX
00FF4A04 0F84 8B020001 JE 00FF4C95
00FF4A09 66:8B6C24 60 MOV BP, WORD PTR SS:[ARG.2]
00FF4A0F 66:85ED      TEST BP, BP
00FF4A12 0F84 7D020001 JE 00FF4C95
00FF4A18 66:81FD FFFF CMP BP, 0FFFF
00FF4A1D 90          NOP
00FF4A1E 90          NOP
00FF4A1F 66:88 0000   MOV AX, 0
00FF4A23 5D          POP EBP
00FF4A24 5F          POP EDI
00FF4A25 5E          POP ESI
00FF4A26 5B          POP EBX
00FF4A27 83C4 48      ADD ESP, 48
00FF4A2A C2 0800      RETN 8
  
```

Ahora si pulsamos Ctrl+F9 vemos que salimos de la función con AX=0:

```

Registers (FPU)
EAX 00FE0000 Ferrocad.00FE0000
ECX 00001429
EDX 00FF49F0 Ferrocad.RNBOFindFirstUnit(x,x)
EBX 7FFD9000
ESP 0012FFB0
EBP 00FE5730 Ferrocad.00FE5730
ESI 7FFD0000 {
EDI 7C924388 ntdll.7C924388
EIP 00FF4A2A Ferrocad.00FF4A2A
  
```

Vamos a por la siguiente.

**.-\*\* Primera trampa: Llamadas a RNBOsproRead \*\*.-**

Si continuamos con F9 paramos enseguida en FF2A78 CALL EDX en el que EDX vale FF3A50. Si entramos en esa subrutina con F7 nos encontramos con esto:

00FF3A4F	90	NOP	
00FF3A50	83EC 10	SUB ESP, 10	
00FF3A53	33C0	XOR EAX, EAX	
00FF3A55	56	PUSH ESI	
00FF3A56	894424 04	MOV DWORD PTR SS:[LOCAL.3], EAX	
00FF3A5A	894424 08	MOV DWORD PTR SS:[LOCAL.2], EAX	
00FF3A5E	894424 0C	MOV DWORD PTR SS:[LOCAL.1], EAX	
00FF3A62	894424 10	MOV DWORD PTR SS:[LOCAL.0], EAX	
00FF3A66	8D4424 04	LEA EAX, [LOCAL.3]	
00FF3A6A	33F6	XOR ESI, ESI	
00FF3A6C	50	PUSH EAX	
00FF3A6D	6A 30	PUSH 30	[Arg3 => OFFSET LOCAL.3 Arg2 = 30 Arg1 = FerroCad.0FEF8BC FerroCad.RNBORRead(x,x,x)]
00FF3A6F	68 BCF8FE00	PUSH OFFSET FerroCad.0FEF8BC	
00FF3A74	E8 A7120000	CALL RNBORead(x,x,x)	
00FF3A79	66:85C0	TEST AX, AX	
00FF3A7C	75 68	JNE SHORT 00FF3AE9	
00FF3A7E	8D4C24 08	LEA ECX, [LOCAL.2]	
00FF3A82	51	PUSH ECX	
00FF3A83	6A 34	PUSH 34	[Arg3 => OFFSET LOCAL.2 Arg2 = 34 Arg1 = FerroCad.0FEF8BC FerroCad.RNBORRead(x,x,x)]
00FF3A85	68 BCF8FE00	PUSH OFFSET FerroCad.0FEF8BC	
00FF3A8A	E8 91120000	CALL RNBORead(x,x,x)	
00FF3A8F	66:85C0	TEST AX, AX	
00FF3A92	75 55	JNE SHORT 00FF3AE9	
00FF3A94	8D5424 0C	LEA EDX, [LOCAL.1]	
00FF3A98	52	PUSH EDX	
00FF3A99	6A 38	PUSH 38	[Arg3 => OFFSET LOCAL.1 Arg2 = 38 Arg1 = FerroCad.0FEF8BC FerroCad.RNBORRead(x,x,x)]
00FF3A9B	68 BCF8FE00	PUSH OFFSET FerroCad.0FEF8BC	
00FF3AA0	E8 7B120000	CALL RNBORead(x,x,x)	
00FF3AA5	66:85C0	TEST AX, AX	
00FF3AA8	75 3F	JNE SHORT 00FF3AE9	
00FF3AA9	8D4424 10	LEA EAX, [LOCAL.0]	
00FF3AAE	50	PUSH EAX	
00FF3AAF	6A 3C	PUSH 3C	[Arg3 => OFFSET LOCAL.0 Arg2 = 3C Arg1 = FerroCad.0FEF8BC FerroCad.RNBORRead(x,x,x)]
00FF3AB1	68 BCF8FE00	PUSH OFFSET FerroCad.0FEF8BC	
00FF3AB6	E8 65120000	CALL RNBORead(x,x,x)	
00FF3ABB	66:85C0	TEST AX, AX	
00FF3ABE	75 29	JNE SHORT 00FF3AE9	
00FF3AC0	66:817C24 04	CMP WORD PTR SS:[LOCAL.3], 0DE9B	
00FF3AC7	75 20	JNE SHORT 00FF3AE9	
00FF3AC9	66:817C24 08	CMP WORD PTR SS:[LOCAL.2], 0A17C	
00FF3AD0	75 17	JNE SHORT 00FF3AE9	
00FF3AD2	66:817C24 0C	CMP WORD PTR SS:[LOCAL.1], 9A8F	
00FF3AD9	75 0E	JNE SHORT 00FF3AE9	
00FF3ADB	66:817C24 10	CMP WORD PTR SS:[LOCAL.0], 74BE	
00FF3AE2	75 05	JNE SHORT 00FF3AE9	
00FF3AE4	BE 01000000	MOV ESI, 1	
00FF3AE9	8BC6	MOV EAX, ESI	
00FF3AEB	5E	POP ESI	
00FF3AEC	83C4 10	ADD ESP, 10	
00FF3AEF	C3	RET	

Primero vemos qué significa cada argumento de la función RNBOsproRead:

## Format

```
unsigned short int RNBOsproRead(
RB_SPRO_APIPACKET packet,
unsigned short int address,
unsigned short int *data
);
```

## Parameters

*packet* A pointer to the RB\_SPRO\_APIPACKET record.

*address* The SentinelSuperPro key memory cell address of the word to read.

*data* A pointer to the location that will contain the data read from the SentinelSuperPro key.

## Return Values

If successful, the function returns **SP\_SUCCESS(0)**.

Vemos que el primer argumento es un puntero a APIPACKET, el segundo es la dirección de la celda de la que se quiere leer el Word, y el tercero es un puntero a la dirección donde se escribirá el dato leído. Si la función tiene éxito, escribirá el Word correspondiente en la dirección indicada por el puntero y además AX valdrá 0.

Lo primero que pensé cuando vi esto es que los Word de las celdas 30, 34, 38 y 3C debían valer DE9B, A17C, 9A8F y 74BE para cumplir con las comparaciones que se realizan a partir de FF3AC0. Sin embargo cuando hacía esto el programa me tiraba fuera. Yo creía que estaba haciendo mal la emulación, pero por más que le daba vueltas no veía qué hacía mal, hasta que pensé: Si estoy haciendo bien la emulación, puede que los datos que creo que debe leer no sean esos.

Así que lo que hice fue emular que en esas posiciones de celda hay FFFF en cada una. Me puse al principio de la función Read y apunté, en cada una de las veces que paró, en qué dirección iba a escribir el Word leído de la celda de la mochila. La primera vez fue en 12FFA8, después 12FFAC, 12FFB0 y 12FFB4. En cada una de esas direcciones escribí a mano FFFF, y esta vez antes que nada parcheé la función para que siempre saliera con AX=0:

Address	Hex dump
0012FFA8	FF FF 00 00 FF FF 00 00 FF FF 00 00 FF FF 00 00
0012FFB8	7A 2A FF 00 00 40 FD 7F F0 FF 12 00 77 70 81 7C
0012FFC8	88 43 92 7C 00 40 FD 7F 00 90 FD 7F ED B6 54 80
0012FFD8	C8 FF 12 00 A0 3C 12 88 FF FF FF FF D8 9A 83 7C
0012FFE8	80 70 81 7C 00 00 00 00 00 00 00 00 00 00 00 00
0012FFF8	F0 27 FF 00 00 00 00 00

00FF4D20	RNBORRead(x,x,x)	56	PUSH ESI
00FF4D21		57	PUSH EDI
00FF4D22		8B7C24 0C	MOV EDI,DWORD PTR SS:[ARG.1]
00FF4D26		85FF	TEST EDI,EDI
00FF4D28		90	NO
00FF4D29		90	NO
00FF4D2A		66:B8 0000	MOV AX,0
00FF4D2E		5F	POP EDI
00FF4D2F		5E	POP ESI
00FF4D30		C2 0C00	RET 0C
00FF4D33		57	PUSH EDI
00FF4D34		E8 372C0000	CALL 00FF7970

Con esto estaría solucionada también esta parte.

Al pasar los 4 Read que hemos visto, si pulsamos F9 de nuevo vamos a parar a:

00FF2CBC		8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]	Registers (FPU)	EAX 00FEF8BC FerroCad.00FEF8BC
00FF2CBF		51	PUSH ECX	ECX 00000000	ECX 00000000
00FF2CC0		50	PUSH EAX	EDX 00FF4D20 FerroCad.RNBORRead(x,x,x)	EDX 00FF4D20
00FE2CC1		FFD2	CALL EDX	EBX 7FFD9000	EBX 7FFD9000
00FF2CC3		8B00 7857FE0	MOV ECX,DWORD PTR DS:[0FE5778]	ESP 0012FFAC	ESP 0012FFAC
00FF2CC9		8941 08	MOV DWORD PTR DS:[ECX+8],EAX	EBP 00FE52E8 FerroCad.00FE52E8	EBP 00FE52E8
00FF2CC9		A1 7857FE00	MOV EAX,DWORD PTR DS:[0FE5778]	ESI 7FFD0000	ESI 7FFD0000
00FF2CD1		83C0 08	ADD EAX,8	EDI 7C924388 ntdll.7C924388	EDI 7C924388
00FF2CD4		A3 7857FE00	MOV DWORD PTR DS:[0FE5778],EAX		
00FF2CD9		E9 88FBFFFF	JMP 00FF2866		

En esta nueva llamada a RNBOsproRead se lee la celda de posición 0 y se escribe su valor en FEFC2:

0012FFB0	00FEF8BC	Arg1 = FerroCad.0FEF8BC
0012FFB4	00000000	Arg2 = 0
0012FFB8	00FEFC22	Arg3 = FerroCad.0FEFC22

Según la documentación de Sentinel en la celda 0 está almacenado el número de serie de la llave, y éste es secuencial y varía desde 0000 a FFFF, por lo que podremos poner ahí cualquier número. Nosotros pondremos, una vez parados en el RET de la función, 1234 en la posición FEFC2, y como la función la parcheamos anteriormente sólo comprobaremos que efectivamente salimos de la función con AX valiendo 0:

Address	Hex dump
00FEFC22	34 12 01 00 00 00 00 00
00FEFCD2	00 00 00 00 00 00 00 00
00FEFCE2	00 00 00 00 00 00 00 00

Ahora pulsamos F9 para continuar, y esta vez no nos saca fuera. Buena señal.

**.-\*\* Empieza lo bueno: RNBOsproQuery \*\*-**

La siguiente parada es en FF2964:

00FF2964		FFD2	CALL EDX	Registers (FPU)	EAX 00FEF8BC FerroCad.00FEF8BC
00FF2966		8B00 7857FE0	MOV ECX,DWORD PTR DS:[0FE5778]	ECX 00000000	ECX 00000000
00FF296C		8941 14	MOV DWORD PTR DS:[ECX+14],EAX	EDX 00FF4E60 FerroCad.RNBORQuery(x,x,x,x,x)	EDX 00FF4E60
00FF296F		A1 7857FE00	MOV EAX,DWORD PTR DS:[0FE5778]	EBX 7FFD9000	EBX 7FFD9000
00FF2974		83C0 14	ADD EAX,14	ESP 0012FFA4	ESP 0012FFA4
00FF2977		A3 7857FE00	MOV DWORD PTR DS:[0FE5778],EAX	EBP 00FE5350 FerroCad.00FE5350	EBP 00FE5350
00FF297C		E9 E5FEFFFF	JMP 00FF2866	ESI 7FFD0000	ESI 7FFD0000
00FF2981		56	PUSH ESI	EDI 7C924388 ntdll.7C924388	EDI 7C924388
00FF2982		52	PUSH EDX		
00FF2983		E8 B8FDFFFF	CALL 00FF2740		

Vemos que se va a llamar a RNBOsproQuery:

## Format

```
unsigned short int RNBOsproQuery(
RB_SPRO_APIPACKET packet,
unsigned short int address,
VOID *queryData,
VOID *response,
unsigned long *response32,
unsigned short int length
);
```

## Parameters

*packet* A pointer to the RB\_SPRO\_APIPACKET record.

*address* The address of the word to query.

*queryData* The pointer to the first byte of the query bytes.

*response* The pointer to the first byte of the response bytes.

*response32* The pointer to the location that will contain a copy of the last four bytes of the query response.

*length* This is the number of query bytes to send to the active algorithm and also the length of the response buffer.

## Return Values

If successful, the function returns **SP\_SUCCESS (0)**.

Esta función llama a un algoritmo activo en la celda especificada (segundo parámetro). A este algoritmo le pasa como argumento los datos situados en la dirección a la que apunta queryData (tercer argumento) y guarda el resultado en la dirección a la que apunta response (cuarto argumento). Tanto queryData como response tienen una longitud en bytes indicada por el argumento length. Si la función tiene éxito devuelve AX=0.

Ahora veremos nuestro caso particular:

0012FFA4	00FEF8BC	#...	Arg1 = FerroCad.0FEF8BC
0012FFA8	00000000	...	Arg2 = 0
0012FFAC	00FEF878	x...	Arg3 = FerroCad.0FEF878
0012FFB0	00FEF87C	!...	Arg4 = FerroCad.0FEF87C
0012FFB4	00000000	...	Arg5 = 0
0012FFB8	00000004	♦...	Arg6 = 4

Vemos que el algoritmo se aplica sobre la celda 0. Este detalle es importante, porque para este algoritmo siempre se cumple que response = queryData. En nuestro caso length=4:

Address	Hex dump	ASCII
00FEF878	7F 40 69 01 00 00 00 00	@e!@...
00FEF880	00 00 00 00 00 00 00 00	.....
00FEF888	00 00 00 00 00 00 00 00	.....
00FEF890	0A 07 01 00 02 00 1A 00	r.0.0.+.
00FEF898	10 00 02 00 18 00 84 03	0.0.†.ä
00FEF8A0	00 00 00 00 00 00 00 00	.....

Podemos ver que queryData = 0169407F por lo que en FEF87C deberemos escribir a mano el mismo valor:

Address	Hex dump	ASCII
00FEF878	7F 40 69 01 7F 40 69 01	@e!@@e!@
00FEF880	00 00 00 00 00 00 00 00	.....
00FEF888	00 00 00 00 00 00 00 00	.....
00FEF890	0A 07 01 00 02 00 1A 00	r.0.0.+.
00FEF898	10 00 02 00 18 00 84 03	0.0.†.ä
00FEF8A0	00 00 00 00 00 00 00 00	.....
00FEF8A8	00 00 00 00 00 00 00 00	.....
00FEF8B0	00 00 00 00 00 00 00 00	.....

Hay que destacar que en cada ejecución del programa el valor de queryData se genera aleatoriamente y cambia. Además vamos a parchear la función para que siempre salga con AX=0:



00FF4E60	RNBOQuery(x,x,x,x,x,x)	56	PUSH ESI
00FF4E61		57	PUSH EDI
00FF4E62		8B7424 0C	MOV ESI,DWORD PTR SS:[ARG.1]
00FF4E66		85F6	TEST ESI,ESI
00FF4E68		90	NOF
00FF4E69		90	NOF
00FF4E6A		66:B8 0000	MOV AX,0
00FF4E6E		5F	POP EDI
00FF4E6F		5E	POP ESI
00FF4E70		C2 1800	RETN 18
00FF4E73		56	PUSH ESI
00FF4E74		E8 F72A0000	CALL 00FF7970
00FF4E79		8BFA	MOV EAX,ESI

Con esto ya tendríamos resuelta la primera llamada a RNBOsproQuery.

## .-\*\* Llegan las complicaciones: 2ª llamada a RNBOsproQuery \*\*-

En este punto llegaremos a la segunda llamada a RNBOsproQuery. Esta va a ser bastante más complicada que la anterior. Veamos:

Si seguimos con F9 paramos primero en un CALL EDX en el que EDX vale FF2600. Si entramos en esa función (que yo he llamado funcQueryData) vemos que lo que hace es coger un número de una tabla que empieza en FE2010. Posteriormente he visto que el número que escoge de la tabla es aleatorio y cambia en cada ejecución del programa.

00FF2600	funcQueryData	A1 88F8FE00	MOV EAX,DWORD PTR DS:[0FEF888]
00FF2605		8B0485 1020F	MOV EAX,DWORD PTR DS:[EAX*4+0FE2010]
00FF260C		C3	RETN
00FF260D		90	NOF

[00FE269C]=E5016FC0  
EAX=000001A3 (decimal 419.)

Traceando con F8 vemos que guarda el valor en FEFD40:

00FF2A78		FFD2	CALL EDX
00FF2A79		8B0D 7857FE00	MOV ECX,DWORD PTR DS:[0FE5778]
00FF2A80		8901	MOV DWORD PTR DS:[ECX],EAX
00FF2A82		E9 0AFDFFFF	JMP 00FF2861
00FF2A87		56	PUSH ESI

EAX=E5016FC0  
[00FEFD40]=000001D1 (decimal 465.)

Vamos a ponerle un BPM en FEFD40 para ver qué hace con ese valor. Pulsamos F9 para continuar y

00FF2C36		8B0D 7857FE00	MOV ECX,DWORD PTR DS:[0FE5778]
00FF2C3C		83C4 08	ADD ESP,8
00FF2C3F		3101	XOR DWORD PTR DS:[ECX],EAX
00FF2C41		E9 1BFCFFFF	JMP 00FF2861
00FF2C46		8B08	MOV ECX,DWORD PTR DS:[EAX]

EAX=FD0699D6  
[00FEFD40]=E5016FC0

Oly para ahí por el BPM, debido a que va a Xorear el valor con FD0699D6 (este valor es constante y no varía en otras ejecuciones del programa), y guarda el resultado en FEFD40:

Address	Hex dump
00FEFD40	16 F6 07 18 00 00 00 00 D6 99 06 FD 00 00 00 00
00FEFD50	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00FEFD60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Continuamos la ejecución con F9 y llegamos a:

00FF23FD	> 48B00	MOV EAX,DWORD PTR DS:[EAX]	Pone [FEFD40]=1807F616 en EAX
00FF23FF	50	PUSH EAX	Arg3
00FF2A00	56	PUSH ESI	Arg2
00FF2A01	52	PUSH EDI	Arg1
00FF2A02	E8 09FCFFFF	CALL 00FF26E0	FerroCad.00FF26E0
00FF2A07	A1 7857FE00	MOV EAX,DWORD PTR DS:[0FE5778]	
00FF2A0C	83C4 0C	ADD ESP,0C	
00FF2A0F	83C0 04	ADD EAX,4	
00FF2A12	A3 7857FE00	MOV DWORD PTR DS:[0FE5778],EAX	
00FF2A17	E9 4AFDFFFF	JMP 00FF2866	

[00FEFD40]=1807F616  
EAX=FerroCad.00FEFD40  
Jump from <ModuleEntryPoint>+0CE

# CracksLatinoS! 2009

Donde guarda el valor en EAX, en el CALL FF26E0 lo que hace es intercambiar los valores de EAX y EDX y guardar el valor en FEF878. Así que ahora pondremos un BPM en FEF878 y además tendremos EDX=1807F616, pero inmediatamente se sobrescribe EDX con otro valor,

```
00FF2719 | > 8B4424 04 | MOV EAX,DWORD PTR SS:[ARG.1]
00FF271D | . 8B5424 0C | MOV EDX,DWORD PTR SS:[ARG.3]
00FF2721 | . 8910 | MOV DWORD PTR DS:[EAX],EDX
00FF2723 | . C3 | RETN
```

Así que podemos continuar con F9 hasta que llegamos a un punto donde se sobrescribe el valor que teníamos almacenado en FEFD40, por lo que podemos quitar ese BPM:

```
00FF2D75 | . 890D 7857FE0 | MOV DWORD PTR DS:[0FE5778],ECX
00FF2D7B | . 8901 | MOV DWORD PTR DS:[ECX],EAX
00FF2D7D | . ^ E9 DFFAFFFF | JMP 00FF2861
00FF2D82 | > 33C0 | XOR EAX,EAX
EAX=FerroCad.00FEF8BC
[00FEFD40]=1807F616
```

FerroCad.<ModuleEntryPoint>+58B

Memory breakpoints					
Address	Size	Module	Type	Status	Comment
00FEF878	00000004	FerroCad	RW	Active	
00FEF87C	00000004	FerroCad	RW	Disabled	
00FEFD40	00000004	FerroCad	RW	Disabled	
00FEFD48	00000004	FerroCad	RW	Disabled	

Damos a F9 otra vez y llegamos a la segunda llamada a RNBOsproQuery en la que se aplica el algoritmo sobre la celda 8 por lo que la facilidad que tuvimos antes de saber fácilmente cuál era el valor de response no la vamos a tener ahora. Además podemos ver que el argumento queryData es FEF878 que es precisamente donde tenemos guardado el resultado de aplicar Xor FD0699D6 al número obtenido de la tabla que está en FE2010:

CPU - main thread, module FerroCad

00FF2958 | . 8B48 0C | MOV ECX,DWORD PTR DS:[EAX+0C]

00FF295B | . 51 | PUSH ECX

00FF295C | . 8B48 10 | MOV ECX,DWORD PTR DS:[EAX+10]

00FF295F | . 8B48 14 | MOV EAX,DWORD PTR DS:[EAX+14]

00FF2962 | . 51 | PUSH ECX

00FF2963 | . 50 | PUSH EAX

00FF2964 | . FF02 | CALL EDI

00FF2966 | . 8B0D 7857FE0 | MOV ECX,DWORD PTR DS:[0FE5778]

00FF296C | . 8941 14 | MOV DWORD PTR DS:[ECX+14],EAX

00FF296F | . A1 7857FE0 | MOV EAX,DWORD PTR DS:[0FE5778]

Registers (FPU)

EAX 00FEF8BC FerroCad.00FEF8BC

ECX 00000008

EDX 00FF4E60 FerroCad.RNBOQuery(X,X,X,X,X,X)

EBX 7FFD4000

ESP 0012FFA4

EBP 00FE5400 FerroCad.00FE5400

ESI 7FFD0000

EDI 7C924388 ntdll.7C924388

0012FFA4 | 00FEF8BC | Arg1 = FerroCad.00FEF8BC

0012FFA8 | 00000008 | Arg2 = 8

0012FFAC | 00FEF878 | Arg3 = FerroCad.00FEF878

0012FFB0 | 00FEF87C | Arg4 = FerroCad.00FEF87C

0012FFB4 | 00000000 | Arg5 = 0

0012FFB8 | 00000004 | Arg6 = 4

Ahora el problema es ver cómo averiguamos la respuesta correcta que debe dar la mochila. En este caso queryData vale 1807F616. Así vamos a llegar hasta el RET de RNBOsproQuery ponemos AX=0 y ponemos un valor cualquiera en FEF87C y le pondremos un memory breakpoint:

Address	Hex dump
00FEF858	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00FEF868	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00FEF878	16 F6 07 18 01 00 00 00 00 00 00 00 00 00 00 00
00FEF888	A3 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Memory breakpoints					
Address	Size	Module	Type	Status	Comment
00FEF878	00000004	FerroCad	RW	Active	
00FEF87C	00000004	FerroCad	RW	Active	
00FEFD40	00000004	FerroCad	RW	Disabled	
00FEFD48	00000004	FerroCad	RW	Disabled	

Ahora pulsamos F9 y llegamos a:

```
00FF2773 | > 8B4424 04 | MOV EAX,DWORD PTR SS:[ARG.1]
00FF2777 | . 8B00 | MOV EAX,DWORD PTR DS:[EAX]
00FF2779 | . C3 | RETN
```

En el que se pasa el contenido de FEF87C a EAX, si ahora traceamos con F8 llegamos a FF2D64:

00FF2D5F	• E8 DCF9FFFF	CALL 00FF2740	L FerroCad.00FF2740
00FF2D64	• 8B0D 7857FE0	MOV ECX, DWORD PTR DS:[0FE5778]	Pone en ECX el valor [FE5778]=FEFD44
00FF2D6A	• 83C4 08	ADD ESP, 8	
00FF2D6D	• 83E9 04	SUB ECX, 4	
00FF2D70	• A3 48FDFE0	MOV DWORD PTR DS:[0FEFD48], EAX	Almacena nuestro response en FEFD48
00FF2D75	• 8B0D 7857FE0	MOV DWORD PTR DS:[0FE5778], ECX	
00FF2D7B	• 8901	MOV DWORD PTR DS:[ECX], EAX	Almacena nuestro response en FEFD40
00FF2D7D	• E9 DFFAFFFF	JMP 00FF2861	Vuelve al inicio del bucle

En los comentarios de las líneas se explica lo que hace el código. Por lo que podemos ver, está guardando nuestra response ficticia en FEFD44 y FEFD40, por lo que pondremos BPM en ambas posiciones de memoria (si no utilizais el Olly 2 podeis utilizar HBP):

Address	Hex dump
00FEFD40	01 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
00FEFD50	03 CD 34 02 00 00 00 00 00 00 00 00 00 00 00 00
00FEFD60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00FEFD70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Address	Size	Module	Type	Status	Comment
00FEF878	00000004	FerroCad	RW	Active	
00FEF87C	00000004	FerroCad	RW	Active	
00FEFD40	00000004	FerroCad	RW	Active	
00FEFD48	00000004	FerroCad	RW	Active	

Damos F9 de nuevo y vuelve a parar en:

00FF2894	• 81F9 10590000	CMP ECX, 5910	
00FF289A	• 8915 48FDFE0	MOV DWORD PTR DS:[0FEFD48], EDX	
00FF28A0	• 892D 7C57FE0	MOV DWORD PTR DS:[0FE577C], EBP	
00FF28A6	• 0F8F 3D020000	JG 00FF2AE9	
00FF28AC	• 0F84 6D040000	JE 00FF2D1F	

EDX=FerroCad.00FEF878  
[00FEFD48]=1

Donde va a sobrescribir lo que tenemos en FEFD48, por lo tanto ya podemos quitar el BPM que teníamos ahí, ya que nuestro valor ya no está, sólo quedarán los otros 3:

Address	Size	Module	Type	Status
00FEF878	00000004	FerroCad	RW	Active
00FEF87C	00000004	FerroCad	RW	Active
00FEFD40	00000004	FerroCad	RW	Active
00FEFD48	00000004	FerroCad	RW	Disabled

Continuamos con F9 y para debido al BPM en FEF878 llegamos a un punto en el que guarda el valor de queryData en EAX y posteriormente lo Xorea con nuestra response:

00FF2C36	• 8B0D 7857FE0	MOV ECX, DWORD PTR DS:[0FE5778]	
00FF2C3C	• 83C4 08	ADD ESP, 8	
00FF2C3F	• 3101	XOR DWORD PTR DS:[ECX], EAX	Xorea nuestra response con 1807F616
00FF2C41	• E9 1BFCFFFF	JMP 00FF2861	Vuelve al inicio del bucle
00FF2C46	> 8B08	MOV ECX, DWORD PTR DS:[EAX]	
00FF2C48	• 51	PUSH ECX	

EAX=1807F616  
[00FEFD40]=00000001

El resultado (1807F617) se guarda en FEFD40.

Address	Hex dump
00FEFD20	00 00 00 00 00 00 00 00 00 00 00 00 04 00 00 00
00FEFD30	00 00 00 00 7C F8 FE 00 78 F8 FE 00 08 00 00 00
00FEFD40	17 F6 07 18 00 00 00 00 78 F8 FE 00 00 00 00 00
00FEFD50	BB 74 34 02 00 00 00 00 00 00 00 00 00 00 00 00
00FEFD60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Esto es muy extraño, pero bueno, pulsemos F9 otra vez con lo que llegamos a FF2BFF:

00FF2BFF	>	8B10	MOV	EDX, DWORD PTR DS:[EAX]	Mete [FEFD40]=1807F617 en EDX
00FF2C01	.	33C9	XOR	ECX, ECX	Hace ECX=0
00FF2C03	.	66:8BCE	MOV	CX, SI	
00FF2C06	.	83C0 04	ADD	EAX, 4	
00FF2C09	.	3BD1	CMP	EDX, ECX	Compara ECX y EDX
00FF2C0B	.	A3 7857FE00	MOV	DWORD PTR DS:[0FE5778], EAX	
00FF2C10	.	0F85 50FCFF	JNE	00FF2866	Si EDX no es 0 salta al inicio del bucle
00FF2C16	.	83E8 04	SUB	EAX, 4	
00FF2C19	.	A3 7857FE00	MOV	DWORD PTR DS:[0FE5778], EAX	
00FF2C1E	.	8928	MOV	DWORD PTR DS:[EAX], EBP	
00FF2C20	.	A1 48FDFE00	MOV	EAX, DWORD PTR DS:[0FEFD48]	
00FF2C25	.	A3 7C57FE00	MOV	DWORD PTR DS:[0FE577C], EAX	
00FF2C2A	.	E9 32FCFFFF	JMP	00FF2861	

Lo que hace el código está comentado en cada línea, pero resumiendo lo que ha hecho el código es lo siguiente:

EDX = XOR response, queryData  
 CMP EDX, 0  
 JNE FF2866

O lo que es lo mismo:

CMP response, queryData  
 JNE FF2866

Vamos que nos está tendiendo otra pequeña trampa, ya que en este caso nunca va a ser response = queryData, ya que la celda sobre la que estamos aplicando la función Query no es la 0, sino la 8, por lo que siempre será response ≠ query Data. De hecho, si forzáramos a que el salto no se tomara el programa nos sacaría a FindNextUnit, lo que es señal que vamos por mal camino. Así que dejamos todo como está y continuamos con F9 y llegamos primero a un CALL EDX en el que EDX=FF2610:

00FF2600	funcQueryData	:	A1 88F8FE00	MOV	EAX, DWORD PTR DS:[0FEF888]
00FF2605		:	8B0485 1020F	MOV	EAX, DWORD PTR DS:[EAX*4+0FE2010]
00FF260C		:	C3	RET	
00FF260D		:	90	NOP	
00FF260E		:	90	NOP	
00FF260F		:	90	NOP	
00FF2610	funcQueryResponse	:	A1 88F8FE00	MOV	EAX, DWORD PTR DS:[0FEF888]
00FF2615		:	8B0485 E027F	MOV	EAX, DWORD PTR DS:[EAX*4+0FE27E0]
00FF261C		:	C3	RET	
00FF261D		:	90	NOP	
00FF261E		:	90	NOP	
00FF261F		:	90	NOP	

[00FE2E6C]=A65A612C  
 EAX=000001A3 (decimal 419.)

Curiosamente, esta función es muy similar a la que vimos antes que cogía un número aleatorio de una tabla para a partir de ahí empezar a calcular queryData, y se encuentra justo debajo Ahí se coge un número almacenado en FEF888 y que varía en cada ejecución y en base a ese número se coge el valor que se encuentra en una tabla que empieza en FE27E0. En este caso va a coger el valor que se encuentra en FE2E6C = A65A612C. Además, si nos fijamos, cuando llegó a la función funcQueryData el valor que había en FEF888 era el mismo que hay ahora, con lo que el elemento que va a coger de la tabla estará situado en una dirección FE27E0 – FE2710 = 7D0. Como tenemos el valor en EAX, vamos a tracear con F8 para ver qué hace con él:

00FF2A78	.	FFD2	CALL	EDX
00FF2A7A	.	8B0D 7857FE00	MOV	ECX, DWORD PTR DS:[0FE5778]
00FF2A80	.	8901	MOV	DWORD PTR DS:[ECX], EAX
00FF2A82	.	E9 DAFFDFFF	JMP	00FF2861
00FF2A87	.	56	PUSH	ESI

EAX=A65A612C  
 [00FEFD40]=A34D3F4C

Vemos que guarda el valor en FEFD40, por lo que ponemos un BPM en esa posición de memoria:



Address	Hex dump
00FEFD40	2C 61 5A A6 00 00 00 00 10 26 FF 00 00 00 00 00
00FEFD50	44 1A 00 00 00 00 00 00 00 00 00 00 00 00 00
00FEFD60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Address	Size	Module	Type	Status	Comment
00FEF87C	00000004	FerroCad	RW	Disabled	
00FEFD40	00000004	FerroCad	RW	Active	
00FEFD48	00000004	FerroCad	RW	Disabled	

Y continuamos con F9 hasta que vuelve a parar en FF2C3F donde hace XOR del valor con 7B2309A7:

00FF2C3F	3101	XOR	DWORD PTR DS:[ECX],EAX
00FF2C41	E9 1BFCFFFF	JMP	00FF2861

EAX=7B2309A7  
[00FEFD40]=A65A612C

Después de muchas repeticiones sabemos que este valor con el que Xorea el valor obtenido de la tabla es siempre constante. El resultado de este XOR (DD79688B) sigue guardado en FEFD40 así que si seguimos con F9:

00FF2C36	8B0D 7857FE0	MOV	ECX, DWORD PTR DS:[0FE5778]
00FF2C3C	83C4 08	ADD	ESP, 8
00FF2C3F	3101	XOR	DWORD PTR DS:[ECX],EAX
00FF2C41	E9 1BFCFFFF	JMP	00FF2861
00FF2C46	8B08	MOV	ECX, DWORD PTR DS:[EAX]
00FF2C48	51	PUSH	ECX

EAX=00000001  
[00FEFD40]=DD79688B

Llegamos al punto en que se compara ese valor con la response que pusimos a la función Query (1)

00FF2773	8B4424 04	MOV	EAX, DWORD PTR SS:[ARG.1]
00FF2777	8B00	MOV	EAX, DWORD PTR DS:[EAX]
00FF2779	C3	RET	

[00FEF87C]=1  
EAX=FerroCad.00FEF87C

00FF2C31	E8 0AFBFFFF	CALL	00FF2740
00FF2C36	8B0D 7857FE0	MOV	ECX, DWORD PTR DS:[0FE5778]
00FF2C3C	83C4 08	ADD	ESP, 8
00FF2C3F	3101	XOR	DWORD PTR DS:[ECX],EAX
00FF2C41	E9 1BFCFFFF	JMP	00FF2861

EAX=00000001  
[00FEFD40]=DD79688B

Si el resultado de este XOR no es 0 (si response ≠ DD79688B) entonces el programa nos tira a FindNextUnit, por lo que debemos hacer que sea 0, y entonces nos llevará al siguiente Query.

Resumiendo, para obtener la respuesta correcta al 2º RNBOsproQuery debemos hacer:

- 1) Tener un BP puesto en FF2600.
- 2) Cuando el programa llegue ahí apuntar la dirección de la que toma el dato, en nuestro caso ha sido FE269C:

00FF2600 FuncQueryData	A1 88F8FE0	MOV	EAX, DWORD PTR DS:[0FEF888]
00FF2605	8B0485 1020F	MOV	EAX, DWORD PTR DS:[EAX*4+0FE2010]
00FF260C	C3	RET	
00FF260D	90	NOP	

[00FE269C]=E5016FC0  
EAX=000001A3 (decimal 419.)

- 3) Sumar 7D0 a ese número (FE269C + 7D0 = FE2E6C)
- 4) Vamos a esa dirección de memoria y apuntamos lo que hay (A65A612C):

Address	Hex dump
00FE2E6C	2C 61 5A A6 03 A6 ED 3D
00FE2E74	2B 4F 29 33 86 BB 4A 73
00FE2E7C	50 C5 8D 3A D5 E0 91 43

- 5) Le aplicamos A65A612C XOR 7B2309A7 = DD79688B

- 6) Apuntamos ese valor, y cuando llegemos a RNBOsproQuery lo ponemos como response.

Con esto ya tendríamos emulado el 2º RNBOsproQuery.

## .-\*\* Repetimos: 3ª llamada a RNBOsproQuery \*\*.-

Una vez superada la 2ª llamada a RNBOsproQuery la tercera es muy fácil ya que se repite el mismo esquema que la anterior, así que no voy a volver a explicar todo el proceso, sino que lo haré según lo resumido antes:

Una vez hemos emulado correctamente el 2º Query mantenemos el BP en la función FF2600. Así cuando seguimos con la ejecución después del Query anterior Olly para precisamente en esa función, pero esta vez cambia la posición de la que lee el número:

```

00FF2600  A1 88F8FE00 MOV EAX,DWORD PTR DS:[0FEF888]
00FF2605  8B0485 1020F MOV EAX,DWORD PTR DS:[EAX*4+0FE2010]
00FF260C  C3 RETN
00FF260D  90 NOP
[0FE25D8]=63786C53
EAX=00000172 (decimal 370.)

```

La posición es FE25D8. Si le sumamos 7D0: FE25D8 + 7D0 = FE2DA8. Si vamos a esa posición tenemos el valor 99231A16:

Address	Hex dump
00FE2DA8	16 1A 23 99 AA B1 59 2B 70 FC 21 B1 7F 04 D6 3F
00FE2DB8	75 AB 87 0B 59 02 74 5E 5D 0A BB C7 AF A9 56 CC

Hacemos 99231A16 XOR 7B2309A7 = E20013B1.

Pulsamos F9 para continuar hasta RNBOsproQuery y vemos que también aplica el algoritmo sobre la celda 8 por lo que en la dirección a la que apunta el parámetro response escribimos ese número:

Address	Hex dump	ASCII
00FEF878	85 F5 7E 9E B1 13 00 E2	â\$~>!!..0
00FEF880	00 00 00 00 00 00 00 00	.....

Llegamos hasta el RET del Query, ponemos EAX=0 y damos F9 y la ejecución continúa sin problemas:

## .-\*\* Nueva llamada a RNBOsproRead \*\*.-

Esta vez lee el contenido de la celda 3F y lo guarda en FEFCC0. Lo solucionamos poniendo en esa posición el valor FFFF:

```

00FF4D20  RNBORead(x,x,x)  $ 56 PUSH ESI
00FF4D21  57 PUSH EDI
00FF4D22  8B7C24 0C MOV EDI,DWORD PTR SS:[ARG.1]
00FF4D26  85FF TEST EDI,EDI
0012FFAC  00FF2CC3 t. RETURN to FerroCad.<Module>
0012FFB0  00FEF8BC Arg1 = FerroCad.0FEF8BC
0012FFB4  0000003F ? Arg2 = 3F
0012FFB8  00FEFCC0 Arg3 = FerroCad.0FEFCC0

```

Address	Hex dump
00FEFCC0	FF FF 34 12 01 00 00 00
00FEFCC8	00 00 00 00 00 00 00 00
00FEFCD0	00 00 00 00 00 00 00 00

Y no se nos olvide poner AX=0 cuando llegemos al RET.

Así damos F9 de nuevo y llegamos otra llamada a EDX.

## -\*\* Sencillo antidebug \*\*-

Este CALL EDX donde EDX vale FF3EF0 es donde se hace la única comprobación antidebug de la protección, mediante una simple llamada poco disimulada a IsDebuggerPresent:

```

00FF3EF0 68 8858FE00 PUSH OFFSET FerroCad.00FE5888
00FF3EF5 68 7C58FE00 PUSH OFFSET FerroCad.00FE587C
00FF3EFA FF15 6011FF00 CALL DWORD PTR DS:[<&KERNEL32.GetModuleLe
00FF3F00 50          PUSH EAX
00FF3F01 FF15 7011FF00 CALL DWORD PTR DS:[<&KERNEL32.GetProcAd
00FF3F07 85C0        TEST EAX,EAX
00FF3F09 A3 5CFDFE00 MOV DWORD PTR DS:[0FEF05C],EAX
00FF3F0E 74 02       JE SHORT 00FF3F12
00FF3F10 FFE0        JMP EAX
00FF3F12 E9 49000000 JMP 00FF3F60

```

Procname = "IsDebuggerPresent"  
Module Name = "KERNEL32"  
hModule = KERNEL32.GetModuleHandleA  
hProc = KERNEL32.GetProcAddress

Sabemos que a la salida de esa API debemos hacer que EAX=0. Y con esto ya queda solucionado el problema del antidebug.

## -\*\* Más problemas: 4ª llamada a RNBOsproQuery \*\*-

Pulsamos F9 y llegamos primero a un CALL EDX en el que se verifican permisos de escritura en una zona de memoria:

```

00FF27A0 A1 7CF8FE00 MOV EAX,DWORD PTR DS:[0FEF87C]
00FF27A5 56          PUSH ESI
00FF27A6 57          PUSH EDI
00FF27A7 B9 05000000 MOV ECX,5
00FF27AC 8D 404800 LEA EAX,[EAX*4+EAX]
00FF27AF BF DC2FFE00 MOV EDI,OFFSET FerroCad.00FE2FDC
00FF27B4 8D 3485 DC2FF LEA ESI,[EAX*4+0FE2FDC]
00FF27BB F3:A5      REP MOVSD DWORD PTR ES:[EDI],DWORD PTR DI
00FF27BD 8B 00 8CF7FE0 MOV ECX,DWORD PTR DS:[0FEF78C]
00FF27C3 8B 15 DC2FFE0 MOV EDX,DWORD PTR DS:[0FE2FDC]
00FF27C9 8D 040A00 LEA EAX,[ECX+EDX]
00FF27CC 8B 00 E02FFE0 MOV ECX,DWORD PTR DS:[0FE2FE0]
00FF27D2 51          PUSH ECX
00FF27D3 50          PUSH EAX
00FF27D4 A3 80F8FE00 MOV DWORD PTR DS:[0FEF880],EAX
00FF27D9 8B 00 78F8FE0 MOV DWORD PTR DS:[0FEF878],ECX
00FF27DF FF15 3410FF00 CALL DWORD PTR DS:[<&KERNEL32.IsBadWrite
00FF27E5 5F          POP EDI
00FF27E6 5E          POP ESI
00FF27E7 C3          RETN

```

Size => [0FE2FE0] = 4096.  
Addr  
KERNEL32.IsBadWritePtr

Aquí no debemos hacer nada, sólo pulsar F9 para llegar al 4º RNBOsproQuery. En este caso es diferente a los anteriores, ya que aunque también aplica el algoritmo Query sobre la celda 8, nos llama la atención que ni pasa por la función FF2600 para generar el queryData ni por la FF2610 para generar el response. Además utiliza el valor que le demos como response como key para desenscriptar una zona del ejecutable. Por tanto, o averiguamos el valor correcto, o el ejecutable no se desenscriptará bien y por tanto no funcionará. Veamos cómo hacerlo.

Nos encontramos aquí:

```

00FF4E60 RNBOQuery(x,x,x,x,x,x) 56          PUSH ESI
00FF4E61 57          PUSH EDI
00FF4E62 8B 7424 0C   MOV ESI,DWORD PTR SS:[ARG.1]
00FF4E66 85F6        TEST ESI,ESI
0012FFA0 C0FF2966 f) . RETURN to FerroCad.<ModuleEntryPoint>+176
0012FFA4 00FEF8BC #0. Arg1 = FerroCad.0FEF8BC
0012FFA8 00000008 0. Arg2 = 8
0012FFAC 00FE2FE4 8. Arg3 = FerroCad.0FE2FE4
0012FFB0 00FEF874 t. Arg4 = FerroCad.0FEF874
0012FFB4 00000000 .... Arg5 = 0
0012FFB8 00000004 . Arg6 = 4

```

Como valor de QueryData tenemos el valor 6654906E:

Address	Hex dump
00FE2FE4	6E 90 54 66 00 00 00 00
00FE2FEC	00 00 00 00 00 00 00 00
00FE2FF4	00 00 00 00 00 00 00 00

Este valor es constante en todas las ejecuciones del programa, un problema menos, pero no sabemos qué valor poner en FEF874. Pondremos un valor cualquiera y le pondremos un BPM para ver qué hace con él, además recordar como siempre poner AX=0 a la salida de la función:

Address	Hex dump
00FEF874	01 00 00 00 00 10 00 00
00FEF87C	00 00 00 00 00 80 FC 00

Si damos a F9 paramos en FF2777 donde se guarda en EAX el valor que hemos escrito en FEF874:

00FF2773	> 8B4424 04	MOV EAX, DWORD PTR SS:[ARG.1]
00FF2777	• 8B00	MOV EAX, DWORD PTR DS:[EAX]
00FF2779	• C3	RETN
00FF277A	> 3D 8E020000	CMP EAX, 28E

[00FEF874]=1  
EAX=FerroCad.00FEF874

Si seguimos con F8 llegamos a:

00FF2D5F	• E8 DCF9FFFF	CALL 00FF2740	L FerroCad.00FF2740
00FF2D64	• 8B0D 7857FE0	MOV ECX, DWORD PTR DS:[0FE5778]	Pone en ECX el valor [FE5778]=FEFD38
00FF2D6A	• 83C4 08	ADD ESP, 8	
00FF2D6D	• 83E9 04	SUB ECX, 4	
00FF2D70	• A3 48FDFE00	MOV DWORD PTR DS:[0FEFD48], EAX	Almacena nuestro response en FEFD48
00FF2D75	• 890D 7857FE0	MOV DWORD PTR DS:[0FE5778], ECX	
00FF2D7B	• 8901	MOV DWORD PTR DS:[ECX], EAX	Almacena nuestro response en FEFD38
00FF2D7D	• ^ E9 DFFAFFFF	JMP 00FF2861	Vuelve al inicio del bucle

EAX=1  
[00FEFD38]=FerroCad.00FE2FE4

Así que ponemos BPM en FEFD38 y FEFD48:

Address	Hex dump	ASCII
00FEFD38	01 00 00 00 00 10 00 00	0...>..
00FEFD40	00 80 FC 00 00 00 00 00	.C?.....
00FEFD48	01 00 00 00 00 00 00 00	0.....
00FEFD50	03 CD 34 02 00 00 00 00	*=40.....

**M Memory breakpoints**

Address	Size	Module	Type	Status
00FEF874	00000004	FerroCad	RW	Active
00FEFD38	00000004	FerroCad	RW	Active
00FEFD48	00000004	FerroCad	RW	Active

Si pulsamos F9 para en una zona en la que se sobrescribe lo que hay en FEFD48 con otro valor por lo que podemos quitar ese BPM, con lo que nos quedamos sólo con los otros 2.

Address	Hex dump	ASCII
00FEFD38	01 00 00 00 00 10 00 00	0...>..
00FEFD40	00 80 FC 00 00 00 00 00	.C?.....
00FEFD48	30 2E FF 00 00 00 00 00	0.....
00FEFD50	23 B3 00 00 00 00 00 00	#l.....

**M Memory breakpoints**

Address	Size	Module	Type	Status
00FEF874	00000004	FerroCad	RW	Active
00FEFD38	00000004	FerroCad	RW	Active
00FEFD48	00000004	FerroCad	RW	Disabled

Ahora volvemos a pulsar en F9 y llegamos a un punto en que empuja nuestra response a la pila para después hacer una llamada a una función que yo he llamado Decrypt y que está situada en FF2E30:

00FF2CB6	> 8B08	MOV ECX, DWORD PTR DS:[EAX]	Guarda nuestro response en ECX
00FF2CB8	• 51	PUSH ECX	Lo empuja a la pila
00FF2CB9	• 8B48 04	MOV ECX, DWORD PTR DS:[EAX+4]	
00FF2CBC	• 8B40 08	MOV EAX, DWORD PTR DS:[EAX+8]	
00FF2CBF	• 51	PUSH ECX	
00FF2CC0	• 50	PUSH EAX	
00FF2CC1	• FF02	CALL EDI	FerroCad.Decrypt

Esta función utiliza como argumentos 3 parámetros: Decrypt(dirección de memoria a desencriptar, longitud a desencriptar en bytes, key desencriptadora). Como keydesencriptadora se utiliza la response que hemos dado a la función RNBOsproQuery:

0012FFB0	00FC8000	.C?..
0012FFB4	00001000	.>...
0012FFB8	00000001	0...



Si traceamos dentro de la función va haciendo una serie de operaciones sobre las direcciones de memoria, las sobrescribe y además va sumando los valores que va obteniendo en EAX, de manera que cuando la función llega al RETN 0C, tenemos un valor en EAX que varía según la key desencryptadora que utilizemos. :

```

CPU - main thread, module FerroCad
00FF2E2E 90 NOP
00FF2E2F 90 NOP
00FF2E30 DeCrypt 8B4C24 08 MOV ECX,DWORD PTR SS:[ARG.2]
00FF2E34 33C0 XOR EAX,EAX
00FF2E36 C1E9 02 SHR ECX,2
00FF2E39 8BD1 MOV EDX,ECX
00FF2E3B 49 DEC ECX
00FF2E3C 85D2 TEST EDX,EDX
00FF2E3E 74 38 JNE SHORT 00FF2E78
00FF2E40 8B5424 04 MOV EDX,DWORD PTR SS:[ARG.1]
00FF2E44 53 PUSH EBX
00FF2E45 56 PUSH ESI
00FF2E46 8D71 01 LEA ESI,[ECX+1]
00FF2E49 8B4C24 14 MOV ECX,DWORD PTR SS:[ARG.3]
00FF2E4D 57 PUSH EDI
00FF2E4E 8BF9 MOV EDI,ECX
00FF2E50 8BD9 MOV EBX,ECX
00FF2E52 C1E7 04 SHL EDI,4
00FF2E55 03F9 ADD EDI,ECX
00FF2E57 83C2 04 ADD EDX,4
00FF2E5A C1EF 09 SHR EDI,9
00FF2E5D C1E3 05 SHL EBX,5
00FF2E60 33FB XOR EDI,EBX
00FF2E62 03CF ADD ECX,EDI
00FF2E64 8B7A FC MOV EDI,DWORD PTR DS:[EDX-4]
00FF2E67 33F9 XOR EDI,ECX
00FF2E69 8BD9 MOV EBX,EDI
00FF2E6B 897A FC MOV DWORD PTR DS:[EDX-4],EDI
00FF2E6E 03C3 ADD EAX,EBX
00FF2E70 33C8 XOR ECX,EAX
00FF2E72 4E DEC ESI
00FF2E73 75 D9 JNE SHORT 00FF2E4E
00FF2E75 5F POP EDI
00FF2E76 5E POP ESI
00FF2E77 5B POP EBX
00FF2E78 C2 0C00 RETN 0C
  
```

En este caso, EAX sale valiendo 3B16C3A5:

```

CPU - main thread, module FerroCad
00FF2E73 75 D9 JNE SHORT 00FF2E4E
00FF2E75 5F POP EDI
00FF2E76 5E POP ESI
00FF2E77 5B POP EBX
00FF2E78 C2 0C00 RETN 0C
00FF2E7B 90 NOP
00FF2E7C 90 NOP
00FF2E7D 90 NOP
00FF2E7E 90 NOP
00FF2E7F 90 NOP

Registers (FPU)
EAX 3B16C3A5
ECX AE8E478E
EDX 00FC9000 FerroCad.00FC9000
EBX 7FFDA000
ESP 0012FFAC
EBP 00FE4F40 FerroCad.00FE4F40
ESI 00120000
EDI 003D3FC0
  
```

Sigamos traceando a ver si hace algo con este valor:

```

00FF2CC1 FFD2 CALL EDX
00FF2CC3 8B0D 7857FE01 MOV ECX,DWORD PTR DS:[0FE5778]
00FF2CC9 8941 08 MOV DWORD PTR DS:[ECX+8],EAX
00FF2CCD A1 7857FE00 MOV EAX,DWORD PTR DS:[0FE5778]
00FF2CD1 83C0 08 ADD EAX,8
00FF2CD4 A3 7857FE00 MOV DWORD PTR DS:[0FE5778],EAX
00FF2CD9 E9 88BFFFFF JMP 00FF2866

EAX=3B16C3A5
[00FEFD40]=3B16C3A5
  
```

Pues en FF2CC9 parece que guarda el valor en FEFD40, así que le ponemos un BPM ahí también y continuamos llegamos a:

```

00FF2C36 8B0D 7857FE01 MOV ECX,DWORD PTR DS:[0FE5778]
00FF2C3C 83C4 08 ADD ESP,8
00FF2C3F 3101 XOR DWORD PTR DS:[ECX],EAX
00FF2C41 E9 1BFCEFFF JMP 00FF2861

EAX=00000000
[00FEFD40]=3B16C3A5
  
```

Donde aplica XOR a nuestro valor con 0 y guarda el resultado en FEFD40, y si damos a F9 parará en:

```

00FF2923 > 8B08 MOV ECX, DWORD PTR DS:[EAX]
00FF2925 . 83C0 04 ADD EAX, 4
00FF2928 . 85C9 TEST ECX, ECX
00FF292A . A3 7857FE00 MOV DWORD PTR DS:[0FE5778], EAX
00FF292F . 0F84 31FFFFFF JE 00FF2866
00FF2935 . 8915 7C57FE00 MOV DWORD PTR DS:[0FE577C], EDX
00FF293B . E9 26FFFFFF JMP 00FF2866

[00FEFD40]=3B16C3A5
ECX=00000234 (decimal 564.)
Jump from <ModuleEntryPoint>+112

```

Donde guarda el resultado de ese XOR en ECX, y dependiendo de si ECX es 0 o distinto de 0, tomará o no el salto condicional que hay en FF292F. He comprobado que si ECX es distinto de 0 la ejecución nos saca fuera del programa, por lo que ya sabemos que para que todo vaya bien el valor de EAX cuando se sale de la función Decrypt debe ser 0.

Para conseguir esto solo puede hacerse con fuerza bruta. Una vez hecho esto, vemos que el valor de response en el 4º RNBOsproQuery debe ser 28E3AE12.

Si reiniciamos todo y volvemos a llegar a este 4º RNBOQuery y ponemos en response el valor obtenido cuando llegamos a la función Decrypt vemos que poco a poco todos los valores que hay a partir de FC8000 se van transformando en ceros:

Address	Hex dump
00FC8000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00FC8010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00FC8020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00FC8030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00FC8040	DD 69 29 B8 70 AB CB 7F 95 C0 AA 4F DA 5D 2B 3D
00FC8050	2A 27 8F 00 C3 76 6F C4 AD 51 AD 6B 60 17 A5 35
00FC8060	CB CF BB 2B 51 72 0C F5 25 BF B5 18 CE 6D 4A 92
00FC8070	68 3B E9 A3 D9 2A 5B D4 23 91 8B A1 F8 53 A8 B6
00FC8080	B7 6B CD 33 E1 33 64 DC 60 E2 AC 88 22 D8 63 4E

Así que ya tenemos el 4º RNBOsproQuery solucionado y podemos quitar todos los BPMs.

## .\* Preparacion de la llegada al OEP y otro RNBOsproRead\*.\*

Si continuamos la ejecución llegamos a un CALL EDX en que llama a una función que comprueba algo de la cabecera del ejecutable pero nosotros no tenemos que hacer nada, sólo continuar con F9:

```

00FF2620 . A1 BC2FE00 MOV EAX, DWORD PTR DS:[0FE2FBC]
00FF2625 . 53 PUSH EBX
00FF2626 . 55 PUSH EBP
00FF2627 . 8B2D 8CF7FE00 MOV EBP, DWORD PTR DS:[0FEF78C]
00FF262D . 8BDD MOV EBX, EBP
00FF262F . 56 PUSH ESI
00FF2630 . 2BD8 SUB EBX, EAX
00FF2632 . 891D 74F8FE00 MOV DWORD PTR DS:[0FEF874], EBX
00FF2638 . 0F84 8D000000 JE 00FF26CB
00FF263E . A1 CC2FE00 MOV EAX, DWORD PTR DS:[0FE2FBC]
00FF2643 . 8D3428 LEA ESI, [EBP+EAX]

```

La siguiente parada es en otro CALL EDX que llama a FF25C0 en que no hace nada:

```

00FF25C0 . A1 C42FE00 MOV EAX, DWORD PTR DS:[0FE2FC4]
00FF25C5 . 85C0 TEST EAX, EAX
00FF25C7 . 74 34 JE SHORT 00FF25FD
00FF25C9 . FF15 2810FF00 CALL DWORD PTR DS:[&KERNEL32.GetVersion]
00FF25CF . 25 000000C0 AND EAX, 000000C0
00FF25D4 . 3D 00000000 CMP EAX, 00000000
00FF25D9 . 74 22 JE SHORT 00FF25FD
00FF25DB . 8B0D C42FE00 MOV ECX, DWORD PTR DS:[0FE2FC4]
00FF25E1 . 8B15 8CF7FE00 MOV EDX, DWORD PTR DS:[0FEF78C]
00FF25E7 . A1 C82FE00 MOV EAX, DWORD PTR DS:[0FE2FC8]
00FF25EC . 68 78F8FE00 PUSH OFFSET Ferrocad.00FEF878
00FF25F1 . 6A 20 PUSH 20
00FF25F3 . 83D1 ADD EDX, ECX
00FF25F5 . 50 PUSH EAX
00FF25F6 . 52 PUSH EDX
00FF25F7 . FF15 2C10FF00 CALL DWORD PTR DS:[&KERNEL32.VirtualProtect]
00FF25FD . 33C0 XOR EAX, EAX
00FF25FF . C3 RETN

```

La siguiente parada tras volver a pulsar en F9 es en otro CALL EDX más, esta vez para llamar a la función que está en FF2250 que creo que lo que hace es crear la IAT del verdadero ejecutable:

```

00FF2250 83EC 1C SUB ESP,1C
00FF2253 8B0D 8CF7FE00 MOV ECX,DWORD PTR DS:[0FEF78C]
00FF2259 A0 4431FE00 MOV AL,BYTE PTR DS:[0FE3144]
00FF225E 53 PUSH EBX
00FF225F 55 PUSH EBP
00FF2260 8B59 3C MOV EBX,DWORD PTR DS:[ECX+3C]
00FF2263 56 PUSH ESI
00FF2264 57 PUSH EDI
00FF2265 33F6 XOR ESI,ESI
00FF2267 8D3C19 LEA EDI,[EBX+ECX]
00FF226A C74424 18 0000 MOV DWORD PTR SS:[LOCAL.4],0

```

A continuación hace otro CALL EDX esta vez para volver a llamar a RNBOsproRead para leer de nuevo la celda 3F y escribirla en FEFCC0. Igual que antes, nosotros escribiremos FFFF en esa dirección, cambiamos AX=0 al salir y continuamos con F9

```

00FF4D20 RNBOsproRead(x,x,x) 56 PUSH ESI
00FF4D21 57 PUSH EDI
00FF4D22 8B7C24 0C MOV EDI,DWORD PTR SS:[ARG.1]
00FF4D26 85FF TEST EDI,EDI
0012FFAC 00FF2CC3 RETURN to FerroCad.<Modu
0012FFB0 00FEF8BC Arg1 = FerroCad.0FEF8BC
0012FFB4 0000003F Arg2 = 3F
0012FFB8 00FEFCC0 Arg3 = FerroCad.0FEFCC0

```

Address	Hex dump
00FEFCC0	FF FF 34 12 01 00 00 00 00 00 00 00 00 00 00
00FEFCD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

## .-\*\* Últimos obstáculos: Creación de hilos\*\*.-

Si al salir del último RNBOsproRead damos a F9 llegamos un CALL EDX a la última función que intenta complicarnos la vida:

```

00FF2190 Crea hilo timer 51 PUSH ECX
00FF2191 51 2010FF00 CALL DWORD PTR DS:[<&KERNEL32.GetVersion>]
00FF2197 25 000000C0 AND EAX,C0000000
00FF219C 3D 00000000 CMP EAX,00000000
00FF21A1 74 63 JE SHORT 00FF2206
00FF21A3 56 PUSH ESI
00FF21A4 FF15 C010FF00 CALL DWORD PTR DS:[<&KERNEL32.GetCurrentProcess>]
00FF21AA 6A 02 PUSH 2
00FF21AC 6A 01 PUSH 1
00FF21AE 8B00 MOV ESI,EAX
00FF21B0 6A 00 PUSH 0
00FF21B2 68 ACF8FE00 PUSH OFFSET FerroCad.00FEF8AC
00FF21B7 56 PUSH ESI
00FF21B8 FF15 BC10FF00 CALL DWORD PTR DS:[<&KERNEL32.GetCurrentThread>]
00FF21BE 50 PUSH EAX
00FF21BF 56 PUSH ESI
00FF21C0 FF15 5811FF00 CALL DWORD PTR DS:[<&KERNEL32.DuplicateHandle>]
00FF21C6 85C0 TEST EAX,EAX
00FF21C8 5E POP ESI
00FF21C9 74 3B JE SHORT 00FF2206
00FF21CB A1 ACF8FE00 MOV EAX,DWORD PTR DS:[0FEF8AC]
00FF21D0 33C0 XOR EAX,EAX
00FF21D2 74 32 JE SHORT 00FF2206
00FF21D4 8D4424 00 LEA EAX,[LOCAL.0]
00FF21D8 50 PUSH EAX
00FF21DB 6A 00 PUSH 0
00FF21DD 68 3032FF00 PUSH FerroCad.00FF3230
00FF21E2 6A 00 PUSH 0
00FF21E4 6A 00 PUSH 0
00FF21E6 FF15 6411FF00 CALL DWORD PTR DS:[<&KERNEL32.CreateThread>]
00FF21EC 85C0 TEST EAX,EAX
00FF21EE 74 15 JE SHORT 00FF2206
00FF21F0 68 A031FF00 PUSH FerroCad.00FF31A0
00FF21F5 C705 B0F8FE00 MOV DWORD PTR DS:[0FEF8B01],1
00FF21FF 68 E8030000 PUSH 3
00FF2204 EB 1A JMP SHORT 00FF2220
00FF2206 33C0 XOR EAX,EAX
00FF2208 68 A030FF00 PUSH FerroCad.00FF30A0
00FF220D 66A1 0620FE00 MOV AX,WORD PTR DS:[0FE2006]
00FF2213 8D4400 LEA EAX,[EAX*4+EAX]
00FF2216 8D4400 LEA EAX,[EAX*4+EAX]
00FF2219 8D4400 LEA EAX,[EAX*4+EAX]
00FF221C C1E1 03 SHL ECX,3
00FF221F 51 PUSH ECX
00FF2220 6A 00 PUSH 0
00FF2222 6A 00 PUSH 0
00FF2224 68 D87FE000 PUSH OFFSET FerroCad.00FE5708
00FF2229 68 D87FE000 PUSH OFFSET FerroCad.00FE5708
00FF222E FF15 6011FF00 CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleA>]
00FF2234 50 PUSH EAX
00FF2235 FF15 7011FF00 CALL DWORD PTR DS:[<&KERNEL32.GetProcAddress>]
00FF2238 FF00 MOV EAX,0
00FF223D A3 90F7FE00 MOV DWORD PTR DS:[0FEF7901],EAX
00FF2242 59 POP ECX
00FF2243 C3 RETN

```

**Kernel32.GetVersion**

**Kernel32.GetCurrentProcess**

**Kernel32.GetCurrentThread**

**Kernel32.DuplicateHandle**

**Kernel32.CreateThread**

**Procname = "SetTimer"**

**Module Name = "USER32"**

**Kernel32.GetModuleHandleA**

**Kernel32.GetProcAddress**

En esta rutina vemos que en FF21E6 se trata de crear un hilo que llama a una función dentro del ejecutable (FF3230). Si vamos ahí vemos que al final hay una llamada a RNBOsproRead:

```

00FF339B | . 3C FF | CMP AL,0FF
00FF339D | ^ 0F84 9EFFFFFF | JE 00FF3241
00FF33A3 | . FF15 4410FF00 | CALL DWORD PTR DS:[<&KERNEL32.GetTickCount>]
00FF33A9 | . 2B05 98F7FE00 | SUB EAX,DWORD PTR DS:[0FEF798]
00FF33AF | . 3D E0930400 | CMP EAX,493E0
00FF33B4 | ^ 0F82 87FEFFFF | JB 00FF3241
00FF33BA | . 66:0FB605 0E20 | MOVZX AX, BYTE PTR DS:[0FE200E]
00FF33C2 | . 68 C0FCFE00 | PUSH OFFSET Ferrocad.00FEFCC0
00FF33C7 | . 50 | PUSH EAX
00FF33C8 | . 68 BCF8FE00 | PUSH OFFSET Ferrocad.00FEF8BC
00FF33CD | . E8 4E190000 | CALL RNBORead(x,x,x)
00FF33D2 | . 66:85C0 | TEST AX,AX
00FF33D5 | ^ 75 0B | JNE SHORT 00FF33E2
00FF33D7 | . 66:813D C0FCFE | CMP WORD PTR DS:[0FEFCC0],0FFFF
00FF33E0 | ^ 74 1B | JE SHORT 00FF33FD
00FF33E2 | . 66:0FB60D 0E20 | MOVZX CX, BYTE PTR DS:[0FE200E]
00FF33EA | . 66:8B15 0A20FE | MOV DX,WORD PTR DS:[0FE200A]
00FF33F1 | . 51 | PUSH ECX
00FF33F2 | . 52 | PUSH EDX
00FF33F3 | . 68 BCF8FE00 | PUSH OFFSET Ferrocad.00FEF8BC
00FF33F8 | . E8 231B0000 | CALL RNBODecrement(x,x,x)
00FF33FD | ^ 8105 98F7FE00 | ADD DWORD PTR DS:[0FEF798],493E0
00FF3407 | ^ E9 35FEFFFF | JMP 00FF3241

```

**KERNEL32.GetTickCount**  
 Arg3 = Ferrocad.0FEFCC0  
 Arg2 = Ferrocad.0FEF8BC  
 Arg1 = Ferrocad.0FEF8BC  
 Ferrocad.RNBORead(x,x,x)

Arg3  
 Arg2 = Ferrocad.0FEF8BC  
 Arg1 = Ferrocad.0FEF8BC  
 Ferrocad.RNBODecrement(x,x,x)

Esto lo que trata es de seguir haciendo llamadas a la mochila, incluso cuando la parte del Shell ha terminado de ejecutarse, por lo tanto lo que vamos a hacer es que cuando lleguemos a la linea FF21E6 vamos a cambiar el parámetro CreationFlag del valor 0 al valor 4 que significa CREATE\_SUSPENDED, para crearlo suspendido, y de esta manera que no se ejecuten esas llamadas a RNBOsproRead

```

00FF21D8 | . 50 | PUSH EAX
00FF21D9 | . 6A 00 | PUSH 0
00FF21DB | . 6A 00 | PUSH 0
00FF21DD | . 68 3032FF00 | PUSH Ferrocad.00FF3230
00FF21E2 | . 6A 00 | PUSH 0
00FF21E4 | . 6A 00 | PUSH 0
00FF21E6 | ^ FF15 6411FF00 | CALL DWORD PTR DS:[<&KERNEL32.CreateThread>]

```

pThreadId => OFFSET LOCAL.0  
 CreationFlags = 0  
 Parameter = NULL  
 StartAddress = Ferrocad.0FF3230  
 StackSize = 0  
 pSecurity = NULL

```

0012FF9C | 00000000 | .... | pSecurity = NULL
0012FFA0 | 00000000 | .... | StackSize = 0
0012FFA4 | 00FF3230 | 02 . | StartAddress = Ferrocad.0FF3230
0012FFA8 | 00000000 | .... | Parameter = NULL
0012FFAC | 00000004 | .... | CreationFlags = CREATE_SUSPENDED
0012FFB0 | 0012FFB4 | + . | pThreadId = 0012FFB4 -> 0

```

Si seguimos traceando con F8 llegamos a:

```

00FF221F | . 51 | PUSH ECX
00FF2220 | > 6A 00 | PUSH 0
00FF2222 | . 6A 00 | PUSH 0
00FF2224 | . 68 D857FE00 | PUSH OFFSET Ferrocad.00FE57D8
00FF2229 | . 68 D857FE00 | PUSH OFFSET Ferrocad.00FE57D8
00FF222E | . FF15 6011FF00 | CALL DWORD PTR DS:[<&KERNEL32.GetModuleHandleA>]
00FF2234 | . 50 | PUSH EAX
00FF2235 | . FF15 7011FF00 | CALL DWORD PTR DS:[<&KERNEL32.GetProcAddress>]
00FF223B | ^ FF00 | CALL EAX
00FF223D | . A3 90F7FE00 | MOV DWORD PTR DS:[0FEF790],EAX
00FF2242 | . 59 | POP ECX
00FF2243 | . C3 | RETN

```

Procname = "SetTimer"  
 ModuleName = "USER32"  
 hModule = KERNEL32.GetModuleHandleA  
 hModule = KERNEL32.GetProcAddress  
 USER32.SetTimer

```

0012FFA4 | 00000000 | .... | hWnd = NULL
0012FFA8 | 00000000 | .... | TimerID = 0
0012FFAC | 000003E8 | b . | Timeout = 1000. ms
0012FFB0 | 00FF31A0 | a1 . | TimerFunc = Ferrocad.0FF31A0

```

Quiere ponernos un timer de manera que la función que hay en FF31A0 se ejecute cada segundo. Si vamos a esa dirección:



```

00FF31A0 | . A1 B4F8FE00 | MOV EAX, DWORD PTR DS:[0FEF8B4]
00FF31A5 | . 53 | PUSH EBX
00FF31A6 | . 33DB | XOR EBX,EBX
00FF31A8 | . 3BC3 | CMP EAX,EBX
00FF31AA | . 74 74 | JE SHORT 00FF3220
00FF31AC | . 391D B8F8FE00 | CMP DWORD PTR DS:[0FEF8B8],EBX
00FF31B2 | . 75 6C | JNE SHORT 00FF3220
00FF31B4 | . 56 | PUSH ESI
00FF31B5 | . C705 B8F8FE00 | MOV DWORD PTR DS:[0FEF8B8],1
00FF31B7 | . 891D B4F8FE00 | MOV DWORD PTR DS:[0FEF8B4],EBX
00FF31C5 | . BE 02000000 | MOV ESI,2
00FF31CA | . E8 C1FCFFFF | CALL 00FF2E90
00FF31CF | . 66 85C0 | TEST AX,AX
00FF31D2 | . 74 3F | JE SHORT 00FF3213
00FF31D4 | . 46 | INC ESI
00FF31D5 | . 891D B4F8FE00 | MOV DWORD PTR DS:[0FEF8B4],EBX
00FF31D8 | . 83FE 01 | CMP ESI,1
00FF31DE | . 7C 29 | JL SHORT 00FF3209
00FF31E0 | . C705 84F8FE00 | MOV DWORD PTR DS:[0FEF8B4],1
00FF31EA | . E8 21020000 | CALL 00FF3410
00FF31EF | . 68 10200000 | PUSH 2010
00FF31F4 | . E8 67EEFFFF | CALL 00FF2060
00FF31F9 | . 50 | PUSH EAX
00FF31FA | . E8 F1080000 | CALL 00FF3AF0
00FF31FF | . 83C4 08 | ADD ESP,8
00FF3202 | . E8 99020000 | CALL 00FF34A0
00FF3207 | . 33F6 | XOR ESI,ESI
00FF3209 | . E8 82FCFFFF | CALL 00FF2E90
00FF320E | . 66 85C0 | TEST AX,AX
00FF3211 | . 75 C1 | JNE SHORT 00FF31D4
00FF3213 | . 891D B4F8FE00 | MOV DWORD PTR DS:[0FEF8B4],EBX
00FF3219 | . 891D B8F8FE00 | MOV DWORD PTR DS:[0FEF8B8],EBX
00FF321F | . 5E | POP ESI
00FF3220 | . 5B | POP EBX
00FF3221 | . C2 1000 | RETN 10

```

Hay una serie de llamadas a otras funciones. Por ejemplo dentro de FF2E90 hay una llamada a RNBOsproQuery y a RNBOsproWrite. Por tanto también debemos tratar de evitar esto. Lo haremos cambiando el parámetro Timeout de la función SetTimer de 3E8 a 0, para que nunca tenga lugar la llamada a esas funciones:

```

0012FFA4 | 00000000 | .... | hWnd = NULL
0012FFA8 | 00000000 | .... | TimerID = 0
0012FFAC | 00000000 | .... | Timeout = 0
0012FFB0 | 00FF31A0 | .! | TimerFunc = FerroCad.00FF31A0

```

### .-\*\* Llegando al OEP\*\*.-

Ahora si pulsamos F9 llegamos a un último CALL EDX esta vez a la función que está en FF3BE0, que básicamente comprueba la integridad del ejecutable, así que podemos continuar hasta el RET de esa función. Una vez allí, vamos poco a poco traceando con F8 hasta que llegamos a FF2DB1, que es un CALL que está casi al final del gran bucle que hemos estado ejecutando hasta el momento:

```

00FF2DAA | . 8B5424 04 | MOV EDX, DWORD PTR SS:[ARG.1]
00FF2DAE | . 50 | PUSH EAX
00FF2DAF | . 51 | PUSH ECX
00FF2DB0 | . 52 | PUSH EDX
00FF2DB1 | . FF15 A04BFE00 | CALL DWORD PTR DS:[0FE4BA0]
00FF2DB7 | . C2 0C00 | RETN 0C

```

Arg3 => [ARG.3]  
Arg2 => [ARG.2]  
Arg1 => [ARG.1]  
FerroCad.0040DCA8

Vemos que está haciendo CALL a 40DCA8. Entremos con F7 y llegamos a:

```

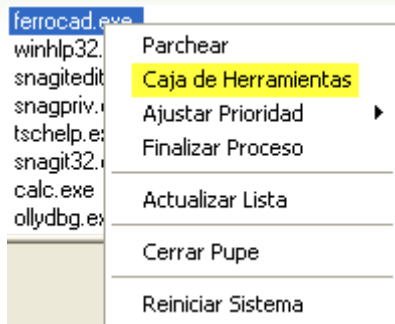
0040DC96 | - FF25 38114000 | JMP DWORD PTR DS:[401138]
0040DC9C | - FF25 9C114000 | JMP DWORD PTR DS:[40119C]
0040DCA2 | - FF25 84124000 | JMP DWORD PTR DS:[401284]
0040DCA8 | 68 40D04000 | PUSH OFFSET FerroCad.0040D040
0040DCAD | E8 F0FFFFFF | CALL <JMP.ThunRTMain>

```

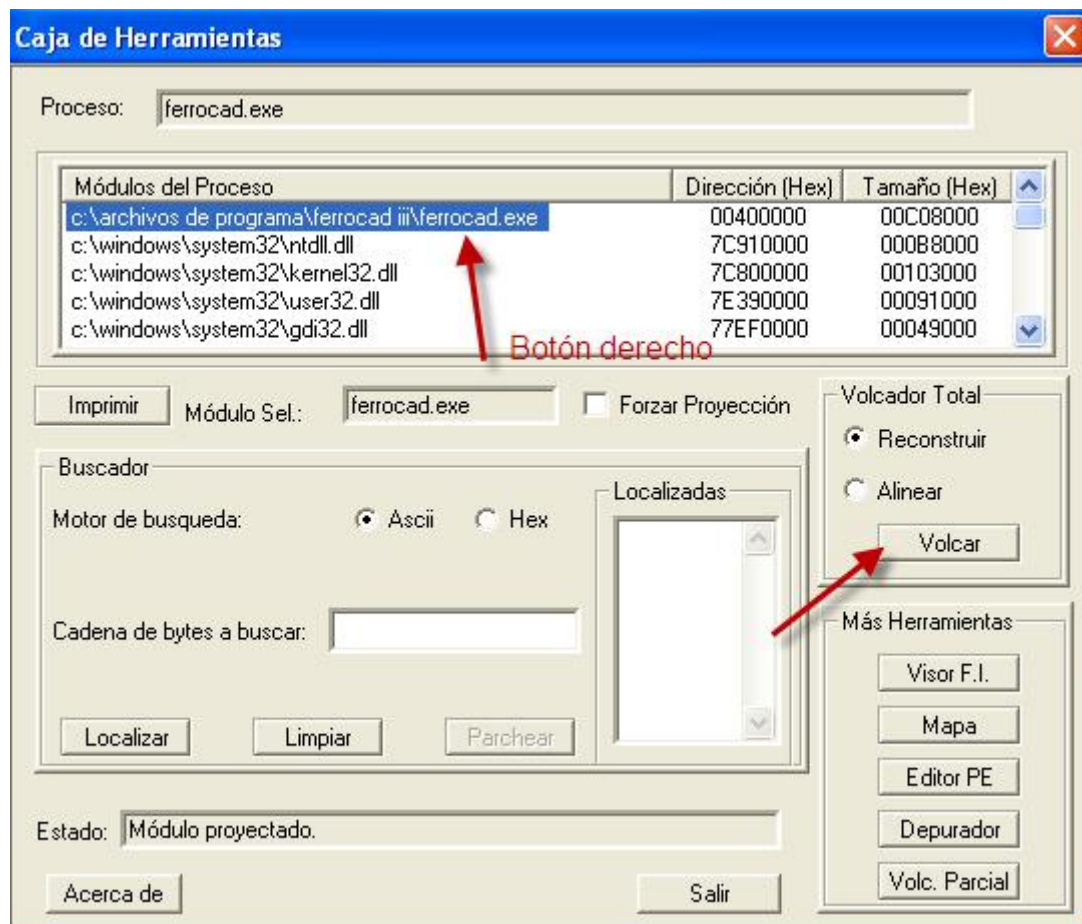
Jump to MSUBUM60.ThunRTMain

Que es el EP típico de los programas hechos en Visual Basic. Así que ESE ES NUESTRO OEP!!!.

Para hacer el dumpeado como no tenemos plugins, lo haré con el PUPE, buscamos el proceso, y con el botón derecho abrimos el menú contextual y pulsamos en la Caja de Herramientas:

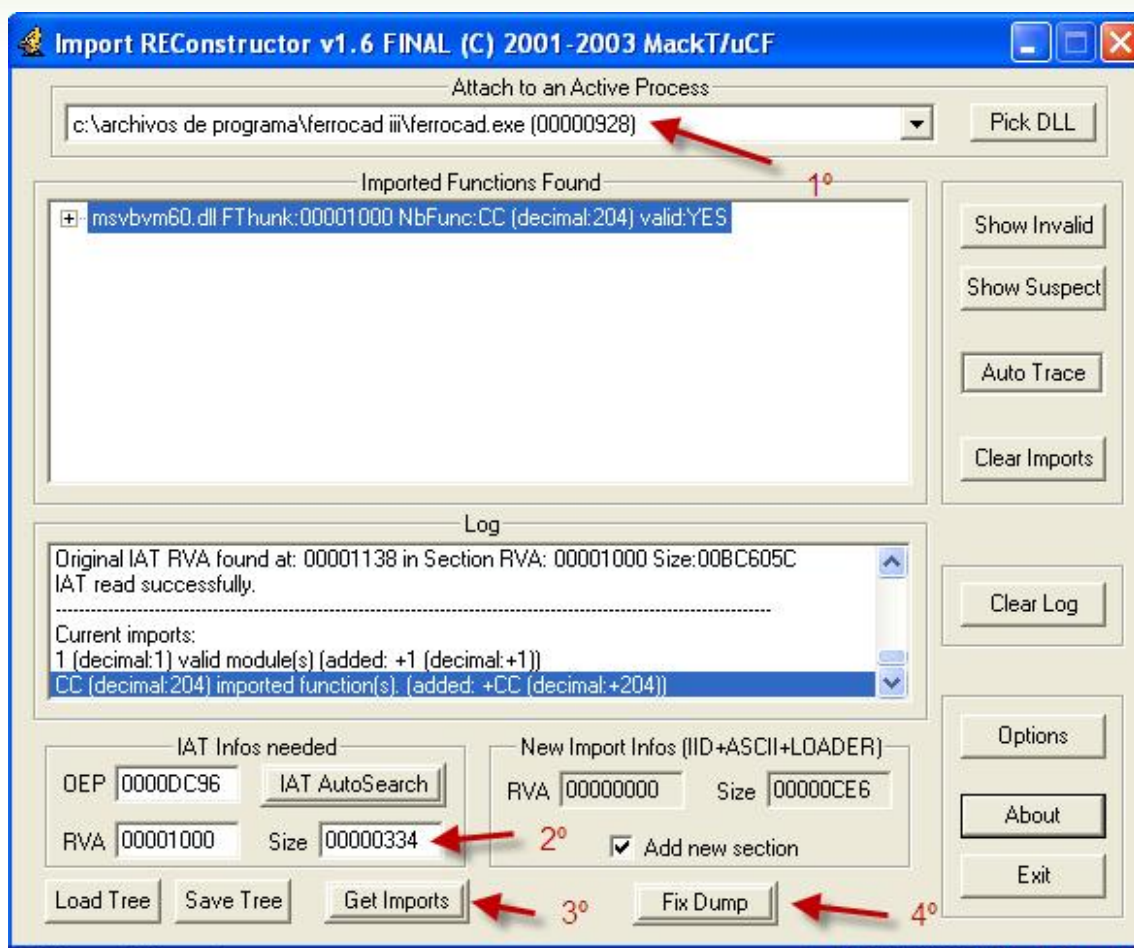


Aparecemos en la siguiente ventana en la que debemos pulsar con el botón derecho en derrocad.exe y luego con el izquierdo sobre el botón Volcar:



Le damos el nombre que queramos y ya lo tenemos.

Sólo falta arreglar la IAT, lo haremos con el Import Reconstructor, donde elegimos 1º el proceso derrocad.exe, después rellenamos los datos de la IAT y pulsamos en GetImports. Salen todas las funciones resueltas, así que damos a Fix Dump, y elegimos como archivo a fijar el dumpeado que generamos con el Pupe:



Vemos que lo genera satisfactoriamente:

```
Fixing a dumped file...
1 (decimal:1) module(s)
CC (decimal:204) imported function(s).
*** New section added successfully. RVA:0002B000 SIZE:00001000
Image Import Descriptor size: 14; Total length: CE6
C:\Archivos de programa\FerroCad III\armadodmpd .exe saved successfully.
```

Cogemos el ejecutable que ha generado el Import Reconstructor, lo ejecutamos y ...  
FUNCIONA!!!:

## .-\*\* Resumiendo \*\*.-

Bueno, pues este es el resumen de todo lo que tuve que hacer para saltar esta protección. Visto así no es nada del otro mundo, sin embargo, a mí me ha llevado casi un mes desde que empecé a darle vueltas hasta que por fin conseguí que funcionara. Pero por eso mismo la satisfacción ha sido mayor que cualquier otro programa que he conseguido destripar. Con esto quiero decir que por muy duro que pueda parecer algo, casi siempre el esfuerzo tiene su recompensa, y la mayoría de protecciones tienen un hueco por donde hincar el diente, así que a no desanimarse.

Básicamente en lo que se resume esta protección es en aplicar las funciones Read y Query y dirigir el flujo del programa en función de las respuestas recibidas, así que si veíamos que el flujo no era el deseado, pues cambiábamos la respuesta de manera que el flujo fuera por donde queríamos.

# CracksLatinoS! 2009

## **-\*\* Saludos y/o Agradecimientos \*\*-**

Este tuto va dedicado a todos los CracksLatinos, y en especial a Ricardo Narvaja, ya que sin su curso de Olly desde cero yo no hubiera sido capaz ni de atreverme a intentar ningún tipo de reto. También quiero nombrar desde aquí a Vortice y El\_Cid que me ayudaron mucho con otro programa que ellos saben cuál es **J**.

Un saludo a todos y hasta el próximo