



## CracksLatinoS! 2010

*-\* Otra mochila Sentinel \*-*

Programa	Ferrocad
Download	ftp://www.metsim.com/pub/METSIM1302.zip
Descripción	
Herramientas	OllyDbg v2.0, IDA.
Dificultad	Media
Compresor/Compilador	
Protección	Sentinel Dongle
Objetivos	Hacer correr el programa sin mochila
Cracker	<b>Lionel</b> lionelgomezdu@yahoo.es
	Fecha: 10/02/10
Tutorial nº	

### **.-\*\* Introducción \*\*-**

Pues parece que estoy destinado a vérmelas con dongles, ya que un miembro de la lista me pidió ayuda para un programa que estaba intentando desproteger y resulta que estaba protegido con una mochila Sentinel.

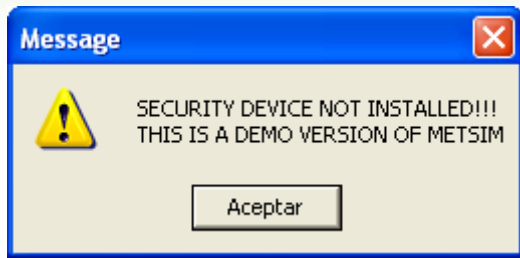
Este tutorial demuestra que los desarrolladores de software creen que porque se gasten un dineral en comprar una protección de este tipo, sus aplicaciones están seguras. El problema es que no ponen demasiado empeño en implantar este método de protección.

En este caso se basan en la lectura de 4 celdas de la mochila, de las cuales, sólo 2 se utilizan para decidir si la mochila es la correcta o no.

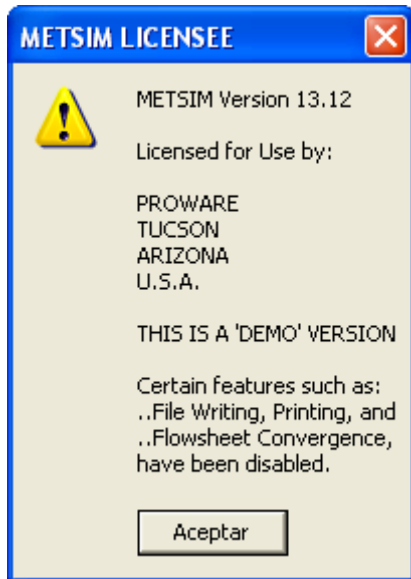
Lo que vamos a hacer es hacer una emulación para que el programa objetivo se comporte como si la dongle estuviera realmente puesta, y además como las funciones de llamada al dongle están en una dll, una vez modificada ésta, nos valdrá para todas las versiones del programa que han ido saliendo posteriormente. Vamos allá:

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

Tras instalar la aplicación la ejecutamos y nos aparece una ventana avisando de que no tenemos puesto el dispositivo de protección.



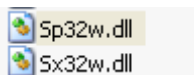
Cuando damos a aceptar nos aparece otra ventana que nos indica que estamos en una versión DEMO y que están deshabilitadas algunas opciones.



¿Cómo sabemos qué tipo de mochila es?. Pues tendremos que utilizar una herramienta de análisis muy potente y que en muy pocos casos se utiliza: la cabeza, jeje. En la página de descarga del programa aparece la opción de descargarnos los drivers de Sentinel, o sea, que está claro ¿no?.

Ahora lo mejor es cargar el programa en IDA y aplicarle el archivo de firmas de Sentinel para ver dónde están las llamadas a las funciones del API, pero tras terminar el análisis no vemos ninguna de las funciones típicas. ¿qué pasa?. Puede que estén en alguna dll.

Así que nos vamos al directorio donde se instaló el programa y allí vemos 2 archivos sospechosos:



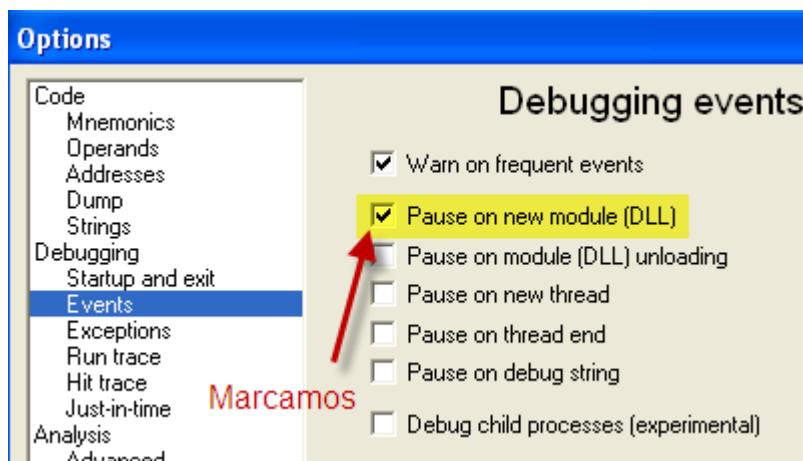
Si abrimos el primero no vemos nada útil, pero si cargamos en Olly Sx32w.dll en las funciones Exportadas vemos:

Names in SX32W			
Address	Section	Type	Name
10000000		Analysed	<STRUCT IMAGE_DOS_HEADER>
10000008		Analysed	<STRUCT IMAGE_NT_SIGNATURE>
10000084		Analysed	<STRUCT IMAGE_FILE_HEADER>
10000098		Analysed	<STRUCT IMAGE_OPTIONAL_HEADER>
100000F8		Analysed	<STRUCT IMAGE_DATA_DIRECTORY>
10000178		Analysed	<STRUCT IMAGE_SECTION_HEADER>
100001A0		Analysed	<STRUCT IMAGE_SECTION_HEADER>
100001C8		Analysed	<STRUCT IMAGE_SECTION_HEADER>
100001F0		Analysed	<STRUCT IMAGE_SECTION_HEADER>
10000218		Analysed	<STRUCT IMAGE_SECTION_HEADER>
10000240		Analysed	<STRUCT IMAGE_SECTION_HEADER>
10005940	.text	Export	RNB0sproFormatPacket
10005980	.text	Export	RNB0sproInitialize
100059E0	.text	Export	RNB0sproSetContactServer
10005AA0	.text	Export	RNB0sproGetContactServer
10005B30	.text	Export	RNB0sproFindFirstUnit
10005DF0	.text	Export	RNB0sproFindNextUnit
10005E60	.text	Export	RNB0sproRead
10005EF0	.text	Export	RNB0sproExtendedRead
10005F90	.text	Export	RNB0sproWrite
10006040	.text	Export	RNB0sproOverwrite
10006110	.text	Export	RNB0sproQuery
100061D0	.text	Export	RNB0sproActivate
10006280	.text	Export	RNB0sproDecrement
10006310	.text	Export	RNB0sproGetHardLimit
100063A0	.text	Export	RNB0sproGetVersion
100064F0	.text	Export	RNB0sproGetFullStatus
10006530	.text	Export	RNB0sproSetProtocol
10006620	.text	Export	RNB0sproGetSubLicense
10006670	.text	Export	RNB0sproReleaseLicense
10006700	.text	Export	RNB0sproSetHeartBeat
10006750	.text	Export	RNB0sproEnumServer
10006770	.text	Export	RNB0sproGetKeyInfo
10014BF0	.text	Export	<ModuleEntryPoint>
1001A1E8	.idata	Import	&ADVAPI32.RegCloseKey
1001A1EC	.idata	Import	&ADVAPI32.RegQueryValueExA
1001A1F0	.idata	Import	&ADVAPI32.GetUserNameA

Así que ya tenemos todos los datos necesarios para ponernos a trabajar.

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

Cargamos el programa en Olly y mientras estamos en el EP nos vamos directamente al mapa de memoria para localizar la dll y colocarle los BPs correspondientes, sin embargo la librería no aparece, por lo que se ve que se carga en tiempo de ejecución. Así que vamos a la configuración de Olly y le ponemos que pare al cargar una dll:



Así que pulsamos F9 y Olly irá parando varias veces hasta llegar a Sx32w.dll, en ese momento la cargamos en la ventana del desensamblador:

M Memory map								
Address	Size	Owner	Section	Contains	Type	Access	Initial	Map
00FE0000	00001000				Priv	RW	RW	
01000000	00053000				Priv	RW	RW	
01100000	04001000				Priv	RW	RW	
10000000	00001000	SX32W		PE header	Img	R	RWE	Cop
10001000	00014000	SX32W	.text	Code	Img	R	RWE	Cop
10015000	00001000	SX32W	.rdata	Exports				
10016000	00004000	SX32W	.data	Data				
1001A000	00001000	SX32W	.idata	Imports				
1001B000	00001000	SX32W	.rsrc	Resources				
1001C000	00002000	SX32W	.reloc	Relocations				
58C30000	00001000	COMCTL32		PE header				
58C31000	00071000	COMCTL32	.text	Code, imports, expo				

Y buscamos las funciones pulsando Ctrl+N, y ponemos un BP en cada una:

N Names in SX32W			
Address	Section	Type	Name
10000000		Analysar	<STRUCT IMAGE_DOS_HEADER>
10000080		Analysar	<STRUCT IMAGE_NT_SIGNATURE>
10000084		Analysar	<STRUCT IMAGE_FILE_HEADER>
10000098		Analysar	<STRUCT IMAGE_OPTIONAL_HEADER>
100000F8		Analysar	<STRUCT IMAGE_DATA_DIRECTORY>
10000178		Analysar	<STRUCT IMAGE_SECTION_HEADER>
100001A0		Analysar	<STRUCT IMAGE_SECTION_HEADER>
100001C8		Analysar	<STRUCT IMAGE_SECTION_HEADER>
100001F0		Analysar	<STRUCT IMAGE_SECTION_HEADER>
10000218		Analysar	<STRUCT IMAGE_SECTION_HEADER>
10000240		Analysar	<STRUCT IMAGE_SECTION_HEADER>
10005940	.text	Export	RNBOSproFormatPacket
10005980	.text	Export	RNBOSproInitialize
100059E0	.text	Export	RNBOSproSetContactServer
10005AA0	.text	Export	RNBOSproGetContactServer
10005B30	.text	Export	RNBOSproFindFirstUnit
10005BF0	.text	Export	RNBOSproFindNextUnit
10005E60	.text	Export	RNBOSproRead
10005EF0	.text	Export	RNBOSproExtendedRead
10005F90	.text	Export	RNBOSproWrite
10006040	.text	Export	RNBOSproOverwrite
10006110	.text	Export	RNBOSproQuery
100061D0	.text	Export	RNBOSproActivate
10006280	.text	Export	RNBOSproDecrement
10006310	.text	Export	RNBOSproGetHardLimit
100063A0	.text	Export	RNBOSproGetVersion
100064F0	.text	Export	RNBOSproGetFullStatus
10006530	.text	Export	RNBOSproSetProtocol
10006620	.text	Export	RNBOSproGetSubLicense
10006670	.text	Export	RNBOSproReleaseLicense
10006700	.text	Export	RNBOSproSetHeartBeat
10006750	.text	Export	RNBOSproEnumServer
10006770	.text	Export	RNBOSproGetKeyInfo
10014BF0	.text	Export	<ModuleEntryPoint>
1001A1E8	.idata	Import	&ADVAPI32.RegCloseKey

### .-\*\* Primera función a emular: RNBOSproFormatPacket\*\*.-

En principio con eso bastaría. Ahora pulsamos F9 y enseguida llegamos a la primera función:

CPU - main thread, module SX32W			
10005940	RNBOSproFormatPacket	56	PUSH ESI
10005941		8B7424 08	MOV ESI, DWORD PTR SS:[ARG.1]
10005945		85F6	TEST ESI, ESI
10005947		75 08	JNE SHORT 10005951
10005949		66:B8 1000	MOV AX, 10
1000594D		5E	POP ESI
1000594E		C2 0800	RET 8
10005951		56	PUSH ESI
10005952		E8 49990000	CALL 1000F2A0
10005957		66:8B08	MOV CX, WORD PTR DS:[EAX]

Según el Developer Guide de Sentinel:

### Format

```
00000000 000000 000 RNBOSproFormatPacket
```

```
RB_SPRO_APIPACKET packet
```

```
00000000 000000 000 0000000000
```

```
00
```

### Parameters

packet A pointer to a DWORD-aligned RB\_SPRO\_APIPACKET

record.

*packetLen* An integer containing the length of the RB\_SPRO\_APIPACKET record in bytes.

## 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. Para hacer la emulación modificaremos la rutina para dejarla de la siguiente manera:

```

CPU - main thread, module SX32W
10005940 RNBOsproFormatPacket 56 PUSH ESI
10005941 8B7424 08 MOV ESI, DWORD PTR SS:[ARG.1]
10005945 85F6 TEST ESI,ESI
10005947 90 NOP
10005948 90 NOP
10005949 66:B8 0000 MOV AX,0
1000594D 5E POP ESI
1000594E C2 0800 RETN 8
10005951 56 PUSH ESI
10005952 E8 49990000 CALL 1000F2A0
10005957 66:8B08 MOV CX, WORD PTR DS:[EAX]

```

Con esto ya estaría emulada la primera función.

## .-\*\* Segunda función a emular: RNBOsproInitialize\*\*-

Si pulsamos F9 llegamos a:

```

CPU - main thread, module SX32W
10005980 RNBOsproInitialize 66:833D 2866 CMP WORD PTR DS:[10016628],0
10005988 56 PUSH ESI
10005989 75 05 JNE SHORT 10005990
1000598B E8 00FFFFFF CALL 10005890
10005990 8B7424 08 MOV ESI, DWORD PTR SS:[ARG.1]
10005994 85F6 TEST ESI,ESI
10005996 75 08 JNE SHORT 100059A0
10005998 66:B8 1000 MOV AX,10
1000599C 5E POP ESI
1000599D C2 0400 RETN 4
100059A0 56 PUSH ESI
100059A1 E8 FA980000 CALL 1000F2A0
100059A6 66:8B08 MOV CX, WORD PTR DS:[EAX]

```

Según el Sentinel Developer Guide:

## Format

```

#####  #####  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. Para hacer la emulación dejaremos la rutina de la siguiente manera:

```

CPU - main thread, module SX32W
10005980 RNBOsproInitialize
10005988 . 66:833D 28661 CMP WORD PTR DS:[10016628],0
10005989 . 56 PUSH ESI
1000598B . 75 05 JNE SHORT 10005990
1000598B . E8 00FFFFFF CALL 10005890
10005990 . 8B7424 08 MOV ESI,DWORD PTR SS:[ARG.1]
10005994 . 85F6 TEST ESI,ESI
10005996 . 90 NOP
10005997 . 90 NOP
10005998 . 66:B8 0000 MOV AX,0
1000599C . 5E POP ESI
1000599D . C2 0400 RETN 4
100059A0 . 56 PUSH ESI
100059A1 . E8 FA980000 CALL 1000F2A0
100059A6 . 66:8B08 MOV CX,WORD PTR DS:[EAX]
100059A9 . 66:81F9 4272 CMP CX,7242
100059AF . 74 07 JF SHORT 100059B7

```

Ya tenemos emulada la segunda función.

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

Pulsamos F9 para continuar y llegamos a:

```

CPU - main thread, module SX32W
10005B30 RNBOsproFindFirstUnit
10005B33 . 83EC 48 SUB ESP,48
10005B3A . 66:C74424 02 MOV WORD PTR SS:[LOCAL.17+2],0
10005B3B . 53 PUSH EBX
10005B3B . 56 PUSH ESI
10005B3C . 8B5C24 54 MOV EBX,DWORD PTR SS:[ARG.1]
10005B40 . 57 PUSH EDI
10005B41 . 55 PUSH EBP
10005B42 . 85DB TEST EBX,EBX
10005B44 . 0F84 8B020000 JE 10005D05
10005B4A . 66:8B6C24 60 MOV BP,WORD PTR SS:[ARG.2]
10005B4F . 66:85ED TEST BP,BP
10005B52 . 0F84 7D020000 JE 10005D05
10005B58 . 66:81FD FFFF CMP BP,0FFFF
10005B5D . 75 0E JNE SHORT 10005B6D
10005B5F . 66:B8 0300 MOV AX,3
10005B63 . 5D POP EBP
10005B64 . 5F POP EDI
10005B65 . 5E POP ESI
10005B66 . 5B POP EBX
10005B67 . 83C4 48 ADD ESP,48
10005B6A . C2 0800 RETN 8
10005B6D . 53 PUSH EBX
10005B6E . E8 2D970000 CALL 1000F2A0
10005B72 . 8BFA MOV ESI,EDI

```

Sabemos que:

## Format

```

#####  #####  RNBOsproFindFirstUnit
RB_SPRO_APIPACKET packet
#####  #####  #####

```

## 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, así que modificaremos de la siguiente forma:

```

CPU - main thread, module SX32W
10005B30 RNBOsproFindFirstUnit
10005B33 83EC 48 SUB ESP,48
10005B3A 66:C74424 02 MOV WORD PTR SS:[LOCAL.17+2],0
10005B38 53 PUSH EBX
10005B38 56 PUSH ESI
10005B3C 8B5C24 54 MOV EBX,DWORD PTR SS:[ARG.1]
10005B40 57 PUSH EDI
10005B41 55 PUSH EBP
10005B42 85DB TEST EBX,EBX
10005B44 0F84 8B020000 JE 10005D05
10005B4A 66:8B6C24 60 MOV BP,WORD PTR SS:[ARG.2]
10005B4F 66:85ED TEST BP,BP
10005B52 0F84 7D020000 JE 10005D05
10005B58 66:81FD FFFF CMP BP,0FFFF
10005B5D 90 NOP
10005B5E 90 NOP
10005B5F 66:B8 0000 MOV AX,0
10005B63 5D POP EBP
10005B64 5F POP EDI
10005B65 5E POP ESI
10005B66 5B POP EBX
10005B67 83C4 48 ADD ESP,48
10005B6A C2 0800 SETN B
10005B6D E3 BCU EBX

```

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

Vamos a por la siguiente.

### .\* Primera llamada 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:

```

10005E60 RNBOsproRead
10005E61 56 PUSH ESI
10005E62 57 PUSH EDI
10005E66 8B7C24 0C MOV EDI,DWORD PTR SS:[ARG.1]
10005E68 85FF TEST EDI,EDI
10005E6A 75 09 JNE SHORT 10005E73
10005E6E 66:B8 1000 MOV AX,10
10005E6F 5F POP EDI
10005E70 5E POP ESI
10005E73 C2 0C00 RETN 0C
10005E74 57 PUSH EDI
10005E77 E8 27940000 CALL 1000F2A0
10005E79 8BF0 MOV ESI,EAX

```

Y en la pila:

```

0012F81C 004EBC99 8#N. RETURN to METSIM.004EBC99
0012F820 04CBFDB0 ASCII "
0012F824 00000008 ...
0012F828 04CC418C IAIF

```

Primero vemos qué significa cada argumento de la función RNBOsproRead:

## Format

\*\*\*\*\* RNBOsproRead

RB\_SPRO\_APIPACKET packet

\*\*\*\*\*

\*\*\*\*\*

..

## 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.

En este caso va a escribir en 4CC418C (esta dirección cambia con cada ejecución del programa) el Word contenido en la celda 08 de la mochila.  
Nosotros de momento lo escribiremos a mano y cambiaremos la rutina para que al llegar al ret de la función también AX valga 0. Ya al final modificaremos la función para emular todas las llamadas a RNBOsproRead que haga el programa:

```

10005E60 RNBOsproRead 56 PUSH ESI
10005E61 57 PUSH EDI
10005E62 8B7C24 0C MOV EDI,DWORD PTR SS:[ARG.1]
10005E66 85FF TEST EDI,EDI
10005E68 90 NOP
10005E69 90 NOP
10005E6A 66:B8 0000 MOV AX,0
10005E6E 5F POP EDI
10005E6F 5E POP ESI
10005E70 C2 0C00 RETN 0C
10005E73 > 57 PUSH EDI
10005E74 F8 27940000 JLT 1000F200

```

Address	Hex dump
04CC418C	08 08 00 00 18 00 00 00
04CC4194	50 00 00 00 01 00 00 00
04CC419C	B4 42 0A 04 0A 0A 0A 0A

Lo que hemos hecho es poner un valor cualquiera, y le pondremos un BPM a esa dirección de memoria. Si damos a F9 el programa para 3 veces para copiar el contenido de esa celda en diferentes posiciones de memoria, a las que iremos poniendo BPM también. La primera vez para en 4EBD36, la segunda en 4FF0AC y la tercera en 509F97. La cuarta vez que para lo hace en 4FEFF0:

```

004FEFF0 > 58B04B8 MOV EAX,DWORD PTR DS:[EDI*4+EAX]
004FEFF3 56 PUSH ESI
004FEFF4 8945 00 MOV DWORD PTR SS:[EBP],EAX
004FEFF7 E8 44E2FEFF CALL 004ED240
004FEFFC 83C4 04 ADD ESP,4
004FEFFF 5F POP EDI
004FF000 5E POP ESI
004FF001 5D POP EBP
004FF002 5B POP EBX
004FF003 C3 RETN

```

[04CC174C]=00000008 (decimal 2056.)  
EAX=04CC1748  
Jump from 4FEFB6

Y cuando llega a 4FEFF4 copia el contenido en 4CC34D8, y ahí ponemos un BPM y podemos quitar todos los demás:

Address	Hex dump	ASCII
04CC34D8	08 08 00 00 18 00 00 00	..↑...
04CC34E0	18 00 00 00 00 00 00 00	↑.....
04CC34E8	00 00 00 00 01 00 00 00	....0...

Memory breakpoints				
Address	Size	Module	Type	Status
04CC174C	00000002	<none>	RW	Disabled
04CC1754	00000002	<none>	RW	Disabled
04CC34D8	00000002	<none>	RW	Active
04CC418C	00000002	<none>	RW	Disabled

### .\* Segunda llamada a RNBOsproRead \*.\*

Esta vez va a leer el contenido de la celda 1C y lo guardará en 4CC560C:

```

0012F81C 004EBC99 0#N. RETURN to METSIM.004EBC99
0012F820 04CBFD80 ASCII "
0012F824 0000001C L...
0012F828 04CC560C .V|...

```

Escribimos a mano el valor 1C1C en esa posición de memoria y le pondremos un BPM. Damos a F9 y parará también 3 veces antes de llegar de nuevo a 4FEFF4. Esas veces que para por lectura pondremos BPM en la nueva posición donde escribe los datos y daremos a continuar hasta llegar a 4FEFF4, ahí escribirá 1C1C en 4CC3F70:



Address	Hex dump	ASCII
04CC3F70	1C 1C 00 00 18 00 00 00	..↑...
04CC3F78	7C 00 00 00 00 00 00 00	!.....
04CC3F80	20 4A CC 04 00 00 00 00	Jf.....

Memory breakpoints				
Address	Size	Module	Type	Status
04CC174C	00000002	<none>	RW	Disabled
04CC1754	00000002	<none>	RW	Disabled
04CC24F4	00000002	<none>	RW	Disabled
04CC24FC	00000002	<none>	RW	Disabled
04CC34D8	00000002	<none>	RW	Active
04CC3F70	00000002	<none>	RW	Active
04CC418C	00000002	<none>	RW	Disabled
04CC560C	00000002	<none>	RW	Disabled

Ponemos un BPM y desactivamos los otros BPMs dejando sólo los de la imagen.

**.-\*\* Tercera llamada a RNBOsproRead \*\*.-**

Cuando damos a F9 para por tercera vez en RNBOsproRead, esta vez para leer el contenido de la celda 30 y guardarlo en 4CC338C. Nosotros escribiremos 3030:

0012F81C	004EBC99	0#N.	RETURN to METSIM.004EBC99
0012F820	04CBFDB0	ASCII 20,"	
0012F824	00000030	0...	
0012F828	04CC338C	13f	

Procedemos igual que antes hasta llegar de nuevo a 4FEFF4 donde guarda los valores en 4CC5668:

Address	Hex dump	ASCII
04CC5668	30 30 00 00 18 00 00 00	00..↑...
04CC5670	28 00 00 00 00 00 00 00	(.....
04CC5678	A4 1A CC 04 91 50 54 4D	ã+fzPTM

Memory breakpoints				
Address	Size	Module	Type	Status
04CC174C	00000002	<none>	RW	Disabled
04CC1754	00000002	<none>	RW	Disabled
04CC24F4	00000002	<none>	RW	Disabled
04CC24FC	00000002	<none>	RW	Disabled
04CC338C	00000002	<none>	RW	Disabled
04CC34D8	00000002	<none>	RW	Active
04CC3F70	00000002	<none>	RW	Active
04CC418C	00000002	<none>	RW	Disabled
04CC560C	00000002	<none>	RW	Disabled
04CC5668	00000002	<none>	RW	Active

Y ponemos un BPM ahí y dejamos sólo lo que vemos en la imagen de arriba.

**.-\*\* Cuarta y última llamada a RNBOsproRead \*\*.-**

Después de dar a F9 llegamos de nuevo a RNBOsproRead en la que lee el contenido de la celda 00. Sabemos que esa celda no es modificable, ya que contiene el número de serie de la pastilla, así que supongo que no hará gran cosa con lo que lea de ahí. Sin embargo vamos a hacer el seguimiento para ver qué hace:

0012F81C	004EBC99	0#N.	RETURN to METSIM.004EBC99
0012F820	04CBFDB0	ASCII 20,"	
0012F824	00000000	...	
0012F828	04CC3F2C	,?f	

Esta vez escribiremos a mano el valor 1234 en 4CC3F2C y le ponemos como siempre un BPM. Vamos dando F9 y cada vez que pare por lectura ponemos BPM en la nueva dirección donde copia el valor hasta llegar de nuevo a 4FEFF4 en donde lo escribe en 4CC248C, donde pondremos un BPM y dejaremos sólo los que vemos:

Address	Hex dump	ASCII
04CC248C	34 12 00 00 18 00 00 00	4...↑...
04CC2494	48 00 00 00 00 00 00 00	H.....
04CC249C	54 32 CC 04 FF FF FF FF	T2If♦

Memory breakpoints					
Address	Size	Module	Type	Status	
04CC174C	00000002	<none>	RW	Disabled	
04CC1754	00000002	<none>	RW	Disabled	
04CC248C	00000002	<none>	RW	Active	
04CC24F4	00000002	<none>	RW	Disabled	
04CC24FC	00000002	<none>	RW	Disabled	
04CC338C	00000002	<none>	RW	Disabled	
04CC34D8	00000002	<none>	RW	Active	
04CC3F2C	00000002	<none>	RW	Disabled	
04CC3F70	00000002	<none>	RW	Active	
04CC418C	00000002	<none>	RW	Disabled	
04CC560C	00000002	<none>	RW	Disabled	
04CC5668	00000002	<none>	RW	Active	

**.\*\* Manipulando bytes: que hace con la celda 1C? \*\*.**

Parece que ya el programa tiene todos los datos que le hacen falta de la mochila y ahora empezará a trabajar con ellos.

Si pulsamos F9 para en 556D15 donde le suma 3E8h (1000d) al valor que había en la celda 00 y la guarda en EAX (1234 + 3E8= 161C):

00556D13	> 8B06	MOV EAX, DWORD PTR DS:[ESI]
00556D15	• 0303	ADD EAX, DWORD PTR DS:[EBX]
00556D17	• 70 0A	J0 SHORT 00556D23
00556D19	• AB	STOS DWORD PTR ES:[EDI]
00556D1A	• 03F2	ADD ESI, EDI
00556D1C	• 03D0	ADD EBX, EBP
00556D1E	• E2 F3	LOOP SHORT 00556D13
00556D20	• 5D	POP EBP
00556D21	• EB 0C	JMP SHORT 00556D2F

[04CC248C]=00001234 (decimal 4660.)  
EAX=00003E8 (decimal 1000.)  
Loop 00556D13: loop variable ECX(-1)

Y en 556D19 guarda el resultado de la suma en 4CC3F2C. Ahí ponemos otro BPM:

Address	Hex dump
04CC3F2C	1C 16 00 00 18 00 00 00
04CC3F34	2C 00 00 00 01 00 00 00
04CC3F3C	00 00 0B 00 1A 00 00 00

Si seguimos con F9 para en 4FDEB6 donde sobrescribe el valor que teníamos guardado en 4CC248C así que podemos quitar ese BPM:

004FDEB2	• 8B5424 14	MOV EDX, DWORD PTR SS:[ARG.3]
004FDEB6	• 8902	MOV DWORD PTR DS:[EDX], EAX
004FDEB8	• B8 01000000	MOV EAX, 1
004FDEBD	• 83C4 08	ADD ESP, 8
004FDEC0	• C3	RET
004FDEC1	> 33C0	XOR EAX, EAX
004FDEC3	• 83C4 08	ADD ESP, 8
004FDEC6	• C3	RET

Imm=1  
EAX=00003E8 (decimal 1000.)

Address	Hex dump	ASCII
04CC248C	E8 03 00 00 18 00 00 00	8...↑...
04CC2494	48 00 00 00 00 00 00 00	H.....
04CC249C	54 32 CC 04 FF FF FF FF	T2If♦

Memory breakpoints					
Address	Size	Module	Type	Status	Comment
04CC248C	00000002	<none>	RW	Disabled	
04CC34D8	00000002	<none>	RW	Active	
04CC3F2C	00000002	<none>	RW	Active	
04CC3F70	00000002	<none>	RW	Active	
04CC5668	00000002	<none>	RW	Active	

Si continuamos con F9 para de nuevo en 556D15 donde le suma 3E8h (1000d) al valor que había en la celda 1C (1C1C + 3E8 = 2004), este valor lo guarda en EAX y en 556D19 guarda el resultado de la suma en 4CC34F0. Ahí ponemos otro BPM:

```

00556D15 | . 0303      | ADD EAX,DWORD PTR DS:[EBX]
00556D17 | . 70 0A     | J0 SHORT 00556D23
00556D19 | . AB       | STOS DWORD PTR ES:[EDI]
00556D1A | . 03F2     | ADD ESI,EDX
00556D1C | . 03D0     | ADD EBX,EBP
00556D1E | . ^ E2 F3   | LOOP SHORT 00556D13
00556D20 | . 5D       | POP EBP

```

EDX=0  
ESI=04CC248C  
Loop 00556D13: loop variable ECX(-1)

Address	Hex dump	ASCII
04CC34F0	04 20 00 00 18 00 00 00	...↑...
04CC34F8	4C 00 00 00 00 00 00 00	L.....
04CC3500	D0 35 CC 04 FF FF FF FF	\$5!♦

### Memory breakpoints

Address	Size	Module	Type	Status	Comment
04CC248C	00000002	<none>	RW	Disabled	
04CC34D8	00000002	<none>	RW	Active	
04CC34F0	00000002	<none>	RW	Active	
04CC3F2C	00000002	<none>	RW	Active	
04CC3F70	00000002	<none>	RW	Active	
04CC5668	00000002	<none>	RW	Active	

Si damos a continuar la siguiente vez que para Olly lo hace en 4E9BC0 donde comprueba si el valor almacenado en 4CC34F0 (que es donde se guardó el resultado de la suma 3E8 + 1C1C = 2004) es distinto de 0.

```

004E9BC0 | > 8B02      | MOV EAX,DWORD PTR DS:[EDX]
004E9BC2 | . 03D6     | ADD EDX,ESI
004E9BC4 | . 3BC1     | CMP EAX,ECX
004E9BC6 | . 7D 02     | JGE SHORT 004E9BCA
004E9BC8 | > 8BC8     | MOV ECX,EAX
004E9BCA | > 3BC7     | CMP EAX,EDI
004E9BCC | . 7E 02     | JLE SHORT 004E9BD0
004E9BCE | > 8BF8     | MOV EDI,EAX
004E9BD0 | > 4B       | DEC EBX
004E9BD1 | . ^ 75 ED   | JNE SHORT 004E9BC0
004E9BD3 | > 68 C0505900 | PUSH OFFSET METSIM.005950C0
004E9BD8 | . 51       | PUSH ECX
004E9BD9 | . E8 D2ED0000 | CALL 004F89B0
004E9BDE | . 83C4 08   | ADD ESP,8
004E9BE1 | . 8BF0     | MOV ESI,EAX
004E9BE3 | . 68 C0505900 | PUSH OFFSET METSIM.005950C0
004E9BE8 | . 57       | PUSH EDI
004E9BE9 | . E8 C2ED0000 | CALL 004F89B0
004E9BEE | . 83C4 08   | ADD ESP,8
004E9BF1 | . 3BF0     | CMP ESI,EAX
004E9BF3 | . 7E 02     | JLE SHORT 004E9BF7
004E9BF5 | > 8BC6     | MOV EAX,ESI
004E9BF7 | > 5F       | POP EDI
004E9BF8 | . 5E       | POP ESI
004E9BF9 | . 5B       | POP EBX
004E9BFA | . C3       | RETN

```

Si es así va a 4E9BD9 donde si entramos con F7 en el CALL 4F89B0 vemos que esta rutina transforma ese valor hexadecimal en decimal (2004h = 8196d), pero de una manera curiosa, ya que lo deja en orden invertido, o sea lo transforma en 6918. Este valor está guardado en 5950C0 cuando llegamos al RET de esta función, entonces ponemos en 5950C0 otro BPM:

004F8A16	> 85C9	TEST ECX,ECX
004F8A18	7E 2A	JLE SHORT 004F8A44
004F8A1A	B8 67666666	MOV EAX,66666667
004F8A1F	F7E9	IMUL ECX
004F8A21	C1FA 02	SAR EDX,2
004F8A24	8BC2	MOV EAX,EDX
004F8A26	C1E8 1F	SHR EAX,1F
004F8A29	03D0	ADD EDX,EAX
004F8A2B	8AC2	MOV AL,DL
004F8A2D	C0E0 03	SHL AL,3
004F8A30	2AC8	SUB CL,AL
004F8A32	8AC2	MOV AL,DL
004F8A34	D0E0	SHL AL,1
004F8A36	2AC8	SUB CL,AL
004F8A38	80C1 30	ADD CL,30
004F8A3B	880E	MOV BYTE PTR DS:[ESI],CL
004F8A3D	46	INC ESI
004F8A3E	85D2	TEST EDX,EDX
004F8A40	8BCA	MOV ECX,EDX
004F8A42	7F D6	JG SHORT 004F8A1A
004F8A44	8A4424 10	MOV AL,BYTE PTR SS:[ARG.2]
004F8A48	84C0	TEST AL,AL
004F8A4A	74 04	JE SHORT 004F8A50
004F8A4C	C606 FD	MOV BYTE PTR DS:[ESI],0FD
004F8A4F	46	INC ESI
004F8A50	8BC6	MOV EAX,ESI
004F8A52	2BC7	SUB EAX,EDI
004F8A54	5F	POP EDI
004F8A55	5E	POP ESI
004F8A56	C3	RET

Address	Hex dump	ASCII
005950C0	36 39 31 38 00 00 00 00	6918....
005950C8	00 00 00 00 00 00 00 00	.....
005950D0	00 00 00 00 00 00 00 00	.....

Al continuar para de nuevo en 4E994F por el BPM que tenemos en 5950C0, lo guarda en EAX, lo empuja a la pila:

004E9945	> 3B1D BC505900	CMP EBX,DWORD PTR DS:[5950BC]
004E9948	75 02	JNE SHORT 004E994F
004E994D	330B	XOR EBX,EBX
004E994F	> 8B45 00	MOV EAX,DWORD PTR SS:[EBP]
004E9952	68 C0505900	PUSH OFFSET METSIM.005950C0
004E9957	50	PUSH EAX
004E9958	83C5 04	ADD EBP,4
004E995B	E8 50F00000	CALL 004F89B0
004E9960	8B5424 24	MOV EDX,DWORD PTR SS:[ARG.3]
004E9964	8BCB	MOV ECX,EBX
004E9966	C1E1 03	SHL ECX,3

[04CC34F0]=00002004 (decimal 8196.)  
EAX=1  
Jump from 4E994B  
Loop 004E9945: loop variable EBP(+4)

Address	Hex dump	ASCII
005950C0	36 39 31 38 00 00 00 00	6918....
005950C8	00 00 00 00 00 00 00 00	.....
005950D0	00 00 00 00 00 00 00 00	.....
005950D8	00 00 00 00 00 00 00 00	.....

Así que vamos traceando con cuidado con F8 hasta que llegamos a:

004E99BD	81FE C0505900	CMP ESI,OFFSET METSIM.005950C0	ASCII "6918"
004E99C3	72 16	JB SHORT 004E99DB	
004E99C5	0FB006	MOVSX EAX,BYTE PTR DS:[ESI]	
004E99C8	50	PUSH EAX	
004E99C9	FF15 7C675900	CALL DWORD PTR DS:[59677C]	
004E99CF	83C4 04	ADD ESP,4	
004E99D2	4E	DEC ESI	
004E99D3	81FE C0505900	CMP ESI,OFFSET METSIM.005950C0	ASCII "6918"
004E99D9	73 EA	JNB SHORT 004E99C5	
004E99DB	> 8B4424 18	MOV EAX,DWORD PTR SS:[ESP+18]	
004E99DF	48	DEC EAX	
004E99E0	894424 18	MOV DWORD PTR SS:[ESP+18],EAX	
004E99E4	0F85 5BFFFFFF	JNE 004E9945	

Ahí entramos con F7 y esta rutina le da la vuelta al número 6918, dejándolo en 8169, que ahora ya es  $1C1C + 3E8 = 2010h = 8169$  y lo deja escrito en 4CA1CF4:

004E9600	A1 B0505900	MOV EAX,DWORD PTR DS:[5950B0]
004E9605	8A4C24 04	MOV CL,BYTE PTR SS:[ARG.1]
004E9609	8808	MOV BYTE PTR DS:[EAX],CL
004E960B	A1 B0505900	MOV EAX,DWORD PTR DS:[5950B0]
004E9610	40	INC EAX
004E9611	A3 B0505900	MOV DWORD PTR DS:[5950B0],EAX
004E9616	C3	RET

Y ahí le ponemos otro BPM:

Address	Hex dump	ASCII
04CA1CF4	38 31 39 36 1C 00 00 00	8196L...
04CA1CFC	50 00 00 00 00 00 00 00	P.....
04CA1D04	68 1C CA 04 1F 00 00 00	hL...?

Memory breakpoints				
Address	Size	Module	Type	Status
005950C0	00000004	METSIM	RW	Active
04CA1CF4	00000004	<none>	RW	Active
04CC248C	00000002	<none>	RW	Disabled
04CC34D8	00000002	<none>	RW	Active
04CC34F0	00000002	<none>	RW	Active
04CC3F2C	00000002	<none>	RW	Active
04CC3F70	00000002	<none>	RW	Active
04CC5668	00000002	<none>	RW	Active

Ahora damos a continuar de nuevo y paramos en 556C9E donde copia las 3 últimas cifras del número 8196 en 4CBFBF0:

```

00556C93 | 8B32      MOV ESI, DWORD PTR DS:[EDX]
00556C95 | 8B7A 04   MOV EDI, DWORD PTR DS:[EDX+4]
00556C98 | F642 0E 01 TEST BYTE PTR DS:[EDX+0E], 01
00556C9C | 75 04     JNE SHORT 00556CA2
00556C9E | F3:A4     REP MOVSB BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
00556CA0 | EB 05     JMP SHORT 00556CA7
00556CA2 | 8A06     MOV AL, BYTE PTR DS:[ESI]
00556CA4 | AA       STOS BYTE PTR ES:[EDI]

```

ECX=3  
[04CA1CF5]=31 ('1')  
[04CBFBF0]=07

Así que ahí también ponemos otro BPM para ver qué hace con eso:

Address	Hex dump	ASCII
04CBFBF0	31 39 36 00 1C 00 00 00	196L...
04CBFBF8	A4 01 00 00 00 00 00 00	?
04CBFC00	00 00 00 00 00 00 00 00	.....

Volvemos a pulsar F9 y paramos en 5260ED donde pone en EAX el valor que habíamos guardado de la celda 08 (0808) y después compara si el valor que tenemos está entre 1 y 1A.

CPU - main thread, module METSIM				
005260E3	894424 38	MOV DWORD PTR SS:[LOCAL.0], EAX		
005260E7	0F83 C7000000	JNB 005261B4		
005260ED	8B03	MOV EAX, DWORD PTR DS:[EBX]		
005260EF	8B4C24 30	MOV ECX, DWORD PTR SS:[LOCAL.2]		
005260F3	83C3 04	ADD EBX, 4		
005260F6	3BC1	CMP EAX, ECX		
005260F8	0F8F 12010000	JG 00526210		
005260FE	3BC2	CMP EAX, EDX		
00526100	0F8C 0A010000	JL 00526210		
00526106	8B4C24 34	MOV ECX, DWORD PTR SS:[LOCAL.1]		
0052610A	03C5	ADD EAX, EBP		

ECX=0000001A (decimal 26.)  
EAX=00000808 (decimal 2056.)

Registers (FPU)	
EAX	00000808
ECX	0000001A
EDX	00000001
EBX	04CC34DC
ESP	0012F87C
EBP	00000000
ESI	04CC34F0
EDI	050EAE27
EIP	005260F6 ME1
C	0 ES 0023 32t
P	0 CS 001B 32t

En nuestro caso la comparación nos lleva a tomar el salto a 526210.

Nos falta saber si la condición correcta es que el valor debe estar entre 1 y 1A, o debe ser  $\leq 0$  y  $\geq 1A$ . Como es imposible que con 2 bytes hexa se represente un número negativo y no tiene sentido que el valor sea 0, porque ese es el valor que se leería en caso de no haber mochila puesta, vamos a suponer que la condición a cumplir es que el resultado de leer la celda 8 debe cumplir estar entre 1 y 1A.

**.\* Manipulando bytes: que hay en la celda 08? \*.**

Parece que tenemos la primera pista, lo que vamos a hacer es reiniciar todo el proceso desde el principio y poner como valor de retorno de RNBOsproRead de la celda 08 el valor 14. Así que cuando llegamos a 5260F6 cumplimos las 2 comparaciones:

```

005260ED > 8B03 MOV EAX,DWORD PTR DS:[EBX]
005260EF . 8B4C24 30 MOV ECX,DWORD PTR SS:[LOCAL.2]
005260F3 . 83C3 04 ADD EBX,4
005260F6 . 3BC1 CMP EAX,ECX
005260F8 . 0F8F 12010000 JG 00526210
005260FE . 3BC2 CMP EAX,EDX
00526100 . 0F8C 0A010000 JL 00526210
00526106 . 8B4C24 34 MOV ECX,DWORD PTR SS:[LOCAL.1]
0052610A . 03C5 ADD EAX,EBP
0052610C . 03C5 ADD EAX,EBP
ECX=0000001A (decimal 26.)
EAX=00000014 (decimal 20.)

```

y entonces seguimos traceando con F8 ya que el valor está en EAX y queremos saber qué hace con él, así que llegamos hasta 52611C, donde pone en CL el contenido de EAX+EDI.

```

0052611C > 8A0C07 MOV CL,BYTE PTR DS:[EAX+EDI]
0052611F . 880E MOV BYTE PTR DS:[ESI],CL
00526121 . 46 INC ESI
00526122 . E9 83000000 JMP 005261AA

```

En EDI tenemos el valor 50EAE27 dirección a partir de la cual tenemos el abecedario en mayúsculas

Address	Hex dump	ASCII
050EAE27	00 41 42 43 44 45 46 47	.,ABCDEFG
050EAE2F	48 49 4A 4B 4C 4D 4E 4F	HIJKLMNO
050EAE37	50 51 52 53 54 55 56 57	PQRSTUVWXYZ
050EAE3F	58 59 5A 00 00 34 00 00	XYZ..4..
050EAE47	00 18 00 00 00 04 00 00	.f...♦..

Como en EAX tenemos guardado el valor 14 resultante de leer la celda 08, lo que está haciendo es transformar ese número en una letra mayúscula en este caso en "T", que en la línea 52611F guarda en la dirección 4CC34F0, y ahí le ponemos un BPM (ya había un BPM puesto ahí antes):

Address	Hex dump	ASCII
04CC34F0	54 20 00 00 18 00 00 00	T ..†...
04CC34F8	4C 00 00 00 00 00 00 00	.....

Si damos a continuar llegamos hasta 556CA4:

```

00556CA2 > 8A06 MOV AL,BYTE PTR DS:[ESI]
00556CA4 . AA STOS BYTE PTR ES:[EDI]
00556CA5 . ^ E2 FB LOOP SHORT 00556CA2
00556CA7 > 8932 MOV DWORD PTR DS:[EDX],ESI
00556CA9 . 897A 04 MOV DWORD PTR DS:[EDX+4],EDI
00556CAC > 5E POP ESI
AL=54 ('T')
[04CA1CF4]=38 ('8')
Loop 00556CA2: loop variable ECX(-1)

```

Donde copia esa letra "T" que teníamos guardada en donde teníamos guardado el número 8196, formando una cadena alfanumérica con esa letra y las 3 últimas cifras del número, o sea T196:

Address	Hex dump	ASCII
04CA1CF4	54 31 39 36 1C 00 00 00	T196L...
04CA1CFC	50 00 00 00 00 00 00 00	P.....
04CA1D04	68 1C CA 04 1F 00 00 00	hL...♦...

Como el BPM ya está puesto de antes pulsamos F9 para continuar, y paramos en 556C9E donde copia esta cadena alfanumérica en 4CA1CF4:

```

00556C9E . F3:A4 REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00556CA0 . EB 05 JMP SHORT 00556CA7
00556CA2 > 8A06 MOV AL,BYTE PTR DS:[ESI]
00556CA4 . AA STOS BYTE PTR ES:[EDI]
Dest=METSIM.00556CA7

```

Address	Hex dump	ASCII
04CA1CED	00 00 00 04 00 00 00 54	.....T
04CA1CF5	31 39 36 1C 00 00 00 50	196L...P

Como allí ya había un BPM continuamos con F9 hasta que para en 522D9D, ahí guarda el el primer carácter de la cadena alfanumérica y si seguimos con F8 hasta 522DB1 guarda ese carácter en 4CBFBF0:

00522D9D	• 8A1C08	MOV BL, BYTE PTR DS:[ECX+EBX]
00522DA0	• 0F87 5F030000	JA 00523105
00522DA6	• FF24BD 24315	JMP DWORD PTR DS:[EDI*4+523124]
00522DAD	> 8B5424 1C	MOV EDX, DWORD PTR SS:[ARG_2]
00522DB1	• 8B1C16	MOV BYTE PTR DS:[EDX+ESI], BL
00522DB4	• 5F	POP EDI
00522DB5	• 5E	POP ESI
00522DB6	• 5B	POP EBX
00522DB7	• 83C4 08	ADD ESP, 8
00522DBA	• C3	RET

Address	Hex dump	ASCII
04CBFBF0	54 31 39 36 00 1C 00 00 00	T196L...
04CBFBF8	A4 01 00 00 00 00 00 00	ñ0.....

Pues parará 3 veces más aquí de manera que al final tendremos en 4CBFCF0 copiada la cadena alfanumérica completa:

Address	Hex dump	ASCII
04CBFBF0	54 31 39 36 1C 00 00 00	T196L...
04CBFBF8	A4 01 00 00 00 00 00 00	ñ0.....
04CBFC00	00 00 00 00 00 00 00 00	.....

Después de poner un BPM a la cadena completa volvemos a pulsar F9 parará 2 veces para escribir en 4CC34F0 y después en 1CA1CF4, por lo tanto podemos deshabilitar esos 2 BPM:

Memory breakpoints				
Address	Size	Module	Type	Status
005950C0	00000004	METSIM	Rw	Active
04CA1CF4	00000004	<none>	Rw	Disabled
04CBFBF0	00000004	<none>	Rw	Active
04CC248C	00000002	<none>	Rw	Disabled
04CC34D8	00000002	<none>	Rw	Active
04CC34F0	00000001	<none>	Rw	Disabled
04CC3F2C	00000002	<none>	Rw	Active
04CC3F70	00000002	<none>	Rw	Active
04CC5668	00000002	<none>	Rw	Active

Damos F9 para continuar de nuevo y para en 556C9E para copiar la cadena alfanumérica desde 4CBFBF0 a 4CC22BC, donde pondremos un nuevo BPM:

00556C9E	• F3:A4	REP MOVS BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
00556CA0	• EB 05	JMP SHORT 00556CA7
00556CA2	> 8A06	MOV AL, BYTE PTR DS:[ESI]
00556CA4	• AA	STOS BYTE PTR ES:[EDI]

ECX=1  
[04CBFBF3]=36 ('6')  
[04CC22BF]=00

Address	Hex dump	ASCII
04CC22BC	54 31 39 36 1C 00 00 00	T196L...
04CC22C4	20 00 00 00 00 00 00 00	.....
04CC22CC	24 48 CC 04 05 00 00 00	\$Hf+...\$

Address	Hex dump	ASCII
04CC22BC	54 31 39 36 1C 00 00 00	T196L...
04CC22C4	20 00 00 00 00 00 00 00	.....
04CC22CC	24 48 CC 04 05 00 00 00	\$Hf+...\$

Y ahora si pulsamos F9 llegamos por fin al objetivo que perseguimos. Paramos en 5212C4 donde estamos en medio de un gran bucle en el que se va comparando nuestra cadena alfanumérica con las que hay en una tabla enorme que se encuentra a partir de la dirección 4C9E1A8:



```

005212C2 | > 8A17 | MOV DL, BYTE PTR DS:[EDI]
005212C4 | * 3A16 | CMP DL, BYTE PTR DS:[ESI]
005212C6 | * 75 0F | JNE SHORT 005212D7
005212C8 | * 8B5424 24 | MOV EDX, DWORD PTR SS:[ARG_3]
005212CC | * 03FA | ADD EDI, EDX
005212CE | * 8B5424 1C | MOV EDX, DWORD PTR SS:[ARG_1]
005212D0 | * 03F2 | ADD ESI, EDX
[04CC22BC]=54 ('T')
DL=50 ('P')
Loop 005212B8: loop variable ESI(+1)
Loop 005212C2: loop variable EAX(-1)

```

Address	Hex dump	ASCII
04C9E1A8	50 30 30 32 4A 31 37 37 4A 30 31 38 4A 30 31 39	P002J177J018J019
04C9E1B8	50 30 31 31 4A 30 32 31 50 30 30 34 50 30 32 33	P011J021P004P023
04C9E1C8	54 37 30 37 54 38 30 38 4A 31 35 31 4B 30 30 37	T707T808J151K007
04C9E1D8	4A 31 36 38 50 30 30 36 50 30 30 37 4B 30 32 32	J168P006P007K022
04C9E1E8	50 30 30 38 50 30 33 32 50 30 30 39 50 30 31 30	P008P032P009P010
04C9E1F8	50 30 33 31 4B 30 34 36 4B 30 30 39 4A 30 32 33	P031K046K009J023
04C9E208	50 30 33 30 50 30 34 30 4A 31 36 37 4A 30 39 30	P030P040J167J090
04C9E218	50 30 31 34 4B 30 31 31 4A 31 37 30 4A 31 37 36	P014K011J170J176
04C9E228	4A 31 36 35 4A 31 36 34 4B 30 35 34 4B 30 31 32	J165J164K054K012
04C9E238	4B 30 37 30 4A 30 32 35 4A 30 32 36 4A 30 32 37	K070J025J026J027
04C9E248	4A 30 32 38 4A 30 32 39 4A 30 33 30 4B 30 39 38	J028J029J030K098
04C9E258	50 30 34 32 54 30 38 39 54 30 39 30 54 30 39 33	P042T089T090T093
04C9E268	54 30 37 37 54 30 30 31 4A 31 36 32 50 30 33 35	T077T001J162P035
04C9E278	4A 31 37 34 4A 31 38 39 4A 31 37 33 4A 30 34 37	J174J189J173J047
04C9E288	54 30 30 32 54 30 35 39 4B 30 30 31 50 30 31 33	T002T059K001P013
04C9E298	54 30 36 35 54 30 30 33 4A 30 30 31 4A 30 30 32	T065T003J001J002
04C9E2A8	4A 31 35 35 4A 30 30 33 54 30 37 36 4A 30 30 34	J155J003T076J004
04C9E2B8	4B 30 35 38 4B 30 37 36 54 30 30 34 4B 30 30 32	K058K076T004K002
04C9E2C8	50 30 30 31 4A 31 36 31 4A 31 35 32 4A 30 30 35	P001J161J152J005
04C9E2D8	4B 30 38 32 4A 30 30 37 54 30 30 35 4B 30 30 33	K082J007T005K003
04C9E2E8	4A 31 38 31 4A 30 30 36 50 30 33 36 4A 30 34 35	J181J006P036J045
04C9E2F8	4A 31 39 31 4A 30 30 38 4B 30 30 34 4B 30 37 34	J191J008K004K074
04C9E308	4B 30 37 35 4A 30 31 30 4A 30 31 32 4B 30 35 31	K075J010J012K051
04C9E318	4A 30 30 39 54 30 30 36 4B 30 30 30 50 30 34 31	J009T006K080P041

Tiene toda la pinta de que nuestra cadena T196 debe coincidir con alguno de los existentes en esa tabla. Sin embargo y como es natural no hemos tenido suerte ☹, no está nuestra cadena en esa tabla.

### \*\*\* Apuntando a matar \*\*\*

Bueno pues antes de reiniciar vamos a elegir una cadena cualquiera de las de ahí arriba, por ejemplo, la primera P002.

Haciendo el proceso que hemos visto al revés, para que tengamos esa cadena cuando lleguemos al punto de comparación, debemos poner como respuesta a la celda 08 el valor 10, ya que al ser la letra P la que está en la decimosexta posición del abecedario. Transformando 16 d a hexadecimal tenemos 10 h.

Para obtener el 002d sólo tenemos que poner como respuesta a la celda 1C el valor 02h. Así ya podemos modificar la función RNBOsproRead para que haga esto automáticamente, además de mantener los valores de las celdas 00 y 30 con los que parece que no trabaja. El resultado es:

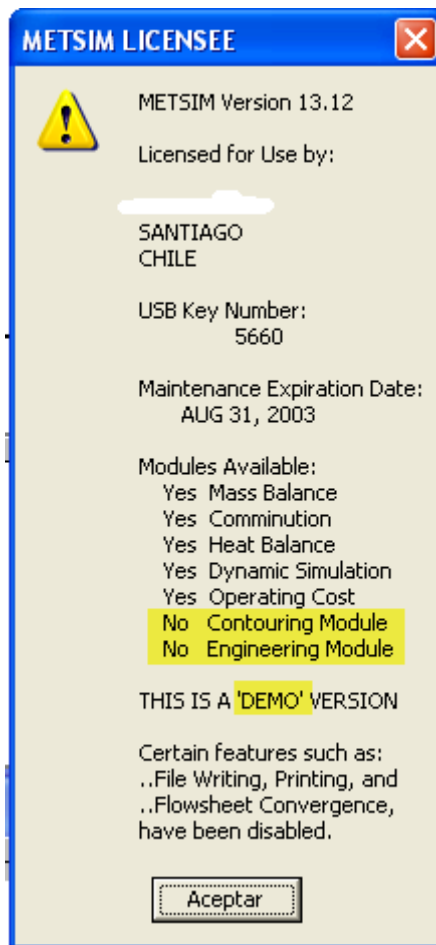
```

10005E60 | 56 | PUSH ESI
10005E61 | 57 | PUSH EDI
10005E62 | 837C24 10 08 | CMP DWORD PTR SS:[ARG_2],8
10005E67 | 0F85 0E000000 | JNE 10005E7B
10005E6D | 8B7C24 14 | MOV EDI, DWORD PTR SS:[ARG_3]
10005E71 | 66:C707 1000 | MOV WORD PTR DS:[EDI],10
10005E76 | E9 46000000 | JMP 10005EC1
10005E7B | 837C24 10 1C | CMP DWORD PTR SS:[ARG_2],10
10005E80 | 0F85 0E000000 | JNE 10005E94
10005E86 | 8B7C24 14 | MOV EDI, DWORD PTR SS:[ARG_3]
10005E8A | 66:C707 0200 | MOV WORD PTR DS:[EDI],2
10005E8F | E9 2D000000 | JMP 10005EC1
10005E94 | 837C24 10 30 | CMP DWORD PTR SS:[ARG_2],30
10005E99 | 0F85 0E000000 | JNE 10005EAD
10005E9F | 8B7C24 14 | MOV EDI, DWORD PTR SS:[ARG_3]
10005EA3 | 66:C707 8000 | MOV WORD PTR DS:[EDI],80
10005EA8 | E9 14000000 | JMP 10005EC1
10005EAD | 837C24 10 00 | CMP DWORD PTR SS:[ARG_2],0
10005EB2 | 0F85 09000000 | JNE 10005EC1
10005EB8 | 8B7C24 14 | MOV EDI, DWORD PTR SS:[ARG_3]
10005EBC | 66:C707 1234 | MOV WORD PTR DS:[EDI],1234
10005EC1 | 66:B8 0000 | MOV AX,0
10005EC5 | 5F | POP EDI
10005EC6 | 5E | POP ESI
10005EC7 | C2 0C00 | RETN 0C

```



Haciendo esta modificación y las que hemos visto antes para el resto de funciones y guardando los cambios en la dll, ejecutamos el programa y:



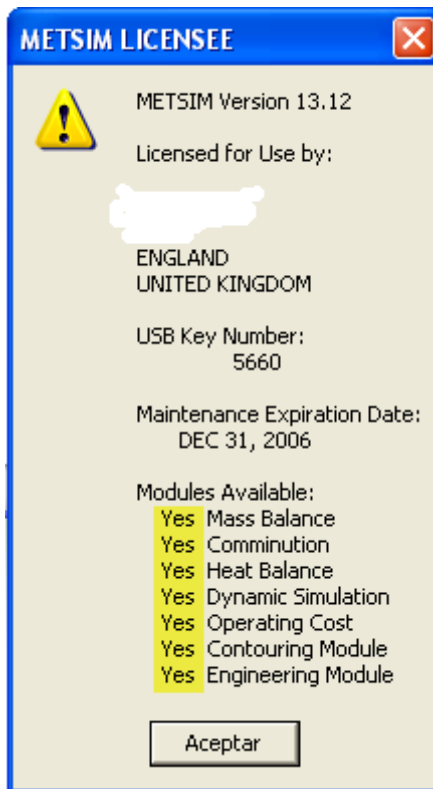
Ha desaparecido el cartel inicial de que no encuentra protección, sin embargo nos sigue diciendo que es una versión DEMO y además hay algunos módulos que no están disponibles.

Podemos ver que el USB Key Number es 5660 d que es 1234h, que es el valor que dimos como respuesta a la celda 00 del dongle. Y después de tracear mucho he visto que no hace nada relevante con la respuesta a la celda 30, por lo que lo único que se me ocurre es ir probando con distintos números de la tabla que vimos anteriormente.

Eso hice y una de las veces probé con T808, con lo que el injerto quedó:

10005E60	56	PUSH ESI	
10005E61	57	PUSH EDI	
10005E62	837C24 10 08	CMP DWORD PTR SS:[ARG.2],8	
10005E67	0F85 0E000000	JNE 10005E7B	
10005E6D	8B7C24 14	MOV EDI,DWORD PTR SS:[ARG.3]	
10005E71	66:C707 1400	MOV WORD PTR DS:[EDI],14	← Para obtener la T
10005E76	E9 46000000	JMP 10005EC1	
10005E7B	837C24 10 1C	CMP DWORD PTR SS:[ARG.2],1C	
10005E80	0F85 0E000000	JNE 10005E94	
10005E86	8B7C24 14	MOV EDI,DWORD PTR SS:[ARG.3]	
10005E8A	66:C707 2803	MOV WORD PTR DS:[EDI],828	← Para obtener el 808
10005E8F	E9 2D000000	JMP 10005EC1	
10005E94	837C24 10 30	CMP DWORD PTR SS:[ARG.2],30	
10005E99	0F85 0E000000	JNE 10005EAD	
10005E9F	8B7C24 14	MOV EDI,DWORD PTR SS:[ARG.3]	
10005EA3	66:C707 8000	MOV WORD PTR DS:[EDI],80	
10005EA8	E9 14000000	JMP 10005EC1	
10005EAD	837C24 10 00	CMP DWORD PTR SS:[ARG.2],0	
10005EB2	0F85 09000000	JNE 10005EC1	
10005EB8	8B7C24 14	MOV EDI,DWORD PTR SS:[ARG.3]	
10005EBC	66:C707 3412	MOV WORD PTR DS:[EDI],1234	
10005EC1	66:B8 0000	MOV AX,0	
10005EC5	5F	POP EDI	
10005EC6	5E	POP ESI	
10005EC7	C2 0C00	RET 0C	

Guardé los cambios y:



Ya lo tenemos todo a full. MISION CUMPLIDA!!

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

Hemos podido ver que aunque un poco enrevesado por la continua copia de valores arriba abajo, haciendo un seguimiento cuidadoso de todos los cambios es fácil deducir cuál debe ser el contenido de las celdas que se leen de la pastilla. El trabajo se ha hecho mucho más fácil gracias al nuevo Olly que permite colocar todos los BPM que quieras. También hubiera podido hacerse utilizando el Olly 1.10 pero al tener la limitación de 1 sólo BPM y los 4 HWBP nos hubiera costado bastante más.

En fin se ha visto lo que decíamos al principio, los desarrolladores confían toda la protección a la pastilla, y si ésta no se implanta bien es relativamente fácil romperla sin disponer de la original.

Es más como la protección está implantada en la dll y no han variado el esquema original, la dll modificada nos permite la ejecución incluso de las nuevas versiones del programa, todo un lujo.

Un saludo a todos y hasta el próximo