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	Lionel 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.

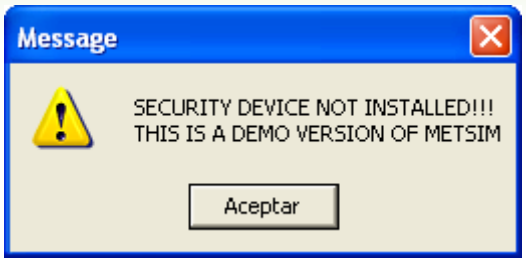
Este tutorial demuestra que los desarrolladores de software creen que porque se gasten un dineral en comprar una protección de este tipo, sus aplicaciones están seguras. El problema es que no ponen demasiado empeño en implantar este método de protección.

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.

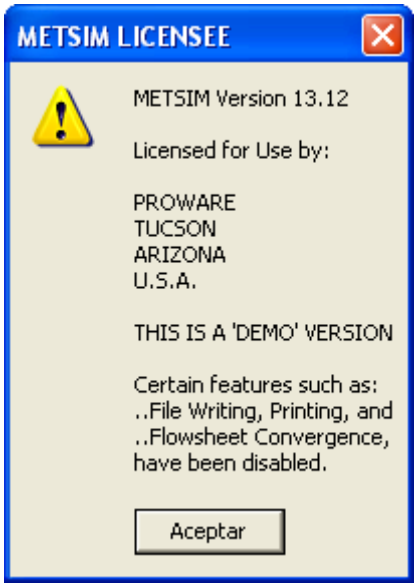
Lo que vamos a hacer es hacer una emulación para que el programa objetivo se comporte como si la dongle estuviera realmente puesta, y además como las funciones de llamada al dongle están en una dll, una vez modificada ésta, nos valdrá para todas las versiones del programa que han ido saliendo posteriormente.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



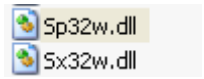
Cuando damos a aceptar nos aparece otra ventana que nos indica que estamos en una versión DEMO y que están deshabilitadas algunas opciones.



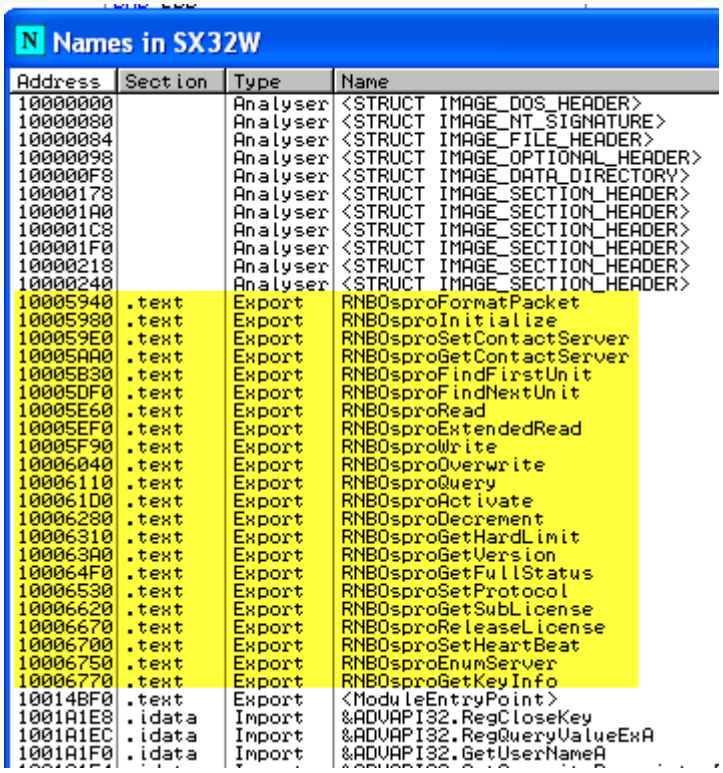
¿Cómo sabemos qué tipo de mochila es?. Pues tendremos que utilizar una herramienta de análisis muy potente y que en muy pocos casos se utiliza: la cabeza, jeje. En la página de descarga del programa aparece la opción de descargarnos los drivers de Sentinel, o sea, que está claro ¿no?.

Ahora lo mejor es cargar el programa en IDA y aplicarle el archivo de signaturas 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:



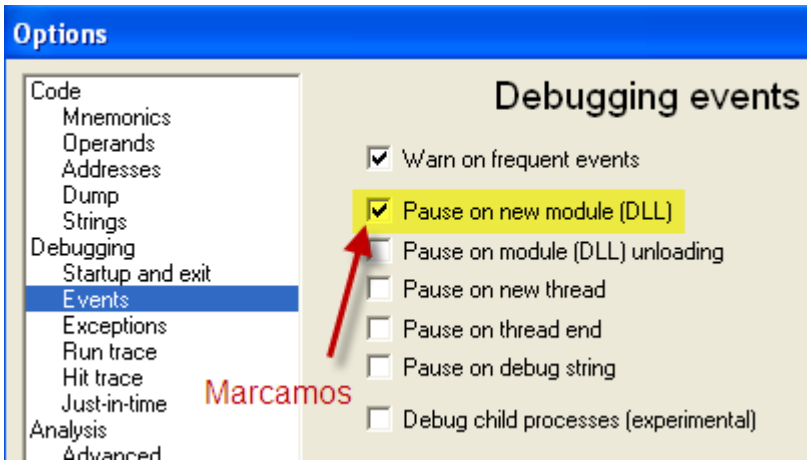
Names in SX32W

Address	Section	Type	Name
10000000		Analyser	<STRUCT IMAGE_DOS_HEADER>
10000080		Analyser	<STRUCT IMAGE_NT_SIGNATURE>
10000084		Analyser	<STRUCT IMAGE_FILE_HEADER>
10000098		Analyser	<STRUCT IMAGE_OPTIONAL_HEADER>
100000F8		Analyser	<STRUCT IMAGE_DATA_DIRECTORY>
10000178		Analyser	<STRUCT IMAGE_SECTION_HEADER>
100001A0		Analyser	<STRUCT IMAGE_SECTION_HEADER>
100001C8		Analyser	<STRUCT IMAGE_SECTION_HEADER>
100001F0		Analyser	<STRUCT IMAGE_SECTION_HEADER>
10000218		Analyser	<STRUCT IMAGE_SECTION_HEADER>
10000240		Analyser	<STRUCT IMAGE_SECTION_HEADER>
10005940	.text	Export	RNBOsproFormatPacket
10005980	.text	Export	RNBOsproInitialize
100059E0	.text	Export	RNBOsproSetContactServer
10005AA0	.text	Export	RNBOsproGetContactServer
10005B30	.text	Export	RNBOsproFindFirstUnit
10005DF0	.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
1001A1EC	.idata	Import	&ADVAPI32.RegQueryValueExA
1001A1F0	.idata	Import	&ADVAPI32.GetUserNameA

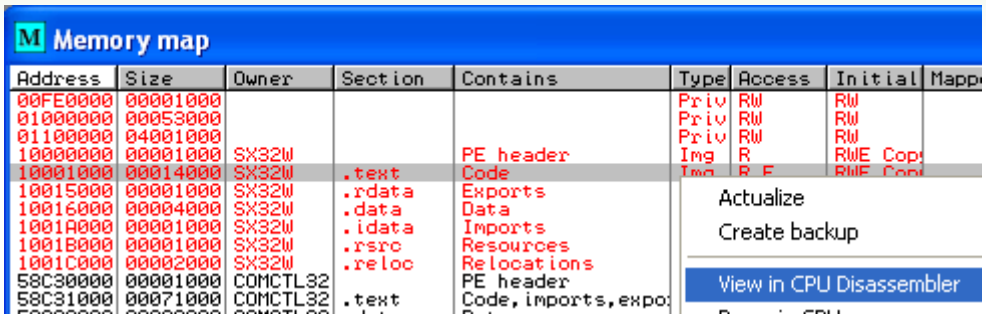
Así que ya tenemos todos los datos necesarios para ponernos a trabajar.

-** Empezando a comprender **-

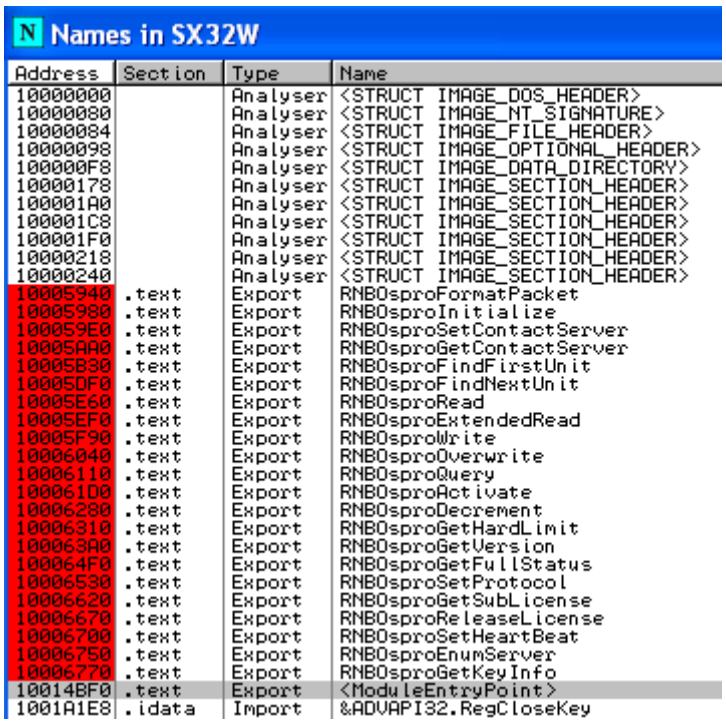
Cargamos el programa en Olly y mientras estamos en el EP nos vamos directamente al mapa de memoria para localizar la dll y colocarle los BPs correspondientes, sin embargo la librería no aparece, por lo que se ve que se carga en tiempo de ejecución. Así que vamos a la configuración de Olly y le ponemos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

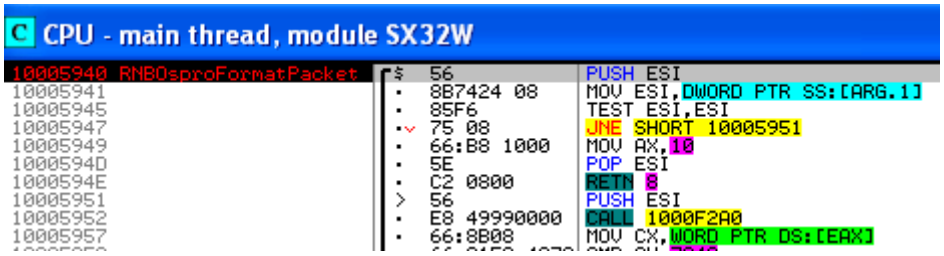


Y buscamos las funciones pulsando Ctrl+N, y ponemos un BP en cada una:



- Primera función a emular: RNBOsproFormatPacket**-**

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



Según el Developer Guide de Sentinel:

Format

▫▫▫▫▫▫▫▫▫ ▫▫▫▫▫ ▫▫▫ RNBOsproFormatPacket▫
RB_SPRO_APIPACKET packet▫
▫▫▫▫▫▫▫▫▫ ▫▫▫▫▫ ▫▫▫ ▫▫▫▫▫▫▫▫▫▫
▫▫

Parameters

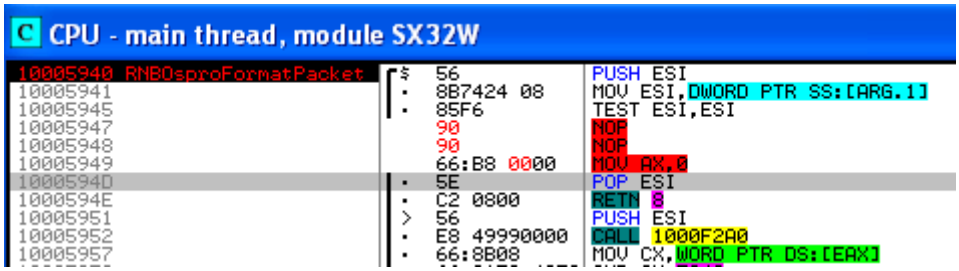
packet A pointer to a DWORD-aligned RB_SPRO_APIPACKET

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)**.

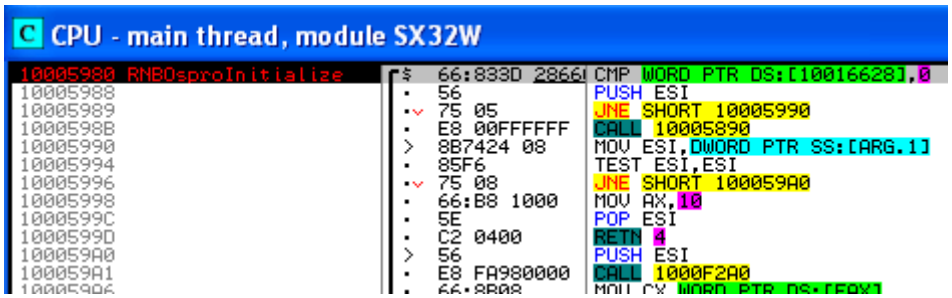
Vemos que si la función tiene éxito, debe devolver 0. Eso equivale a poner AX=0. Para hacer la emulación modificaremos la rutina para dejarla de la siguiente manera:



Con esto ya estaría emulada la primera función.

- Segunda función a emular: RNBOsproInitialize**-**

Si pulsamos F9 llegamos a:



Según el Sentinel Developer Guide:

Format

```
▫▫▫▫▫▫▫▫ ▫▫▫▫▫ ▫▫▫ RNBOsproInitialize▫
RB_SPRO_APIPACKET packet
▫▫
```

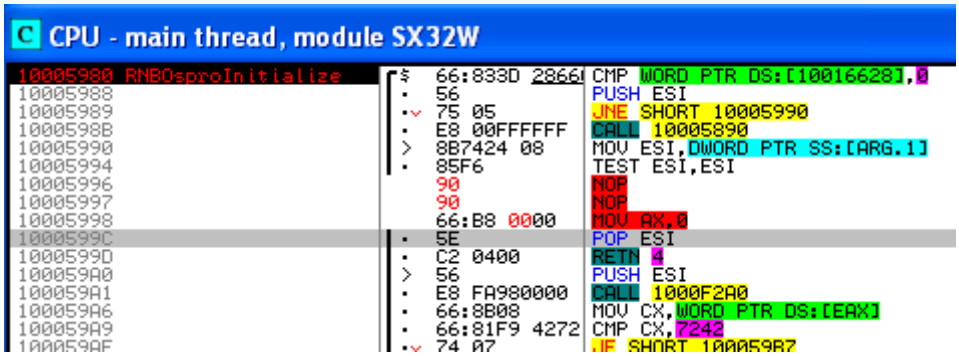
Parameters

packet A pointer to the RB_SPRO_APIPACKET record.

Return Values

If successful, the function returns **SP_SUCCESS (0)**

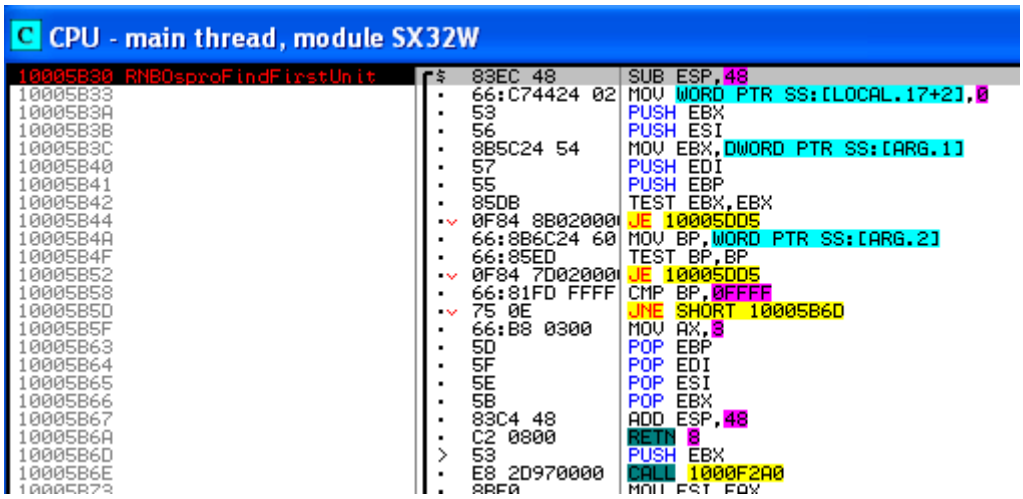
Vemos que si la función tiene éxito, debe devolver 0. Eso equivale a poner AX=0. Para hacer la emulación dejaremos la rutina de la siguiente manera:



Ya tenemos emulada la segunda función.

- 3ª función: RNBOsproFindFirstUnit **-**

Pulsamos F9 para continuar y llegamos a:



Sabemos que:

Format

□□□□□□□□ □□□□□ □□□ RNBOsproFindFirstUnit□
RB_SPRO_APIPACKET packet□
□□□□□□□□ □□□□□ □□□ □□□□□□□□□□□□
□□

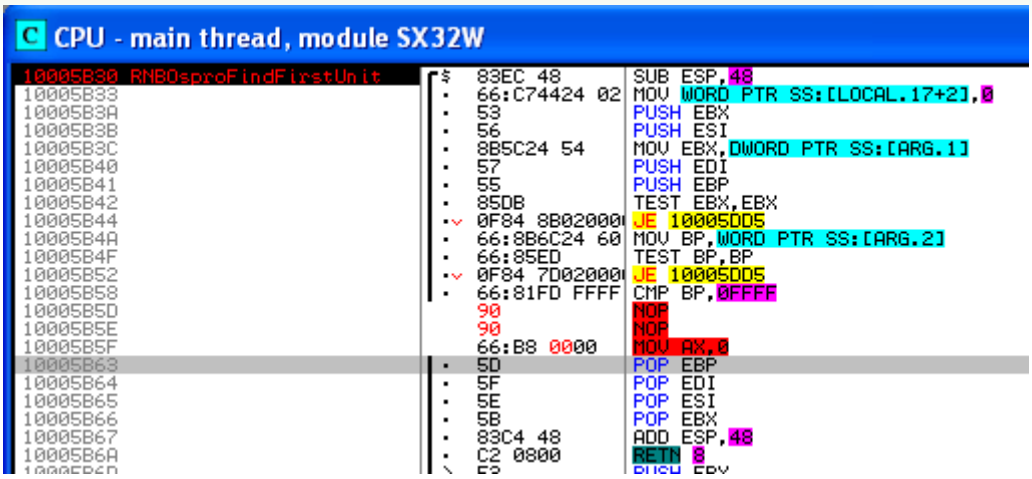
Parameters

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:

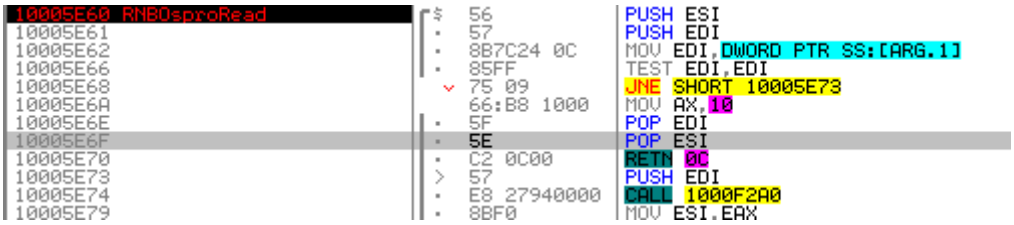


Ahora si pulsamos Ctrl+F9 vemos que salimos de la función con AX=0:

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:



Y en la pila:

Primero vemos qué significa cada argumento de la función RNBOsproRead:

Format

RNBOsproRead
RB_SPRO_APIPACKET packet

Parameters

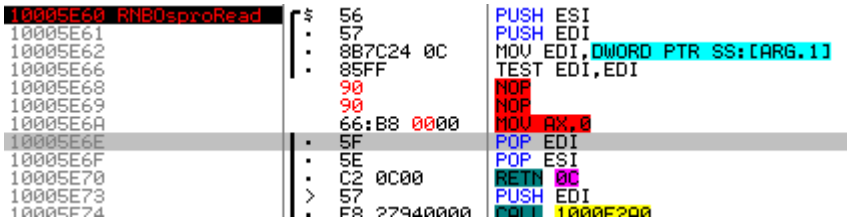
packet A pointer to the RB_SPRO_APIPACKET record.
address The SentinelSuperPro key memory cell address of the word to read.
data A pointer to the location that will contain the data read from the SentinelSuperPro key.

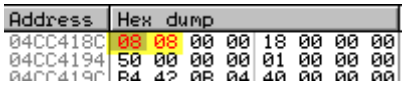
Return Values

If successful, the function returns **SP_SUCCESS(0)**.

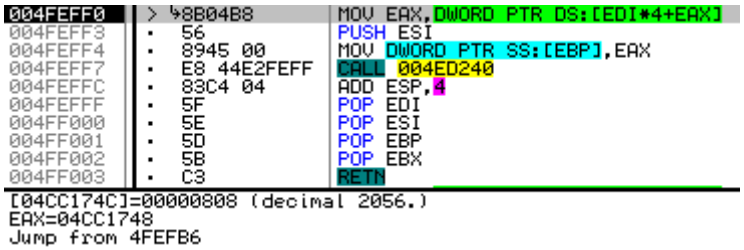
Vemos que el primer argumento es un puntero a APIPACKET, el segundo es la dirección de la celda de la que se quiere leer el Word, y el tercero es un puntero a la dirección donde se escribirá el dato leído. Si la función tiene éxito, escribirá el Word correspondiente en la dirección indicada por el puntero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

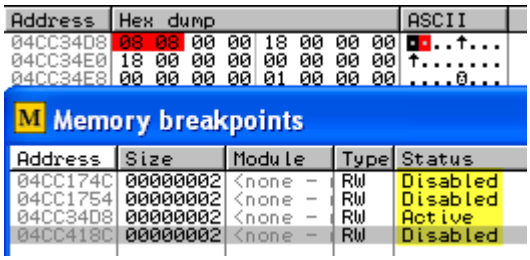




Lo que hemos hecho es poner un valor cualquiera, y le pondremos un BPM a esa dirección de memoria. Si damos a F9 el programa para 3 veces para copiar el contenido de esa celda en diferentes posiciones de memoria, a las que iremos poniendo BPM también. La primera vez para en 4EBD36 , la segunda en 4FF0AC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



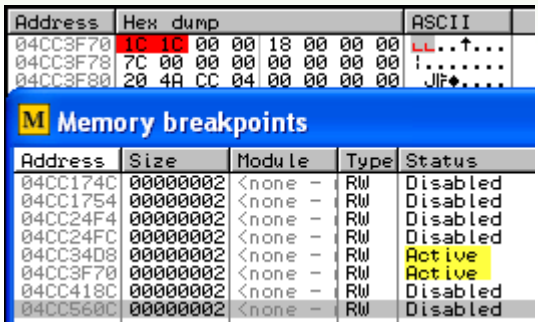
Y cuando llega a 4FEFF4 copia el contenido en 4CC34D8, y ahí ponemos un BPM y podemos quitar todos los demás:



- Segunda llamada a RNBOsproRead **-**

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

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:

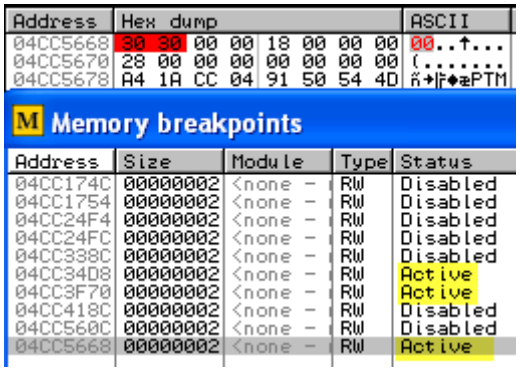


Ponemos un BPM y desactivamos los otros BPMs dejando sólo los de la imagen.

-** 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:

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

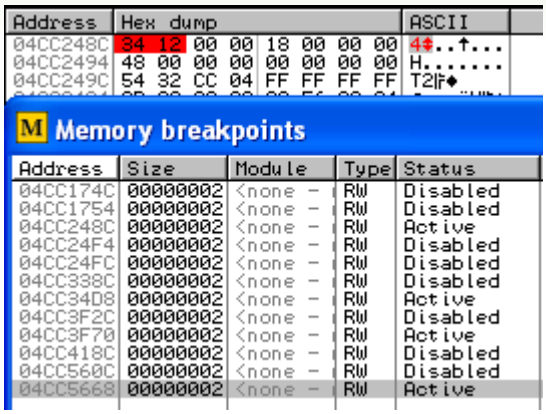


Y ponemos un BPM ahí y dejamos sólo lo que vemos en la imagen de arriba.

-** 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:

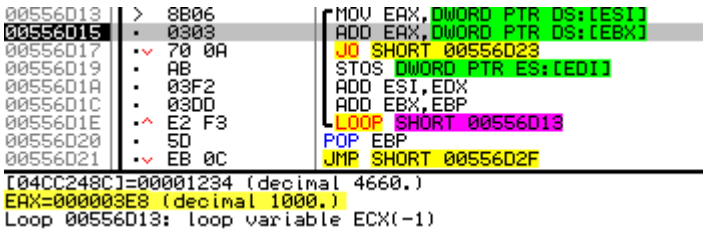
Esta vez escribiremos a mano el valor 1234 en 4CC3F2C y le ponemos como siempre un BPM. Vamos dando F9 y cada vez que pare por lectura ponemos BPM en la nueva dirección donde copia el valor hasta llegar de nuevo a 4FEFF4 en donde lo escribe en 4CC248C, donde pondremos un BPM y dejaremos sólo los que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



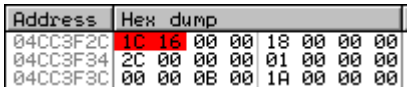
- Manipulando bytes: que hace con la celda 1C? **-**

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

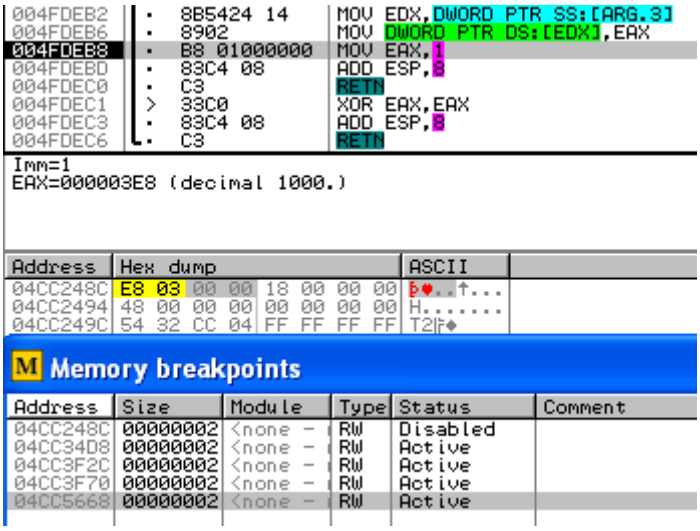
Si pulsamos F9 para en 556D15 donde le suma 3E8h (1000d) al valor que había en la celda 00 y la guarda en EAX (1234 + 3E8= 161C):



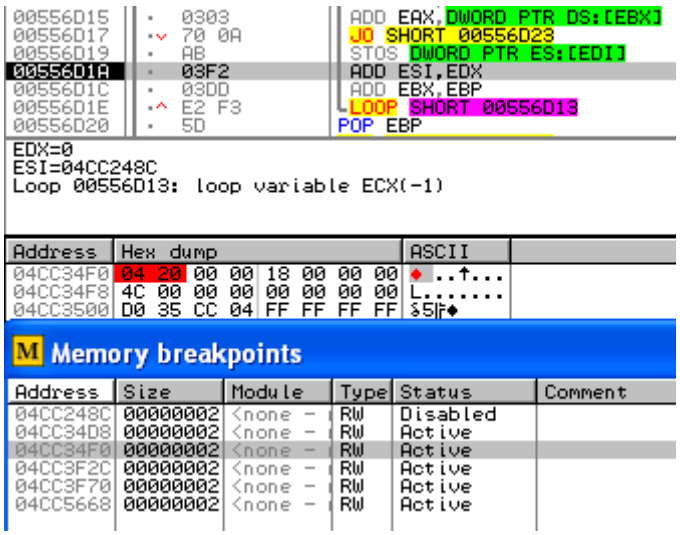
Y en 556D19 guarda el resultado de la suma en 4CC3F2C. Ahí ponemos otro BPM:



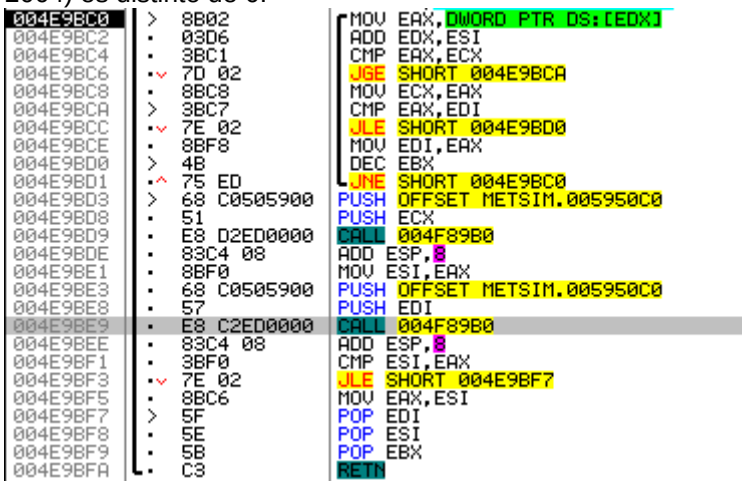
Si seguimos con F9 para en 4FDEB6 donde sobrescribe el valor que teníamos guardado en 4CC248C así que podemos quitar ese BPM:



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:



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.



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            RETN
```

```
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 4E994Fpor el BPM que tenemos en 5950C0, lo guarda en EAX, lo empuja a la pila:

```
004E9945  > 3B1D BC50590( CMP EBX,DWORD PTR DS:[5950BC]
004E994B  . 75 02         JNE SHORT 004E994F
004E994D  . 33DB          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 ........
```

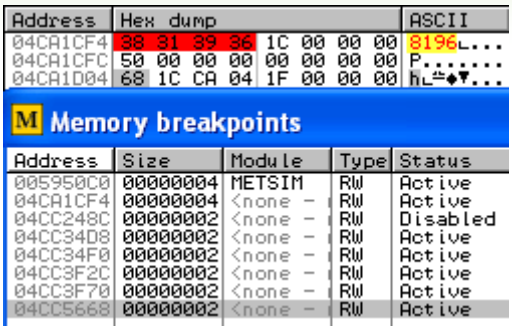
Así que vamos traceando con cuidado con F8 hasta que llegamos a:

```
004E99BD  . 81FE C050590( CMP ESI,OFFSET METSIM.005950C0   ASCII "6918"
004E99C3  . 72 16         JB SHORT 004E99DB
004E99C5  > 0FBE06        MOVSX EAX,BYTE PTR DS:[ESI]
004E99C8  . 50            PUSH EAX
004E99C9  . FF15 7C67590( CALL DWORD PTR DS:[59677C]
004E99CF  . 83C4 04       ADD ESP,4
004E99D2  . 4E            DEC ESI
004E99D3  . 81FE C050590( 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 5BFFFFF( JNE 004E9945
```

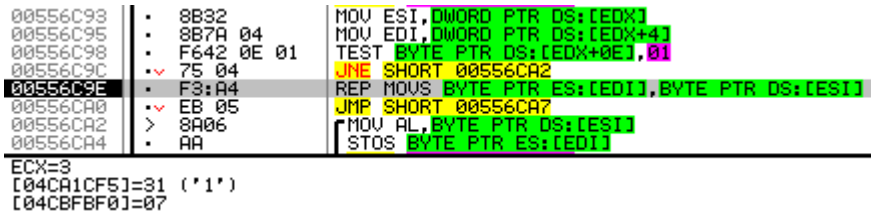
Ahí entramos con F7 y esta rutina le da la vuelta al número 6918, dejándolo en 8169, que ahora ya es 1C1C + 3E8 = 2010h = 8169 y lo deja escrito en 4CA1CF4:

```
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            RETN
```

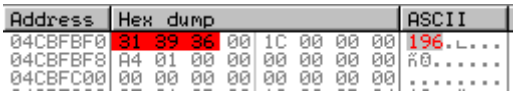
Y ahí le ponemos otro BPM:



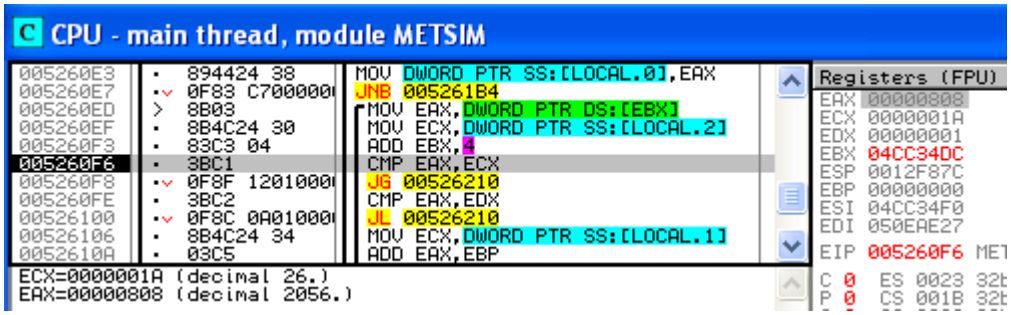
Ahora damos a continuar de nuevo y paramos en 556C9E conde copia las 3 últimas cifras del número 8196 en 4CBFBF0:



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



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.



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 <=0 y >= 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  .v 0F8F 1201000( JG 00526210
005260FE  .  3BC2          CMP EAX,EDX
00526100  .v 0F8C 0A01000( JL 00526210
00526106  .  8B4C24 34     MOV ECX,DWORD PTR SS:[LOCAL.1]
0052610A  .  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  .v 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	PQRSTUVW
050EAE3F	58 59 5A 00 00 34 00 00	XYZ..4..
050EAE47	00 18 00 00 00 04 00 00	.↑...♦..

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	L.......

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 numero 8196, formando una cadena alfanumérica con esa letra y las 3 últimas cifras del número, o sea T196:

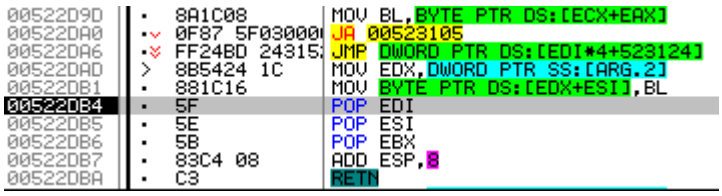
Address	Hex dump	ASCII
04CA1CF4	54 31 39 36 1C 00 00 00	T196∟...
04CA1CFC	50 00 00 00 00 00 00 00	P.......
04CA1D04	68 1C CA 04 1F 00 00 00	h∟╩♦▼...

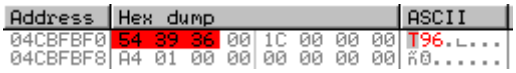
Como el BPM ya está puesto de antes pulsamos F9 para continuar, y paramos en 556C9E donde copia esta cadena alfanumérica en 4CA1CF4:

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

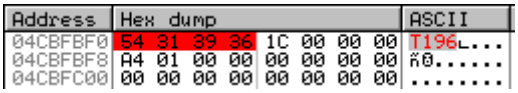
Address	Hex dump	ASCII
04CA1CED	00 00 00 04 00 00 00 54	...♦...T
04CA1CF5	31 39 36 1C 00 00 00 50	196∟...P

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:

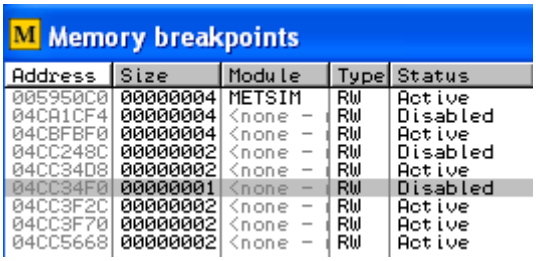




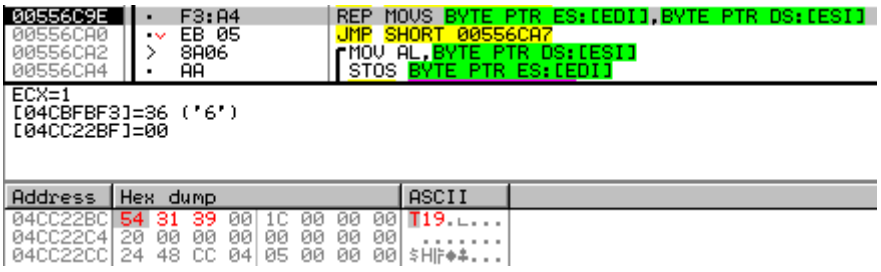
Pues parará 3 veces más aquí de manera que al final tendremos en 4CBFCF0 copiada la cadena alfanumérica completa:

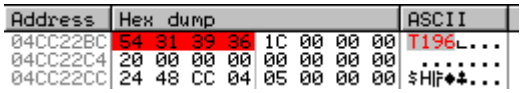


Después de poner un BPM a la cadena completa volvemos a pulsar F9 parará 2 veces para escribir en 4CC34F0 y después en 1CA1CF4, por lo tanto podemos deshabilitar esos 2 BPM:



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





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  |.v 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]
005212D2  |. 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 38 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 **-

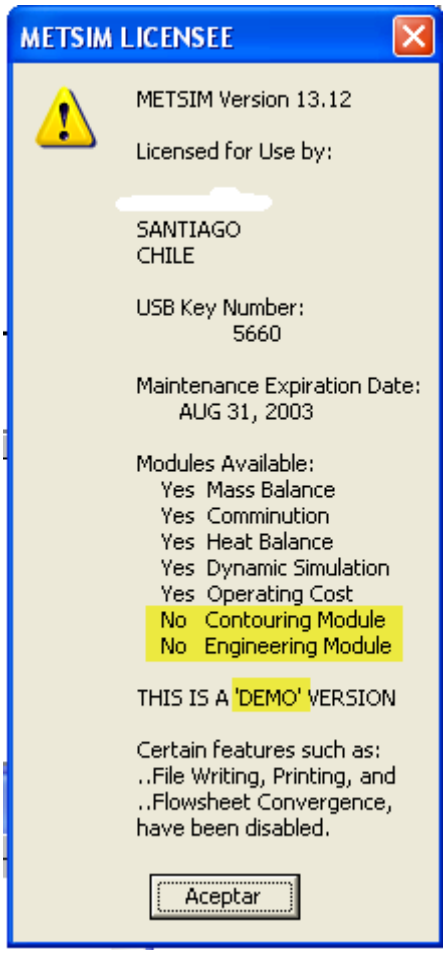
Bueno pues antes de reiniciar vamos a elegir una cadena cualquiera de las de ahí arriba, por ejemplo, la primera P002.

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  |.v 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  |.v E9 46000000    JMP 10005EC1
10005E7B  |> 837C24 10 1C    CMP DWORD PTR SS:[ARG.2],1C
10005E80  |.v 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  |.v E9 2D000000    JMP 10005EC1
10005E94  |> 837C24 10 30    CMP DWORD PTR SS:[ARG.2],30
10005E99  |.v 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  |.v E9 14000000    JMP 10005EC1
10005EAD  |> 837C24 10 00    CMP DWORD PTR SS:[ARG.2],0
10005EB2  |.v 0F85 0900000(  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        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:



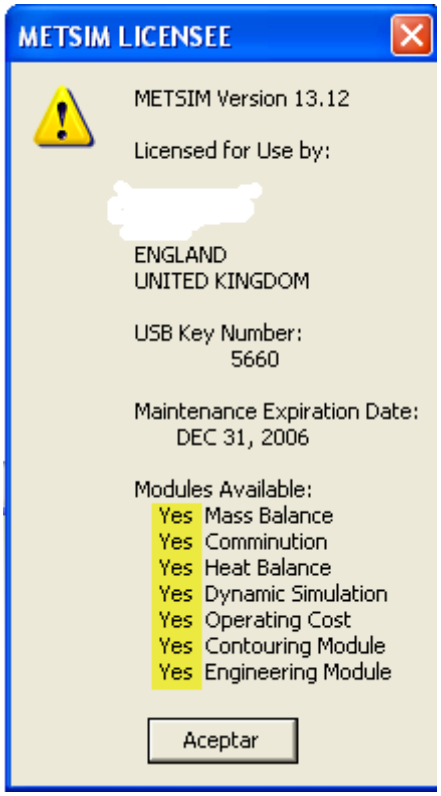
Ha desaparecido el cartel inicial de que no encuentra protección, sin embargo nos sigue diciendo que es una versión DEMO y además hay algunos módulos que no están disponibles.

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  .v 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  .v E9 46000000   JMP 10005EC1
10005E7B  >  837C24 10 1C  CMP DWORD PTR SS:[ARG.2],1C
10005E80  .v 0F85 0E000000 JNE 10005E94
10005E86  .  8B7C24 14     MOV EDI,DWORD PTR SS:[ARG.3]
10005E8A     66:C707 2803  MOV WORD PTR DS:[EDI],328     <- Para obtener el
10005E8F  .v E9 2D000000   JMP 10005EC1                     808
10005E94  >  837C24 10 30  CMP DWORD PTR SS:[ARG.2],30
10005E99  .v 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  .v E9 14000000   JMP 10005EC1
10005EAD  >  837C24 10 00  CMP DWORD PTR SS:[ARG.2],0
10005EB2  .v 0F85 0900000( 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  L. C2 0C00       RETN 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