

CracksLatinoS! 2010



CracksLatinoS! 2010

-* Desprotegiendo Eutron Smartkey *-

Programa	CECAP
Download	
Descripción	
Herramientas	OllyDbg v2.0
Dificultad	Media
Compresor/Compilador	
Protección	Eutron Smartkey Dongle
Objetivos	Hacer correr el programa sin mochila "a full"
Cracker	Lionel lionelgomezdu@yahoo.es
	Fecha: 10/03/10
Tutorial n°	

.** Introducción **.

Parece que últimamente sólo llegan a mis manos programas protegidos con algún tipo de mochila. Si embargo en este caso la novedad es que nunca antes había tenido delante uno protegido con este tipo de dongle. La principal dificultad que me he encontrado es que no he conseguido encontrar ningún tutorial escrito sobre esta protección, lo que ha hecho que tenga que ponerme a estudiar todo desde cero, y además la situación empeora considerando que la aplicación está escrita en Visual Basic, con lo que el traceo se hace mucho más "desagradable".

Como suele ocurrir en estos casos ha sido fundamental la lectura del manual del usuario de la llave, que tan amablemente la empresa fabricante de la mochila pone a nuestra disposición en su página web. Ése es pues el único punto de partida. A partir de aquí todo queda en manos de nuestro (escaso) ingenio.

Debo decir que lo que voy a explicar aquí es el resumen de varias semanas de trabajo leyendo, investigando, probando y traceando. Al principio casi me di por vencido, pero finalmente, tirando poco a poco del hilo conseguí comprender finalmente el funcionamiento. Digo esto para todos aquellos que cuando ven un programa protegido con una mochila enseguida se van para atrás. Por mi experiencia puedo decir que en la mayoría de los casos los programadores de aplicaciones que dejan la protección de su programa a un dongle en general es porque no son capaces de protegerlo decentemente sin él, y por tanto, tampoco se preocupan demasiado en diseñar un buen esquema de protección con todas las posibilidades que ofrece este tipo de llaves. Así que ya sabéis: a insistir.

CracksLatinoS! 2010

Sin más, vamos a comenzar.

.-** Explorando el objetivo **.-

Todo lo que voy a contar en este apartado es un pequeño resumen de lo más importante que se cuenta en el manual del usuario del dongle en cuanto a las funciones del api de que dispone el programador para proteger sus aplicaciones con este sistema de protección.

Existen varios tipos de mochilas Smartkey (FX, PR, SP y XM), , que básicamente se diferencian en la cantidad de memoria interna de que disponen. Y además hay un tipo en particular cuyo uso es exclusivo para redes que son las NET y que además tiene un juego de comandos especial.

El sistema se basa en la ejecución de una función a la que se le pasan una serie de argumentos a través de una estructura definida de la siguiente forma:

```
struct smartkey (  
word lpt;  
word command;  
byte label[16];  
byte password[16];  
byte data[64];  
word fail_counter;  
word status;  
byte ext_data[352];  
)
```

donde:

<i>Lpt</i>	Identificador del puerto paralelo o USB donde está colocada la llave
<i>Command</i>	Codigo del comando que se ejecutará
<i>label</i>	Etiqueta de la llave. Es necesaria para todos los comandos.
<i>password</i>	Es necesaria para todos los comandos que requieren acceso a la memoria del dongle.
<i>Data</i>	Contenido de la memoria interna de la llave y buffer genérico para las operaciones que requieren intercambio de datos con la llave.
<i>fail_counter</i>	Contador de los accesos fallidos a la llave.
<i>status</i>	Resultado de la ejecución del comando. Un valor 0 indica que el comando se ejecutó correctamente..
<i>ext_data</i>	Contenido de la memoria extendido de la llave.

Debe considerarse que, aunque algunos campos tienen el mismo nombre que los registros físicos de la llave, son entidades diferentes. Por ejemplo, el contenido de la memoria de la pastilla está realmente presente en el campo data de la estructura sólo durante las operaciones de lectura y escritura de la pastilla. Por ejemplo, durante la operación de Scrambilig o codificación, el campo data contiene los datos para la codificación entre el PC y el dongle. La función Scrambling utiliza el campo data como variable para soportar sus propias operaciones, y no modifica el contenido de la memoria interna de la pastilla.

Para ejecutar un comando hay que llevar a cabo las siguientes tareas:

CracksLatinoS! 2010

- 1) Declarar una variable de tipo estructura con los campos mencionados anteriormente
- 2) Rellenar los campos de las variables con los valores requeridos por el comando. En particular, el campo command debe definirse para indicar qué función va a ejecutarse, y además deben rellenarse todos los campos necesarios para ejecutar esa función.
- 3) Llamar a la función definida en el driver de la SmartKey, pasándole la estructura definida. Esta función generalmente se llama msclink().
- 4) Leer el resultado del campo status el resultado del comando y cualquier otro valor de salida.

Los comandos que pueden utilizarse con los tipos de llave que no son NET son los siguientes (entre paréntesis pongo la letra que hay que pasar a la estructura para indicar ese comando):

Locating (L): Sirve para localizar en qué puerto está conectado el dongle.

Scrambling (S): Similar a la función Encode de HASP. Se le pasa como argumento un valor y le aplica un algoritmo transformándolo en otro valor. Dongles con Id-Codes diferentes dan como resultado valores diferentes cuando se le pasan como datos valores iguales.

Reading (R): Lee el contenido de la memoria de la llave.

Writing (W): Escribe en la memoria interna de la llave.

Block Reading (BR): Lee porciones de la memoria interna de la pastilla.

Block Writing (BW): Escribe trozos de la memoria interna.

Fixing (F): Sirve para evitar que se modifique más el label, el password y los datos de la memoria interna de la pastilla.

Programing (P): Modifica label y el password de la pastilla.

Comparing (C): Busca en todos los puertos si se encuentra una mochila con el label y el password dados.

Model Reading (M): Obtiene el modelo del dongle que tenemos conectado.

Serial Number (N): Obtiene el número de serie de la pastilla.

Extended Model Reading (H): Obtiene más datos acerca del modelo de dongle conectado.

Fix Reading (X): Comprueba si la pastilla ha sido "fixeada" con el comando Fixing.

Fail Counter Reading (A): Lee el contenido del registro fail counter de la pastilla.

AES Set Mode (G): Establece las claves para el algoritmo AES

AES Scramble Mode (A): Alternativa a la función Scrambling, utilizando el algoritmo AES.

Todos estos son los comandos standard de los modelos "normales". Aparte tenemos las llaves de tipo NET, que soportan además comandos particulares de la red, que son los siguientes:

Open mode (O): Se utiliza para activar la comunicación con la llave.

Access Mode (A): Activa la ejecución de cualquiera de los comandos standard.

CracksLatinoS! 2010

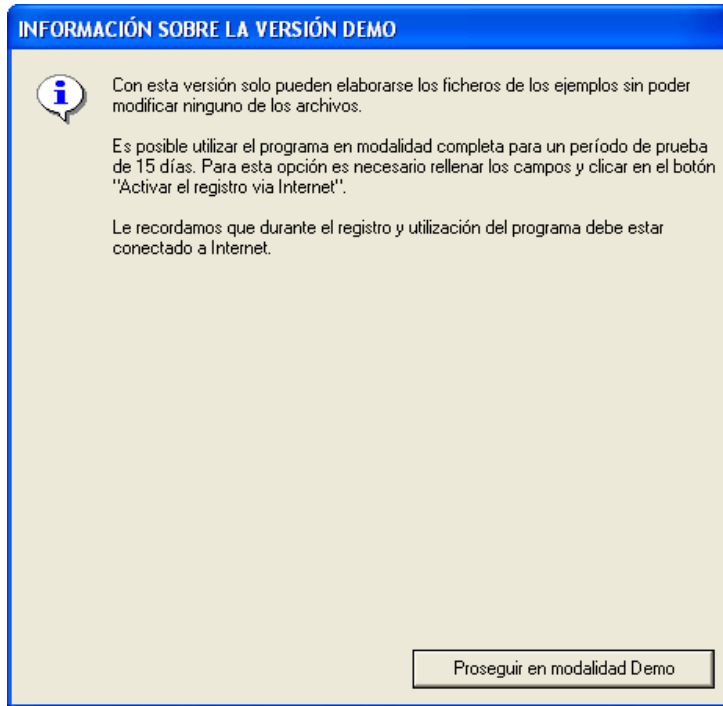
User Number Mode (U): Obtiene el número de usuarios conectados al dongle.

Close Mode (C): Cierra la comunicación con la pastilla.

Bueno, tras esta breve explicación creo que ya podemos empezar a meternos de lleno en nuestro objetivo.

.-** Primeros pasos **.-

Arrancaremos el programa a ver si nos dice algo:

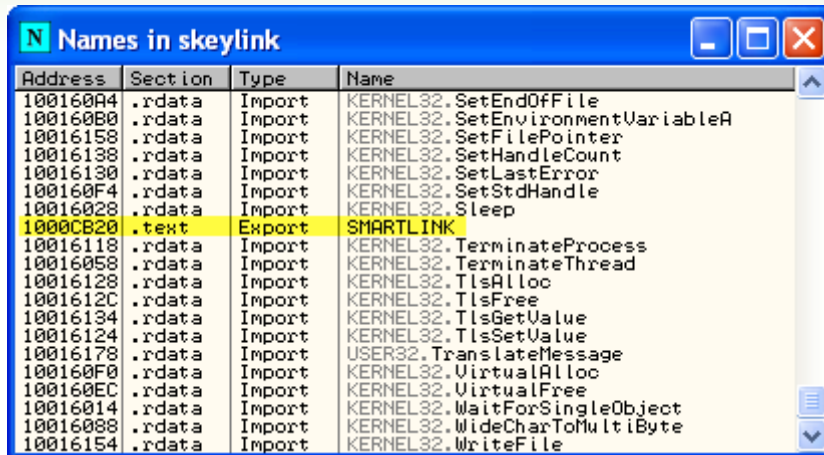


Vemos que directamente nos indica que es una versión demo, pero si leemos el manual del usuario del programa en cuestión nos dice que para funcionar debemos tener colocada una mochila y además ahí mismo nos indica la marca de esta mochila: Eutron. Así que lo tenemos más fácil. Además, si miramos en el directorio donde se instaló la aplicación vemos dos archivos sospechosos:

 skeydrv.dll	60 KB
 skeylink.dll	128 KB

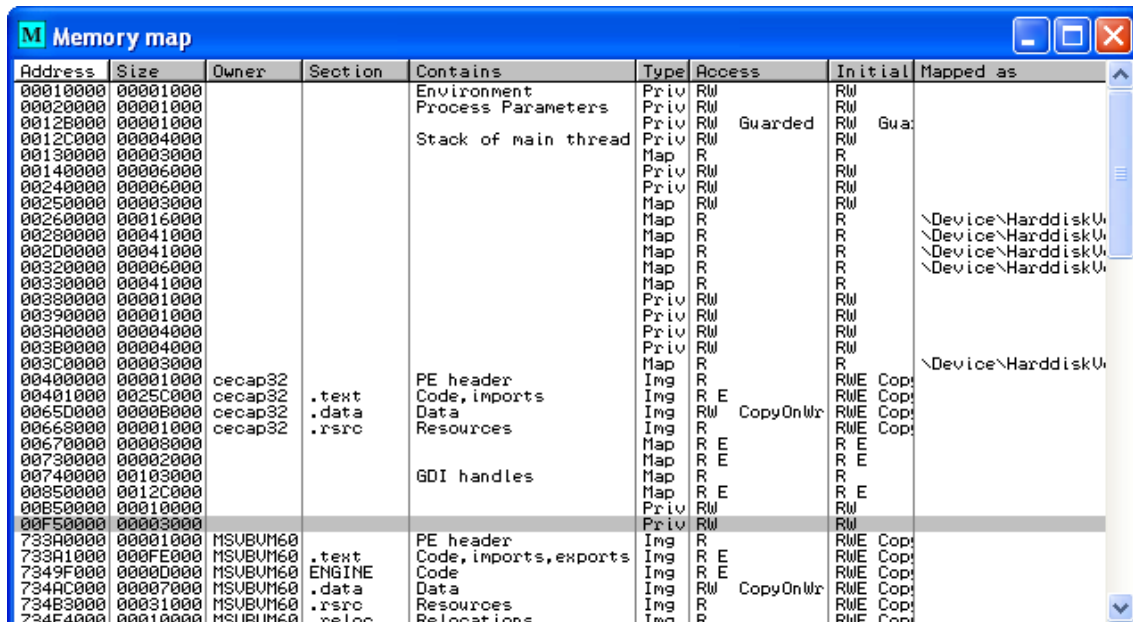
Cargamos el skeylink.dll en Olly (en la versión 1.10, ya que la 2.0 no permite todavía cargar dlls), y nos vamos a ver qué funciones exporta e importa:

CracksLatinoS! 2010



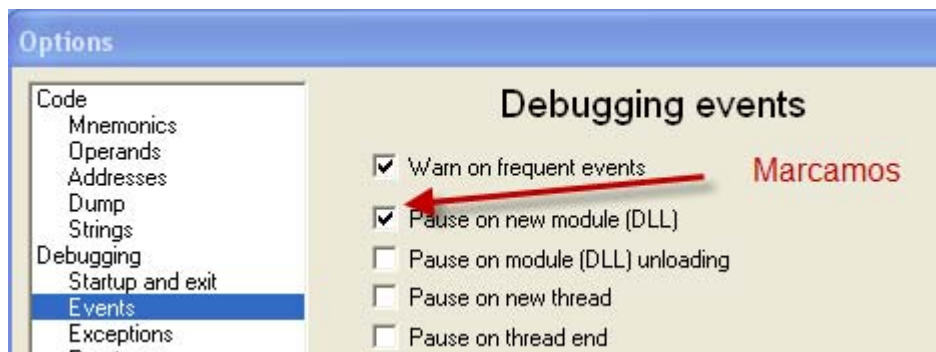
Address	Section	Type	Name
10016004	.rdata	Import	KERNEL32.SetEndOfFile
10016008	.rdata	Import	KERNEL32.SetEnvironmentVariableA
10016158	.rdata	Import	KERNEL32.SetFilePointer
10016138	.rdata	Import	KERNEL32.SetHandleCount
10016130	.rdata	Import	KERNEL32.SetLastError
100160F4	.rdata	Import	KERNEL32.SetStdHandle
10016028	.rdata	Import	KERNEL32.Sleep
1000CB20	.text	Export	SMARTLINK
10016118	.rdata	Import	KERNEL32.TerminateProcess
10016058	.rdata	Import	KERNEL32.TerminateThread
10016128	.rdata	Import	KERNEL32.TlsAlloc
1001612C	.rdata	Import	KERNEL32.TlsFree
10016134	.rdata	Import	KERNEL32.TlsGetValue
10016124	.rdata	Import	KERNEL32.TlsSetValue
10016178	.rdata	Import	USER32.TranslateMessage
100160F0	.rdata	Import	KERNEL32.VirtualAlloc
100160EC	.rdata	Import	KERNEL32.VirtualFree
10016014	.rdata	Import	KERNEL32.WaitForSingleObject
10016088	.rdata	Import	KERNEL32.WideCharToMultiByte
10016154	.rdata	Import	KERNEL32.WriteFile

Podemos observar que sólo exporta una función y además se llama Smartlink. Esto es muy, muy sospechoso. Así que ahora sí que cargamos en Olly 2 nuestra aplicación y nos vamos a buscar si tiene la dll cargada en memoria:



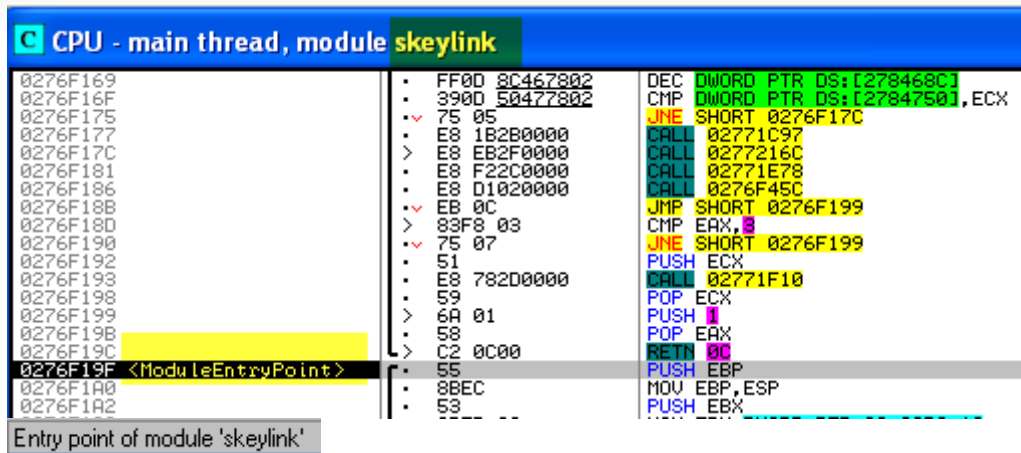
Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000			Environment	Priv	RW	RW	
00020000	00001000			Process Parameters	Priv	RW	RW	
0012B000	00001000			Stack of main thread	Priv	RW	RW	
0012C000	00004000				Priv	RW	RW	
00130000	00003000				Map	R	R	
00140000	00006000				Priv	RW	RW	
00240000	00006000				Priv	RW	RW	
00250000	00003000				Map	RW	RW	
00260000	00016000				Map	R	R	
00280000	00041000				Map	R	R	\Device\HarddiskU
002D0000	00041000				Map	R	R	\Device\HarddiskU
00320000	00006000				Map	R	R	\Device\HarddiskU
00330000	00041000				Map	R	R	\Device\HarddiskU
00380000	00001000				Priv	RW	RW	
00390000	00001000				Priv	RW	RW	
003A0000	00004000				Priv	RW	RW	
003B0000	00004000				Priv	RW	RW	
003C0000	00003000				Map	R	R	\Device\HarddiskU
00400000	00001000	cecapi32		PE header	Ing	R	RWE	Cop
00401000	0025C000	cecapi32	.text	Code, imports	Ing	R	RWE	Cop
0065D000	0000B000	cecapi32	.data	Data	Ing	RW	RWE	Cop
00668000	00001000	cecapi32	.rsrc	Resources	Ing	RW	RWE	Cop
00670000	00003000				Map	R	R	
00730000	00002000				Map	R	R	
00740000	00103000			GDI handles	Map	R	R	
00850000	0012C000				Map	R	R	
00B50000	00010000				Priv	RW	RW	
00F50000	00003000				Priv	RW	RW	
733A0000	00001000	MSUBUM60		PE header	Ing	R	RWE	Cop
733A1000	000FE000	MSUBUM60	.text	Code, imports, exports	Ing	R	RWE	Cop
7349F000	0000D000	MSUBUM60	ENGINE	Code	Ing	R	RWE	Cop
734AC000	00007000	MSUBUM60	.data	Data	Ing	RW	RWE	Cop
734B3000	00031000	MSUBUM60	.rsrc	Resources	Ing	RW	RWE	Cop
734E4000	00010000	MSUBUM60	.reloc	Relocations	Ing	R	RWE	Cop

Podemos ver que no, o sea que quizá la cargue en tiempo de ejecución, por tanto vamos a configurar Olly para que pare cada vez que carga una librería:



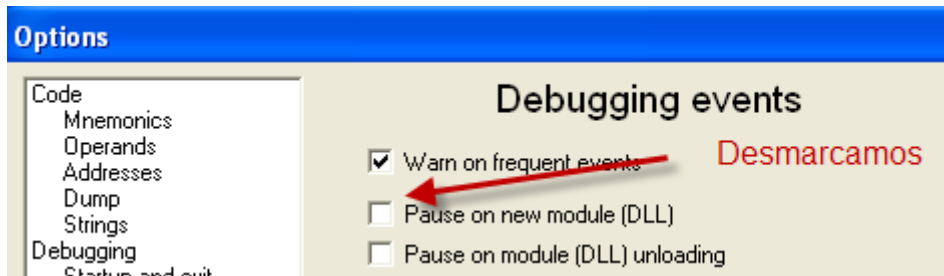
Y ahora ya podemos arrancar el programa con F9. Parará bastantes veces, pero finalmente llegamos a:

CracksLatinoS! 2010

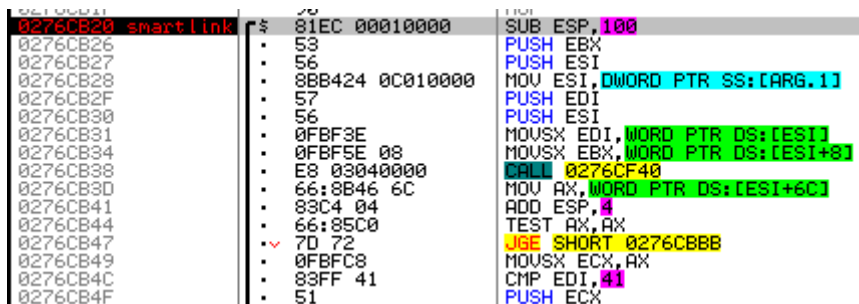


```
CPU - main thread, module keylink
0276F169  FF0D 8C467802 DEC DWORD PTR DS:[278468C]
0276F16F  390D 50477802 CMP DWORD PTR DS:[2784750],ECX
0276F175  75 05 JNE SHORT 0276F17C
0276F177  E8 1B2B0000 CALL 02771C97
0276F17C  E8 EB2F0000 CALL 0277216C
0276F181  E8 F22C0000 CALL 02771E78
0276F186  E8 D1020000 CALL 0276F45C
0276F18B  EB 0C JMP SHORT 0276F199
0276F18D  83F8 03 CMP EAX,3
0276F190  75 07 JNE SHORT 0276F199
0276F192  51 PUSH ECX
0276F193  E8 782D0000 CALL 02771F10
0276F198  59 POP ECX
0276F199  6A 01 PUSH 1
0276F19B  58 POP EAX
0276F19C  C2 0C00 RETN 0C
0276F19F <ModuleEntryPoint> 55 PUSH EBP
0276F1A0  8BEC MOV EBP,ESP
0276F1A2  53 PUSH EBX
Entry point of module 'keylink'
```

Este es el Entry Point de keylink.dll. Ahora vamos a buscar la función Smartlink que vimos antes y le pondremos un BP, y además desmarcaremos la casilla de parar Olly cada vez que cargue un librería ya que ya no lo necesitamos:

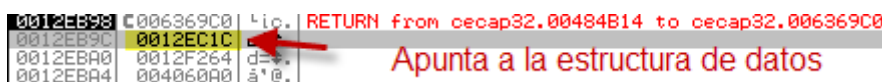


Así que si damos a F9 enseguida pararemos por el BP que hemos puesto:



```
0276CB20 smartlink 81EC 00010000 SUB ESP,100
0276CB26 53 PUSH EBX
0276CB27 56 PUSH ESI
0276CB28 8BB424 0C010000 MOV ESI,DWORD PTR SS:[ARG.1]
0276CB2F 57 PUSH EDI
0276CB30 56 PUSH ESI
0276CB31 0FBF3E MOVSX EDI,WORD PTR DS:[ESI]
0276CB34 0FBF5E 08 MOVSX EBX,WORD PTR DS:[ESI+8]
0276CB38 E8 03040000 CALL 0276CF40
0276CB3D 66:8B46 6C MOV AX,WORD PTR DS:[ESI+6C]
0276CB41 83C4 04 ADD ESP,4
0276CB44 66:85C0 TEST AX,AX
0276CB47 7D 72 JGE SHORT 0276CBBB
0276CB49 0FBFC8 MOVSX ECX,AX
0276CB4C 83FF 41 CMP EDI,41
0276CB4F 51 PUSH ECX
```

Si vemos la pila, debe apuntar a la estructura de datos que mencionábamos en el apartado anterior:



```
0012EB98 C006369C0 |<|. RETURN from cecap32.00484B14 to cecap32.006369C0
0012EB9C 0012EC1C |<|.
0012EBA0 0012F264 |d=s|.
0012EBA4 004060A0 |a*0. |
```

Y en esa posición de la pila tenemos lo que vemos a continuación, que es la estructura completamente definida para una llave tipo NET, donde podemos observar claramente que el comando que se va a utilizar es el "O" de Open con label = geo&soft y contraseña = moss.kat:

CracksLatinoS! 2010

Address	Hex dump	ASCII
0012EC1C	4F 00 00 00 00 00 00 00 00 00 67 65 6F 26 73 6F	0.....geo&so
0012EC20	66 74 00 00 00 00 00 00 00 00 6D 6F 73 73 2E 6B	ft.....moss.k
0012EC3C	61 74 00 00 00 00 00 00 00 00 00 00 00 00 00 00	at.....
0012EC4C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EC5C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EC6C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EC7C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EC8C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012EC9C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0012ECAC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

	Net_Command
	Command
	Label
	Password
	Data
	Fail Counter
	Status

- Emulación del primer comando **-**

Si empezamos a tracear con F8 desde donde estamos que es 276CB20 llegamos enseguida a 276CB38 CALL 276CF40, en el que entramos con F7 y vemos:

0276CF40	\$ A1 7C467802	MOV EAX, DWORD PTR DS:[278467C]	keylink.0276CF40(guessed Arg1)
0276CF45	85C0	TEST EAX, EAX	
0276CF47	75 4A	JNE SHORT 0276CF98	
0276CF49	8B00 78467802	MOV ECX, DWORD PTR DS:[2784678]	
0276CF4F	85C9	TEST ECX, ECX	
0276CF51	75 18	JNE SHORT 0276CF68	
0276CF53	E8 58080000	CALL 0276D7B0	
0276CF58	A1 7C467802	MOV EAX, DWORD PTR DS:[278467C]	
0276CF5D	85C0	TEST EAX, EAX	
0276CF5F	75 32	JNE SHORT 0276CF98	
0276CF61	8B00 78467802	MOV ECX, DWORD PTR DS:[2784678]	
0276CF67	85C9	TEST ECX, ECX	
0276CF69	74 18	JE SHORT 0276CF83	
0276CF68	8B4424 04	MOV EAX, DWORD PTR SS:[ARG.1]	
0276CF6F	83C0 6C	ADD EAX, 6C	
0276CF72	66:8908	MOV WORD PTR DS:[EAX], CX	
0276CF75	C705 78467802 0000	MOV DWORD PTR DS:[2784678], 0	
0276CF7F	66:8B00	MOV AX, WORD PTR DS:[EAX]	
0276CF82	C3	RET	
0276CF83	8B4C24 04	MOV ECX, DWORD PTR SS:[ARG.1]	
0276CF87	0D41 6C	LEA EAX, [ECX+6C]	
0276CF8A	66:C700 F5FF	MOV WORD PTR DS:[EAX], 0FFFF	
0276CF8F	66:8B00	MOV AX, WORD PTR DS:[EAX]	
0276CF92	C3	RET	
0276CF93	56	PUSH ESI	
0276CF94	8B7424 08	MOV ESI, DWORD PTR SS:[ARG.1]	
0276CF98	66:8B16	MOV DX, WORD PTR DS:[ESI]	
0276CF9B	52	PUSH EDX	[Arg1 keylink.0276D040
0276CF9C	E8 9F000000	CALL 0276D040	
0276CFA1	66:8906	MOV WORD PTR DS:[ESI], AX	
0276CFA4	83C4 04	ADD ESP, 4	
0276CFA7	0F8FC0	MOVSX EAX, AX	
0276CFAA	83C0 BF	ADD EAX, -4F	
0276CFAD	83F8 10	CMP EAX, 10	Switch (cases 41..51, 5 exits)
0276CFB0	77 4B	JG SHORT 0276CFD0	
0276CFB2	33C9	XOR ECX, ECX	
0276CFB4	8A88 20007602	MOV CL, BYTE PTR DS:[EAX+276D020]	
0276CFBA	FF2480 0C007602	JMP DWORD PTR DS:[ECX*4+276D00C]	Case 4F ('O') of switch keylink.276CFAA
0276CFB1	56	PUSH ESI	
0276CFB2	E8 F9000000	CALL Open	
0276CFB7	66:8B46 6C	MOV AX, WORD PTR DS:[ESI+6C]	
0276CFB8	83C4 04	ADD ESP, 4	
0276CFB9	5E	POP ESI	
0276CFBD	C3	RET	
0276CFD0	56	PUSH ESI	
0276CFD1	E8 9A030000	CALL Access	Case 41 ('A') of switch keylink.276CFAA
0276CFD6	66:8B46 6C	MOV AX, WORD PTR DS:[ESI+6C]	
0276CFDA	83C4 04	ADD ESP, 4	
0276CFDD	5E	POP ESI	
0276CFDE	C3	RET	
0276CFDF	56	PUSH ESI	
0276CFE0	E8 CB060000	CALL Close	Case 43 ('C') of switch keylink.276CFAA
0276CFE5	66:8B46 6C	MOV AX, WORD PTR DS:[ESI+6C]	
0276CFE9	83C4 04	ADD ESP, 4	
0276CFEC	5E	POP ESI	
0276CFED	C3	RET	
0276CFEE	56	PUSH ESI	
0276CFEF	E8 3C070000	CALL Quit	Case 51 ('Q') of switch keylink.276CFAA
0276CFF4	66:8B46 6C	MOV AX, WORD PTR DS:[ESI+6C]	
0276CFF8	83C4 04	ADD ESP, 4	
0276CFFB	5E	POP ESI	
0276CFFC	C3	RET	
0276CFFD	66:C746 6C FEFF	MOV WORD PTR DS:[ESI+6C], 0FFFF	Default case of switch keylink.276CFAA
0276D003	66:8B46 6C	MOV AX, WORD PTR DS:[ESI+6C]	
0276D007	5E	POP ESI	
0276D008	C3	RET	

CracksLatinoS! 2010

Es importante ver que en 276CFBA hay un salto que en función del valor de ECX salta al caso "O", "A", "C" o "Q" del switch que hay en 276CFAA, que casualmente son los valores de las funciones disponibles cuando la llave es del tipo NET. Por tanto vamos con F8 traceando hasta que llegamos a: 276CFC2 CALL 276D0C0 a la que entramos con F7:

```

0276D0C0 Open 81EC 04010000 SUB ESP,104
0276D0C6      53      PUSH EBX
0276D0C7      56      PUSH ESI
0276D0C8      8BB424 10010000 MOV ESI,DWORD PTR SS:[ARG.1]
0276D0CF      57      PUSH EDI
0276D0D0      68 EC8F7702 PUSH OFFSET skeyLink.02778FFC
0276D0D5      56      PUSH ESI
0276D0D6      33DB     XOR EBX,EBX
0276D0D8      E8 5383FFFF CALL 02765430
0276D0DD      83C4 08     ADD ESP,8
0276D0E0      E8 6B010000 CALL 0276D250
0276D0E5      8BC8     MOV ECX,EAX
0276D0E7      85C9     TEST ECX,ECX
0276D0E9      894C24 0C  MOV DWORD PTR SS:[LOCAL.64],ECX
0276D0ED      7D 10     JGE SHORT 0276D0FF
0276D0EF      66:C746 6C F2FF MOV WORD PTR DS:[ESI+6C],0FFF2
0276D0F5      5F      POP EDI
0276D0F6      5E      POP ESI
0276D0F7      5B      POP EBX
0276D0F8      81C4 04010000 ADD ESP,104
0276D0FE      C3      RETN
0276D0FF      A1 30437802 MOV EAX,DWORD PTR DS:[2784330]

```

Estamos en el punto de entrada de la función Open. Si buscamos en el manual de usuario vemos los parámetros de entrada y el output de dicha función:

Input	NET_COMMAND LABEL PASSWORD	'O' Label Password
Output	NET_PASS DATA[0] STATUS	Net Password Protocol type used: = 0, LOCAL = 2, ANP = 3, TCPIP Status ==0 Success !=0 Error

Vemos que devuelve 3 valores: el NET_PASS, el protocolo usado y el status. Nosotros haremos la emulación de manera que el campo status que se devuelva sea 0, los otros 2 no los cambiamos ya que en este programa no se utilizan (esto lo he comprobado poniendo un BPM en la zona de memoria en donde se pondrían, y el programa no accede a esa zona):

```

0276D0C0 Open 81EC 04010000 SUB ESP,104
0276D0C6      53      PUSH EBX
0276D0C7      56      PUSH ESI
0276D0C8      8BB424 10010000 MOV ESI,DWORD PTR SS:[ESP+110]
0276D0CF      57      PUSH EDI
0276D0D0      68 EC8F7702 PUSH OFFSET skeyLink.02778FFC
0276D0D5      56      PUSH ESI
0276D0D6      33DB     XOR EBX,EBX
0276D0D8      E8 5383FFFF CALL 02765430
0276D0DD      83C4 08     ADD ESP,8
0276D0E0      E8 6B010000 CALL 0276D250
0276D0E5      8BC8     MOV ECX,EAX
0276D0E7      85C9     TEST ECX,ECX
0276D0E9      894C24 0C  MOV DWORD PTR SS:[LOCAL.64],ECX
0276D0ED      90      NOP
0276D0EE      90      NOP
0276D0EF      66:C746 6C 0000 MOV WORD PTR DS:[ESI+6C],0
0276D0F5      5F      POP EDI
0276D0F6      5E      POP ESI
0276D0F7      5B      POP EBX
0276D0F8      81C4 04010000 ADD ESP,104
0276D0FE      C3      RETN

```

Con esto queda solucionada la primera llamada a la mochila.

CracksLatinoS! 2010

._** Emulación del segundo comando **. _

Si pulsamos F9 paramos de nuevo en 276CB20, y esta vez con el NET_COMMAND "A" (Access Mode) y haciendo una llamada a la función Read (R), y si llegamos de nuevo hasta el salto que hay en 276CFBA nos llevará hasta 276CFD1 CALL 276D370:

```
0276CFBA > FF248D 0CD07602 JMP D0000 PTR DS:[ECX*4+276D00C]
0276CFC1 > 56 PUSH ESI
0276CFC2 > E8 F9000000 CALL Open
0276CFC7 > 66:8B46 6C MOV AX,WORD PTR DS:[ESI+6C]
0276CFCE > 83C4 04 ADD ESP,4
0276CFD0 > 5E POP ESI
0276CFD1 > C3 RETN
0276CFD6 > 56 PUSH ESI
0276CFD7 > E8 9A030000 CALL Access
0276CFDE > 66:8B46 6C MOV AX,WORD PTR DS:[ESI+6C]
0276CFE0 > 83C4 04 ADD ESP,4
0276CFE2 > 5E POP ESI
0276CFE4 > C3 RETN
```

Address	Hex dump	ASCII
0012EC1C	41 00 00 00 00 00 00 00 72 00 67 65 6F 26 73 6F	A.....r.geo&so
0012EC2C	66 74 00 00 00 00 00 00 6D 6F 73 73 2E 6B	ft.....moss.k
0012EC3C	61 74 00 00 00 00 00 00 00 00 00 00 00 00	at.....
0012EC4C	00 00 00 00 00 00 00 00 00 00 00 00 00 00

La llamada al comando "A" requiere los siguientes inputs y devuelve:

Input	NET_COMMAND NET_PASS COMMAND	'A' Net Password Standard command
Output	STATUS	Status ==0 Success !=0 Error

El parámetro NET_PASS lo obtendría del resultado del comando Open ejecutado anteriormente, y el parámetro COMMAND es el comando standard de la mochila que queremos ejecutar, en este caso es el comando Reading "R".

O sea que habilita el acceso a la mochila para ejecutar el comando "r" (Reading). Veamos qué parámetros requiere el comando "R":

Input	NET_COMMAND LPT LABEL PASSWORD	'R' Port label password
Output	DATA EXT_DATA FAIL_CTR STATUS	Read data Read Extended data Fail Counter Status ==0 Success !=0 Error

Vemos que al comando "R" se le pasa como parámetros el puerto en el que se encuentra la mochila, el label y la contraseña y devuelve en el campo "data" los datos que lee de la llave. Además si la función tiene éxito devuelve en el campo "Status" el valor 0.

Lo que tendremos que hacer será un injerto tal que deje status= 0 y además deje escrito en la zona data de la estructura unos datos cualesquiera. Después investigaremos qué datos debe contener para que funcione:

De momento, lo que haremos será una modificación de manera que se nos devuelva 0 en el campo status, y cambiaremos el contenido del campo data a mano. Cuando sepamos qué debe haber en ese campo haremos el injerto definitivo.

CracksLatinoS! 2010

Ahora entramos con F7 en la función 276D370 y lo modificamos de la siguiente manera:

```

0276D370 Access 81EC 00010000 SUB ESP,100
0276D376 53 PUSH EBX
0276D377 56 PUSH ESI
0276D378 8BB424 0C010000 MOV ESI,DWORD PTR SS:[ARG.1]
0276D37F 68 58907702 PUSH OFFSET skeyLink.02779058
0276D384 56 PUSH ESI
0276D385 E8 A680FFFF CALL 02765430
0276D38A 8B46 02 MOV EAX,DWORD PTR DS:[ESI+2]
0276D38D 50 PUSH EAX
0276D38E E8 4D010000 CALL 0276D4E0
0276D393 8BD8 MOV EBX,EAX
0276D395 83C4 0C ADD ESP,0C
0276D398 85DB TEST EBX,EBX
0276D39A 90 NOP
0276D39B 90 NOP
0276D39C 66:C746 6C 0000 MOV WORD PTR DS:[ESI+6C],0
0276D3A2 5E POP ESI
0276D3A3 5B POP EBX
0276D3A4 81C4 00010000 ADD ESP,100
0276D3AA C3 RETN
0276D3AB 66:8A4F 08 MOV CX,WORD PTR DS:[ESI+8]
  
```

De esta manera nos devolverá 0 en el campo status y si seguimos con F8 hasta llegar al RET, entonces modificaremos a mano el campo data y lo dejaremos de la siguiente manera, y le pondremos un BPM:

Address	Hex dump	ASCII
0012EC1C	41 00 00 00 00 00 00 00 72 00 67 65 6F 26 73 6F	A.....r.geo&so
0012EC2C	66 74 00 00 00 00 00 00 00 00 6D 6F 73 73 2E 6B	ft.....moss.k
0012EC3C	61 74 00 00 00 00 00 00 00 00 01 02 03 04 05 06	at.....00000000
0012EC4C	07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16	-0...00000000
0012EC5C	17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26	+1++L#A!'"##\$%&
0012EC6C	27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36	'()*+,-./0123456
0012EC7C	37 38 39 3A 3B 3C 3D 3E 3F 40 00 00 00 00 00 00	789:;<=?0.....
0012EC8C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Podemos ver que el campo “status” (en 12EC88) está a cero, y que tenemos relleno el campo “data” con los valores que se supone que hemos leído de la mochila. De esta manera es más fácil cuando pare por leer esa porción de memoria saber qué posición de memoria está leyendo.

Ahora ya podemos pulsar F9 para que el programa continúe y enseguida para por el BPM al leer el primer byte de los que hemos puesto en el campo data.

```

00636408 8A1401 MOV DL,BYTE PTR DS:[EAX+ECX]
0063640B 8D45 DC LEA EAX,[EBP-24]
0063640E 52 PUSH EDX
0063640F 50 PUSH EAX
00636410 FF15 18124000 CALL DWORD PTR DS:[<&MSUBUM60._vbaVarCat>]
00636416 8D4D BC LEA ECX,[EBP-44]
00636419 8D55 DC LEA EDX,[EBP-24]
0063641C 51 PUSH ECX
0063641D 8D45 CC LEA EAX,[EBP-34]
00636420 52 PUSH EDX
00636421 50 PUSH EAX
00636422 FF15 48124000 CALL DWORD PTR DS:[<&MSUBUM60._vbaVarCat>]
00636428 50 PUSH EAX
00636429 FF15 54104000 CALL DWORD PTR DS:[<&MSUBUM60._vbaStrUvarMove>]
0063642F 8BD0 MOV EDX,EAX
00636431 8BCB MOV ECX,EBX
00636433 FF15 24134000 CALL DWORD PTR DS:[<&MSUBUM60._vbaStrUvarMove>]
00636439 8D4D CC LEA ECX,[EBP-34]
0063643C 8D55 DC LEA EDX,[EBP-24]
0063643F 51 PUSH ECX
00636440 52 PUSH EDX
00636441 6A 02 PUSH 2
00636443 FF15 60104000 CALL DWORD PTR DS:[<&MSUBUM60._vbaFreeVarLst>]
00636449 B8 01000000 MOV EAX,1
0063644E 83C4 0C ADD ESP,0C
00636451 66:03C6 ADD AX,SI
00636454 70 36 J0 SHORT 0063648C
00636456 8BF0 MOV ESI,EAX
00636458 E9 46FFFFFF JMP 006363A3
0063645D 68 79646300 PUSH cecap32.00636479
00636462 74 14 J4 SHORT 00636478
Stack [0012EC46]=01
DL=00
  
```

Estamos inmersos en un bucle que lo que está haciendo es copiar todo el campo “data” en otra dirección de memoria (), pero intercalando 00 entre cada byte, una especie de formato ANSI:

Así cuando llegamos a salir del bucle en 63645D tenemos en 157D00C lo que hemos dicho:

CracksLatinoS! 2010

Address	Hex dump
0157D00C	01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00
0157D01C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00
0157D02C	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00
0157D03C	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00
0157D04C	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00
0157D05C	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00
0157D06C	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00
0157D07C	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00
0157D08C	00 00 54 00 45 00 00 00 44 00 41 00 4C 00 20 00
0157D09C	53 00 4F 00 52 00 47 00 45 00 4E 00 54 00 45 00

Le ponemos un BPM en la zona que vemos y ahora podemos pulsar F9 para continuar, y vuelve a parar por el último BPM que hemos puesto en:

770F4C85	F3:A5	REP MOVSD	DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
770F4CA7	8BCA	MOV	ECX,EDX
770F4CA9	83E1 03	AND	ECX,00000003
770F4CAC	F3:A4	REP MOVSD	BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
770F4CAE	5F	POP	EDI
770F4CAF	EB BF	JMP	SHORT 770F4C70
770F4CB1	33C0	XOR	EAX,EAX
770F4CB3	EB C9	JMP	SHORT 770F4C7E
770F4CB5	90	NOB	

ECX=00000018 (decimal 24.)
[017D00BC]=00120011
[01729C7C]=00120011

Donde va copiando todos los valores a otra zona de memoria más (14D0C8C):

Address	Hex dump	ASCII
014D0C8C	01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00	0.0.0.0.0.0.0.0.
014D0C9C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	...0...0.0.0.
014D0CAC	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00	0.0.0.0.0.0.0.0.
014D0CBC	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	0.0.0.0.0.0.0.0.
014D0CCC	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	0.0.0.0.0.0.0.0.
014D0CDC	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	0.0.0.0.0.0.0.0.
014D0CEC	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00	0.0.0.0.0.0.0.0.
014D0CFC	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	0.0.0.0.0.0.0.0.
014D0D0C	00 00 41 00 4C 00 20 00 65 00 78 00 63 00 65 00	..A.L..e.x.c.e.
014D0D1C	73 00 69 00 76 00 61 00 2E 00 00 00 2E 00 00 00	s.i.v.a.....

Le ponemos otro BPM y continuamos y para una vez para hacer un comprobación sin importancia (los gajes de estar en VB) y la siguiente vez que para lo hace para copiar de nuevo los mismos valores en otra zona de memoria (14E43F6):

7710A9B7	C1E9 02	SHR	ECX,2
7710A9B9	F3:A5	REP MOVSD	DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
7710A9BC	8BC8	MOV	ECX,EAX
7710A9BE	83E1 03	AND	ECX,00000003
7710A9C1	F3:A4	REP MOVSD	BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
7710A9C3	33C0	XOR	EAX,EAX
7710A9C5	5F	POP	EDI
7710A9C6	5E	POP	ESI
7710A9C7	5B	POP	EBX
7710A9C8	5D	POP	EBP
7710A9C9	C2 0C00	RET	0C

Por tanto le ponemos también un BPM y seguimos con F9:

Address	Hex dump	ASCII
014E43F6	01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00	0.0.0.0.0.0.0.0.
014E4406	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	...0...0.0.0.
014E4416	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00	0.0.0.0.0.0.0.0.
014E4426	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	0.0.0.0.0.0.0.0.
014E4436	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	0.0.0.0.0.0.0.0.
014E4446	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	0.0.0.0.0.0.0.0.
014E4456	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00	0.0.0.0.0.0.0.0.
014E4466	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	0.0.0.0.0.0.0.0.
014E4476	00 00 55 00 45 00 53 00 54 00 41 00 20 00 56 00	..U.E.S.T.A..U.
014E4486	45 00 52 00 53 00 49 00 4F 00 4E 00 45 00 20 00	E.R.S.I.O.N.E..

Ahora Olly salta cuando va a copiar el último word de este último bloque en otro sitio:

CracksLatinoS! 2010

770F4CAC	F8:A4	REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:
770F4CAE	5F	POP EDI
770F4CAF	EB BF	JMP SHORT 770F4C70
770F4CB1	33C0	XOR EAX,EAX
770F4CB3	EB C9	JMP SHORT 770F4C7E
770F4CB5	90	NOP

ECX=1
[014E4475]=00
[015374FD]=00

Address	Hex dump	ASCII
015374EC	00 00 00 00 07 00 07 00 F8 07 18 02 02 00 00 00°·†00..
015374FC	40 00 00 00 03 00 04 00 05 00 06 00 07 00 08 00	@...·.·.·.·.·.
0153750C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	...·.·.·.·.·.·.

Y si pulsamos F9 de nuevo la siguiente parada la hace para comparar este valor con 0:

7710A80A	33C0	XOR EAX,EAX
7710A80C	66:F3:A7	REPE CMPS WORD PTR DS:[ESI],WORD PTR ES:[EDI]
7710A80F	74 05	JE SHORT 7710A816
7710A811	1BC0	SBB EAX,EAX
7710A813	8308 FF	SBB EAX,-1
7710A816	5F	POP EDI
7710A817	5E	POP ESI
7710A818	5D	POP EBP
7710A819	C2 0C00	RET 0C
7710A81C	F6C3 01	TEST BL,01

ECX=1
[014E974C]=0000
[015374FC]=0040

Address	Hex dump	ASCII
015374FC	40 00 00 00 03 00 04 00 05 00 06 00 07 00 08 00	@...·.·.·.·.·.
0153750C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	...·.·.·.·.·.·.

Esto parece interesante, vamos a continuar traceando con F8 hasta llegar al código del programa, y aparecemos aquí:

0063C450	50	PUSH EAX	Arg1
0063C451	FF15 80114001	CALL DWORD PTR DS:[<&MSUBUM60.__vbaStrCmp>]	MSUBUM60.__vbaStrCmp
0063C457	8BD8	MOV EBX,EAX	
0063C459	8D4D 80	LEA ECX,[EBP-80]	
0063C45C	F7DB	NEG EBX	Converts EBX to boolean
0063C45E	8D55 84	LEA EDX,[EBP-7C]	
0063C461	51	PUSH ECX	
0063C462	1BD8	SBB EBX,EBX	
0063C464	8D45 88	LEA EAX,[EBP-78]	
0063C467	52	PUSH EDX	
0063C468	43	INC EBX	Arg3
0063C469	50	PUSH EAX	Arg2
0063C46A	6A 03	PUSH 3	Arg1 = 3
0063C46C	F7DB	NEG EBX	
0063C46E	FF15 AC124001	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeStrList>]	MSUBUM60.__vbaFreeStrList
0063C474	83C4 10	ADD ESP,10	
0063C477	8D8D 64FFFFFF	LEA ECX,[EBP-9C]	
0063C47D	FF15 3C104001	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeVar>]	MSUBUM60.__vbaFreeVar
0063C483	66:85DB	TEST BX,BX	
0063C486	0F84 80000000	JE 0063C50C	
0063C48C	8D4D AC	LEA ECX,[EBP-54]	
0063C48F	8D95 64FFFFFF	LEA EDX,[EBP-9C]	
0063C495	51	PUSH ECX	Arg2

Vemos que acabamos de salir de un StrCmp y más abajo en 63C486 hay un salto condicional que depende del resultado de esa comparación. Como no sabemos si lo que hay que hacer es saltar o no saltar vamos a dejar esto así y ver donde nos lleva el flujo de ejecución. Así que vamos a dar a F9 a ver qué pasa. Llegamos a otra zona en la que se va copiando de nuevo la misma serie de bytes en una zona que ya teníamos puesto un BPM, pero desplazada 2 bytes:

CracksLatinoS! 2010

770F4CA3	8BF8	--	MOV	EDI,EAX
770F4CA5	F8:A5		REP MOVSD	DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
770F4CA7	8BCA		MOV	ECX,EDX
770F4CA9	83E1 03		AND	ECX,00000003
770F4CAC	F8:A4		REP MOVSB	BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
770F4CAE	5F		POP	EDI
770F4CAF	EB BF		JMP	SHORT 770F4C70
770F4CB1	33C0		XOR	EAX,EAX
770F4CB3	EB C9		JMP	SHORT 770F4C7E
770F4CB5	90		NOP	

ECX=0000001C (decimal 28.)
 [0157D01C]=000A0009
 [014E4404]=00090008

Address	Hex dump	ASCII
014E43F4	01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00	0.0.0.0.0.0.0.0.
014E4404	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	1.2.3.4.5.6.7.8.
014E4414	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00	9.0.1.2.3.4.5.6.7.
014E4424	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	8.9.0.1.2.3.4.5.6.7.
014E4434	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	0.1.2.3.4.5.6.7.8.
014E4444	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	9.0.1.2.3.4.5.6.7.
014E4454	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00	0.1.2.3.4.5.6.7.8.
014E4464	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	9.0.1.2.3.4.5.6.7.
014E4474	41 00 42 00 43 00 44 00 45 00 46 00 47 00 48 00	0.1.2.3.4.5.6.7.8.
014E4484	49 00 4A 00 4B 00 4C 00 4D 00 4E 00 4F 00 50 00	9.0.1.2.3.4.5.6.7.

Así que ahora ponemos el BPM de manera que abarque todos los bytes:

Address	Hex dump	ASCII
014E43F4	01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00	0.0.0.0.0.0.0.0.
014E4404	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	1.2.3.4.5.6.7.8.
014E4414	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00	9.0.1.2.3.4.5.6.7.
014E4424	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	0.1.2.3.4.5.6.7.8.
014E4434	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	9.0.1.2.3.4.5.6.7.
014E4444	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	0.1.2.3.4.5.6.7.8.
014E4454	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00	9.0.1.2.3.4.5.6.7.
014E4464	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	0.1.2.3.4.5.6.7.8.
014E4474	41 00 42 00 43 00 44 00 45 00 46 00 47 00 48 00	9.0.1.2.3.4.5.6.7.
014E4484	49 00 4A 00 4B 00 4C 00 4D 00 4E 00 4F 00 50 00	0.1.2.3.4.5.6.7.8.

Y pulsamos de nuevo F9 para continuar, y vuelva a parar para copiar de nuevo todo en 14F337C, en donde volvemos a poner otro BPM:

Address	Hex dump	ASCII
014F337C	01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00	0.0.0.0.0.0.0.0.
014F338C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	1.2.3.4.5.6.7.8.
014F339C	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00	9.0.1.2.3.4.5.6.7.
014F33AC	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	0.1.2.3.4.5.6.7.8.
014F33BC	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	9.0.1.2.3.4.5.6.7.
014F33CC	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	0.1.2.3.4.5.6.7.8.
014F33DC	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00	9.0.1.2.3.4.5.6.7.
014F33EC	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	0.1.2.3.4.5.6.7.8.
014F33FC	00 00 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA
014F340C	AB AB AB AB EE FE EE FE EE FE EE FE EE FE EE FE

Si pulsamos F9 ahora, viene realmente la parte interesante:

770F4CAC	F8:A4		REP MOVSB	BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
770F4CAE	5F		POP	EDI
770F4CAF	EB BF		JMP	SHORT 770F4C70
770F4CB1	33C0		XOR	EAX,EAX
770F4CB3	EB C9		JMP	SHORT 770F4C7E
770F4CB5	90		NOP	
770F4CB6	90		NOP	
770F4CB7	90		NOP	

ECX=2
 [014F337C]=01
 [015374FC]=40 ('@')

Address	Hex dump	ASCII
015374FC	40 00 00 00 03 00 04 00 05 00 06 00 07 00 08 00	@.....
0153750C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	1.2.3.4.5.6.7.8.
0153751C	AB AB AB AB 00 00 00 00 00 00 00 00 0F 00 07 00

Se copia los 2 primeros bytes del bloque de memoria 14F337C y los copia en 15374FC donde ya teníamos puesto un BPM anteriormente, ahora ahí tendremos:

Address	Hex dump
015374FC	01 00 00 00 03 00 04 00 05 00 06 00 07 00 08 00
0153750C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00
0153751C	AB AB AB AB 00 00 00 00 00 00 00 00 0F 00 07 00

Si salimos con F8 hasta el código del programa aparecemos aquí:

CracksLatinoS! 2010

0063CF08	6A 01	PUSH 1	Arg2 = 1
0063CF0A	50	PUSH EAX	Arg1
0063CF0B	FF15 10134000	CALL DWORD PTR DS:[<&MSUBUM60.#616>]	MSUBUM60.rtcLeftCharBstr
0063CF0D	8B00	MOV EDX, EAX	
0063CF0E	8D4D 88	LEA ECX, [EBP-78]	
0063CF0F	FFD6	CALL ESI	
0063CF10	50	PUSH EAX	Arg2
0063CF11	68 FC794800	PUSH cecap32.004879FC	Arg1 = cecap32.4879FC
0063CF12	FF15 80114000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaStrCmp>]	MSUBUM60.__vbaStrCmp
0063CF14	8B08	MOV EBX, EAX	
0063CF15	8D4D 88	LEA ECX, [EBP-78]	
0063CF16	F7DB	NEG EBX	Converts EBX to boolean
0063CF17	1BDB	SBB EBX, EBX	
0063CF18	F7DB	NEG EBX	
0063CF19	F7DB	NEG EBX	
0063CF1A	FF15 68134000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeStr>]	MSUBUM60.__vbaFreeStr
0063CF1B	66:85DB	TEST BX, BX	
0063CF1C	74 1A	JE SHORT 0063D016	
0063CF1D	8B4D BC	MOV ECX, DWORD PTR SS:[EBP-44]	

Vemos que 5 líneas más abajo va a ejecutarse un StrCmp, con lo que si llegamos hasta ahí con F8 podremos ver qué va a comparar:

0063CF0B	FF15 10134000	CALL DWORD PTR DS:[<&MSUBUM60.#616>]	MSUBUM60.rtcLeftCharBstr
0063CF0D	8B00	MOV EDX, EAX	
0063CF0E	8D4D 88	LEA ECX, [EBP-78]	
0063CF0F	FFD6	CALL ESI	
0063CF10	50	PUSH EAX	Arg2
0063CF11	68 FC794800	PUSH cecap32.004879FC	Arg1 = cecap32.4879FC
0063CF12	FF15 80114000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaStrCmp>]	MSUBUM60.__vbaStrCmp
0063CF14	8B08	MOV EBX, EAX	
0063CF15	8D4D 88	LEA ECX, [EBP-78]	
0063CF16	F7DB	NEG EBX	Converts EBX to boolean
0063CF17	1BDB	SBB EBX, EBX	
0063CF18	F7DB	NEG EBX	
0063CF19	F7DB	NEG EBX	
0063CF1A	FF15 68134000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeStr>]	MSUBUM60.__vbaFreeStr
0063CF1B	66:85DB	TEST BX, BX	
0063CF1C	74 1A	JE SHORT 0063D016	
0063CF1D	8B4D BC	MOV ECX, DWORD PTR SS:[EBP-44]	

[00401180]=734893DA (MSUBUM60.__vbaStrCmp)

Address	Hex dump	ASCII	0012EE1C	004879FC	*vH.
004879FC	2C 00 00 00 12 00 00 00 20 00 20 00 20 00 20 00	...+... ..	0012EE20	015374FC	*tS0
00487A0C	20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00	0012EE24	0012F264	d=.
00487A2C	20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00	0012FF2A	004A60AA	A'.

Vemos que va a comparar lo que teníamos almacenado en 15374FC, que es el primer byte que supusimos que había en la memoria de la pastilla, con 2C. En este caso parece lógico pensar que ese primer valor sí que debe ser 2C, en lugar del 01 que teníamos puesto, así que de momento vamos a cambiar el "01" que hay en 15374FC por un "2C" y veremos qué pasa (observad que 2C en ASCII se corresponde con la coma ",":

Address	Hex dump
015374FC	2C 00 00 00 03 00 04 00 05 00 06 00 07 00 08 00
0153750C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00
0153751C	AB AB AB AB 00 00 00 00 00 00 00 00 0F 00 07 00
0153752C	C3 07 18 02 3C 00 00 00 43 00 0F 00 6E 00 74 00

Ahora sigamos traceando con F8 y al pasar por el StrCmp salta por el BPM, así que seguiremos con F8 hasta salir al código del programa justo en 63CFE4, y ahí continuamos con F8, y cuando pasamos con F8 por encima de 63D022:

0063D01F	6A 01	PUSH 1	Arg2 = 1
0063D021	52	PUSH EDX	Arg1
0063D022	FF15 28134000	CALL DWORD PTR DS:[<&MSUBUM60.#616>]	MSUBUM60.rtcRightCharBstr
0063D024	8B00	MOV EDX, EAX	
0063D025	8D4D 88	LEA ECX, [EBP-78]	
0063D026	FFD6	CALL ESI	
0063D027	50	PUSH EAX	Arg2
0063D028	68 FC794800	PUSH cecap32.004879FC	Arg1 = cecap32.4879FC
0063D029	FF15 80114000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaStrCmp>]	MSUBUM60.__vbaStrCmp
0063D02B	8B08	MOV EBX, EAX	
0063D02C	8D4D 88	LEA ECX, [EBP-78]	
0063D02D	F7DB	NEG EBX	Converts EBX to boolean
0063D02E	1BDB	SBB EBX, EBX	
0063D02F	F7DB	NEG EBX	
0063D030	F7DB	NEG EBX	
0063D031	FF15 68134000	CALL DWORD PTR DS:[<&MSUBUM60.__vbaFreeStr>]	MSUBUM60.__vbaFreeStr
0063D032	66:85DB	TEST BX, BX	
0063D033	74 12	JE SHORT 0063D065	
0063D034	8B45 BC	MOV EAX, DWORD PTR SS:[EBP-44]	
0063D035	50	PUSH EAX	

[00401328]=73486CE2 (MSUBUM60.rtcRightCharBstr)

Address	Hex dump	ASCII	0012EE1C	014F337C	1300
014F337C	01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00	0.0.0.0.0.0.0.0.	0012EE20	00000001	0...
014F338C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	...0.0.0.0.0.0.0.	0012EE24	0012F264	d=.
014F339C	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00	1.2.3.4.5.6.7.8.	0012EE28	004060A0	A'0.
014F33AC	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	9.10.11.12.13.14.15.16.	0012EE2C	00639167	gac.
014F33BC	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	17.18.19.20.21.22.23.24.	0012EE30	0000010B	0B..
014F33CC	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	25.26.27.28.29.30.31.32.	0012EE34	0012EE9C	E'.
014F33DC	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00	33.34.35.36.37.38.39.40.	0012EE38	00F51FC4	-V\$.
014F33EC	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	41.42.43.44.45.46.47.48.	0012EE3C	0012EF60	''\$.
014F33FC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012EE40	00000000	0...
			0012FF44	00F77777	A'.

RETURN from MS!

ASCII "6f@"

CracksLatinoS! 2010

Salta por el BPM en 14F33FA que copiará el valor 40 en 15374FC:

Address	Hex	dump
015374FC	40 00	00 00 03 00 04 00 05 00 06 00 07 00 08 00
0153750C	09 00	0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00
0153751C	AB AB	AB AB 00 00 00 00 00 00 00 00 0F 00 07 00

Mantenemos el BPM que ya teníamos puesto en esa posición y seguimos con F8 hasta volver a 63D028, y si seguimos con F8 hasta el StrCmp que hay en 63D035 vemos que compara ese 40 de nuevo con el valor 2C “,”. Por tanto parece ser que la cadena que estamos buscando debe empezar y terminar con comas “,”. Así que vamos a cambiar también a mano el 40 por 2C antes de que se ejecute ese StrCmp:

Address	Hex dump
015374FC	2C 00 00 00 03 00 04 00 05 00 06 00 07 00 08 00
0153750C	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00
0153751C	AB AB AB AB 00 00 00 00 00 00 00 00 00 00 00 00
0153752C	C3 07 18 02 3C 00 00 00 43 00 6F 00 6E 00 74 00

Ahora sigamos con F8 hasta que volvemos al código y si seguimos con F8, cuando llegamos a 63D08A y pulsar F8 salta para escribir en 15374FC, por lo que podemos quitar ese BPM, y seguir con F8 hasta que llegamos a 63D0C1:

0063D0C1	8B3D 8412400	MOV EDI,DWORD PTR DS:[&MSUBUM60.__vbaInStr>]	
0063D0C7	50	PUSH EAX	Arg2
0063D0C8	6A 00	PUSH 0	Arg1 = 0
0063D0CA	FFD7	CALL EDI	MSUBUM60.__vbaInStr
0063D0CC	8B4D BC	MOV ECX,DWORD PTR SS:[EBP-44]	
0063D0CF	33DB	XOR EBX,EBX	
0063D0D1	85C0	TEST EAX,EAX	
0063D0D3	0F9FC3	SETG BL	
0063D0D6	6A 01	PUSH 1	
0063D0D8	51	PUSH ECX	Arg1
0063D0D9	F7DB	NEG EBX	
0063D0DE	FF15 8010400	CALL DWORD PTR DS:[&MSUBUM60.#517>]	MSUBUM60.rtcLowerCaseBstr
0063D0E1	8B00	MOV EAX,0	

Address	Hex dump	0012EE14	00000000	...	Arg1 = 0
014F337C	01 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00	0012EE18	0152963C	Arg0	Arg2 = UNICODE ",,,"
014F3380	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	0012EE1C	014F3378	1300	Arg3 = 14F337C
014F3384	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00	0012EE20	00000001	0...	Arg4 = 1
014F3388	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	0012EE24		d=.	
014F338C	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	0012EE28	00406000	0...	
014F3390	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	0012EE2C	00639167	g=.	RETURN from MSUBUM60...
014F3394	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00	0012EE30	00000000	0...	
014F3398	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	0012EE34	0012EE34	0...	
014F33FC	00 00 AD BA 00 F0 AD BA 00 F0 AD BA AB AB AB AB	0012EE38	00F51F4C	0...	
014F340C	EE FE EE FE EE FE EE FE EE FE EE FE EE FE EE FE EE	0012EE3C	0012EF60	0...	ASCII "%f@"
		0012EE40	00000000	0...	

Ahí vemos que se va a buscar en qué posición de la cadena que hay en 14F337C se encuentra la cadena "7," y en 63D0D1 si la ha encontrado entonces en EAX habrá un número distinto de cero, esto hará que SETG BL ponga en BL en valor 1. Así que lo que parece es que en el interior de esa cadena que tenemos debe estar la cadena "7," como sabemos de antes que nuestra cadena debe empezar y terminar por "," vamos a modificar lo que hay a partir de 14F337C por:

Address	Hex	dump	ASCII
014F337C	20 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00		. @
014F3380	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	
014F339C	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00	
014F33A0	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	
014F33BC	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	
014F33C0	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	
014F33DC	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00		1 2 3 4 5 6 7 8 .
014F33E0	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00		9 . : ; < = . , -
014F33FC	00 00 AD BA 00 F0 AD BA 00 F0 AD BA AB AB AB AB	
014F3400	AB AB AB AB EE FE EE FE EE FE EE FE 00 00 00 00	

Y al pasar con F8 por encima de InStr salta por el BPM. Como salta muchas veces vamos a deshabilitar este BPM y ejecutar con F8 hasta volver al código del programa en 63D0CC y una vez ahí, volvemos a activar ese BPM, y continuamos con F8 hasta que llegamos a 63D0DB donde va a pasar a minúsculas la cadena que tenemos en 14F337C:

CracksLatinoS! 2010

0063D008	51	PUSH ECX	[Arg1 MSUBUM60.rtcLowerCaseBstr
0063D009	F7D8	NEG EBX	
0063D00B	FF15 8010400	CALL DWORD PTR DS:[<&MSUBUM60.#517>]	UNICODE ",educational,"
0063D00E1	8B00	MOV EDX, EAX	
0063D00E3	808D 74FFFFFF	LEA ECX, [EBP-8C]	UNICODE ",universale,"
0063D00E9	FFD6	CALL ESI	
0063D00EB	50	PUSH EAX	[Arg1 MSUBUM60.rtcLowerCaseBstr
0063D00EC	68 105B4900	PUSH cecap32.00495B10	
0063D00F1	6A 00	PUSH 0	UNICODE ",educational,"
0063D00F3	FFD7	CALL EDI	
0063D00F5	30D2	XOR EDX, EDX	UNICODE ",universale,"
0063D00F7	6A 01	PUSH 1	
0063D00F9	85C0	TEST EAX, EAX	[Arg1 MSUBUM60.rtcLowerCaseBstr
0063D00FB	8B45 BC	MOV EAX, DWORD PTR SS:[EBP-44]	
0063D00FE	0F9FC2	SETG DL	UNICODE ",educational,"
0063D101	F7DA	NEG EDX	
0063D103	50	PUSH EAX	[Arg1 MSUBUM60.rtcLowerCaseBstr
0063D104	0BD0	OR EBX, EDX	
0063D106	FF15 8010400	CALL DWORD PTR DS:[<&MSUBUM60.#517>]	UNICODE ",educational,"
0063D10C	8B00	MOV EDX, EAX	
0063D108E	808D 78FFFFFF	LEA ECX, [EBP-88]	UNICODE ",universale,"
0063D114	FFD6	CALL ESI	
0063D116	50	PUSH EAX	[Arg1 MSUBUM60.rtcLowerCaseBstr
0063D117	68 88594900	PUSH cecap32.00495988	
0063D11C	6A 00	PUSH 0	UNICODE ",educational,"
0063D11E	FFD7	CALL EDI	
0063D120	33C9	XOR ECX, ECX	

[00401080]=734873AD (MSUBUM60.rtcLowerCaseBstr)

Address	Hex	dump	ASCII
014F337C	2C 08 82 00 03 08 04 00 05 00 06 00 07 00 08 00	00000001 8...	0012EE1C 014F337C 1300 Arg1 = 14F337C
014F3380	09 00 8A 00 06 08 0C 00 0E 00 0F 00 10 00 00 00	0012EE20 00000001 8...	
014F3384	11 00 14 00 13 00 14 00 15 00 16 00 17 00 18 00	0012EE24 0012F264 d...	
014F3388	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	0012EE28 00406A00 a...	
014F338C	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	0012EE2C 00039167 gac...	RETURN from MSUE
014F3390	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	0012EE30 0000010B b...	
014F3394	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00	0012EE34 0012EE9C k...	
014F3398	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	0012EE38 00F51FC4 -v...	
014F339C	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	0012EE3C 0012EF60 i...	ASCII "'6f'"
014F33A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012EE40 00000000 j...	

Como ahí también para muchas veces por el BPM, lo deshabilitamos y seguimos con F8 hasta 63D0E1 y volvemos a habilitarlo. Entonces vemos que ha copiado esta cadena en 14E43F4 (se supone que en caso de que en la cadena original hubiera alguna letra mayúscula la hubiera pasado a minúscula en la cadena que ha escrito en 14E43F4):

Address	Hex dump																ASCII
014E43F4	2C	00	02	00	03	00	04	00	05	00	06	00	07	00	08	00
014E4400	09	00	0A	00	0B	00	0C	00	0D	00	0E	00	0F	00	10	00
014E4414	11	00	12	00	13	00	14	00	15	00	16	00	17	00	18	00
014E4428	19	00	1A	00	1B	00	1C	00	1D	00	1E	00	1F	00	20	00
014E4434	21	00	22	00	23	00	24	00	25	00	26	00	27	00	28	00
014E4448	29	00	2A	00	2B	00	2C	00	2D	00	2E	00	2F	00	30	00
014E4454	31	00	32	00	33	00	34	00	35	00	36	00	37	00	38	00
014E4468	39	00	3A	00	3B	00	3C	00	3D	00	3E	00	3F	00	40	00
014E4474	00	00	00	00	55	00	45	00	53	00	4A	00	41	00	20	00
014E4488	56	00	45	00	52	00	53	00	49	00	5F	00	4E	00	45	00

Y ahora es cuando viene el meollo. Si llegamos con F8 hasta 63D0F3 vemos que va a comprobar si en nuestra cadena se encuentra la cadena “educational,” y más adelante en 63D11E comprobará si se encuentra la cadena “universal,” y en los dos casos si la encuentra guarda el valor 1 en un registro y luego le hace OR con el valor proveniente de la comparación anterior y guarda el resultado de este OR en EBX.:

006300D8	FF15 80104000	CALL DWORD PTR DS:[&MSUBUM60. #517>]	MSUBUM60.rtcLowerCaseBstr
006300D9	8D00	MOV EDX, EAX	
006300E3	8D80 74FFFFFF	LEA ECX, [EBP-8C]	
006300E9	FFD6	CALL ESI	
006300EB	50	PUSH EAX	
006300EC	68 105B4900	PUSH cecap32.00495B10	UNICODE ",educational,"
006300F1	6A 00	PUSH 0	
006300F3	FFD7	CALL EDI	MSUBUM60.__vbaInStr
006300F5	33D2	XOR EDX, EDX	
006300F7	6A 01	PUSH 1	
006300F9	85C0	TEST EAX, EAX	
006300FB	8B45 BC	MOV EAX, DWORD PTR SS:[EBP-44]	
006300FE	8F9FC2	SETG DL	
00630101	F7D8	NEG EDX	
00630103	50	PUSH EAX	[Arg1
00630104	0BD8	OR EBX, EDX	MSUBUM60.rtcLowerCaseBstr
00630106	FF15 80104000	CALL DWORD PTR DS:[&MSUBUM60. #517>]	
0063010C	8D00	MOV EDX, EAX	
0063010E	8D80 78FFFFFF	LEA ECX, [EBP-88]	
00630114	FFD6	CALL ESI	
00630116	50	PUSH EAX	
00630117	68 88594900	PUSH cecap32.00495988	UNICODE ",universale,"
0063011C	6A 00	PUSH 0	
0063011E	FFD7	CALL EDI	
00630120	33C9	XOR ECX, ECX	
EDI=733AA27E (MSUBUM60.__vbaInStr)			

Address	Hex dump	001EE14	00000000	...	Arg1 = 0
014E43F4	2C 00 00 02 00 03 00 04 00 05 00 06 00 07 00 08 00	001EE140	00495B10	↑↑↑	Arg2 = UNICODE "", educational, "
014E4400	09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00 10 00	001EE1C0	014E43F4	↑CND	Arg3 = 14E43F4
014E4410	11 00 12 00 13 00 14 00 15 00 16 00 17 00 18 00	001EE200	00000000	...	Arg4 = 1
014E4420	19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00 20 00	001EE240	0012F264	d-c-	
014E4430	21 00 22 00 23 00 24 00 25 00 26 00 27 00 28 00	001EE280	004060A0	â'0-	
014E4440	29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00 30 00	001EE2C0	00039167	gac-	RETURN from MSUBUM0. __vbaFree0b,
014E4450	31 00 32 00 33 00 34 00 35 00 36 00 37 00 38 00	001EE300	0000010B	0B-	
014E4460	39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00 40 00	001EE340	0012FF9C	ç-	
014E4470	41 00 42 00 43 00 44 00 45 00 46 00 47 00 48 00	001EE380	00F51FC4	-7S-	
014E4480	49 00 4A 00 4B 00 4C 00 4D 00 4E 00 4F 00 50 00	001EE3C0	0012FF60	↑S-	ASCII "6f0"

CracksLatinoS! 2010

Si miramos más adelante dependiendo de si EBX es 0 o no el salto que hay en 63D15A se tomará o no:

0063D147	804D 88	LEA ECX, [EBP-78]	
0063D14A	50	PUSH EAX	
0063D14B	51	PUSH ECX	
0063D14C	6A 06	PUSH 6	
0063D14E	FF15 AC124000	CALL DWORD PTR DS:[&MSUBUM60.__vbaFree	[Arg3 Arg2 Arg1 = 6 MSUBUM60.__vbaFreeStrList
0063D154	83C4 1C	ADD ESP, 1C	
0063D157	66:85DB	TEST BX, BX	
0063D15A	0F84 42010000	JE 0063D2A2	
0063D160	6A 00	PUSH 0	
0063D162	6A 03	PUSH 3	
0063D164	6A 01	PUSH 1	

Podemos entonces concluir que con que se cumpla una de estas 3 condiciones el salto no se tomará. Si me dieran a elegir entre una de las 3 opciones yo escogería la opción de poner “universale,” porque da a entender que esto no va a llevar ningún tipo de limitación.

Así que ya sabemos que debemos modificar la función Access de manera que nos devuelva en el campo data la cadena “universale,” y en el campo status el valor 0. Podría hacerse así:

0289D370	81EC 00010000	SUB ESP, 100	
0289D376	53	PUSH EBX	
0289D377	56	PUSH ESI	
0289D378	8BB424 0C010000	MOV ESI, DWORD PTR SS:[ESP+10C]	
0289D37F	68 58908A02	PUSH OFFSET keylink.028A9058	ASCII "Enter access call"
0289D384	56	PUSH ESI	
0289D385	E8 A680FFFF	CALL 02895430	
0289D38A	8B46 02	MOV EAX, DWORD PTR DS:[ESI+2]	
0289D38D	50	PUSH EAX	[Arg1 keylink.0289D4E0
0289D38E	E8 4D010000	CALL 0289D4E0	
0289D393	8BD8	MOV EBX, EAX	
0289D395	83C4 0C	ADD ESP, 0C	
0289D398	85DB	TEST EBX, EBX	
0289D39A	90	NOP	
0289D39B	90	NOP	
0289D39C	66:C746 6C 00	MOV WORD PTR DS:[ESI+6C], 0	Pone Status a cero
0289D3A2	66:C746 2A 20	MOV WORD PTR DS:[ESI+2A], 752C	Vamos escribiendo
0289D3A8	C746 2C 6E69	MOV DWORD PTR DS:[ESI+2C], 6576696E	la cadena universale
0289D3AF	C746 30 7273	MOV DWORD PTR DS:[ESI+30], 6C617372	en el campo data
0289D3B6	90	NOP	
0289D3B7	66:C746 34 65	MOV WORD PTR DS:[ESI+34], 2C65	
0289D3BD	5E	POP ESI	
0289D3BE	5B	POP EBX	
0289D3BF	81C4 00010000	ADD ESP, 100	
0289D3C5	C3	RETN	
0289D3C6	75 1E	JNE SHORT 0289D3E6	
0289D3C8	8BD3	MOV EDX, EBX	
0289D3CA	C1F2 05	SHI FDX, 5	

Guardamos los cambios (recordemos que estos cambios los estamos haciendo en la librería keylink.dll) y cerramos Olly.

Si arrancamos el programa ya con la dll modificada:



CracksLatinoS! 2010

Vemos que el programa arranca sin que aparezca el cartel de versión DEMO. Así que de nuevo lo hemos conseguido.

.-** 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