



Estudio de una Protección Hardlock por El Cid



Programa	Protegido por Dongle
Download	Web ()
Descripción	
Herramientas	OllyDbg v2.0 y v1.10, ImpRect, Cerebro, Atención.
Dificultad	Media
Compresor	Parece que no
Protección	Dongle tipo Hardlock
Objetivos	Emular algunas de las llamadas al dongle (Servicios)
Fecha:	26/11/2010
Autor	El Cid
Referencias	1. Windows Win32 API reference



Estudio de una Protección Hardlock por El Cid

NOTA PREVIA

Disclaimer

- Este documento puede verse, en una primera lectura, como un sistema de debuggear un programa y del sistema dongle de protección. Ésa es sólo una visión parcial de la realidad. La información contenida en el presente documento, sirve igualmente para establecer medidas anti-debugging más adecuadas y eficaces, para el estudio técnico de los citados sistemas de protección, su funcionalidad y eficacia. Igualmente es un banco de trabajo para el análisis de la estructura de los programas en Windows y un sin fin de aplicaciones adicionales que sería prolijo referir aquí.
- Sin embargo, dado que todo lo anterior solo tiene como objetivo el aprendizaje y la práctica de la programación bajo Windows, y resto de los elementos citados antes, bajo un prisma exclusivamente didáctico y técnico – científico, si pensais utilizar el programa bajo una base comercial, debeis comprarlo y abandonar ahora mismo la lectura del presente documento.
- En absoluto me hago responsable del mal uso que cada cual pueda hacer de la información técnica facilitada.



Estudio de una Protección Hardlock por El Cid

INDICE

1. <u>Consideraciones Previas</u>	4
2. <u>Habilitación</u>	4
3. <u>Parcheo de Servicios</u>	12
4. <u>Zona para el injerto e Injerto</u>	21
5. <u>Epílogo</u>	21
 <u>Anexo 1: Borradores. Análisis previo</u>	 22
<u>Anexo 2: La protección con Dongle: El “Big Picture”</u>	26



Estudio de una Protección Hardlock por El Cid

1. Consideraciones Previas

En los foros suelen plantearse, de manera recurrente, preguntas acerca de los dongles, ya que se tienen por los sistemas más seguros de protección del software.

En esta ocasión, después de ver una pregunta en un foro y contestar algunos e-mail al respecto, como parece que las cosas no terminaban de estar claras, decidí escribir un tuto del software en cuestión.

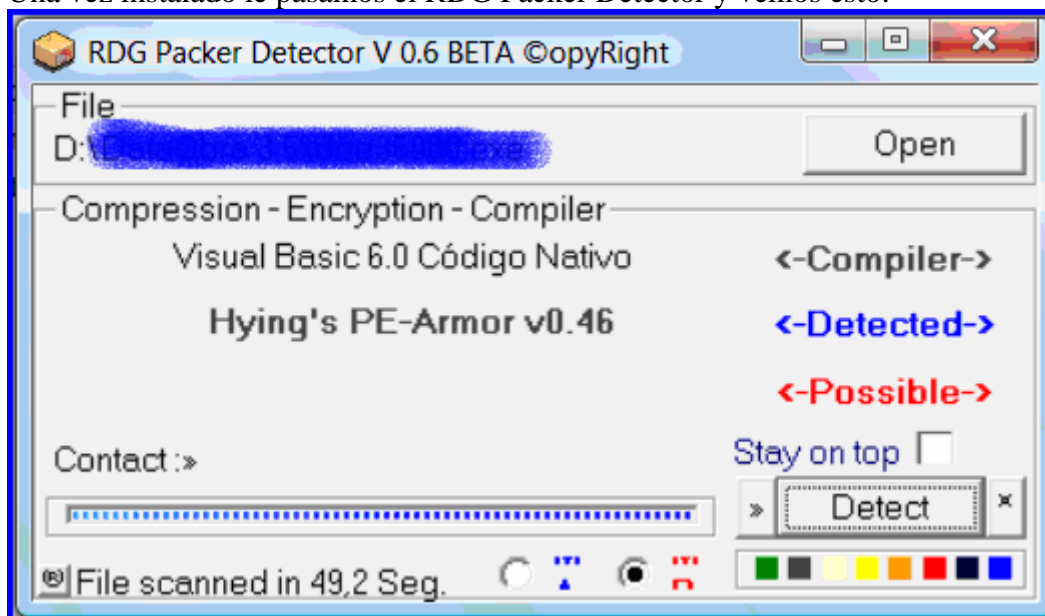
Pero escribir un tuto “como Dios manda”, lleva tiempo que es precisamente de lo que menos se suele disponer. Por éso y para no retrasarlo más, os presento estas líneas que no es lo que me hubiera gustado escribir, pero que siempre es mejor que nada, dado que las cosas cuando pierden actualidad también suelen perder interés, aparte de que se corre el riesgo de que terminen en la lista de “asuntos pendientes”, lo que creo que es todavía peor.

Pido vuestra comprensión e indulgencia y un juicio no demasiado severo.

Y dicho lo anterior en descargo de mi conciencia, vayamos directamente al grano sin más preámbulos.

2. “Habilitación”

Una vez instalado le pasamos el RDG Packer Detector y vemos esto:



O sea que es un VB nativo y además tiene un packer. Uffff muy mal rollito.

Bueno, ejecutamos directamente el progy:

Aparece esta ventana:



Estudio de una Protección Hardlock por El Cid

Bienvenido a [redacted] Por favor complete el formulario de habilitación.

Aceptar

Y tras aceptar, ésta otra:

Habilitación [X]

Paso 1: Ingrese datos del Usuario

Nombre Completo

Profesion

Empresa

E-Mail

Telefono ()

Ciudad

Provincia

Paso 2: Solicite el envio del codigo de habilitacion

Si no dispone de conexion a internet, vea las instrucciones **Habilitacion Manual**

Paso 3: Habilitar

Codigo

Conozca mas sobre [redacted] .com

Donde vemos que tendremos que introducir unos datos y después insertar un código que quizá esté relacionado con ellos.

Ponemos unos datos cualesquiera.



Estudio de una Protección Hardlock por El Cid

Probamos un código cualquiera, por ej.: ZZ (deben ser letras pq no admite números)

Habilitación

Paso 1: Ingrese datos del Usuario

Nombre Completo: pepe perez

Profesion: ingeniero

Empresa: pepe

E-Mail: pepe@pepe.com

Telefono: (11111) 1111111111

Ciudad: Cordoba

Provincia: Cordoba

Paso 2: Solicite el envio del codigo de habilitacion

Solicitar Código Online

Si no dispone de conexion a internet, vea las instrucciones en el archivo **Habilitacion Manu**

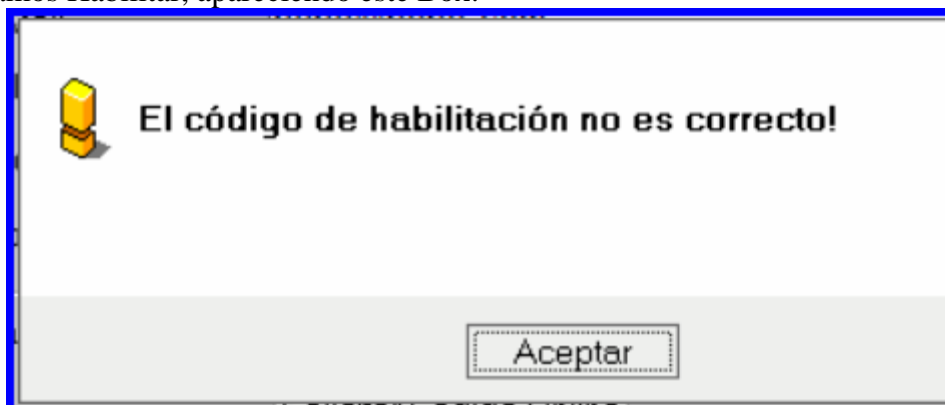
Paso 3: Habilitar

Codigo: ZZ

Habilitar

Conozca mas sobre [redacted] .com

Y pulsamos Habilitar, apareciendo éste Box:



O sea que el ZZ no vale, ¡qué mala suerte! Je, je.

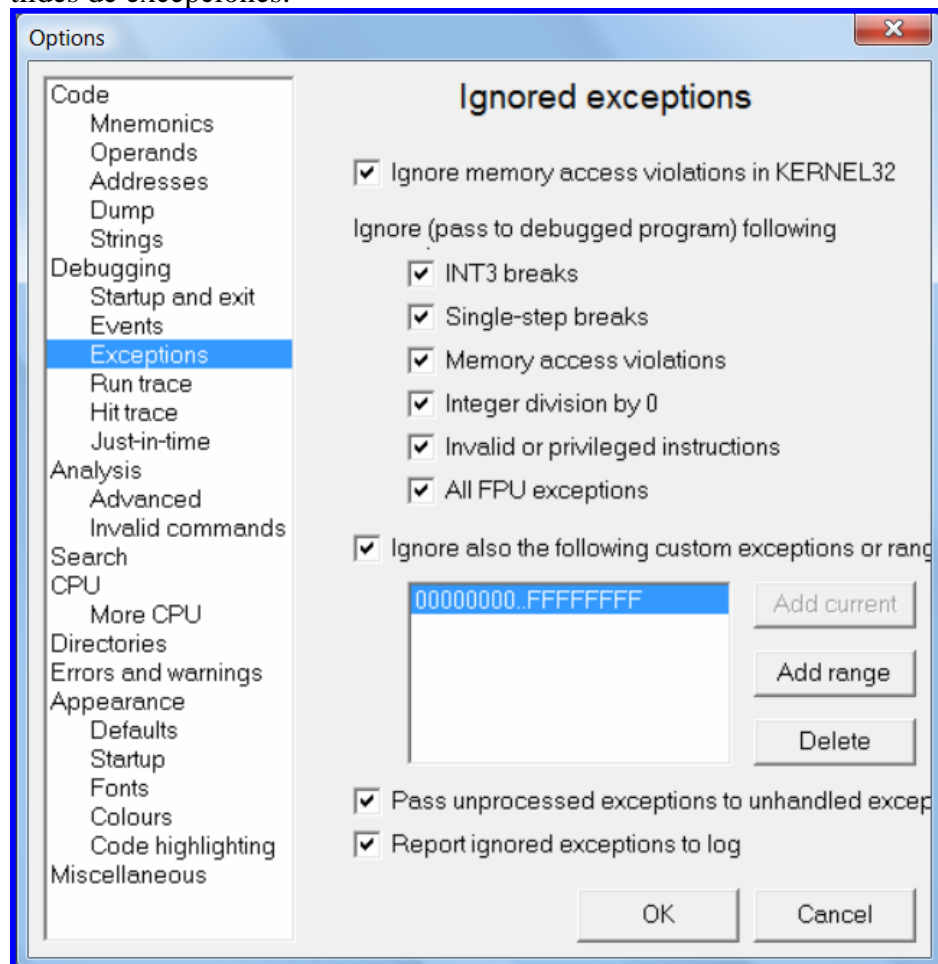
Bien, vamos a cargar el programa en Olly pero antes nos gustaría saber si tiene algún sistema antidebug, para estar sobreaviso.

Yo lo que hago a veces, es cargar el progy en Olly v2.00 (¡¡ o en la v2.01 nueva release!!), ya que como éste no admite plugins, si el progy corre sin problemas en él, será



Estudio de una Protección Hardlock por El Cid

bastante probable que no lleve ningún antidebug. Así que lo hacemos marcando todas las tildes de excepciones:



Os diré que se me ha atascado un par de veces en el análisis inicial pero lo he interrumpido y se ha estabilizado.

Ahora doy Run y el progy corre y aparece el box de habilitación, por lo que deducimos que no tiene ningún sistema antidebug. Por tanto cerramos Olly v2.0 y abrimos Olly v1.1. Lo podíamos haber hecho en la v2.0 pero prefiero la 1.1 pq muchos de nosotros nos sentimos aún más a gusto con ella. Bueno de una manera u otra abrimos el programa y ahí lo tenemos parado en el OEP.



Estudio de una Protección Hardlock por El Cid

Vemos las características de los VB nativos, es decir el JMP al módulo de la máquina virtual de VB: MSVBVM60.

CPU - main thread, module doe35000		
Address	H	Disassembly
00445FB0	PUSH	doe35000.00446BD0
00445FB5	CALL	<JMP.&MSVBVM60.#100>
00445FBA	ADD	BYTE PTR DS:[EAX], AL
00445FBC	PUSH	30000000
00445FC1	ADD	BYTE PTR DS:[EAX], AL
00445FC3	ADD	BYTE PTR DS:[EAX], DL
00445FC6	ADD	BYTE PTR DS:[EAX], AL
00445FC8	INC	EAX
00445FC9	ADD	BYTE PTR DS:[EAX], AL
00445FCB	ADD	BYTE PTR DS:[ECX], CL

Nuestro propósito en este momento es detectar el momento y lugar del proggy, en donde se comprueba el Código introducido. Para ello nos guiaremos del literal que nos ha aparecido en el box: “El código de habilitación no es correcto”.

Bien pues voy a buscarlo por la memoria. Ojo pq tiene acentos y esto es una dificultad añadida ya que puede dar lugar a errores de búsqueda. Por ello no buscaré la cadena completa sino sólo parte de ella.

Hacemos: Search for→All referenced text Springs y el ventana R que aparece de Olly buscamos el texto:

“no es correcto!”

Aquí lo hemos encontrado:

R Text strings referenced in doe35000:.text		
Address	Disassembly	
00505B82	UNICODE	" de prod"
00505B92	UNICODE	"ucto no "
00505BA2	UNICODE	"es valid"
00505BB2	UNICODE	"o",0
00505C20	UNICODE	"n no es "
00505C30	UNICODE	"correcto"
00505C40	UNICODE	"!",0
00505C76	DD	doe35000.00550000
00505C96	DD	doe35000.00520050
00505CAF	DD	doe35000.00520050



Estudio de una Protección Hardlock por El Cid

Si vamos a esta posición de la memoria, vemos el string (en Unicode):

Address	Hex dump	ASCII
00505BE4	8B C7 A5 2A 89 34 B3 88	iÃÑ*ë4 ê
00505BEC	52 00 00 00 45 00 6C 00	R...E.l.
00505BF4	20 00 63 00 F3 00 64 00	.c.¾.d.
00505BFC	69 00 67 00 6F 00 20 00	i.g.o. .
00505C04	64 00 65 00 20 00 68 00	d.e. .h.
00505C0C	61 00 62 00 69 00 6C 00	a.b.i.l.
00505C14	69 00 74 00 61 00 63 00	i.t.a.c.
00505C1C	69 00 F3 00 6E 00 20 00	i.¾.n. .
00505C24	6E 00 6F 00 20 00 65 00	n.o. .e.
00505C2C	73 00 20 00 63 00 6F 00	s. .c.o.
00505C34	72 00 72 00 65 00 63 00	r.r.e.c.
00505C3C	74 00 6F 00 21 00 00 00	t.o.!... .
00505C44	14 00 00 00 20 00 48 00	¶... .H.
00505C4C	61 00 62 00 69 00 43 00	a.b.i.C.

Pongamos un MBP on access en el string y demos RUN:

Olly se detiene aquí:

Address	He	Disassembly
75BB9B30	REP	MOVSD WORD PTR ES:[EDI], DWORD PTR DS:[ESI]
75BB9B32	JMP	DWORD PTR DS:[EDX*4+75BB9988]
75BB9B39	MOV	AL, BYTE PTR DS:[ESI]
75BB9B3B	MOV	BYTE PTR DS:[EDI], AL
75BB9B3D	MOV	AL, BYTE PTR DS:[ESI+1]
75BB9B40	MOV	BYTE PTR DS:[EDI+1], AL
75BB9B43	MOV	EAX, DWORD PTR SS:[EBP+8]
75BB9B46	POP	ESI
75BB9B47	POP	EDI
75BB9B48	LEAVE	
75BB9B49	RET	
75BB9B4A	MOV	EAX, DWORD PTR DS:[ESI+ECX*4-C]

ECX=00000014 (decimal 20.)
DS:[ESI]=[00505BF0]=006C0045
ES:[EDI]=[01B2CEF4]=00470046

Vemos que con la inst REP MOVSD toma los datos del string en 505BF0 y lo lleva a otra pos (EDI=01B2CEF4), con lo que lo va a machacar. ¿Y qué hay allí?

Miremos en la memoria:

Address	Hex dump	ASCII
01B2CEF4	46 00 47 00 48 00 49 00	F.G.H.I.
01B2CEFC	2D 00 41 00 43 00 4B 00	- .A.C.K.
01B2CF04	4E 00 2D 00 4B 00 4A 00	N. - .K.J.
01B2CF0C	49 00 52 00 2D 00 43 00	I.R. - .C.
01B2CF14	4E 00 53 00 48 00 2D 00	N.S.H. - .
01B2CF1C	57 00 58 00 52 00 42 00	W.X.R.B.
01B2CF24	2D 00 45 00 4F 00 52 00	- .E.O.R.
01B2CF2C	54 00 2D 00 00 00 74 00	T. - . . . t.

Caramba esto parece un serial o código. Si miro en el stack lo veo más claro:



Estudio de una Protección Hardlock por El Cid

Address	Value	Comment
0012E930	01B2CEF4	UNICODE "FGHI-ACKN-KJIR-CNSH-WXRB-EORT-"
0012E934	00000052	
0012E938	0012E958	
0012E93C	761F47FA	RETURN to 0LEAUT32-761F47FA from <JMP &msvcrt.memcpy>
0012E940	01B2CEF4	UNICODE "FGHI-ACKN-KJIR-CNSH-WXRB-EORT-"
0012E944	00505BF0	doe35000.00505BF0
0012E948	00000052	
0012E94C	08693DB0	

Estas pos de memoria (01B2CEF4), si miramos en el mapa de memoria, vemos:

01960000	00103000				Map	R
01A70000	000FB000				Priv	RW
01B70000	0037F000				Map	R

, que pertenecen a una sección privada que habrá creado el progy en tiempo de ejecución. El código de habilitación será entonces (si estamos en lo cierto):

"FGHI-ACKN-KJIR-CNSH-WXRB-EORT"

A vosotros os aparecerán otras letras pq el código tiene relación con el usuario y *con el equipo* (yo creo que solo con el equipo). Las que aparezcan son las que tenéis que copiar.

Vamos a probar. Para ello introducimos este código en el box:

Habilitación

Paso 1: Ingrese datos del Usuario

Nombre Completo

pepe perez

Profesion

ingeniero

Empresa

pepe

E-Mail

pepe@pepe.com

Telefono

(11111) 1111111111

Ciudad

cordoba

Provincia

Cordoba

Paso 2: Solicite el envio del codigo de habilitacion

Solicitar Codigo Online

Si no dispone de conexion a internet, vea las instrucciones en el Manual de Habilitación

Paso 3: Habilitar

Codigo

FGHI-ACKN-KJIR-CNSH-WXRB-EORT

Habilitar

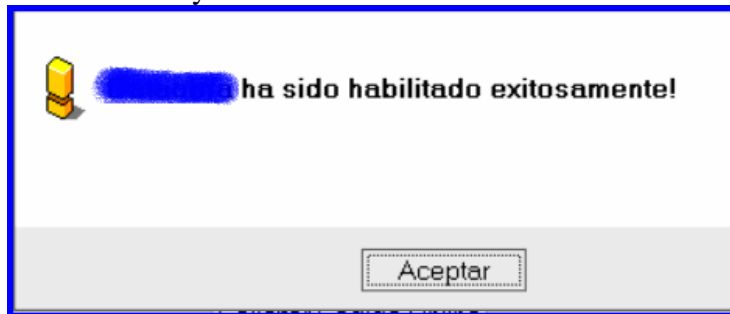
Conozca mas sobre

com

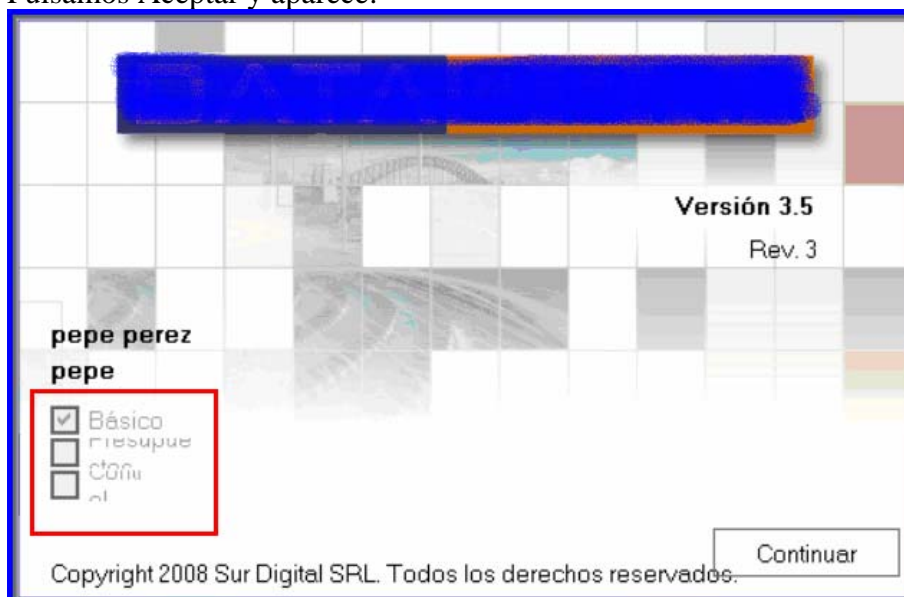


Estudio de una Protección Hardlock por El Cid

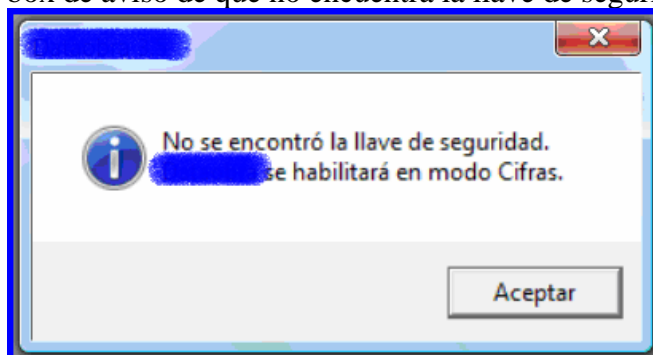
Pulsamos el botón “Habilitar” y:



Pulsamos Aceptar y aparece:



Observamos que solo está activada la casilla Básico. Pulsamos Continuar y nos aparece la box de aviso de que no encuentra la llave de seguridad.



Es decir el proggy está habilitado pero ahora va a buscar la llave y no la encuentra. Luego nuestro trabajo ahora será tratar de emular la llave.

Nota.- Los datos de la “habilitación” sólo es preciso introducirlos (de manera correcta) una vez ya quedan registrados en el fichero



Estudio de una Protección Hardlock por El Cid

dof35000.cfg

En ocasiones, si “jugais” con la habilitación, puede ser necesario reemplazar este fichero por la versión inicial del programa cuando se instala, para volver a una instalación limpia. Con ello podrías habilitaros como otro usuario por ejemplo.

Reiniciamos Olly:

CPU - main thread, module doe35000		
Address	Hex	Disassembly
00445FB0	\$	PUSH 00446BD0
00445FB5	.	CALL <JMP.&MSVBVM60.#100>
00445FBA	.	ADD BYTE PTR DS:[EAX], AL
00445FBC	.	PUSH 30000000
00445FC1	.	ADD BYTE PTR DS:[EAX], AL
00445FC3	.	ADD BYTE PTR DS:[EAX], DL
00445FC6	.	ADD BYTE PTR DS:[EAX], AL
00445FC8	.	INC EAX
00445FC9	.	ADD BYTE PTR DS:[EAX], AL
00445FCB	.	ADD BYTE PTR DS:[ECX], CL
00445FCD	.	XOR EAX, 2B0F46A0
00445FD2	.	LAHF
00445FD3	.	INC ESP
00445FD4	.	PUSHED

Aquí estamos detenidos y empezaremos a estudiar la protección de la llave.

3. PARCHEO DE SERVICIOS

Notas.-

- Para aclaración de algunas dudas o justificaciones de lo que se hace, cuando no esté claro el por qué, se pueden consultar los Borradores del Anexo 1 en Pág. 22, sin que exista garantía de encontrar allí la explicación.
- Para tener una visión general de la estructura de las llamadas a los dongles en general, ver el Anexo 2.



Estudio de una Protección Hardlock por El Cid

Y con ésto comenzamos.

Las llamadas a todos los servicios son éstas:

References in doe35000:text to <Dongle_Nivel_1>		
Address	Disassembly	
0178C5C8	CALL	<Dongle_Nivel_1>
0178C5E2	CALL	<Dongle_Nivel_1>
0178C771	CALL	<Dongle_Nivel_1>
0178C84C	CALL	<Dongle_Nivel_1>
0178C876	CALL	<Dongle_Nivel_1>
0178C89E	CALL	<Dongle_Nivel_1>
0178C997	CALL	<Dongle_Nivel_1>
0178C9CF	CALL	<Dongle_Nivel_1>
0178CA00	CALL	<Dongle_Nivel_1>
0178CA38	CALL	<Dongle_Nivel_1>
0178CA69	CALL	<Dongle_Nivel_1>
0178CAAB	CALL	<Dongle_Nivel_1>
0178CAD3	CALL	<Dongle_Nivel_1>
0178CB54	CALL	<Dongle_Nivel_1>
0178CB8C	CALL	<Dongle_Nivel_1>
0178CBB4	CALL	<Dongle_Nivel_1>

El primer servicio emulado (antes se pasan por otros servicios que no es preciso emular) está aquí:

CPU - main thread, module doe35000					Registers (FPU)	
Address	Hex	Disassembly	Comment			
0178C892	.	LEA ECX, DWORD PTR SS:[EBP-CC]			EAX	0000FFFC
0178C898	.	PUSH ECX			ECX	0012F998
0178C899	.	PUSH EBX			EDX	AE669ACA
0178C89A	.	PUSH EBX			EBX	00000000
0178C89B	.	PUSH EBX			ESP	0012F97C
0178C89C	.	PUSH 8			EBP	0012FA64
0178C89E	.	CALL doe35000.004F4C8C	Serv 1ero		ESI	7276C195 MS
0178C8A3	.	MOV DWORD PTR SS:[EBP-C8], EAX			EDI	72786A8E MS
0178C8A9	.	CALL ESI			EIP	0178C8A3 de
0178C8AB	.	MOV EDX, DWORD PTR SS:[EBP-C8]			C 1	ES 0023 32
0178C8B1	.	PUSH EDX			P 0	CS 001B 32
0178C8B2	.	CALL DWORD PTR DS:[&MSVBVM60.__vbaSt	MSVBVM60.		A 1	SS 0023 32
0178C8B8	.	MOV EDX, EAX			Z 0	DS 0023 32
0178C8BA	.	LEA ECX, DWORD PTR SS:[EBP-34]			S 1	FS 003B 32
0178C8BD	.	MOV EDI, DWORD PTR DS:[&MSVBVM60.__	MSVBVM60.		T 0	GS 0000 NU
0178C8C3	.	CALL EDI	<&MSVBVM60		D 0	
0178C8C5	.	XOR EAX, EAX			O 0	LastErr EP
0178C8C7	?	NOP			EFL	00000293 (I
0178C8C8	.	MOV DWORD PTR DS:[18259DC], EAX			ST0	empty -NAN
0178C8CD	.	MOV WORD PTR SS:[EBP-30], 0ED9A			ST1	empty -???
EAX=0000FFFC Stack SS:[0012F99C]=22C4AE66					ST2	empty -???

Como vemos EAX=FFFC. Lo parcheamos a EAX=0. Inicialmente lo hacemos a mano, después lo hemos hecho con la CALL al Injerto (observar que se han puesto labels en ambos destinos para mejor legibilidad):



Estudio de una Protección Hardlock por El Cid

0178C899	.	PUSH	EBX
0178C89A	.	PUSH	EBX
0178C89B	.	PUSH	EBX
0178C89C	.	PUSH	8
0178C89E	.	CALL	<DONGLE_NIVEL_1>
0178C8A3	.	CALL	<INJERTO>
0178C8A8	.	NOP	
0178C8A9	.	CALL	ESI
0178C8AB	.	MOV	EDX, DWORD PTR SS:[EBP-C8]
0178C8B1	.	PUSH	EDX

Vemos que tenemos tb que hacer el MOV original

```
MOV     DWORD PTR SS:[EBP-C8], EAX
```

que hemos suprimido para poner la CALL al injerto.

Address	Hex	Disassembly
00448180	?	MOV AX, 0
00448184	.	NOP
00448185	.	MOV DWORD PTR SS:[EBP-C8], EAX
0044818B	.	RETN
0044818C	.	NOP

Pasado este Serv llegamos a una zona muy próxima de especial atención:

0178C8C5	.	MOV	EAX, DWORD PTR SS:[EBP-54]	
0178C8C8	.	MOV	DWORD PTR DS:[18259DC], EAX	
0178C8CD	.	CMP	DWORD PTR SS:[EBP-30], 0ED9A	Chico Malo
0178C8D3	.	JE	0178C980	
0178C8D9	.	MOV	DWORD PTR SS:[EBP-AC], 0055ECB0	UNICODE "No se encontr"
0178C8E3	.	MOV	ESI, 8	
0178C8E8	.	MOV	DWORD PTR SS:[EBP-B4], ESI	
0178C8EE	.	PUSH	0D	
0178C8F0	.	LEA	ECX, DWORD PTR SS:[EBP-74]	Arg2 = 0000000D
0178C8F3	.	PUSH	ECX	
0178C8F4	.	CALL	DWORD PTR DS:[<MSVBVM60.#608>]	Arg1 = rtcVarBstrFromAnsi
0178C8FA	.	MOV	DWORD PTR SS:[EBP-BC], 0055ED00	UNICODE "se habi"
0178C904	.	MOV	DWORD PTR SS:[EBP-C4], ESI	
0178C90A	.	LEA	EDX, DWORD PTR SS:[EBP-B4]	
0178C910	.	PUSH	EDX	
0178C911	.	LEA	EAX, DWORD PTR SS:[EBP-74]	
0178C914	.	PUSH	EAX	

Stack SS:[0012FA34]=AE66

Vemos marcada una instrucción que compara los dos valores marcados. Si no son iguales no saltará y entraremos en la zona chico malo pq ya vemos los literales de “No se encontró... bla, bla, bla”

Por tanto una opción es cambiar el JE por un JMP. Ésto me dio problemas pq parece que después utiliza el valor en (EBP-30). Por tanto parché la inst para poner ese valor. Pero entonces no me queda sitio para el CMP así que habría que injertarlo, que es muy fácil y no daría problemas.

Sin embargo me dí cuenta que el flag Z esta a 1 de operaciones anteriores o sea que si no comparamos nada, saltamos seguro. Esta solución no es muy ortodoxa pero es rápida. Pensé cambiarlo después pero no lo hice.



Estudio de una Protección Hardlock por El Cid

Queda así:

0178C8C5	MOV	EAX, DWORD PTR SS:[EBP-54]		
0178C8C8	MOV	DWORD PTR DS:[18259DC1], EAX		
0178C8CD	MOV	WORD PTR SS:[EBP-30], 00000001		
0178C8D3	JE	0178C980		
0178C8D9	MOV	DWORD PTR SS:[EBP-AC], 0055ECB0	UNICODE "No se encontr	
0178C8E0	MOV	ESI, 8		
0178C8E8	MOV	DWORD PTR SS:[EBP-B4], ESI		
0178C8F0	PUSH	ECX	Arg2 = 00000000	
0178C8F3	LEA	ECX, DWORD PTR SS:[EBP-74]		
0178C8F4	PUSH	ECX	Arg1 = rtcVarBstrFromAnsi	
0178C8FA	CALL	DWORD PTR DS:[<&MSVBVM60.#608>]	UNICODE " se h	
0178C8FA	MOV	DWORD PTR SS:[EBP-BC], 0055ED00		
0178C904	MOV	DWORD PTR SS:[EBP-C4], ESI		
0178C90A	LEA	EDX, DWORD PTR SS:[EBP-B4]		
0178C910	PUSH	EDX		

Con el salto evitamos la zona chico malo.

Seguimos con F8, llegando al Serv 2°.

Segundo Servicio

0178C98B	LEA	ECX, DWORD PTR SS:[EBP-CC]		
0178C991	PUSH	ECX		
0178C992	PUSH	EBX		
0178C993	PUSH	EBX		
0178C994	PUSH	EBX		
0178C995	PUSH	8		
0178C997	CALL	doe35000.004F4C8C	Serv 2ndo	
0178C99C	MOV	DWORD PTR SS:[EBP-C8], EAX	Parchea a EAX = 0	
0178C9A2	CALL	ESI		
0178C9A4	MOV	EDX, DWORD PTR SS:[EBP-C8]		

Vemos el mismo esquema de llamada:

```
PUSH ECX
PUSH EBX
PUSH EBX
PUSH EBX
PUSH 8
CALL 004FC8C
```

Es decir que tiene toda la pinta de que es una llamada al driver, cuya dir será: 004FC8C
Pasamos el CALL con F8 y parcheamos AX=0.

Address	Hex	Disassembly	Comment	Registers (FPU)
0178C97B	JMP	doe35000.0178CDD7		EAX 00000000
0178C980	CALL	doe35000.0178C570		ECX 0012F998
0178C985	MOV	DWORD PTR SS:[EBP-CC], EBX		EDX 0012FFFFD
0178C98B	LEA	ECX, DWORD PTR SS:[EBP-CC]		EBX 00000000
0178C991	PUSH	ECX		ESP 0012F97C
0178C992	PUSH	EBX		EBP 0012FA64
0178C993	PUSH	EBX		ESI 7276C195 MS
0178C994	PUSH	EBX		EDI 72786A74 MS
0178C995	PUSH	8		EIP 0178C99C do
0178C997	CALL	doe35000.004F4C8C	Serv 2ndo	C 1 ES 0023 32
0178C99C	MOV	DWORD PTR SS:[EBP-C8], EAX	Parchea a EAX = 0	P 0 CS 001B 32
0178C9A2	CALL	ESI		A 1 SS 0023 32
0178C9A4	MOV	EDX, DWORD PTR SS:[EBP-C8]		N 0 DS 0023 32
0178C9AA	PUSH	EDX		S 1 FS 003B 32
0178C9AB	CALL	DWORD PTR DS:[<&MSVBVM60. vbaStrI2	MSVBVM60. vbaStrI2	

Podemos hacerlo con *el mismo* injerto, ya que vemos que el valor de EAX se mueve a la misma pos del stack que antes (EBP-C8).



Estudio de una Protección Hardlock por El Cid

Con ello queda así:

0178C995	.	PUSH	8		
0178C997	.	CALL	<DONGLE_NIVEL_1>		Serv 2
0178C99C	.	CALL	<INJERTO>		
0178C9A1	.	NOP			
0178C9A2	.	CALL	ESI		
0178C9A4	.	MOV	EDX, DWORD PTR SS:[EBP-C8]		
0178C9AA	.	PUSH	EDX		
0178C9AB	.	CALL	DWORD PTR DS:[&MSVBVM60.__vbaStrI2]		MSVBVM60.
0178C9B1	.	MOV	EDX, EAX		
0178C9B3	.	LEA	ECX, DWORD PTR SS:[EBP-34]		

Seguimos con F8:

Address	Hex	Disassembly	Comment
0178C995	.	PUSH 8	
0178C997	.	CALL doe35000.004F4C8C	Serv 2ndo
0178C99C	.	MOV DWORD PTR SS:[EBP-C8], EAX	Parchear EAX = 0
0178C9A2	.	CALL ESI	
0178C9A4	.	MOV EDX, DWORD PTR SS:[EBP-C8]	
0178C9AA	.	PUSH EDX	
0178C9AB	.	CALL DWORD PTR DS:[&MSVBVM60.__vbaStrI2]	MSVBVM60.__vbaStrI2
0178C9B1	.	MOV EDX, EAX	Ya no hace falta Parchear [EAX]="1"
0178C9B3	.	LEA ECX, DWORD PTR SS:[EBP-34]	
0178C9B6	.	CALL EDI	
0178C9B8	.	MOV DWORD PTR SS:[EBP-CC], EBX	
0178C9BE	.	LEA EAX, DWORD PTR SS:[EBP-CC]	
0178C9C4	.	PUSH EAX	

Seguimos con F8:

Llegamos a:

Servicio 3º

Address	Hex	Disassembly	Comment	Registers
0178C9B3	.	LEA ECX, DWORD PTR SS:[EBP-34]		EAX 22C4AE66
0178C9B6	.	CALL EDI		ECX AE669ACA
0178C9B8	.	MOV DWORD PTR SS:[EBP-CC], EBX		EDX AE669ACA
0178C9BE	.	LEA EAX, DWORD PTR SS:[EBP-CC]		EBX 00000000
0178C9C4	.	PUSH EAX		ESP 0012F97C
0178C9C5	.	PUSH 22C461BA		EBP 0012FA64
0178C9CA	.	PUSH EBX		ESI 7276C195
0178C9CB	.	PUSH EBX		EDI 72786A74
0178C9CD	.	PUSH 3		EIP 0178C9D4
0178C9CF	.	CALL <DONGLE_NIVEL_1>	Serv 3 (N Serv=3)	C 0 ES 0023
0178C9D4	.	MOV DWORD PTR SS:[EBP-C8], EAX		P 1 CS 001B
0178C9DA	.	CALL ESI		A 1 SS 0023
0178C9DC	.	CMP WORD PTR SS:[EBP-C8], 0ED9A		Z 0 DS 0023
0178C9E5	.	JNZ SHORT 0178C9EE		S 0 FS 003B
0178C9E7	.	MOV DWORD PTR SS:[EBP-2C], -1		T 0 GS 0000
0178C9EE	.	MOV DWORD PTR SS:[EBP-CC], EBX		

Aparece de nuevo el CMP de antes con el mismo valor: ED9A h

Después de hacer varias pruebas he llegado a la conclusión de que no se debe saltar ni parchear el valor, sino dejarlo como está, pq si no el programa no corre. Estos valores tienen relación con el tiempo (GetTickCount) o sea que es un método antidebugging Así que lo dejamos (e incluso podemos quitar el BP de 0178C9CF, ya que no hay que parchear nada).



Estudio de una Protección Hardlock por El Cid

Servicio 4º

La emulación es igual que el Serv 2(Nº Serv=8)

0178C9FE	.	PUSH	8		
0178CA00	.	CALL	<DONGLE_NIVEL_1>	Serv 4 (N Serv=8)	
0178CA05	.	CALL	<INJERTO>		
0178CA0A	.	NOP			
0178CA0B	.	CALL	ESI		
0178CA0D	.	MOV	EDX, DWORD PTR SS:[EBP-C8]		
0178CA13	.	PUSH	EDX		
0178CA14	.	CALL	DWORD PTR DS:[<&MSVBVM60.__vbaSt	MSVBVM60.__vbaStrI2	
0178CA1A	.	MOV	EDX, EAX		
0178CA1C	.	LEA	ECX, DWORD PTR SS:[EBP-34]		
0178CA1F	.	CALL	EDI		
0178CA21	.	MOV	DWORD PTR SS:[EBP-CC], EBX		
0178CA27	.	LEA	EAX, DWORD PTR SS:[EBP-CC]		
0178CA2D	.	PUSH	EAX		

Damos Run

Servicio que no es preciso emular:

0178CA2D	.	PUSH	EAX		
0178CA2E	.	PUSH	22C461BA		
0178CA33	.	PUSH	EBX		
0178CA34	.	PUSH	2		
0178CA36	.	PUSH	3		
0178CA38	.	CALL	<DONGLE_NIVEL_1>		
0178CA3D	.	MOV	DWORD PTR SS:[EBP-C8], EAX		
0178CA43	.	CALL	ESI		
0178CA45	.	CMP	WORD PTR SS:[EBP-C8], 0ED9A		
0178CA4E	✓	JNZ	SHORT 0178CA57		
0178CA50	.	MOV	DWORD PTR SS:[EBP-24], -1		
0178CA57	>	MOV	DWORD PTR SS:[EBP-CC], EBX		

Servicio 5 (Nº Serv = 8): Se emula como el 2º, es decir con el injerto, sin más

0178CA63	.	PUSH	ECX, DWORD PTR SS:[EBP-C8]		
0178CA64	.	PUSH	EBX		
0178CA65	.	PUSH	EBX		
0178CA66	.	PUSH	EBX		
0178CA67	.	PUSH	8		
0178CA69	.	CALL	<DONGLE_NIVEL_1>		
0178CA6E	.	CALL	<INJERTO>		
0178CA73	.	NOP			
0178CA74	.	CALL	ESI		
0178CA76	.	MOV	EDX, DWORD PTR SS:[EBP-C8]		
0178CA7C	.	PUSH	EDX		
0178CA7D	.	CALL	DWORD PTR DS:[<&MSVBVM60.__vbaSt		
0178CA83	.	MOV	EDX, EAX		



Estudio de una Protección Hardlock por El Cid

Damos Run y llegamos a :

0178CB1A	:	CALL	DWORD PTR DS:[&MSVBVM60.__vbaEx
0178CB20	:	PUSH	0178CDE1
0178CB25	:	JMP	0178CDD7
0178CB2A	>	CMP	WORD PTR SS:[EBP-2C], BX
0178CB2E	>	JE	0178CBED
0178CB34	:	CMP	WORD PTR DS:[182504C], BX
0178CB3B	:	JE	SHORT 0178CB75
0178CB3D	:	CALL	0178C570
0178CB42	:	MOV	DWORD PTR SS:[EBP-CC], EBX
0178CB48	:	LEA	ECX, DWORD PTR SS:[EBP-CC]

Vemos que vamos a saltar. Si seguimos el salto con Follow, vamos a:

0178CBED	>	MOV	EDX, 004DCBE8	MSVBVM60.__vbaStrCopy
0178CBF2	:	LEA	ECX, DWORD PTR SS:[EBP-28]	UNICODE "CIFR"
0178CBF5	:	CALL	DWORD PTR DS:[&MSVBVM60.__vbaSt	MSVBVM60.__vbaStrCopy
0178CBFB	:	PUSH	0055ED54	UNICODE "Nro. Llave: "
0178CC00	:	MOV	ECX, DWORD PTR DS:[18259DC]	
0178CC06	:	PUSH	ECX	
0178CC07	:	CALL	DWORD PTR DS:[&MSVBVM60.__vbaSt	MSVBVM60.__vbaStrI4
0178CC0D	:	MOV	EDX, EAX	
0178CC0F	:	LEA	ECX, DWORD PTR SS:[EBP-58]	
0178CC12	:	CALL	EDI	
0178CC14	:	PUSH	EAX	
0178CC15	:	MOV	ESI, DWORD PTR DS:[&MSVBVM60.__	Address
0178CC1B	:	CALL	ESI	MSVBVM60.__vbaStrCat
0178CC1D	:	MOV	EDX, EAX	__vbaStrCat
0178CC1F	:	LEA	ECX, DWORD PTR SS:[EBP-5C]	
0178CC22	:	CALL	EDI	
0178CC24	:	PUSH	EAX	
0178CC25	:	PUSH	0055ED94	UNICODE " - Licencias agota
0178CC2A	:	CALL	ESI	
0178CC2C	:	MOV	DWORD PTR SS:[EBP-7C], EAX	
0178CC2F	:	MOV	ESI, 8	
0178CC34	:	MOV	DWORD PTR SS:[EBP-84], ESI	
0178CC3A	:	PUSH	00	Arg2 = 00000000
0178CC3C	:	LEA	EDX, DWORD PTR SS:[EBP-74]	Arg1
0178CC3F	:	PUSH	EDX	rtcVarBstrFromAnsi
0178CC40	:	CALL	DWORD PTR DS:[&MSVBVM60.#608>]	UNICODE " se habili
0178CC46	:	MOV	DWORD PTR SS:[EBP-AC], 0055ED00	
0178CC50	:	MOV	DWORD PTR SS:[EBP-B4], ESI	
0178CC56	:	LEA	EAX, DWORD PTR SS:[EBP-84]	

Donde ya estamos viendo los literales correspondientes a chico malo. Luego no debemos tomar ese salto, por tanto lo ponemos a JNZ:

0178CB1A	:	CALL	DWORD PTR DS:[&MSVBVM60.__vbaEx	MS
0178CB20	:	PUSH	0178CDE1	
0178CB25	:	JMP	0178CDD7	
0178CB2A	>	CMP	WORD PTR SS:[EBP-2C], BX	
0178CB2E	>	JNZ	0178CBED	
0178CB34	:	CMP	WORD PTR DS:[182504C], BX	
0178CB3B	:	JE	SHORT 0178CB75	
0178CB3D	:	CALL	0178C570	
0178CB42	:	MOV	DWORD PTR SS:[EBP-CC], EBX	
0178CB48	:	LEA	ECX, DWORD PTR SS:[EBP-CC]	
0178CB4E	:	PUSH	ECX	
0178CB4F	:	PUSH	EBX	
0178CB50	:	PUSH	EBX	



Estudio de una Protección Hardlock por El Cid

Con ello vemos que no saltamos en éste y sí en el siguiente JE:

0178CB2A	>	CMP	WORD PTR SS:[EBP-2C], BX
0178CB2E	✓	JNZ	0178CBED
0178CB34	.	CMP	WORD PTR DS:[182504C], BX
0178CB3B	✓	JE	SHORT 0178CB75
0178CB3D	.	CALL	0178C570
0178CB42	.	MOV	DWORD PTR SS:[EBP-CC], EBX
0178CB48	.	LEA	ECX, DWORD PTR SS:[EBP-CC]
0178CB4E	.	PUSH	ECX
0178CB4F	.	PUSH	EBX
0178CB50	.	PUSH	EBX
0178CB51	.	PUSH	EBX
0178CB52	.	PUSH	8
0178CB54	.	CALL	<DONGLE_NIVEL_1>
0178CB59	.	MOV	DWORD PTR SS:[EBP-C8], EAX
0178CB5F	.	CALL	ESI
0178CB61	.	MOV	EDX, DWORD PTR SS:[EBP-C8]
0178CB67	.	PUSH	EDX
0178CB68	.	CALL	DWORD PTR DS:[<&MSVBVM60.__vbaSt
0178CB6E	.	MOV	EDX, EAX
0178CB70	.	LEA	ECX, DWORD PTR SS:[EBP-34]
0178CB73	.	CALL	EDI
0178CB75	>	MOV	DWORD PTR SS:[EBP-CC], 0178CE00
0178CB7F	.	LEA	EAX, DWORD PTR SS:[EBP-CC]
0178CB85	.	PUSH	EAX
0178CB86	.	PUSH	EBX
0178CB87	.	PUSH	EBX
0178CB88	.	PUSH	1
0178CB8A	.	PUSH	6
0178CB8C	.	CALL	<DONGLE_NIVEL_1>
0178CB91	.	CALL	<INJERTO>
0178CB96	.	NOP	

Lo que nos lleva a un servicio N°=6 que es necesario emular y lo hacemos con el mismo injerto.

Aquí vemos otra imagen del mismo sitio en otro estudio anterior

Servicios 5°, 6° y Salto Final

0178CB88	.	PUSH	1	
0178CB8A	.	PUSH	6	
0178CB8C	.	CALL	<doe35000.Dongle_Nivel_1>	Serv 5to
0178CB91	.	CALL	<doe35000.Injerto>	Parchear AX = 0
0178CB96	.	NOP		
0178CB97	.	CALL	NEAR ESI	MSVBUM60.__vbaSetSystemError
0178CB99	.	MOV	ECX, DWORD PTR SS:[EBP-C8]	
0178CB9F	.	MOV	DWORD PTR SS:[EBP-30], ECX	
0178CBA2	.	MOV	DWORD PTR SS:[EBP-CC], EBX	
0178CBA8	.	LEA	EDX, DWORD PTR SS:[EBP-CC]	
0178CBAB	.	PUSH	EDX	
0178CBAD	.	PUSH	EBX	
0178CBAF	.	PUSH	EBX	
0178CBB0	.	PUSH	EBX	
0178CBB1	.	PUSH	8	
0178CBB2	.	PUSH	8	
0178CBB4	.	CALL	<doe35000.Dongle_Nivel_1>	Serv 6to
0178CBB9	.	CALL	<doe35000.Injerto>	Parchear EAX = 0
0178CBBE	.	NOP		
0178CBBF	.	CALL	NEAR ESI	MSVBUM60.__vbaSetSystemError
0178CBC1	.	MOV	EAX, DWORD PTR SS:[EBP-C8]	
0178CBC7	.	PUSH	EAX	
0178CBC8	.	CALL	NEAR DWORD PTR DS:[<&MSVBUM60.__vbaS	MSVBUM60.__vbaStrI2
0178CBC9	.	MOV	EDX, EAX	
0178CBCD	.	LEA	ECX, DWORD PTR SS:[EBP-34]	
0178CBCE	.	CALL	NEAR EDI	MSVBUM60.__vbaStrMove
0178CBDB	.	CMP	WORD PTR SS:[EBP-30], BX	
0178CBDB	✓	JGE	doe35000.0178CCD5	aqui hay que saltar
0178CBDF	.	MOV	EDX, doe35000.004DCBE8	UNICODE "CIFR"



Estudio de una Protección Hardlock por El Cid

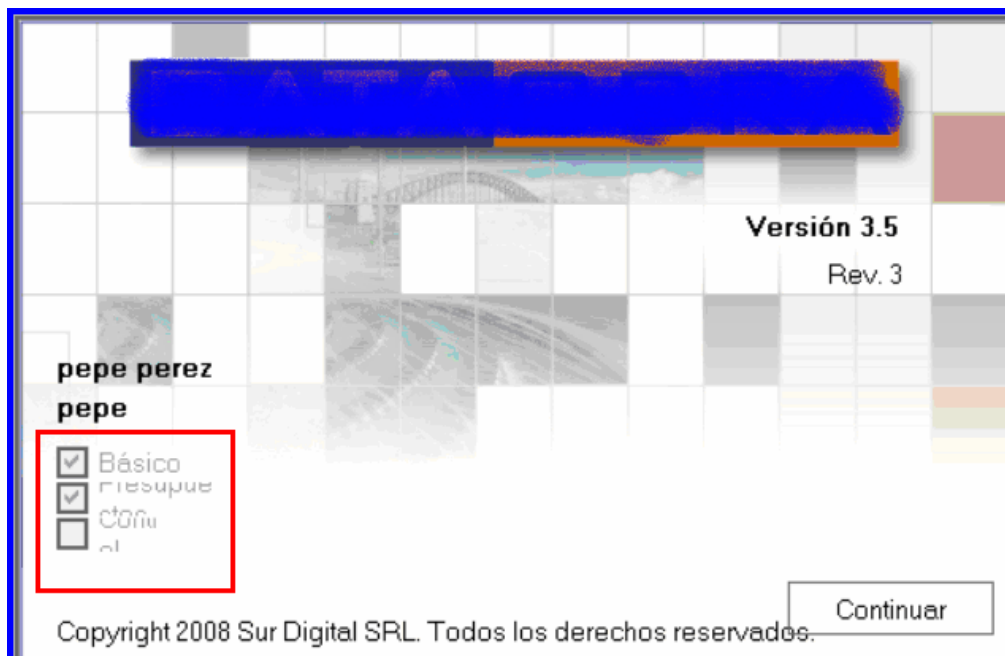
El siguiente servicio y último tb se emula con el injerto.

Con lo que llegamos al salto final:

0178CBD3	CALL	EDI	MSVBVM60.__vbaStrCat
0178CBD5	CMP	WORD PTR SS:[EBP-30], BX	
0178CBD9	JGE	0178CCD5	
0178CBDF	MOV	EDX, 004DCBE8	UNICODE "CIFR"
0178CBE4	LEA	ECX, DWORD PTR SS:[EBP-28]	MSVBVM60.__vbaStrCopy
0178CBE7	CALL	DWORD PTR DS:[&MSVBVM60.__vbaStrCopy]	UNICODE "CIFR"
0178CBED	MOV	EDX, 004DCBE8	MSVBVM60.__vbaStrCopy
0178CBF2	LEA	ECX, DWORD PTR SS:[EBP-28]	UNICODE "Nro. Llave: "
0178CBF5	CALL	DWORD PTR DS:[&MSVBVM60.__vbaStrCopy]	MSVBVM60.__vbaStrI4
0178CBF8	PUSH	0055ED54	
0178CC00	MOV	ECX, DWORD PTR DS:[18259DC]	
0178CC06	PUSH	ECX	
0178CC07	CALL	DWORD PTR DS:[&MSVBVM60.__vbaStrCat]	MSVBVM60.__vbaStrI4
0178CC0D	MOV	EDX, EAX	
0178CC0F	LEA	ECX, DWORD PTR SS:[EBP-58]	
0178CC12	CALL	EDI	Address MSVBVM60.__vbaStrCat
0178CC14	PUSH	EAX	__vbaStrCat
0178CC15	MOV	ESI, DWORD PTR DS:[&MSVBVM60.__vbaStrCat]	
0178CC1B	CALL	ESI	
0178CC1D	MOV	EDX, EAX	
0178CC1F	LEA	ECX, DWORD PTR SS:[EBP-5C]	
0178CC22	CALL	EDI	
0178CC24	PUSH	EAX	
0178CC25	PUSH	0055ED94	UNICODE " - Licencias agotadas"
0178CC2A	CALL	ESI	
0178CC2C	MOV	DWORD PTR SS:[EBP-7C], EAX	
0178CC2F	MOV	ESI, 8	
0178CC34	MOV	DWORD PTR SS:[EBP-84], ESI	
0178CC3A	PUSH	0D	Arg2 = 0000000D
0178CC3C	LEA	EDX, DWORD PTR SS:[EBP-74]	

Vemos que vamos a saltar y q debemos hacerlo pq así evitamos esta zona de chico malo.

Ahora se da Run y el programa arranca con los módulos 1 y 2 habilitados.



Queda por habilitar el 3er módulo, quizá con los servicios que no se han emulado.



Estudio de una Protección Hardlock por El Cid

4. Zona para el injerto e Injerto

Busqué una zona para el injerto bastante grande por si acaso y luego resultó que el injerto es mínimo. El inicio de la zona del injerto es éste:

0044817D	00	DB	00
0044817E	00	DB	00
0044817F	00	DB	00
00448180	00	DB	00
00448181	00	DB	00
00448182	00	DB	00
00448183	00	DB	00
00448184	00	DB	00
00448185	00	DB	00
00448186	00	DB	00
00448187	00	DB	00
00448188	00	DB	00

El injerto completo es éste. Aún se puede suprimir la NOP.

Address	Hex	Disassembly
00448180	?	MOV AX, 0
00448184	.	NOP
00448185	.	MOV DWORD PTR SS:[EBP-C8], EAX
0044818B	.	RETN
0044818C	.	NOP

Ahora solo queda por hacer Copy to Executable→Save File y queda operativo.

Y con ésto damos por terminado este pequeño estudio de esta protección Dongle. Espero os sirva de ejemplo a los que empezáis a tratar con ellas de cómo se pueden tratar y abordar.

5. Epílogo

Hemos querido mostrar brevemente alguna de las operaciones que los dongles realizan para la protección del software que, como otras técnicas tiene ventajas e inconvenientes. No hemos realizado la emulación completa de todos los servicios, sino tan sólo de los necesarios para habilitar 2 de los 3 niveles posibles, dado que conforme se ha expresado repetidamente el objetivo era simplemente estudiar y analizar este tipo de protecciones.



Estudio de una Protección Hardlock por El Cid

ANEXO 1

Borradores. Análisis previo

0055ECAC	4A 00 00 00	4E 00 6F 00	J...N.o.
0055ECB4	20 00 73 00	65 00 20 00	.s.e.
0055ECBC	65 00 6E 00	63 00 6F 00	e.n.c.o.
0055ECC4	6E 00 74 00	72 00 F3 00	n.t.r.¾.
0055ECCC	20 00 6C 00	61 00 20 00	.l.a.
0055ECD4	6C 00 6C 00	61 00 76 00	l.l.a.v.
0055ECDC	65 00 20 00	64 00 65 00	e. .d.e.
0055ECE4	20 00 73 00	65 00 67 00	.s.e.g.
0055ECEC	75 00 72 00	69 00 64 00	u.r.i.d.
0055ECF4	61 00 64 00	2E 00 00 00	a.d....
0055ECFC	4C 00 00 00	44 00 00 00	L...
0055ED04	00 00 00 00	00 00 00 00	
0055ED0C	00 00 00 00	20 00 73 00	...s.
0055ED14	65 00 20 00	68 00 61 00	e. .h.a.
0055ED1C	62 00 69 00	6C 00 69 00	b.i.l.i.
0055ED24	74 00 61 00	72 00 E1 00	t.a.r.ß.
0055ED2C	20 00 65 00	6E 00 20 00	.e.n.
0055ED34	6D 00 6F 00	64 00 6F 00	m.o.d.o.
0055ED3C	20 00 43 00	69 00 66 00	.C.i.f.

0178C8B2	CALL	DWORD PTR DS:[<&MSVBVM60.__vbaStrI2>]	MSVBVM60.__vbaStrI2
0178C8B8	MOV	EDX, EAX	
0178C8BA	LEA	ECX, DWORD PTR SS:[EBP-34]	
0178C8BD	MOV	EDI, DWORD PTR DS:[<&MSVBVM60.__vbaStrMove	MSVBVM60.__vbaStrMove
0178C8C3	CALL	EDI	<&MSVBVM60.__vbaStrMove>
0178C8C5	MOV	EAX, DWORD PTR SS:[EBP-54]	
0178C8C8	MOV	DWORD PTR DS:[18259DC], EAX	
0178C8CD	CMP	WORD PTR SS:[EBP-30], 0ED9A	
0178C8D3	JE	doe35000.0178C980	
0178C8D9	MOV	DWORD PTR SS:[EBP-AC], doe35000.0055ECB0	UNICODE "No se encontr"
0178C8E3	MOV	ESI, 8	
0178C8E8	MOV	DWORD PTR SS:[EBP-B4], ESI	
0178C8EE	PUSH	0D	
0178C8F0	LEA	ECX, DWORD PTR SS:[EBP-74]	
0178C8F3	PUSH	ECX	
0178C8F4	CALL	DWORD PTR DS:[<&MSVBVM60.#608>]	MSVBVM60.rtcVarBstrFromA
0178C8FA	MOV	DWORD PTR SS:[EBP-BC], doe35000.0055ED00	UNICODE " se hab
0178C904	MOV	DWORD PTR SS:[EBP-C4], ESI	
0178C90A	LEA	EDX, DWORD PTR SS:[EBP-B4]	
0178C910	PUSH	EDX	
0178C911	LEA	EAX, DWORD PTR SS:[EBP-74]	
0178C914	PUSH	EAX	
0178C915	LEA	ECX, DWORD PTR SS:[EBP-74]	

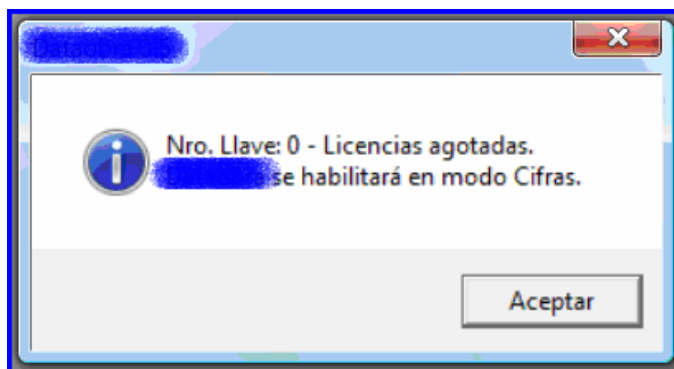
Después de modificar salto y poner a cero [EAX] en 0178C8B8 y parchear este valor:



Estudio de una Protección Hardlock por El Cid

0178C8C8	.	MOV	DWORD PTR DS:[18259DC], EAX
0178C8CD	.	CMP	WORD PTR SS:[EBP-30], 0ED9A
0178C8D3	.	JE	doe35000.0178C980
0178C8D9	.	MOV	DWORD PTR SS:[EBP-AC], doe35000
0178C8E3	.	MOV	ESI, 8
0178C8E5	.	MOV	EDI, 0055ED48
Stack SS:[0012FA34]=ED9A			
Address	Hex dump	ASCII	
0012FA34	9A ED C4 22 00 00 00 00	Ÿ-". . . .	
0012FA3C	6C A5 31 00 00 00 00 00	lÑ1. . . .	
0012FA44	E0 E5 42 00 BC FB 12 00	ôB.Ÿ'†. .	
0012FA4C	26 59 44 00 7C F9 12 00	&YD.†. . .	

Para saltar en el JE (y que no de errores luego, pq no vale solo parchear el JE por un JMP), doy Run y:



Los strings están aquí:

0055ED48	61 00 73 00 2E 00 00 00	a.s.
0055ED4E	00 00 18 00 00 00 4E 00	..†...N.
0055ED56	72 00 6F 00 2E 00 20 00	r.o.
0055ED5E	4C 00 6C 00 61 00 76 00	L.l.a.v. . . .
0055ED66	65 00 3A 00 20 00 00 00	e.:.
0055ED6E	00 00 18 00 00 00 4C 00	..†...L.
0055ED8E	00 00 2C 00 00 00 20 00
0055ED96	2D 00 20 00 4C 00 69 00	-..L.i.
0055ED9E	63 00 65 00 6E 00 63 00	c.e.n.c.
0055EDA6	69 00 61 00 73 00 20 00	i.a.s. . . .
0055EDAE	61 00 67 00 6F 00 74 00	a.g.o.t.
0055EDB6	61 00 64 00 61 00 73 00	a.d.a.s.
0055EDBE	2E 00 00 00 00 00 14 00¶.
0055EDC6	00 00 6D 00 6C 00 50 00	..m.l.P.

Repito lo anterior y pongo un MBP on access en esta parte de la memoria a ver desde donde se llama.



Estudio de una Protección Hardlock por El Cid

Desde aquí:

77DA9E15	8B	MOV	EAX, DWORD PTR DS:[ESI+ECX*4-18]
77DA9E19	89	MOV	DWORD PTR DS:[EDI+ECX*4-18], EAX
77DA9E1D	^ E9	JMP	msvcrt.77DA9B61
77DA9E22	8B	MOV	EAX, DWORD PTR DS:[ESI+ECX*4-1C]
77DA9E26	89	MOV	DWORD PTR DS:[EDI+ECX*4-1C], EAX
77DA9E2A	^ EB	JMP	SHORT msvcrt.77DA9E15
77DA9E2C	FF	JMP	DWORD PTR DS:[ECX*4+77DA9998]
77DA9E33	F3	REP	MOVS DWORD PTR ES:[EDI], DWORD PTR DS:[ESI]
77DA9E35	FF	JMP	DWORD PTR DS:[EDX*4+77DAA018]
77DA9E3C	8B	MOV	BYTE PTR DS:[EDI], AL
77DA9E3E	83	ADD	EDI, 1
77DA9E41	83	SUB	EDX, 1
77DA9E44	^ 75	JNZ	SHORT msvcrt.77DA9E3C
77DA9E46	^ E9	JMP	msvcrt.77DA98C3
77DA9E4B	2B	SUB	EDX, ECX
77DA9E4D	8B	MOV	BYTE PTR DS:[EDI], AL
77DA9E4F	83	ADD	EDI, 1
77DA9E52	83	SUB	ECX, 1
77DA9E55	^ 75	JNZ	SHORT msvcrt.77DA9E4D
77DA9E57	^ E9	JMP	msvcrt.77DA989D
77DA9E5C	8B	MOV	EAX, DWORD PTR SS:[ESP+4]
77DA9E60	C3	RET	

Parcheo a 1:

774AA2BF	56	PUSH	ESI
774AA2C0	E8	CALL	<JMP.&msvcrt.memcpy>
774AA2C5	83	ADD	ESP, 18
774AA2C8	33	XOR	EAX, EAX
774AA2CA	5F	POP	EDI
774AA2CB	5E	POP	ESI
774AA2CC	5B	POP	EBX
774AA2CD	5D	POP	EBP
774AA2CE	C2	RET	0C
774AA2D1	90	NOP	
774AA2D2	90	NOP	
774AA2D3	90	NOP	
774AA2D4	05	ADD	EAX, 204
774AA2D9	00	ADD	BYTE PTR DS:[EAX], AL

ESP=0012F934

Address	Hex dump	ASCII
09C72F9C	31 00 00 00 AB AB AB AB	1...½½½½
09C72FA4	AB AB AB AB 00 00 00 00	½½½½. . .
09C72FAC	00 00 00 00 C0 53 17 48LSH

Pero poco después me vuelve a salir el literal de “licencias agotadas”:



Estudio de una Protección Hardlock por El Cid

0178CC22	.	CALL	EDI
0178CC24	.	PUSH	EAX
0178CC25	.	PUSH	doe35000.0055ED94
0178CC2A	.	CALL	ESI
0178CC2C	.	MOV	DWORD PTR SS:[EBP-7C], EAX
0178CC2F	.	MOV	ESI, 8
0178CC34	.	MOV	DWORD PTR SS:[EBP-84], ESI

0055ED94=doe35000.0055ED94 (UNICODE " - Licencias agotadas.")



Estudio de una Protección Hardlock por El Cid

ANEXO 2: La protección con Dongle: El “Big Picture”

NOTA.- Tomado de un tuto anterior referente a mochilas tipo Hasp, algunos detalles como el número de parámetros de las llamadas al dongle y otros similares, pueden no ser de aplicación aquí.

Con esta expresión me quiero referir al esquema general o macroestructura del sistema Dongle de protección, existente dentro de un programa.

Queda fuera del propósito de estas líneas la protección Dongle mediante el sistema denominado “Envelope”, que guarda más relación con el empaquetado de ejecutables y está tratado en muchos otros foros, siendo en general más sencillo.

La protección Dongle de un programa, consiste en intercalar dentro del código del mismo, una serie de llamadas (CALLs) a determinadas funciones del sistema de protección. Estas funciones se encuentran en unas librerías que vienen con el sistema Dongle al adquirirlo y que se linkan al programa que las usa, de manera estática o dinámica, preferentemente esta última.

Dichas CALLs terminan por dirigirse a la mochila conectada al equipo, quien devuelve unas respuestas que el programa recibe y analiza para ver si se ajustan a lo esperado. Si es así, la respuesta se considera adecuada y el flujo del programa avanza hasta el siguiente punto de control, si existe, o hasta el programa propiamente dicho, si ya no hay ningún otro punto de verificación. Si la respuesta no es la esperada, se considera un error (del tipo de: la mochila no está presente, no es la del programa, está estropeada, etc.) y el programa normalmente se detiene. Este proceso se produce de manera totalmente transparente para el usuario, quien solo tiene que ocuparse de llamar a las rutinas de la librería que desee en cada momento y de analizar las respuestas obtenidas.

En la terminología de los sistemas Dongle, a cada una de las CALLs anteriores se le denomina “Servicio”. Así pues un Servicio Dongle, es una llamada a una rutina de la librería de la misma (una DLL normalmente). Los Servicios se designan normalmente por un número. Así por ejemplo existe el Servicio 1, el 5, el 6, el 32h, el 33h y así sucesivamente.

Para la protección de un programa se utilizan normalmente una serie de servicios consecutivos, siendo el programador quien decide el tipo de Servicio que utiliza en cada caso y el número de veces que emplea cada uno (excepto algunos, como luego veremos). Consecutivos, no debe interpretarse en el sentido de yuxtapuestos, ya que entre las diferentes llamadas a los Servicios, puede existir (y lo lógico es que haya) intercalado, código propio de la aplicación. La ubicación de las llamadas a los servicios dentro del código del programa, es decisión a tomar por el programador y puede decirse que una adecuada ubicación dentro del mismo, redundará en una mayor efectividad del sistema de protección.

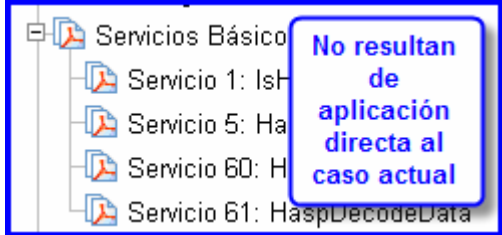
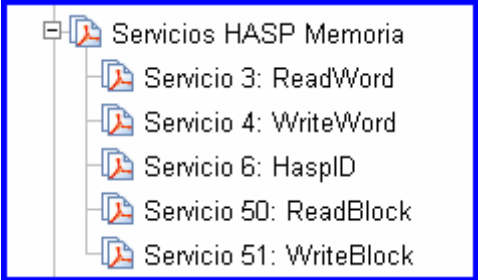
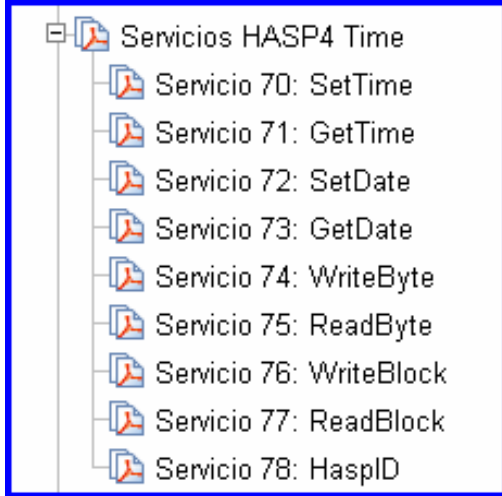

En general, puede decirse que cuantos más Servicios utilice un programa, mayor será la protección alcanzada por éste, si bien el nivel de protección no es lineal ni ilimitado, dado que una vez usados todos los tipos de Servicios posibles, no se podría incrementar la seguridad por la vía de la diversificación de Servicios y en cuanto a la vía de la reiteración de los mismos, tampoco ofrecería una mayor protección ya que si un sistema es capaz de ejercer ingeniería inversa sobre un determinado servicio una vez,



Estudio de una Protección Hardlock por El Cid

de igual forma la podrá realizar tantas veces se utilice el mismo. Pero, hablando en general, no cabe duda de que una mayor utilización de servicios aumenta la dificultad para la desprotección de un programa y contribuye a desanimar a los potenciales Ingenieros Inversos.

Cada tipo de mochila de las vistas antes, tiene su propio conjunto de Servicios específicos, además de existir algunos servicios que son comunes para todas ellas. En la tabla adjunta se pueden ver los servicios que hay para cada tipo de mochila de un tipo concreto de Dongle, tomados del Manual del Usuario de la misma. Los números del servicio están en decimal.

Las llamadas (CALLs) a los servicios Dongle tienen este prototipo o formato:

Sintaxis	hasp (Servicio, CódigoSemilla, LotNum, Password1, Password2, Par1, Par2, Par3)
	Sin aplicación directa al caso actual

Como vemos, una llamada a un servicio tiene varios parámetros, de los que unos son de entrada y el resto de salida, es decir son las respuestas que da la Dongle.

Normalmente, un programa está preparado para que sea capaz de reconocer varios tipos de mochila, según sea la licencia del cliente que lo tenga instalado.

Lo anterior quiere decir que dentro del código de un mismo programa, estarán normalmente todas las CALLs correspondientes a los diversos tipos de mochilas, ya que debe servir para todas las que comercialice la empresa del programa en cuestión. Es decir el código que se genera por el programador, es único y conforme sean las



Estudio de una Protección Hardlock por El Cid

respuestas que obtenga de la mochila instalada, el flujo del programa se dirigirá a unas o a otras partes del mismo. Esto no significa que el programa vaya a pasar por todas las CALL, ya que sólo pasará por aquellas que corresponden al tipo de mochila conectada y a las comunes.

Lo anterior implica que, cuando se estén emulando las respuestas de una mochila, se deberá respetar siempre este hecho, pues de lo contrario se producirá un Error que el sistema atribuye normalmente a Error interno de la pastilla de hardware. Lo veremos después.

Esto de los servicios, parece bastante liso, aunque, como todo en la vida, una vez que se va entendiendo se convierte en una cosa como las demás.

Lo anterior no es óbice para que, sobre todo al principio, pueda causar confusión llegar a comprender la estructura general de las llamadas a los servicios desde un programa y su ubicación e imbricación con el código del mismo. Al menos éso fue lo que, al comienzo, más problemas me causaba a mí, que tengo tendencia a racionalizar todo lo que hago y a no pasar a los detalles de las cosas, hasta no haber comprendido el esquema general del sistema en que me muevo.

Por éso y tratando de que las explicaciones que siguen sean útiles y comprensibles para el mayor número de personas, especialmente para los novatos, he preparado un esquema de lo que podría ser el flujo de llamadas (CALLs) de la protección Dongle de un programa genérico.

Lo que sigue es tan solo un ejemplo de lo que podría ser un esquema de llamadas a los Servicios Dongle. Para un mismo programa, son posibles otros muchos esquemas de llamadas (recordad que lo decide el programador), pero usaremos este para explicar una serie de aspectos de este sistema de protección.

A los diferentes tipos de mochilas las he designado con letras mayúsculas para no vincularme a ninguna concreta de los tipos reales existentes, que en este momento no es lo relevante. A los servicios los he designado con la letra correspondiente a la mochila a que se refiere y un número correlativo, por el mismo motivo. Por ejemplo el Servicio B2 representa el 2º servicio llamado en un programa protegido con una mochila de tipo B, en el esquema concreto adoptado.

Y sin más preámbulo, pasemos a analizar el gráfico.

- 1 Es el punto que representa la entrada en los Servicios Dongle y normalmente representa la primera llamada a los mismos desde el código del programa que se está protegiendo. Normalmente es el Servicio N° 1
- 2 Si la respuesta de la Dongle al servicio anterior, resulta aceptable para el programa, se pasa al siguiente servicio representado en la fig. como **K0**. En éste se trata de ver cual es el tipo de mochila conectada al PC. Normalmente es el Servicio 5

Supongamos que tenemos conectada una mochila de tipo B. Entonces el flujo de nuestro programa se dirigirá por la rama correspondiente a la misma, llamándose a los Servicios B1 y B2. En este último, podría realizarse un bucle como se quiere indicar en la figura con el número **3**, en que este servicio se llamaría varias veces.

En el momento en que cualquier respuesta no sea la adecuada, entraríamos en una rutina de error, lo que normalmente conduciría al programa a detenerse, después de enviar algún mensaje de error.



Estudio de una Protección Hardlock por El Cid

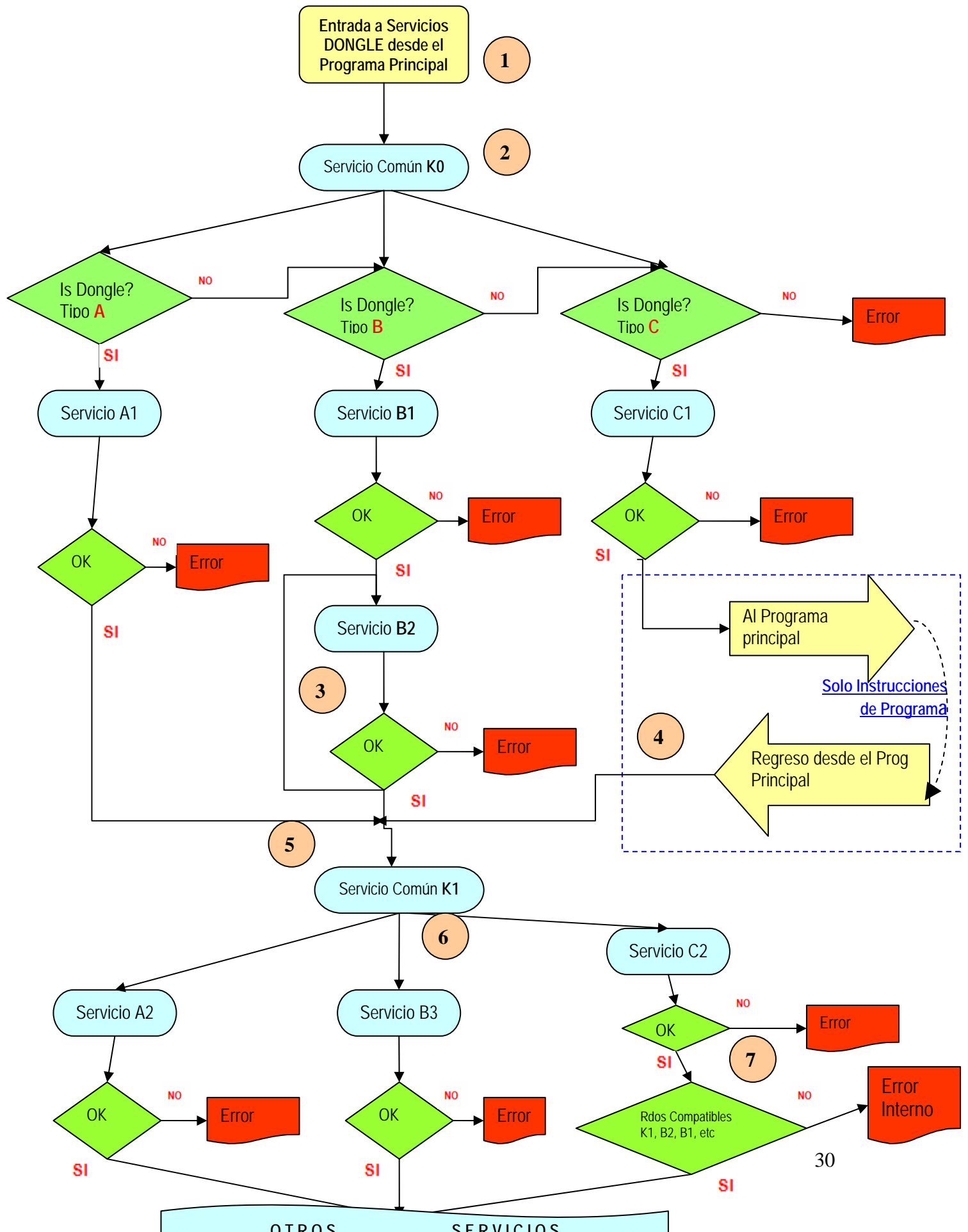
Si, por el contrario, tuviéramos conectada una mochila del tipo C, efectuaríamos un servicio **C1** y si la respuesta es correcta regresaríamos al programa principal, donde se podría ejecutar parte del código del mismo. Es lo que se quiere representar con el cuadrado de puntos número 4. Esto, podría realmente ocurrir entre cada 2 llamadas cualesquiera a los servicios, si bien en la figura nos hemos limitado a ponerlo en esta parte, a fin de no restar claridad la misma.

Una vez hecho lo anterior llegaremos al punto 5 para llamar al Servicio **K1**, que vemos es común con las mochilas de tipo A y B. Quiere esto decir, que si durante el análisis de un programa nos encontrásemos en la entrada a un servicio digamos K1, sólo por este hecho no podríamos saber si procedemos de la rama correspondiente a una mochila A, a una B o a una C, puesto que es común a todos ellos. Ésto que en la entrada del servicio no es demasiado relevante, resulta sin embargo crucial en la salida.

Lo anterior se ve más claro si consideramos la pregunta que surge en este punto que es ¿Qué servicio se ejecutará a continuación del **K1**? La respuesta es: depende del tipo de mochila que el sistema ha identificado en los Servicios anteriores, como la que está conectada al equipo, de manera que se llamará al servicio A2 ó al B3 o al C2, dependiendo de si nuestra mochila ha quedado determinada para el programa, como de tipo A, B o C respectivamente (bien sea la mochila real o la imaginaria, si estamos en emulación).



Estudio de una Protección Hardlock por El Cid





Estudio de una Protección Hardlock por El Cid

Supongamos que nuestra mochila fuera del tipo B y nos encontrásemos a la salida del Servicio K1 marcado con el número 6, después de haber ejecutado los K0, B1 y B2. En la ejecución normal, con la mochila real conectada, el programa se dirigirá de manera automática al Servicio B3. Ahora bien, cuando estemos en emulación, pudiera ser que de manera inadvertida o por parchear inadecuadamente alguna instrucción (normalmente alguna JE, JNZ), el flujo del programa alcanzase el servicio C2. Pues bien, aunque la emulación de este servicio por sí mismo sea la correcta, el programa detectará ésto como una incongruencia y lo interpretará normalmente como un error interno de Dongle, emitiendo el mensaje correspondiente. Es lo que se quiere indicar en la figura con el número 7. (Recordad que hemos dicho que entre cada 2 servicios consecutivos cualesquiera, puede haber -- y de hecho habrá -- grupo/s de instrucciones del programa principal como los representados en 4). Nótese también, que si no disponemos de la mochila, no sabremos cual es el itinerario correcto para el flujo del programa, por lo que deberemos ser muy cuidadosos a la hora de parchear saltos que puedan llevarnos por caminos intrínsecamente incoherentes o incompatibles con las emulaciones realizadas de los servicios anteriores.

Bien, creo que con lo dicho hasta aquí es más que suficiente por ahora y confío en que haya servido para aclarar algo respecto de los servicios. A lo mejor os ha parecido demasiado oscuro o prolijo, pero creo que es interesante y desde luego a mí me hubiera venido muy bien saberlo cuando empecé con estas cosas. Me hubiera ahorrado mucho tiempo y evitado malos ratos de confusión y desorientación.

En todo caso, con la práctica se irán aclarando los puntos que no hayan quedado todavía suficientemente claros. A modo de resumen, destaco los que yo creo que son los más importantes:

- La protección Dongle se establece por medio de CALLs a librerías
- El programa verifica si las respuestas obtenidas son las esperadas
- Cada tipo de mochila tiene su juego de instrucciones CALL. Además hay algunas comunes.
- Una Call a un Servicio Dongle tiene varios parámetros
- Las llamadas a los servicios están intercaladas y diseminadas por entre el código propio de la aplicación.
- El conjunto de llamadas a Servicios de un programa (los conjuntos, sería más propio decir), deben ser compatibles y no pueden mezclarse servicios de distintos tipos de pastillas (salvo los comunes), porque éso se interpreta como un error de hardware y el programa normalmente se detiene.