



2008

Generando Claves de TimePassages



Author: Nacho-Dj

Graphic Editor: Shub-Nigurrath

ARTeam

Julio 2008



DISCLAIMER

All code included with this tutorial is free to use and modify; we only ask that you mention where you found it. This tutorial is also free to distribute in its current unaltered form, with all the included supplements.

All the commercial programs used within this tutorial have been used only for the purpose of demonstrating the theories and methods described. No distribution of patched applications has been done under any media or host. The applications used were most of the times already been patched by other fellows, and cracked versions were available since a lot of time. ARTeam or the authors of the papers shouldn't be considered responsible for damages to the companies holding rights on those programs. The scope of this document as well as any other ARTeam tutorial is of sharing knowledge and teaching how to patch applications, how to bypass protections and generally speaking how to improve the RCE art. We are not releasing any cracked application.

VERIFICATION

ARTeam.esfv can be opened in the ARTeamESFVChecker to verify all files have been released by ARTeam and are unaltered. The ARTeamESFVChecker can be obtained in the release section of the ARTeam site: <http://releases.accessroot.com>





TABLE OF CONTENTS

Disclaimer	2
Verification	2
1 PREFACIO	5
1.1 Introducción	6
1.2 Enlaces	6
2 CONOCIENDO EL OBJETIVO	7
2.1 Empezando.....	7
2.2 Buscando pistas.....	9
2.3 Usando el Programa	10
3 ANÁLISIS	11
3.1 Enfrentándonos a la encriptación	11
3.2 1.4.2 Estableciendo Licencia Comercial	11
3.3 Procesando un texto predefinido	14
3.4 Encriptado del texto	16
3.5 Operaciones con el Nombre de usuario y el Número de serie	17
3.6 Operaciones con el código de activación	18
3.6.1 Tratamiento de números.....	18
3.6.2 Tratamiento de caracteres.....	19
3.6.3 Tratamiento del resto de caracteres	20
3.7 Primer XOR.....	21
3.8 Segundo XOR.....	22
3.9 Último paso por rutina de encriptación	23
3.10 Trazado final.....	25
4 INVERSIÓN DEL PROCESO	27
4.1 Observaciones.....	27



4.2	Aprovechando las propiedades de XOR.....	27
4.3	Ejemplo práctico.....	29
4.4	Invirtiendo "Operaciones con el código de activación"	29
4.4.1	Tercer grupo de caracteres	30
4.4.2	Primer grupo de caracteres	31
4.4.3	Segundo grupo de caracteres	31
4.5	Habilitando todas las modalidades de registro	33
4.5.1	Habilitando Basic Edition y Standard Edition	33
4.5.2	Generando claves para otras versiones del programa	33
4.6	Pseudocode.....	34
5	CONCLUSIÓN	36
6	SALUDOS	37



1 PREFACIO

En este documento se hace un estudio pormenorizado del modo de obtener un generador de claves para el programa comercial TimePassages.

La principal característica que hace atractiva esta tarea es que el programa usa un modo de protección que incluye la encriptación.

También es atractivo que no exista hasta el momento ningún generador de claves para este programa.

Un parcheado en determinadas zonas de código hace que el programa parezca estar registrado, pero un análisis más detallado revela que no es así, debido a la parte de encriptado existente.

Nos enfrentamos, en fin, a una buena protección, y a través de su análisis se podrá obtener un nuevo punto de vista sobre el modo de enfrentarnos a otros programas que usen métodos similares.

Feliz entretenimiento

Nacho_dj



1.1 INTRODUCCIÓN

En este tutorial vamos a analizar una aplicación que usa un poderoso método de protección, y tras ello, sacar las conclusiones adecuadas que nos permitan construir un generador de claves válidas para su registro.

El modo de encontrar alguna clave válida para el registro podría ser más o menos complicado, como en casi todos los programas protegidos, salvo por una cosa: de no estar debidamente registrados, no podremos acceder al contenido de determinados archivos que usa nuestro programa. ¿A qué se debe esto?

Bien, estos archivos han sido encriptados, de tal modo que tratar de ver su contenido es tarea inútil. Sólo veríamos signos extraños, que no son otra cosa que valores aparentemente arbitrarios de los números binarios que componen el archivo. Si usamos una clave válida para este programa sí podremos ver el contenido de los archivos como textos legibles.

Por lo tanto, algo tenemos adelantado: una clave de registro buena activa, de algún modo, el proceso de descryptado de los archivos protegidos.

Empecemos pues a abordar el asunto...

1.2 ENLACES

En este caso, se trata de un ejecutable no comprimido ni protegido, de modo que se puede depurar sin problemas.

El enlace a la página del producto aquí analizado es el siguiente:

<http://www.astrograph.com/downloads/>

Para descargarnos el ejecutable tenemos el siguiente enlace:

<http://www.astrograph.com/downloads/Win50Base.exe>



2 CONOCIENDO EL OBJETIVO

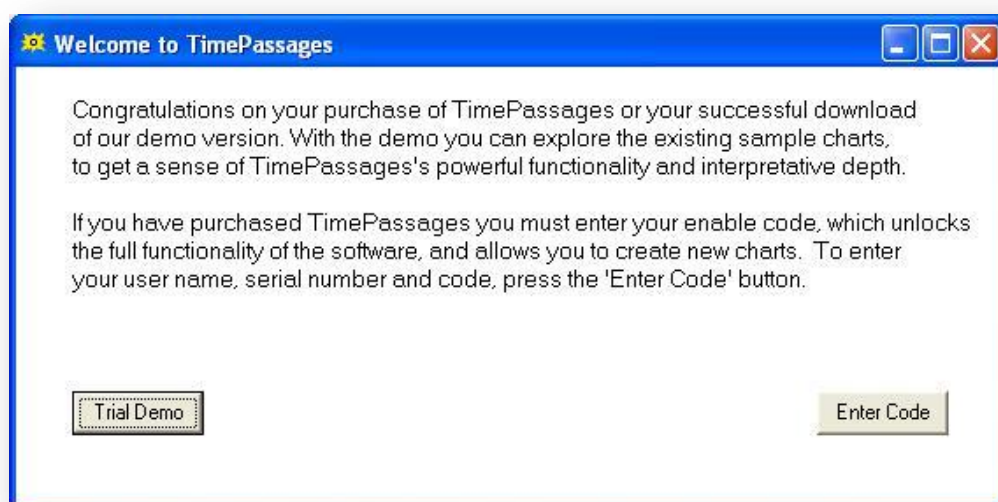
2.1 EMPEZANDO

Posiblemente nos encontremos con que el programa no se ejecute a la primera, ya que necesita tener una impresora configurada. Bien, sigamos los pasos que nos indica el proveedor de sistema operativo para instalarla:

<http://support.microsoft.com/kb/141839/sp-sp/>

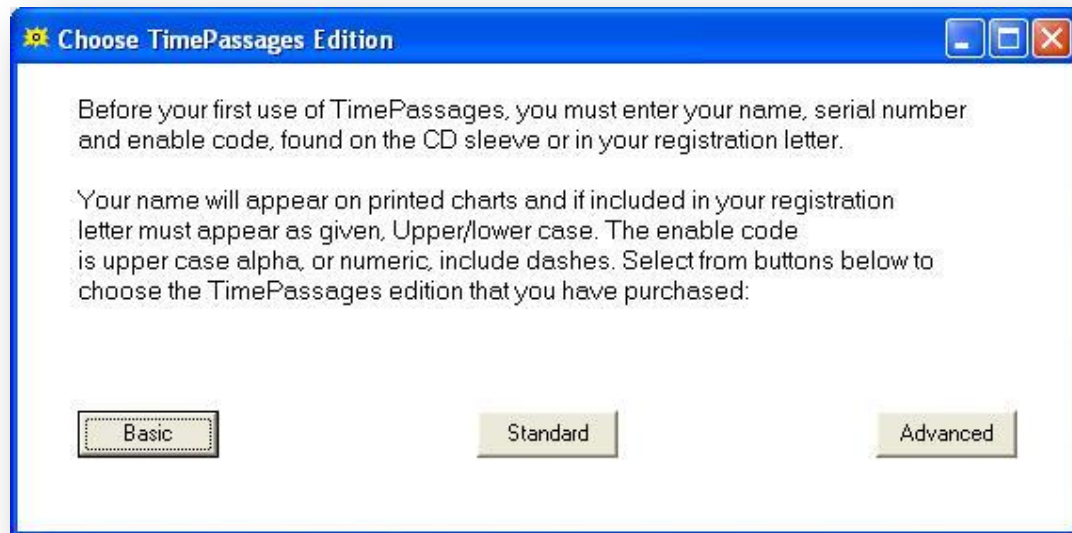
Lo primero que se nos puede ocurrir para desproteger un archivo es introducir una clave cualquiera, a ver qué ocurre.

Para registrarnos, abrimos el programa. Se nos mostrará una nag que nos pregunta si queremos entrar en modo prueba o registrarnos.



Como somos muy valientes, elegimos la versión en que hay que registrarse.

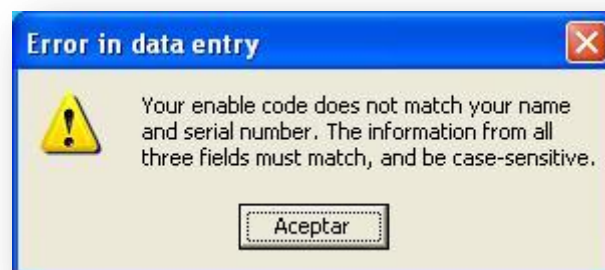
Una vez pulsado el botón, se nos muestra de nuevo otra nag, esta vez para preguntarnos qué modalidad nos interesa para registrarnos.



Ya que tenemos los bolsillos vacíos, nos da igual una que otra, así que elegimos la opción avanzada.

Bien, en este caso concreto se nos piden tres campos: Nombre, Número de Serie y Clave de Activación.

Probemos a introducir los que nos parezca. Pulsamos OK y rápidamente nos aparece el típico mensaje de error (qué lástima que no sea el de que estamos registrados).



Bueno, ya tenemos suficiente información para ponernos en marcha.



2.2 BUSCANDO PISTAS

Acudamos a nuestro buen amigo OllyDbg para analizar el programa desde dentro.

¿Por dónde empezar? Un buen punto de partida es buscar el mensaje de error al intentar registrarnos, el de chico malo.

Así pues, cerramos el programa y lo cargamos en OllyDbg.

Buscamos en Olly las cadenas de caracteres que comiencen por "Your enable code". Usemos *Search for* → *All referenced text strings*.

Ahora usemos la opción *Search for text*. Tecleamos **Your enable code** y marcamos *Entire scope*. Pulsamos OK.

Nos aparece una. Si pulsamos enter sobre ella aparecemos en esta línea:

004AC46C	55	PUSH EBP	
004AC46D	8BEC	MOV EBP,ESP	
004AC46F	83C4 D8	ADD ESP,-28	
004AC472	53	PUSH EBX	
004AC473	8BD8	MOV EBX,EAX	
004AC475	B8 7C856200	MOV EAX,TimePass.0062857C	
004AC47A	E8 DDA81300	CALL TimePass.005E6D5C	
004AC47F	66:C745 E8 0C00	MOV WORD PTR SS:[EBP-18],0C	
004AC485	BA 6D5C6200	MOV EDX,TimePass.00625C6D	ASCII "Your enable code does not mat
004AC48A	8D45 FC	LEA EAX,[LOCAL.1]	
004AC48D	E8 36A61400	CALL TimePass.005F6AC8	
004AC492	FF45 F4	INC [LOCAL.3]	
004AC495	8B10	MOV EDX,DWORD PTR DS:[EAX]	
004AC497	8BC3	MOV EAX,EBX	
004AC499	E8 1A020000	CALL TimePass.004AC6B8	
004AC49E	FF4D F4	DEC [LOCAL.3]	
004AC4A1	8D45 FC	LEA EAX,[LOCAL.1]	
004AC4A4	BA 02000000	MOV EDX,2	
004AC4A9	E8 32A81400	CALL TimePass.005F6CE0	
004AC4AE	8B4D D8	MOV ECX,[LOCAL.10]	
004AC4B1	64:890D 00000000	MOV DWORD PTR FS:[0],ECX	
004AC4B8	5B	POP EBX	00C6F240
004AC4B9	8BE5	MOV ESP,EBP	
004AC4BB	5D	POP EBP	00C6F240
004AC4BC	C3	RETN	

Poniendo un punto de ruptura en esta línea (pulsando F2 sobre la línea seleccionada), efectivamente se comprueba que al intentar registrarnos llegamos aquí. Pulsemos F9 para ejecutar el programa, y repitamos los pasos anteriores para intentar registrarnos.

Tras una sencilla investigación, que no voy a detallar, llegamos a la conclusión que un parche en el salto condicional en 0x520324 para volverlo incondicional haría que siempre que nos registremos se acepte cualquier clave de activación.



Podemos comprobarlo viendo la información que nos da la ventana About: *Barra de Menús -> Help -> About TimePassages*:



Pero ¿está correctamente desprotegido?

2.3 USANDO EL PROGRAMA

Para comprobarlo, lancemos el programa de nuevo desde el principio con ese parche aplicado. Se abre correctamente. Ahora seleccionamos en la barra de menús *File → New...*

De momento hemos podido abrir una nueva ficha, algo que antes no se podía. Bien, rellenemos el nombre de ciudad y elijamos el país. Pulsamos Lookup, y... vemos que salen extraños valores. ¿Qué hemos hecho mal?

Bien, en principio nada, pero algo nos falta por conseguir: que la información que proporcionan los archivos quede descriptada. Porque ese es el problema que tenemos, y que habrá que resolver.



3 ANÁLISIS

3.1 ENFRENTÁNDONOS A LA ENCRIPCIÓN

El simple parcheo del programa para que quedemos registrados no ha sido suficiente. Habrá que estudiar detalladamente qué solución podemos dar. Nada mejor que empezar a investigar de nuevo con nuestro depurador OllyDbg.

Para ello, lanzamos el programa con Olly, eligiendo *Trial Demo*, y una vez abierto, buscamos en la barra de menús: *Help* → *Enable Features* → *Commercial Edition*. Rellenamos los campos con cualquier valor que sea fácilmente reconocible.

Antes de pulsar OK, vamos a buscar un buen punto de ruptura. En los códigos de encriptación se suelen usar las instrucciones **XOR**.

Si en Olly seleccionamos *Search for* → *All commands* e introducimos XOR EAX,EBX obtenemos un listado con varias ocurrencias, todas en una zona bastante concreta. Pulsamos enter sobre la primera ocurrencia (0x49B91B) y vamos a parar a un código bastante típico de encriptado/desencriptado.

Marquemos el principio de la rutina con un punto de ruptura (breakpoint, se añade/elimina pulsando F2 tras haber seleccionado la línea), en la sentencia 0x49B2EC.

3.2 1.4.2 ESTABLECIENDO LICENCIA COMERCIAL

Ahora, tras rellenar con cualquier dato la ventana de registro, pulsamos OK, y esto nos hace caer en el punto de ruptura seleccionado.

Trazando con F8, al llegar a la dirección 0x49B2F8 nos detenemos para seleccionar *Follow in dump -> Memory Address*.

Observamos en la dirección 0x12F970 el valor 0x67452301, en 0x12F974 -> 0xEFCDAB89, en 0x12F978 -> 0x98BADCEF y en 0x12F97C -> 0x10325476 (Estas direcciones y todas las indicadas más adelante podrían variar según la máquina donde se esté ejecutando el programa). Son valores iniciales usados típicamente en la encriptación:

```
0012F970  01234567 89ABCDEF EFCDAB89 67452301
```

Seguimos trazando con F8 y vemos que se van usando todos y cada uno de los anteriores valores.



Si nos fijamos se empiezan a efectuar operaciones entre los valores anteriores y, en la dirección 0x49B330 se asigna el valor numérico contenido en 0x12F8D8, que si observamos en el volcado de memoria no es otra cosa que un texto contenido en el ejecutable:

```

0049B32C . 23C8      AND ECX,EAX
0049B32E . 0BD1      OR EDX,ECX
0049B330 . 8B4D BC   MOV ECX,[LOCAL.17]
0049B333 . 03D1      ADD EDX,ECX
0049B335 . 81C2 78A46AD7 ADD EDX,D76AA478
0049B338 . 03DA      ADD EBX,EDX
0049B33D . 8BD3      MOV EDX,EBX

```

Stack SS:[0012F8D8]=656D6954
ECX=10325476

Address	Hex dump	ASCII
0012F8D8	54 69 6D 65 50 61 73 73 61 67 65 73 28 74 6D 29	TimePassages(tm)
0012F8E8	20 43 6F 6D 6D 65 72 63 69 61 6C 20 4C 69 63 65	Commercial Lice
0012F8F8	6E 73 65 20 56 61 6C 69 64 61 74 69 6F 6E 80 00	nse ValidationC.
0012F908	00 00 00 00 00 00 00 00 70 01 00 00 00 00 00 00	p0.....
0012F918	76 54 32 10 38 F9 12 00 2F B2 49 00 70 F9 12 00	VT28-t./@I.p-t.

Ahora el estudio de las operaciones se puede complicar, pero no nos vamos a entretener en eso. Simplemente, vamos observando cada vez que se asigna a un registro un valor de la zona donde aparece el texto que hemos observado en el gráfico. Pasamos ahora muchas líneas para llegar al final de la rutina, y ponemos un punto de ruptura en 0x49BD0F. En esa dirección se añade el valor de uno de los registros al valor inicial de entrada, del siguiente modo:

[12F970]=67452301, EBX=A57CDC79 → Tras F8, en el registro tenemos 0CC1FF7A.

Las siguientes direcciones de memoria consecutivas también verán modificados su valores al finalizar la rutina:

0012F970 7AFFC10C 43B4CFB9 7AA6D14E 7548FD50

Ahora seguimos trazando con F7, vamos a ver dónde se usan cualquiera de estos valores. Encontramos estas instrucciones (Figura 1):

```

0049BD59 > 8A18 MOV BL, BYTE PTR DS:[EAX]
0049BD5B . 80E3 FF AND BL, 0FF
0049BD5D . 881A MOV BYTE PTR DS:[EDX], BL
0049BD5F . C1EB 08 MOV EBX, DWORD PTR DS:[EAX]
0049BD61 . 80E3 FF SHR EBX, 8
0049BD63 . 885A 01 AND BL, 0FF
0049BD65 . 8B18 MOV BYTE PTR DS:[EDX+1], BL
0049BD67 . C1EB 10 MOV EBX, DWORD PTR DS:[EAX]
0049BD69 . 80E3 FF SHR EBX, 10
0049BD6B . 885A 02 AND BL, 0FF
0049BD6D . 8B18 MOV BYTE PTR DS:[EDX+2], BL
0049BD6F . C1EB 18 MOV EBX, DWORD PTR DS:[EAX]
0049BD71 . 80E3 FF SHR EBX, 18
0049BD73 . 885A 03 AND BL, 0FF
0049BD75 . 83C0 04 MOV BYTE PTR DS:[EDX+3], BL
0049BD77 . 46 ADD EAX, 4
0049BD79 . 83C2 04 INC ESI
0049BD7B . 83C1 04 ADD EDX, 4
0049BD7D . 3BF9 ADD ECX, 4
0049BD7F . 77 CA CMP EDI, ECX
0049BD81 . 5F JA SHORT TimePass.0049BD59
0049BD83 . 5F POP EDI

```

Stack DS:[0012F970]=7A ('z')
BL=70 ('p')
Jump from 0049BD80

Address	Hex dump
0012F970	7A FF C1 0C 43 B4 CF B9 7A A6 D1 4E 75 48 FD 50
0012F980	00 02 00 00 00 00 00 00 F4 69 6D 65 50 61 73 73

Figura 1

Al salir del bucle, se habrán transferido los 4 valores a las direcciones 12F9C8, 12F9CC, 12F9D0 y 12F9D4.

Seguimos trazando con F8, y llegamos a esta zona, donde se vuelven a asignar los valores, pero esta vez sólo se usan los dos primeros (Figura 2):

```

00520620 > 8A18 MOV BL, BYTE PTR DS:[EAX]
00520622 . 881A MOV BYTE PTR DS:[EDX], BL
00520624 . 41 INC ECX
00520626 . 42 INC EDX
00520628 . 40 INC EAX
0052062A . 3BF1 CMP ESI, ECX
0052062C . 7F F5 JG SHORT TimePass.00520620
0052062E . 5E POP ESI

```

Stack DS:[0012F9C8]=7A ('z')
BL=34 ('4')
Jump from 00520629

Address	Hex dump
0012F9C8	7A FF C1 0C 43 B4 CF B9 7A A6 D1 4E 75 48 FD 50
0012F9D0	00 FF 43 00 00 00 00 00 F4 69 6D 65 50 61 73 73

Figura 2



Al llegar a RETN tendremos asignados los dos primeros valores en 12F9FC y 12FA10:

0012F9FC **7AFFC10C 43B4CFB9**

Estas dos palabras se usarán posteriormente, de momento las reservamos.

3.3 PROCESANDO UN TEXTO PREDEFINIDO

Pulsando F9 caemos de nuevo en la rutina de encriptado. ¿Qué se está procesando ahora? Tracemos hasta 0x49B330 y lo veremos:

0049B32C	. 23C8	AND ECX, EAX	
0049B32E	. 0BD1	OR EDX, ECX	
0049B330	. 8B4D BC	MOV ECX, [LOCAL.17]	
0049B333	. 03D1	ADD EDX, ECX	
0049B335	. 81C2 78A46AD7	ADD EDX, 076AA478	
0049B338	. 03DA	ADD EBX, EDX	

Stack SS:[0012F900]=20776F4E	
ECX=10325476	

Address	Hex dump	ASCII
0012F900	4E 6F 77 20 69 73 20 74 68 65 20 74 69 6D 65 20	Now is the time
0012F910	66 6F 72 20 61 6C 6C 20 67 6F 6F 64 20 6D 65 6E	for all good men
0012F920	20 74 6F 20 63 6F 6D 65 20 74 6F 20 74 68 65 20	to come to the
0012F930	61 69 64 20 6F 66 20 74 68 65 69 72 20 63 6F 75	aid of their cou
0012F940	76 54 32 10 60 F9 12 00 2F B2 49 00 80 F9 12 00	ut2'-'-.@I.C-.

Se trata de parte de un texto que viene en el ejecutable, en la dirección 0x641730. Lo podemos ver con un editor hexadecimal abriendo el archivo .exe, en el offset 0x240130.

Bien, al final de la rutina, se obtienen los valores:

0012F980 61867A19 2132052B 5E280D5C D5AA3EED

Hay un segunda entrada en la rutina, ahora viene la continuación del texto anterior, del mismo bloque:

Address	Hex dump	ASCII
0012F900	6E 74 72 79 2E 20 46 6F 75 72 20 71 75 69 63 6B	ntry. Four quick
0012F910	20 73 68 65 65 70 20 6F 75 74 77 69 74 20 6D 61	sheep outwit ma
0012F920	6E 79 20 7A 69 6E 67 79 20 77 69 74 6C 65 73 73	ny zingy witless
0012F930	20 66 6F 78 65 73 2E 20 54 69 6D 65 50 61 73 73	foxes. TimePass
0012F940	10 23 0F C7 60 F9 12 00 40 B2 49 00 80 F9 12 00	ut2'-'-.@I.C-.



Al final del bloque, los valores son:

0012F980 A39C2C28 4EF984E4 C68A5D7C E5CD4CB4

En una nueva pasada por la rutina de encriptación, tenemos lo siguiente como dato de entrada:

Address	Hex dump	ASCII
0012F8E4	61 67 65 73 28 74 6D 29 20 69 73 20 79 6F 75 72	ages(tm) is your
0012F8F4	20 61 73 74 72 6F 6C 6F 67 69 63 61 6C 20 63 68	astrological ch
0012F904	6F 69 63 65 2E 80 00 00 00 00 00 00 00 00 00 00	oice.Ç.....
0012F914	00 00 00 00 00 00 00 00 28 05 00 00 00 00 00 00	(.....
0012F924	FF 6D 4C 04 44 5D 12 00 2F 02 40 00 00 5D 12 00	~!lP▲mI▲

Al final, tenemos estos valores:

0012F980 BC1BFC15 2F8708E9 4D84DBBF D1420FC4

Si trazamos con F7, encontramos que los valores anteriores se trasladan en el código de la [Figura 1](#):

00C8B9A0 BC1BFC15 2F8708E9 4D84DBBF D1420FC4



3.4 ENCRIPTADO DEL TEXTO

Al trazar, encontramos que se combinan ahora las dos dobles palabras obtenidas en el capítulo [Estableciendo Licencia Comercial](#) y el texto del capítulo anterior:

```

005206C9 > 8BC1 MOV EAX,ECX
005206CB . 99 CDQ
005206CC . F77D 14 IDIV [ARG.4]
005206CF . 8B45 10 MOV EAX,[ARG.3]
005206D2 . 8A1410 MOV DL, BYTE PTR DS:[EAX+EDX]
005206D5 . 3016 XOR BYTE PTR DS:[ESI],DL
005206D7 . 41 INC ESI
005206D8 . 46 INC ESI
005206D9 > 3BD9 CMP EBX,ECX
005206DB . 7F EC JG SHORT TimePass.005206C9
005206DD . 8BC7 MOV EAX,EDI

```

Stack DS:[0012F9FC]=7A ('z')
DL=00

Address	Hex dump	ASCII
00C8A7B4	4E 6F 77 20 69 73 20 74 68 65 20 74 69 6D 65 20	Now is the time
00C8A7C4	66 6F 72 20 61 6C 6C 20 67 6F 6F 64 20 6D 65 6E	for all good men
00C8A7D4	20 74 6F 20 63 6F 6D 65 20 74 6F 20 74 68 65 20	to come to the
00C8A7E4	61 69 64 20 6F 66 20 74 68 65 69 72 20 63 6F 75	aid of their cou
00C8A7F4	6E 74 72 79 2E 20 46 6F 75 72 20 71 75 69 63 6B	ntry. Four quick
00C8A804	20 73 68 65 65 70 20 6F 75 74 77 69 74 20 6D 61	sheep outwit ma
00C8A814	6E 79 20 7A 69 6E 67 79 20 77 69 74 6C 65 73 73	ny zingy witless
00C8A824	20 66 6F 78 65 73 2E 20 54 69 6D 65 50 61 73 73	foxes. TimePass
00C8A834	61 67 65 73 28 74 6D 29 20 69 73 20 79 6F 75 72	ages(tn) is your
00C8A844	20 61 73 74 72 6F 6C 6F 67 69 63 61 6C 20 63 68	astrological ch
00C8A854	6F 69 63 65 2E 00 00 00 00 00 00 00 00 00 00	oice.....
00C8A864	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Al finalizar la codificación del texto, quedará así:

```

005206C9 > 8BC1 MOV EAX,ECX
005206CB . 99 CDQ
005206CC . F77D 14 IDIV [ARG.4]
005206CF . 8B45 10 MOV EAX,[ARG.3]
005206D2 . 8A1410 MOV DL, BYTE PTR DS:[EAX+EDX]
005206D5 . 3016 XOR BYTE PTR DS:[ESI],DL
005206D7 . 41 INC ESI
005206D8 . 46 INC ESI
005206D9 > 3BD9 CMP EBX,ECX
005206DB . 7F EC JG SHORT TimePass.005206C9
005206DD . 8BC7 MOV EAX,EDI

```

EDI=00C8A7A0
EAX=0012F9FC

Address	Hex dump	ASCII
00C8A7B4	34 90 B6 2C 2A C7 EF CD 12 9A E1 78 2A D9 AA 99	4E4, *A' = 00B6 * 10
00C8A7C4	1C 90 B3 2C 22 D8 A3 99 10 90 AE 68 63 D9 AA D7	LEI, "i00#e-hc-1-i
00C8A7D4	5A 8B AE 2C 20 D8 A2 DC 5A 8B AE 2C 37 DC AA 99	Zi<, 00Zi<, 700
00C8A7E4	18 96 A5 2C 2C D2 EF CD 12 9A A8 7E 63 D7 A0 CC	+0N,,E'=00c"ci0if
00C8A7F4	14 8B B3 75 6D 94 89 D6 0F 8D E1 7D 36 D0 AC D2	0i um0ei*ip>6!%E
00C8A804	5A 8C A9 69 26 C4 EF D6 0F 8B B6 65 37 94 A2 D8	Zi0i0-'i*iae700i
00C8A814	14 86 E1 76 2A DA A8 C0 5A 8B A8 78 2F D1 BC CA	00B0*rc0Z000/000
00C8A824	5A 99 AE 74 26 C7 E1 99 2E 96 AC 69 13 D5 BC CA	Z0<t&00.0%!!'00
00C8A834	18 98 A4 7F 6B C0 A2 90 5A 96 B2 2C 3A D8 BA CB	+000k'0eZ000,0000
00C8A844	5A 9E B2 78 31 D8 A3 D6 10 96 A2 6D 2F 94 AC D1	Z>00x100i#000/0%0
00C8A854	15 96 A2 69 6D 00 00 00 00 00 00 00 00 00 00	0000000000000000
00C8A864	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0000000000000000



3.5 OPERACIONES CON EL NOMBRE DE USUARIO Y EL NÚMERO DE SERIE

Si hemos ido dejando puntos de ruptura en cada una de las instrucciones indicadas anteriormente, pulsando F9 llegamos de nuevo a la rutina de encriptación 0x49B2EC.

Una vez más, en 0x49B2F8 se asignan los valores iniciales ya conocidos de encriptación:

```
0012F904  01234567 89ABCDEF FEDCBA98 76543210
```

Se va a procesar ahora una cadena que contiene el nombre de usuario y el número de serie, concatenados con otros parámetros introducidos por el programa (hacer volcado usando *Follow in dump* a la memoria al llegar a 0x49B330:

Address	Hex dump	ASCII
0012F86C	4E 61 63 68 6F 5F 64 6A 00 40 35 2E 30 7C 00 31	Nacho_dj.05.01.1
0012F87C	32 33 34 35 36 37 38 80 00 00 00 00 00 00 00	23456780.....
0012F88C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0.....
0012F89C	00 00 00 00 00 00 00 00 B8 00 00 00 00 00 00	0.....
0012F8AC	76 54 32 10 CC F8 12 00 2F B2 49 00 04 F9 12 00	VT2 F°\$./@1.♦\$.

Al final de la rutina, y teniendo en cuenta lo anteriormente visto, tenemos estos nuevos valores:

```
0012F904  8C9C7FFB 8CEE612D 639379A7 AE40993C
```

Ahora seguimos trazando con F7, y llegamos a las instrucciones de la [Figura 1](#) de nuevo.

Al salir del bucle, se habrán transferido los 4 anteriores valores a las direcciones 12F95C, 12F960, 12F964 y 12F968.

Seguimos trazando con F8, y llegamos a las instrucciones de la [Figura 2](#), obteniendo esto:

```
0012F9C4  8C9C7FFB 8CEE612D
```

Si nos fijamos, se ha obtenido algo equivalente a lo que se tenía al final del capítulo [Estableciendo Licencia Comercial](#).

Como antes, reservamos estas dos dobles palabras, porque en un instante serán usadas.



3.6 OPERACIONES CON EL CÓDIGO DE ACTIVACIÓN

Continuamos trazando con F8.

En la dirección 0x5203B0 hay una interesante llamada. De hecho, es fundamental para nuestros objetivos.

La analizaremos ahora, pero ya adelanto que es donde se procesa el código de activación.

Entramos con F7 en la rutina. Sólo se mencionará aquí lo que es realmente útil para nuestros objetivos.

En 0x5203F4 podemos observar nuestro código de activación. Vamos por buen camino :)

```

005203F0 | . 85D2      TEST EDX,EDX
005203F2 | . 74 04      JE SHORT TimePass.005203F8
005203F4 | . 8B0E      MOV ECX,DWORD PTR DS:[ESI]
005203F6 | . EB 05      JMP SHORT TimePass.005203FD
005203F8 | . B8 30F340B8 MOV ECX,TimePass.005203F8
Stack DS:[0012FAF4]=00CDE578, (ASCII "Idonotknowwhichoneshoulditbe")
ECX=0012FAF4
    
```

3.6.1 TRATAMIENTO DE NÚMEROS

Seguimos trazando. Llegamos a un bucle donde podremos comprobar que se va operando con los caracteres de nuestra clave.

```

00520485 | . EB 13      JMP SHORT TimePass.0052049A
00520487 | . 8BC8      MOV ECX,EAX
00520489 | . 03C9      ADD ECX,ECX
0052048B | . 8D0C89    LEA ECX,DWORD PTR DS:[ECX+ECX*4]
0052048E | . 89C8      MOV EAX,ECX
00520490 | . 0FBF0A    MOVSX ECX,BYTE PTR DS:[EDX]
00520493 | . 83C1 D0    ADD ECX,-30
00520496 | . 03C1      ADD EAX,ECX
00520498 | . 43        INC EBX
00520499 | . 42        INC EDX
0052049A | . 0FBF0A    MOVSX ECX,BYTE PTR DS:[EDX]
0052049D | . 83F9 2D    CMP ECX,2D
005204A0 | . 74 13      JE SHORT TimePass.005204B5
005204A2 | . 8B0E      MOV ECX,DWORD PTR DS:[ESI]
005204A4 | . 85C9      TEST ECX,ECX
005204A6 | . 74 07      JE SHORT TimePass.005204AF
005204A8 | . 8B0E      MOV ECX,DWORD PTR DS:[ESI]
005204AA | . 8B49 FC    MOV ECX,DWORD PTR DS:[ECX-4]
005204AD | . EB 02      JMP SHORT TimePass.005204B1
005204AF | . 33C9      XOR ECX,ECX
005204B1 | . 3BD9      CMP EBX,ECX
005204B3 | . 7C D2      JL SHORT TimePass.005204B7
005204B5 | . 43        INC EBX
    
```

Si vamos trazando cuidadosamente, observaremos que lo primero que se hace es buscar un '-' (0x2D). Como nuestra clave del ejemplo no lo tiene se sale del bucle.

Solución: vamos a usar una clave que tenga un guión.

Pongamos un punto de ruptura en 0x52049A; después tecleamos ALT+B para ver la lista de los breakpoints, hacemos disable (deshabilitar) en todos menos el último que hemos puesto, y pulsamos F9. Así sale el [mensaje de error](#) al registrarnos y podemos volver a introducir los datos de nuestro registro.



Escribamos un código de activación (Enabling Code) con un guión en medio. Ahora pulsamos OK y caeremos en 0x52049A.

Bueno, ahora en 0x52049A nos aparece la cadena que hemos escrito tras el guión. Esto va bien.

Tracemos para ver qué calcula. A la primera pasada no parece que se vea nada útil, pero a partir de que se tome el primer carácter en 0x520490 empezamos a averiguar qué ocurre. Lo que se hace en 0x520493 es restarle 0x30. ¿Qué nos sugiere esto?

Si hemos visto mucho código de ejecutables antes que este podremos suponer qué es lo que se está haciendo.

Parece que se espera un número, de modo que al restarle 0x30 (código ASCII del cero en hexadecimal) tenemos el valor numérico del carácter que representa.

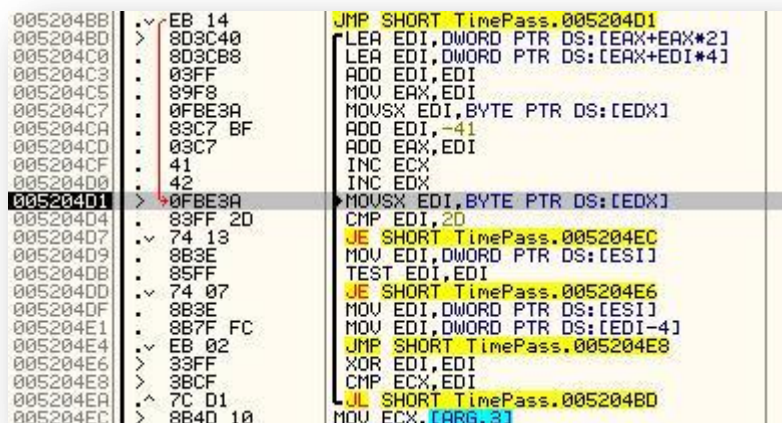
Bien, pulsemos F9 hasta obtener el mensaje de error, y volvamos a introducir un código de activación, pero con números tras el guión. Pulsemos OK.

Al llegar al mismo código, nos encontramos con que la línea 0x520489 duplica lo que hay en ECX, y la siguiente línea lo multiplica por 5, de modo que se ha multiplicado por 10 la cantidad que había acumulada y se le ha añadido el valor numérico de la cifra extraída. Bien, ya tenemos el algoritmo que contabiliza las cifras numéricas.

Fijémonos que en EAX queda la suma de todo lo calculado. Al seguir trazando con F8 terminamos el bucle.

3.6.2 TRATAMIENTO DE CARACTERES

El siguiente bucle que encontramos se parece al anterior bastante:



Sin embargo, vemos que en lugar de restarle 0x30 al valor binario de un carácter, le resta 0x41.

¿Qué es 0x41? Ni más ni menos que el valor numérico del carácter 'A'. De modo que se va a ir tomando un carácter, el primero del código de activación.



Después si seguimos la línea 5204BD se observa que se multiplica lo acumulado por 3, y en la siguiente línea se multiplica por 4 lo anterior y se le suma otra vez lo acumulado. En la posterior línea se duplica el valor de EDI.

Conclusión: se ha multiplicado lo acumulado por 26. Qué casualidad, el número de letras del código ASCII, en este caso mayúsculas...

Seguir trazando nos genera un nuevo valor del acumulado en EAX. Seguimos trazando fuera del bucle y observamos que se coloca el valor de EAX en la dirección 0x12F990 (sentencias en 0x5204F4, 0x520502, 0x520510 y 0x52051B, pero con los bytes bajos y altos intercambiados).

3.6.3 TRATAMIENTO DEL RESTO DE CARACTERES

Usando F8 seguimos y llegamos a otro bucle:

00520525	> EB 14	JMP SHORT TimePass.0052053B
00520527	> 80C40	LEA ECX,DWORD PTR DS:[EAX+EAX*2]
0052052A	. 80C88	LEA ECX,DWORD PTR DS:[EAX+ECX*4]
0052052D	. 03C9	ADD ECX,ECX
0052052F	. 89C8	MOV EAX,ECX
00520531	. 0FB80A	MOVSX ECX,BYTE PTR DS:[EDX]
00520534	. 83C1 BF	ADD ECX,-41
00520537	. 03C1	ADD EAX,ECX
00520539	. 43	INC EBX
0052053A	. 42	INC EDX
0052053B	> 8B0E	MOV ECX,DWORD PTR DS:[ESI]
0052053D	. 85C9	TEST ECX,ECX
0052053F	> 74 07	JE SHORT TimePass.00520548
00520541	. 8B0E	MOV ECX,DWORD PTR DS:[ESI]
00520543	. 8B49 FC	MOV ECX,DWORD PTR DS:[ECX-4]
00520546	> EB 02	JMP SHORT TimePass.0052054A
00520548	> 33C9	XOR ECX,ECX
0052054A	> 3BD9	CMP EBX,ECX
0052054C	> 7C D9	JL SHORT TimePass.00520527
0052054E	. 8B4D 10	MOV ECX,CARG.31

Este es similar al anterior, pero con un código introducido de tipo XXXXXX-000000 (las X son caracteres y los 0 son números) no conseguimos que sea procesado en el bucle. ¿Cómo hacer?

Simplemente, colocamos un '-' tras lo anterior y más caracteres en mayúsculas. Volvemos a pulsar F9 e introducimos el nuevo código, hasta llegar de nuevo a este bucle.

Así pues, con un código de activación del tipo XXXXXX-000000-XXXXXX conseguimos que entre en los tres bucles encontrados.

En este bucle se calcula algo similar al código anterior, sin embargo se comienza sin acumulado anterior, y el resultado va a parar a la doble palabra contigua a la del anterior tratamiento, en este caso 0x12F994, también con los bytes altos y bajos intercambiados.

Usando el código del ejemplo, XXXXXX-000000-XXXXXX, se obtienen estas dos dobles palabras (dependen del código de activación usado):

0012F990 **10F09611 10F09611**



3.7 PRIMER XOR

Seguimos trazando con F8, para salir de la rutina. De inmediato encontramos otro sector de código, este muy interesante:

```

005203C3  > 8A18  MOV BL, BYTE PTR DS:[EAX]
005203C5  . 301A  XOR BYTE PTR DS:[EDX], BL
005203C7  . 41     INC ECX
005203C8  . 42     INC EDX
005203C9  . 40     INC EAX
005203CA  . 3BF1   CMP ESI, ECX
005203CC  . 7F F5  JG SHORT TimePass.005203C3
005203CE  > 5F     POP EDI

```

BL=38 ('8')
Stack DS:[0012F9C4]=8C ('I')

Address	Hex dump
0012F9C4	8C 9C 7F FB 8C EE 61 2D 4E 11 54 00 F5 29 54 00
0012F9C5	9C FB 12 88 88 9F 4D 88 88 88 12 88 88 87 88 88

Aquí se van a combinar con un **XOR** las dos palabras dobles que teníamos reservadas (ver [Operaciones con el Nombre de usuario y el número de serie](#)) con las que acabamos de obtener del código de activación. Interesante ¿no?

El resultado queda en 0x12F9C4 y 0x12F9C8.

0012F9C4 **9C6CE9EA 9C1EF73C**



3.8 SEGUNDO XOR

Seguimos trazando con F8 hasta 0x520376. Usando F7, trazamos dentro de la llamada hasta 0x5202C4, y volvemos a entrar con F7. En el interior de la rutina, en 0x5E67FB se está copiando el bloque que se encriptó al final del capítulo [Encriptado del texto](#) a otra dirección, en este caso 0xDC0070.

Tras seguir trazando unas pocas líneas de nuevo con F8, seguimos hasta llegar a esta sección de código (Figura 3):

```

00520203 .> EB 00      JMP SHORT TimePass.005202E2
00520205 .> 8BC1      MOV EAX,ECX
00520207 .> 41        INC ECX
00520208 .> 99        CDQ
00520209 .> F77D 14   IDIV DWORD PTR SS:[EBP+14]
0052020C .> 8A0417    MOV AL, BYTE PTR DS:[EDI+EDX]
0052020F .> 3003      XOR BYTE PTR DS:[EBX],AL
005202E1 .> 43        INC EBX
005202E2 .> 8B56 10   MOV EDX, DWORD PTR DS:[ESI+10]
005202E5 .> 3BD1      CMP EDX,ECX
005202E7 .> 77 EC     JA SHORT TimePass.005202D5

```

Stack DS:[0012F9C4]=9C ('&')
AL=00

Address	Hex dump	ASCII
00DC0070	34 90 B6 2C 2A C7 EF CD 12 9A E1 78 2A D9 AA 99	4eA, #A' = 00x#-0
00DC0080	1C 90 B3 2C 22 08 A3 99 10 90 AE 68 63 D9 AA D7	LEI, "i0#e-ho-i
00DC0090	5A 8B AE 2C 20 0B A2 DC 5A 8B AE 2C 37 DC AA 99	Zi<, 0Zi<, 7-0
00DC00A0	1B 96 A5 2C 2C D2 EF CD 12 9A A8 7E 63 D7 A0 CC	+uA, ,E' = 00c'ciA f
00DC00B0	14 8B B3 75 6D 94 89 D6 0F 8D E1 7D 36 DD AC D2	ii um0ei*ib)6!%E
00DC00C0	5A 8C A9 69 26 C4 EF D6 0F 8B B6 65 37 94 A2 D8	Zi0i8- 'i*iae700i
00DC00D0	14 86 E1 76 2A DA A8 C0 5A 88 A8 78 2F D1 BC CA	iiBv*rc'Zecx/p#
00DC00E0	5A 99 AE 74 26 C7 E1 99 2E 96 AC 69 13 D5 BC CA	Z0<t&A00.0%i!!'s
00DC00F0	1B 98 A4 7F 68 C0 A2 90 5A 96 B2 2C 3A DB BA CB	+uAk'oeZ00, : f
00DC0100	5A 9E B2 78 31 DB A3 D6 1D 96 A2 6D 2F 94 AC D1	Z>00x1ii#00m/0%0
00DC0110	15 96 A2 69 6D 00 00 00 00 00 00 00 00 00 00	S00im.....
00DC0120	00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Figura 3

Ahora examinemos cuidadosamente según trazamos.

Se van a combinar mediante un **XOR** las dos palabras resultantes del capítulo anterior con un bloque de bytes que en la imagen se ve completo. Al terminar, todos los bytes del bloque quedan modificados. ¿Nos sugiere algo esto? Fijémonos en el tamaño del bloque: 0xA5

Lo que ocurre es que este bloque está sufriendo una **encriptación**. ¿O más bien un desencriptado?

Además, este bloque proviene de un texto que fue encriptado previamente, que se indica al inicio del capítulo [Encriptado del texto](#).



Tras el bucle, el bloque queda como sigue (Figura 4):

```

005202D5 > EB 0D      MOV EAX, ECX
005202D7 . 41          INC ECX
005202D8 . 99          CDQ
005202D9 . F77D 14     IDIV DWORD PTR SS:[EBP+14]
005202DC . 8A0417      MOV AL, BYTE PTR DS:[EDI+EDX]
005202DF . 3003        XOR BYTE PTR DS:[EBX], AL
005202E1 . 43          INC EBX
005202E2 > 8B56 10     MOV EDX, DWORD PTR DS:[ESI+10]
005202E5 . 3BD1        CMP EDX, ECX
005202E7 . 77 EC       JA SHORT TimePass.005202D5
005202E9 . 8D4D A0     LEA ECX, DWORD PTR SS:[EBP-60]

```

Stack address=0012F944
ECX=000000A5

Address	Hex dump	ASCII
00DC0070	A8 FC 5F C6 B6 D9 18 F1 8E F6 08 92 B6 C7 5D A5	z SA t A EA J N
00DC0080	80 FC 5A C6 BE C6 54 A5 81 FC 47 82 FF C7 5D EB	C Z g s T N u G e A J U
00DC0090	C6 E7 47 C6 BC C5 55 E0 C6 E7 47 C6 AB C2 5D A5	g G s t U O g G s t J N
00DC00A0	87 FA 4C C6 B0 CC 18 F1 8E F6 41 94 FF C9 57 F0	C L s t t A A O F W -
00DC00B0	88 E7 5A 9F F1 8A 7E EA 93 E1 08 97 AA C3 5B EE	e Z f t e U O B u t C -
00DC00C0	C6 E0 40 83 BA DA 18 EA 93 E7 5F 8F AB 8A 55 E4	s O e a l l r t O O e A s e U O
00DC00D0	88 EA 08 9C B6 C4 5F FC C6 E4 41 92 B3 CF 4B F6	e O e A - s s A E I B K +
00DC00E0	C6 F5 47 9E BA D9 16 A5 B2 FA 45 83 8F CB 4B F6	s s G x l l - n s s E a A r K +
00DC00F0	87 F4 4D 95 F7 DE 55 AC C6 FA 58 C6 A6 C5 4D F7	C q M o - i U s C s a t M -
00DC0100	C6 F2 58 92 AD C5 54 EA 81 FA 4B 87 B3 8A 5B ED	s - C E t T O u K q l e L Y
00DC0110	89 FA 4B 83 F1 00 00 00 00 00 00 00 00 00 00 00	e K a t
00DC0120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Figura 4

Está claro que no ha sido descryptado adecuadamente, por no haber usado un código de activación "bueno".

3.9 ÚLTIMO PASO POR Rutina de ENcriptación

Sigamos con F9, tras haber habilitado de nuevo todos los puntos de ruptura.

Caemos de nuevo en la rutina de encriptación (ver [Enfrentándonos a la encriptación](#)), donde se entra como siempre inicializando los valores para el proceso:

0012F944 01234567 89ABCDEF FEDCBA98 76543210



Y un poco más abajo, vemos qué se está procesando:

Address	Hex dump	ASCII
0012F8B4	A8 FC 5F C6 B6 D9 18 F1 8E F6 08 92 B6 C7 5D A5	¿? \$A-↑:A+■EAAJN
0012F8C4	80 FC 5A C6 BE C6 54 A5 81 FC 47 82 FF C7 5D EB	C*Z\$%\$T\$u?G\$ \$10
0012F8D4	C6 E7 47 C6 BC C5 55 E0 C6 E7 47 C6 A8 C2 5D A5	\$fG\$+U0\$eG\$+JN
0012F8E4	87 FA 4C C6 B0 CC 18 F1 8E F6 41 94 FF C9 57 F0	C·L\$ ↑↑:A+A0 fW-
0012F8F4	76 54 32 10 14 F9 12 00 2F B2 49 00 44 F9 12 00	UT2▶ ·\$./\$I.D·\$.

Esto no es más ni menos que una parte del bloque de bytes obtenido al final del capítulo [Segundo XOR](#).

Al final de la rutina se sale con los valores:

0012F944 49CAB6D4 4BDE6803 8A1DC07E D9536000

Hay una segunda entrada a la rutina, con los valores anteriores. El bloque de bytes es la continuación del bloque anterior:

Address	Hex dump	ASCII
0012F8B4	88 E7 5A 9F F1 8A 7E EA 93 E1 08 97 AA C3 5B EE	\$fZf+e~00\$u-+[-
0012F8C4	C6 E0 40 83 BA DA 18 EA 93 E7 5F 8F AB 8A 55 E4	\$00\$ r↑00\$ _A%eU\$
0012F8D4	88 EA 08 9C B6 C4 5F FC C6 E4 41 92 B3 CF 4B F6	\$0\$EA-? \$SRAE BK+
0012F8E4	C6 F5 47 9E BA D9 16 A5 B2 FA 45 83 8F CB 4B F6	\$S\$× -·N\$E\$A\$K+
0012F8F4	08 F2 60 00 14 F9 12 00 40 B2 49 00 44 F9 12 00	LS·\$+·\$I.D·\$.

Al final de la rutina se sale con los valores:

0012F944 E0405740 8035A60D 4636F1FE F34A0F74

Se produce una tercera entrada a la rutina, con los anteriores valores. El bloque es la tercera parte del bloque obtenido en el capítulo anterior:

Address	Hex dump	ASCII
0012F898	87 F4 4D 95 F7 DE 55 AC C6 FA 5B C6 A6 C5 4D F7	çM0·iU\$·[\$+M·
0012F8A8	C6 F2 5B 92 AD C5 54 EA 81 FA 4B 87 B3 8A 5B ED	\$=[Ei+T0ü·KçleCÿ
0012F8B8	89 FA 4B 83 F1 80 00 00 00 00 00 00 00 00 00	e·K\$+C.....
0012F8C8	00 00 00 00 00 00 00 00 28 05 00 00 00 00 00	(.....
0012F8D8	F3 4A 0F 74 F8 F8 12 00 2F B2 49 00 44 F9 12 00	%.\$+°\$./\$I.D·\$.



Al final de la rutina se sale con los valores:

0012F944 D13C814E F9A923BC D716574E B0ABA423

Nótese de nuevo al final del bloque de bytes la existencia del byte 0x80 y una cantidad en la decimoquinta doble palabra.

3.10 TRAZADO FINAL

Si usamos F8, llegaremos a la parte de código de transferencia de valores de la [Figura 1](#); esta vez se transfieren los valores a:

0012F934 D13C814E F9A923BC D716574E B0ABA423

Trazamos con F8, y llegamos a esta parte de código, que es donde se valida la bondad de nuestro código de activación:

00520319	. 8D55 90	LEA EDI, DWORD PTR SS:[EBP-70]
0052031C	. 8BC6	MOV EAX, ESI
0052031E	> 8A08	MOV CL, BYTE PTR DS:[EAX]
00520320	. 8A02	MOV AL, BYTE PTR DS:[EDI]
00520322	. 3AC8	CMP CL, AL
00520324	74 0E	JE SHORT TimePass.00520334
00520326	. C645 FF 00	MOV BYTE PTR SS:[EBP-1], 0
00520328	. EB 02	JMP SHORT TimePass.00520324

Donde en 0x52031E se toma tan sólo el primer byte de lo que hay en EAX, que no es más que:

EAX=C8BEE0

Y si volcamos en memoria:

00C8BEE0 BC1BFC15 2F8708E9 4D84DBBF D1420FC4

...que son los valores obtenidos al final de [Procesando un Texto Predefinido](#).



Ahora en 0x520320 se toma el primer byte de:

0012F934 D13C814E F9A923BC D716574E B0ABA423

obtenido en el capítulo anterior.

Y se comparan estos bytes. Si no son iguales no estamos registrados y vemos que al pulsar F9 sale el [mensaje de error](#). ¡Mala suerte!



4 INVERSIÓN DEL PROCESO

4.1 OBSERVACIONES

Ahora vamos a ir hacia atrás a partir de lo que tenemos.

En el capítulo anterior se ha realizado una comparación. Si esta se produce con los mismos valores estaremos registrados.

Hay dos bytes que se comparan, extraídos de diferentes resultados. Uno de ellos es fijo, y es el que proviene de lo que se ha analizado en el capítulo [Procesando un Texto Predefinido](#).

El otro byte proviene del resultado de haber sido procesado nuestro usuario, número de serie y código de activación.

Se trata de que ambos resultados coincidan.

¿Cómo podremos conseguirlo? Vayamos hacia atrás con los valores buenos en el capítulo [Último paso por rutina de encriptación](#).

4.2 APROVECHANDO LAS PROPIEDADES DE XOR

Para que las 4 últimas dobles palabras de este capítulo hubieran coincidido con las buenas, habría que haber introducido en esta rutina de encriptación un texto sin encriptar, como el que se procesó en el capítulo [Procesando un Texto Predefinido](#).

Ahora bien, este bloque codificado proviene de hacer un **XOR**, como se indica en la [Figura 4](#). El truco por tanto es hacer que el bloque producido en el **XOR** quede descriptado. Esto es posible por la reversibilidad de la instrucción **XOR**.

Por ejemplo, hagamos una prueba: $10 \text{ XOR } 01 \rightarrow 11$. Si ahora quiero obtener el primer término a partir del resultado (11) y del segundo término (01), quedará: $11 \text{ XOR } 01 \rightarrow 10$. Esta reversibilidad siempre se cumple para cualquier valor procesado en un **XOR**.

Así pues, tomemos el texto fijo a la entrada del capítulo [Encriptado del texto](#). Vemos que en ese capítulo se encripta el texto haciendo un **XOR** cíclicamente con dos palabras, por lo tanto, sólo necesitamos saber el resultado del primer **XOR** con las dos primeras dobles palabras del texto:



Primeras dos dobles palabras del texto:

4E6F7720 69732074

XOR

Dobles palabras de [Estableciendo Protección Comercial](#)

7AFFC10C 43B4CFB9

Como resultado, tenemos:

3490B62C 2AC7EFCD

Juego de Dobles palabras nº 1

Ahora fijémonos en el principio del capítulo [Segundo XOR](#). Las dos palabras que se usan al principio de este capítulo deben coincidir con Juego de Dobles palabras nº 1. ¿Y de dónde se obtienen estas palabras? Sigamos hacia atrás.

Llegamos al capítulo [Primer XOR](#), donde para recordar, se hacía un XOR de:

Dobles Palabras obtenidas según capítulo [Operaciones con el Nombre de usuario y el número de serie](#)

XOR

Dobles Palabras obtenidas según capítulo [Operaciones con el código de activación](#).

Gracias a la reversibilidad del **XOR**, nos limitamos a hacer otro **XOR** entre:

Juego de Dobles palabras nº 1

XOR

Dobles Palabras obtenidas según capítulo [Operaciones con el Nombre de usuario y el número de serie](#)



Obteniendo el valor que se ha de tener al acabar el capítulo [Operaciones con el código de activación](#) para que nos permita registrarnos.

4.3 EJEMPLO PRÁCTICO

En el ejemplo aquí seguido, obtenemos:

3490B62C 2AC7EFCD (**Juego de Dobles palabras nº 1**)

XOR

8C9C7FFB 8CEE612D (Proviene del usuario y número de serie)

Siendo el resultado:

B80CC9D7 A6298EE0

Juego de Dobles palabras nº 2

En resumen, tenemos lo que se ha de obtener al final del capítulo [Operaciones con el código de activación](#).

4.4 INVIRTIENDO "OPERACIONES CON EL CÓDIGO DE ACTIVACIÓN"

En este capítulo habíamos entrado con el código de activación de tipo XXXXXX-000000-XXXXXX y no había funcionado. Sin embargo, sabemos que puede tener ese formato, o al menos el código está preparado para manejar ese formato.

Vayamos hacia atrás en el código estudiado anteriormente.

La segunda palabra de Juego de Dobles palabras nº 2 debe proceder del procesado de lo que hay tras el segundo '-' de nuestro código de activación.



4.4.1 TERCER GRUPO DE CARACTERES

Si revisamos el código que produce esta segunda doble palabra (ver [Tratamiento del Resto de caracteres](#)):

```

00520525 > EB 14 JMP SHORT TimePass.0052053B
00520527 > 800C40 LEA ECX,DWORD PTR DS:[EAX+EAX*2]
0052052A > 800C88 LEA ECX,DWORD PTR DS:[EAX+ECX*4]
0052052D > 03C9 ADD ECX,ECX
0052052F > 89C8 MOV EAX,ECX
00520531 > 0FBF0A MOVSX ECX, BYTE PTR DS:[EDX]
00520534 > 83C1 BF ADD ECX,-41
00520537 > 03C1 ADD EAX,ECX
00520539 > 43 INC EBX
0052053A > 42 INC EDX
0052053B > 8B0E MOV ECX,DWORD PTR DS:[ESI]
0052053D > 85C9 TEST ECX,ECX
0052053F > 74 07 JE SHORT TimePass.00520548
00520541 > 8B0E MOV ECX,DWORD PTR DS:[ESI]
00520543 > 8B49 FC MOV ECX,DWORD PTR DS:[ECX-4]
00520546 > EB 02 JMP SHORT TimePass.0052054A
00520548 > 33C9 XOR ECX,ECX
0052054A > 3BD9 CMP EBX,ECX
0052054C > 7C D9 JL SHORT TimePass.00520527
0052054E > 8B4D 10 MOV ECX,[ARG_3]
    
```

El modo de invertirlo es sencillo:

Aux ← Valor de Segunda Doble Palabra

Bucle

Resto ← Aux **MOD** 26

Aux ← Aux - Resto

Aux ← Aux **DIV** 26

Código ASCII de Car ← Resto + 0x41

Hasta Aux = 0

Nota: 26 es el número de caracteres alfabéticos disponibles en el código ASCII. 41 es el valor hexadecimal ASCII del carácter 'A'.

Obtenemos el último carácter de la clave. Si hacemos un bucle con lo anterior y hasta que Aux sea cero, se obtiene la cadena de caracteres siguiente:

LGUSNGU

Esto irá a parar en nuestra clave al tercer lugar, donde ya tenemos: XXXXXX-000000-LGUSNGU



4.4.2 PRIMER GRUPO DE CARACTERES

La primera doble palabra del *Juego de Dobles palabras nº 2* nos proporciona las dos primeras partes de la clave.

Para empezar, tratamos la cadena de caracteres con la que empieza el código de activación, esto lo realizamos con el mismo procedimiento que el paso anterior:

```
Aux ← Valor de Primera Doble Palabra
```

```
i ← 0
```

Bucle

```
Resto ← Aux MOD 26
```

```
Aux ← Aux - Resto
```

```
Aux ← Aux DIV 26
```

```
Código ASCII de Car ← Resto + 0x41
```

```
Inc(i)
```

Hasta i = 4

Al hacer el bucle para este procedimiento, si no ponemos límite es posible que sólo se generen caracteres y no quede Aux distinto de cero para el siguiente paso, por lo tanto limitamos el bucle a 4 caracteres (4 iteraciones). Para cualquier otro número de caracteres también funcionaría.

Cuando hemos terminado el bucle, disponemos de los primeros caracteres del código de activación:

VWHR-000000-LGUSNGU

4.4.3 SEGUNDO GRUPO DE CARACTERES

Pero aún no hemos terminado. Hemos salido del bucle con *Valor de doble palabra* distinto de cero, por haberlo limitado a sólo 4 iteraciones. Ahora viene la segunda parte, que es donde se generan los números situados en el centro de la clave de activación. El algoritmo consiste en invertir el código de [Tratamiento de números](#):

**Bucle**

Resto $\leftarrow$ Aux **MOD** 10

Aux $\leftarrow$ Aux - Resto

Aux $\leftarrow$ Aux **DIV** 10

Código ASCII de Car $\leftarrow$ Resto + 0x30

Hasta Aux = 0

El resultado lo sustituimos en los '000000' del código de activación que hemos ido construyendo. Por razones obvias no lo reproducimos aquí. :P

Nota: 10 es el número de caracteres numéricos disponibles en el código ASCII. 30 es el valor hexadecimal ASCII del caracter '0'.

El límite al bucle es cuando Valor de doble palabra sea cero. Con esto ya tenemos el código de activación completo, y por tanto, cubierto el objetivo de generar un número de activación que depende del usuario y número de serie.

Si nos registramos ahora, podremos comprobar: *Barra de Menús -> Help -> About:*





4.5 HABILITANDO TODAS LAS MODALIDADES DE REGISTRO

Hay más modalidades para registrarse. Aquí se indican todas:

- Basic Edition
- Standard Edition
- Advanced Edition
- Compatibility Add-on
- Relationship Add-on
- Commercial License

Para conseguir el registro adecuado en todas, tan sólo necesitamos tener la referencia equivalente al *Juego de Dobles Palabras N° 1* para cada una de las modalidades.

Tan sólo necesitaremos como entrada del **XOR** lo equivalente a [Estableciendo Protección Comercial](#) pero para la modalidad en que deseamos registrarnos, que será generada por otro texto diferente.

El modo práctico de realizarlo es crear una tabla que contenga dos doubles palabras para cada modalidad de registro, porque siempre serán constantes.

4.5.1 HABILITANDO BASIC EDITION Y STANDARD EDITION

Como hemos empezado por el registro más difícil, este caso no implica ninguna complicación. Tan sólo cambia la parte correspondiente a [Operaciones con el Nombre de usuario y el número de serie](#), ya que tan sólo se introduce automáticamente la cadena de caracteres 'BE' o 'SE' tras el nombre de usuario.

Lo podremos ver haciendo un volcado a memoria según se explica en el capítulo [Operaciones con el Nombre de usuario y el número de serie](#).

4.5.2 GENERANDO CLAVES PARA OTRAS VERSIONES DEL PROGRAMA

El modo de operar es igual que el capítulo anterior, ya habremos comprobado que al generarse los códigos provenientes de nuestro usuario y número de serie aparecía en medio la cadena '5.0'. Tan sólo habrá que modificarla para la versión del programa que se vaya a usar, por ejemplo: '4.2'.



4.6 PSEUDOCODE

Codifica: Rutina localizada en 0x49B2EC

D. P. de Texto $\leftarrow$ Constante

D.P. Usuario+Serial $\leftarrow$ *Codifica* (User Name + Versión + Serial Number)

D.P. de Tipo Protección $\leftarrow$ *Codifica* (Texto de Tipo de Protección)

Juego de Dobles Palabras n° 1 (1) $\leftarrow$ D. P. de Texto (1) **XOR** D. P. de Tipo Protección (1)

Juego de Dobles Palabras n° 1 (2) $\leftarrow$ D. P. de Texto (2) **XOR** D. P. de Tipo Protección (2)

D. P. 1 $\leftarrow$ Juego de Dobles Palabras n° 1 (1) **XOR** D.P. Usuario+Serial (1)

D. P. 2 $\leftarrow$ Juego de Dobles Palabras n° 1 (2) **XOR** D.P. Usuario+Serial (2)

Aux $\leftarrow$ Valor de D. P. 2

Serial3 $\leftarrow$ ''

Bucle

Resto $\leftarrow$ Aux **MOD** 26

Aux $\leftarrow$ Aux - Resto

Aux $\leftarrow$ Aux **DIV** 26

Char $\leftarrow$ Resto + 0x41

Serial3 $\leftarrow$ Char + Serial3

Hasta Aux = 0

Aux $\leftarrow$ D. P. 1

Serial1 $\leftarrow$ ''

i $\leftarrow$ 0

**Bucle**

Resto $\leftarrow$ Aux MOD 26

Aux $\leftarrow$ Aux - Resto

Aux $\leftarrow$ Aux DIV 26

Char $\leftarrow$ Resto + 0x41

Serial1 $\leftarrow$ Char + Serial1

Inc(i)

Hasta i = 4

Serial2 $\leftarrow$ ``

Bucle

Resto $\leftarrow$ Aux MOD 10

Aux $\leftarrow$ Aux - Resto

Aux $\leftarrow$ Aux DIV 10

Car $\leftarrow$ Resto + 0x30

Serial2 $\leftarrow$ Car + Serial2

Hasta Aux = 0

Enabling Code $\leftarrow$ Serial1 + '-' + Serial2 + '-' + Serial3

Nota: La lista completa véase el código incluido el código Delphi



5 CONCLUSIÓN

Todo lo aquí explicado ha sido probado y se ha codificado un generador universal de claves que funciona para cualquier combinación de usuario y número de serie. Incluso son válidos código de activación no separados por guiones. Depende como deseemos codificar nuestro generador de claves, así se generarán los códigos de activación.

En fin, esperamos que haya sido de utilidad el documento y que sirva para que al encontrarnos con desensamblados semejantes no nos sintamos intimidados y podamos asumir el reto de investigar "más allá" de lo imaginable...



6 SALUDOS

Este documento va dedicado a todos mis compañeros de ARTeam y a Shub por editarlo. También un agradecimiento a Baxter, de TeamICU por haber provocado que me decidiera a realizar este trabajo. A todos los amigos de TeamICU, Seek n Destroy, RCE Forums, The Reverse Code Engineering Community, Exetools, BiW, UnPackcN, UIC, CRK, CTB y un saludo especial a crackslatinos.

Y por supuesto, a ti, que te has decidido a leer este documento, y que sin tu apoyo y colaboración no valdría la pena el tiempo dedicado a esta tarea.

Nacho_dj / ARTeam - 24 de Junio de 2008

