

ESTUDIO DE PROTECCIONES BASICO PARA PRINCIPIANTES (También llamado cracking)

Impartido por Ratón (Nivel principiante)

Nota

Cada capitulo ira acompañado de su crackme correspondiente.No facilitare paginas de donde bajarse herramientas ni enlaces a paginas de crackers, la intención es que busquéis en Internet todo lo necesario. Seguro que encontraréis mas paginas de herramientas, tutoriales y utilidades relacionadas con este tema que las que yo pueda deciros.

Con esto solo quiero fomentar vuestro interés, además se que la búsqueda os proporcionara gratas sorpresas.

A todos un saludo.

Capitulo 4

Victima

Time Trial creado por Detten

Herramientas

Olly Debugger.

Peid - Detector de protecciones.

Patch FX

Instinto

Objetivo

Aprender el manejo básico de herramientas nuevas: Patch FX

Evitar que un programa caduque por límite de tiempo.

Crear un parche o crack para ese programa.

Al ataque

Este crackme emula la protección de esos programas trial que pasado un cierto número de días caduca y nos quedamos sin poder usar (a no ser que lo compremos).

Hacemos copia del crackme y del archivo DATA.DET. Muy importante hacer esto en este capitulo

Analizamos con Peid y vemos que no esta protegido.

Ejecutamos el crackme y vemos esta ventana que nos dice que nos quedan 3 días u ocho usos o ejecuciones del programa.



Si ejecutamos 8 veces el programa aparece un cartel avisando de que pasó el periodo de evaluación, si adelantamos el reloj del sistema unos días aparece el mismo cartel.



Lo cargamos en Olly y a buscar la String mala con las strings references, la encontramos en la dirección [0040114D](#)

Dos líneas mas arriba esta el principio de la instrucción (marcado con **>**) en la dirección [00401146](#)

Pulsamos Control + R estando encima de la dirección [00401146](#) para ver desde donde se llama para que aparezca el cartel malo (sorry this crackme...) Y vemos que hay tres saltos condicionales que nos llevan hasta la zona mala.

004010B4	. A1 E4304000	MOV EAX, DWORD PTR DS:[4030E4]
004010B9	. 3BC1	CMP EAX, ECX
004010BB	√ 0F85 85000000	JNZ timetria.00401146
004010C1	. 66:8B0D B1304	MOV CX, WORD PTR DS:[4030B1]
004010C8	. 66:81F1 6969	XOR CX, 6969
004010CD	. 66:A1 EA304000	MOV AX, WORD PTR DS:[4030EA]
004010D3	. 66:2BC1	SUB AX, CX
004010D6	. 66:83F8 03	CMF AX, 3
004010D4	√ 77 6A	JA SHORT timetria.00401146
004010DC	. 2805 00304000	SUB BYTE PTR DS:[403000], AL
004010E2	> A0 B5304000	MOV AL, BYTE PTR DS:[4030B5]
004010E7	. 34 69	XOR AL, 69
004010E9	. 3C 00	CMF AL, 0
004010EB	√ 74 59	JE SHORT timetria.00401146
004010ED	. FEC8	DEC AL
004010EF	. A2 01304000	MOV BYTE PTR DS:[403001], AL
004010F4	. 34 69	XOR AL, 69
004010F6	. A2 B5304000	MOV BYTE PTR DS:[4030B5], AL
004010FB	. 6A 00	PUSH 0
004010FD	. 6A 00	PUSH 0
004010FF	. 6A 00	PUSH 0
00401101	. FF35 14314000	PUSH DWORD PTR DS:[403114]
00401107	. E8 18010000	CALL <JMP.&KERNEL32.SetFilePointer>
0040110C	. 6A 00	PUSH 0
0040110E	. 68 E0304000	PUSH timetria.004030E0
00401113	. 6A 00	PUSH 0
00401115	. 68 AB304000	PUSH timetria.004030AB
0040111A	. FF35 14314000	PUSH DWORD PTR DS:[403114]
00401120	. E8 05010000	CALL <JMP.&KERNEL32.WriteFile>
00401125	. 833D E0304000	CMF DWORD PTR DS:[4030E0], 0B
0040112C	√ 75 2D	JNZ SHORT timetria.0040115B
0040112E	. 6A 00	PUSH 0
00401130	. 68 74114000	PUSH timetria.00401174
00401135	. 6A 00	PUSH 0
00401137	. 6A 01	PUSH 1
00401139	. FF35 F4304000	PUSH DWORD PTR DS:[4030F4]
0040113F	. E8 B0000000	CALL <JMP.&USER32.ShowDialogBoxParamA>
00401144	√ EB 28	JMP SHORT timetria.0040116E
00401146	> 6A 30	PUSH 30
00401148	. 68 97304000	PUSH timetria.00403097
0040114D	. 68 76304000	PUSH timetria.00403076
00401152	. 6A 0A	PUSH A

```

Origin = FILE_BEGIN
pOffsetHi = NULL
OffsetLo = 0
hFile = NULL
SetFilePointer
pOverlapped = NULL
pBytesWritten = timetria.004030E0
nBytesToWrite = B (11.)
Buffer = timetria.004030AB
hFile = NULL
WriteFile

iParam = NULL
DlgProc = timetria.00401174
hOwner = NULL
pTemplate = 1
hInst = NULL
DialogBoxParamA

Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
Title = "Too Bad ;)"
Text = "Sorry, this crackme has expired!"
hOwner = NULL

```

Vemos los saltos que hacen aparecer el cartel Sorry... (marcados en rojo) Para ello sobre la dirección 00401146 (marcada en gris) pulsamos Control + R

Los tres saltos nos envían a 00401146

Para crackear este programa usaremos un poco de intuición o imaginación.

El objetivo es evitar que los saltos nos envíen a la zona mala.

Primero vamos a evitar que cuando lo usemos más de 8 veces aparezca el cartel.

Cerramos Olly.

Ejecutamos el programa 9 veces hasta que salga el cartel de tiempo expirado.

Abrimos Olly, lo cargamos y ponemos un Breakpoint (F2) en cada uno de los saltos para ver cual es el que nos hace saltar a la zona mala cuando el programa rebasa el límite de ejecuciones.

Run o F9 y el programa arranca e inmediatamente para en el primer Breakpoint que pusimos

004010B9	. 3BC1	CMP EAX, ECX
004010BB	√ 0F85 85000000	JNZ timetria.00401146
004010C1	. 66:8B0D B1304	MOV CX, WORD PTR DS:[4030B1]
004010C8	. 66:81F1 6969	XOR CX, 6969
004010CD	. 66:A1 EA304000	MOV AX, WORD PTR DS:[4030EA]

Jump is NOT taken
00401146=timetria.00401146

Primera parada de Olly vemos que no toma el salto

Observamos lo que nos dice Olly en la parte inferior de la ventana de ensamblado Jump is not taken = no hace el salto, por tanto no nos lleva a la zona mala, por tanto ese no es el salto que debemos alterar para evitar el limite de ejecuciones del programa.

Pulsamos F9 otra vez (una sola vez) para continuar corriendo el programa e inmediatamente para en nuestro segundo Breakpoint

```

004010D6 . 66:83F8 03 | CMP AX,3
004010DA . 77 6A | JA SHORT timetria.00401146
004010DC . 2805 00304000 | SUB BYTE PTR DS:[403000],AL
004010E2 . 00 B5304000 | MOV AL,BYTE PTR DS:[4030B5]
004010E7 . 34 69 | XOR AL,69
Jump is NOT taken
00401146=timetria.00401146

```

Segunda parada de Olly vemos que tampoco toma el salto

Lo analizamos y vemos que tampoco toma el salto, este salto tampoco se encarga del límite de ejecuciones

Pulsamos F9 otra vez (una sola vez) para continuar corriendo el programa e inmediatamente para en nuestro tercer Breakpoint

```

004010E9 . 3C 00 | CMP AL,0
004010EB . 74 59 | JE SHORT timetria.00401146
004010ED . FEC8 | DEC AL
004010EF . A2 01304000 | MOV BYTE PTR DS:[403001],AL
004010F4 . 34 69 | XOR AL,69
004010F6 . A2 B5304000 | MOV BYTE PTR DS:[4030B5],AL
Jump is taken
00401146=timetria.00401146

```

El salto si se cumple desde aquí este es el salto del límite de ejecuciones

Aquí la cosa cambia vemos que el salto si es realizado al llegar a esta instrucción nos dice Jump is taken así que si alteramos ese salto evitaremos ir a la zona mala, a la zona en la que se llama a la MessageBox con la frase Sorry this crackme has expired.

Encima del salto vemos CMP AL, 0 compara AL con cero y si es igual salta, intuimos que en AL se va registrando el numero de veces que se ejecuta el programa y cuando llega a cero se cumple la condición que pide el salto para ejecutarse.

Tenemos que intuir e imaginar cuando no sabemos, por eso es interesante ir viendo poco a poco el ensamblador (yo el primero) para ir sabiendo realmente por que hacemos las cosas, aunque de momento nos baste con estas pocas instrucciones.

```

004010E7 . 34 69 | XOR AL,69
004010E9 . 3C 00 | CMP AL,0
004010EB . 74 59 | JE SHORT timetria.00401146
004010ED . FEC8 | DEC AL
004010EF . A2 01304000 | MOV BYTE PTR DS:[403001],AL
004010F4 . 34 69 | XOR AL,69
004010F6 . A2 B5304000 | MOV BYTE PTR DS:[4030B5],AL
004010FB . 34 69 | XOR AL,69
004010FD . 3C 00 | CMP AL,0
004010FF . 74 59 | JE SHORT timetria.00401146
00401101 . 44 00 | INC AL
00401107 . 44 00 | INC AL
0040110C . 44 00 | INC AL
0040110E . 44 00 | INC AL
00401113 . 44 00 | INC AL
00401115 . 44 00 | INC AL
0040111A . 44 00 | INC AL
00401120 . 44 00 | INC AL
00401125 . 44 00 | INC AL
0040112C . 44 00 | INC AL
0040112E . 44 00 | INC AL
00401130 . 68 74114000 | PUSH timetria.00401174

```



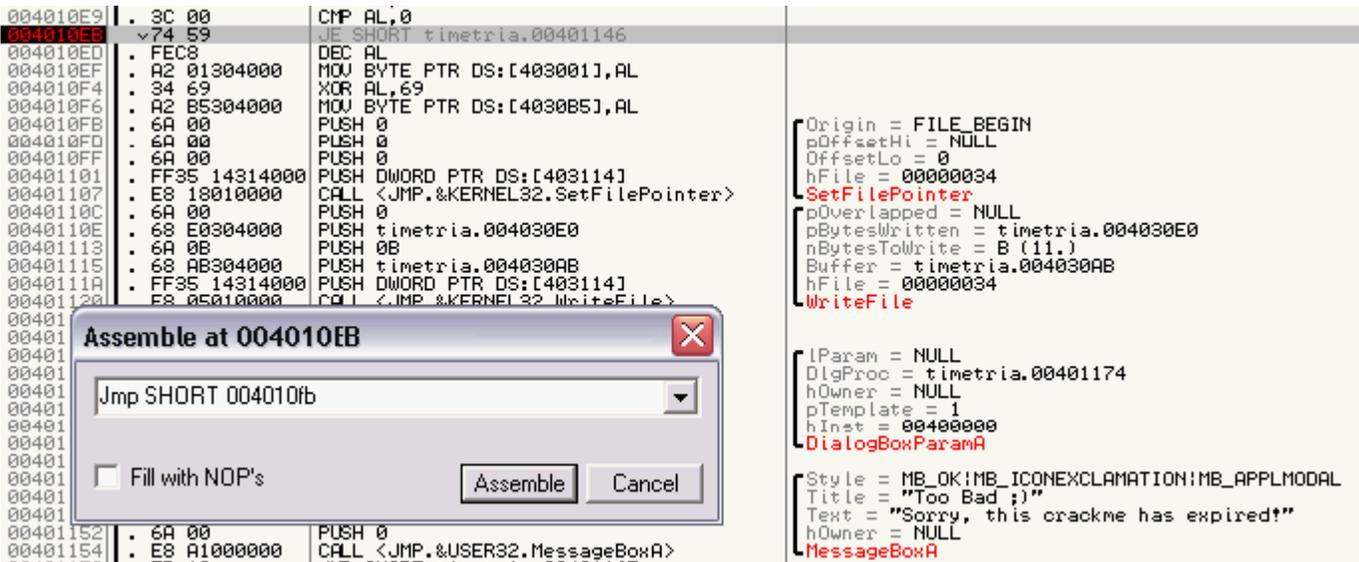
El resultado

Veremos donde nos lleva la imaginación en este caso

Cambiaremos el salto de la siguiente manera

Doble Click sobre la instrucción que vamos a cambiar

Cambiamos [JE SHORT 00401146](#) por [JMP SHORT 004010FB](#) para que salte siempre a 004010FB

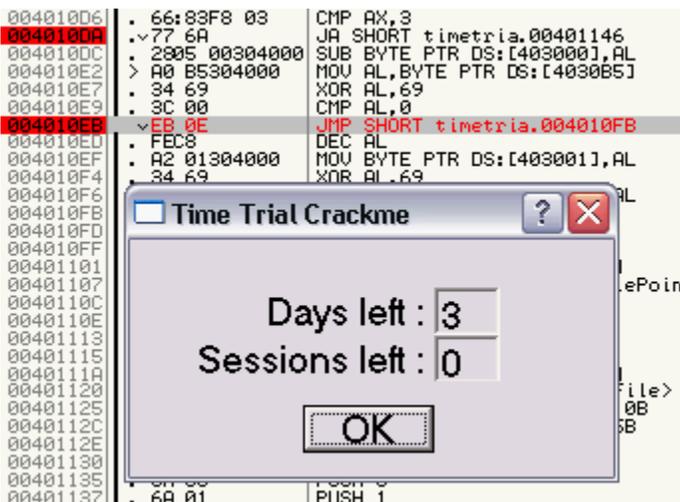


Cambiando el destino del salto

Por que allí y no a otro lugar ¿?

Tenemos que evitar el salto y que el programa siga ejecutándose, podría haber hecho que saltara a la siguiente instrucción dentro del código (004010ED) pero es un DEC una operación de decremento dos líneas mas abajo hay un XOR otra operación (Mirad las Cracker notas, por ejemplo el apartado crackear limites de runtime, aunque deberíais leerlas completas), prefiero por instinto evitarlas y me voy al sitio que me parece mejor para ubicar el destino del salto: 004010Fb un PUSH lo hago por que se parece bastante a un principio de instrucción y evito las operaciones anteriores y espero que el programa se desarrolle normalmente pese a saltarme esas operaciones.

Cambio como digo [JE SHORT 00401146](#) por [JMP SHORT 004010FB](#) para que salte siempre a [004010FB](#) y pulso F9 a ver que pasa



Premio

Pues parece que acerté ya no sale el cartel malo y el programa se ejecuta normalmente aunque queden 0 Sessions o ejecuciones del programa.

Esto se llama buscarse la vida.

Hacemos los cambios en el ejecutable con Copy to executable - Selection.

Ahora toca evitar que caduque pasados tres días

Cerramos Olly y avanzamos unos días el reloj del sistema, abrimos Olly volvemos a abrir el ejecutable (ahora retocado)

Nos quedan dos saltos por analizar de los tres que nos llevaban a la zona mala, pongamos Breakpoints en ellos y corramos Olly con F9

Para en el primero y observamos que al haber transcurrido más de 3 días el primero de los saltos cumple la condición y nos envía a la zona mala

Si lo cambiamos evitaremos que el programa caduque a los tres días.

```
004010B4 | . A1 E4304000 | MOV EAX,DWORD PTR DS:[4030E4]
004010B9 | . 3BC1 | CMP EAX,EAX
004010BB | . 0F85 85000000 | JNZ timetria.00401146
004010C1 | . 66:8B0D B1304 | MOV CX,WORD PTR DS:[4030B1]
004010C9 | . 66:01F1 6969 | XOR CX,6969
004010CD | . 66:A1 EA30400 | MOV AX,WORD PTR DS:[4030EA]
004010D3 | . 66:2BC1 | SUB AX,CX
004010D6 | . 66:83F8 03 | CMP AX,3
004010D9 | . 77 6A | JA SHORT timetria.00401146
004010DC | . 2005 00304000 | SUB BYTE PTR DS:[403000],AL
004010E2 | > A0 B5304000 | MOV AL,BYTE PTR DS:[4030B5]
004010E7 | . 34 69 | XOR AL,69
004010E9 | . 3C 00 | CMP AL,0
004010EB | . 74 59 | JE SHORT timetria.00401146
004010ED | . FEC8 | DEC AL
004010EF | . A2 01304000 | MOV BYTE PTR DS:[403001],AL
Jump is taken
00401146=timetria.00401146
```

Vemos que ahora el primer salto cumple la condición Para evitar el limite de días debemos cambiarlo

Como antes nos dio resultado cambiaremos el destino de este salto al mismo lugar del código que cambiamos el salto anterior.

mirad la imagen que sigue a estas líneas

Después de cambiarlo pulsamos F9 y ya no sale el cartel, nos quedan siempre 3 días y 8

ejecuciones según el cartelito pero realmente tanto las ejecuciones como los días son ilimitados

```

004010B4 . A1 E4304000 MOV EAX,DWORD PTR DS:[4030E4]
004010B9 . 3BC1 CMP EAX,ECX
004010BB < EB 3E JMP SHORT timetria.004010FB
004010BD . 8500 TEST DWORD PTR DS:[EAX],EAX
004010BF . 0000 ADD BYTE PTR DS:[EAX],AL
004010C1 . 66:8B0D B1304 MOV CX,WORD PTR DS:[4030B1]
004010C8 . 66:81F1 6969 XOR CX,6969
004010CD . 66:A1 EA30400 MOV AX,WORD PTR DS:[4030EA]
004010D3 . 66:2BC1 SUB AX,CX
004010D6 . 66:33F8 03 CMP AX,3
004010D8 < 77 6A JA SHORT timetria.00401146
004010DC . 2805 00304000 SUB BYTE PTR DS:[403000],AL
004010E2 > A0 B5304000 MOV AL,BYTE PTR DS:[4030B5]
004010E7 . 34 69 XOR AL,69
004010E9 . 3C 00 CMP AL,0
004010EB < EB 0E JMP SHORT timetria.004010FB
004010ED . FED8 DEC AL
004010EF . A2 01304000 MOV BYTE PTR DS:[403001],AL
004010F4 . 34 69 XOR AL,69
004010F6 .
004010FB .
004010FD .
004010FF .
00401101 .
00401107 .
0040110C .
0040110E .
00401113 .
00401115 .
0040111A .
00401120 .
00401125 .
0040112C .
0040112E .
00401130 .
00401135 .

```



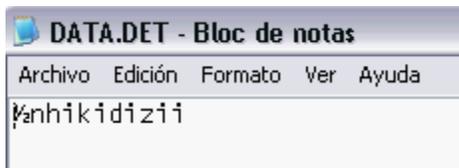
Cambiamos el destino de los saltos

Guardamos los cambios con Copy to executable Selection y ahora le damos un nombre distinto al del ejecutable para identificarlo como parcheado como por ejemplo: [timetrial_con_saltos_cambiados.exe](#).

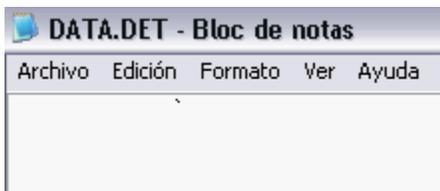
Lo hicimos.

Antes de seguir fijémonos en una cosa: este crackme viene acompañado de otro archivo llamado DATA.DET del cual dije que hiciéramos copia al principio del capítulo.

Abramos el DATA.DET que esta en la carpeta del crackme con el que hemos estado trabajando para ver que contiene, lo abrimos con el bloc de notas, en mi caso veo esta línea.



Abramos el DATA.DET que esta en la carpeta del crackme original (la copia que siempre nos reservamos) con el que no hemos trabajado para ver que contiene, lo abrimos con el bloc de notas, vemos esta línea parece que no hay nada pero fijaros hay espacios y un `



Si después de pasado el periodo de prueba hubiéramos sustituido el archivo DATA.DET que teníamos en la carpeta del crackme por la copia original habríamos vuelto a tener el programa con otros tres días u ocho ejecuciones.

Probadlo si queréis.

Este apunte es para que tengáis en cuenta que algunos programas pueden necesitar de otros archivos a la hora de comprobar vuestro registro.

Lo veremos en siguientes capítulos.

Patch FX

Tenemos ahora dos ejecutables, el original [timetrial.exe](#) y la copia con los saltos cambiados (craqueada) [timetrial_con_saltos_cambiados.exe](#)

Vamos a ver una forma fácil de hacer un crack como los que algunos piratones hacen para programas comerciales (nosotros somos buena gente y nunca haremos esto).

Abrimos Patch FX



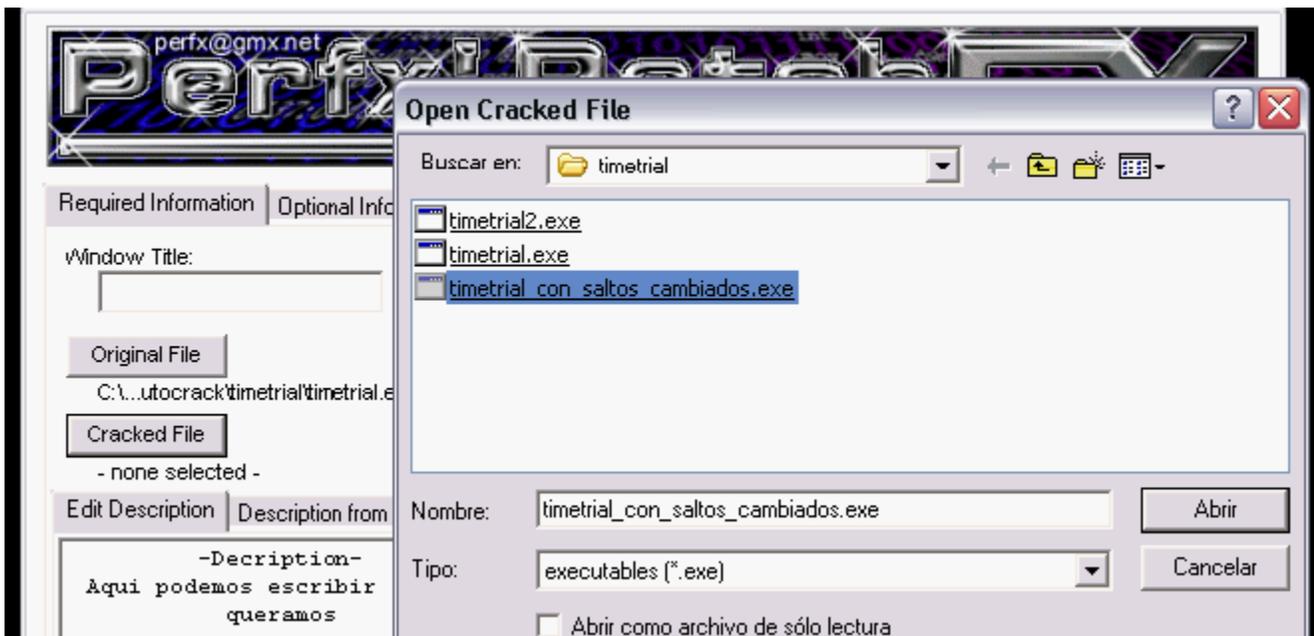
Patch FX ventana principal

En la ventana principal (Required information) pulsamos original file y cargamos el crackme original [timetrial.exe](#)



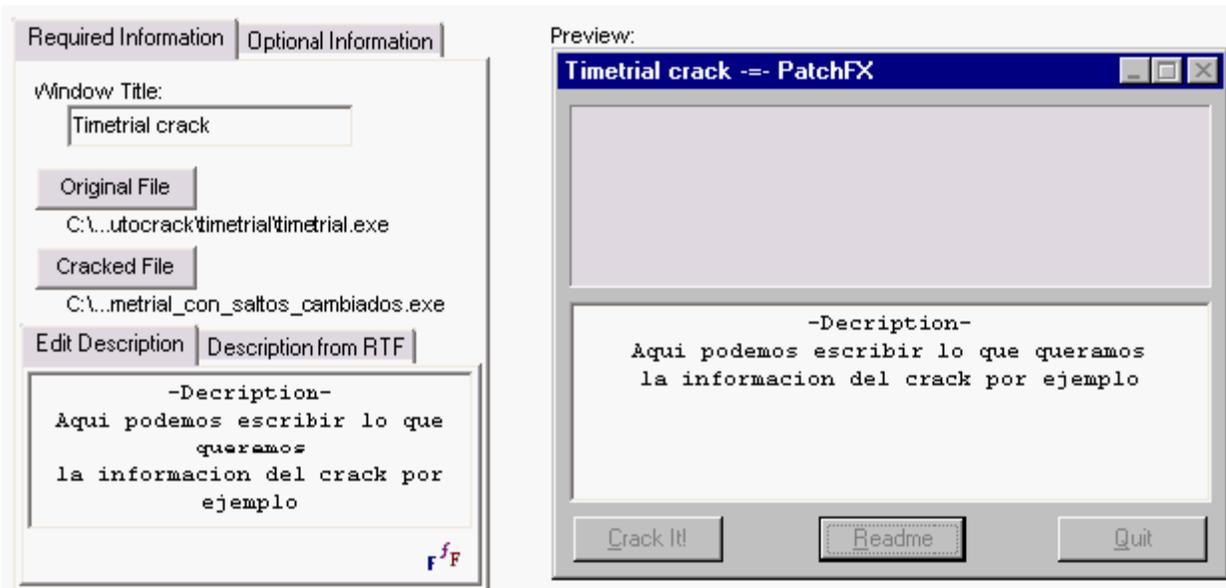
Cargando el original

En la ventana principal pulsamos Cracked file y cargamos el crackeado por nosotros [timetrial_con_saltos_cambiados.exe](#)



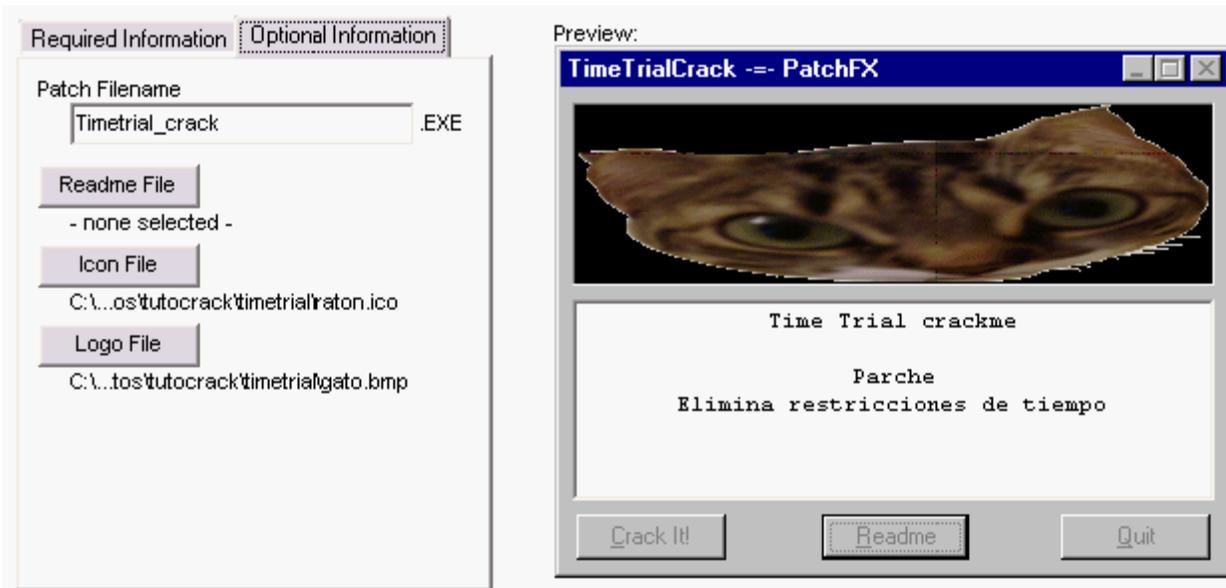
Cargando el parcheado

Le ponemos nombre a la ventana del crackme (Windows title y una pequeña descripción



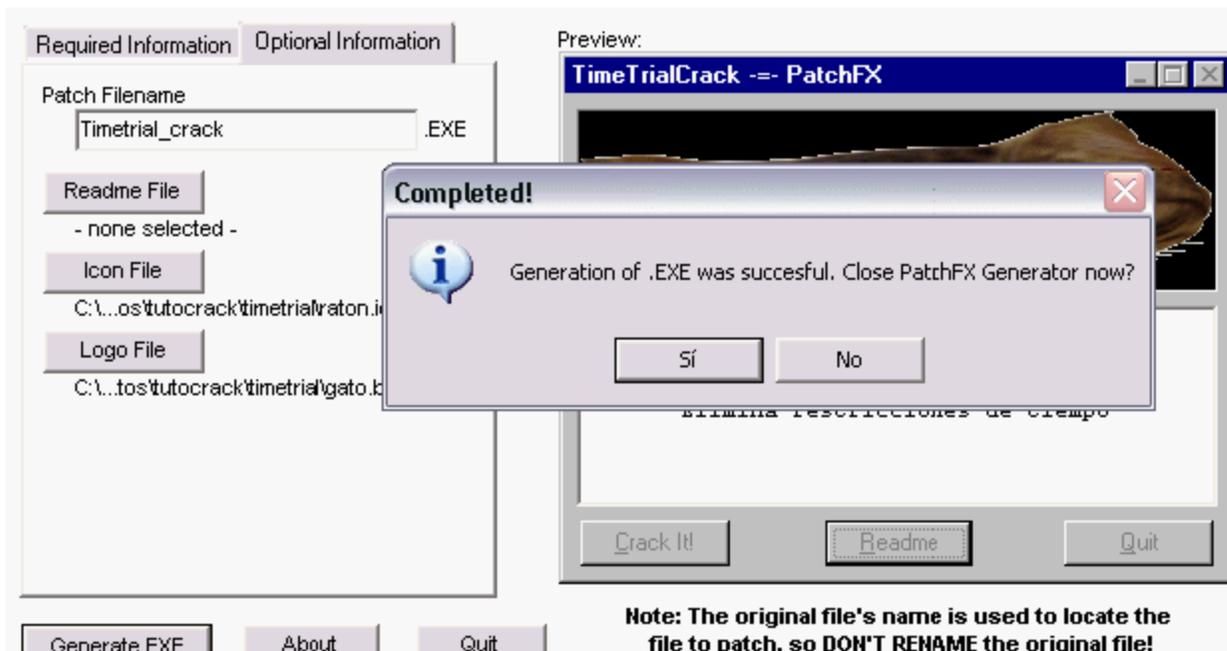
Detalle de la ventana de descripción

Pasamos a la ventana 'Optional information' y en 'Patch filename' ponemos el nombre que queremos darle a nuestro crack y si queremos un icono y una imagen (debe de tener unas medidas, el mismo programa os informa de ello), también se puede incluir un readme.txt. Para ver el resultado de este ejercicio nos bastaría con rellenar solo la 'Required information'.



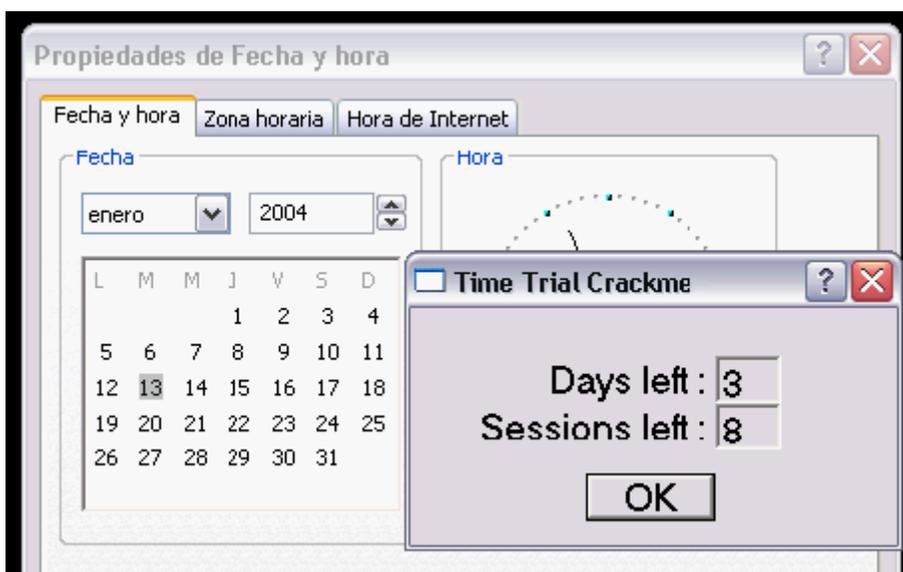
Creamos el parche (crack) a nuestro gusto

Cuando este a nuestro gusto pulsamos el botón 'Generate EXE' y aparece este cartel que aceptaremos pulsando 'SI'.



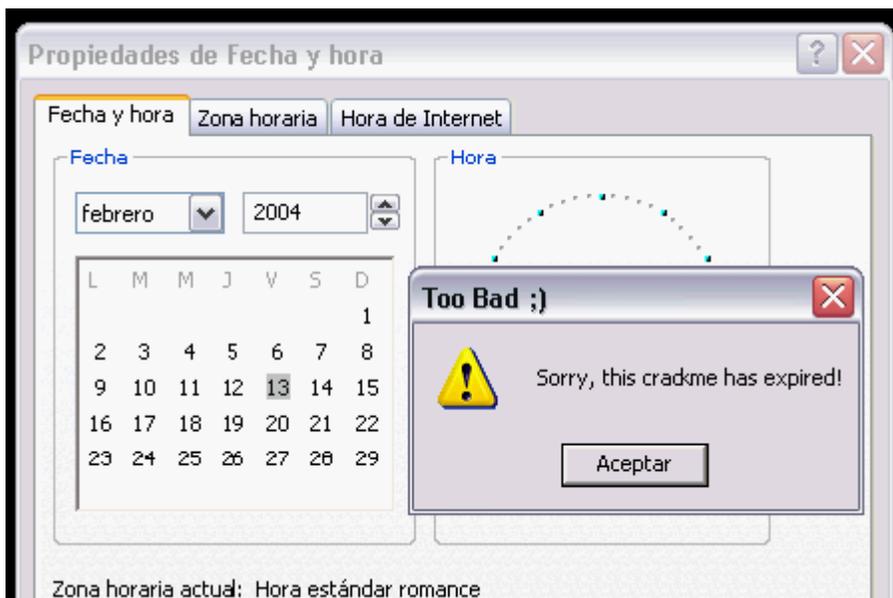
El momento de creación del crack o parche

Patch FX ha creado el crack y lo ha guardado en la carpeta donde tengáis el programa Patch FX
Lo buscamos, lo copiamos a la carpeta donde tenemos nuestros crackmes y ejecutamos [timetrial.exe](#)



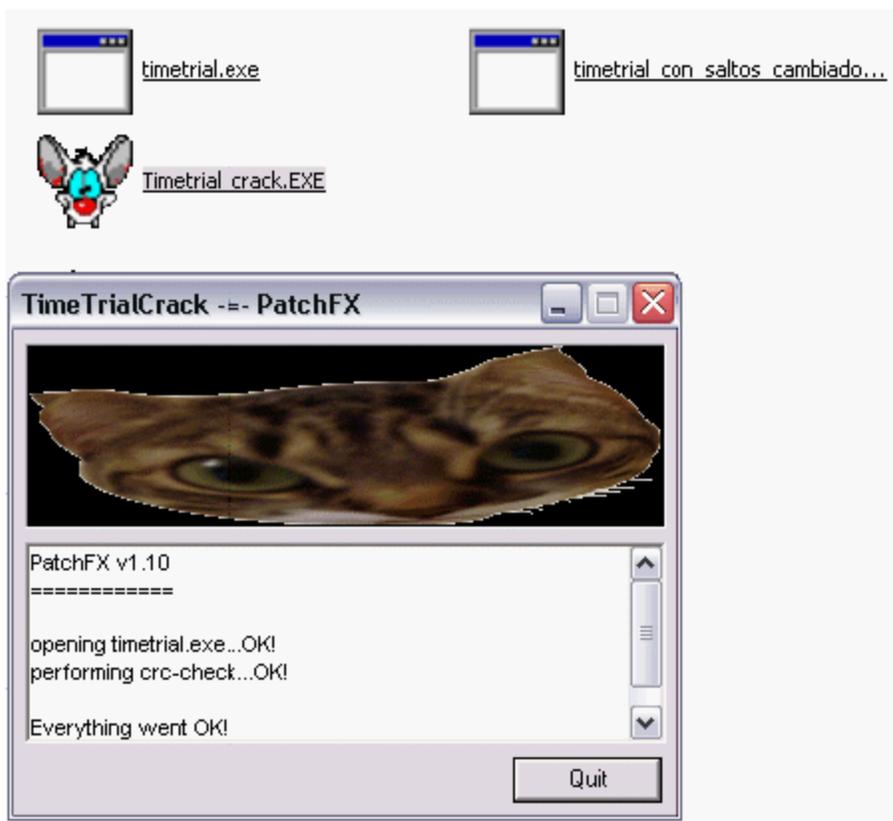
El día de hoy primera ejecución del programa

A DIA de hoy funciona, pero si cerramos el programa y adelantamos un mes el reloj cuando lo volvemos a ejecutar aparece el cartel malo



Viaje en el tiempo

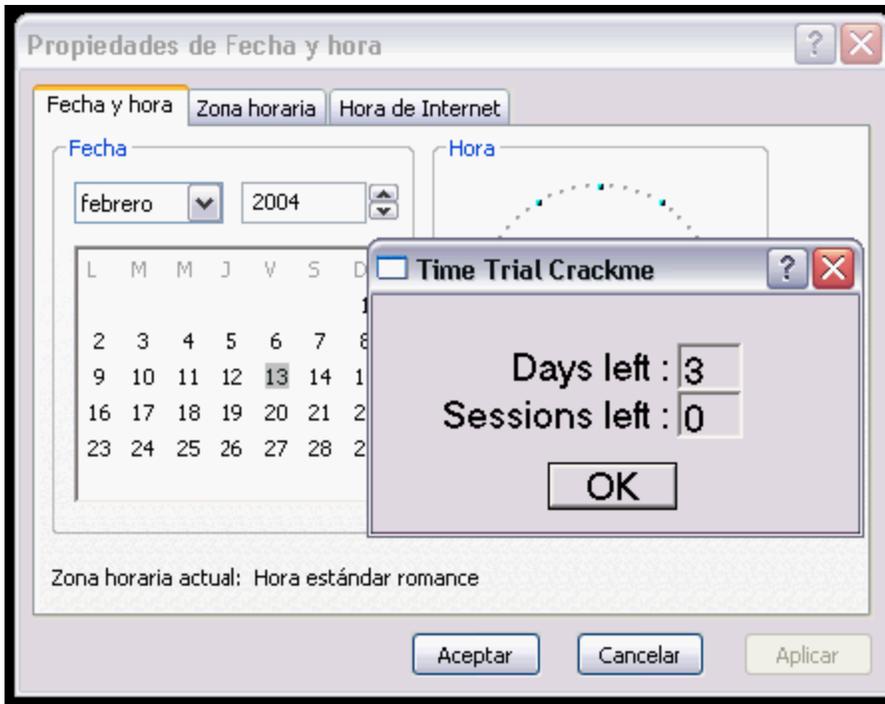
Cerramos el programa y ejecutamos el crack [Timetrial_crack.exe](#) (es el nombre que yo le di al crearlo con patch FX, podéis ver que tiene el icono que le indique al programa y en la ventana aparece el nombre y la foto que escogí al crearlo con Patch FX)



OK OK OK

Nos dice que todo Ok, comprobémoslo, pulsamos Quit para cerrar nuestro crack y abrimos el ejecutable original [timetrial.exe](#)

Mirad la fecha del reloj lo hemos parcheado con éxito, hemos creado nuestro primer crack.



Programa crackeado con éxito utilizando nuestro propio parche

Miramos al cielo y damos las gracias al creador de este programa al menos 10 veces (opcional)

Explicación ratonil del funcionamiento de Patch FX:

Este programa funciona haciendo una comparación entre los ejecutables, el original y el parcheado " ve " los bytes alterados y "se lo monta"

para hacernos un programita que cuando lo ejecutemos cambie esos bytes en el programa original.

Advertí que la explicación era ratonil.

Dentro de vosotros el lado oscuro a revelarse empieza, tranquilidad jóvenes padawans.

Un problema que podáis tener en este capítulo puede ser confusión con el nombre del crackme original y el parcheado o el cambio de fecha al jugar con el reloj.

Bueno, fin del capítulo cuarto.

Hemos analizado un nuevo tipo de protección y hemos aprendido a hacer un crack para poder aplicarlo a nuestras aplicaciones (rebuznancia).

La coña de siempre: practicadlo sin el tutorial.

Leed los documentos de apoyo adjuntos a los capítulos anteriores, os ayudaran a comprender mejor el tema.

[Para el siguiente capítulo bajaros Crackers tools aunque con la calculadora de Windows servirá.](#)

Gracias

A todas las personas que colaboran desde el foro de HackxCrack para llevar adelante el curso, tanto los que colaboran aportando sus conocimientos como complemento al curso como a los que postean sus dudas para que aprendamos todos y por supuesto a los moderadores del mismo

A todos los crackers y programadores de los cuales he aprendido y sigo aprendiendo.

A los creadores de crackmes

En especial y sin menospreciar a nadie a **Ricardo Narvaja** por su aportación y su trabajo sobre el estudio de las protecciones y sus tutoriales en castellano y a **Makkakko** por sus tutoriales con Olly Debugger (Recomendados 100%) y por supuesto a **Shoulck** por la ayuda desinteresada que me esta prestando a costa de algo tan preciado como su tiempo.

A ti que me estas leyendo

Ratón Enero 2004