



Software	Productos Movavi (www.movavi.com)
Protección	SERIAL. Activación por Internet. Themida/WinLicence
HERRAMIENTAS	<p>Windows 7 Home Premium SP1 x32 Bits (SO donde trabajamos) X64DBG (Feb 14 2018)</p> <ul style="list-style-type: none"> • Plugin ScyllaHide Release v1.4 <p>Exeinfo PE v0.0.5.6 sign 2019.05.22 HxD v1.7.7.0 dUP2 Diablo's Universal Patcher v2.26.1</p> <p>DESCARGAR HERRAMIENTAS</p> <p>DESCARGAR TUTO+ARCHIVOS</p>
SOLUCIÓN	Crear nuestro Activador (PATCH).
AUTOR	LUISFECAB
RELEASE	Julio 16 2019 [TUTORIAL 017]

INTRODUCCIÓN

Mi hermanita acababa de salir de vacaciones de la universidad y me contó que tuvo que hacer un trabajo de la universidad, que era hacer un video. Resulta que tenía plazo de subir ese video hasta media noche y como siempre estaba al límite de tiempo, así que buscó por Internet algún programa para hacer videos y halló el <Movavi Video Editor>, y lo utilizó confiada de que si le colocaba alguna marca de prueba fuera algo decente, pero la realidad es que le terminó colocando un aviso que le tapaba casi todo el video, y pues ni modos, ya cerca de la media noche no tuvo otra opción que enviarlo con esa cosa horrorosa pegado al video.

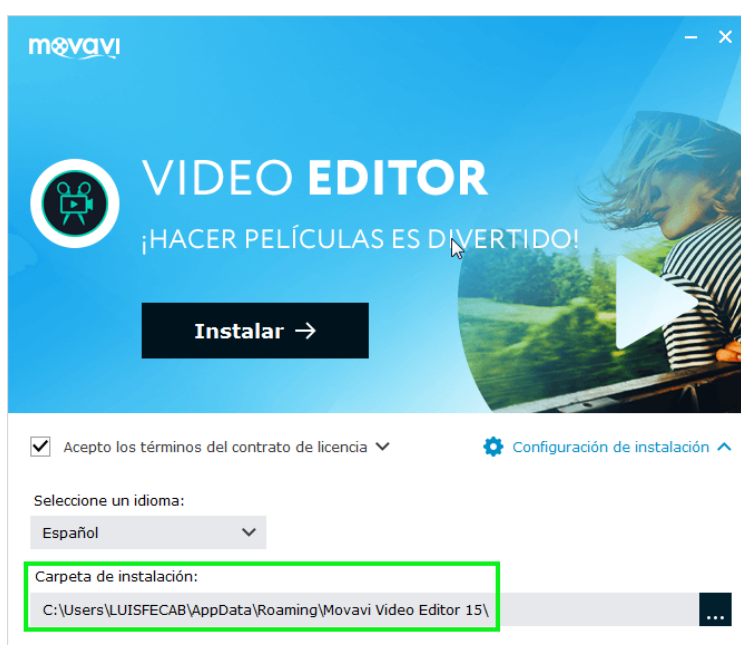
Cuando me contó lo sucedido y que todavía no le pasaba la rabia por ese aviso, le dije que tomaríamos venganza. Está bien que pongan algo, pero que no se pasen de esa forma. Y parece que todo se alineó para que tomáramos nuestra venganza y que además pudiera escribir este tutorial, recordé que DavicoRm hace rato lo había reventado y hasta nos compartió un Patch hecho con el <dUP2 Diablo's Universal Patcher>, más lo que me contó mi hermana y si le sumamos que unos días antes me había comunicado con DavicoRm para que me compartiera el <SpeedConnect Internet Accelerator v8.0 Keygen by URET> que está codeado en .NET y que tiene la opción para hacer un Patch, pues me decidí a crackear el <Movavi Video Editor> y programar mi propio Patch.

Espero que este tuto aporte algo nuevo y que sea de ayuda para futuras consultas, creo que lo más relevante y nuevo es el Patch en .NET. Saludos a mis amigos de CracksLatinoS y PeruCrackerS.

ANÁLISIS INICAL

Este programa <Movavi Video Editor> viene para 32 o 64 Bits; al ir a su página web para descargar la versión de prueba se descargará la versión adecuada según tu sistema operativo. El de mi hermana era de 64 Bits, y con ese fue que lo crackie inicialmente, pero este escrito lo voy a hacer en una versión de 32 Bits. La diferencia prácticamente no es nada porque al final de cuentas utiliza los mismos procedimientos.

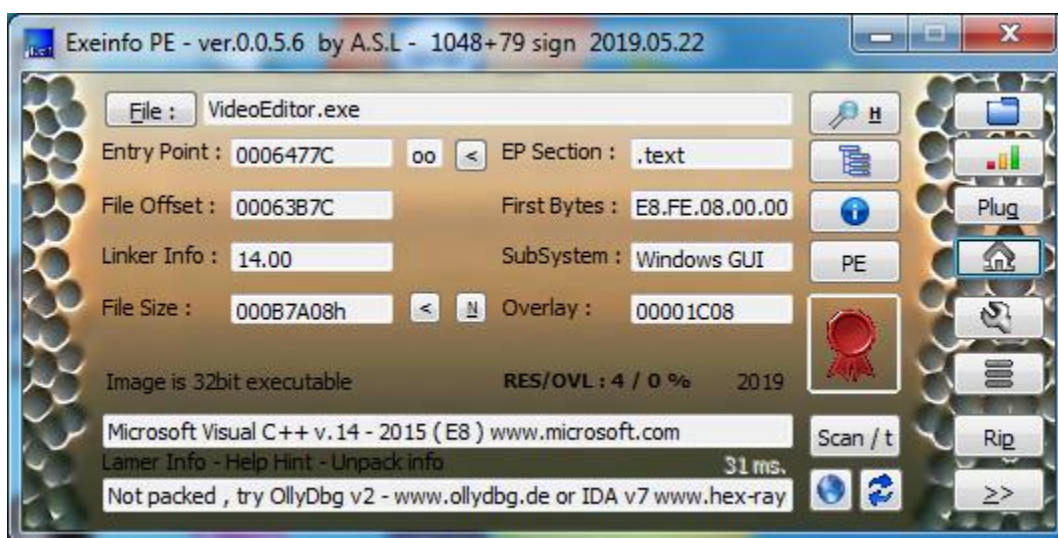
Instalamos el programa como hacemos con todos, nada del otro mundo.



Lo único que llama la atención es que su carpeta de instalación por defecto no es la **%Program files%** o **%Archivos de programa%**. Eso no implica nada, solo que sus programadores no quieren dejar su obra con el resto del montón. Terminamos de instalarlo y lo ejecutamos.



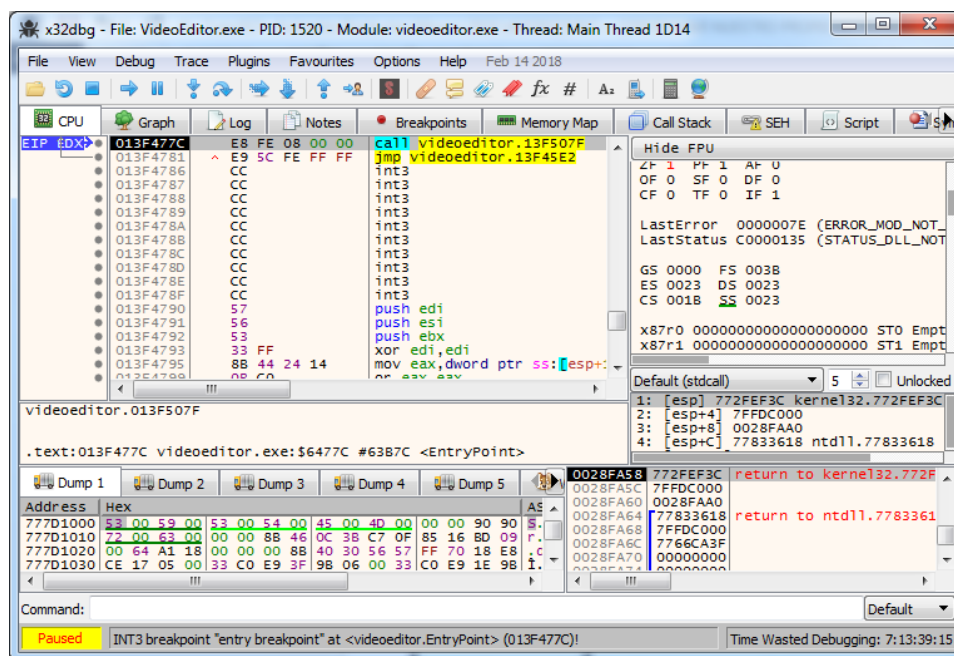
Desgraciadamente mi periodo de prueba, que son 7 días, ya caducó. En verdad son bien marrulleros, no les basta con tirarse los videos que uno hace con su espantosa marca, sino que también después de 7 días ya te joden diciéndote que ya no puedes seguir usando su producto. Más adelante cuando pillemos su "**ZONA CALIENTE**", podremos arrancarlo como si tuviéramos días de prueba y ver con qué nos sale. Bien, pues ni modo, cerrémoslo y revisamos el ejecutable <**VideoEditor.exe**> con el <**Exeinfo PE v0.0.5.6 sign 2019.05.22**>.



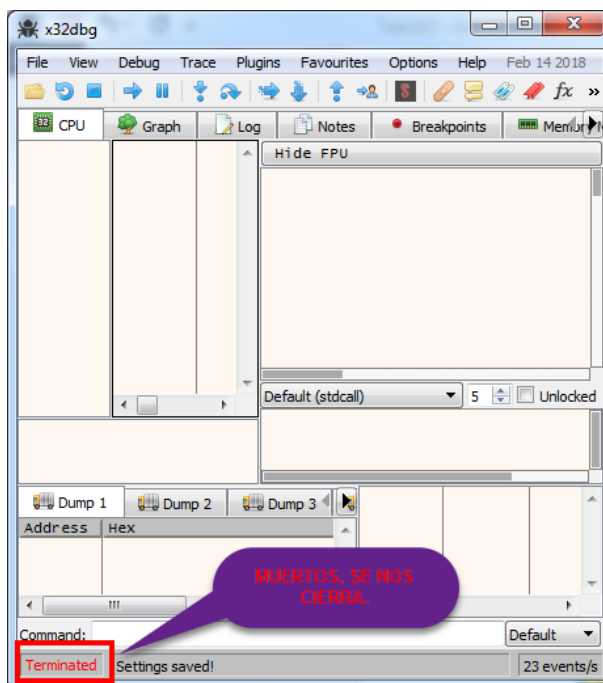
Pero si no trae nada extraño, dice que es un **MVC++** y que no tiene Packer, cosa que es maravilloso porque de lo contrario no podría crackearlo, pero entonces por que dije que en **Protección** tiene un **Themida/WinLicence**; pues eso será respondido en la siguiente sección.

AL ATAQUE

Entrémosle de una, lo abrimos con nuestro <x64DBG>.

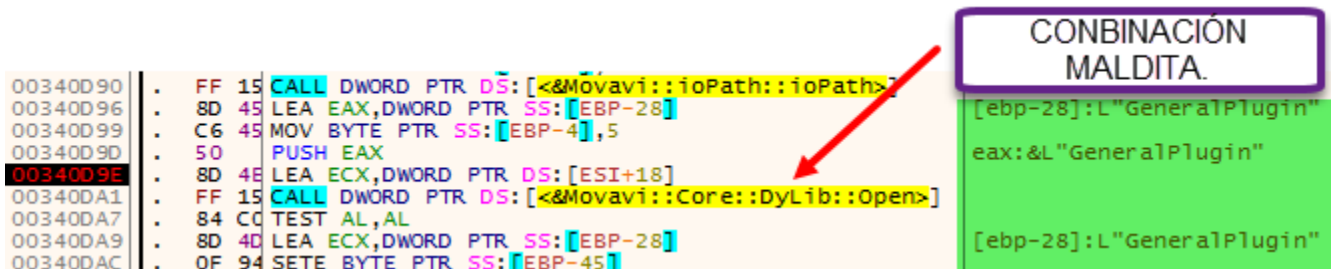


Ejecutémolo con <F9> para ver si nos corre el programa.

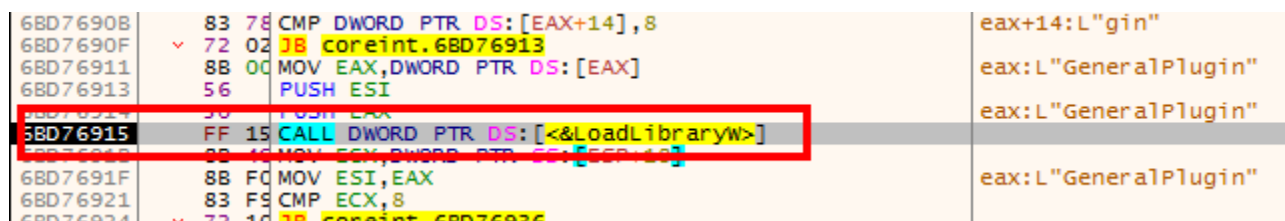


El programa no se ejecuta, nos termina la ejecución. Este programa tiene algo oculto que no nos deja **Debuggearlo**. Yo lo que hice como novato que soy, fue tracearlo con mucha paciencia hasta ir acercándome lentamente al lugar donde todo se iba al

carajo. La verdad nunca llegué a ese lugar, pero ese traceo que repetí sin parar por todo un día, me permitió ir distinguiendo lo que el programa iba haciendo y pude notar que utilizaba una **DLL** de nombre `<CoreInt.dll>` y que curiosamente cuando cerca de ella tenía la **String** `"GeneralPlugin"` me terminaba jodiendo.



Esa combinación maldita termina por cerrarme todo. Supongo que la tal `<CoreInt.dll>` ha de ser fijo librerías de **C++** para hacer correctamente su trabajo. Bueno, revisando por todos lados y hasta nopeando instrucciones, invirtiendo saltos y cambiando las direcciones de ejecución de código me llevó a sospechar de ese `"GeneralPlugin"`. Si seguimos traceando dentro del `CALL DWORD PTR DS:[<&Movavi::Core::DyLib::Open>` hasta llegar al `CALL DWORD PTR DS:[<&LoadLibraryW>]`.



Miremos qué podemos saber de esa **API**.

NAME

LoadLibraryA (KERNEL32.@)

SYNOPSIS

```
HMODULE LoadLibraryA(
    LPCSTR libname
)
```

DESCRIPTION

Load a dll file into the process address space.

PARAMS

libname [In] Name of the file to load.

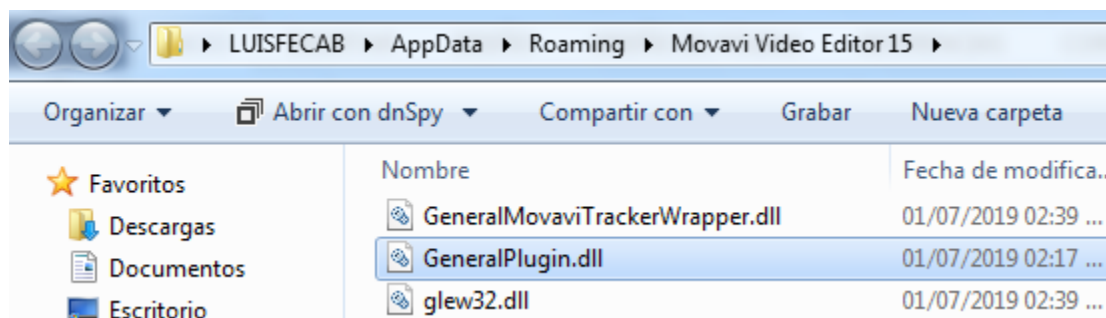
RETURNS

Success: A handle to the loaded dll.

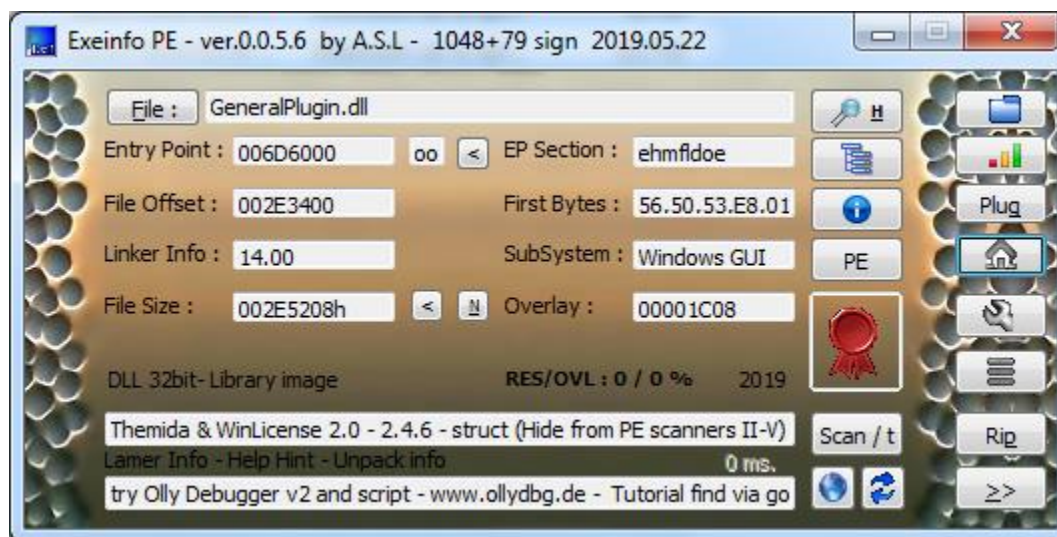
Failure: A NULL handle. Use [GetLastError](#) to determine the cause.

Ajá! Dice que se utiliza para cargar una **DLL**. Creo que si es la **Kernal32.LoadLibraryW** puede cargar otros tipos de módulos pero para nuestro caso cargara la **DLL** `<GeneralPlugin.dll>`. Esta API tiene un solo parámetro, **libname**, que es el nombre

del archivo. Buscando un poco más información de esta **API**; el parámetro **libname** puede incluir también la dirección donde está ubicada la **DLL**. Recordemos que los programas buscan inicialmente las **DLL** donde se están ejecutando y parece que este es nuestro caso porque si tratamos de pasar esa **API**, **CALL DWORD PTR DS:[&LoadLibraryW]** con **<F8>**, terminamos mal. Esa **DLL** es la "rata canequera" que nos está jugando chueco.



Ahí está, dentro del directorio de instalación del **<Movavi Video Editor>**. Entonces, revisémosla con el **<Exeinfo PE v0.0.5.6 sign 2019.05.22>**.



¡Madre mía! Estoy jodido, eso pensé yo; no puedo desempacar ni mi maleta de viaje, mucho menos podré con este Packer que es considerado duro y que viene en pacha.



Themida®

Avanzado Sistema de Protección Software para Windows

Avanzado sistema de protección software desarrollado para programadores software que desean proteger sus aplicaciones con las tecnologías más avanzadas contra la ingeniería software y la piratería.

[Resumen](#) [Características](#) [Imágenes](#) [Descargar](#) [Comprar](#)



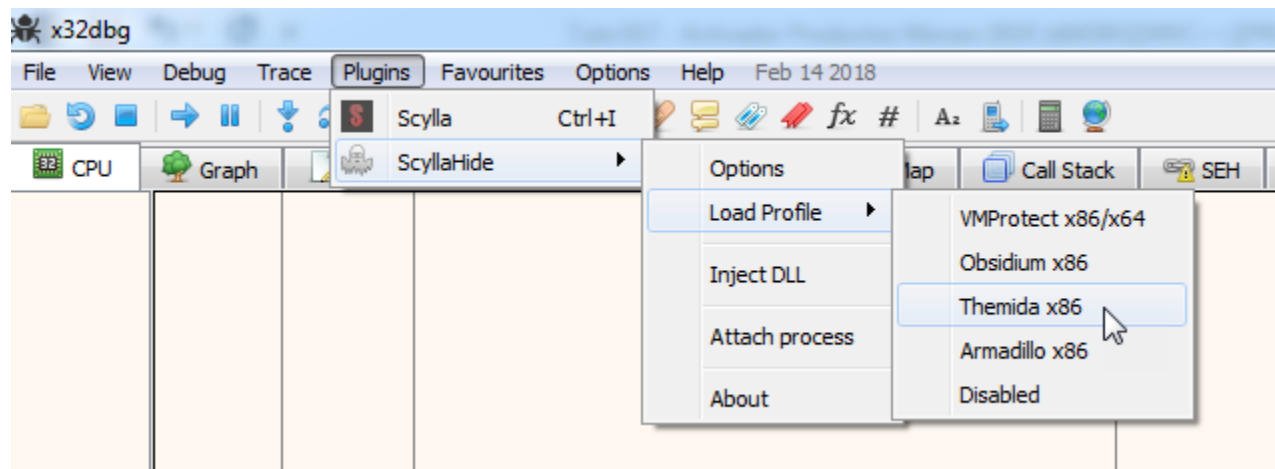
WinLicense®

Protector de Software Profesional y Gestor de Licencias

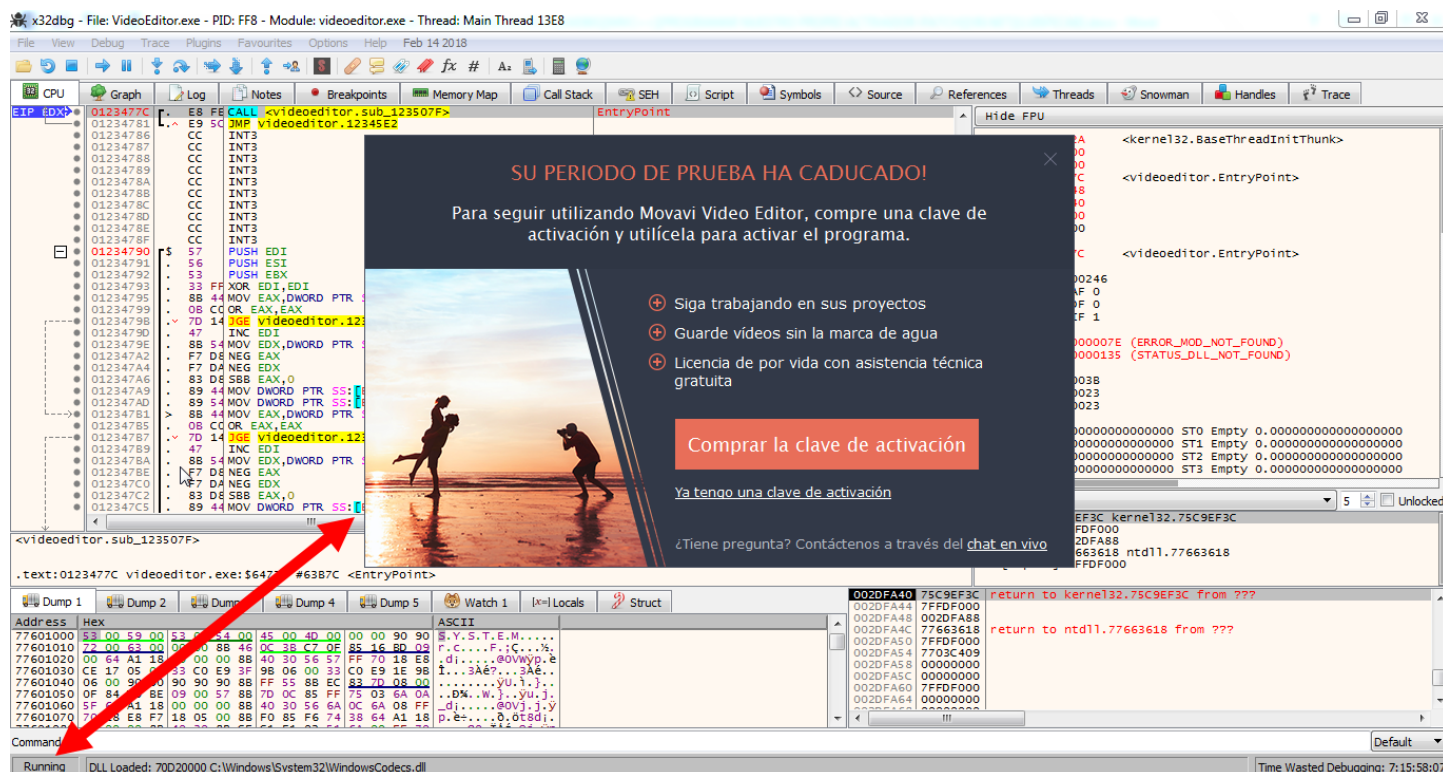
Combina el poder de la protección software de Themida junto a un avanzado sistema de gestión de licencias, ofreciendo una amplia gama de técnicas de registro de software, las cuáles permiten una segura distribución de versiones trial para todo tipo de aplicaciones.

[Resumen](#) [Características](#) [Imágenes](#) [Descargar](#) [Comprar](#)

Pero pensé que sería bueno poder hacer correr el programa en el **<x64DBG>** (por lo menos aunque fuera eso), y es aquí en donde entra en juego el **Plugin ScyllaHide**. El **Plugin ScyllaHide** es una parte vital que todos debemos tener en nuestro **<x64DBG>**; en mi caso nunca se lo agregué a mi **<x64DBG>** porque nunca lo había necesitado, claro que ya conocía de su gran capacidad de protegernos de técnicas antidebug pero me había dicho a mí mismo que lo utilizaría cuando el momento de ser puesto a prueba llegara, y este ha llegado. He utilizado la versión **<ScyllaHide Release v1.4>** que creo no es la versión más actual pero es que esta la tenía esperando para este momento.



Trae configuraciones predeterminadas para cuatro Packers que son considerados duros, hay que tener buenas bases y experiencia en Unpacking para enfrentarlos. Como vemos, ahí aparece la opción que dice **"Themida x86"**. La seleccionamos y probamos si esta vez si corre el **<VideoEditor.exe>**.



Una maravilla, el programa se ejecutó sin problemas, el Plugin <ScyllaHide Release v1.4> hace un magnífico trabajo, nos protege del Themida/WinLicence y puede que hasta de más.

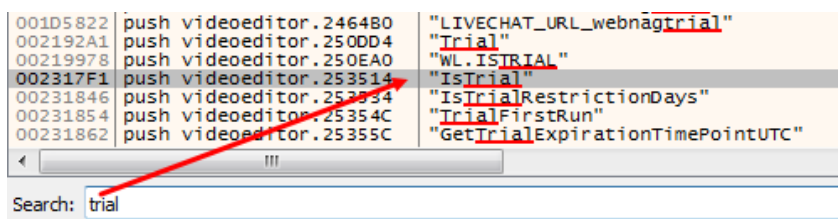
Luego recordé que con roce2005, un amigo de la lista CrackSLatinos que trabaja con el <CONCAR con Loader>, que hicieron por separado DavicoRm e Invison; son dos tutoriales muy buenos, [1614](#) y [1636](#). Pues resulta que crackeamos un programa adicional que trae el <CONCAR> para el 2019 y es el <e-CONCAR Pro 2019> que es para facturas electrónicas XML. Bueno, el cuento es que utilizaba una DLL desde la cual el programa hacía la validación. Esta DLL traía un Dongle HardKey con Themida y lo que hicimos fue forzar nuestra respuesta deseada desde el ejecutable y no desde la DLL. Aquí haremos lo mismo, que el <VideoEditor.exe> llame a su fulana <GeneralPlugin.dll> para validar su estado de Activación que traerá la respuesta no deseada, pero que nosotros cambiaremos después en el <VideoEditor.exe> cuando regrese a este. Con eso nos evitamos qué lidiar con esa DLL que está protegida hasta los dientes, y claro está, agradecer a la suerte que el <VideoEditor.exe> no estuviera empacado con Themida porque ahí sí, este novato aficionado a escribir tutoriales para todos ustedes hubiera quedado reventado y no reventando estos productos de MOVAVI.


Ya pasando por alto y no enfrentarnos a la <GeneralPlugin.dll>. El siguiente paso era llegar a la "ZONA CALIENTE" del <VideoEditor.exe> donde se hacía la validación llamando la <GeneralPlugin.dll> y ahí poder cambiar la respuesta de retorno de la <GeneralPlugin.dll>. ¿Y cómo llegué a la "ZONA CALIENTE"?, pues de pura suerte, como les conté hace rato, probando por todos lados, nopeando, invirtiendo saltos y probando mi intuición. Aquí no tengo algo nuevo que aportar, todo fue un ensayo y error, hasta que llegué al lugar correcto.

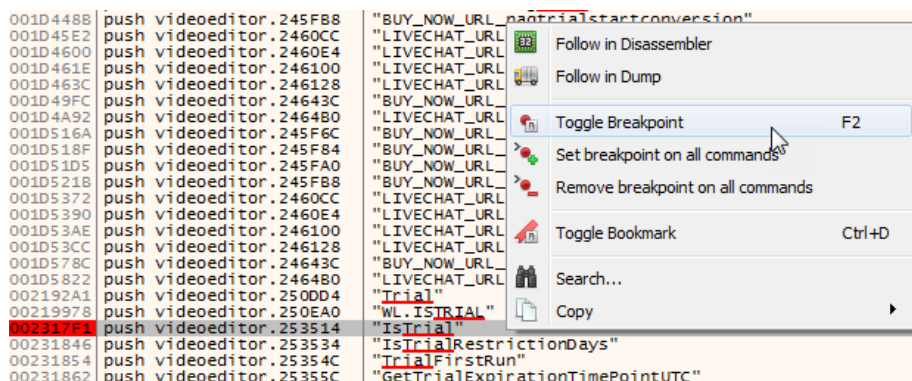
Address	Disassembly	Comment
012317E0	PUSH EBX	
012317E1	MOV EBX,DWORD PTR SS:[ESP+8]	
012317E5	PUSH ESI	
012317E6	PUSH EDI	
012317E7	LEA EDI,DWORD PTR DS:[ECX+18]	
012317EA	MOV DWORD PTR DS:[EBX+1C],0	
012317F1	PUSH videoeditor.1253514	1253514: "IsTrial"
012317F6	LEA ESI,DWORD PTR DS:[EBX+1C]	
012317F9	MOV BYTE PTR DS:[EBX],1	
012317FC	PUSH EDI	
012317FD	MOV DWORD PTR DS:[EBX+4],0	
01231804	MOV DWORD PTR DS:[EBX+8],0	
01231808	MOV WORD PTR DS:[EBX+C],1	
01231811	MOV DWORD PTR DS:[EBX+10],0	
01231818	MOV DWORD PTR DS:[EBX+14],0	
0123181F	MOV DWORD PTR DS:[EBX+18],0	
01231826	CALL <videoeditor.sub_122CC10>	
01231828	PUSH videoeditor.125351C	125351C: "DaysLeft"
01231830	PUSH EDI	
01231831	MOV BYTE PTR DS:[EBX],AL	
01231833	CALL <videoeditor.sub_1228C90>	
01231838	PUSH videoeditor.1253528	1253528: "TotalDays"
0123183D	PUSH EDI	
0123183E	MOV DWORD PTR DS:[EBX+4],EAX	
01231841	CALL <videoeditor.sub_1228C90>	
01231846	PUSH videoeditor.1253534	1253534: "IsTrialRestrictionDays"
01231848	PUSH EDI	
0123184C	MOV DWORD PTR DS:[EBX+8],EAX	
0123184F	CALL <videoeditor.sub_122CC10>	
01231854	PUSH videoeditor.125354C	125354C: "TrialFirstRun"
01231859	PUSH EDI	
0123185A	MOV BYTE PTR DS:[EBX+C],AL	
0123185D	CALL <videoeditor.sub_122CC10>	
01231862	PUSH videoeditor.125355C	125355C: "GetTrialExpirationTimePointUTC"
01231867	PUSH EDI	

En la dirección 012317E0 se inicia la rutina donde valida todo cuando ejecutamos el <VideoEditor.exe>. Pero antes un poco de reflexión propositiva. Me encanta

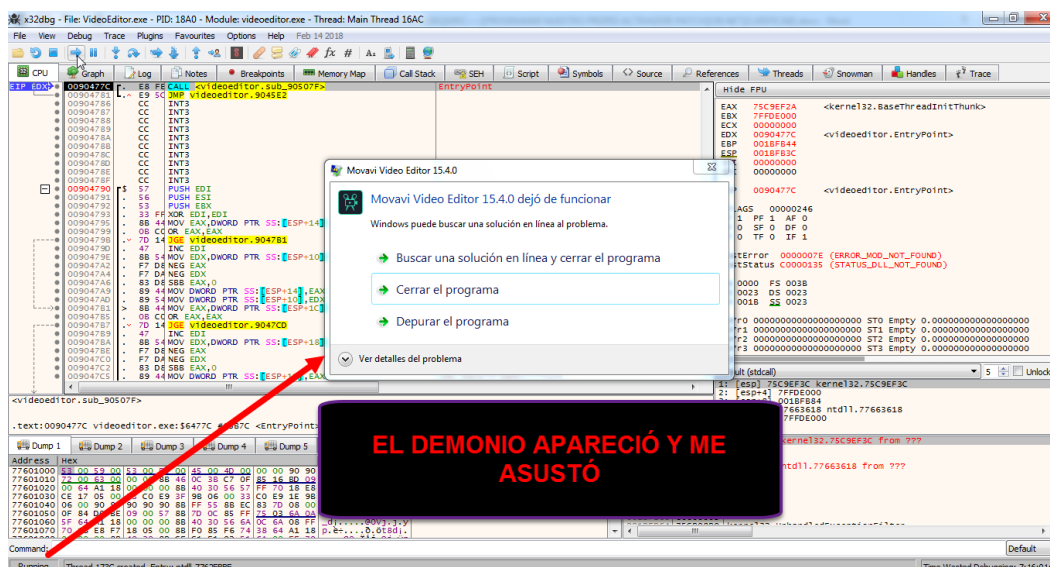
escribir estos tutoriales porque siempre aprendo algo nuevo o me recuerdan que pude haber analizado de forma diferente la situación, por ejemplo, ahora que llegué a la **"ZONA CALIENTE"** hay unas **Strings** que contienen **"Trial"**. En uno de mis tutoriales, el [1662](#), escribí que era muy recomendable buscar en las **Strings** la palabra **"Trial"** que de esa forma es posible llegar a la zona de interés, y entonces qué me pasó, por qué no hice eso; pues por novato.



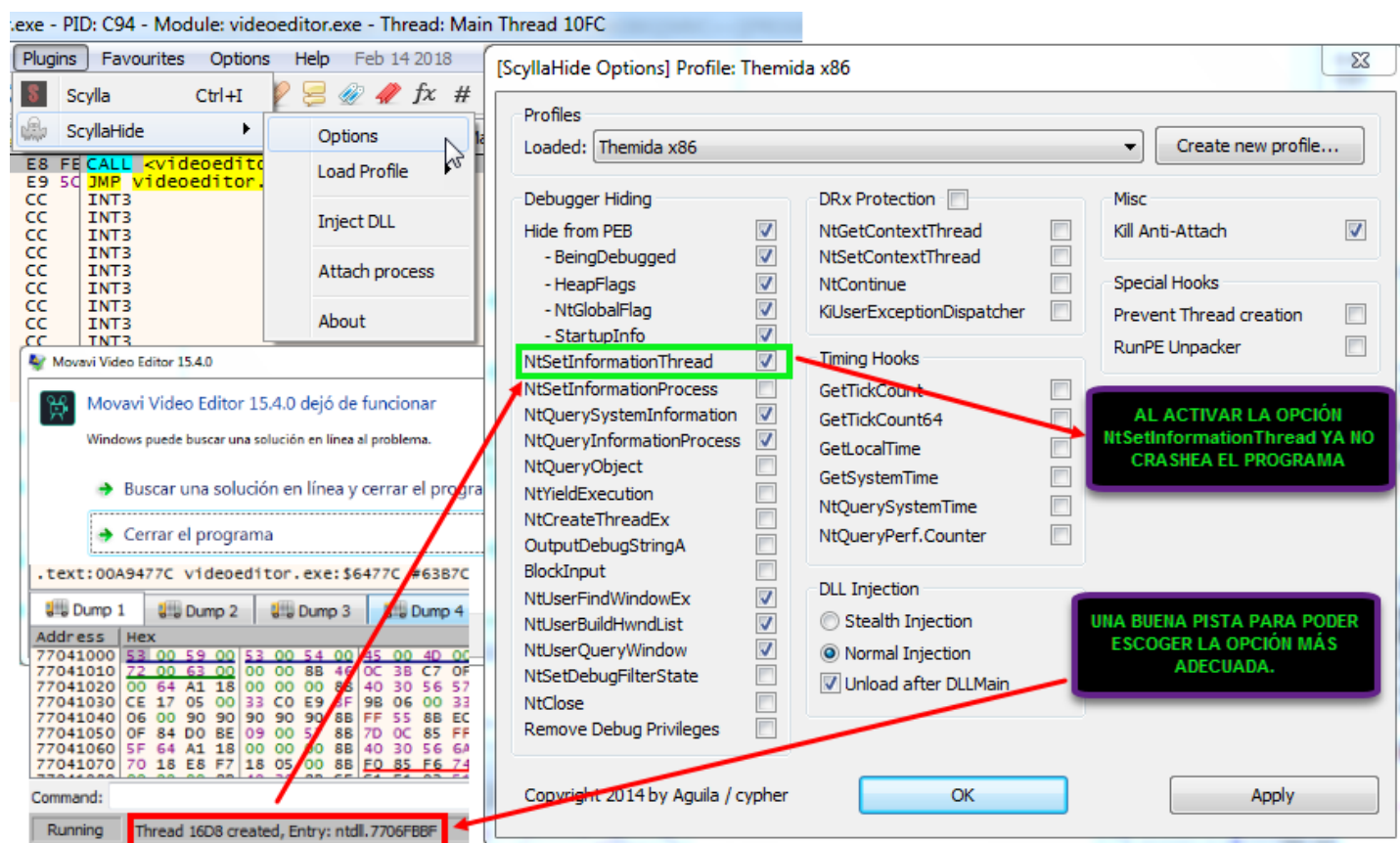
Al reiniciar el programa con  y al realizar la búsqueda aparecen unas cuantas pero no miles para hacerlo imposible, además ya con lo que hemos aprendido podríamos haber escogido las que potencialmente serían las más indicadas, es más, habíamos podido ponerle su **<BREAKPOINT>** a todas e ir traceando en cada parada, seguro eso hubiera sido mejor que estar probando a medio ciegas. Este novato nada que aprende.



Haber novato **LUISFECAB** ponle un **<BREAKPOINT>** a la **String "IsTrial"**, recuerda que ya sabemos que esa es la buena y córrelo con **<F9>**.



Pero qué es esto, si todo parecía ir lo más de bien. ¡Nooooooo! Y ahora me sale esta sorpresa. Bueno, esto fue lo mejor que me pudo pasar, porque así podemos escribir algo nuevo para tener en cuenta, si en futuros trabajitos nos sucede algo parecido. Aquí es este caso en particular el novato **LUISFECAB** como que si pensó un poquito y después de 16 tutoriales aprendió algo, bueno en realidad no son sus tutoriales, si no haber hecho el curso del **Maestro Ricardo**, [OLLY DESDE CERO](#). Entonces pensé, esto sucedió cuando puse el <BREAKPOINT>, Uhhhh! el **Themida** me está pillando ese <BREAKPOINT> así que tocará ir agregando opciones a la configuración predeterminada que trae el **Plugin <ScyllaHide Release v1.4>** y ver si logro hacerlo correr con el <BREAKPOINT> activado.



De esa forma encontré la solución a ese impase. Probé como cuatro veces, hasta que le acerté a la opción adecuada y era seleccionar **NtSetInformacionThread**, claro y si hubiera sido más observador me hubiera fijado en la pista que me dio el <x64DBG>. Reiniciemos y ejecutamos con <F9>.

```

011717EA . C7 43 MOV DWORD PTR DS:[EBX+1C],0
011717F1 . 68 14 PUSH videoeditor.1193514 1193514:"IsTrial"
011717F6 . 8D 73 LEA ESI,DWORD PTR DS:[EBX+1C]
011717F9 . C6 03 MOV BYTE PTR DS:[EBX],1
011717FC . 57 PUSH EDI
011717FD . C7 43 MOV DWORD PTR DS:[EBX+4],0
01171804 . C7 43 MOV DWORD PTR DS:[EBX+8],0
01171808 . 66 C7 MOV WORD PTR DS:[EBX+C],1
01171811 . C7 43 MOV DWORD PTR DS:[EBX+10],0
01171818 . C7 43 MOV DWORD PTR DS:[EBX+14],0
0117181F . C7 43 MOV DWORD PTR DS:[EBX+18],0
01171826 . E8 58 CALL <videoeditor.sub_116CC10> 119351C:"DaysLeft"
01171828 . 68 1C PUSH videoeditor.119351C
01171830 . 57 PUSH EDI
01171831 . 88 03 MOV BYTE PTR DS:[EBX],AL
01171833 . E8 58 CALL <videoeditor.sub_116BC90> 1193528:"TotalDays"
01171838 . 68 28 PUSH videoeditor.1193528
0117183D . 57 PUSH EDI
0117183E . 89 43 MOV DWORD PTR DS:[EBX+4],EAX
01171841 . E8 4A CALL <videoeditor.sub_116BC90> 1193534:"IsTrialRestrictionDays"
01171846 . 68 34 PUSH videoeditor.1193534
01171848 . 57 PUSH EDI
0117184C . 89 43 MOV DWORD PTR DS:[EBX+8],EAX
0117184F . E8 BC CALL <videoeditor.sub_116CC10> 119354C:"TrialFirstRun"
01171854 . 68 4C PUSH videoeditor.119354C
01171859 . 57 PUSH EDI
0117185A . 88 43 MOV BYTE PTR DS:[EBX+C],AL
0117185D . E8 AE CALL <videoeditor.sub_116CC10> 119355C:"GetTrialExpirationTimePointUTC"
01171862 . 68 5C PUSH videoeditor.119355C
01171867 . 57 PUSH EDI
01171868 . 88 43 MOV BYTE PTR DS:[EBX+D],AL

```

Nos detuvimos en nuestro <BREAKPOINT> "IsTrial" en la dirección 011717F1 que en realidad viene siendo insignificante porque esta cambia cada vez que reiniciamos la aplicación en el <x64DBG>; y esto se debe a que hace uso de la técnica ASLR, que más que una técnica Antidebugging sirve para proteger la aplicación de vulnerabilidades que sean aprovechadas mediante Exploiting. Sigamos en lo nuestro, en la dirección 01171826 CALL <videoeditor.sub_10CC10>, ese CALL nos retornará si nuestra aplicación es Trial o si es FULL (IsTrial). Luego hace más consultas para saber si tenemos días de prueba y otras cuantas consultas más, que vienen siendo irrelevantes porque en 01171826 CALL <videoeditor.sub_10CC10> se decide todo. Pasemos el 01171826 CALL <videoeditor.sub_10CC10> con <F8> y miremos qué retorna.

PARADOS EN CALL <videoeditor.sub_10CC10>

```

0011181F . C7 43 MOV DWORD PTR DS:[EBX+18],0
00111826 . E8 58 CALL <videoeditor.sub_10CC10> 13351C:"DaysLeft"
00111828 . 68 1C PUSH videoeditor.13351C
00111830 . 57 PUSH EDI
00111831 . 88 03 MOV BYTE PTR DS:[EBX],AL
00111833 . E8 58 CALL <videoeditor.sub_10BC90> 133528:"TotalDays"
00111838 . 68 28 PUSH videoeditor.133528
0011183D . 57 PUSH EDI
0011183E . 89 43 MOV DWORD PTR DS:[EBX+4],EAX
00111841 . E8 4A CALL <videoeditor.sub_10BC90> 133534:"IsTrialRestrictionDays"
00111846 . 68 34 PUSH videoeditor.133534
00111848 . 57 PUSH EDI

```

DESPUES DEL CALL <videoeditor.sub_10CC10>

		Hide FPU
0011181F	C7 43 MOV DWORD PTR DS:[EBX+18],0	
00111826	E8 58 CALL <videoeditor.sub_10CC10>	
00111828	68 1C PUSH videoeditor.13351C	EAX 00000001
00111830	57 PUSH EDI	EBX 003DF494
00111831	88 03 MOV BYTE PTR DS:[EBX],AL	ECX DAF347B2
00111833	E8 58 CALL <videoeditor.sub_10BC90>	EDX 088B0000
00111838	68 28 PUSH videoeditor.133528	EBP 003DF4EC
0011183D	57 PUSH EDI	ESP 003DF468
0011183E	89 43 MOV DWORD PTR DS:[EBX+4],EAX	ESI 003DF480
00111841	E8 4A CALL <videoeditor.sub_10BC90>	EDI 0800A7F8
00111846	68 34 PUSH videoeditor.133534	
00111848	57 PUSH EDI	

Como siempre, todo es cuestión de ir probando y cambiar lo retornado por un valor contrario porque ya sabemos que los valores que obtenemos nos llevarán por mal camino, si es SI entonces nosotros decimos NO, así que como vemos después del CALL <videoeditor.sub_10CC10> tenemos en EAX=01 y que será guardado en 00111828 MOV BYTE PTR

Activador Productos Movavi 2019 [PROGRAMAR NUESTRO PROPIO ACTIVADOR-PATCH][VB.NET]

DS:[EBX],AL para luego decidir más adelante si es **TRIAL** o **FULL**. Cambiemos el valor de **EAX=01** por **EAX=00**.

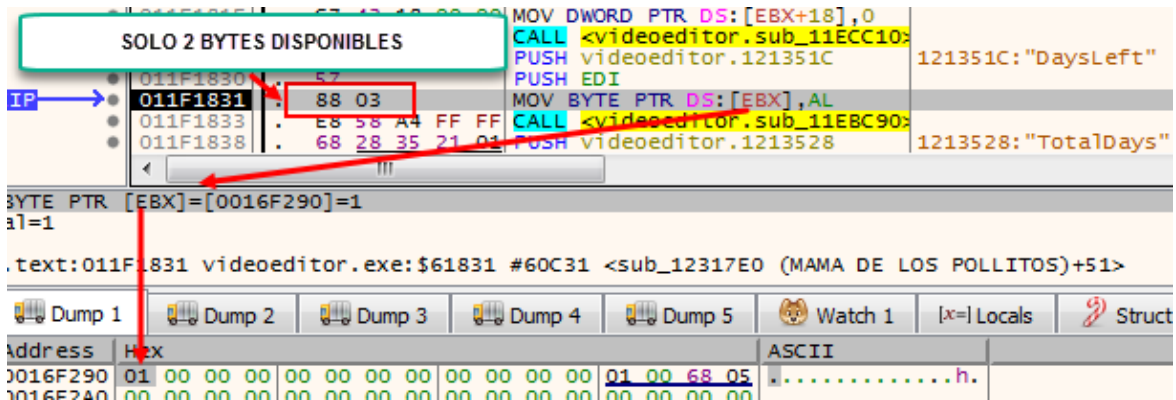
Address	Hex	ASC
003DF494	00 00 00 00 00 00 00 00 00 00 00 00 01 00 FF 07	...
003DF4A4	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
003DF4B4	07 00 00 00 7A B3 CE DA DC F4 3D 00 1E 1A 12 00	...

Cambiamos a **EAX=00** y seguimos hasta guardar el valor en **MEMORIA**. Como es cuestión de probar, ejecutemos todo con **<F9>** para ver qué nos sale.

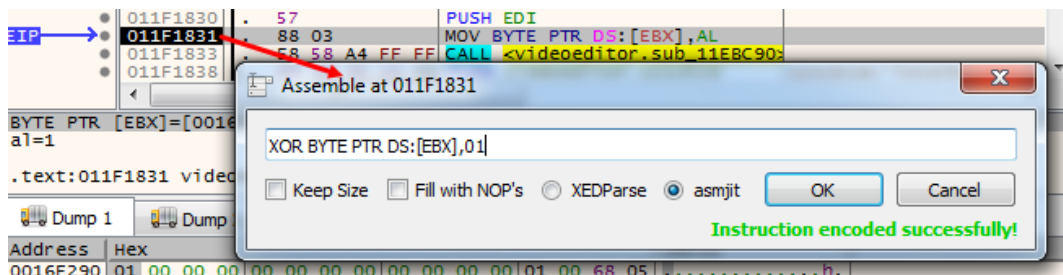


Se ejecutó completamente y adiós **NAG** de prueba caducada. Entonces podemos decir que hemos **Crackeado** el programa en **MEMORIA**, ahora debemos lograr que siempre tengamos **EAX=00**. A buscar un lugar con espacio para hacer nuestro **PATCH**.

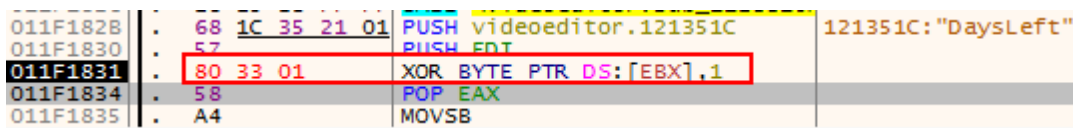
Si miramos, no hay lugar para poder hacer nuestro **PATCH** porque lo cambios que hagamos ocuparán más **BYTES** de los que tenemos. Por ejemplo, **[EBX]** siempre tiene un valor de **0x01**. Reiniciemos todo y lo vemos.



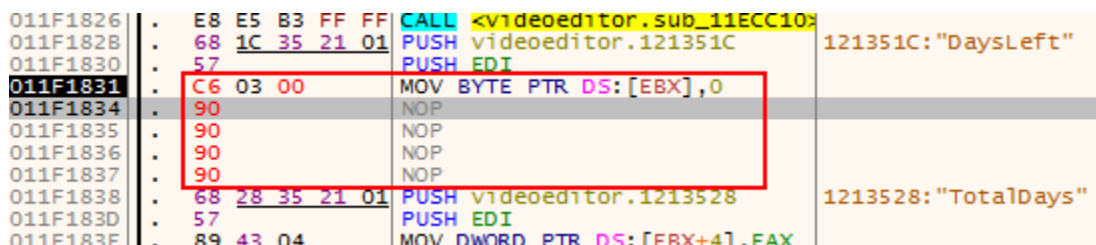
Voy a practicar haciendo los cambios aquí para recordar que diferentes instrucciones requieren más, o menos **BYTES** según la instrucción. Como dijimos **[EBX]=0x01**, entonces podríamos **XOREAR**: **XOR BYTE PTR DS:[EBX],01**.



Haremos nuestro **XOR** y veremos si cabe. Esta instrucción para que funcione se supone debe ocupar 2 **BYTES**. Ensamblemos para ver cómo queda.



Muy mal, me ocupó 3 **BYTES** y con eso me machacó parte del código. No me sirve. Podríamos probar un **MOV BYTE PTR DS:[EBX],0**.



Peor con un **MOV**, este me ocupa 5 **BYTES** de más. Puede que haya otra forma de lograr de aplicar el **PATCH** en esa parte pero este, su humilde servidor solo conoce esas dos. Lo que vamos a hacer es buscar si tenemos espacio dentro del **CALL <videoeditor.sub_10CC10>**.

```

0123CC10 55 PUSH EBP
0123CC11 8B EC MOV EBP,ESP
0123CC13 6A FF PUSH FFFFFFFF
0123CC15 68 C4 PUSH <videoeditor.sub_12510C4>
0123CC1A 64 A1 MOV EAX,DWORD PTR ES:[0]
0123CC20 50 PUSH EAX
0123CC21 81 EC SUB ESP,8C
0123CC27 A1 80 MOV EAX,DWORD PTR DS:[1282080]
0123CC2C 33 C9 XOR EAX,EBP
0123CC2E 89 45 MOV DWORD PTR SS:[EBP-10],EAX
0123CC31 56 PUSH ESI
0123CC32 57 PUSH EDI
0123CC33 50 PUSH EAX
0123CC34 8D 45 LEA EAX,DWORD PTR SS:[EBP-C]
0123CC37 64 A1 MOV EAX,DWORD PTR ES:[0]
0123CC3A 8B EC MOV EBP,ESP
0123CC3C 5D POP EBP
0123CC3D 5C POP ECX
0123CC3E 5B POP EDI
0123CC3F 5A POP ESI
0123CC40 8B EC MOV EBP,ESP
0123CC41 33 CD XOR ECX,EBP
0123CC42 E8 24 CALL videoeditor.12444C7
0123CC43 8B EC MOV EBP,ESP
0123CC44 5D POP EBP
0123CC45 5C POP ECX
0123CC46 C3 RET
0123CC47 CC INT3
0123CC48 CC INT3
0123CC49 CC INT3
0123CC4A CC INT3
0123CC4B CC INT3
0123CC4C CC INT3
0123CC4D CC INT3
0123CC4E CC INT3
0123CC4F CC INT3
    
```

En la imagen de arriba tenemos el `CALL <videoeditor.sub_10CC10>`. Volvamos a recordar que las direcciones no concuerdan por culpa del **ASLR**. Lo importante es tener claro que estamos en el `CALL` indicado. Si bajamos hasta el final del procedimiento podemos ver que después del `RET` tenemos espacio suficiente para nuestro **PATCH**. Aquí podemos hacer el **XOR** o **MOV** que hace rato no pudimos hacer por falta de espacio. Bien amigos, hagámoslo con **XOR**.

```

0123CD9C 33 CD XOR ECX,EBP
0123CD9E E8 24 CALL videoeditor.12444C7
0123CDA3 8B EC MOV EBP,ESP
0123CDA5 5D POP EBP
0123CDA6 33 C0 XOR EAX,EAX
0123CDA7 C3 RET
0123CDA8 CC INT3
0123CDA9 CC INT3
    
```

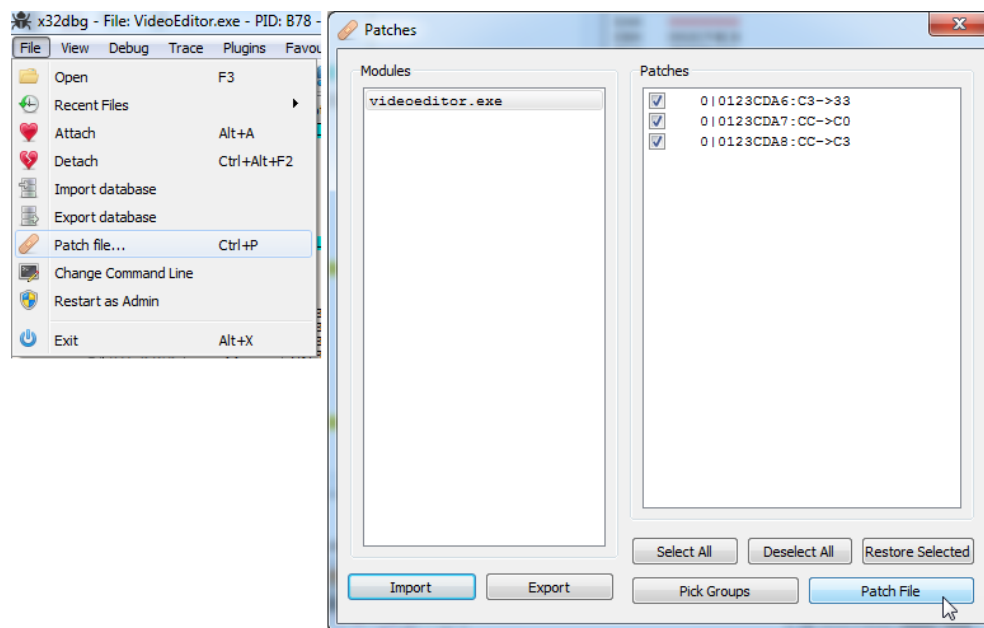
Perfecto, con eso tendremos que **EAX=00**. Tracemos hasta llegar al `RET`.

```

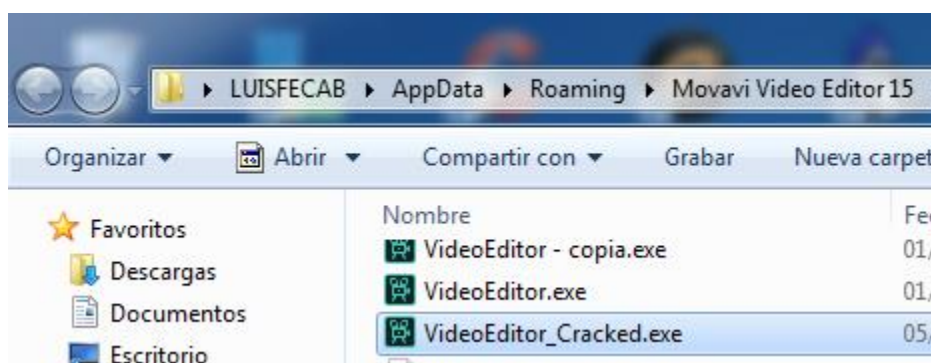
0123CD98 5E POP ESI
0123CD99 8B 4D MOV ECX,DWORD PTR SS:[EBP-10]
0123CD9C 33 CD XOR ECX,EBP
0123CD9E E8 24 CALL videoeditor.12444C7
0123CDA3 8B EC MOV EBP,ESP
0123CDA5 5D POP EBP
0123CDA6 33 C0 XOR EAX,EAX
0123CDA7 C3 RET
0123CDA8 CC INT3
0123CDA9 CC INT3
0123CDAA CC INT3
0123CDAB CC INT3
0123CDAC CC INT3
    
```

Register	Value
EAX	00000000
EBX	002CF8C8
ECX	3D85D421
EDX	068C0000
EBP	002CF920
ESP	002CF898
ESI	002CF8E4
EDI	05EAA7F8

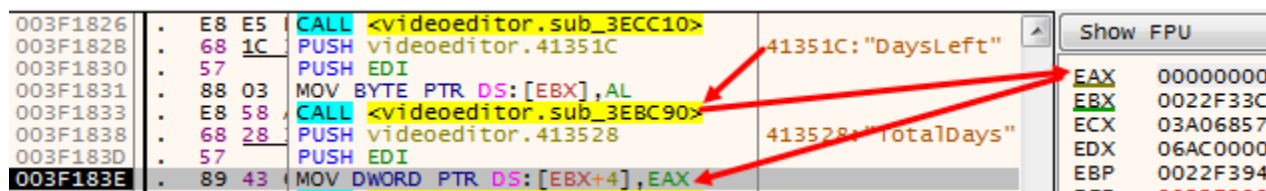
Listo, **EAX=00**. Ya sabemos que funciona. Solo nos queda guardar los cambios, y con eso ya nos hacemos a nuestro **CRACK**.



Lo voy a guardar como **<VideoEditor_Cracked.exe>**.



Bien, ya lo tengo guardado. Lo pruebo y funciona muy bien. No coloco ninguna captura porque ya la colocamos hace rato. Recordemos que tenemos pendiente hacer correr el programa en versión de prueba, solo como para mostrar las diferencias entre **FULL** y **TRIAL**. Para hacerlo solo debemos hacerle pensar que todavía tenemos días de prueba, y eso se hace exactamente igual que hicimos con la comprobación **"IsTrial"**, solo que ahora lo hacemos con la comprobación **"DaysLeft"** o **"TotalDays"**. Al cambiar el valor retornado con **"DaysLeft"** solo podemos obtener máximo 7 días de prueba, pero **"TotalDays"** retorna el número de días que nos quedan como prueba y editando ese valor podemos colocar los días de prueba que queramos, pero claro con esa horrible marca que ponen de nada sirven tener días de prueba. Bueno, hagámoslo correr con días de prueba.



Activador Productos Movavi 2019 [PROGRAMAR NUESTRO PROPIO ACTIVADOR-PATCH][VB.NET]

Hacemos la consulta "DaysLeft" y nos retorna **EAX=0**. Con cambiarlo a **EAX=1** ya tendremos nuestros 7 días de prueba.

003F1826	. E8 E5	CALL <videoeditor.sub_3ECC10>		41351C: "DaysLeft"	↑	Show FPU
003F182B	. 68 1C	PUSH videoeditor.41351C				EAX 00000001
003F1830	. 57	PUSH EDI				EBX 0022F33C
003F1831	. 88 03	MOV BYTE PTR DS:[EBX],AL				ECX 03A06857
003F1833	. E8 58	CALL <videoeditor.sub_3EBC90>		413528: "TotalDays"		EDX 06AC0000
003F1838	. 68 28	PUSH videoeditor.413528				EBP 0022F394
003F183D	. 57	PUSH EDI				ESP 0022F2F8

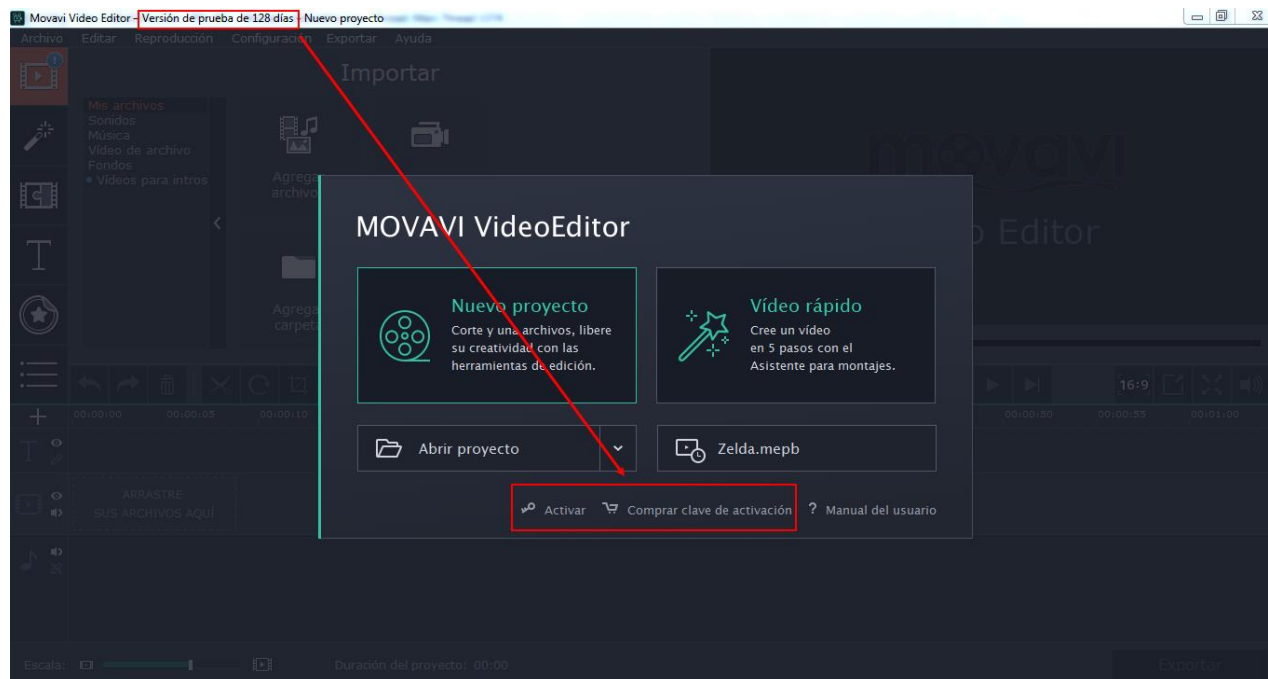
Ya hemos cambiado a **EAX=1**. Solo debemos ver lo que nos retorna "TotalDays".

003F1833	. E8 58	CALL <videoeditor.sub_3EBC90>		413528: "TotalDays"	↑	Show FPU
003F1838	. 68 28	PUSH videoeditor.413528				EAX 00000007
003F183D	. 57	PUSH EDI				EBX 0022F33C
003F183E	. 89 43	MOV DWORD PTR DS:[EBX+4],EAX				ECX 03A06857
003F1841	. E8 4A	CALL <videoeditor.sub_3EBC90>		413534: "IsTrialRes"		EDX 06AC0000
003F1846	. 68 34	PUSH videoeditor.413534				EBP 0022F394
003F184B	. 57	PUSH EDI				ESP 0022F2F8
003F184C	. 89 43	MOV DWORD PTR DS:[EBX+8],EAX				EAX 00000007
003F184F	. E8 BC	CALL <videoeditor.sub_3ECC10>		413540: "TrialFirst"		EBX 0022F33C

"TotalDays" retornó en **EAX=7**, que serían los 7 días de prueba. Si editamos por ejemplo **EAX=80**.

003F1833	. E8 58	CALL <videoeditor.sub_3EBC90>		413528: "TotalDays"	↑	Show FPU
003F1838	. 68 28	PUSH videoeditor.413528				EAX 00000080
003F183D	. 57	PUSH EDI				EBX 0022F33C
003F183E	. 89 43	MOV DWORD PTR DS:[EBX+4],EAX				ECX 03A06857
003F1841	. E8 4A	CALL <videoeditor.sub_3EBC90>		413534: "IsTrialRes"		EDX 06AC0000
003F1846	. 68 34	PUSH videoeditor.413534				EBP 0022F394
003F184B	. 57	PUSH EDI				ESP 0022F2F8
003F184C	. 89 43	MOV DWORD PTR DS:[EBX+8],EAX				EAX 00000080
003F184F	. E8 BC	CALL <videoeditor.sub_3ECC10>		413540: "TrialFirst"		EBX 0022F33C

Serían entonces $0 \times 80 = 128$ días para probar el programa. Hagámoslo correr, <F9>, para que veamos con qué nos sale.



Se carga con 128 días de prueba y nos muestra las opciones para activarlo. Lo sé, lo admito, se preguntarán: ¿si ya ha crackeado el programa; para qué ha colocado este de poder ejecutarlo en modo **TRIAL?**, y mi respuesta es que siempre debemos ir más allá para poder mejorar nuestras habilidades en **Cracking**.

Como dijo **DavicoRm** cuando crackeo las aplicaciones de **MOVAVI**, todas tienen la misma rutina de validación, así que solo buscamos la **String "IsTrial"** para llegar a la **"ZONA CALIENTE"** y luego hacemos el **PATCH** al final de la rutina que valida **"IsTrial"**. No creo que volver a hacer lo mismo con otra aplicación **MOVAVI** sea necesario, pienso que hasta aquí hemos hecho una muy buena explicación de cómo crackear cualquier aplicación, no importa que sea de 32 o 64 Bits.

Ahora seguiremos con programar nuestro propio **PATCH**.

PROGRAMAR NUESTRO PATCH



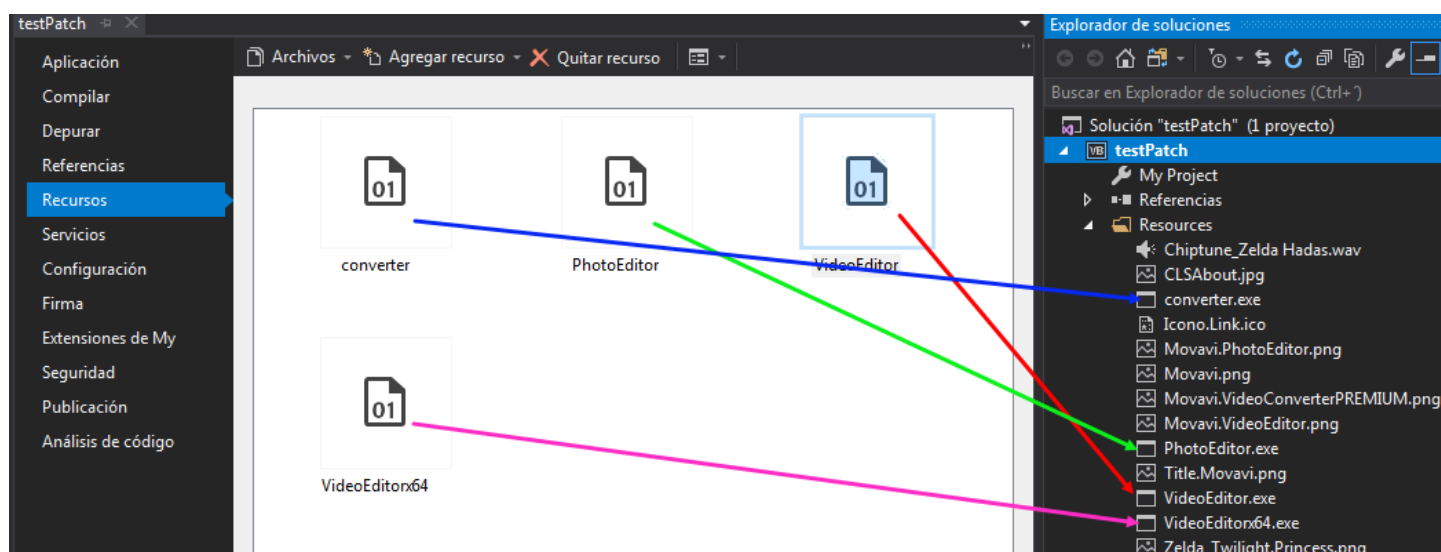
He puesto una captura a todo tamaño, es que estoy muy contento con el resultado, y lo mejor no es eso, si no lo aprendido, que es hacer un **PATCH** en **VB.NET**, y debo

admitir que casi nadie le gusta el **Visual Basic .NET** pero es lo que mejor manejo y para sumar me gusta programar en este.

Aquí entra en juego el <[SpeedConnect Internet Accelerator v8.0 Keygen by URET](#)> que está codeado en **.NET**, de ahí pude hacer esta primera **Versión 2019.01** que es con el **MÉTODO #1**. Lo que haremos ahora será mostrar los tres métodos para hacer un **PATCH**.




MÉTODO #1: PEGAR DESDE RESOURCES.

Este método reemplazara el archivo original por el archivo crackeado el cual está guardado en la carpeta **Resources** del proyecto.



Este método sirve para parchear una única versión de un programa.

Al utilizar este método para nuestro **Activador** terminamos teniendo un archivo demasiado pesado, así que no es ideal porque se deben agregar todos los **Cracks**, lo que aumenta su tamaño y de paso solo sirve para una versión en específico.

Nombre	Fecha de modifica...	Tipo	Tamaño
 testPatch.exe	28/06/2019 05:09 ...	Aplicación	8.971 KB
 testPatch.pdb	28/06/2019 05:09 ...	Base de datos de ...	46 KB
 testPatch.xml	28/06/2019 05:09 ...	Archivo XML	3 KB

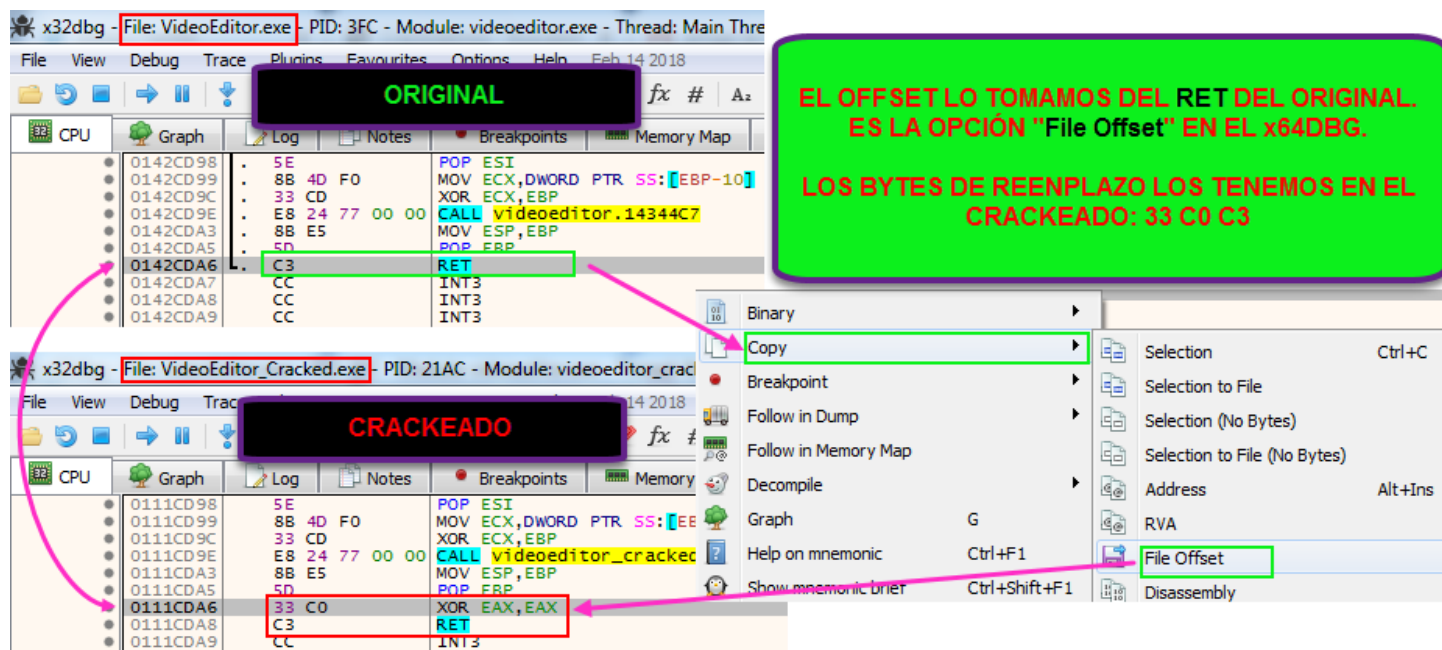
Como vemos el **Activador** pesa casi **9 MB** y eso que solo tiene soporte para cuatro programas, imagínense si le hubiéramos agregados más archivos crackeados al **Resources**. Ahora lo más importante el code en **VB.NET**.

```
If System.IO.File.Exists(loc) Then
    System.IO.File.Move(loc, loc + ".BAK") ' Pasamos el archivo original como un BACKUP
    System.IO.File.WriteAllBytes(loc, resourceBytes) ' Creamos el archivo crackeado que tenemos en Resources
    '
    ' loc=Dirección completa del archivo a parchear
    ' resourceBytes = My.Resources.VideoEditor
End If
```

MÉTODO #2: PACTH CON OFFSET.

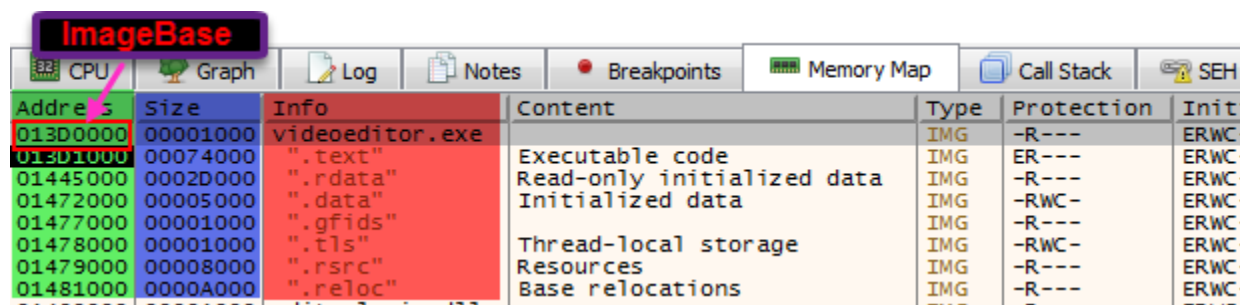
Este método solo reemplaza valores **HEXADECIMALES** en el **Offset** donde están ubicados esos valores a ser reemplazados dentro del **TARGET**. Este método sirve para parchear una única versión de un programa pero tiene la ventaja de que ya al **Activador** no será tan pesado porque nos evitamos guardar el archivo crackeado en el **Resources** del proyecto.

Para este caso debemos conocer el **Offset** de nuestra aplicación. Busquemos el **Offset** para el <VideoEditor.exe>, recordemos que estamos trabajando con la aplicación de 32 Bits.



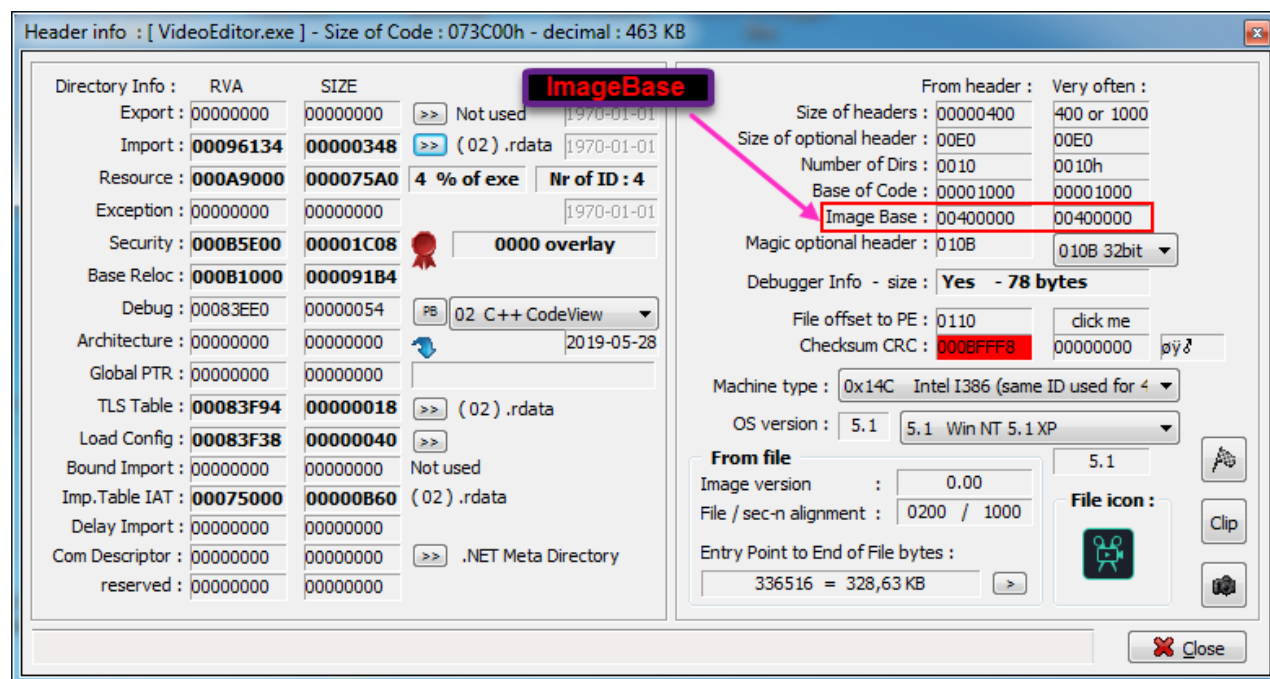
La imagen de arriba es bien clara, si tenemos claro esto de trabajar con direcciones. Copiamos el **Offset** donde hicimos el **PATHC**. Lo copio del archivo **ORIGINAL** que sería en el **RET**, **Offset=0x5C1A6**. Puede sonar como una burrada pero explico cómo entiendo esto del **Offset** en este caso. Ya el <x64DBG> nos hace referencia, "**File Offset**", viene siendo una especie de puntero o dirección que nos lleva a ese lugar dentro del **File** o binario que tenemos en el disco. Y los **BYTES** a reemplazar los tenemos en el archivo **CRACKEADO**, que serían {33 C0 C3}.

En mi caso esto de las direcciones siempre me confunden un poco, pero que este sea el momento de dejar mis explicaciones y de paso me aclaro mis dudas. Hacer estas explicaciones siempre me preocupa porque como saben estoy en proceso de aprendizaje y lo que menos quiero es compartir algo que esté errado, y que en lugar de aportar algo que ayude, termine por aportar algo incorrecto que luego confunda en lugar de ayudar. Vayamos a la pestaña <Memory Map>.



Address	Size	Info	Content	Type	Protection	Init
013D0000	00001000	videoeditor.exe		IMG	-R---	ERWC
013D1000	00074000	".text"	Executable code	IMG	ER---	ERWC
01445000	0002D000	".rdata"	Read-only initialized data	IMG	-R---	ERWC
01472000	00005000	".data"	Initialized data	IMG	-RWC-	ERWC
01477000	00001000	".gids"		IMG	-R---	ERWC
01478000	00001000	".tls"	Thread-local storage	IMG	-RWC-	ERWC
01479000	00008000	".rsrc"	Resources	IMG	-R---	ERWC
01481000	0000A000	".reloc"	Base relocations	IMG	-R---	ERWC

Tenemos la dirección (**Address**) que también es conocida como **VA (Virtual Address)**, el tamaño (**Size**) y las secciones (**Info**). Bueno, eso ya lo conocemos, aquí lo importante es mostrar que esa primer dirección o **VA (Virtual Address)** que tenemos sería la **ImageBase** del archivo cargado en un **Debugger**; debido a que tenemos **ASLR** cada vez que lo carguemos tendremos una **ImageBase** diferente para poder ubicarnos dentro de las direcciones; si no tuviéramos **ASLR** siempre tendríamos las mismas direcciones y por consiguiente la misma **ImageBase**. En realidad la **ImageBase** siempre es la misma solo que todo cambia debido al **ASLR**; si lo miramos en un **PE Editor** donde podemos obtener los datos del **Header** sin ser afectado por el **ASLR**. Voy a utilizar la utilidad que trae el **<Exeinfo PE v0.0.5.6 sign 2019.05.22>**.



Hay muchísima información del **Header** pero para nuestro caso de estudio solo nos interesa la **ImageBase=0x400000**. Casi siempre esa es la **ImageBase** de todos los programas. Si nosotros no tuviéramos **ASLR** esa sería la **ImageBase** que tendríamos dentro del **<x64DBG>**. Ahora, si nos referimos a **RVA (Relative Virtual Address)** que es la **Dirección Virtual Relativa** a la **ImageBase**. Recuerden que nuestra **ImageBase** nos cambia en cada inicio del **<x64DBG>** por el **ASLR**. Resumiendo, **RVA = VA - ImageBase**.

Nos queda entender el cómo hallar un determinado **Offset** que nos da la verdadera ubicación de los **BYTES** que forman un binario en disco. Para hallar el **Offset** necesitamos revisar las secciones un poco más explicadas que como las presenta el **<x64DBG>**. Sigo utilizando las utilidades que trae el **<Exeinfo PE v0.0.5.6 sign 2019.05.22>**.

RVA **PARA HALLAR EL OFFSET**

Nr	Virtual offset	Virtual size	RAW Data offset	RAW size	Flags	Name	First bytes (hex)	First Ascii 20h b...	sect. St
01 ep	00001000	00073B21	00000400	00073C00	60000020	.text	68 D4 63 4A 00 E8 46 D7 00	h c J F h @ G J ...	
02 im	00075000	0002CDB6	00074000	0002CE00	40000040	.rdata	F0 B7 09 00 5A B8 09 00 26	Z & z L ...	
03	000A2000	00004D78	000A0E00	00004400	C0000040	.data	24 5E 47 00 30 5E 47 00 00	\$ ^ G 0 ^ G G ...	
04	000A7000	00000054	000A5200	00000200	40000040	.gids	32 00 00 00 31 00 00 00 0A	2 1 b ! ! ...	
05	000A8000	00000009	000A5400	00000200	C0000040	.tls	00 00 00 00 00 00 00 80 00	...	
06 rs	000A9000	000075A0	000A5600	00007600	40000040	.rsrc	00 00 00 00 00 00 00 00 04	J J L 0 ...	
07	000B1000	000091B4	000ACC00	00009200	42000040	.reloc	00 10 00 00 E4 01 00 00 01	+ 0 0!0...	

Overlay : 08 1C 00 00 00 02 02 00 30 82 1B FC 06 09 2A 86 48 86 F7 0D 01 07 02 A0 82 1B ED 30 82 1B E9 02 | 0 1 0 +- *H 1 + 0 +- 1

End of file : 9A E7 D3 CE B9 69 7B 4A BE D3 0F 67 75 18 E1 CE D5 FE 4C 1E 5C 8C 0D D5 6D 14 E0 EA BE B6 23 C1 | i J %gu i L m l #

Section status : 01 ☒ Executable ☒ Readable ☐ Writable

Section size : 463 KB

All sections size : 727,5 KB

Cave S-Stat Close

-> RAW decimal size : 474112 bytes = 463,00 kb = 0,45 MB <- code Section

En la captura de arriba tenemos las secciones, y nos cae de maravilla para mostrar que la **RVA** también es llamada **Virtual Offset**. Aprovechemos y hagamos la siguiente **PREGUNTA**: Si el inicio de la segunda sección (**Name=.rdata**) tiene una **Virtual Offset** o **RVA** igual a **0x75000**, ¿cuál sería la **VA** de la segunda sección en nuestro **<x64DBG>**? **RESPUESTA**: Esto será válido para una sola ejecución, ya que la **ImageBase** cambia debido al **ASLR**. Miremos nuestra **ImageBase**.

CPU ImageBase = 0x01170000 Breakpoints Memory Map

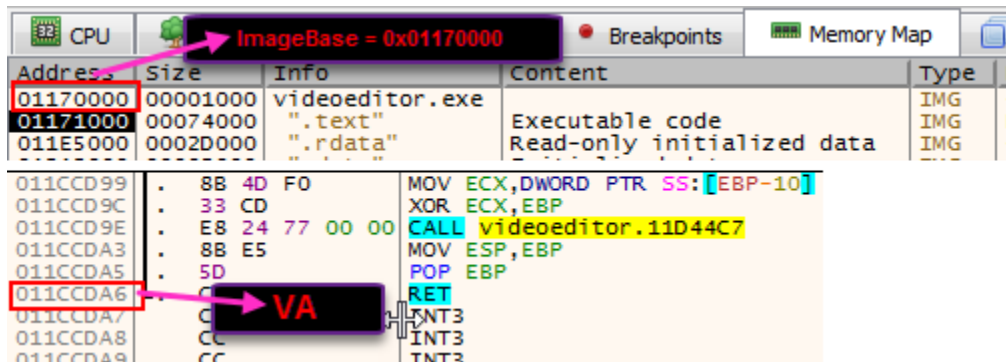
Address	Size	Info	Content	Type
01170000	00001000	videoeditor.exe	Executable code	IMG
01171000	00074000	".text"	Read-only initialized data	IMG
011E5000	0002D000	".rdata"	Read-only initialized data	IMG

ImageBase = 0x01170000, **RVA = 0x75000**. Sabemos que **RVA = VA - ImageBase**, entonces despejamos **VA = RVA + ImageBase**, quedándonos **VA = 0x75000 + 0x01170000**, **VA = 0x11E5000**.

CPU Graph Log Notes Breakpoints Memory Map

Address	Size	Info	Content	Type
01170000	00001000	videoeditor.exe	Executable code	IMG
01171000	00074000	".text"	Read-only initialized data	IMG
011E5000	0002D000	".rdata"	Read-only initialized data	IMG
01212000	00005000	".data"	Initialized data	IMG

Como vemos, el valor concuerda con el que muestra el <x64DBG> en el "Memory Map". Ahora sigamos con el **Offset**. Aquí debemos tomar los datos que pertenecen a la sección donde está la **VA** a la que le queremos hallar el **Offset**.



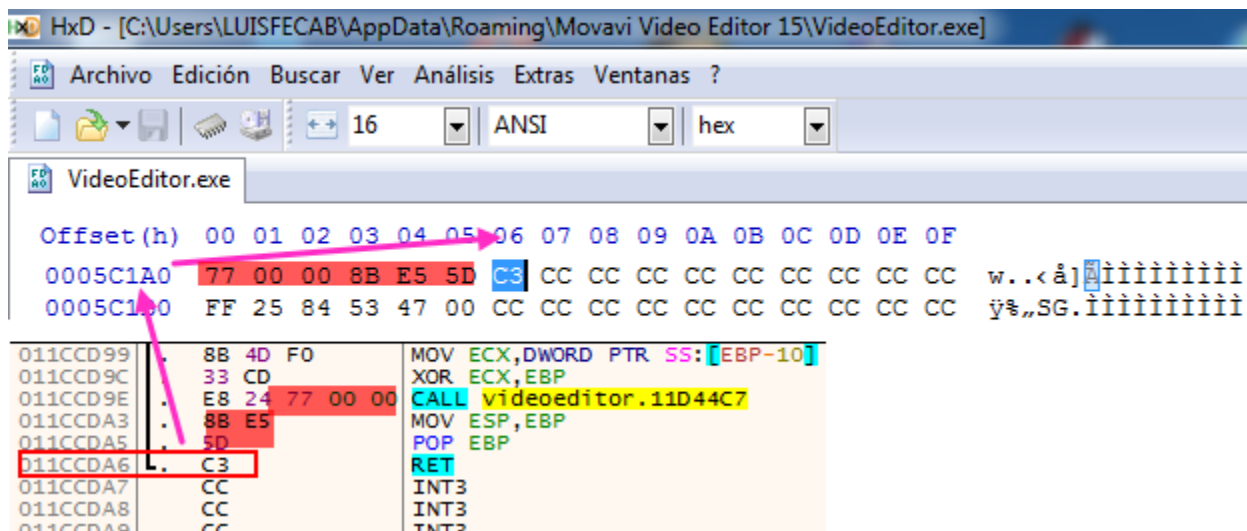
La **VA** = **0x011CCDA6**, **ImageBase** = **0x01170000**. De la captura de la secciones obtenemos la **RVA** del inicio de la sección, **RVA.seccion.text** = **0x1000** y el **RAWDataOffset.seccion.text** = **0x400**. Todo se reduce a la fórmula:

$$\text{Offset} = \text{VA} - \text{ImageBase} - \text{RAWDataOffset.seccion.text} + \text{RVA.seccion.text}$$

$$\text{Offset} = 0x011CCDA6 - 0x01170000 - 0x1000 + 0x400$$

$$\text{Offset} = 0x5C1A6$$

Si cargamos el <VideoEditor.exe> **ORIGINAL** en un Editor Hexadecimal. Voy a utilizar el <HxD v1.7.7.0> y busco ese **Offset** = **0x5C1A6**.



Como vemos son los mismos **BYTES** en ambos el <HxD v1.7.7.0> y en el <x64DBG>, y es a partir de ese **Offset** que se colocaran los 3 **BYTES**: {33 0xC0 0xC3}.

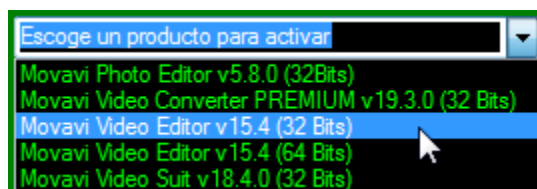
Después de tanta explicación, haremos el **PACTH** en **VB.NET**. Ya tenemos el **Offset** y los **BYTES**. Vamos a explicar un poco más el código que en el **METODO #1**.

```
Public Class PatchMovavi

    Dim ex, ey As Integer
    Dim Arrastre As Boolean

    Private TargetFile As String      'Ruta completa del archivo
    Private oTargetFile As String()   'Para agregar los nombres de los archivos a parchear. Pueden ser más de uno a parchear.
    Private resourceBytes As Byte()   'BYTES que se reemplazaran en TargetFile
    Private _Offset As Integer()      'Ubicación donde reemplazaremos resoucesBytes del TargetFile. Puerder ser más de un Offset.
    Private iOffset As Integer = 0    'Valor Offset dentro del Array _Offset
    Private noPatch As Boolean         'Variable para deterner la ejecución si este falla.
```

Tenemos estas variables **Private** que nos servirán para hacer el **PATCH**. En esta versión mejoramos el **PATCH**, ahora se parchean varios archivos al mismo tiempo dependiendo de la aplicación instalada. Para aplicar el **PATCH** debemos escoger primero el producto a activar (Parchear) en el ComboBox.



Cuando hacemos esto se ejecuta el código donde cargaremos las variables **Private** con la información para Activar el producto que en este caso será el **<Movavi Video Editor v15.4 (32 Bits)>**.

```
Private Sub cmbBox_SelectedIndexChanged(sender As Object, e As EventArgs) Handles cmbBox.SelectedIndexChanged
    txtInfo.Text += String.Concat("Producto: ", cmbBox.SelectedItem, vbCrLf & vbCrLf)
    txtInfo.SelectionStart = txtInfo.TextLength
    txtInfo.ScrollToCaret()

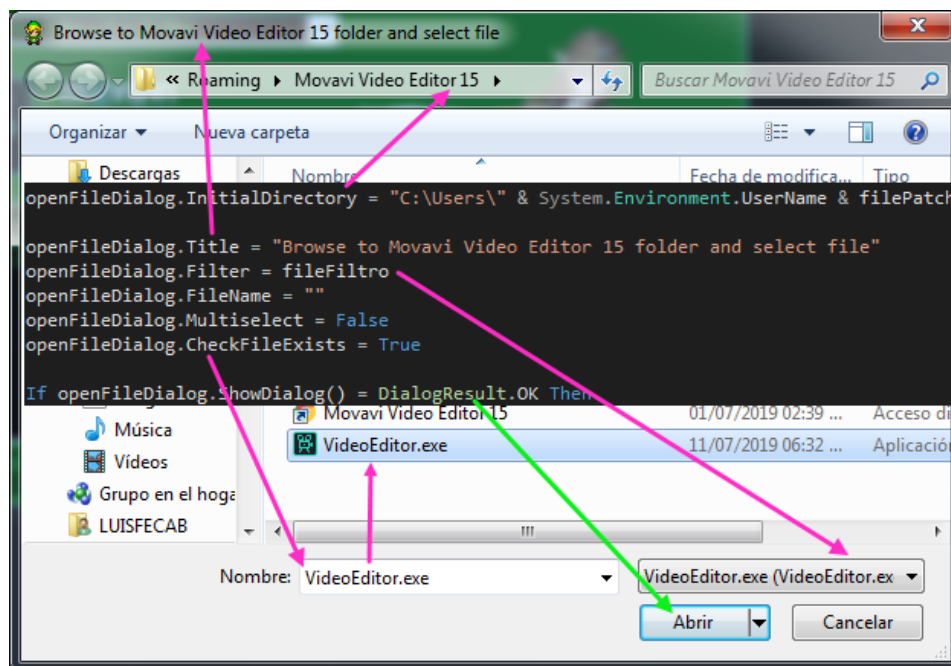
    Select Case cmbBox.SelectedIndex
        Case 0 'x32 bits
```

Con el **Select Case cmbBox.SelectedIndex** determinamos el producto seleccionado. En la captura de arriba solo mostramos el inicio porque mostrar toda la captura ocuparía mucho espacio. Ahora miremos la parte del código del **Select Case cmbBox.SelectedIndex** que se ejecuta al haber nosotros seleccionado el **<Movavi Video Editor v15.4 (32 Bits)>**.

```
        btnPatch.Enabled = True
    Case 2 'x32 bits
        picProducts.Image = My.Resources.Movavi_VideoEditor
        filePatch = "\AppData\Roaming\Movavi Video Editor 15"
        oTargetFile = {"VideoEditor.exe", "VideoCapture.exe"}
        fileFiltro = oTargetFile(0).ToString & "|" & oTargetFile(0).ToString
        _Offset = {&H5C1A6, &HC51F6}
        resourceBytes = {&H33, &HC0, &HC3}
        btnPatch.Enabled = True
```

La variable **oTergatFile** es un **Array** que tendrá los nombres de los archivos a parchear que en este caso son dos. La variable **_Offset** es otro **Array** que tiene los **Offset** de cada archivo que los tenemos en **oTargetFile**. Y por último tenemos el **Arrray resourceBytes** que son los **BYTES** que reemplazarán a los originales. Después

de escoger nuestro producto se activará nuestro botón **"PATCH"**. Si aplicamos el **PATCH** se ejecutará el siguiente código.



Escogemos el archivo a parchear. Miremos ahora el código un poco más completo.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles btnPatch.Click

    openFileDialog.InitialDirectory = "C:\Users\" & System.Environment.UserName & filePatch

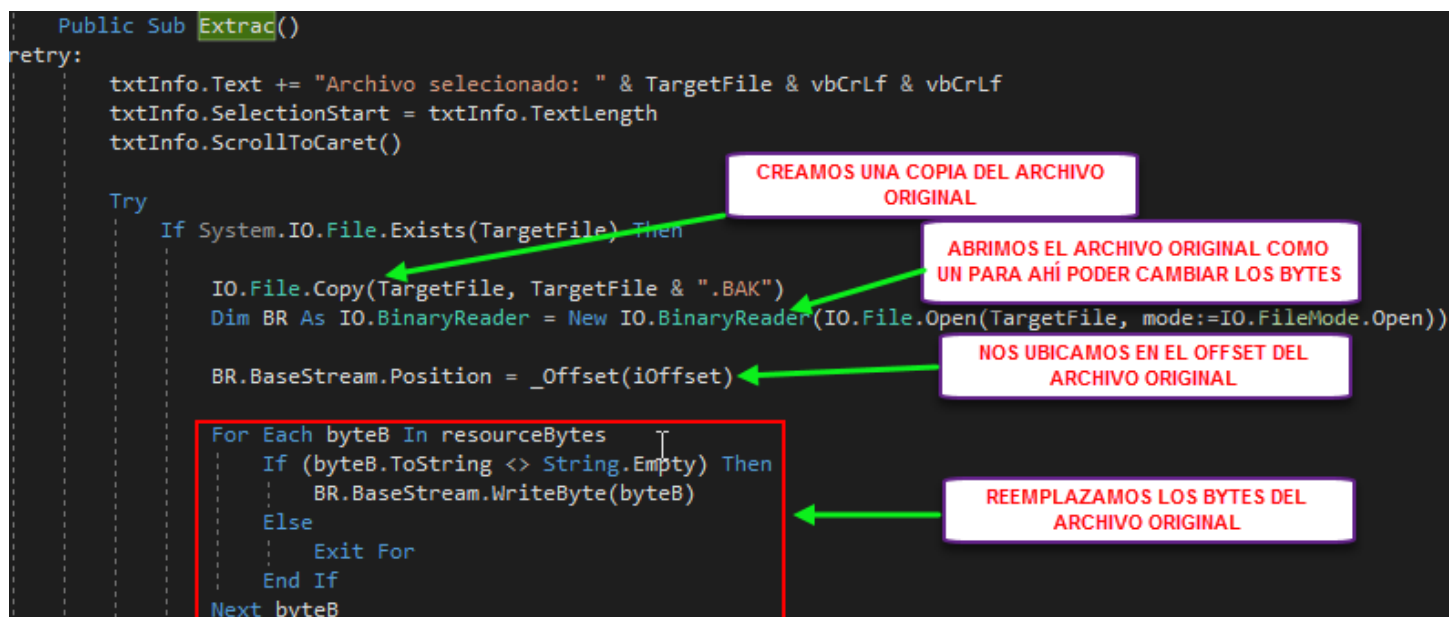
    openFileDialog.Title = "Browse to Movavi Video Editor 15 folder and select file"
    openFileDialog.Filter = fileFiltro
    openFileDialog.FileName = ""
    openFileDialog.Multiselect = False
    openFileDialog.CheckFileExists = True

    If openFileDialog.ShowDialog() = DialogResult.OK Then
        btnPatch.Enabled = False
        Dim directoryName As String = System.IO.Path.GetDirectoryName(openFileDialog.FileName)

        Debug.Print(_Offset.Length.ToString)
        For iOffset = 0 To _Offset.Length - 1
            If noPatch Then Exit Sub
            TargetFile = directoryName & "\" & oTargetFile(iOffset)
            Extrac()
        Next
    End If
End Sub
```

El **RECUADRO ROJO** no abrirá un **openFileDialog** que nos mostrará la carpeta de instalación donde tenemos el programa a parchear. Si escogemos al archivo tomamos el **If openFileDialog.ShowDialog() = DialogResult.OK Then**, y haremos el **For iOffset = 0 To _Offset.Length - 1**, que se repetirá por cada archivo a Parchear; en este caso son dos archivos. También hubiera podido hacer el **For iOffset = 0 To _Offset.Length - 1** en función

de la longitud del **Array** **oTargetFile**, quedando **For** **iOffset = 0 To oTargetFile.Length - 1**, ya que esas variables siempre tendrán la misma cantidad de elementos en el **Array**. Dentro del **For** tenemos **TargetFile = directoryName & "\" & oTargetFile(iOffset)**, que es la ruta hacia el archivo a Parchear. Luego tenemos el procedimiento **Extrac()**, que es el que hará el **Parcheo**.



Podemos ver qué hace esas líneas de código que son las que hacen el **Parcheo**. He explicado casi todo el código porque para utilizar el **METODO #3** se agregarán una función y un poco más en el código del procedimiento **Public Sub Extrac()**, pero que con lo que explicamos se le puede hacer seguimiento y entenderlo.

Nombre	Fecha de modificación	Tipo	Tamaño
TempPE	06/07/2019 08:52 a.m.	Carpeta de archivos	
testPatch.exe	12/07/2019 02:14 p.m.	Aplicación	830 KB
DesignTimeReso...	11/07/2019 03:30 p.m.	Archivo CACHE	2 KB
...

Podemos ver que este **PATCH** es muchísimo más liviano, menos de 1 MB, es más, sería más liviano si no le hubiera puesto la música e imágenes.

MÉTODO #3: PACTH BUSCAR Y REEMPLAZAR BYTES.

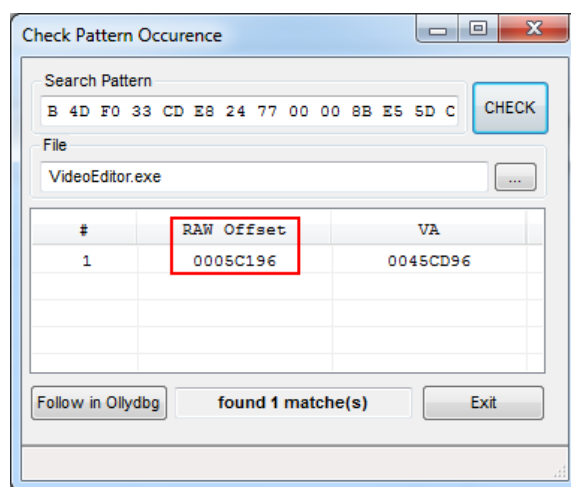
Este es método es el más completo porque **permite parchear todas versiones de un programa que tengan la misma validación**, como ocurre con los productos **MOVAVI**. Aquí debemos asegurarnos de que nuestro **PATRÓN DE BYTES** buscado no se repita porque si encuentra el **PATRÓN DE BYTES** y no era el que debíamos parchear, entonces lo más seguro es que terminamos con un programa corrupto que no se ejecuta. Aclaremos que en nuestro caso el **Parcheo** se hace en una solo **Dirección** u **Offset** pero puede que en futuros **PATCH** que programemos se hagan en otros lugares. Ahora debemos buscar

un **PATRÓN DE BYTES** que no se repita, y ya con eso hacemos la **Versión 2019.03** de nuestro **PATCH**.

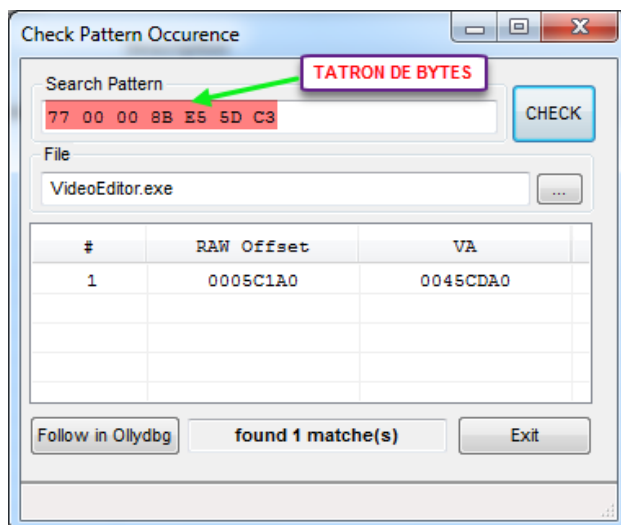
PATRÓN.BYTES
 59 5F 5E 8B 4D F0 33 CD E8 24 77 00 00 8B E5 5D C3

009DCD96 009DCD97 009DCD98 009DCD99 009DCD9C 009DCD9E 009DCDA3 009DCDA5 009DCDA6	. 59 . 5F . 5E . 8B 4D F0 . 33 CD . E8 24 77 00 00 . 8B E5 . 5D . C3	POP ECX POP EDI POP ESI MOV ECX,DWORD PTR SS:[EBP-10] XOR ECX,EBP CALL videoregister.9E44C7 MOV ESP,EBP POP EBP RET
--	--	---

Iniciemos la búsqueda con los siguientes **BYTES**: {59 5F 5E 8B 4D F0 33 CD E8 24 77 00 00 8B E5 5D C3} y a partir de ahí ir reduciendo el **PATRÓN DE BYTES** hasta que sigamos teniendo una sola concordancia. Y qué mejor manera de hacer esto que utilizar el <dup2 Diablo's Universal Patcher v2.26.1>.



Tenemos una ocurrencia. Esos **BYTES** perfectamente podríamos utilizarlo como nuestro **PATRÓN DE BYTES**, pero sería mejor reducirlo hasta su menor tamaño. Entonces empezaré quitando el primer **BYTE** y chequeando hasta que tenga una sola concordancia.



Nos queda así; **PATRÓN DE BYTES**: {77 00 00 8B E5 5D C3}. Como estoy escribiendo el tutorial sobre la marcha, se me pasó algo importante con respecto al **PATRÓN DE BYTES**. Miremos la siguiente captura.

009DCD9E	. E8 24 77 00 00	CALL videoeditor.9E44C7	009DCD9E	. E8 24 77 00 00	CALL videoeditor.9E44C7
009DCDA3	. 8B E5	MOV ESP,EBP	009DCDA3	. 8B E5	MOV ESP,EBP
009DCDA5	. 5D	POP EBP	009DCDA5	. 5D	POP EBP
009DCDA6	. C3	RET	009DCDA6	. 33 C0	XOR EAX,EAX
			009DCDA8	. C3	RET

Como vemos nuestro **PATCH** ocupa espacio nuevo, dos **BYTES** más para ser exactos, y en nuestra búsqueda del **PATRÓN DE BYTES** se me olvidó meter los **BYTE** en la búsqueda, esos **BYTE** son el **CC**. Mirémoslo.

009DCD9E	. E8 24 77 00 00	CALL videoeditor.9E44C7
009DCDA3	. 8B E5	MOV ESP,EBP
009DCDA5	. 5D	POP EBP
009DCDA6	. C3	RET
009DCDA7	. CC	INT3
009DCDA8	. CC	INT3
009DCDA9	. CC	INT3
009DCDAA	. CC	INT3

Simplemente debemos agregarlo a nuestro **PATRÓN DE BYTES**: {77 00 00 8B E5 5D C3 CC CC}. Ya tenemos todo lo necesario, ahora hagamos nuestro **BYTES DE REEMPLAZO**: {77 00 00 8B E5 5D 33 C0 C3}. Como tenemos dos conjuntos de **BYTES** el de la búsqueda y reemplazo, opté por agregar dos colecciones en donde colocaba como **LLAVE** (primer elemento de la colección) el programa y como **VALOR** (Segundo elemento de la colección) los **BYTES**.

```
Private coleccionFindHex As New System.Collections.Generic.SortedList(Of String, Byte())
Private coleccionReplaceHex As New System.Collections.Generic.SortedList(Of String, Byte())
```

Una colección para como **PATRÓN DE BYTES** y la otra como **BYTES DE REEMPLAZO**.

```
coleccionFindHex.Add("VideoEditor.exe(32)", {&H77, &H0, &H0, &H8B, &HE5, &H5D, &HC3, &HCC, &HCC})
coleccionFindHex.Add("VideoEditor.exe(64)", {&HE8, &HF7, &H88, &H0, &H0, &H48, &H8B, &H9C, &H24, &H60, &H1, &H1})
coleccionFindHex.Add("VideoCapture.exe(32)", {&H8C, &H0, &H0, &H8B, &HE5, &H5D, &HC3, &HCC, &HCC})
coleccionFindHex.Add("VideoCapture.exe(64)", {&HE8, &H87, &HA2, &H0, &H0, &H48, &H8B, &H9C, &H24, &H60, &H1, &H1})
coleccionFindHex.Add("PhotoEditor.exe(32)", {&H5E, &H0, &H0, &H8B, &HE5, &H5D, &HC3, &HCC, &HCC})
coleccionFindHex.Add("PhotoEditor.exe(64)", {&HE8, &H87, &H6B, &H0, &H0, &H48, &H8B, &H9C, &H24, &H60, &H1, &H1})
coleccionFindHex.Add("converter.exe(32)", {&H7A, &H0, &H0, &H8B, &HE5, &H5D, &HC3, &HCC, &HCC})
coleccionFindHex.Add("converter.exe(64)", {&HE8, &H37, &H8E, &H0, &H0, &H48, &H8B, &H9C, &H24, &H60, &H1, &H1})
```

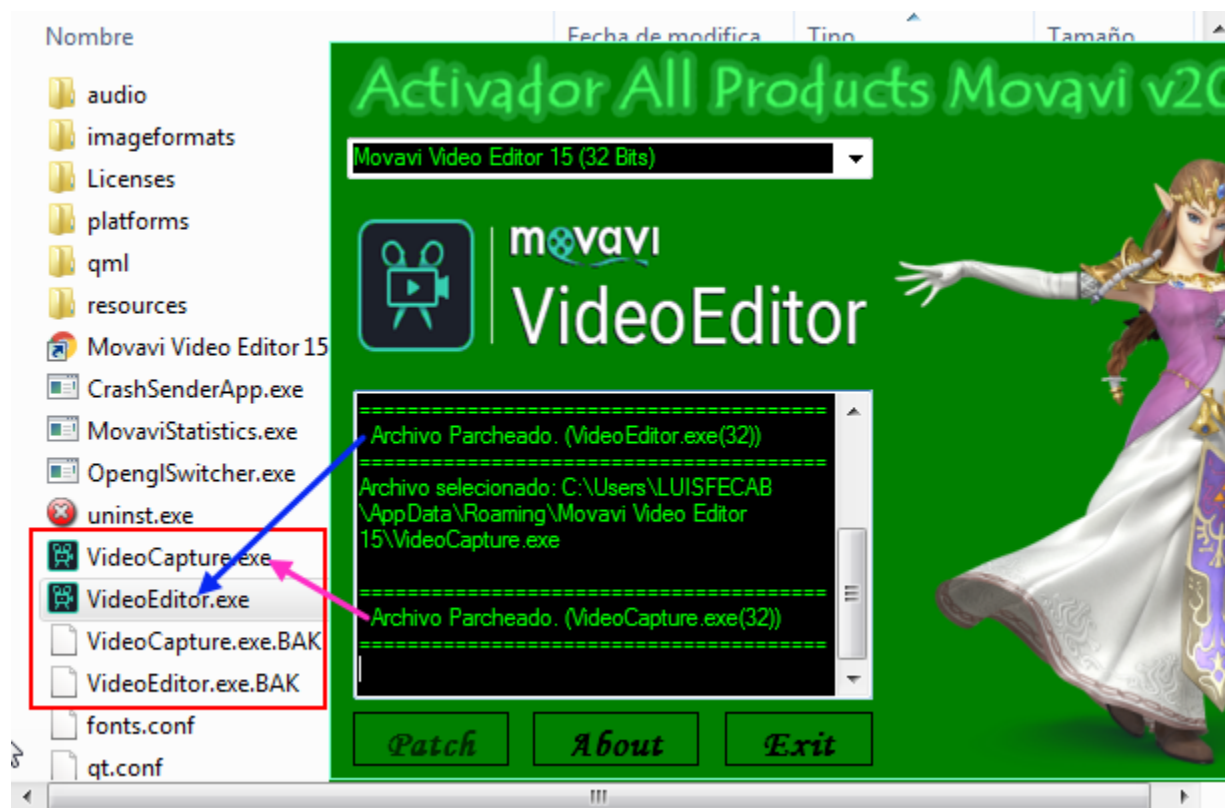
Resulta que cada programa tenía un **PATRÓN DE BYTES** diferentes, y se me hizo fácil agregarlo para cada programa, uno para 32 Bits y otro para 64 Bits. Luego debía hacer lo mismo pero para su contraparte en el reemplazo.

```

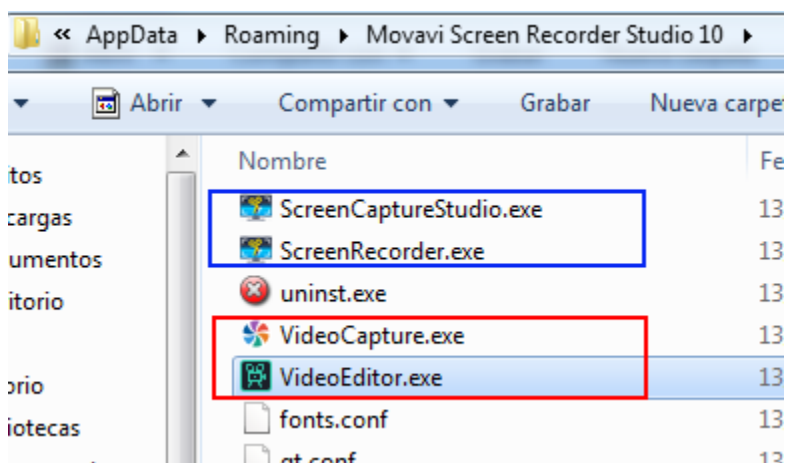
coleccionReplaceHex.Add("VideoEditor.exe(32)", {&H77, &H0, &H0, &H8B, &HE5, &H5D, &H33, &HC0, &HC3})
coleccionReplaceHex.Add("VideoEditor.exe(64)", {&HE8, &HF7, &H88, &H0, &H0, &H48, &H8B, &H9C, &H24, &H60,
coleccionReplaceHex.Add("VideoCapture.exe(32)", {&H8C, &H0, &H0, &H8B, &HE5, &H5D, &H33, &HC0, &HC3})
coleccionReplaceHex.Add("VideoCapture.exe(64)", {&HE8, &H87, &HA2, &H0, &H0, &H48, &H8B, &H9C, &H24, &H60,
coleccionReplaceHex.Add("ScreenCapture.exe(32)", {&H6C, &H0, &H0, &H8B, &HE5, &H5D, &H33, &HC0, &HC3})
coleccionReplaceHex.Add("PhotoEditor.exe(32)", {&H5E, &H0, &H0, &H8B, &HE5, &H5D, &H33, &HC0, &HC3})
coleccionReplaceHex.Add("PhotoEditor.exe(64)", {&HE8, &H87, &H6B, &H0, &H0, &H48, &H8B, &H9C, &H24, &H60,
coleccionReplaceHex.Add("converter.exe(32)", {&H7A, &H0, &H0, &H8B, &HE5, &H5D, &H33, &HC0, &HC3})
coleccionReplaceHex.Add("converter.exe(64)", {&HE8, &H37, &H8E, &H0, &H0, &H48, &H8B, &H9C, &H24, &H60, &

```

Y estos son los **BYTES DE REEMPLAZO**. En las primeras prueba todo funcionó muy bien. Mi lógica era que un mismo programa y sus futuras versiones deben tener ese mismo **PATRÓN DE BYTES** cuando haga su validación, claro, siempre y cuando sigan utilizando en mismo método. Resulta que, al probarlo en el programa sobre el que estamos basándonos para escribir este tutorial, **<Movavi Video Editor 15 (32 Bits)>**, funcionó muy bien.

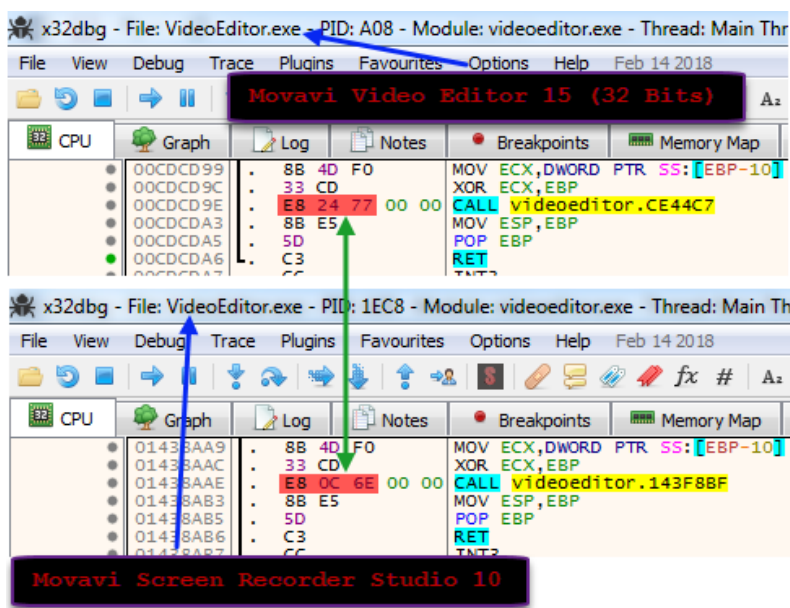


Muy contento, todo parecía ir tan bien. Pero las cosas no son como parecen, cuando lo probé con otro producto el **<Movavi Screen Recorder Studio 10>**, que es un producto más completo en donde viene con esos dos programas que acabo de Activar.



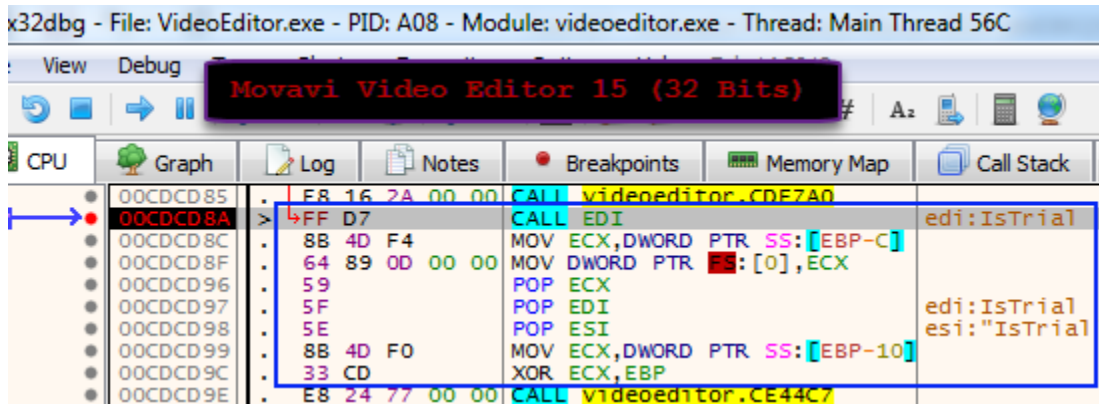
Aquí sería **Parchear** 4 programas pero al hacerlo el **<VideoCapture.exe>** y el **<VideoEditor.exe>** no se activaban, me salían **TRIAL**. Y aquí me preguntada, en dónde fallo si hace ratito los acabo de activar en el producto **<Movavi Video Editor 15 (32 Bits)>**. Bien, pues les cuento que estaba destinado a fallar porque venía mal desde el **PATRÓN DE BYTES: {77 00 00 8B E5 5D C3 CC CC}**.

Por eso, me encanta escribir estos tutoriales porque puedo compartir mis errores con todos ustedes, que de seguro es dónde más aprendemos. Cada vez que escribo un tutorial alguna enseñanza me deja y voy sacando mis propias ideas que mientras me funcionen las tomaré como ciertas, y aquí ocurrió exactamente eso. Tenía la idea que ese **PATRÓN DE BYTES** siempre sería el mismo para versiones o compilaciones del mismo programa, y lo sigo creyendo así, pero creo que eso se cumple si nos referimos a instrucciones que realizan cálculos. La idea es tomar esos **BYTES** como referencia, y pienso que mi error en mi **PATRÓN DE BYTES: {77 00 00 8B E5 5D C3 CC CC}**, fue que tomé **BYTES** que hacen parte de un **CALL** directo otro procedimiento desde una dirección, y creo que esos si pueden cambiar debido a alguna compilación nueva. Miremos los dos programas, que se suponen debe tener los mismos **BYTES**, pero no.

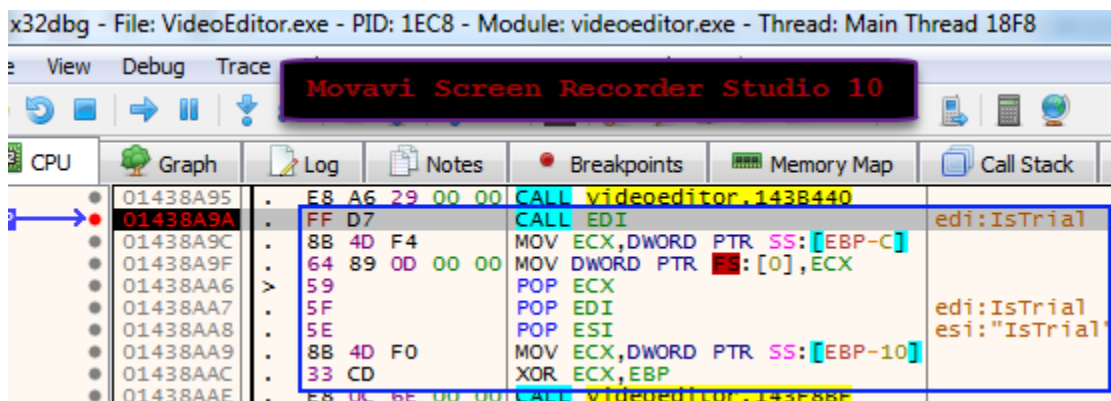


Activador Productos Movavi 2019 [PROGRAMAR NUESTRO PROPIO ACTIVADOR-PATCH][VB.NET]

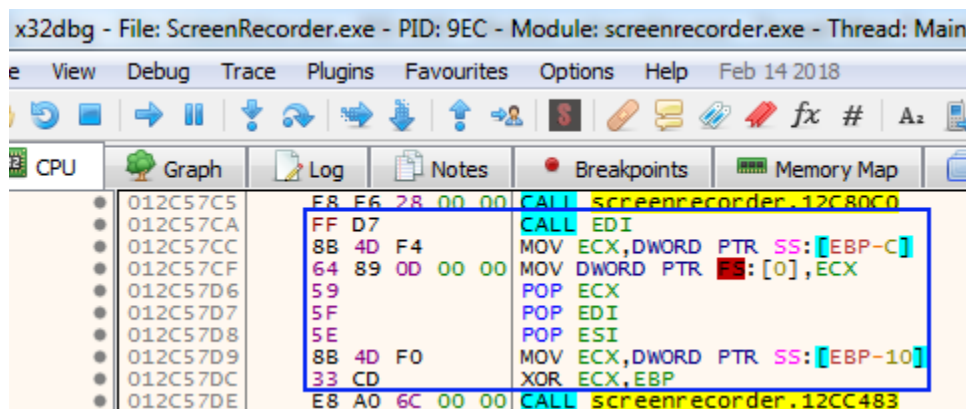
Como vemos todos los **BYTES** son iguales menos los que hacen referencia al **CALL**. Debido a eso el Activador no me funciona porque mi **PATRÓN DE BYTES** ya no sirve para todos. Después de analizar esto, pienso que debo buscar otro **PATRÓN DE BYTES** que no tenga presente un **CALL** directo.



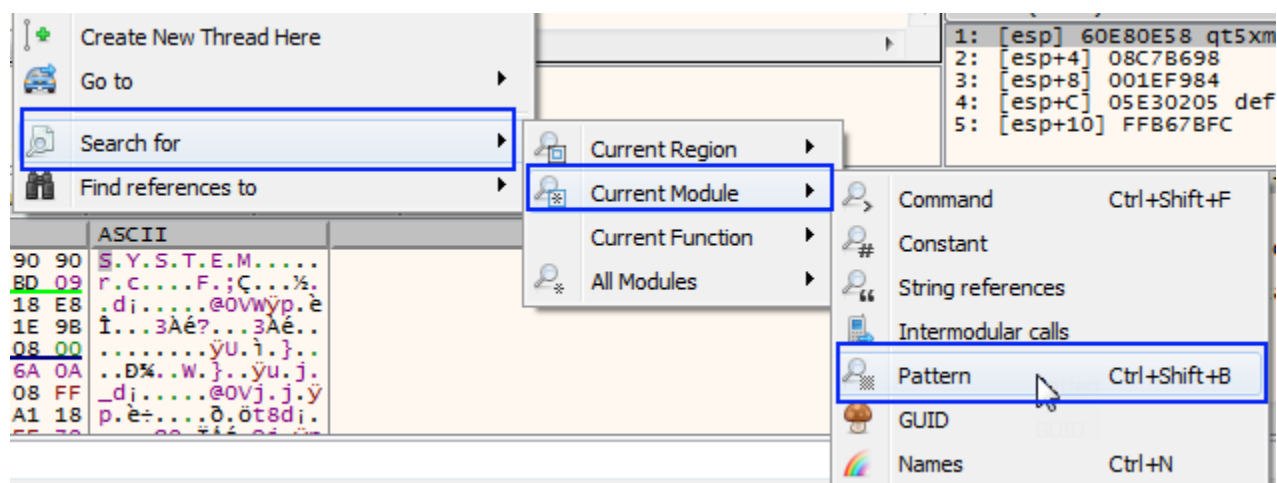
La captura de arriba es dónde voy a sacar mi nuevo **PATRÓN DE BYTES**. Notemos que voy a tomar un **CALL EDI** y que es donde se valida **IsTrial**, que si recordamos se hace en la **<GeneralPlugin.dll>** que viene con **Themida/WinLicence**. Ahora, miremos eso mismo en el otro Producto.



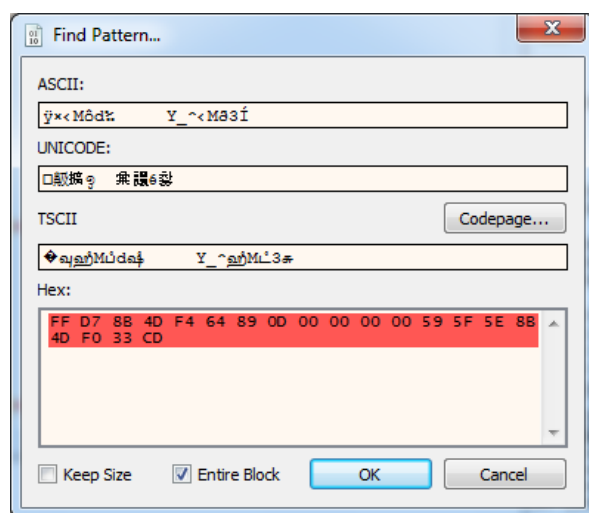
Como podemos ver, ahí si tenemos los mismos **BYTES**. Aclaremos esto del porqué tomé el **CALL EDI**, si ya había dicho que nada de tomar alguno. Pues me refería a un **CALL** directo, y si vemos este **CALL** está dado por un registro, **CALL EDI**, y además no cambia en ninguno de los dos, así que tomo eso como válido para hacer mi nuevo **PATRÓN DE BYTES**: **{FF D7 8B 4D F4 64 89 0D 00 00 00 59 5F 5E 8B 4D F0 33 CD}**. Es un **PATRÓN DE BYTES** más largo pero lo que importa es que es igual para cualquier compilado de **<VideoEditor.exe>** y por ahora puedo suponer que para futuras también. Es más, si miramos en otro programa como el **<ScreenRecorder.exe>**.



Son los mismos **BYTES**. Esto es mucho mejor porque con lo que hemos visto hasta aquí podemos pensar que con un solo **PATRÓN DE BYTES** será suficiente para todos. Ahora, el cambio lo haremos en el **CALL EDI (FF D7)**, que son dos **BYTES**, y si recordamos nuestro **XOR EAX,EAX(33 C0)** ocupa dos **BYTES**, lo cual es perfecto. Solo nos queda buscar si este **PATRÓN DE BYTES** se nos repite, y lo haremos en el **<x64DBG>**.



Buscamos nuestro **PATRÓN DE BYTES: {FF D7 8B 4D F4 64 89 0D 00 00 00 00 59 5F 5E 8B 4D F0 33 CD}**.



Veremos, si después de tanta vuelta esto no sirve.

Pattern: FFD78B4DF464890D...	
Address	Disassembly
01437B1A	call edi
01437E7A	call edi
0143801A	call edi
01438A9A	call edi

Se repite cuatro veces, y con la lógica con que venimos trabajando en este tutorial no nos serviría; pero aquí es donde aplicamos lo que de vez en cuando decimos, que siempre hay que ir más allá. Puse un <BREAKPOINT> en esos cuatro **CALL** y ejecuté el programa para ver qué pasaba allí.

- File: VideoEditor.exe - PID: 1EC8 - Module: videoeditor.exe - Thread: Main Thread 18F8			
Debug Trace Plugins Favourites Options Help Feb 14 2018			
Graph Log Notes Breakpoints Memory Map Call Stack			
01437B1A	FF D7	CALL EDI	edi:DaysLeft
01437B1C	8B 4D F4	MOV ECX,DWORD PTR SS:[EBP-4]	
01437B1F	64 89 0D 00 00	MOV DWORD PTR ES:[0],ECX	
01437B26	59	POP ECX	
01437B27	5F	POP EDI	edi:DaysLeft
01437B28	5E	POP ESI	esi:"DaysLeft"
01437B29	8B 4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
01437B2C	33 CD	XOR ECX,EBP	
01437B1A	FF D7	CALL EDI	edi:TotalDays
01437B1C	8B 4D F4	MOV ECX,DWORD PTR SS:[EBP-4]	
01437B1F	64 89 0D 00 00	MOV DWORD PTR ES:[0],ECX	
01437B26	59	POP ECX	
01437B27	5F	POP EDI	edi:TotalDays
01437B28	5E	POP ESI	esi:"TotalDays"
01437B29	8B 4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
01437B2C	33 CD	XOR ECX,EBP	
01438A9A	FF D7	CALL EDI	edi:IsTrialRestrictionDays
01438A9C	8B 4D F4	MOV ECX,DWORD PTR SS:[EBP-4]	
01438A9F	64 89 0D 00 00	MOV DWORD PTR ES:[0],ECX	
01438AA6	59	POP ECX	
01438AA7	5F	POP EDI	edi:IsTrialRestrictionDays
01438AA8	5E	POP ESI	esi:"IsTrialRestrictionDays"
01438AA9	8B 4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
01438AAC	33 CD	XOR ECX,EBP	
01438AAF	FF 0C 6F 00 00	CALL videoeditor.143F88F	
01438A9A	FF D7	CALL EDI	edi:TrialFirstRun
01438A9C	8B 4D F4	MOV ECX,DWORD PTR SS:[EBP-4]	
01438A9F	64 89 0D 00 00	MOV DWORD PTR ES:[0],ECX	
01438AA6	59	POP ECX	
01438AA7	5F	POP EDI	edi:TrialFirstRun
01438AA8	5E	POP ESI	esi:"TrialFirstRun"
01438AA9	8B 4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
01438AAC	33 CD	XOR ECX,EBP	
01438AAE	E8 0C 6E 00 00	CALL videoeditor.143F88F	

Se detiene otras dos veces más pero siempre por la misma cosa, para validar su estado de Activación. Para resumir, cambié ese **CALL EDI** (**FF D7**) por nuestro **XOR EAX,EAX(33 C0)** y el programa arrancó **FULL**, así que nuestro **PATCH** haga esos seis cambios que al final sirven igual y que al parecer sirve funcionan para todos los programas. A cruzar los dedos porque hasta aquí todo lo he supuesto. Nos queda es reprogramar con este nuevo **PATRÓN**, único para todos. Recordemos que estamos trabajando con 32 Bits, así que para los 64 Bits debe ser otro diferente pero sacado del mismo lugar.

PATRÓN DE BUSQUEDA

```
Dim pHex32F As Byte() = {&HFF, &HD7, &H8B, &H4D, &HF4, &H64, &H89, &HD, &H0, &H0, &H0, &H0, &H59, &H5F, &H5E, &H8B, &H4D, &HF0}
Dim pHex64F As Byte() = {&HFF, &HD7, &H48, &H8B, &H4D, &H20, &H48, &H33, &HCC}
Dim pHex32R As Byte() = {&H33, &HC0, &H8B, &H4D, &HF4, &H64, &H89, &HD, &H0, &H0, &H0, &H0, &H59, &H5F, &H5E, &H8B, &H4D, &HF0}
Dim pHex64R As Byte() = {&H33, &HC0, &H48, &H8B, &H4D, &H20, &H48, &H33, &HCC}
```

PATRÓN DE REEMPLAZO

Esos son los **PATRONES DE BYTES**. Si observan bien la única diferencia son los dos primeros **BYTES** que son el cambio que hicimos, **{FF D7}=>{33 C0}**. Lo ideal sería poder para este **MÉTODO #3** poder utilizar una búsqueda y reemplazo con **REGEX**, porque de esa forma podríamos crear un **PATRÓN** en donde podría tomar como válido **BYTES** que no son iguales al **PATRÓN** inicial pero que se toman como correctos gracias al **REGEX**. Por desgracia por más que busqué cómo hacerlo no lo pude encontrar, así que por ahora seguiremos trabajando con lo que sabemos; quién quita que más adelante aprendamos como usar el con **MÉTODO #3** con **REGEX**. Sigamos mirando en qué cambió el código.

```
Public Sub Extrac()
retry:
    txtInfo.Text += "Archivo seleccionado: " & TargetFile & vbCrLf & vbCrLf

    Try
        If System.IO.File.Exists(TargetFile) Then
            Dim F As Integer = 0 : Dim R As Integer = 0

            Dim BR As Byte() = IO.File.ReadAllBytes(TargetFile)
            IO.File.Move(TargetFile, TargetFile & ".BAK")

            For F = 0 To CInt(BR.Length)
                If Not DP(BR, F) Then
                    Continue For
                End If

                For R = 0 To patronHex.Length - 1
                    BR(F + R) = resourceBytes(R)
                Next R
            Next F

            IO.File.WriteAllBytes(TargetFile, BR)
        End If
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
End Sub
```

Recordemos que **Public Sub Extrac()**, es donde se hace el **PATCH**. Aquí hacemos uso de una función para buscar el **PATRÓN**, y si se halla el **PATRÓN** reemplazamos los **BYTES** o hacemos el **PATCH**. Miremos la función que es lo nuevo.

```
Private Shared Function DP(ByRef secuencia As Byte(), ByRef position As Integer) As Boolean
    If position + patronHex.Length > secuencia.Length Then
        Return False
    End If

    For i As Integer = 0 To patronHex.Length - 1
        If patronHex(i) <> secuencia(position + i) Then
            Return False
        End If
    Next

    Return True
End Function
```

Revisa si nuestro **PATRÓN** está a partir de la "position" que trae la función. Aquí es donde deberíamos agregar el **REGEX** pero mi conocimiento y nivel de programación no me da. Ahora, a poner a prueba esta actualización del Activador.



Funciona bien. Lo probé con todos y no falló. Creo que este será mi versión final, solo a esperar que salgan nuevas versiones de productos **MOVAVI** y ponerlo a prueba.

PARA TERMINAR

Por el camino vamos aprendiendo. Tratando de dejar algo de material para futuras consultas, y si alguien quiere entender algo de programar en **.NET** este tutorial creo que puede ayudar.

Ahora que estaba por terminar salió una nueva versión del <**Movavi Video Editor 15 Business v15.5.0**>, lo cual fue ideal para ponerlo a prueba con una nueva versión, y al activar esta nueva versión el Activador hizo su trabajo.

Agradecerles a todos mis amigos de **CracksLatinos** y **PeruCrackers** por toda la ayuda y conocimiento que siempre me brindan.

Me despido, espero haya sido un fructífero tutorial.

@LUISEFECAB