

Acerca de Jamcast version 1.5.1.111

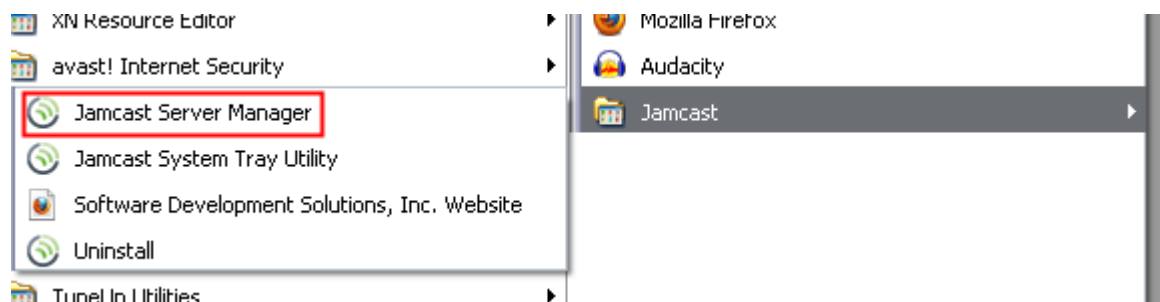


Hola listeros.

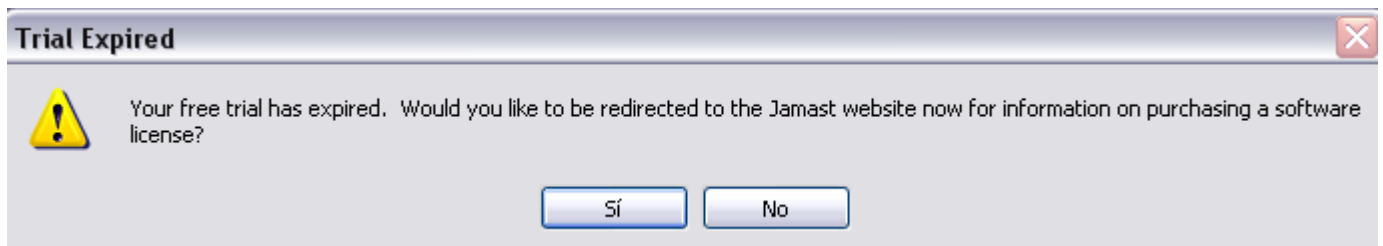
Escribo este pequeño tuto sobre Jamcast y lo que voy encontrando pues para que no me pase lo de siempre: cosas escritas en borrador que al final se pierden y se olvidan. Me pareció buena idea pasarlo a limpio y así se mantiene y puede que anime a alguien mas para seguir mirando conmigo. El amigo Sherab de la lista de CracksLatinos pidió algo de info sobre el protector SmartAssembly para NET y quise bajarlo para echarle una miradita.

Para mas info <http://www.sdstechnologies.com/>

Bien, yo me lo bajé hace tiempo y cuando lo corro me figura como expirado. Solo te dan 15 dias para probarlo. Cuando se instala, tenemos esto:



Si le doy click a la pestaña Jamcast Server Manager me aparece el mensaje de que lo tengo caducado:



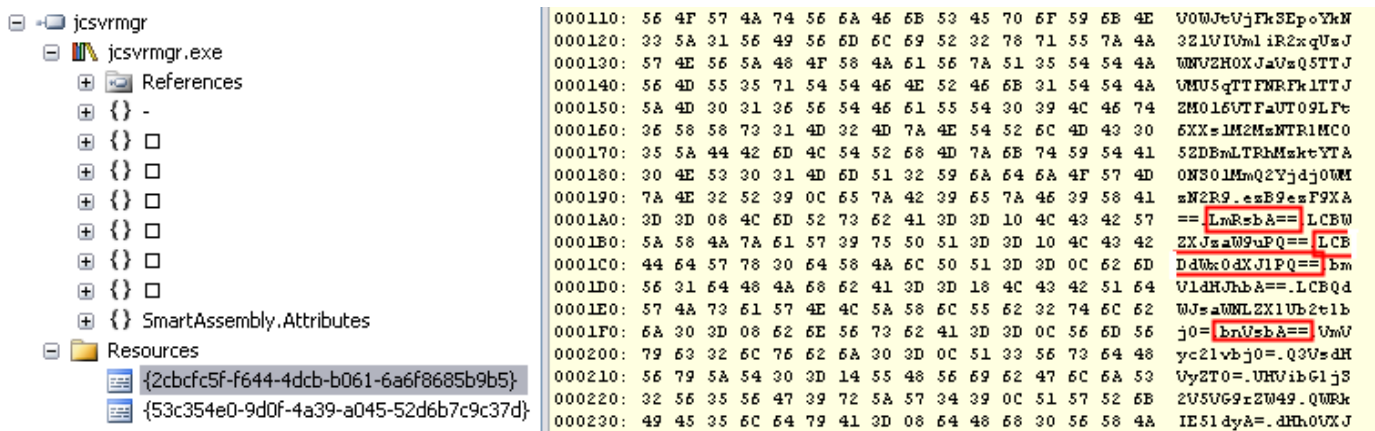
Basicamente me viene a decir que mi periodo de pruebas expiró y que si quiero, me pueden redirigir a la pagina del Jamcast para comprar una licencia. Si le digo que no, se me abre el programa:



Bah, lo cierro y empezamos a pensar. Tenemos cadenas de texto para buscar en Reflector. Vamos tirando del hilo e intentamos ir solucionando esto pero no es así de sencillo.

El programa está protegido con SmartAssembly. Está ofuscado, tiene un alto grado de Control Flow, es decir, en un bloque de código, el flujo de ejecución salta mucho de una zona a otra y se complica mucho su seguimiento. Además, las cadenas de texto están encriptadas en formato Base64 y la única referencia que podemos tener de ellas es un número.

SmartAssembly codifica las cadenas de texto y las guarda en los recursos. Esto lo podemos ver en Reflector:



Con algún ejemplo en rojo de cadenas codificadas en Base64. Cuando el programa necesita una cadena de texto codificada, le entrega a la zona de código encargada de decodificar dicha cadena, un número. Vean unos ejemplos:

```

L_0005: brtrue.s L_0018
L_0007: ldc.i4 0x3006
L_000c: call string 0.0::0(int32)
L_0011: br.st_001e

```

```

: ldftd string 0.0::0
: ldc.i4 0x3184
: call string 0.0::0(int32)
: call string [mscorlib]System.String::Concat(string, string, string)
: stloc.s str6

```

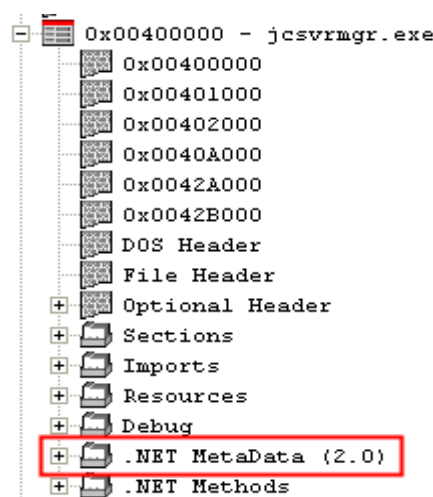
En la primera imagen, se toma el entero en formato hexadecimal 0x3006 y se le entrega a una call lo mismo que en la otra imagen, que se toma 0x3184. Esa call que se ve toda ofuscada es la que se encarga de tomar una cadena u otra de los recursos y decodificarla. Obviamente, para decodificar una cadena que está en Base64, hace falta algo y ese algo es el espacio de nombres System.Convert::FromBase64String de la librería mscorlib. Bien, una vez tenemos un número se entra en la call y buscamos System.Convert::FromBase64String:

```

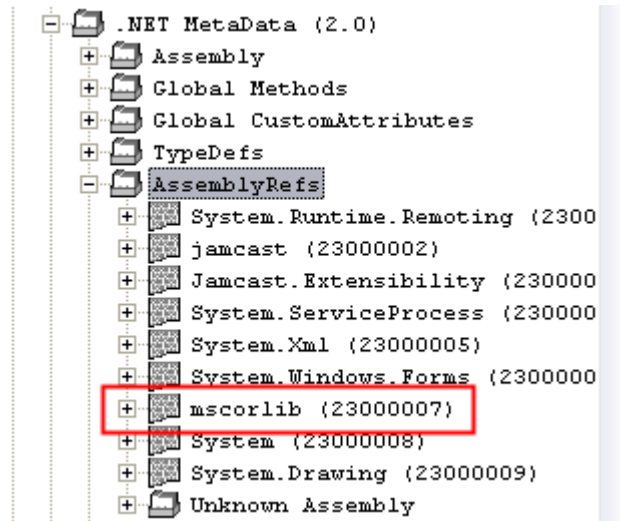
: callvirt instance string [mscorlib]System.Text.Encoding::GetString(uint8[], int32, int32)
: call uint8[] [mscorlib]System.Convert::FromBase64String(string)
: stloc.s buffer3
: call class [mscorlib]System.Text.Encoding [mscorlib]System.Text.Encoding::get_UTF8()

```

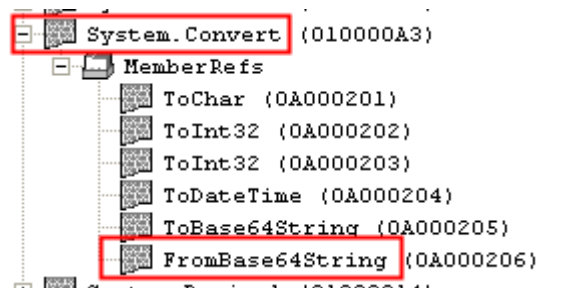
Pues cada vez que se entre en la call decodificadora y se pase por System.Convert::FromBase64String obtenemos una cadena desencriptada y entendible. Vale, esto en Reflector mas o menos se comprende pero si queremos hacer una prueba con un depurador, ¿donde pongo un breakpoint?. Muy sencillo: yo que suelo utilizar PeBrowse, cuando cargo el programa, me voy a la sección NET Metadata del programa cargado:



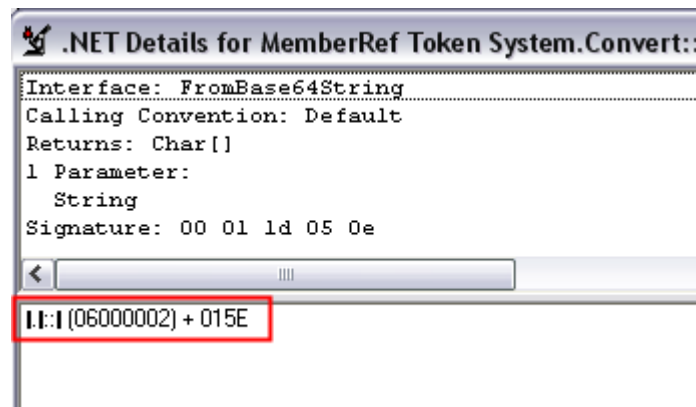
Lo despliego y busco en la seccion AssemblyRefs:



Bien, dijimos antes, que el espacio de nombres System.Convert::FromBase64String está en la libreria mscorlib. Despliego la seccion mscorlib y busco System.Convert:



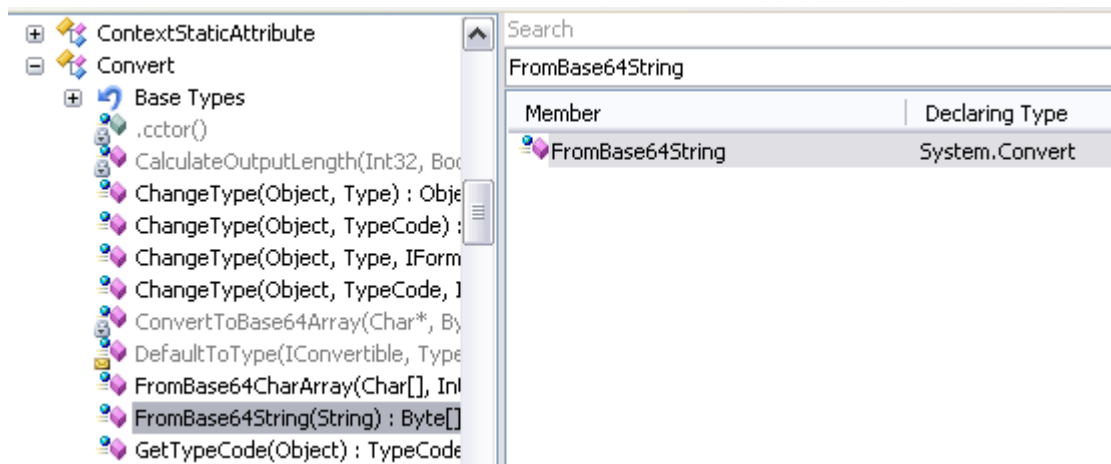
Vale, ahora que tengo localizado lo que necesito, le hago doble click y me sale una ventana indicandome desde dónde se hace referencia en el programa, a la funcion FromBase64String:



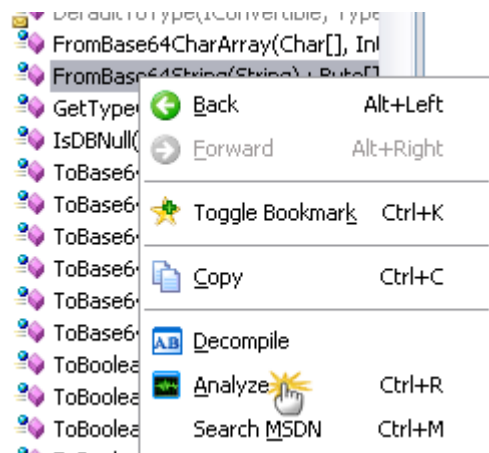
Ahí en recuadro rojo, dónde está en el programa la referencia a FromBase64String. De nuevo le hago doble click y me lleva a la zona de código concreta:

IL_0159: 6F5D02000A	callvirt String System.Text.Encoding::GetString(Char[], Int32, Int32)
IL_015E: 280602000A	call Char[] System.Convert::FromBase64String(String)
IL_0163: 1307	stloc.s 0x07
IL_0165: 285E02000A	call Class System.Text.Encoding System.Text.Encoding::get_UTF8()
IL_016A: 1107	ldloc.s 0x07
IL_016C: 16	ldc.i4.0
IL_016D: 1107	ldloc.s 0x07
IL_016F: 8E	ldlen
IL_0170: 69	conv.i4
IL_0171: 6F5D02000A	callvirt String System.Text.Encoding::GetString(Char[], Int32, Int32)
IL_0176: 28E601000A	call String System.String::Intern(String)
IL_017B: 2A	ret

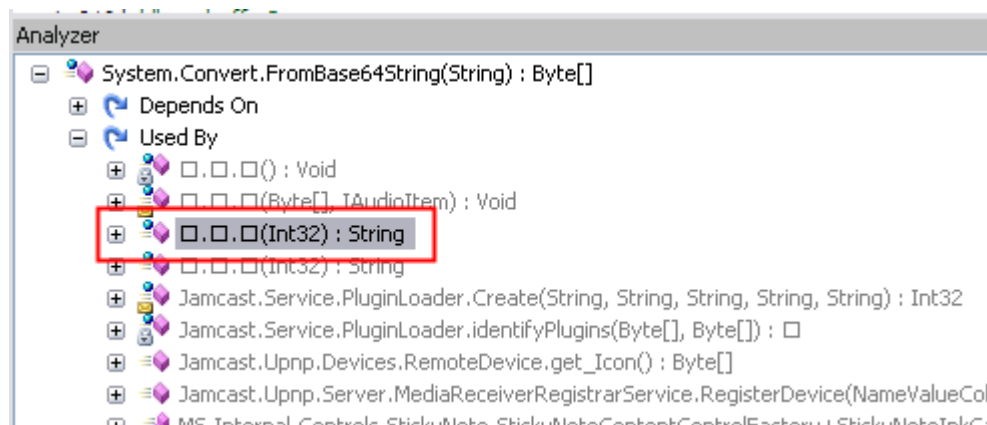
Tengo entonces en PeBrowse, localizado el punto donde el programa decodifica las strings que estan en formato Base64. Si esto mismo lo busco en Reflector:



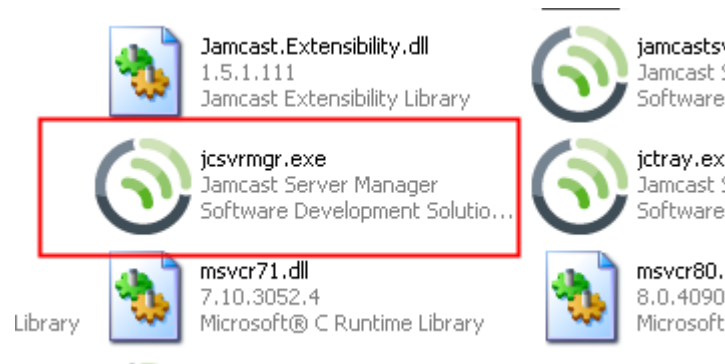
Tengo localizado el miembro FromBase64String. En la ventana de la izquierda, si me situo encima de FromBase64String y con el boton derecho del raton le digo que analize el miembro:



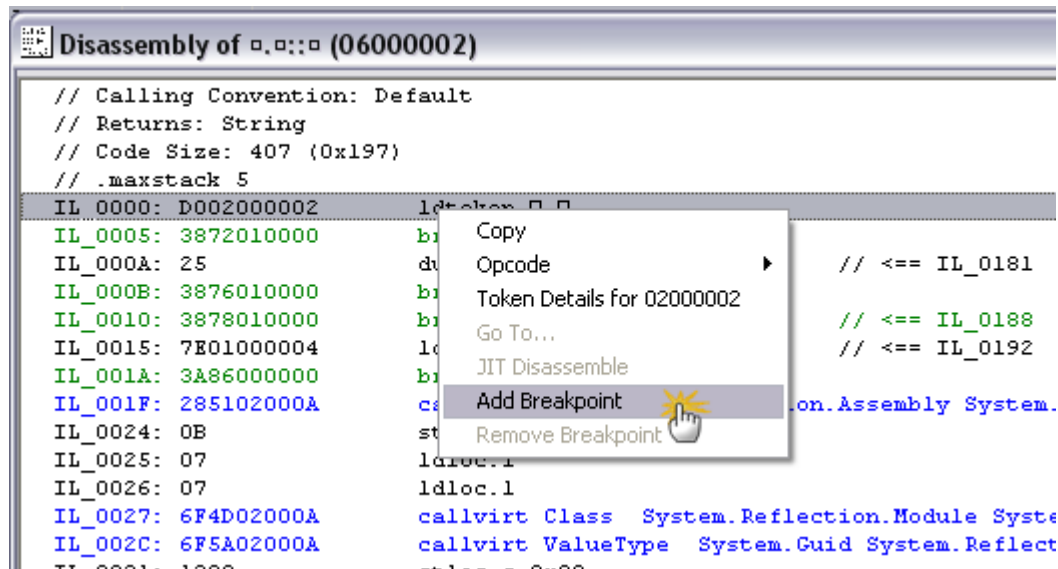
Me empieza a buscar desde donde se utiliza FromBase64String:



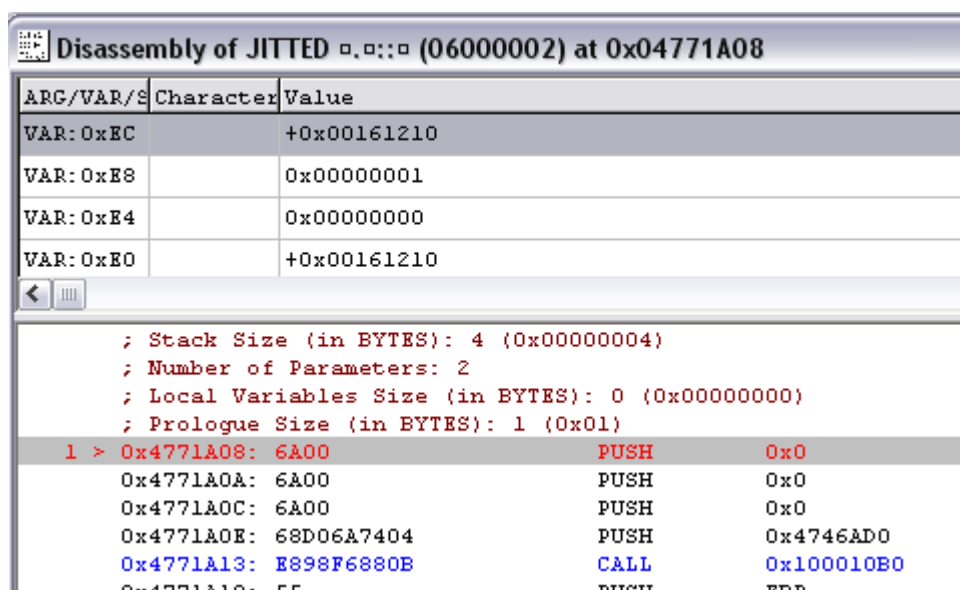
El resultado del analisis me dice que FromBase64String se está utilizando desde varios sitios. Reflector encuentra mas resultados por que incluso analiza las dll's que utiliza el programa y en PeBrowse, solo tengo cargado el ejecutable principal y aun no llamó a ninguna dll ni nada:



Bien, en PeBrowse, en la zona de código donde me encontré la referencia a FromBase64String, tiro para arriba con la barra de desplazamiento y pongo un BP al principio, en la primera línea de código:



Me sitúo en la primera línea, la selecciono y con el botón derecho del ratón le digo que me ponga un bp. Esto significa, que cuando el programa cargado en PeBrowse empiece a correr, se parará cuando se acceda a lo que interesa en este momento que es ver qué cadenas de texto obtenemos gracias a FromBase64String. Así que con el BP puesto, doy Run a PeBrowse:



Ya después de hacerle Run, se paró como esperaba en el BP que le puse. Si miro los registros actuales:

```

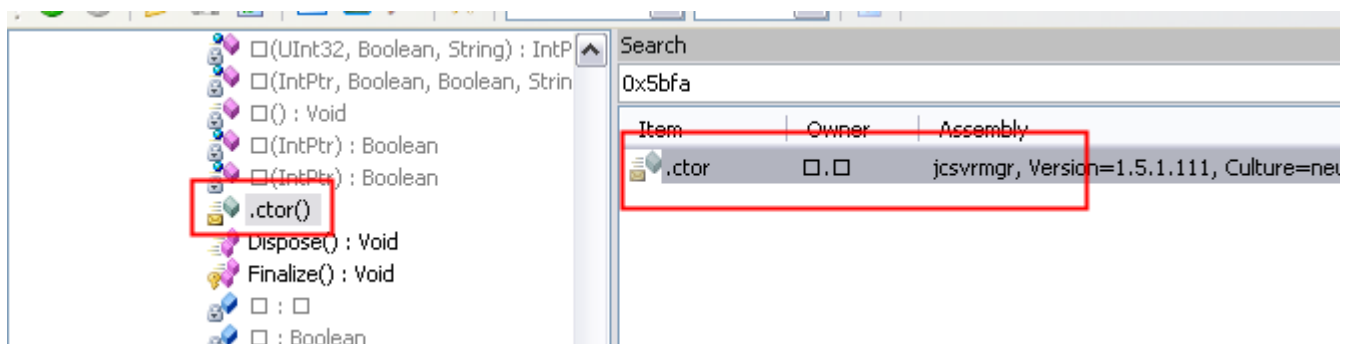
Registers at EIP: 0x04771A08
EAX: +0x04746AD0
EBX: +0x0012F48C (Thread 0x1EC Stack Area)
ECX: 0x00005BFA
EDX: 0x00000000
EDI: +0x0012F450 (Thread 0x1EC Stack Area)
ESI: +0x01432B14 (.NET GC Heap (Small Object))
ESP: +0x0012F37C

```

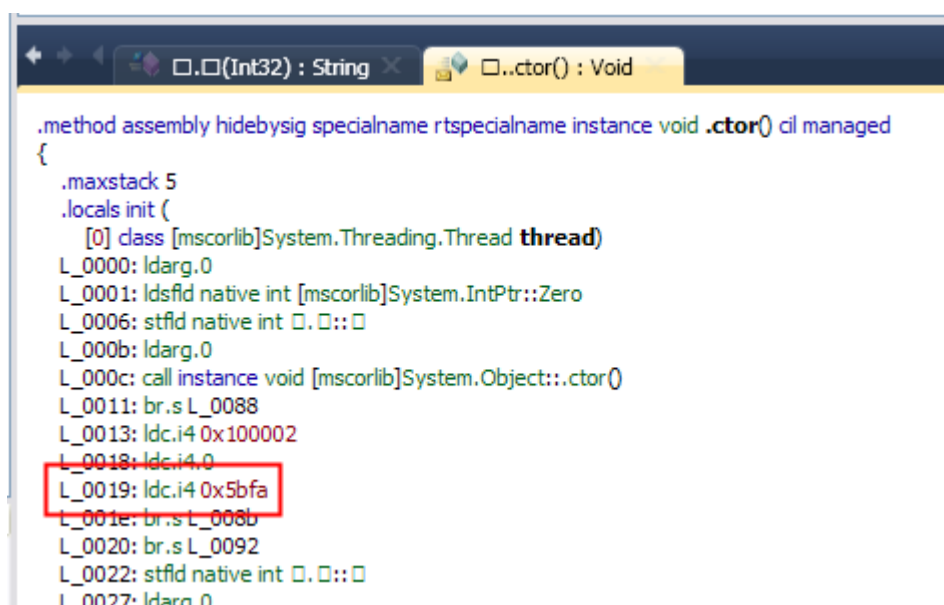
Despues de todas las pruebas que fui haciendo, descubri que siempre es igual. A la zona de codigo encargada de desencriptar, siempre se entra con un valor numerico en ECX y que en este caso, es 5BFA. Bien, si quisiera saber de donde viene ese valor, solo tengo que buscarlo en Reflector:



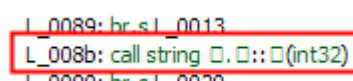
Le pulso en el icono de busqueda y le escribo lo que quiero buscarle seleccionando la tercera opcion que es la busqueda de strings ó constantes. 0x5bfa es una constante que utiliza el programa como cifra y cuando la encuentra así me lo hace saber:



Le hago doble click y Reflector me lleva donde encontró la constante 0x5bfa:



En la linea L_0019 está la constante 0x5bfa que en este caso es un integro. La siguiente linea L_001e es un salto que nos lleva a la linea L_008b y allí tenemos:



El bloque de código donde se descripta y justo donde está parado PeBrowse. Recuerden, que a la rutina que se encarga de descriptar, se entra con un entero (int32) que en este caso es el hexadecimal 5BFA. Bien, no voy a describir línea por línea todo lo que se hace, pero traceando nos encontramos lo siguiente:

Assembly code snippet:

```

1 0x4771AD8: 0F856E010000 JNE 0x4771C4C ; (*+0x174)
; IL_001F: call System.Reflection.Assembly::GetExecutingAssembly()
1 0x4771ADE: FF15B8C02801 CALL DWORD PTR [0x128C0B8]; (0x0128C0B8)
1 > 0x4771AE4: 898530FFFFFF MOV DWORD PTR [EBP-0xD0],EAX; VAR:0xD0
; IL_0024: stloc.1

```

Memory dump (EAX: +0x014317C4 (.NET GC Heap (Small Object))):

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Hex	ASCII
0x014317C4	08	C6	28	01	00	00	00	00	00	00	00	00	00	00	00	00	..(.....	
0x014317D4	68	17	65	03	00	00	00	00	00	34	91	9B	00	2D	00	00	h.e...4...-	
0x014317E4	2C	00	00	00	43	00	3A	00	5C	00	41	00	72	00	63	00	,...C...A.r.c.	
0x014317F4	68	00	69	00	76	00	6F	00	73	00	20	00	64	00	65	00	h.i.v.o.s..d.e.	
0x01431804	20	00	70	00	72	00	6F	00	67	00	72	00	61	00	6D	00	.p.r.o.g.r.a.m.	
0x01431814	61	00	5C	00	4A	00	61	00	6D	00	63	00	61	00	73	00	a.\.J.a.m.c.a.s.	
0x01431824	74	00	5C	00	6A	00	63	00	73	00	76	00	72	00	6D	00	t.\.j.c.s.v.r.m.	
0x01431834	67	00	72	00	2E	00	65	00	78	00	65	00	00	00	00	00	g.r...e.x.e....	
0x01431844	00	00	00	00	20	46	28	01	24	00	00	00	22	00	3C	00F{.\$.\"."<.	
0x01431854	3E	00	7C	00	00	00	01	00	02	00	03	00	04	00	05	00	>.	

Se hace una llamada a GetExecutingAssembly o lo que es lo mismo, el ensamblado que se está ejecutando y el resultado lo tenemos en EAX. A continuación, el recurso con el que se va a jugar:

Assembly code snippet:

```

; IL_0035: ldstr "B"
; IL_003A: call System.Guid::ToString()
1 0x4771B27: 8D7DB8 LEA EDI,[EBP-0x48] ; VAR:0x48
1 0x4771B2A: 8D7580 LEA ESI,[EBP-0x80] ; VAR:0x80
1 0x4771B2D: F30F7E06 MOVQ XMM0,MMWORD PTR [ESI]
1 0x4771B31: 660FD607 MOVQ MMWORD PTR [EDI],XMM0
1 0x4771B35: F30F7E4608 MOVQ XMM0,MMWORD PTR [ESI+0x8]
1 0x4771B3A: 660FD64708 MOVQ MMWORD PTR [EDI+0x8],XMM0
1 0x4771B3F: 8D4DB8 LEA ECX,[EBP-0x48] ; VAR:0x48
1 0x4771B42: 8B15BC204302 MOV EDX,DWORD PTR [0x24320BC]
1 0x4771B48: FF158CFB2801 CALL DWORD PTR [0x128FB8C]
1 > 0x4771B4E: 898524FFFFFF MOV DWORD PTR [EBP-0xDC],EAX; VAR:0xDC

```

Memory dump (EAX: +0x01433C7C (.NET GC Heap (Small Object))):

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Hex	ASCII
0x01433C7C	34	91	9B	00	27	00	00	00	26	00	00	00	7B	00	32	00	4...'.2.	
0x01433C8C	63	00	62	00	63	00	66	00	63	00	35	00	66	00	2D	00	c.b.c.f.c.5.f.-	
0x01433C9C	66	00	36	00	34	00	34	00	2D	00	34	00	64	00	63	00	f.6.4.4.-.4.d.c.	
0x01433CAC	62	00	2D	00	62	00	30	00	36	00	31	00	2D	00	36	00	b.-.b.0.6.1.-.6.	
0x01433CBC	61	00	36	00	66	00	38	00	36	00	38	00	35	00	62	00	a.6.f.8.6.8.5.b.	
0x01433CCC	39	00	62	00	35	00	7D	00	00	00	00	00	00	00	00	00	9.b.5.).....	
0x01433CDC	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Lo siguiente es restarle a EAX, que en este momento tiene nuestro 5BFA, el contenido de la posición de memoria 993250 que contiene 2FE2, leídos en orden inverso:

Assembly code snippet:

```

; IL_00B2: callvirt System.IO.Stream::set_Position()
1 0x4771C4C: 8B45D8 MOV EAX,DWORD PTR [EBP-0x28]; VA:
1 > 0x4771C4F: 2B0550329900 SUB EAX,DWORD PTR [0x993250]
0x4771C55: 99 CDQ

```

Registers at EIP: 0x04771C4F

Register	Value
EAX	0x00005BFA
EBX	+0x0012F48C (Thread 0x988)

El resultado de la resta es 2C18 que servirá para establecer una posición en el contenido del recurso con el que se va a jugar, es decir, donde están todas las cadenas encriptadas. Esto se hace con el espacio de nombres System.IO.Stream::Set_Position.

Ahora, del stream o flujo de bytes que contiene el recurso, se lee el byte de la posicion 2C18:

```

; IL_00BC: callvirt System.IO.Stream::ReadByte()
1 0x4771C63: 8B0DE81E4302 MOV ECX,DWORD PTR [0x2431EE8]
1 0x4771C69: 8B01 MOV EAX,DWORD PTR [ECX]
1 0x4771C6B: FF90A0000000 CALL DWORD PTR [EAX+0xA0]
1 > 0x4771C71: 8945B0 MOV DWORD PTR [EBP-0x50],EAX;
; IL_00C1: stloc.s 0x05
0x4771C74: 8B45B0 MOV EAX,DWORD PTR [EBP-0x50];
0x4771C77: 8945CC MOV DWORD PTR [EBP-0x34],EAX;

```

ANALYZE 0x0012F378

Registers at EIP: 0x04771C71

EAX: 0x00000018 (ERROR_BAD_LENGTH)

EBX: +0x0012F48C (Thread 0x988 Stack Area)

En la posicion 2C18 del stream, tenemos el byte 18. Si el recurso lo abrimos con un editor hexadecimal y buscamos dicha posicion:

2C00	3356	755A	4334	3D10	5548	4A76	596D	786C	3VuZC4=.UHJvYmx1
2C10	6253	4237	4D48	303D	1854	4739	6A59	5778	bSB7MH0=TG9jYWx
2C20	6361	6D4E	6663	3268	7664	3139	6D62	334A	camNfc2hvd19mb3J
2C30	7414	5647	567A	6443	427A	6457	4E6A	5A57	t.VGVzdCBzdWNjZW
2C40	566B	5A57	5168	1056	4756	7A64	4342	6D59	VkZWQh.VGVzdCBmY

Ahora, se leen 18 bytes desde el byte siguiente al puntero de posicion:

```

; IL_0159: callvirt System.Text.Encoding::GetString()
1 0x4771DB2: 6A00 PUSH 0x0
1 0x4771DB4: 8B8578FFFFFF MOV EAX,DWORD PTR [EBP-0x88]; VAR
1 0x4771DBA: FF7004 PUSH DWORD PTR [EAX+0x4]
1 0x4771DBD: 8B9578FFFFFF MOV EDX,DWORD PTR [EBP-0x88]; VAR
1 0x4771DC3: 8B8D48FFFFFF MOV ECX,DWORD PTR [EBP-0xB8]; VAR
1 0x4771DC9: 8B01 MOV EAX,DWORD PTR [ECX]
1 0x4771DCB: FF90DC000000 CALL DWORD PTR [EAX+0xDC]
1 > 0x4771DD1: 898544FFFFFF MOV DWORD PTR [EBP-0xBC],EAX; VAR
; IL_015E: call System.Convert::FromBase64String()
0x4771DD7: 8B8D44FFFFFF MOV ECX,DWORD PTR [EBP-0xBC]; VAR
0x4771DDD: E855FD9275 CALL GetAddrOfContractShutdownFlag
0x4771DE2: 898540FFFFFF MOV DWORD PTR [EBP-0xC0],EAX; VAR
; IL_0163: stloc.s 0x07

```

EAX: +0x01435504 (.NET GC Heap (Small Object))

	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	456789ABCDEF0123
0x01435504	34	91	9B	00	19	00	00	00	18	00	00	00	54	00	47	00	4.....T.G.
0x01435514	39	00	6A	00	59	00	57	00	78	00	63	00	61	00	6D	00	9.j.Y.W.x.c.a.m.
0x01435524	4E	00	66	00	63	00	32	00	68	00	76	00	64	00	31	00	N.f.c.2.h.v.d.l.
0x01435534	39	00	6D	00	62	00	33	00	4A	00	74	00	00	00	00	00	9.m.b.3.J.t.....
0x01435544	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x01435554	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Y se decodifica con FromBase64String:


```

1 0x4771DD1: 898544FFFFFF MOV     DWORD PTR [EBP-0xBC], EAX; VA
; IL_015E: call System.Convert::FromBase64String()
1 0x4771DD7: 8B8D44FFFFFF MOV     ECX, DWORD PTR [EBP-0xBC]; VA
1 0x4771DDD: E855FD9275 CALL    GetAddrOfContractShutoffFlag
1 > 0x4771DE2: 898540FFFFFF MOV     DWORD PTR [EBP-0xC0], EAX; VA
; IL_0163: stloc.s 0x07
0x4771DE8: 8B8540FFFFFF MOV     EAX, DWORD PTR [EBP-0xC0]; VA
0x4771DEE: 89856CFFFFFF MOV     DWORD PTR [EBP-0x94], EAX; VA

```

EAX: +0x01435548 (.NET GC Heap (Small Object))

	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	89ABCDEF01234567
0x01435548	CO	4E	28	01	12	00	00	00	4C	6F	63	61	6C	5C	6A	63	.N(....Local\jc
0x01435558	5F	73	68	6F	77	5F	66	6F	72	6D	00	00	00	00	00	00	_show_form.....
0x01435568	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x01435578	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x01435588	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x01435598	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Y aquí tenemos la primera cadena decodificada que necesita el programa. Si por ejemplo ponemos un BP en la línea 0x4771DE2 y le vamos dando Run al PeBrowse, imaginaran que en EAX iremos obteniendo las diferentes cadenas de texto que el programa usa para su funcionamiento. Pongo entonces un BP en 0x4771DE2 y le voy dando Run al PeBrowse y vemos unos ejemplos de las cadenas decodificadas:

EAX: +0x01442830 (.NET GC Heap (Small Object))

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0x01442830	CO	4E	28	01	0B	00	00	00	4A	61	6D	63	61	73	74	49	.N(....JamcastI
0x01442840	63	6F	6E	00	00	00	00	00	00	00	00	00	00	00	00	00	con.....
0x01442850	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x01442860	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

EAX: +0x0149E31C

	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	CDEF0123456789AB
0x0149E31C	CO	4E	28	01	17	00	00	00	66	6F	6C	64	65	72	5F	69	.N(....folder i
0x0149E32C	63	6F	6E	5F	31	36	5F	78	5F	31	36	2E	70	6E	67	00	con_16_x_16.png.
0x0149E33C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

EAX: +0x014A025C

	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	CDEF0123456789AB
0x014A025C	CO	4E	28	01	1C	00	00	00	41	75	74	68	6F	72	69	7A	.N(....Authoriz
0x014A026C	65	64	20	66	6F	72	20	72	65	6D	6F	74	65	20	61	63	ed for remote ac
0x014A027C	63	65	73	73	00	00	00	00	00	00	00	00	00	00	00	00	cess.....
0x014A028C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

EAX: +0x014A12D4

	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	456789ABCDEF0123
0x014A12D4	CO	4E	28	01	95	00	00	00	43	68	6F	6F	73	65	20	61	.N(....Choose a
0x014A12E4	20	66	6F	6C	64	65	72	20	74	6F	20	73	68	61	72	65	folder to share
0x014A12F4	2E	20	20	41	6C	6C	20	73	75	70	70	6F	72	74	65	64	. All supported
0x014A1304	20	6D	65	64	69	61	20	77	69	74	68	69	6E	20	74	68	media within th
0x014A1314	65	20	73	65	6C	65	63	74	65	64	20	66	6F	6C	64	65	e selected folde
0x014A1324	72	20	61	6E	64	20	69	74	73	20	73	75	62	66	6F	6C	r and its subfol
0x014A1334	64	65	72	73	20	77	69	6C	6C	20	62	65	20	61	76	61	ders will be ava
0x014A1344	69	6C	61	62	6C	65	20	74	6F	20	55	50	6E	50	20	64	ilable to UPnP d
0x014A1354	65	76	69	63	65	73	20	6F	6E	20	79	6F	75	72	20	6C	evices on your l
0x014A1364	6F	63	61	6C	20	6E	65	74	77	6F	72	6B	2E	00	00	00	ocal network....
0x014A1374	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Despues de unos cuantos clicks en PeBrowse al F5 (Run) voy mirando en EAX las cadenas decodificadas. Pero andar así esperando ver algun mensaje interesante puede llevarnos horas por que imaginense la cantidad de cadenas de texto que pueda necesitar el programa tan solo para inicializarse. Igual pueden ser cientos de ellas. En mi caso, como ya hace tiempo que instalé el programa, lo tengo caducado y cuando lo abro, me aparece esta nag de aviso:

Trial Expired

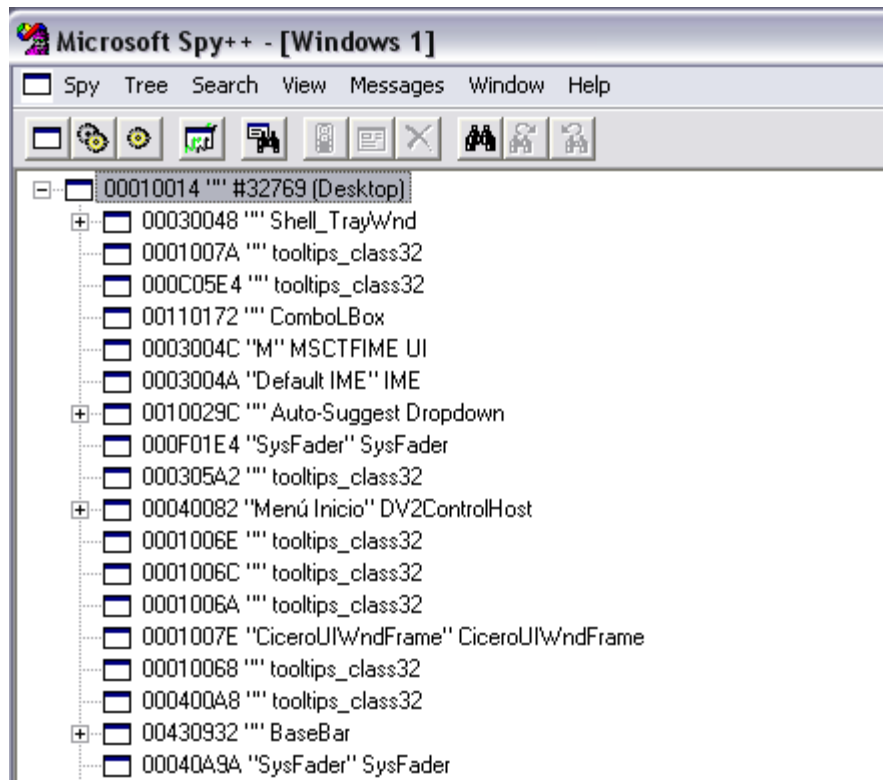


Your free trial has expired. Would you like to be redirected to the Jamast website now for information on purchasing a software license?

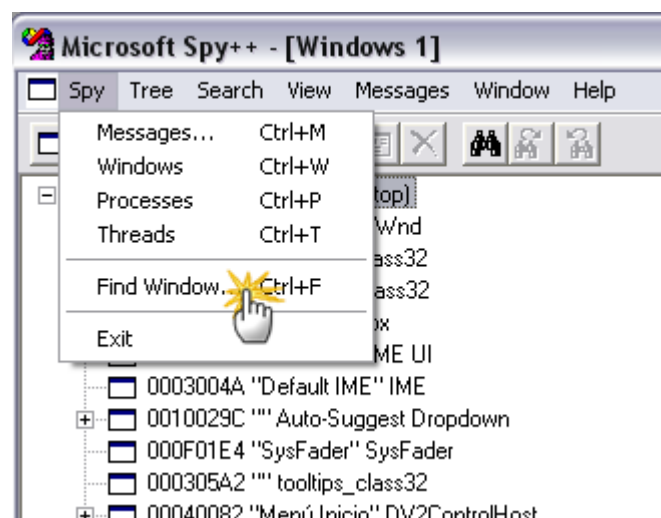
Sí

No

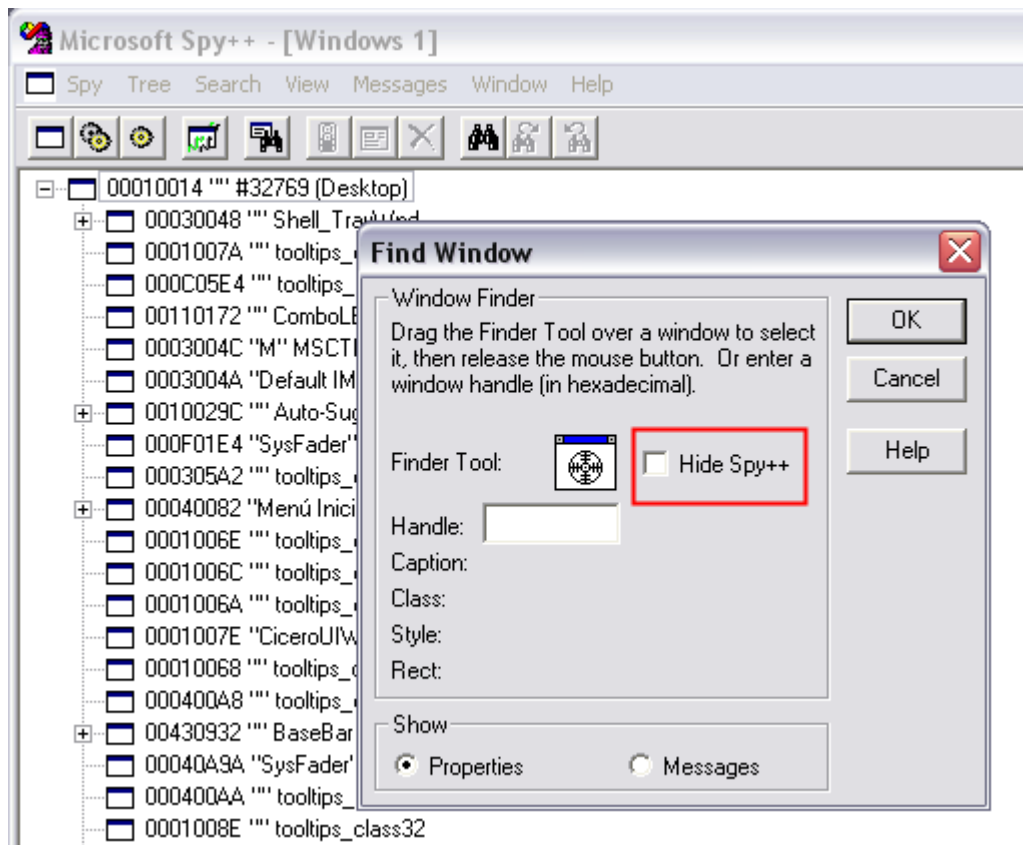
Vale, en algun momento, el programa pasó por FromBase64String, decodificó y utilizó esta cadena de texto para avisarme que ya me caducó. Entonces se me ocurrió lo siguiente: con la caja de mensajes abierta, utilizo un gestor de ventanas para capturar precisamente eso: ventanas con sus handles textos etc. Yo utilizo el que trae la plataforma NET:



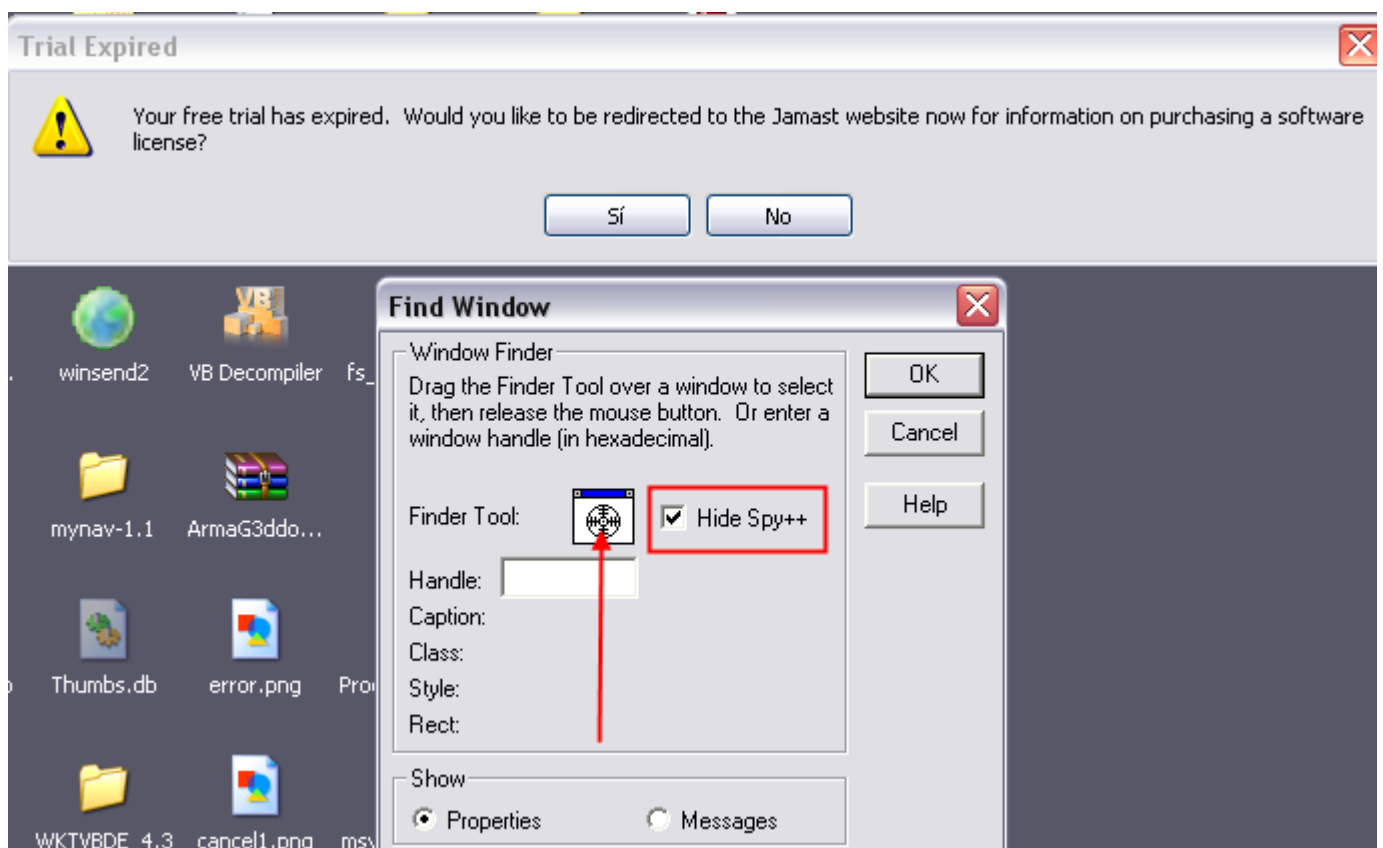
Una vez abierto, me voy al menu y elijo Spy, Find Window:



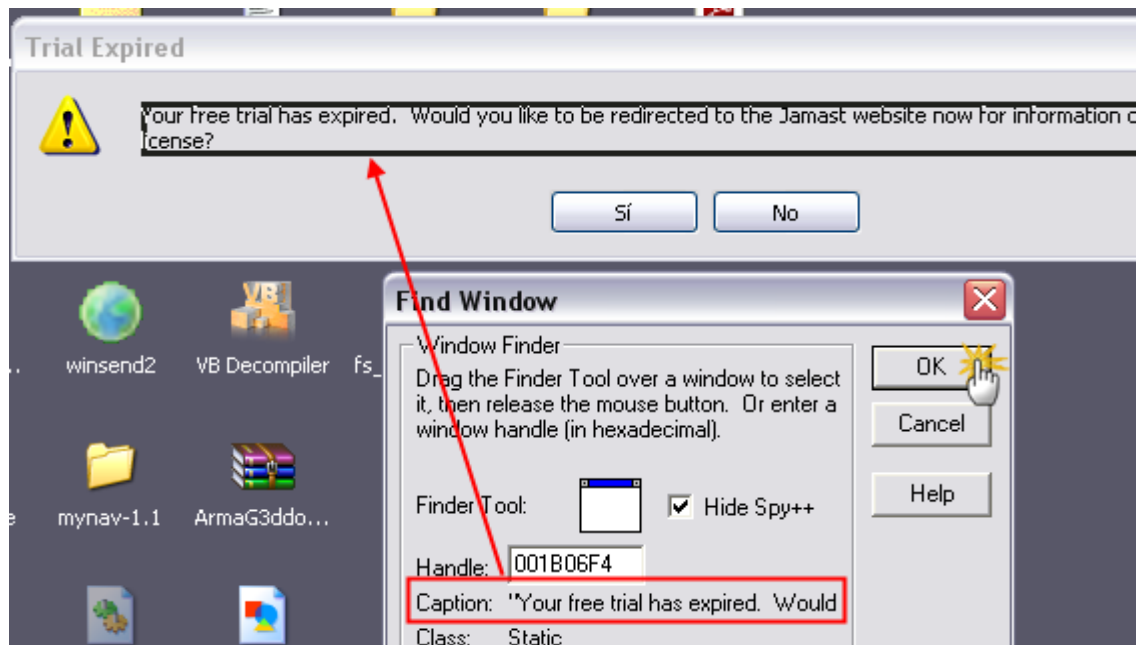
Y en la siguiente imagen vemos lo que me aparece:



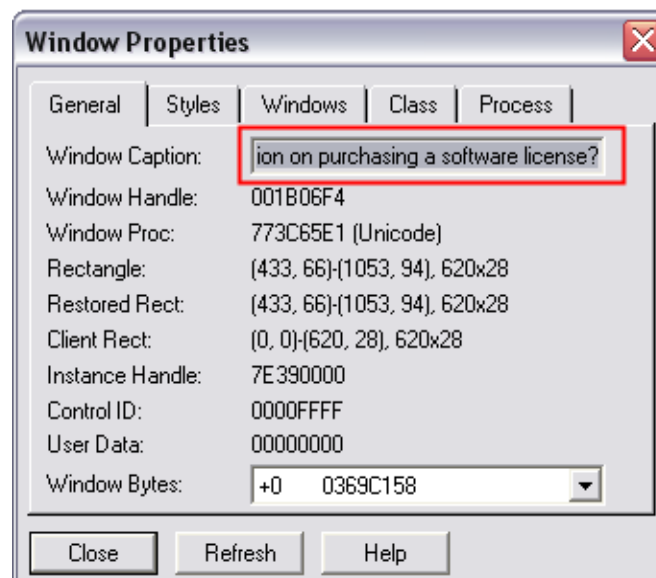
En la casilla de Hide Spy++ le hago click para activarla con lo que la ventana del Microsoft Spy principal desaparezca y no moleste:



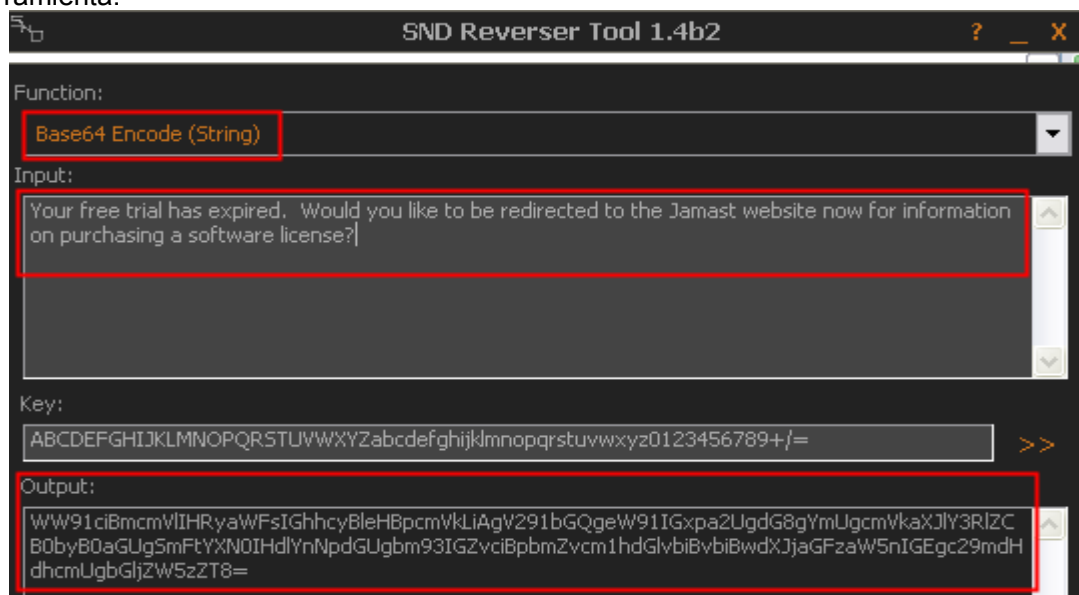
Bien, ya tengo el capturador preparado y el mensaje de aviso de caducado. Señalado con la flechita está el simbolito ese como de un punto de mira. Solo tengo que situarme encima de el y arrastrarlo hasta donde está la cadena de texto avisandome del trial expirado:



Cuando arrastro el punto de mira hacia el texto, éste se encuadra automáticamente en negro y captura el caption del control. Una vez que lo tengo capturado, pulso OK :



El resultado, con la cadena de texto capturada. Solo tengo que copiar la línea entera de texto para utilizarla en la siguiente herramienta:



Con la SND Reverser Tool, elijo la funcion Base64 Encode (string). En el Input le pego la cadena de texto que copié antes y en el Output ya de seguido me aparece el resultado codificado a formato Base64. Como supondran, este resultado codificado lo voy a buscar en el archivo de recurso:

	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F	0123456789ABCDEF
08B0	6B4C	673D	3D80	B857	5739	3163	6942	6D63	kLg==€,WU91ciBmc
08C0	6D56	6C49	4852	7961	5746	7349	4768	6863	mVlIHRyaWFsIGhhc
08D0	7942	6C65	4842	7063	6D56	6B4C	6941	6756	yBleHBpcmVkJiAgV
08E0	3239	3162	4751	6765	5739	3149	4778	7061	291bGQgeW91IGxpa
08F0	3255	6764	4738	6759	6D55	6763	6D56	6B61	2UgdG8gYmUgc mVka
0900	584A	6C59	3352	6C5A	4342	3062	7942	3061	XJlY3RlZCB0byB0a
0910	4755	6753	6D46	7459	584E	3049	4864	6C59	GUgSmFtYXNOIHdlY
0920	6E4E	7064	4755	6762	6D39	3349	475A	7663	nNpdGUgbm93IGZvc
0930	6942	7062	6D5A	7663	6D31	6864	476C	7662	iBpbmZvcmlhdGlvb
0940	6942	7662	6942	7764	584A	6A61	4746	7A61	iBvbiBwdXJjaGFza
0950	5735	6E49	4745	6763	3239	6D64	4864	6863	W5nIGEgc29mdHdhc
0960	6D55	6762	476C	6A5A	5735	7A5A	5438	3D14	mUgbG1jZW5zZT8=.
0970	5648	4A70	5957	7767	5258	6877	6158	4A6C	VHJpYWwgRXhwaXJl
0980	5A41	3D3D	3461	4852	3063	446F	764C	3364	ZA==4aHR0cDovL3d

En PSPad, busco en el archivo de recurso la cadena codificada obtenida con la cripto herramienta. Se ve que la cadena en si empieza en el desplazamiento 8B7 y el byte precedente B8 indica el largo total de la cadena encriptada. B8=184 caracteres. Pues bien, si recordamos de antes, el programa toma un entero que lo mete en la zona de codigo que desencripta. A ese entero le restaba 2FE2 y el resultado era el byte de posicionamiento en el archivo de recurso y a partir de ahí en adelante leía los bytes necesarios. Entonces, hagamos lo contrario: si el byte de posicionamiento es 8B6, entonces 8B6+2FE2=3898 y este valor entero lo deberíamos poder encontrar en Reflector. Hago la prueba:

Search


0x3898

Item	Owner
------	-------

Humm, que raro. No encuentra el entero que yo esperaba. Se me ocurrió utilizar el desplazamiento 8B5 en vez del 8B6. 8B5+2FE2=3897:

Search

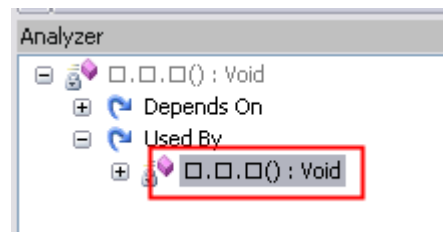
0x3897

Item	Owner
 <input type="checkbox"/>	<input type="checkbox"/> .

Ey, ahora si que encontró el entero 3897. Hago doble click al resultado y Reflector me lleva directamente allí:

```
□.□(Int32) : String x □.□() : Void
L_001d: ldftd class [System.Windows.Forms]System.Windows.Forms.Label □.□::□
L_0022: ldc.i4.1
L_0023: callvirt instance void [System.Windows.Forms]System.Windows.Forms.Control::set_Visible(bool)
L_0028: ldc.i4 0x3897
L_002d: call string □.□::□(int32)
L_0032: ldc.i4 0x3951
L_0037: call string □.□::□(int32)
L_003c: ldc.i4.4
L_003d: ldc.i4.s 0x30
L_003f: call valuetype [System.Windows.Forms]System.Windows.Forms.DialogResult [System.Windows.Forms.MessageBox::Show(st
L_0044: stloc.0
L_0045: ldloc.0
L_0046: ldc.i4.6
L_0047: bne.un.s L_0059
L_0049: ldc.i4 0x3966
L_004e: call string □.□::□(int32)
L_0053: call class [System]System.Diagnostics.Process [System]System.Diagnostics.Process::Start(string)
L_0058: nop
```

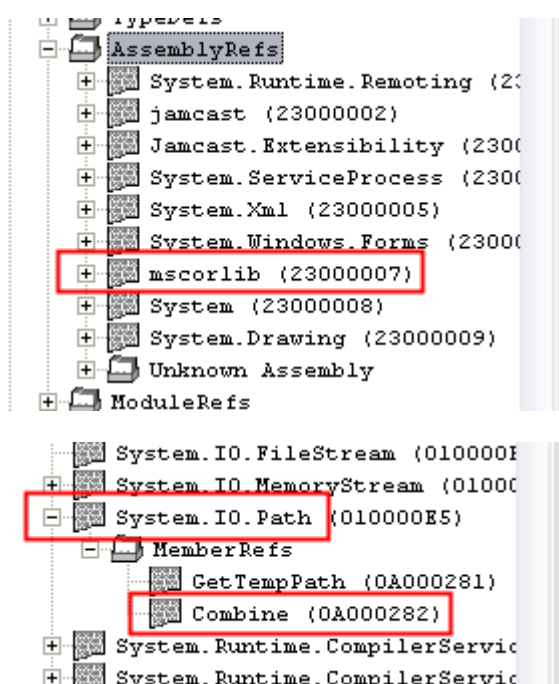
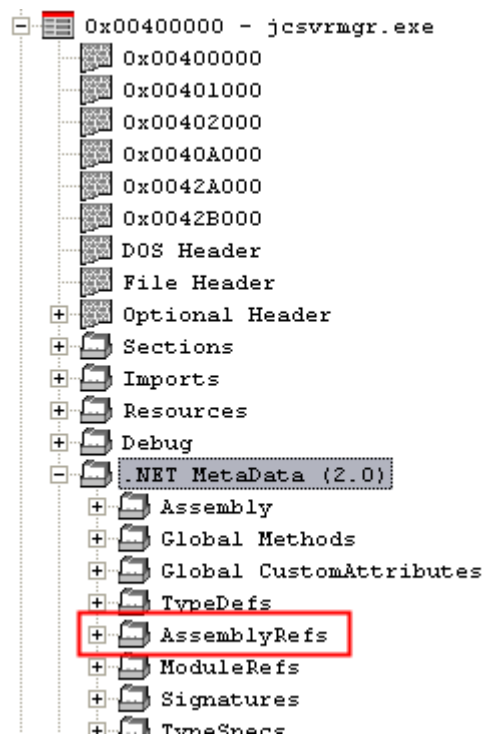
La imagen es mas que explicativa. Como está todo ofuscado, todo este codigo lo tenemos que ver en la vista IL y si cambio a cualquier otra vista, Reflector no puede resolver. Pero se entiende bien el funcionamiento: se toma el entero 0x3897 y se entra en la call decodificadora y la cadena resultante se utiliza en una caja de mensajes. Voy a analizar este codigo y que Reflector me diga quién lo usa:



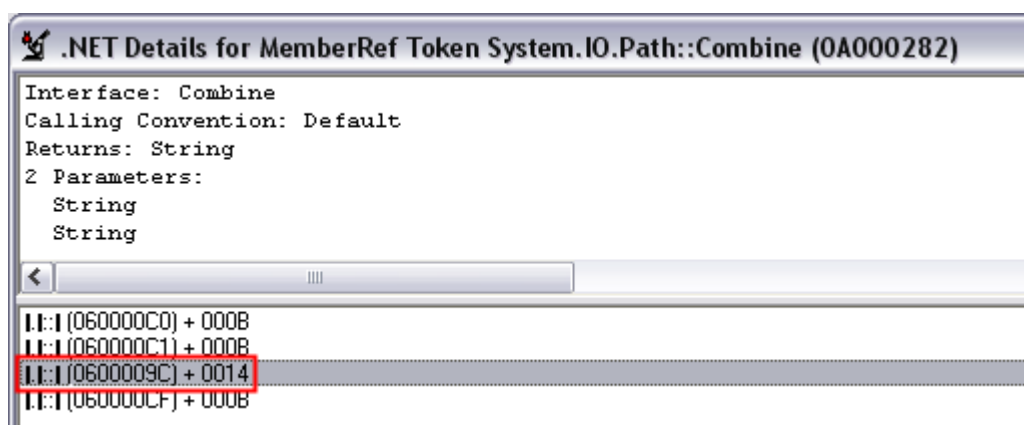
Reflector me encontró, que ese bloque de codigo se utiliza desde lo que encuadré en rojo. Me voy allí:

```
□.□(Int32) : String x □.□() : Void
.maxstack 3
.locals init (
    [0] string str,
    [1] class [System.Xml]System.Xml.XmlDocument document,
    [2] valuetype [mscorlib]System.DateTime time,
    [3] valuetype [mscorlib]System.DateTime time2,
    [4] string str2)
L_0000: br L_0121
L_0005: br L_012b
L_000a: ldc.i4 0x3821
L_000f: br L_0135
L_0014: call string [mscorlib]System.IO.Path::Combine(string, string)
L_0019: stloc.0
L_001a: ldloc.0
L_001b: call bool [mscorlib]System.IO.File::Exists(string)
L_0020: brtrue.s L_002d
L_0022: ldarg.0
L_0023: call instance void □.□::□()
L_0028: leave L_0148
L_002d: newobj instance void [System.Xml]System.Xml.XmlDocument::ctor()
L_0032: stloc.1
L_0033: ldloc.1
```

Y de lo que mas se puede destacar es que en la linea 0014 se hace una llamada a System.IO.Path::Combine a la que se le entregan dos cadenas de texto. Esto hace que apartir de dos cadenas de texto se componga una ruta hacia algun archivo. Bien, en la linea 001b se comprueba si existe un archivo en cierta ruta. En la linea 0020 se comprueba el resultado: si es verdad que existe, se salta a la linea 002b donde se empieza a trabajar con un documento en formato Xml. Si por el contrario es falso y no existe, se llega a la linea 0023 donde iríamos a la call que se encarga de manejar la cadena de texto de chico malo y nos saldría el mensaje con lo del trial caducado. Entonces, si busco en PeBrowse dónde utiliza el programa (por ejemplo) el espacio de nombres System.IO.Path::Combine, contenido en la mscorlib, puedo poner un BP allí y ver de qué archivo y qué ruta se trata:



Bien, si hago doble click en Combine, me aparece desde dónde se utiliza en el programa dicha orden:



Ok. Se encuentran cuatro ocurrencias pero para no extenderme mucho les digo que es el tercer item. Le hago doble click:

Disassembly of 0600009C

Name	References
ARG: This IL_0022, IL_006C, IL_00BD, IL_00CA, IL_00DC, IL_0107, IL_0113, IL_0140	

```

// .maxstack 3
IL_0000: 381C010000    br IL_0121
IL_0005: 3821010000    br IL_012B           // <== IL_0126
IL_000A: 2021380000    ldc.i4 0x00003821    // <== IL_0130
IL_000F: 3821010000    br IL_0135
IL_0014: 288202000A    call String System.IO.Path::Combine(String, String) // <== IL 013A
IL_0019: 0A          stloc.0
IL_001A: 06          ldloc.0
IL_001B: 287C02000A    call Boolean System.IO.File::Exists(String)
IL_0020: 2D0B        brtrue.s IL_002D
IL_0022: 02          ldarg.0              // ARG: This
IL_0023: 289D000006    call Void []::[]()
IL_0028: DD1B010000    leave IL_0148
IL_002D: 73B300000A    newobj Void System.Xml.XmlDocument::.ctor() // <== IL_0020
IL_0032: 0B          stloc.1
IL_0033: 07          ldloc.1
IL_0034: 06          ldloc.0
IL_0035: 6FB500000A    callvirt Void System.Xml.XmlDocument::Load(String)
IL_003A: 07          ldloc.1
IL_003B: 6FB400000A    callvirt Class System.Xml.XmlElement System.Xml.XmlDocument::get_Do
IL_0040: 6FB100000A    callvirt Class System.Xml.XmlAttributeCollection System.Xml.XmlNode
IL_0045: 2032380000    ldc.i4 0x00003832
IL_004A: 2802000006    call String []::[](Int32)
IL_004F: 6FB200000A    callvirt Class System.Xml.XmlAttribute System.Xml.XmlAttributeColle
IL_0054: 6FB000000A    callvirt String System.Xml.XmlNode::get_Value()
IL_0059: 280402000A    call ValueType System.DateTime System.Convert::ToDateTime(String)
  
```

Ciertamente es el mismo código que vimos en Reflector. Si en la línea IL_0014 pongo un BP y corro el programa, obtendremos qué ruta y sobre qué archivo se está preguntando si existe o no:

```

0x5558BC6: 895590        MOV     DWORD PTR [EBP-0x70], EDI; VAR: 0x70
; IL_0000: br IL_0121
; IL_0005: br IL_012B
; IL_000A: ldc.i4 0x00003821
; IL_000F: br IL_0135
; IL_0014: call System.IO.Path::Combine()
1 0x5558BC9: 90          NOP
1 0x5558BCA: E95A030000    JMP     0x5558F29
1 0x5558BCF: 8B4588        MOV     EAX, DWORD PTR [EBP-0x78]; VAR: 0x78; <==
1 0x5558BD2: 894584        MOV     DWORD PTR [EBP-0x7C], EAX; VAR: 0x7C
1 0x5558BD5: E964030000    JMP     0x5558F3E
1 0x5558BDA: 8B857CFFFFFF  MOV     EAX, DWORD PTR [EBP-0x84]; VAR: 0x84; <==
1 0x5558BE0: 898578FFFFFF  MOV     DWORD PTR [EBP-0x88], EAX; VAR: 0x88
1 0x5558BE6: C745CC21380000 MOV     DWORD PTR [EBP-0x34], 0x3821; VAR: 0x34
1 0x5558BED: E968030000    JMP     0x5558F5A
1 0x5558BF2: 8B956CFFFFFF  MOV     EDI, DWORD PTR [EBP-0x94]; VAR: 0x94; <==
1 0x5558BF8: 8B8D70FFFFFF  MOV     ECX, DWORD PTR [EBP-0x90]; VAR: 0x90
1 0x5558BFE: E82D3734FF    CALL    0x489C330 ; (0x0489C330)
1 > 0x5558C03: 898568FFFFFF  MOV     DWORD PTR [EBP-0x98], EAX; VAR: 0x98
; IL_0019: stloc.0
0x5558C09: 8B8568FFFFFF  MOV     EAX, DWORD PTR [EBP-0x98]; VAR: 0x98
0x5558C0F: 894598        MOV     DWORD PTR [EBP-0x68], EAX; VAR: 0x68
; IL_001A: 06
  
```

El breakpoint puesto y que se nota por la línea de código en rojo. Después de darle Run, se para en el BP y cuando traceo unas líneas, paso la llamada a System.IO.Path::Combine y en EAX obtengo el resultado:

```

1  0x5558BFE: 82D3734FF      CALL    0x489C330      ; (0x0489C330)
1 > 0x5558C03: 898568FFFFFF      MOV     DWORD PTR [EBP-0x98],EAX; VAR:0x98

```

EAX: +0x014DD3D4 (.NET GC Heap (Small Object))

	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	456789ABCDEF0123
0x014DD3D4	34	91	9B	00	2C	00	00	00	2B	00	00	00	43	00	3A	00	4...+...C...
0x014DD3E4	5C	00	41	00	72	00	63	00	68	00	69	00	76	00	6F	00	\.A.r.c.h.i.v.o.
0x014DD3F4	73	00	20	00	64	00	65	00	20	00	70	00	72	00	6F	00	s..d.e..p.r.o.
0x014DD404	67	00	72	00	61	00	6D	00	61	00	5C	00	4A	00	61	00	g.r.a.m.a.\.J.a.
0x014DD414	6D	00	63	00	61	00	73	00	74	00	5C	00	6A	00	61	00	m.c.a.s.t.\.j.a.
0x014DD424	6D	00	63	00	61	00	73	00	74	00	2E	00	6C	00	69	00	m.c.a.s.t...l.i.
0x014DD434	63	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	c.....

Esta imagen lo dice todo. Se obtiene una ruta con un archivo: "C:\Archivos de programa\Jamcast\jamcast.lic". Yo desde luego en la ruta esa, donde instalé el Jamcast, no tengo el archivo de licencia "jamcast.lic" y cuando el programa pregunta si existe...

```

; IL_U01A: ldloc.U
; IL_001B: call System.IO.File::Exists()
1  0x5558C12: 8B4D98      MOV     ECX,DWORD PTR [EBP-0x68]
1  0x5558C15: FF15FCDD9B04 CALL    DWORD PTR [0x49BDDFC]
1 > 0x5558C1B: 8945C8      MOV     DWORD PTR [EBP-0x38],EAX

```

Registers at EIP: 0x05558C1B

EAX: 0x00000000 (STATUS_SUCCESS)
EBX: +0x04C30864

Obtengo en EAX un bonito cero. Y el código me lleva directamente a sacarme el cartel que ya sabemos. En el caso de que existiera realmente el archivo de licencia, el salto pasaría por encima de la llamada al cartel y empezaría a trabajar con un documento Xml. Intuyo y es seguro, que el archivo de licencia debe contener una estructura típica de documento Xml y de ahí va leyendo lo que tenga que leer:

```

; IL_U01A: ldloc.U
; IL_001B: call System.IO.File::Exists()
1  0x5558C12: 8B4D98      MOV     ECX,DWORD PTR [EBP-0x68]; VAR:0x68
1  0x5558C15: FF15FCDD9B04 CALL    DWORD PTR [0x49BDDFC]
1  0x5558C1B: 8945C8      MOV     DWORD PTR [EBP-0x38],EAX; VAR:0x38
; IL_0020: brtrue.s IL_002D
1  0x5558C1E: 837DC800    CMP     DWORD PTR [EBP-0x38],0x0; VAR:0x38
1 > 0x5558C22: 750E      JNE     0x5558C32      ; (*+0x10)
; IL_0022: ldarg.0
; IL_0023: call 0.0:0.0()
0x5558C24: 8B4D98      MOV     ECX,DWORD PTR [EBP-0x64]; VAR:0x64
0x5558C27: E8E0B7E3FF CALL    0x539440C
; IL_0028: leave IL_U148
0x5558C2C: 90      NOP
0x5558C2D: E964030000 JMP     0x5558F96
; IL_002D: newobj System.Xml.XmlDocument::.ctor()
0x5558C32: B9808E5005 MOV     ECX,0x5508E80 ; <=0x05558C22(*-0x10)
0x5558C37: E898399274 CALL    DllUnregisterServerInternal + 0x05B8
0x5558C3C: 898564FFFFFF MOV     DWORD PTR [EBP-0x9C],EAX; VAR:0x9C

```

ANALYZE **NO JUMP** **0x0012E9CC** **Line 62 of 362**

El valor de EAX (cero) lo movió a la variable 38, comparó el contenido de la variable 38 con 0 y al ser iguales, el salto no se ejecuta. Solo se ejecuta si No fueran iguales. Por lo tanto me lleva a la call y me saca el cartel chungo.

Reinicio de nuevo PeBrowse y en el directorio donde instalé Jamcast, armo un archivo de licencia para que el programa vea que al menos existe y vemos la diferencia:



```

; IL_001B: call System.IO.File::Exists()
1 0x5569862: 8B4D98 MOV ECX,DWORD PTR [EBP-0x68]; VAR
1 0x5569865: FF15FCDD9B04 CALL DWORD PTR [0x49BDDFC]
1 > 0x556986B: 8945C8 MOV DWORD PTR [EBP-0x38],EAX; VAR
; IL_0020: brtrue.s IL_002D

ANALYZE 0x0012E9CC Line 57 of 362

Registers at EIP: 0x0556986B
EAX: 0x00000001 (ERROR_INVALID_FUNCTION)

```

Reinicié todo y puse un BP en la System.IO.File::Exist. Le di Run y se paró en el BP. Paso la call con F10 y en EAX tengo ahora un 1, señal que existe el jamcast.lic. En la comparacion, compara 1 con 0 y como NO son iguales, el salto se ejecuta y comienza a manejar lo del tema de los Xml:

```

; IL_0020: brtrue.s IL_002D
1 0x556986E: 837DC800 CMP DWORD PTR [EBP-0x38],0x0; VAR:0x38
1 v 0x5569872: 750E JNE 0x5569882 ; (*+0x10)
; IL_0022: ldarg.0
; IL_0023: call 0.0:0.0()
0x5569874: 8B4D9C MOV ECX,DWORD PTR [EBP-0x64]; VAR:0x64
0x5569877: E8E0ABE2FF CALL 0x539445C
; IL_0028: leave IL_0148
0x556987C: 90 NOP
0x556987D: E964030000 JMP 0x5569BE6
; IL_002D: newobj System.Xml.XmlDocument::.ctor()
* 0x5569882: B9BC905005 MOV ECX,0x55090BC ; <=0x05569872(*-0x10)
0x5569887: E8482D9174 CALL DllUnregisterServerInternal + 0x05B8
0x556988C: 898564FFFFFF MOV DWORD PTR [EBP-0x9C],EAX; VAR:0x9C
0x5569892: 8B8D64FFFFFF MOV ECX,DWORD PTR [EBP-0x9C]; VAR:0x9C
0x5569898: FF1584925005 CALL DWORD PTR [0x5509284]
; IL_0032: stloc.1

```

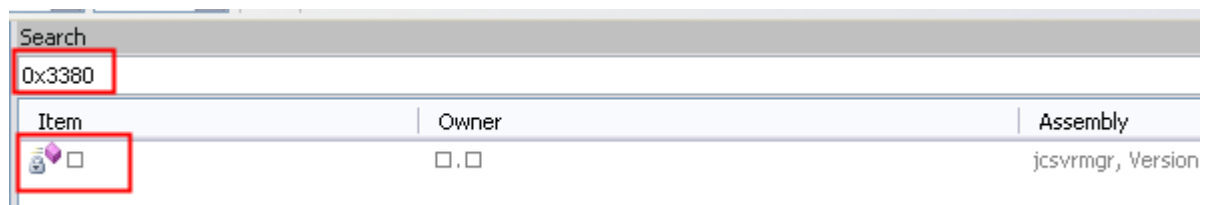
Y hasta aquí es lo que fui descubriendo. A ver si me pongo en descubrir cómo es la estructura que debe contener el archivo de licencia. Estructura que casi estoy seguro 100% que debe ser una estructura típica Xml. También me gustaría investigar esto:

	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F	0123456789ABCDEF
08B0	6B4C	673D	3D80	B857	5739	3163	6942	6D63	kLg==€,WU91ciBmc
08C0	6D56	6C49	4852	7961	5746	7349	4768	6863	mVlIHRyaWfsIGhhc
08D0	7942	6C65	4842	7063	6D56	6B4C	6941	6756	yBleHBpcmVklLiAgV
08E0	3239	3162	4751	6765	5739	3149	4778	7061	291bGQgeW91IGxpa
08F0	3255	6764	4738	6759	6D55	6763	6D56	6B61	2UgdG8gYmUgcmVka
0900	584A	6C59	3352	6C5A	4342	3062	7942	3061	XJlY3RlZCB0byBOa
0910	4755	6753	6D46	7459	584E	3049	4864	6C59	GUgSmFtYXNOIHdlY
0920	6E4E	7064	4755	6762	6D39	3349	475A	7663	nNpdGUgbm93IGZvc
0930	6942	7062	6D5A	7663	6D31	6864	476C	7662	iBpbmZvcmlhdGlvb
0940	6942	7662	6942	7764	584A	6A61	4746	7A61	iBvbiBwdXJjaGFza
0950	5735	6E49	4745	6763	3239	6D64	4864	6863	W5nIGEgc29mdHdhc
0960	6D55	6762	476C	6A5A	5735	7A5A	5438	3D14	mUgbGljZW5zZT8=.
0970	5648	4A70	5957	7767	5258	6877	6158	4A6C	VHJpYWwgRXhwaXJl
0980	5A41	3D3D	3461	4852	3063	446F	764C	3364	ZA==4aHR0cDovL3d

Cuando buscamos una cadena encriptada, por ejemplo esta que ya conocen de antes, se supone que el byte anterior a la cadena, el byte B8 que está en el desplazamiento 8B6, tendría que valernos para encontrar el integro en el programa que hace referencia a una cadena. Veran en la pagina 13, que tuve que utilizar el desplazamiento 8B5 para encontrar el integro que necesitaba. Vean los dos bytes precedentes a la cadena encriptada en si. Son los bytes 80 y B8. Ese byte 80 no se si es un marcador que pone el mismo protector. Se repite la misma secuencia de bytes en algunas zonas del archivo de recurso. Los bytes 80 y XX. Digo XX por que puede ser otro byte. Vean este ejemplo:

0380	455A	585A	7059	3255	674C	5341	3D10	526E	EZA4pyZUGLSA=.Rn
0390	4A70	5A57	356B	6248	6C4F	5957	316C	8080	JpZW5kbHl0YW1l€€
03A0	5647	686C	4947	526C	646D	6C6A	5A53	647A	VGhlIGRldm1jZSdz
03B0	4948	4279	6233	426C	636E	5270	5A58	4D67	IHBvb3BlcnRpZXMg
03C0	6147	4632	5A53	4269	5A57	5675	4947	3176	aGF2ZSBiZWVuIG1v
03D0	5A47	6C6D	6157	566B	4C69	4167	5632	3931	ZGlmaWVklLiAgV291
03E0	6247	5167	6557	3931	4947	7870	6132	5567	bGQgeW91IGxpa2Ug
03F0	6447	3867	6332	4632	5A53	4235	6233	5679	dG8gc2F2ZSB5b3Vy
0400	4947	4E6F	5957	356E	5A58	4D67	596D	566D	IGNoYW5nZXMGYmVm
0410	6233	4A6C	4947	4E73	6233	4E70	626D	632F	b3JlIGNsb3Npbmc/
0420	2452	4756	3261	574E	6C49	4642	7962	3342	\$RGV2aWNlIFByb3B
0430	6C63	6E52	705A	584D	6751	3268	6862	6D64	lcnRpZXMgQ2hhbmd
0440	6C5A	413D	3D08	5A33	4A70	5A41	3D3D	0C59	lZA==.Z3JpZA==.Y
0450	3231	6B51	5842	7762	486B	3D08	5158	4277	21kQXBwbHk=.QXBw

Se trata de otra cadena encriptada. La cadena en si empieza por "VGhlIGRldm..." y vean los dos bytes precedentes. Son los bytes 80 y 80. Si la secuencia de bytes es 80 XX, el primer byte de los precedentes es el que me sirve de puntero para averiguar el integro que necesito. El primer byte 80 está en el desplazamiento 39E. Bien, 39E+2FE2=3380. Si busco ese integro en Reflector:



Me encuentra desde dónde se utiliza ese integro. Le hago doble click al item resultante y me lleva allí directamente:

