

Neófito Reversando .NET [Entrega #3][LUISFECAB]



HERRAMIENTAS	Windows 7 Home Premium SP1 x32 Bits (SO donde trabajamos) dnSpy v6.0.5 DESCARGAR HERRAMIENTAS DESCARGAR TUTO+ARCHIVOS
AUTOR	LUISFECAB
RELEASE	Agosto 24 2019 [TUTORIAL 20]

INTRODUCCIÓN

Vamos a seguir analizando nuestro CrackMe <CrackMe#2.YouCan.exe> para ir aprendiendo nuevas cositas y formas para que las podemos aplicar en el futuro. Me hubiera gustado haber podido hacer todo el CrackMe en una sola Entrega pero se me hacía demasiado largo.

Quedamos en que ya habíamos logrado evitar al "CHICO MALO", cambiando el código con el **EDITOR IL** porque el propósito era abordar el **IL** o (**CIL**). Tenemos muchas cosas pendientes por hacer y terminar, como por ejemplo evitar ese otro "CHICO MALO" que nos apareció en el último momento, y vaya uno a saber si habrán más por ahí.

Haremos que nuestro **SERIAL** hueco "muydifícil" sea tomado como verdadero y con eso finalizaremos este escrito.

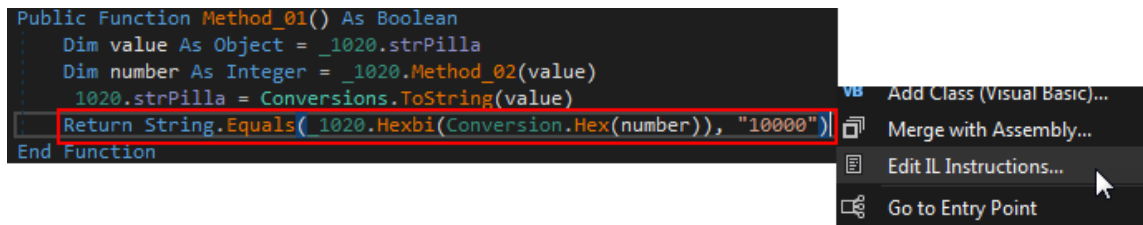
Saludos a mis amigos de **CracksLatinoS** y **PeruCrackerS**.

ENTREGA #3

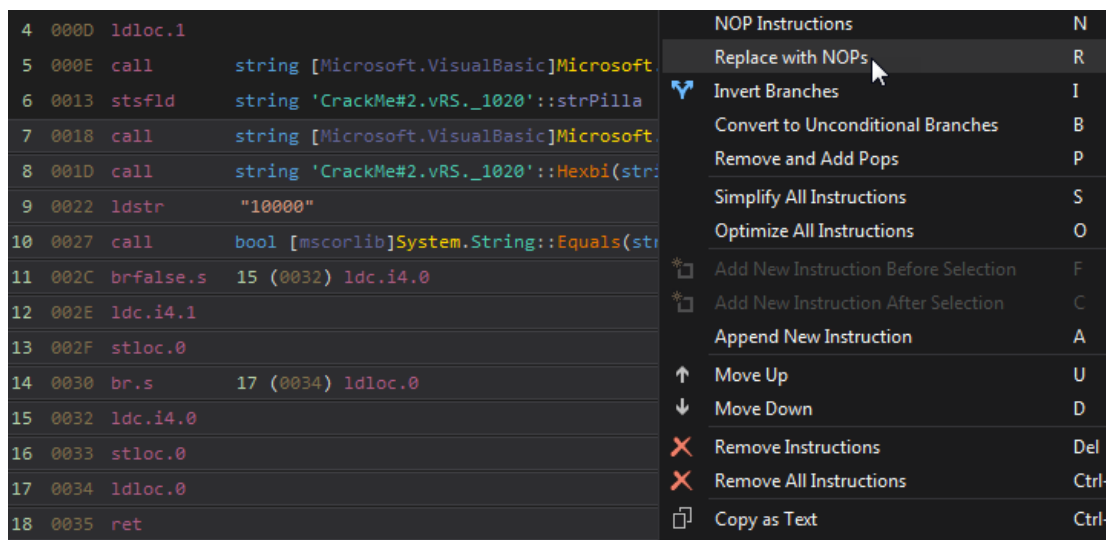
Miremos nuestro Método `_1020.Method_01()`, el cual editamos.

```
Public Function Method_01() As Boolean
    Return True
End Function
```

Recordemos que hicimos nuestro cambio al inicio del Método y que debido a eso todo el resto del código se desechó. En nuestro caso, ese código no era relevante para el correcto funcionamiento de nuestro CrackMe, pero y si no fuera ese el caso y al no ejecutarse algo fallara en la ejecución del programa. Entonces, nuestro cambio no debería hacerse al inicio, si no en el final de nuestro método, donde se encuentra nuestro retorno. Vamos a hacer ese cambio, suponiendo que el cambio en el inicio nos hace fallar el CrackMe. Carguemos nuestro CrackMe original en el <dnSpy>.

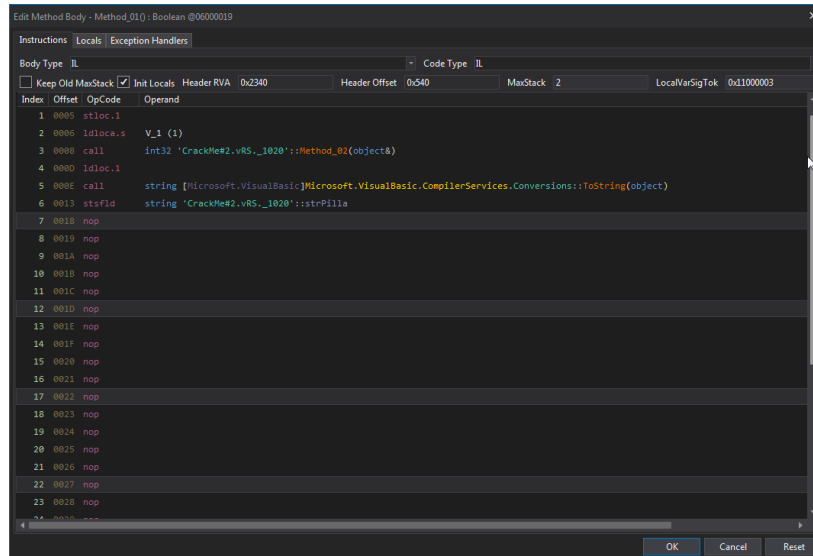


Como aprendimos, posicionados sobre la línea de código que queremos editar, en este caso la última, <Clic Derecho->Edit IL Instructions...> para abrir el **EDITOR IL** y tener todas las **INSTRUCCIONES** resaltadas que componen esa línea de código.

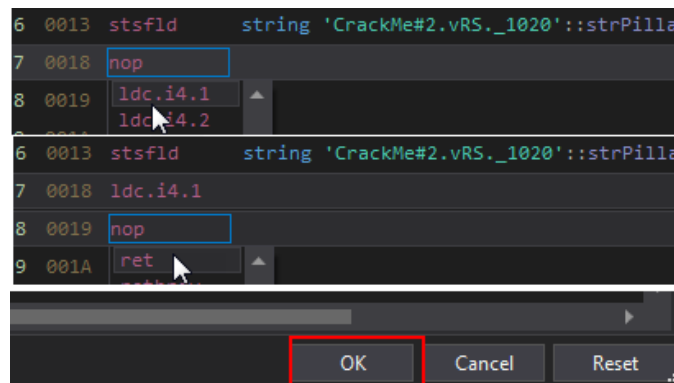


Lo que haremos es **NOPEAR** todas las **INSTRUCCIONES** que componen nuestra línea de código, las cuales están seleccionadas; para eso hacemos <Clic Derecho->Replace with NOPs> sobre cualquiera de estas para reemplazarlas todas de una.

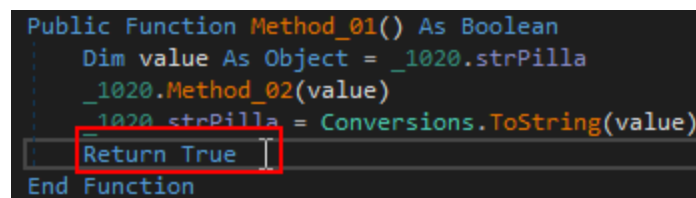
Neófito Reversando .NET [Entrega #3][LUISFECAB]



Como podemos ver todo quedó **NOPEADO**, sin nada de código. Paso seguido, agregamos nuestras dos **INSTRUCCIONES**, la que pone **1 (TRUE)** al **STACK**, **ldc.i4.1**, y nuestro retorno, **ret**.



Hemos puesto nuestras dos **INSTRUCCIONES** y compilamos nuestros cambios con "OK".



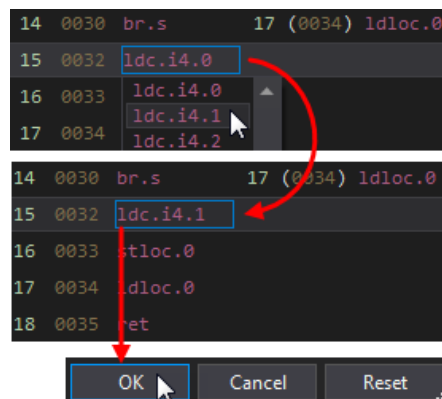
Hemos hecho nuestro cambio correctamente. Lo que nos restaría, es guardar los cambios <File->Save Module...> y probar si evitamos a ese primer "**CHICO MALO**". Yo voy a evitar repetir eso porque ya es hacer lo mismo que se hizo, pero te aconsejo lo hagas tu si te estás iniciando en Reversar .NET porque esa es la mejor manera de aprender. Espero no se me impacienten amigos, pero seguiremos en este mismo Método porque quiero mostrar más formas de obtener nuestro **1 (TRUE)**, ustedes saben, solo quiero que aprendamos que podemos hacer lo mismo de diferentes formas e ir mejorando nuestra lógica para lograr nuestro objetivo, Crackear nuestros **TARGETS**, ya saben, solo por propósitos educativos. Miremos nuestro **1020.Method_01()** original en la vista **IL**.

Neófito Reversando .NET [Entrega #3][LUISFECAB]

```
25      /* 0x00000578 2C04      */ IL_002C: brfalse.s IL_0032
26
27      /* 0x0000057A 17       */ IL_002E: ldc.i4.1
28      /* 0x0000057B 0A       */ IL_002F: stloc.0
29      /* 0x0000057C 2B02      */ IL_0030: br.s      IL_0034
30
31      /* 0x0000057E 16       */ IL_0032: ldc.i4.0
32      /* 0x0000057F 0A       */ IL_0033: stloc.0
33
34      /* 0x00000580 06       */ IL_0034: ldloc.0
35      /* 0x00000581 2A       */ IL_0035: ret
36  } // end of method 1020::Method 01
```

RECORDEMOS: PONEMOS 0 (FALSE). LO CAMBIAREMOS POR 1 (TRUE).

Esas son las últimas **INTRUCCIONES** que están relacionadas porque dependen del salto, **IL_002C: brfalse.s IL_0032**. Lo que haremos es que donde pasábamos 0 (**FALSE**) al **STACK** nosotros tengamos nuestro 1 (**TRUE**). Ya saben, nos posicionamos sobre **IL_0032: ldc.i4.0** y desde ahí entramos al **EDITOR IL** para hacer el cambio.

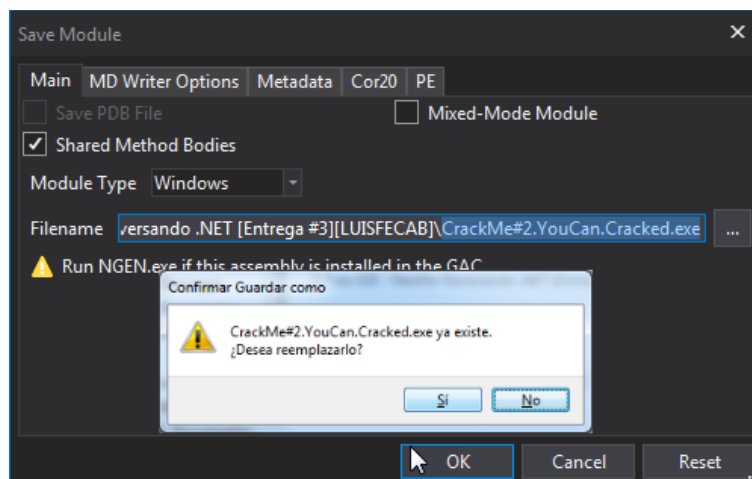


Ahora solo hicimos nuestro cambio por **ldc.i4.1**. Compilamos con el botón "OK", y vamos a ver cómo nos quedan los cambios en la vista de código **Visual Basic**.

```
1020.strPilla = Conversions.ToString(value) ORIGINAL
Return String.Equals( 1020.Hexbi(Conversion.Hex(number)), "10000")

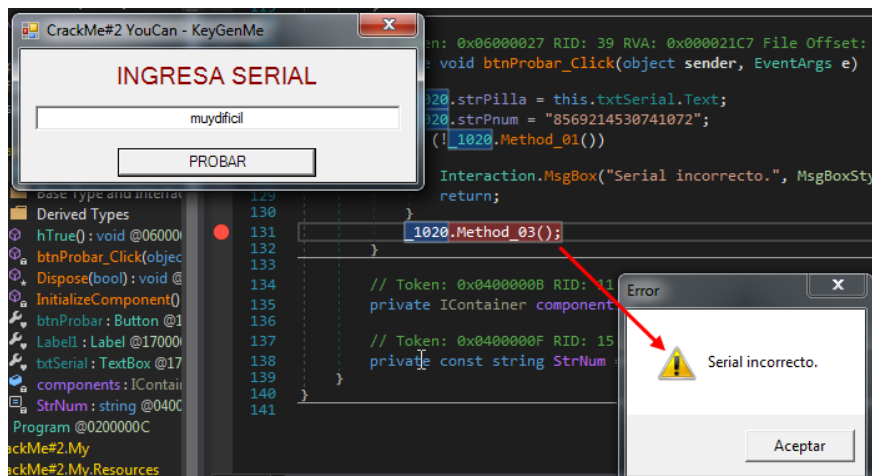
1020.strPilla = Conversions.ToString(value) PATCHED
Return Not String.Equals( 1020.Hexbi(Conversion.Hex(number)), "10000") OrElse True
```

Cambió un poco las condiciones pero no se eliminó nuestro código original, este cambio si lo voy a guardar, sobrescribo mi **<CrackMe#2.YouCan.Cracked.exe>**.

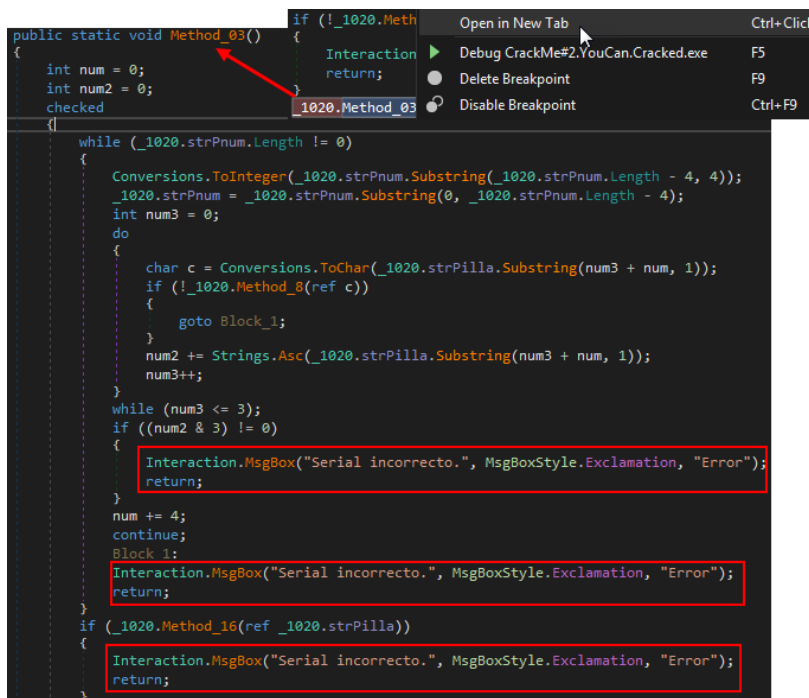


Neófito Reversando .NET [Entrega #3][LUISFECAB]

De aquí en adelante seguiremos analizando nuestro <CrackMe#2.YouCan.Cracked.exe> hasta que lo Crackiemos completamente para que nos reciba cualquier **SERIAL**. Una adivinanza para aquellos que me han leído desde mis inicios con mis tutoriales, ¿Cuál será el **SERIAL** que estaré usando yo y que deseo sea tomado como verdadero?



Ese Método `1020.Method_03()`, que en la **Entrega #2** fuimos a Tracear todos confiados con <F10> para pasarlo por encima y evitarnos la fatiga, nos terminó mostrando un nuevo "CHICO MALO" por el vecindario. Podríamos utilizar la técnica del **CALL STACK** para ubicarnos en él, pero mejor lo voy a abrir en otra pestaña para ver qué encontramos, <Click Derecho->Open in New Tab>.



¡Hay madre mía! Lo que hay es una pandilla conformada por "CHICOS MALOS" que debemos evitar y de paso unas rutinas de código que nos pueden confundir si apenas estamos empezando a entender el código .NET, pero esa fue la idea de codear el <CrackMe#2.YouCan.exe> de esa forma, que nos ofrezca rutinas y código nuevo para

Neófito Reversando .NET [Entrega #3][LUISFECAB]

analizarlo y aprender cosas nuevas; justo ahora estamos enfocados en aprender para hacerles cambios en código **IL**, y por último le sacaremos el **SERIAL**.

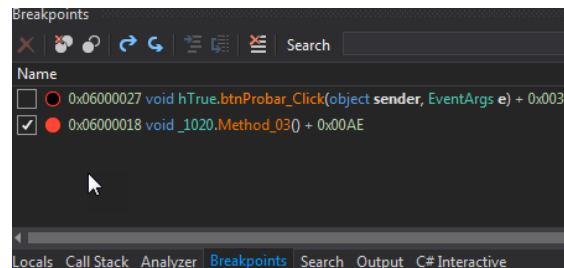
```
12      public static void Method_03()
13      {
14          int num = 0;
15          int num2 = 0;
16          checked
17          {
18              while (_1020.strPnum.Length != 0)
19              {
```

Estoy con la vista **C#**, pienso que es importante no quedarnos siempre con una vista de código de alto nivel, a mí me gusta siempre utilizar las tres, ya sea **C#**, **Visual Basic** o **IL** porque eso nos permite ir familiarizándonos con la estructura de estos lenguajes de programación. Como vemos, he puesto un **<BREAKPOINT>** en la línea 18 donde se inicia un **while (_1020.strPnum.Length != 0)**. Se repetirá ese bucle mientras la longitud **_1020.strPnum** sea diferente de cero. Uhhhhh! Me parece conocida, yo como que la he visto.

```
private void btnProbar_Click(object sender, EventArgs e)
{
    _1020.strPilla = this.txtSerial.Text;
    _1020.strPnum = "8569214530741072";
}
```

AQUÍ ES UTILIZADA

_1020.strPnum es utilizada en nuestro **btnProbar_Click**, así que entraremos a **while (_1020.strPnum.Length != 0)** porque no está vacía. Probemos nuestro **SERIAL** para detenernos en nuestro **<BREAKPOINT>**, aprovecho y quitaré los demás que tenía puestos.



Bien, ahora si en serio, pruebo mi **SERIAL** para llegar al **<BREAKPOINT>**.

```
18      while (_1020.strPnum.Length != 0)
19      {
20          Conversions.ToInteger(_1020.strPnum.Substring(_1020.strPnum.Length - 4, 4));
21          _1020.strPnum = _1020.strPnum.Substring(0, _1020.strPnum.Length - 4);
22          int num3 = 0;
23          do
24          {
25              char c = Conversions.ToChar(_1020.strPilla.Substring(num3 + num, 1));
```

Nos enfocaremos en Tracear con **<F10>** para evitarnos entrar a cualquier otro Método presente, no nos preocupemos por poder entender todo el código, solo seguiremos el Traceo y enfocarnos en la parte que nos banda al pandillero, "**CHICO MALO**". Así que, pínchele a **⇧** o dale sin afán a **<F10>** para no perdernos durante el Traceo.

Neófito Reversando .NET [Entrega #3][LUISFECAB]

```
23
24
25 char c = Conversions.ToChar(_1020.strPilla.Substring(num3 + num, 1));
26 if (!_1020.Method_8(ref c))
27 {
28     goto Block_1;
29 }
30 num2 += Strings.Asc(_1020.strPilla.Substring(num3 + num, 1));
31 num3++;
32 }
33 while (num3 <= 3);
```

Entramos a otro bucle que en este caso es un **do...while**, nada de qué preocuparnos, nosotros seguiremos con calma con nuestro Traceo.

```
39 num += 4;
40 continue;
41 Block 1:
42 Interaction.MessageBox("Serial incorrecto.", MsgBoxStyle.Exclamation, "Error");
43 return;
```

Joder!!!! Sin darme cuenta terminé por el mal camino, eso ocurrió cuando pasé por la línea 26 **if (!_1020.Method_8(ref c))**, que está apenas iniciando el **do...while**. Pues es otra comprobación **TRUE** o **FALSE**. Sigamos la ejecución, aceptemos el mensaje maluco, probamos de nuevo y lleguemos hasta ese **if**.

```
25 char c = Conversions.ToChar(_1020.strPilla.Substring(num3 + num, 1));
26 if (!_1020.Method_8(ref c))
27 {
28     goto Block_1;
29 }
```

El Método **_1020.Method_8(ref c)** nos debe retornar **TRUE** porque aquí debemos evitar tomar el **if**, de lo contrario tomamos el salto **goto Block_1** que nos lleva al "**CHICO MALO**". Algo de programación en **C#**, en **C#** utilizamos el **!** para especificar diferente de algo, es como el **Not** en **VB.NET**; miren las diferencias cambiando las vistas de código en el **<dnSpy>**. Pues nos tocó entrar a **_1020.Method_8** y obligarlo a retornar **TRUE**.

```
private static bool Method_8(ref char strChar)
{
    return Strings.Asc(strChar) > 64 & Strings.Asc(strChar) < 91;
}
```

Parece que algo hace con un carácter, y la verdad me importa muy poco; lo que quiero, es que mi **SERIAL** sea aceptado; así que forzaremos el retorno a **TRUE**, cosa que ya sabemos hacer. Recuerden, estamos es aprendiendo **IL (CIL)** y eso debemos hacer, nada de editarlo en código de alto nivel.

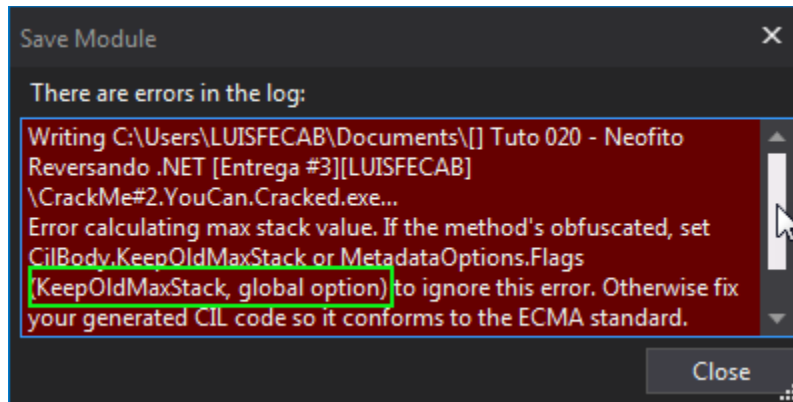
Index	Offset	OpCode	Operand
0	0000	ldc.i4.1	
1	0001	ret	
2	0002	call	int32 [Microsoft.VisualBasic]
3	0007	ldc.i4.s	0x40
4	0009	cgt	

OK

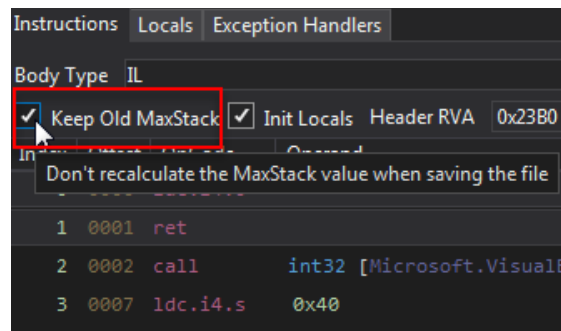
```
private static bool Method_8(ref char strChar)
{
    return true;
}
```


Neófito Reversando .NET [Entrega #3][LUISFECAB]

Como tenemos una sola línea de código, tendremos que todas las **INSTRUCCIONES IL** están relación entre si y por eso debemos agregar nuestros cambios al inicio, **ldc.i4.1** que es poner **1 (TRUE)** en el tope del **STACK** y **ret** para retornar con ese valor. Guardamos los cambios, sobrescribiendo el CrackMe Crackeado; no olviden detener la ejecución antes de guardar.



Me surgió algún tipo de error que no me permite guardar los cambios. Nos informa que hubo un error calculando el nuevo tamaño del stack para ese método, pero también por fortuna nos recomienda una posible solución, aunque se refiere como si tuviéramos nuestro Método ofuscado, que no es el caso pero no importa, vamos a probar hacer nuestros cambios seleccionando la opción **KeepOldMaxStack**. Debemos regresar a nuestro **EDITOR IL**.



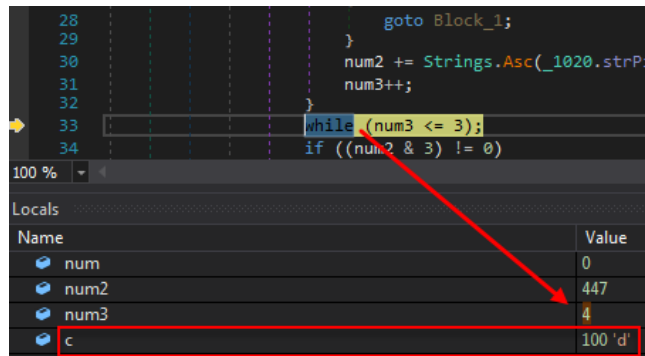
Debemos seleccionar la opción **"Keep Old MaxStack"**, luego compilamos con **"OK"**. Ahora sí, intento guardar mis cambios con **<File->Save Module...>**. Funcionó de maravilla. Me encanta que este tipo de problemas nos surjan sin previo aviso porque así aprendemos a solucionarlos, es más amigos, a mí nunca me había sucedido este inconveniente y que mejor que me ocurriera aquí en el tutorial **"Neófito Reversando .NET"**.

Bueno, probemos nuestros cambios y lleguemos a **if (!_1020.Method_8(ref c))** para ver si evitamos entrar al **if**, para ver por dónde nos lleva el camino.

```
25 char c = Conversions.ToChar(_1020.strPilla.Substring(num3 + num, 1));
26 if (!_1020.Method_8(ref c))
27 {
28     goto Block_1;
29 }
```

Neófito Reversando .NET [Entrega #3][LUISFECAB]

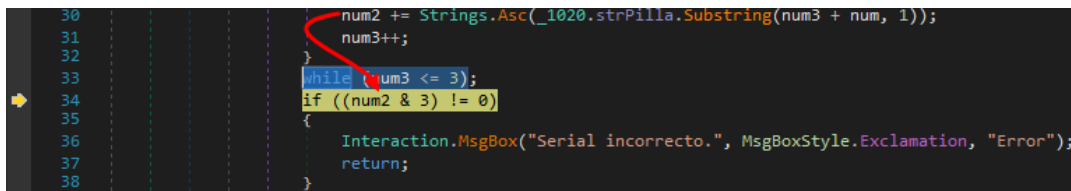
Listo, ya estoy parado en el **if**. Presionemos <F10> y esperemos que estemos en lo correcto y no entremos, y podamos seguir recorriendo el bucle **do...while**. Sin problemas nos evitamos el **if**.



```
28      goto Block_1;
29    }
30    num2 += Strings.Asc(_1020.strPi
31    num3++;
32  }
33  while (num3 <= 3);
34  if ((num2 & 3) != 0)
```

Name	Value
num	0
num2	447
num3	4
c	100 'd'

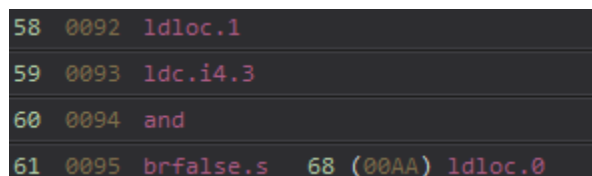
Perfecto, vamos por buen camino. Recorrimos el **do...while**, que lo más seguro algo hace con nuestro **SERIAL** y que lo obligamos a que todo fuera correcto. Si miramos en **LOCALS**, **num3=4**, eso quiere decir que yo ya recorrí el **do...while** y voy a salir de él porque ya no cumplo la condición **while (num3 <= 3)**. Con eso tenemos un pandillero menos.



```
30    num2 += Strings.Asc(_1020.strPilla.Substring(num3 + num, 1));
31    num3++;
32  }
33  while (num3 <= 3);
34  if ((num2 & 3) != 0)
35  {
36    Interaction.MsgBox("Serial incorrecto.", MsgBoxStyle.Exclamation, "Error");
37    return;
38  }
```

Otro **if ((num2 & 3) != 0)**. Este ya no depende del resultado que retorne algún Método, sino del resultado de una operación de Bits, es este caso un **AND** que en **C#.NET** se denota con **&**. Creo que debemos tener una base de conocimientos de operaciones entre Bits, como también poder trabajar con números Binarios y Hexadecimales. Creo que me faltó dejar eso claro en el inicio de estas entregas. Si crees que te falta conocer sobre ese tema te aconsejo busques información por Internet, por fortuna se encuentra en gran cantidad y además es una teoría relativamente sencilla que en menos de un día puedes abarcar.

Miremos posibles soluciones, **num2 += Strings.Asc(_1020.strPilla.Substring(num3 + num, 1))** es un número que se origina en **do...while**. Podríamos poner a **num2** a que siempre nos retorne un número que al hacer el **AND** nos dé como resultado **0** para evitar entrar al **if** y librarnos de ese otro pandillero, pero bueno, y si yo no sé nada de operaciones de Bits, será que puedo buscar otra solución. Por supuesto, podemos ponerle al **if** directamente un **FALSE** (**if (false)**) para que nunca entre, y es por esa opción que me decanto yo también. Voy al **EDITOR IL**.



```
58 0092 ldloc.1
59 0093 ldc.i4.3
60 0094 and
61 0095 brfalse.s 68 (00AA) ldloc.0
```

Neófito Reversando .NET [Entrega #3][LUISFECAB]

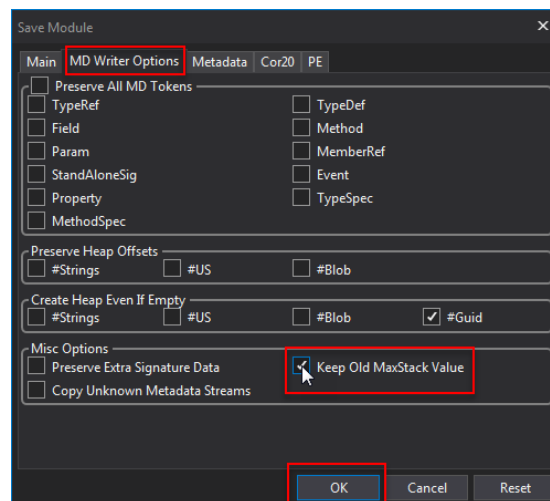
Nuestro `if ((num2 & 3) != 0)` está compuesto por cuatro **INSTRUCCIONES**. `ldloc.1` que carga nuestra variable `num2`, `ldc.i4.3` carga el número en el **STACK**, **and** calcula con nuestros dos valores previos y el `brfalse.s` que salta o no dependiendo del resultado de **and**. Recordemos cuándo salta `brfalse.s`, cuando sea **FALSE**, nulo o cero. Entonces ponerle su **0** como referencia a `brfalse.s.`

```
58 0092 ldc.i4.0
59 0093 nop
60 0094 nop
61 0095 brfalse.s 68 (00AA) ldloc.0
62 0097 ldstr "Serial incorrecto."
OK
```

Hemos cambiado la primera **INSTRUCCIÓN** `ldloc.1` por nuestro **0** (`ldc.i4.0`) y **NOPEAMOS** las otras dos para que no interfieran con `brfalse.s`.

```
34 | if (false)
35 | {
36 |     Interaction.MsgBox("Serial incorrecto.", MsgBoxStyle.Exclamation, "Error");
37 |     return;
38 | }
```

Parece que con ese cambio nos evitamos entrar al `if` y con eso al **"CHICO MALO"**. Lo de siempre, a guardar los cambios, iniciar todo y llegar a esa comprobación.



Debemos guardar con la opción **"Keep Old MaxStack Value"** seleccionada, porque de lo contrario tenemos el error de hace rato pero lo haremos desde la opciones de **<File->Save Module...>**. Diferentes formas para lo mismo.

```
34 | if (false)
35 | {
36 |     Interaction.MsgBox("Serial incorrecto.", MsgBoxStyle.Exclamation, "Error");
37 |     return;
38 | }
39 | num += 4;
40 | continue;
```

Ya nos encontramos detenidos en el `if`. Como venimos ya con varios cambios del mismo tipo me siento muy seguro de que a este pandillero lo burlamos sin problemas. Tenga pandillero su **<F10>**.

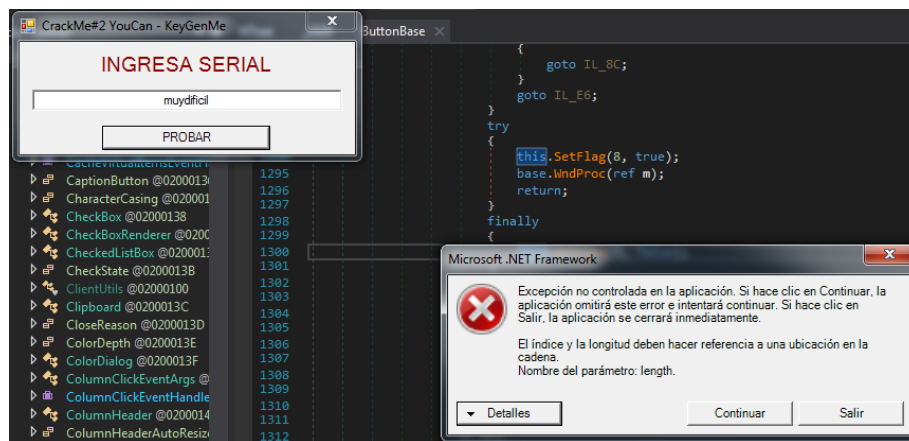
Neófito Reversando .NET [Entrega #3][LUISFECAB]

```
34         if (false)
35         {
36             Interaction.MsgBox("Serial incorrecto.", MsgBoxStyle.Exclamation, "Error");
37             return;
38         }
39         num += 4;
40         continue;
```

Listo, otra comprobación que vencemos. Tenemos un `continue` que nos retorna al inicio del `while (_1020.strPnum.Length != 0)`.

```
18         while (_1020.strPnum.Length != 0)
19         {
20             Conversions.ToInteger(_1020.strPnum.Substring(_1020.strPnum.Length - 4, 4));
21             _1020.strPnum = _1020.strPnum.Substring(0, _1020.strPnum.Length - 4);
22             int num3 = 0;
```

Efectivamente hemos empezado desde el inicio, solo nos queda seguir Traceando con **<F10>** para ver qué sucede.



Linda Excepción nos resultó, esos nos pasa por querer obligar a este CrackMe a recibir nuestro **SERIAL** a la fuerza. Muy cierto lo que dijo mi amigo **PEX** que este CrackMe no es tan sencillo para aquellos que se están iniciando desde cero, pero mejor así porque nos está permitiendo abarcar y aprender mucho más de lo esperado.

El Mensaje de la Excepción trae mucha información para entender en dónde ocurrió esta.

```
23         do
24         {
25             char c = Conversions.ToChar(_1020.strPilla.Substring(num3 + num, 1));
26             if (!_1020.Method_8(ref c))
27             {
28                 goto Block_1;
```

Traceando llegamos a `char c = Conversions.ToChar(_1020.strPilla.Substring(num3 + num, 1))` que es donde se nos origina la Excepción. Esta Excepción se origina porque se llega a un punto en que no tenemos carácter que retornar para `char c`. Pienso que hice un CrackMe no difícil para sacarle el **SERIAL** para los que saben, pero para obligarlo a aceptar cualquier **SERIAL** tenemos que hacerle más de lo que pensaba inicialmente.

La Excepción se origina por los cambios que hemos hecho, ya que antes se comprueba que el **SERIAL** cumpla unos requisitos mínimos para poder llegar a este punto y nosotros lo metimos a la fuerza, y ahí tenemos la consecuencia. La solución más

Neófito Reversando .NET [Entrega #3][LUISFECAB]

sencilla que se me ocurre es editar esa línea de código para que **char c** pase a tener un único carácter. Miremos esa línea de código en el **EDITOR IL**.

```
24 0044 ldsfld string 'CrackMe#2.vRS._1020'::strPilla
25 0049 ldloc.2
26 004A ldloc.0
27 004B add.ovf
28 004C ldc.i4.1
29 004D callvirt instance string [mscorlib]System.String::Substring(int32, int32)
30 0052 call char [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.Conversions::ToChar(string)
31 0057 stloc.3
```

Debemos **NOPEAR** desde el inicio hasta **callvirt instance string [mscorlib]System.String::Substring(int32, int32)** que sería en Método que sacaría nuestro carácter.

```
24 0044 nop
25 0045 nop
26 0046 nop
27 0047 nop
28 0048 nop
29 0049 nop
30 004A call char [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.Conversions::ToChar(string)
31 004F stloc.3
```

NOPEAMOS las **INSTRUCCIONES** que no se necesitan y dejamos las que van a recibir nuestro carácter, que es una String para convertirla en Char. Debemos poner nuestro carácter, yo pondré la "a".

```
29 0049 ldstr "a"
30 004E call char [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.Conversions::ToChar(string)
31 0053 stloc.3
```

Para agregar una String en código **IL** utilizamos la **INSTRUCCIÓN ldstr**, que tiene como **OPCODE** el **0x72**. Al poner esa **INSTRUCCIÓN** se nos habilita la opción de agregar la String que queramos. Compilamos con "OK" para ver los cambios.

```
16 char c = Conversions.ToChar("a");
17 if (!_1020.Method_8(ref c))

42 /* 0x00000480 00 */ IL_0044: nop
43 /* 0x00000481 00 */ IL_0045: nop
44 /* 0x00000482 00 */ IL_0046: nop
45 /* 0x00000483 00 */ IL_0047: nop
46 /* 0x00000484 00 */ IL_0048: nop
47 /* 0x00000485 72???????? */ IL_0049: ldstr "a"
48 /* 0x0000048A 283600000A */ IL_004E: call char [Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.Conversions::ToChar(string)
```

Los cambios en la vista **C#** parecen que están bien pero como estamos enfocados en **IL**, ponemos esa vista y vemos las **INSTRUCCIONES** que **NOPEAMOS** que tienen su **nop**, eso parece estar bien pero si miramos nuestra **INSTRUCCIÓN ldstr**, en los **BYTES** tenemos **72????????**, solo aparece nuestro **OPCODE 0x72** y luego **????????**. Recordemos que antes teníamos un **callvirt**.

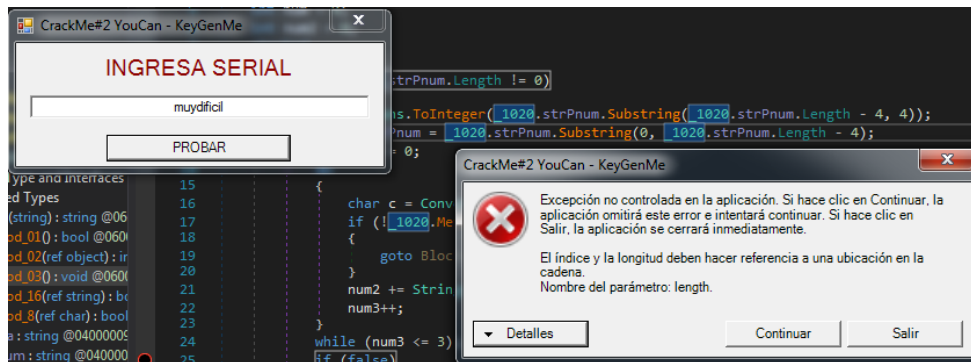
Neófito Reversando .NET [Entrega #3][LUISFECAB]

```
6F340000A */ IL_004D: callvirt instance string [mscorlib]System.String::Substring(int32, int32)
```

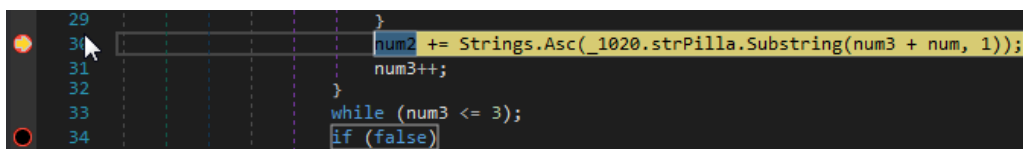
Resulta que en la **INSTRUCCIÓN** **callvirt** tenemos un **TOKEN** y al cambiarla por nuestra **ldstr**, esta requiere un nuevo **TOKEN** que hará referencia donde se guarda nuestra nueva String y como no hemos guardado nuestros cambios entonces esos **BYTES** no se conocen y es por eso que tenemos esos **????????**. Guardemos nuestros cambios, recordando activar el "Keep Old MaxStack Value".

```
/* 0x00000485 724B000070 */ IL_0049: ldstr      "a"  
/* 0x0000048A 283600000A */ IL_004E: call      char [Microsoft.VisualBasic]  
Microsoft.VisualBasic.CompilerServices.Conversions::ToChar(string)
```

Ahora si vemos los **BYTES** que son el **TOKEN** donde está nuestra String "a". Cositas que pueden ser irrelevantes pero a mi gusta saber el porqué es que suceden. Retomemos la prueba de nuestro **SERIAL**.



¡Ay Dios! De nuevo la Excepción. Y yo que probé el **SERIAL** confiado sin tener ningún **<BREAKPOINT>** y sin Tracearlo. Bueno, nos tocó intentarlo de nuevo pero Traceando con **<F10>** en el Método **_1020.Method_03()** para ver si falla en el mismo lugar o es otro.



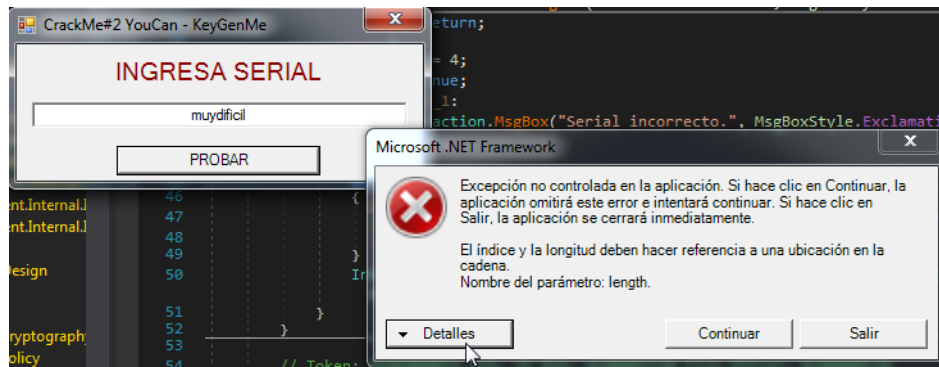
Ahora la Excepción ocurre en **num2 += Strings.Asc(_1020.strPilla.Substring(num3 + num, 1))**. Se debe a lo mismo, que nuestro **SERIAL** **strPilla** no cumple el requisito para que **Substring(num3 + num, 1)** se ejecute correctamente. Recordemos que habíamos planteado cambiar la **num2** con un valor constante para cuando hiciéramos el **AND (if ((num2 & 3) != 0))** obtuviéramos un valor tal que evitáramos tomar el **if**, pero nosotros terminamos fue haciendo este cambio, **if (false)**. Amigos, no nos queda de otra que cambiar para que sume **num2** un valor constante y evitar esa Excepción quitando todo ese código **Strings.Asc(_1020.strPilla.Substring(num3 + num, 1))**. Haremos lo mismo que en la otra Excepción pero aquí agregaremos un número ya que **num2** es una variable que almacena un número entero. Escogemos esa línea de código y desde ahí abrimos nuestro **EDITOR IL**.

```
41 0070 ldloc.1 CARGAMOS VARIABLE num2
42 0071 nop
43 0072 nop
44 0073 nop
45 0074 nop
46 0075 ldc.i4.8 AGREGAMOS 8
47 0076 nop
48 0077 nop
49 0078 add.ovf SUMAMOS num2+8
50 0079 stloc.1 GUARDAMOS num2 STACK
```

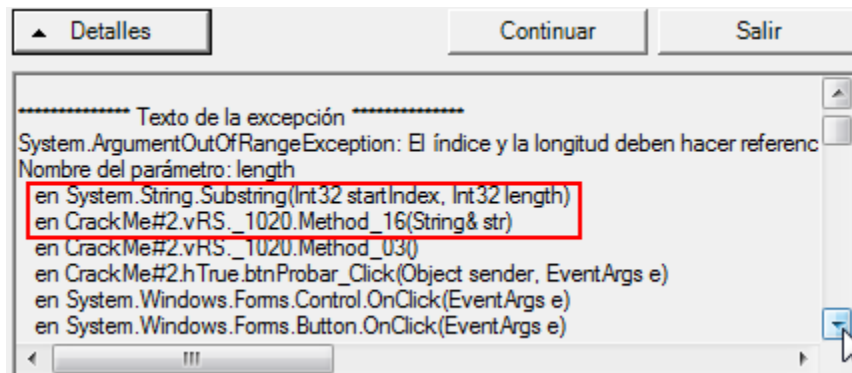
```
30      num2 += 8;
31      num3++;
32  }
```

```
45         if (_1020.Method_16(ref _1020.strPilla))
46         {
47             Interaction.MsgBox("Serial incorrecto.", MsgBoxStyle.Exclamation, "Error");
48             return;
49         }
```


Neófito Reversando .NET [Entrega #3][LUISFECAB]



Esto es una pesadilla. Sigo sufriendo por más Excepciones. ¿Y ésta de dónde saldrá? Lo más seguro que del Método `1020.Method_16(ref_1020.strPilla)` porque es lo que se ejecutaba. Miremos los detalles de la Excepción para confirmarlo.



Efectivamente ocurre en el `1020.Method_16(ref_1020.strPilla)` y nos dice que es cuando va a hacer un `System.String.Substring(Int32 startIndex, Int32 length)`. Miremos dentro de ese Método para ver qué hace.

```
private static bool Method_16(ref string str)
{
    int num = 0;
    checked
    {
        for (;;)
        {
            string value = str.Substring(0, 4);
            str = str.Substring(4, str.Length - 4);
            if (str.Contains(value))
            {
                break;
            }
            num += 4;
            if (num > 8)
            {
                goto Block_2;
            }
        }
        return true;
    }
    Block_2:
    return false;
}
```

En alguno de esos dos `str.Substring` se debe generar nuestra Excepción. No vamos a cambiar nada para evitar la Excepción, si no que haremos que nos retorne **FALSE** en el inicio y con eso no evitamos qué lidiar con ese código. Vamos al **EDITOR IL** y hacemos nuestros cambios.

Neófito Reversando .NET [Entrega #3][LUISFECAB]

```
0 0000 ldc.i4.0
1 0001 ret

private static bool Method_16(ref string str)
{
    return false;
}
```

Hemos hecho nuestro cambio. Guardamos los cambios con **<File->Save Module...>**. Ahora si me siento confiado que evitamos al último pandillero y por eso pruebo mi CrackMe crackeado solito.



Por fin, casi que no. Hemos logrado que aceptara nuestro **SERIAL**. Lo hemos destrozado tanto por dentro que hasta muestra al "**CHICO BUENO**" sin ningún **SERIAL**.

PARA TERMINAR

Hemos terminado la primera parte del análisis del <CrackMe#2.YouCan.exe> que era aprender a hacer cambios en .NET. Cuando tengamos una aplicación que requiera cambiarle algo ya con lo aprendido aquí lo podemos hacer. Sin querer queriendo vimos cómo lidiar con Excepciones, con errores al tratar de guardar cambios y cómo solucionarlos.

Amigos, nos queda pendiente analizarlo para entender su código y ver que hace para validar el **SERIAL** y desde ahí hacer el KeyGen.

Muy contento porque este CrackMe lo resolvió **By KarMa [954] ©Full** y con eso me basta para saber que el esfuerzo está valiendo la pena.

Me despido de todos y nos leeremos Dios mediante en la **Entrega #4**.

@LUISFECAB