

Funcionamiento de .NETSHRINK, un poco de reflexión

by VortiCe

Introducción.

Hace tiempo, Eddy publicó un tutorial de .net que trataba de obtener un programa comprimido de la memoria (dump de Olly) posteriormente NCR, posteó una herramienta que comprime .NET entre otras cosas .netshrink.

Entonces, decidí ir a la pagina para bajarlo y probarlo. Ya de paso, estudiar un poco como funciona este compresor, y curiosear un poco...

NOTA: Igual esto está obsoleto, pero, es una de las cosas que comencé hace tiempo y que como no tengo mucho pues me ha llevado un horror terminar de escribir esta "Cosa". Siento que pasase tanto tiempo Lo posteo por si a alguien le pueda interesar...

Si voy a la página web: <http://www.pelock.com/>
Podemos ver esto:

.netshrink is an executable compressor for managed files. It uses LZMA compression library and can decrease your file size by 50%. It can also protect your files with a password.

.netshrink uses LZMA compression library to achieve maximum compression ratios. Password protection uses verification based on SHA256 hash function and 256 bit AES / Rijndael encryption.

What's supported:

- compression of .NET executables only
- password protection
- command line options

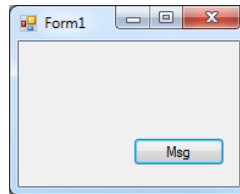
Tiene Buena pinta así que nos pondremos manos a la obra...

Antes de comenzar.

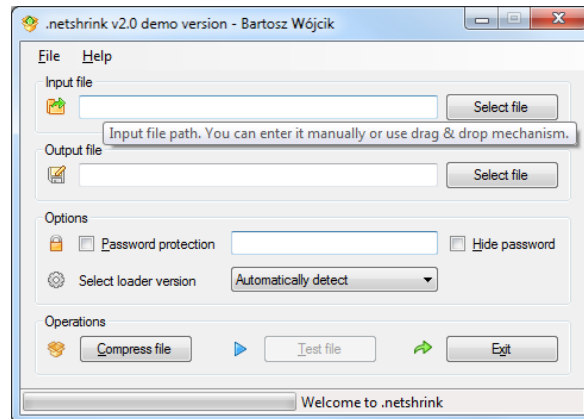
Quiero dejar claro que en este tutorial no se va a crackear nada, es un estudio del compresor. No me hago responsable del uso que se le pueda dar a este documento, está escrito solo con fines didácticos.

Utilizando la herramienta.



Una vez instalada, he creado una aplicación en .NET Framework 3.5 (Hola mundo) con un Formulario y un botón que muestra un MessageBox.



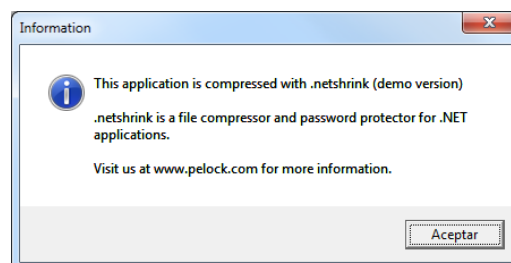
Algo sencillo, para nuestros fines, suficiente. Arrancamos nuestra aplicación que tiene este aspecto:



También reconozco que no es nada compleja al contrario es muy sencilla. Fichero de entrada, fichero de salida, si deseamos con clave o sin ella y comprimir el fichero. En nuestra primera prueba, lo haremos sin protección. Y obtenemos el siguiente resultado

| | | | | |
|---|-------------------|------------------|------------|-------|
|  | prueba.exe | 01/12/2010 9:10 | Aplicación | 8 KB |
|  | prueba-packed.exe | 01/12/2010 13:38 | Aplicación | 26 KB |

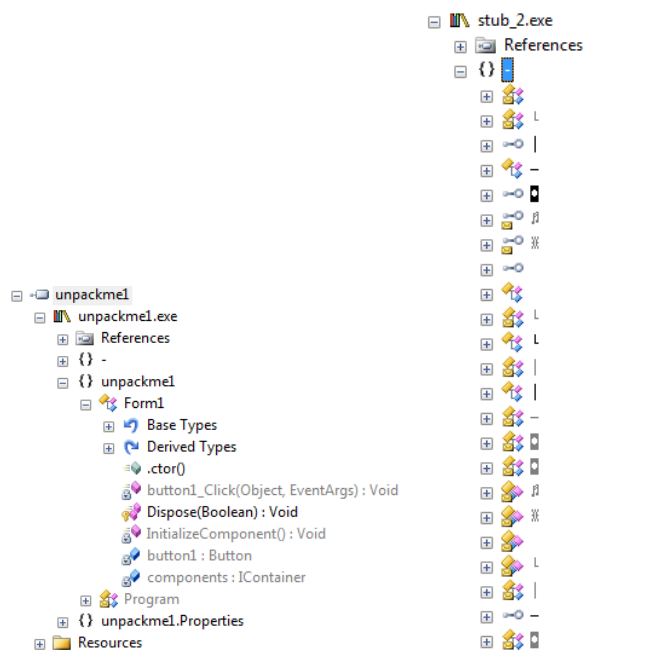
Supongo que comprimir 8 Kb no es posible y entre la compresión y unas cosas y otras, ocupa más que el original. Si le doy dos clics, me abre una nag diciendo que esta aplicación está comprimida con una demostración del netshrink.



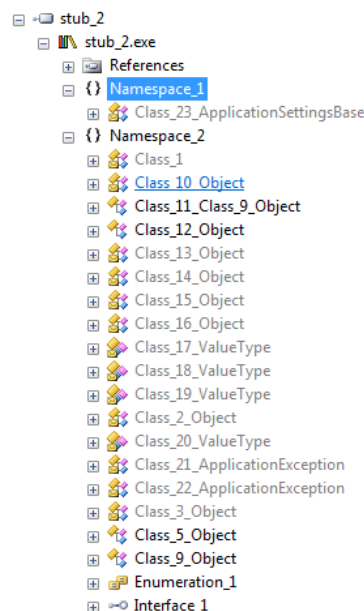
Si pulsamos sobre aceptar, aparece nuestra aplicación correctamente. Y podemos ejecutarla como siempre.

Que sucedió

Abramos la aplicación en reflector a ver qué ha sucedido. Al abrirlo, tenemos un antes y un después que vemos a continuación:



Supongo que no será necesario decir cual es antes y cual es después... En realidad me llamo mucho la atención ver tantas “Clases”. Se supone que está comprimido, no he leído nada de ofuscación del código ni nada por el estilo, sin embargo, veo que hay mogollón de cosas nuevas... Aquí hacemos un pequeño inciso. He usado el DeOfuscator 0.5 para intentar tener una visión de todo esto, después de deofuscarlo la aplicación ha dejado de funcionar, pero si lo abrimos nuevamente con reflector ya tiene mejor pinta...



La pregunta que muchos estaréis haciendo posiblemente es ¿Qué estoy haciendo? Pues como dije en la introducción curiosear y ver cómo funciona este compresor. Después de mirar un par de minutos entre clases y funciones, hay una en especial que me llama la atención.

```

internal sealed class Class_10_Object
{
    [STAThread]
    private static void Procedure_1(string[]  )
    {
        MemoryStream stream = new MemoryStream();
        Stream stream2 = new FileStream(Application.ExecutablePath.Substring(Application.ExecutablePath.LastIndexOf("\\") + 1), FileMode.Open, FileAccess.Read);
        stream2.Seek(-8L, SeekOrigin.End);
        long num = 0L;
        for (int i = 0; i < 8; i++)
        {
            int num3 = stream2.ReadByte();
            num |= ((byte) num3) << (8 * i);
        }
        stream2.Seek(-num - 8L, SeekOrigin.Current);
        byte[] buffer = new byte[5];
        stream2.Read(buffer, 0, 5);
        Class_5_Object obj2 = new Class_5_Object();
        obj2.Procedure_6(buffer);
        long num4 = 0L;
        for (int j = 0; j < 8; j++)
        {
            int num6 = stream2.ReadByte();
            num4 |= ((byte) num6) << (8 * j);
        }
        obj2.Procedure_5(stream2, stream, num, num4, null);
        stream2.Close();
        Assembly assembly = Assembly.Load(stream.ToArray());
        if (assembly != null)
        {
            string text = Class_13_Object.Function_String_1(-1411270442);
            string caption = Class_13_Object.Function_String_1(-1411270647);
            MessageBox.Show(text, caption, MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
            if (assembly.EntryPoint.GetParameters().Length == 0)
            {
                assembly.EntryPoint.Invoke(null, null);
            }
            else
            {
                assembly.EntryPoint.Invoke(null, new object[] {   });
            }
        }
    }
}

```

No voy a dar clases de programación, pero, en esta función se carga nuestra aplicación en un Stream y después de varias operaciones llama a una función a la que le pasa un MemoryStream y un Stream (nuestra aplicación). Posiblemente, estemos ante las funciones propias que desempaquetan nuestra aplicación. Además si nos fijamos bien, tenemos un `MessageBox.Show...` y posteriormente la función `assembly.EntryPoint.Invoke`, la cual nos permitirá llamar a un ensamblado en memoria. Bueno la cosa está clara, partimos como premisa, que se desempaqueta y lo guarda en memoria en ese MemoryStream. Vamos a desempaquetar esto.

(Mentí con el tema de las clases de programación :-)

El tema principal, es que utiliza el espacio de nombres de .net llamado `system.reflection` (reflexión)

a través de la reflexión, se pueden acceder a métodos, propiedades o cualquier cosa que queramos de un ensamblado. en este caso estamos haciendo una llamada al punto de entrada del ensamblado (entrypoint) que llamará a nuestro programa principal. En el anexo I (siiii, he creado un anexo) pongo una tontería de un ejemplo de reflexión, en ese ejemplo, llamo a través de reflexión a un método de una clase (como dije, una tontería, pero algo básico para entender que hace esto que es el fin de esta cosa (que no tutorial)).

Desempaquetando el paquete.

Esto no es nada nuevo, ni diferente ya que Eddy en un tutorial (no recuerdo cual (Lo siento Eddy tengo mala memoria)) lo explico.

Volviendo al tema, si, cargamos el formulario en Olly, nos vamos a Memory Map y buscamos Assembly Version. En mi caso han aparecido unos pocos, pero, he podido identificar el que me interesa de dos formas, El nombre del formulario principal que aparece y porque si vemos el trozo del dump inicial disponemos del Mz que es la cabecera.

¿Qué quiero enseñar con esto?

Pues igual que se hace un `setvalue` a una propiedad `X` de una clase, se pueden llamar a métodos, si resulta que ese método es el punto inicial del programa, puedo llamar a un programa dentro de otro. ¿Para qué? bueno, ahí pueden existir múltiples razones, compresión, encriptación razones víricas, tu pones la imaginación...

Recuerdo, que quería escribir mas pero, no recuerdo de que.

Por lo tanto hasta aquí llega esta cosa que no quiero ni llamar tutorial si no pérdida de tiempo.

Saludos a todos los que la hayan leído :-)