



# Ingeniería Inversa Práctica - Cómo Crear Exploits en Entornos .NET



**Julio Ureña**

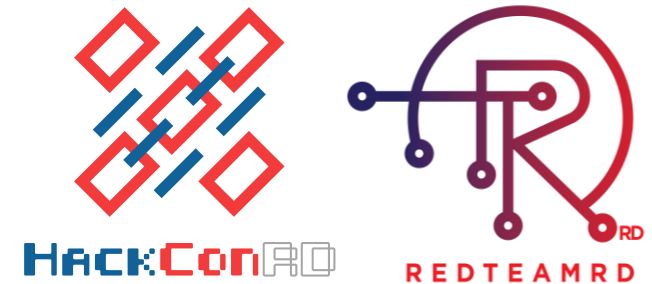
Líder RedTeamRD  
juliourena@plaintext.do



# PS> Julio Ureña

- ❑ Hacker: PlainText o "**El Hacker Cristiano**"
- ❑ ~15 años de experiencia en roles técnicos y ejecutivos
- ❑ Fundador y CTO en **PlainText Cybersecurity Solutions**
- ❑ Fundador y Líder del **RedTeamRD & HackConRD**
- ❑ Algunas certificaciones de ciberseguridad

- ❑ Twitter / LinkedIn / Instagram: @JulioUrena
- ❑ Email: juliourena@plaintext.do
- ❑ YouTube: <https://www.youtube.com/c/JulioUreña>

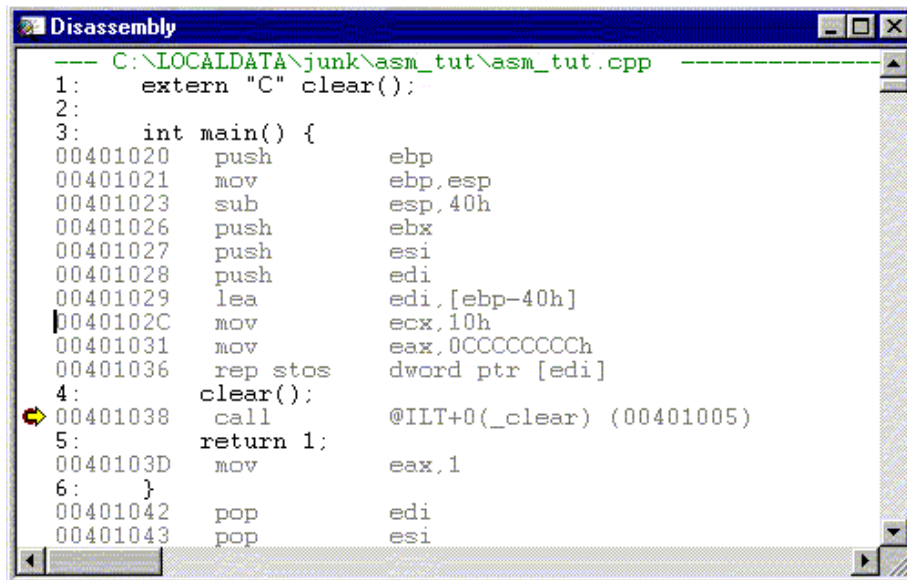


# Ingeniería Inversa



# Ingeniería Inversa

La ingeniería inversa es el proceso de desensamblar, analizar, y comprender cómo funciona un sistema, software, o dispositivo con el fin de reproducir, mejorar, o descubrir su diseño, funcionamiento interno, o código fuente.

A screenshot of a disassembler window titled 'Disassembly'. It shows the assembly code for a C program named 'asm\_tut.cpp'. The code includes a 'main' function that pushes several registers onto the stack, calls a 'clear' function, and then returns 1. The assembly instructions are listed with their corresponding memory addresses and hex values.

```
--- C:\LOCALDATA\junk\asm_tut\asm_tut.cpp ---
1:  extern "C" clear();
2:
3:  int main() {
00401020  push     ebp
00401021  mov      ebp,esp
00401023  sub      esp,40h
00401026  push     ebx
00401027  push     esi
00401028  push     edi
00401029  lea      edi,[ebp-40h]
0040102C  mov      ecx,10h
00401031  mov      eax,0CCCCCCCCh
00401036  rep stos dword ptr [edi]
4:  clear();
00401038  call     @ILT+0(_clear) (00401005)
5:  return 1;
0040103D  mov      eax,1
6:  }
00401042  pop      edi
00401043  pop      esi
```





# Ingeniería **Inversa** - dnSpy

**dnSpy** es una herramienta de código abierto que permite descompilar, depurar, y editar aplicaciones **.NET**. Es ampliamente utilizada para la ingeniería inversa, permitiendo a los usuarios explorar el código fuente y modificar aplicaciones en tiempo de ejecución.

Debido a la forma en que el framework .NET funciona y cómo genera el código binario cuando escribes y compilas una aplicación en .NET (C#, VB.NET, etc.), el código fuente no se convierte directamente en código de máquina nativo como ocurre en lenguajes como C o C++. En su lugar, se convierte en un lenguaje intermedio llamado **Intermediate Language (IL)**. El código IL es independiente de la plataforma y se compila **"Just-In-Time" (JIT)** en código nativo en el momento de la ejecución.

# Análisis de Código Fuente

El **análisis de código** fuente en el ámbito de la seguridad es una técnica utilizada para revisar el código de una aplicación en busca de **vulnerabilidades** que puedan ser explotadas por atacantes. Este análisis es esencial para asegurar que el software sea resistente a ataques y cumpla con los estándares de seguridad.

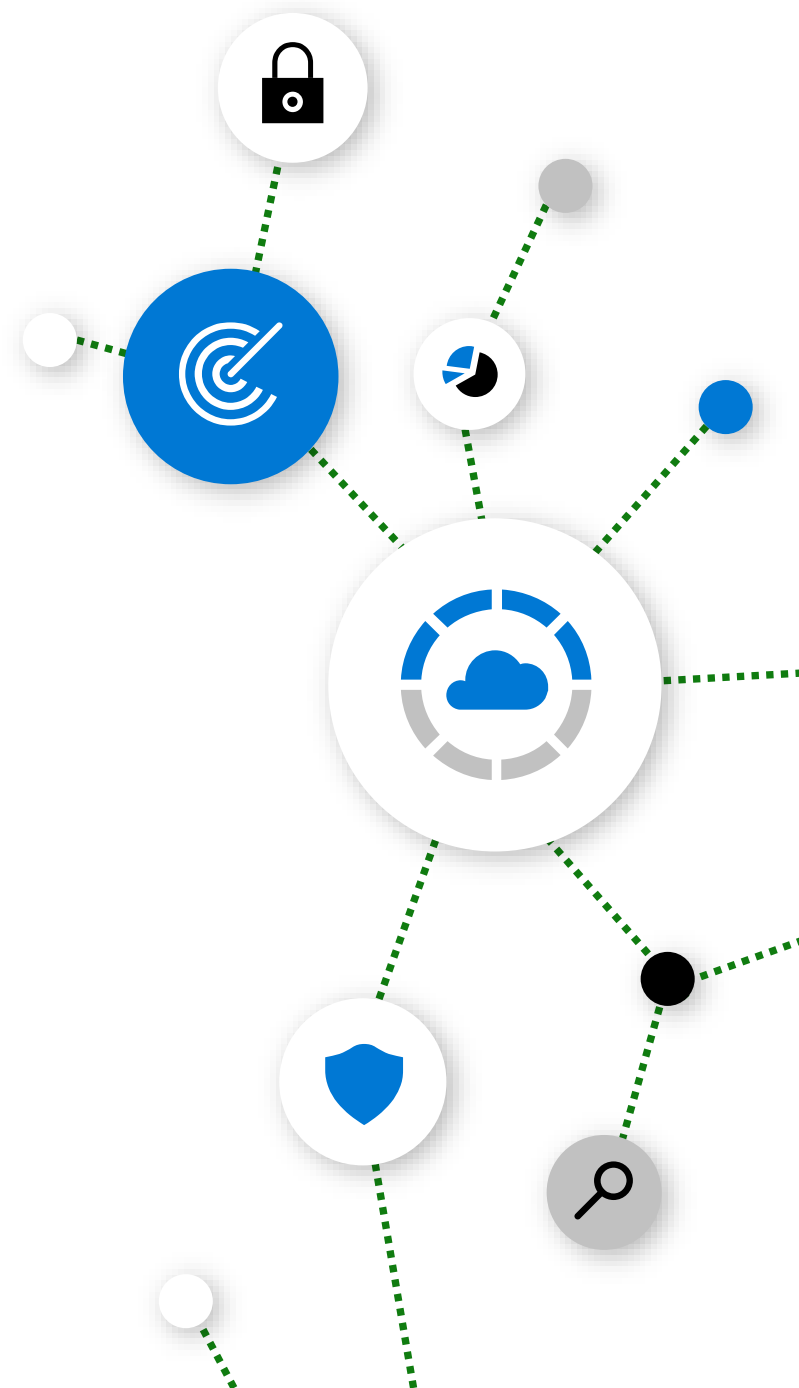
**Herramientas:** **Graudit** es una herramienta para el análisis de código fuente. Permite detectar rápidamente componentes del código que podrían ser vulnerables a abusos, como consultas SQL o métodos que podrían ser explotados por atacantes.





# Graudit

<https://github.com/wireghoul/graudit/>



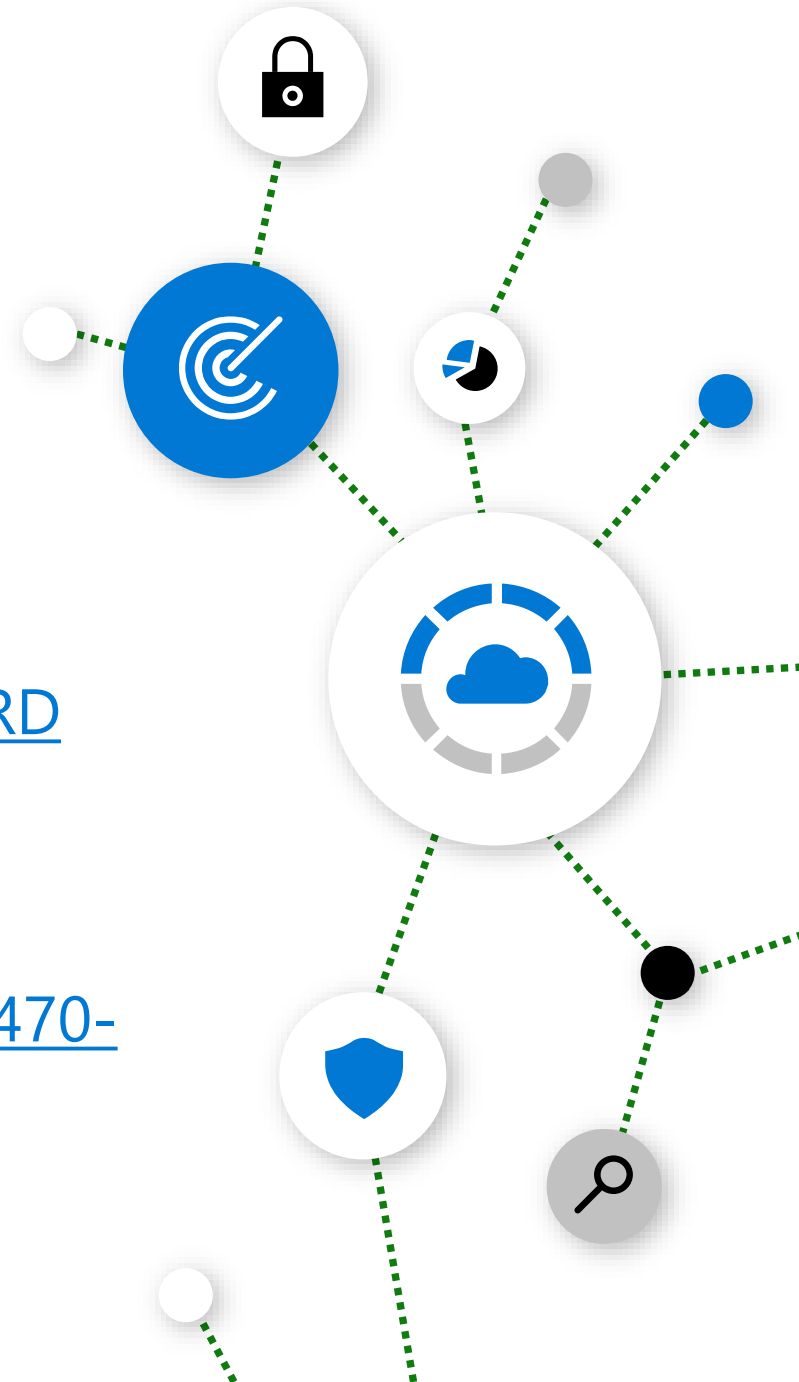


# TinyWall – CVE-2019-19470

<https://github.com/juliourena/CVE-2019-19470-RedTeamRD>

**CodeWhite Security - Descubrimiento**

<https://codewhitesec.blogspot.com/2020/01/cve-2019-19470-rumble-in-pipe.html>





# Laboratorio

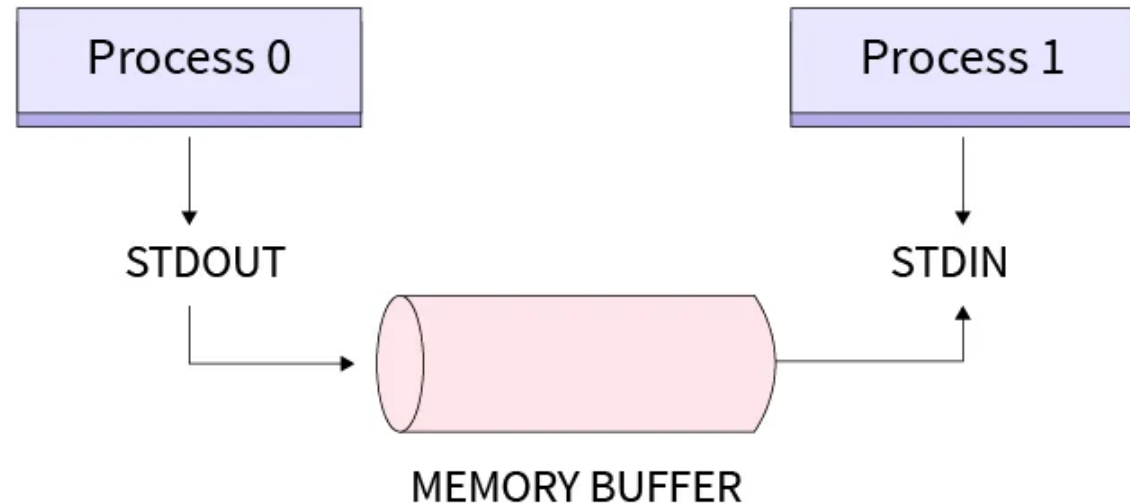
1. Uso de **dnSpy** para extraer código fuente de TinyWall.
2. Uso de **Graudit** para identificar posibles componentes que puedan ser explotados.
3. Análisis de código fuente TinyWall. Resaltar:
  - Serialización
  - NamedPipes
  - Autenticación



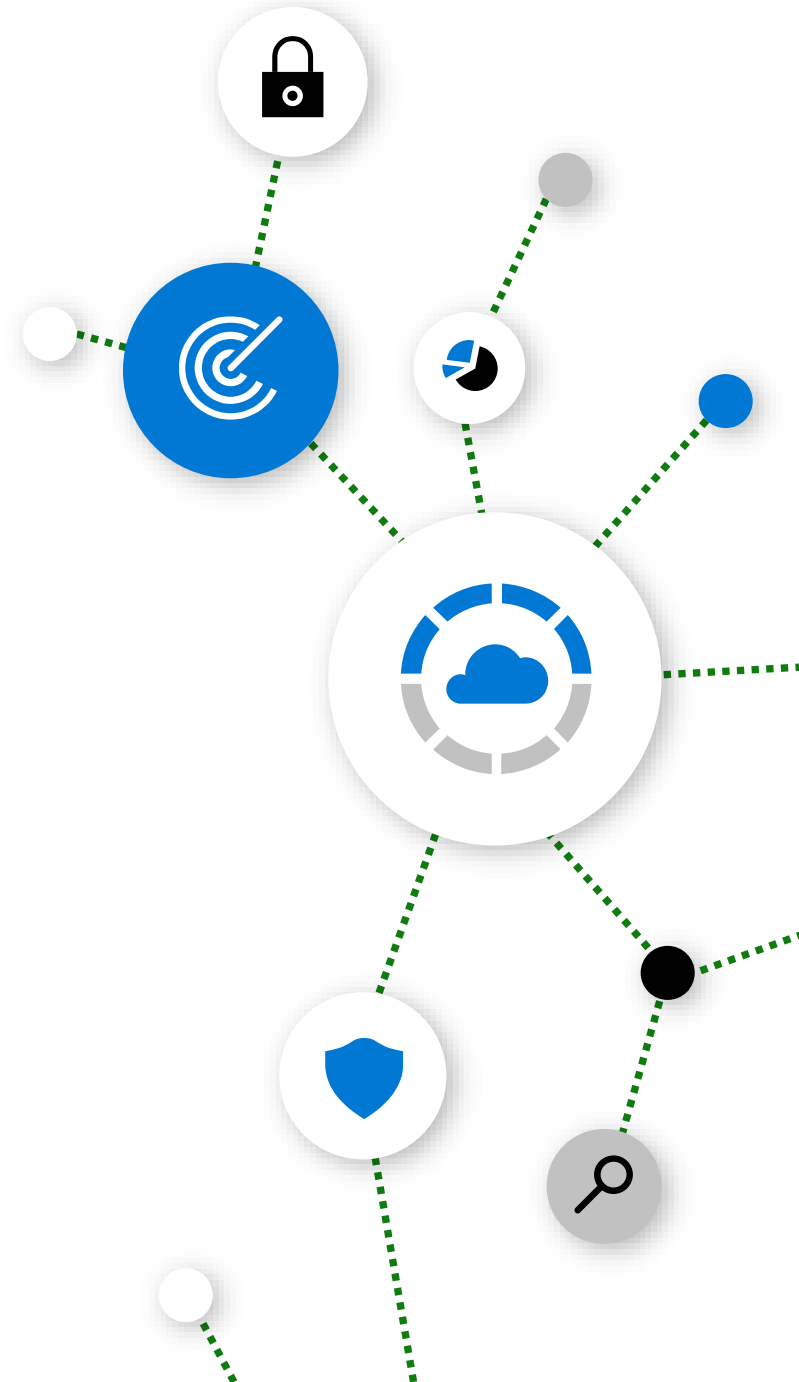
# Named Pipes

Un **Named Pipe** es un mecanismo de comunicación entre procesos (Inter-Process Communication, IPC) que permite a dos procesos en un sistema operativo compartir información de manera directa y segura.

Imaginemos que un **Named Pipe** es un canal de comunicación o un tubo (pipe) que conecta dos puntos (procesos), donde un proceso puede enviar datos por un extremo del tubo, y el otro proceso puede recibirlos por el otro extremo.



# Ejemplo de NamedPipes



# Serialización

La **serialización** es el proceso de convertir un objeto en un formato que puede ser fácilmente almacenado o transmitido, como JSON, XML, o un formato binario. Este proceso permite que los datos o el estado de un objeto sean guardados o enviados a través de la red y luego recuperados mediante deserialización.

```
public class Persona
{
    public string Nombre { get; set; }
    public int Edad { get; set; }
}

Persona persona = new Persona { Nombre = "Juan", Edad = 30 };
string jsonString = JsonSerializer.Serialize(persona);
```

{"Nombre":"Juan","Edad":30}

# Deserialización

La **deserialización** es el proceso mediante el cual un objeto que ha sido previamente serializado (convertido a un formato de datos como JSON, XML, o binario) es reconstruido a su estado original en la memoria de un programa. Es una técnica común en la programación que permite almacenar o transmitir objetos complejos y luego recuperarlos para su uso en aplicaciones.

```
// Deserializar la cadena JSON a un objeto Persona
Persona persona = JsonSerializer.Deserialize<Persona>(jsonString);

Console.WriteLine("Objeto deserializado:");
Console.WriteLine($"Nombre: {persona.Nombre}, Edad: {persona.Edad}");

// Ejemplo de salida:
// Nombre: Juan, Edad: 30
```

# Serialización - BinaryFormatter

**BinaryFormatter** es una clase en .NET utilizada para serializar y deserializar objetos en formato binario.

Aunque es potente y flexible, **BinaryFormatter** es conocido por sus riesgos de seguridad, ya que permite la ejecución de código durante la deserialización si no se maneja adecuadamente

```
static void Main()
{
    // Crear una instancia de Persona
    Persona persona = new Persona { Nombre = "Juan", Edad = 30 };

    // Serializar el objeto Persona a un flujo de memoria
    BinaryFormatter formatter = new BinaryFormatter();
    using (MemoryStream ms = new MemoryStream())
    {
        formatter.Serialize(ms, persona);
        byte[] data = ms.ToArray();

        // Guardar los datos serializados en un archivo
        File.WriteAllBytes("persona.bin", data);
    }

    Console.WriteLine("Objeto Persona serializado y guardado como 'persona.bin'.");
}
```

# Deserialización - BinaryFormatter

```
static void Main()
{
    // Leer los datos serializados desde un archivo
    byte[] data = File.ReadAllBytes("persona.bin");

    // Deserializar los datos a un objeto Persona
    BinaryFormatter formatter = new BinaryFormatter();
    using (MemoryStream ms = new MemoryStream(data))
    {
        Persona persona = (Persona)formatter.Deserialize(ms);

        Console.WriteLine("Objeto deserializado:");
        Console.WriteLine($"Nombre: {persona.Nombre}, Edad: {persona.Edad}");
    }
}
```

# BinaryFormatter Malicioso

```
[Serializable]
public class Persona
{
    public string Nombre { get; set; }
    public int Edad { get; set; }

    // Constructor malicioso que se ejecutará durante la deserialización
    public Persona()
    {
        // Código malicioso: Ejecuta la línea de comandos
        Console.WriteLine(";Código malicioso ejecutado!");
        Process.Start("cmd.exe");
    }
}
```



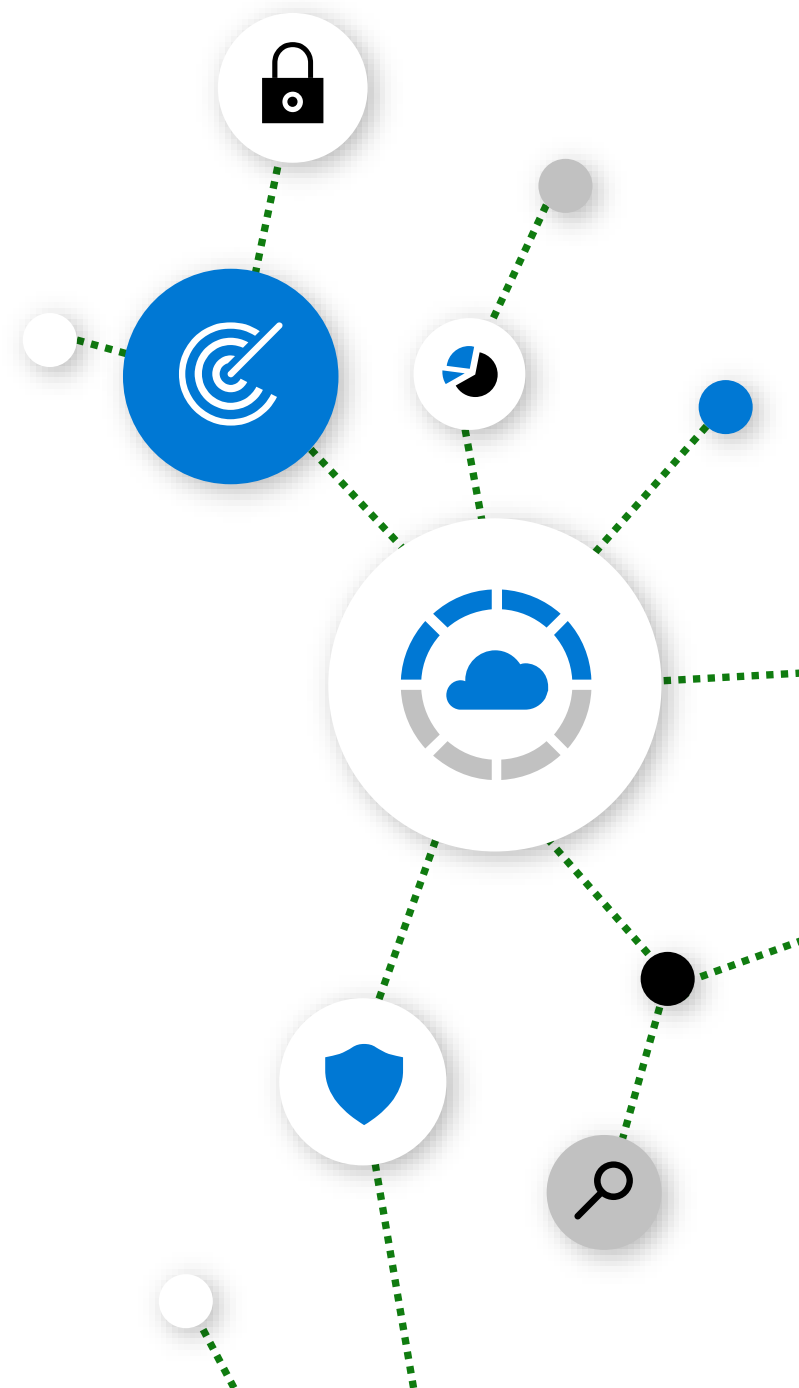
# Ysoserial .NET

**Ysoserial.net** es una herramienta utilizada para generar payloads maliciosos que explotan vulnerabilidades de deserialización en aplicaciones .NET. Inspirada en la herramienta original ysoserial para Java, ysoserial.net crea cadenas de objetos que, cuando son deserializadas por una aplicación vulnerable, pueden desencadenar la ejecución de código arbitrario.

<https://github.com/pwntester/ysoserial.net>



# Ejemplo de Deserialización



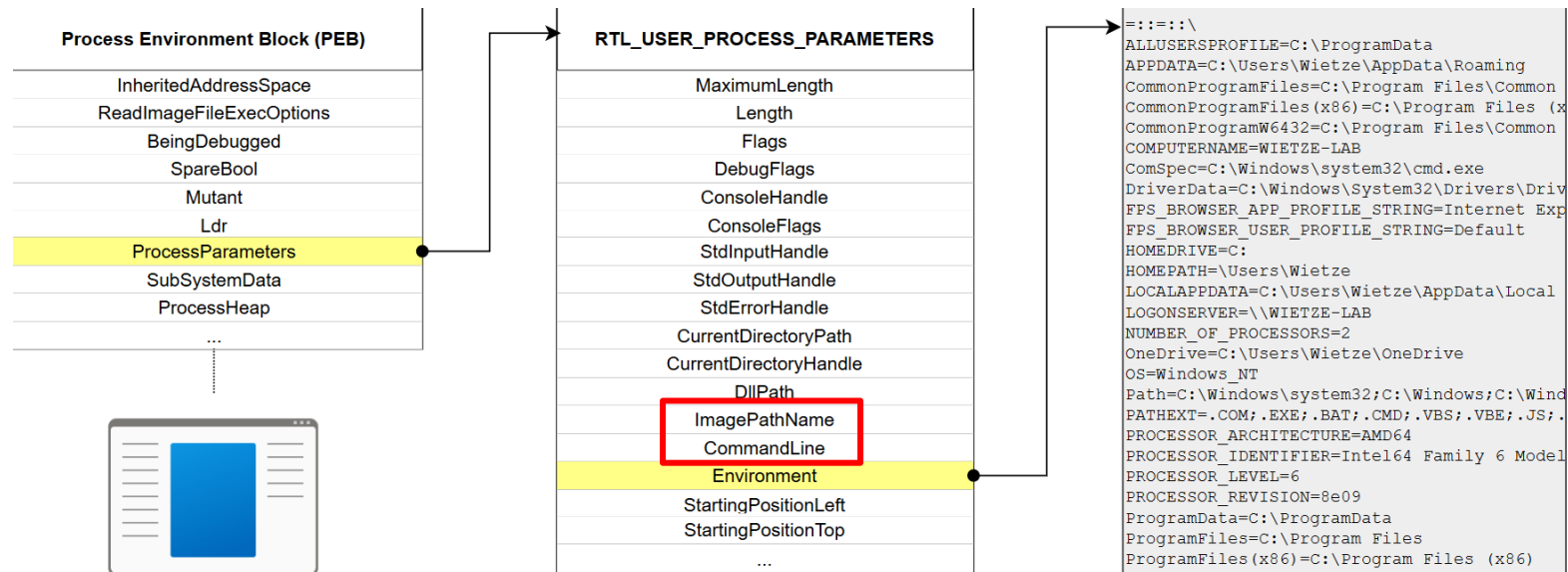
# WinDbg

**WinDbg** es una herramienta de depuración avanzada desarrollada por Microsoft, utilizada para analizar y depurar aplicaciones en Windows, investigar fallos del sistema (BSODs), y realizar análisis forenses de procesos en ejecución o archivos de volcado de memoria. Es una de las herramientas más poderosas para ingenieros de software, analistas de seguridad, y profesionales que necesitan comprender el comportamiento interno de aplicaciones y el sistema operativo Windows.



# Process Enviroment Block (PEB)

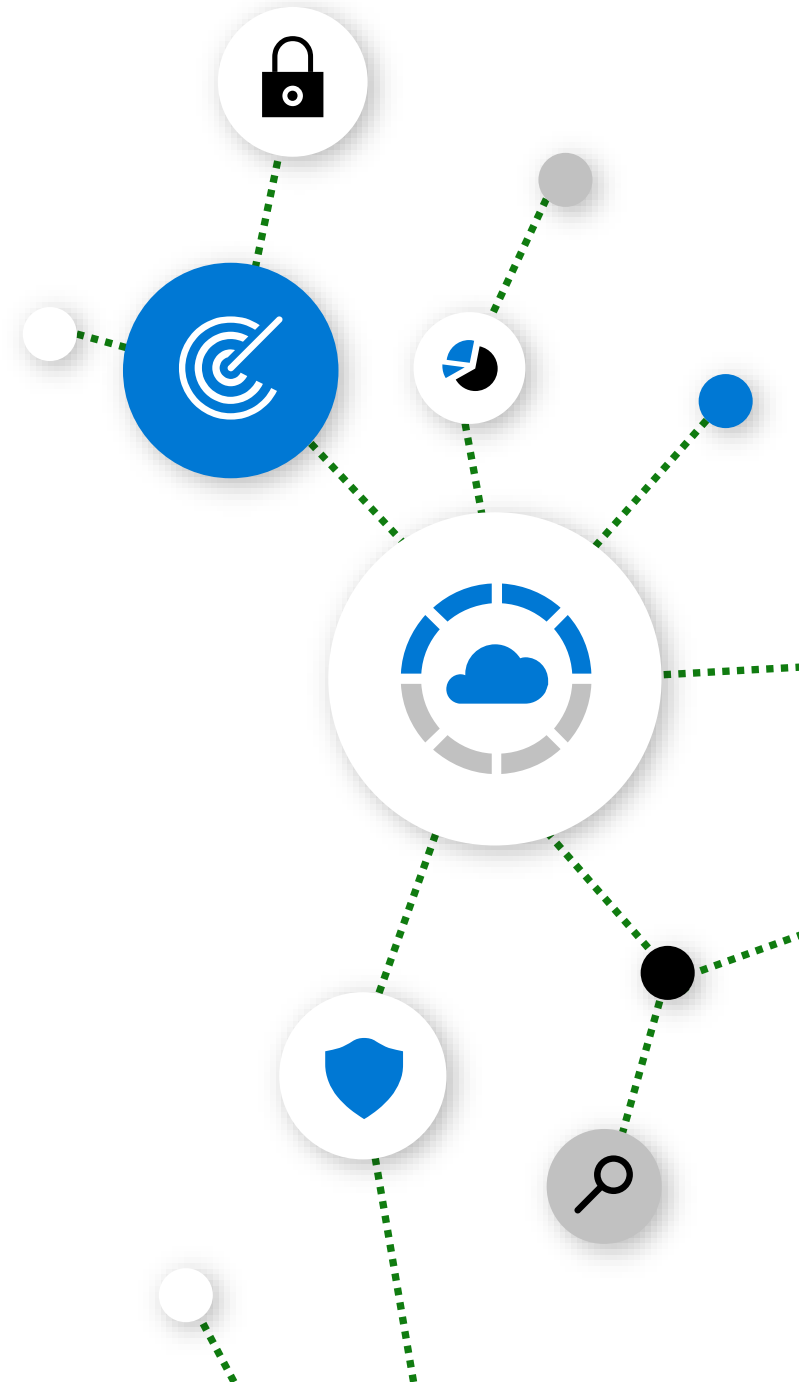
El **Process Environment Block (PEB)** es una estructura de datos en memoria utilizada por el sistema operativo Windows para almacenar información sobre un proceso en ejecución. Esta estructura contiene detalles cruciales como el command line del proceso, la ubicación de las variables de entorno, los parámetros de ejecución, y varios punteros a otras estructuras importantes del sistema.





# Autenticación en TinyWall

**Uso de Windbg para cambiar el nombre del proceso...**





# Creación del Exploit:

1. Cambiar el **MainModule.FileName** de nuestro binario al de TinyWall.exe
2. Establecer una conexión con el NamedPipe **TinyWallController**
3. Crear nuestro payload con **ysoserial .NET** y **Metasploit**.
4. Enviar el payload por la conexión del NamedPipe.

