Scripting con PowerShell: 5 ejemplos de automatización extrema (y cómo pueden congelar tu sistema)

PowerShell es, sin duda, una de las herramientas más potentes para los administradores de sistemas Windows. Nos permite automatizar tareas complejas en segundos, gestionar servidores remotos y realizar auditorías de seguridad. Sin embargo, **un gran poder conlleva una gran responsabilidad**.

Debido a que PowerShell tiene acceso directo a las APIs profundas del sistema operativo, un bucle mal diseñado o un comando mal ejecutado pueden transformarse en lo que en ciberseguridad se conoce como un **ataque DoS local (Denegación de Servicio)** o una «bomba lógica».

En este artículo, analizaremos desde un punto de vista puramente técnico y educativo 5 ejemplos de scripts que demuestran la potencia extrema de PowerShell, incluyendo aquellos capaces de dejar un sistema completamente congelado.

El peligro de la saturación de recursos

Cuando ejecutamos scripts que consumen el 100% de la CPU, agotan la memoria RAM o llenan el espacio en disco duro, Windows pierde la capacidad de gestionar sus propios procesos básicos. Esto provoca que el sistema se «cuelgue» o muestre el temido pantallazo azul (BSOD).

A continuación, vemos cómo se estructuran estos scripts y qué impacto real tienen sobre el hardware.

> ⚠️ ***ADVERTENCIA DE SEGURIDAD:*** *Los siguientes fragmentos de código tienen fines educativos y de pruebas de estrés (stress testing). No los ejecutes en entornos de producción. Si deseas probarlos, hazlo siempre dentro de una* ***Máquina Virtual (VM)*** *aislada.*

1. La «Bomba Fork» (Bucle infinito de procesos)

Este script utiliza un bucle while($true) que nunca termina. Su único objetivo es arrancar instancias del Bloc de notas (notepad.exe) a la máxima velocidad que permita el procesador.

Al no incluir una pausa o retraso, la CPU se satura al 100% en cuestión de segundos y la tabla de procesos de Windows colapsa.

PowerShell

```
# Bucle infinito que abre subprocesos sin detenerse
while ($true) {
    Start-Process notepad.exe
}
```

- **Impacto:** Congelamiento total del sistema operativo por agotamiento de hilos de la CPU.

2. Denegación en el Navegador (Apertura masiva de pestañas)

Similar al concepto anterior, pero enfocado en el software. Este script abre una URL específica en Google Chrome de manera indefinida. Dado que los navegadores modernos crean un proceso independiente por cada pestaña y consumen una cantidad considerable de memoria, la memoria RAM del equipo llegará a su límite rápidamente.

PowerShell

```
# Abre la misma URL en Chrome en un bucle sin fin
while ($true) {
    Start-Process "chrome.exe" "https://www.google.com"
}
```

- **Impacto:** Bloqueo del navegador y congelamiento de Windows por falta de memoria RAM disponible.

3. Creación masiva de directorios en el disco

Este código utiliza un bucle for para crear 10,000 carpetas numeradas secuencialmente en la ubicación actual. Aunque este script en particular no congelará tu ordenador de inmediato, genera un pico de actividad brutal en el almacenamiento (I/O de disco) y ralentiza drásticamente el Explorador de Archivos de Windows.

PowerShell

```
# Crea 10,000 carpetas indexadas en segundos
for ($i = 1; $i -le 10000; $i++) {
    New-Item -Path ".\Carpeta_$i" -ItemType Directory -Force
}
```

- **Impacto:** Degradación del rendimiento del disco duro y saturación de la tabla de archivos (MFT).

4. Auditoría y Rastreo de Archivos en Red (Uso de Carpetas Compartidas)

Este es un ejemplo clásico de un script que los administradores utilizan para buscar vulnerabilidades (y que los atacantes emplean para la fase de reconocimiento). Escanea una ruta de red compartida de forma recursiva buscando

archivos específicos (Excel, Word, PDF, texto) cuyos nombres contengan palabras clave como «password» o «contabilidad».

PowerShell

```
# Define la ruta de la carpeta compartida en la empresa
$rutaRed = "\\ServidorEmpresa\Compartido"

# Busca archivos específicos de forma recursiva ignorando errores de acceso
Get-ChildItem -Path $rutaRed -Recurse -Include *.xlsx, *.docx, *.txt, *.pdf -ErrorAction SilentlyContinue |
    Where-Object { $_.Name -like "*password*" -or $_.Name -like "*contabilidad*" } |
    Select-Object FullName, Length
```

- **Impacto:** Alto consumo de ancho de banda en la red local y exposición potencial de información confidencial.

5. El comando de borrado masivo y recursivo

El equivalente en PowerShell al peligroso comando rm -rf de Linux. Al combinar Remove-Item con el parámetro -Recurse (borra todo el contenido y subcarpetas) y -Force (elimina archivos ocultos o de sistema sin pedir confirmación), puedes vaciar directorios enteros al instante. Si se ejecuta en la raíz del sistema (C:\), el sistema operativo quedará inservible.

PowerShell

```
# BORRADO TOTAL (Simulado de manera segura en una carpeta de pruebas)
Remove-Item -Path "C:\CarpetaPrueba" -Recurse -Force
```

- **Impacto:** Pérdida irreversible de datos o destrucción del sistema operativo si se aplica en rutas críticas.

Tabla comparativa de impactos

Acción en PowerShell	¿Cuelga el sistema?	Recurso Afectado	Riesgo Principal
Bucle infinito de procesos	✅ **Sí**	CPU / Procesador	Inestabilidad inmediata del sistema.
Apertura masiva de URLs	✅ **Sí**	Memoria RAM	Desbordamiento de memoria.
Creación de miles de carpetas	⚠️ **A veces**	Almacenamiento (Disco)	Lentitud extrema en el explorador.
Escaneo masivo de red	❌ No	Red / Ancho de banda	Filtración o exfiltración de datos.
Borrado recursivo forzado	⚠️ Solo si toca C:\	Integridad de Datos	Pérdida de información del sistema.

¿Cómo detener un script de PowerShell fuera de control?

Si alguna vez ejecutas un script de automatización y notas que el sistema empieza a ralentizarse, mantén la calma e intenta aplicar las siguientes contramedidas antes de que el equipo deje de responder por completo:

1. **Detención inmediata:** Haz clic en la ventana de la consola donde corre el script y presiona la combinación de teclas **Ctrl + C**. Esto envía una señal de interrupción al intérprete de comandos.
2. **Matar procesos por comando:** Si la consola original no

responde pero tienes otra ventana de PowerShell abierta como administrador, puedes forzar el cierre de los procesos saturados ejecutando:PowerShellStop-Process -Name "notepad" -Force Stop-Process -Name "chrome" -Force

3. **Administrador de tareas:** Usa **Ctrl + Shift + Esc**, busca el proceso PowerShell.exe o Windows PowerShell y haz clic en "Finalizar tarea".

Conclusión

PowerShell es un aliado indispensable para la ingeniería de sistemas y la ciberseguridad, pero su flexibilidad requiere un diseño de scripts meticuloso. Añadir mecanismos de control, como límites en los bucles (counters) o pausas temporales (Start-Sleep), es vital para evitar que una herramienta de optimización acabe congelando la infraestructura de tu empresa.