POWERSHELL: Eventos WMI

1. Catalogación de eventos WMI

Eventos: Acciones generadas por un sistema o Framework informando de un echo. En este caso generadas por WMI, como puede ser el inicio de un proceso, la lectura de un fichero o la modificación de una clave de registro.

Es posible subscribirse a un evento y decidir que acción tomar cuando este ocurra, estas subscripciones pueden ser:

- **Subscripción temporal:** Subscripción activa mientras que el proceso que la ejecuta permanezca activo.
- **Subscripción permanente:** Subscripción almacenada en base de datos CIM y persiste mientras permanezca en ella, por lo que permanece ante reinicios.

Las subscripciones tienen tres componentes:

- **Filtro:** Consulta **WQL** para definir a que evento nos queremos subscribir.
- **Consumidor:** Definición de la acción a tomar cuando el filtro se active.
- **Enlace:** Relación entre filtro y consumidor con el evento en cuestión.

Y existen tres tipos de filtros para eventos:

- **Intrínsecos:** Monitorizan eventos que tienen representación directa en el repositorio CIM.
  A la hora de monitorizar eventos Intrínsecos, tenemos tres clases básicas según el tipo de acción que genera el evento:
  - **Creación:** Clase **__InstanceCreationEvent**
  - **Borrado:** Clase **__InstanceDeletionEvent**
  - **Modificación:** Clase **__InstanceModificationEvent**
- **Extrínsecos:** Eventos no relacionados directamente en el repositorio CIM (Registro, tablas enrutamiento, etc.). Estos tendrán clases específicas para monitorización de eventos:
  Ejemplo para registro:
  - Clase **RegistryKeyChangeEvent**
  - Clase **RegistryTreeChangeEvent**
  - Clase **RegistryValueChangeEvent**
- **Temporizadores:** Subconjunto de eventos intrínsecos que ocurren secuencialmente en el tiempo (relacionados con la hora **Win32_LocalTime** o **Win32_UTCTime**).

Para tener una referencia visual de los conceptos vistos anteriormente, vamos a desarrollar unos pequeños scripts para listar las clases de tipo evento, según si son intrínsecos o extrínsecos:

- **Listado de clases tipo Evento:**

```
function Get-WmiEventClass {
        Param(
        # Espacio de nombres donde iniciar la búsqueda
        [string]$Namespace='ROOT\CIMv2',
        # Filtro para acotar búsquedas
                [string]$Filter
    )
    # Selección de meta-clases de tipo __Event
        Get-WmiObject -ErrorAction SilentlyContinue -Query "SELECT * FROM meta_class WHERE (__This ISA '__Event') AND (__Class like '%$Filter%')"
}
```

- **Obtención subgrupo de clases Evento de tipo Extrínseco:**

```
function Get-WmiExtrinsicEvent {
        Param(
        # Espacio de nombres donde iniciar la búsqueda
        [string]$Namespace='ROOT\CIMv2',
        # Filtro para acotar búsquedas
        [string]$Filter
    )
    # Clases derivadas de __ExtrinsicEvent pero relacionadas con gestión de errores.
        $ExclusionList = @(
        '__SystemEvent',
        '__EventDroppedEvent',
        '__EventQueueOverflowEvent',
        '__QOSFailureEvent',
        '__ConsumerFailureEvent'
    )
    # Selección de meta-clases de tipo __Event
    Get-WmiObject -ErrorAction SilentlyContinue -Namespace $Namespace -Query "SELECT * FROM meta_class WHERE (__This ISA '__Event') AND (__Class like '%$Filter%')" |
        # Eliminación de clase temporizadora, e inclusión de clasess de tipo __ExtrinsecEvent no excluidas.
                Where-Object{$_.Name -eq '__TimerEvent' -or ($_.Derivation.Contains('__ExtrinsicEvent') -and ($ExclusionList -notcontains $_.Name))}
}
```

- **Obtención subgrupo de clases Evento de tipo Intrínseco:**

```
function Get-WmiIntrinsicEvent {
        Param(
        # Espacio de nombres donde iniciar la búsqueda
        [string]$Namespace='ROOT\CIMv2',
        # Filtro para acotar búsquedas
                [string]$Filter
    )
    # Exclusión de clases extrinsecas y temporizadoras
        $ExclusionList = @(
        '__ExtrinsicEvent',
        '__TimerEvent'
    )
    # Selección de meta-clases de tipo __Event
    Get-WmiObject -ErrorAction SilentlyContinue -Namespace $Namespace -Query "SELECT * FROM meta_class WHERE (__This ISA '__Event') AND (__Class like '%$Filter%')" |
        # Exclusión de clases derivadas de __ExtrinsicEvent o que no cumplan filtro especificado.
                Where-Object{ (-not $_.Derivation.Contains('__ExtrinsicEvent') -and ($ExclusionList -notcontains $_.Name)) }
}
```

Finalmente demos un sentido práctico a las selecciones anteriores, realizando búsquedas más acotadas:

- **Obtención de clase tipo Evento que contenga el patrón 'ShutdownEvent' y visualización de sus propiedades:**
  $evento = Get-WmiEventClass -Filter ShudownEvent
  $evento.Properties | Format-Table

- **Listado de eventos Extrínsecos con patrón 'Instance':**
  Get-WmiIntrinsicEvent -Namespace root\hardware -Filter Instance | Format-List

- **Eventos Intrínsecos con patrón 'LogonFailed' y detalles de sus propiedades:**
  Get-WmiExtrinsicEvent -Filter LogonFailed | Select-Object -ExpandProperty Properties

Hasta ahora, hemos visto como se catalogan las Clases que gestionan los eventos, sus tipos, elementos que conforman una subscripción, y las dos clases de subscripciones que hay. A nivel teórico ya conocemos los conceptos de estructuración genéricos necesarios.

Importante:

- La subscripción a un evento puede ser **temporal**, o **persistente**.
- Una subscripción se divide en **filtro**, **consumidor** y **enlace**.
- Existen tres tipos de filtros de eventos, **intrínsecos**, **extrínsecos** y **temporizadores**.

2. Definición de filtros con WQL

Como hemos visto anteriormente, una subscripción se divide en tres bloques, uno de los cuales es el Filtro, este se utiliza para definir sobre que evento queremos realizarla subscripción. Para ello se ha de seleccionar la Clase adecuada y se referencia que tipo de acción queremos monitorizar, Creación, Modificación, Borrado, etc...

Para la selección de clases, WMI utiliza la sintaxis WQL[1] (SQL para WMI) la cual es muy similar a SQL tradicional, con ciertos operadores específicos según tipo de selección deseada.

A grandes rasgos una consulta WQL tiene la siguiente sintaxis:

```
SELECT [Propiedad] FROM [Clase WMI] WHERE [Expresión]
```

La cláusula SELECT la utilizaremos para seleccionar las propiedades que deseemos del evento seleccionado. Puede ser desde su forma más sencilla (*), hasta selecciones más detalladas indicando las propiedades exactas a obtener.

```
SELECT * FROM [Clase Evento]
SELECT Propiedad1, Propiedad2, Propiedad3, FROM [Clase Evento]
```

En la cláusula FROM, hemos de indicar la clase Evento según que acciones se deseen monitorizar, para conocer las clases de tipo evento disponibles, siempre podemos utilizar el WMI Explorer 2.0 o los scripts vistos en puntos anteriores.

```
SELECT * FROM __InstanceCreationEvent
SELECT TargetInstance FROM __InstanceModificationEvent
SELECT * FROM CIM_InstModification
SELECT * FROM RegistryValueChangeEvent
```

Las expresiones a utilizar en la cláusula WHERE van a depender de la clase evento con que se esté trabajando. En las ocasiones en que tratemos con eventos intrínsecos, se hará necesario utilizar la expresión:

```
WHERE TargetInstance ISA '[Nombre de Clase]'
```

Donde el operador ISA comprueba que la propiedad TargetInstance es de la clase indicada.

```
SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_Process'
```

Esta consulta seleccionaría todos los eventos de creación de instancias de tipo Win32_Process, o lo que es lo mismo, se utilizaría para monitorizar la creación de procesos en el sistema.

Como nota a la consulta de ejemplo anterior, el operador WITHIN indica el tiempo que se desea estar recolectando información antes de recibir la notificación, para lo cual Microsoft recomienda no establecer intervalos cortos para evitar sobrecargas de procesamiento (5 Segundos para entornos de pruebas y 30 segundos como mínimo para entornos en producción).

```
SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_USBHub'
```

La consulta anterior, monitoriza en busca de creación de instancias de clase Win32_USBHub, es decir, notifica la detección de nuevos dispositivos de almacenamiento USB.

Si por el contrario quisiéramos tratar con eventos Extrínsecos, la consulta ha de utilizar las clases adecuadas que se pueden localizar con los scripts anteriores o con WMI Explore 2.0 como vimos anteriormente:

```
SELECT * FROM RegistryKeyChangeEvent WHERE Hive='HKEY_LOCAL_MACHINE' AND KeyPath='Software\Microsoft\Windows\CurrentVersin\Run'
```

Esta consulta busca eventos de modificación de claves de registro en la ruta especificada en la cláusula WHERE.

Importante: A grandes rasgos siempre tendremos que especificar el tipo de **evento**, la **clase** de la instancia que genera el evento, y los posibles **filtros** para afinar la búsqueda.

3. Datos para definición de acciones

Una vez se tiene definida la consulta para el filtro, la acción (Consumidor) será en esencia Powershell puro, en el cual tendremos ciertas variables globales[2] específicas del evento al que nos hemos subscrito, dispuestas para utilizar y programar la acción exacta a realizar:

- **$event:** Objeto **PSEventArgs**[3] que representa el evento que se está procesando.
- **$eventArgs:** Contiene el objeto que representa el primer argumento del evento que se esté tratando, y es un "alias" a la propiedad **SourceEventArgs** que podemos encontrar en la variable global **$event**, utilizada para simplificar codificaciones.

A groso modo, con estas dos variables tendremos todo lo necesario para dar forma al código de la acción.

```
$eventArgs == $event.SourceEventArgs
```

Cabe destacar que, en la mayoría de los casos, los datos más relevantes que nos van a interesar son los referentes a las propiedades de la instancia de clase, que habitualmente encontraremos en:

```
$Event.SourceArgs.NewEvent.TargetInstance
```

Pero ya que esto no es una certeza para el 100% de los casos, es muy recomendable depurar el código de la acción e inspeccionar la variable **$event** en busca de la localización de los datos deseados. Como podría ser:

```
{
    # Debug
    $Global: evento = $event
    $Datos = $Event.SourceArgs.NewEvent.TargetInstance
    Write-Host "Author:", $Datos.Author
    Write-Host "TaskName:", $Datos.TaskName
    Write-Host "Actions.Execute:", $DatosEvent.Execute
    Write-Host "Actions.Arguments:", $Datos.Arguments
}
```

En el bloque anterior, se define una variable global llamada $evento, que se podrá inspeccionar inclusive fuera del contexto de ejecución del evento, y posteriormente se muestran por pantalla ciertos detalles de la instancia que ha provocado el evento. En caso de no obtener los resultados esperados, podemos inspeccionar la variable $evento para tareas de depuración.

4. CmdLets para gestión de Subscripciones

La posibilidad para la gestión de subscripciones varía según la versión de Powershell y a su vez del tipo de subscripción deseada.

A. Subscripciones Temporales

Powershell 1.0 (Componente opcional en XP SP2/ 2003 SP1 / Vista):
En la versión 1.0 no dispondremos de cmdlets para la creación de subscripciones a eventos WMI, teniendo que utilizar las clases nativas .NET. Por otro lado, para la obtención de un listado de eventos o su eliminación tendríamos los cmdlets.

- **Get-EventSubscriber** [[-SourceIdentifier] <String>] [-Force] [<CommonParameters>]
- **Get-EventSubscriber** [-SubscriptionId] <Int32> [-Force] [<CommonParameters>]
- **Unregister-Event** [-SourceIdentifier] <String> [-Confirm] [-Force] [-WhatIf] [<CommonParameters>]
- **Unregister-Event** [-SubscriptionId] <Int32> [-Confirm] [-Force] [-WhatIf] [<CommonParameters>]

Se ha de anotar que, aunque estos cmdlets puedan listar o eliminar subscripciones a eventos WMI, no son específicos para estos, sino para eventos .NET (Objetos .NET), ya que en un entorno Powershell no podemos olvidar que existes eventos generados por objetos WMI, .NET o por Powershell.

Dado que en su versión 1.0 Powershell era un componente opcional, es habitual diseñar pensando en compatibilidad como mínimo con powershell 2.0.

Powershell 2.0 (W7 / W2008 R2 / XP SP3 / W2003 SP2 / Vista SP 1):
En su versión 2.0 para listado o eliminación tenemos los dos cmdlets vistos en la versión 1.0, pero aparece un cmdlet específico para creación de subscripción temporal.

- **Register-WmiEvent** [-Class] <string> [[-SourceIdentifier] <string>] [[-Action] <scriptblock>] [<CommonParameters>]
- **Register-WmiEvent** [-Query] <string> [[-SourceIdentifier] <string>] [[-Action] <scriptblock>] [<CommonParameters>]

Powershell 3.0 (W8 / W2012 / W7 SP1 / W2008 SP1 / W2008 R2 SP1): A partir de la versión 3.0 aparece un cmdlet específico para eventos CIM, con compatibilidad hacia atrás.

- **Register-CimIndicationEvent** [-ClassName] <string> [[-SourceIdentifier] <string>] [[-Action] <scriptblock>] [<CommonParameters>]
- **Register-CimIndicationEvent** [-Query] <string> [[-SourceIdentifier] <string>] [[-Action] <scriptblock>] [<CommonParameters>]

En las subscripciones temporales, el "Enlace" lo realiza el propio cmdlet correspondiente, por lo que solo se han de establecer el filtro y el consumidor. Aunque es posible definir el filtro a través del parámetro Class o del parámetro Query, este último siempre permitirá afinar más los parámetros de selección.

B. Subscripciones Persistentes

Al igual que con las subscripciones temporales, según la versión de Powershell utilizada tendremos unos cmdlets u otros.

Powershell 2.0 (W7 / W2008 R2 / XP SP3 / W2003 SP2 / Vista SP 1):

- **Set-WmiInstance:** https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/set-wmiinstance?view=powershell-5.1

Powershell 3.0+ (W8 / W2012 / W7 SP1 / W2008 SP1 / W2008 R2 SP1):

- **New-CimInstance:** https://technet.microsoft.com/en-us/itpro/powershell/windows/cimcmdlets/new-ciminstance

Finalmente, las acciones de obtención y borrado de subscripciones persistentes se realizan con los cmdlets:

- **Get-WMIObject:** https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-wmiobject?view=powershell-5.1
- **Remove-WmiObject:** https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/remove-wmiobject?view=powershell-5.1

También hay que tener en cuenta que las acciones realizadas a partir de un evento persistente se ejecutan como SYSTEM y que estas subscripciones persisten ante reinicios.
Como último comentario, para definir las acciones a realizar, tenemos exactamente cinco tipos de consumidores:

- **ActiveScriptEventConsumer:** Ejecuta script predefinido ya sea JScript o VBScript. Disponible desde W2000+
- **CommandLineEventConsumer:** Ejecuta el proceso indicado. Disponible desde XP+
- **LogFileEventConsumer:** Escribe datos customizables a fichero de log. Disponible desde XP+
- **NTEventLogEventConsumer:** Genera un Evento NT. Disponible desde XP+
- **SMTPEventConsumer:** Envía un correo electrónico. Disponible desde W2000+

5. Subscripción a eventos temporales

Partiendo de lo visto anteriormente vamos con ejemplos de subscripciones temporales ahora ya con una estructura lo más real posible.

A. Subscripción temporal (a evento intrínseco):

A la hora de tratar con eventos intrínsecos la definición de filtro puede simplificarse como ya vimos en las notas sobre WQL.

```
SELECT * FROM __Instance[Creation|Deletion|Modification]Event WITHIN [5..30] WHERE TargetInstance ISA '[Clase]'
```

Utilizando este tipo de consulta, los cmdlets siguientes muestran un mensaje por pantalla cada vez que se cree o termine un proceso, indicando su nombre.

> **Ejemplo:**

```
Register-WmiEvent -SourceIdentifier MiSubscripcion_1 `
-Query "Select * From __InstanceCreationEvent WITHIN 5 Where TargetInstance ISA 'Win32_Process'"
-Action  {
```

```
    Write-Warning "Creado nuevo proceso", ($Event.SourceArgs.newevent.targetinstance.CommandLine)
}
Register-WmiEvent -SourceIdentifier MiSubscripcion_2
-Query "Select * From __InstanceDeletionEvent WITHIN 5 Where TargetInstance ISA 'Win32_Process'"
-Action  {
    Write-Warning "Creado nuevo proceso", ($Event.SourceArgs.newevent.targetinstance.CommandLine)
}
```

Para mostrar el nombre de una tarea programada en el momento de su creación, se puede utilizar la siguiente subscripción.

Ejemplo:

```
Register-WmiEvent -SourceIdentifier MiSubscripcion_3
-Namespace ROOT\Microsoft\Windows\TaskScheduler -Query "SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE targetinstance ISA 'MSFT_ScheduledTask'"
-Action  {
    Write-Host "Creada nueva tarea programada con comando: ", ($Event.SourceArgs.NewEvent.TargetInstance.Actions.Execute)
}
```

B. Subscripción temporal (Evento Extrínseco):

La subscripción a eventos extrínsecos es idéntica a las vistas anteriormente, solo hay que variar la consulta de selección de evento utilizando las clases específicas disponibles,

A continuación, se muestran dos ejemplos de monitorización de creación de procesos o de modificación en clave de auto-ejecución.

Ejemplo:

```
Register-WmiEvent -SourceIdentifier MiSubscription_4
-Class Win32_ProcessStartTrace
-Action {
    Write-Warning "Creado nuevo proceso", ($Event.SourceArgs.newevent.ProcessName)
}

Register-WmiEvent -SourceIdentifier MiSubscription_5
-Query "SELECT * FROM RegistryKeyChangeEvent WHERE Hive='HKEY_LOCAL_MACHINE' AND KeyPath ='SOFTWARE\Microsoft\Windows\CurrentVersion\Run'"
-Action {
    Write-Host "Posible persistencia en Run"
}
```

En el último ejemplo para subscripción a un evento extrínseco veamos cómo detectar la aparición de un nuevo dispositivo de almacenamiento. En este caso solo se muestra la unidad en la que dicho nuevo dispositivo está mapeado.

Ejemplo:

```
Register-WmiEvent -SourceIdentifier MiSubscription_6
-Namespace root\cimv2 -Query "SELECT * FROM Win32_VolumeChangeEvent WHERE EventType = 2"
-Action {
       Write-Host "Insertado dispositivo en unidad ", ($Event.SourceArgs.NewEvent.DriveName)
}
```

C. Subscripción temporal (Evento Temporizador):

El siguiente ejemplo muestra un contador numérico cada 30 segundos.

Ejemplo:

```
Register-WmiEvent -SourceIdentifier MiSubscription_7
-Query "SELECT * FROM __InstanceModificationEvent WHERE TargetInstance ISA 'Win32_LocalTime' AND (TargetInstance.Second=30 OR TargetInstance.Second=1)"
-Action {
       $global:contador += 1
       Write-Host "Contador: ", ($contador)
}
```

D. Subscripción con variantes CIM:

La subscripción a eventos CIM, no es más que la utilización de las clases CIM, en vez de las clásicas WMI, y actualización hacia la que Microsoft está tendiendo desde la versión de Powershell 3.0, recordando que es posible utilizar estos cmdlets también para eventos WMI.

Los siguientes ejemplos muestran como monitorizar la creación de tareas programadas, la creación de procesos o la conexión de nuevos dispositivos de almacenamiento.

Ejemplo:

```
Register-CimIndicationEvent -SourceIdentifier MiSubscription_9
-Namespace root/Microsoft/Windows/TaskScheduler
-Query "SELECT * FROM CIM_InstCreation WITHIN 5 WHERE TargetInstance ISA 'MSFT_ScheduledTask'" -Action {
    Write-Host "Se ha creado una nueva tarea programada."
}

Register-CimIndicationEvent -SourceIdentifier MiSubscription_10
-ClassName "Win32_ProcessStartTrace"
-Action {
    Write-Host "Se ha creado un nuevo proceso."
}

Register-CimIndicationEvent -SourceIdentifier MiSubscription_11
-Namespace root/cimv2
-Query "SELECT * FROM Win32_VolumeChangeEvent WHERE EventType = 2" -Action {
    Write-Host "Detectado nuevo dispositivo de almacenamiento."
}
```

Para todos los ejemplos vistos anteriormente, las subscripciones pueden eliminarse mediante el cmdlet **Unregister-Event -SourceIdentifier MiSubscription_x**

E. Listado y eliminación de subscripciones temporales:

Una vez creada una subscripción temporal, puede llegar el momento en que se desee ver un listado de subscripciones temporales existentes, o eliminar una subscripción concreta. Como se vio en la sección 5, esto se realiza con los cmdlets, **Get-EventSubscriber** y **Unregister-Event**.

El cmdlet **Get-EventSubscriber** tiene dos parámetros principales, **-SourceIdentifier** y **-SubscriptionId** ambos utilizados para seleccionar la subscripción a seleccionar, ya sea por su nombre identificador o por su número de identificación.

Y finalmente si se desea eliminar una subscripción, utilizaremos **Unregister-Event** donde igualmente podemos seleccionar la subscripción a eliminar por nombre o por identificador.

6. Subscripción a eventos Persistentes

A la hora de crear subscripciones persistentes hemos de instanciar las clases filtro, consumidor y enlace de forma independiente utilizando el cmdlet Set-WmiInstance.

A. Definición de instancia filtro:

La instancia Filtro la definiremos utilizando las consultas WQL vistas anteriormente, añadiendo las propiedades específicas para un filtro, que serían:

- **Name:** Nombre único para identificar la instancia creada.
- **EventNamespace:** Espacio de nombres donde se cataloga el evento a monitorizar.
- **QueryLanguage:** Lenguaje utilizado para definir la consulta.
- **Query:** Consulta para selección de evento.

> Ejemplo:

```
$filtro = Set-WmiInstance -Namespace root\CIMv2 -Class __EventFilter
-Arguments @{
    Name = "MiSubscripcion_12"
    EventNamespace = "root\CIMv2"
    QueryLanguage = "WQL"
    Query = "SELECT * FROM __InstanceCreationEvent WHERE TargetInstance ISA ‘Win32_Process’"
}
```

Dado que estamos creando una instancia filtro, la propiedad **Class** siempre tendrá el valor **__EventFilter**. Por lo demás variando el espacio de nombres y la consulta, tendremos la posibilidad de crear filtros para cualquier evento existente en el repositorio CIM.

B. Definición de instancia Consumidor:

A la hora de crear una instancia consumidora, la propiedad Class la tendremos que establecer a una de las cinco clases posibles:

- **ActiveScriptEventConsumer**
- **CommandLineEventConsumer**
- **LogFileEventConsumer**
- **NTEventLogEventConsumer**
- **SMTPEventConsumer**

> Ejemplos:

```
$Consumidor = Set-WmiInstance -Namespace root\Subscription -Class LogFileEventConsumer
-Arguments @{
Name = "MiSubscripcion_13"
Text = ‘Proceso %ProcessName% creado.’
FileName = ‘C:\Windows\Temp\WMISubscripciones.Log’
}

$Consumidor = Set-WmiInstance -Namespace root\Subscription -Class ActiveScriptEventConsumer
-Arguments @{
Name = "MiSubscripcion_13"
ScriptFileName = ‘C:\Windows\Temp\consumidor.vbs’
ScriptEngine = ‘VBScript’
}

$Consumidor = Set-WmiInstance -Namespace root\Subscription -Class NTEventLogConsumer
-Arguments @{
Name = "MiSubscripcion_13"
EventID = 100
EventType = 2
SourceName = ‘WMI NTEventLog event consumer’
Category = 0
InsertionStringTemplates = { "Proceso %ProcessName% creado." }
}

$Consumidor = Set-WmiInstance -Namespace root\Subscription -Class CommandLineEventConsumer
-Arguments @{
        Name = "MiSubscripcion_13"
        ExecutablePath = "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe"
        CommandLineTemplate =
‘C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe
-C `"Get-Date|Out-File C:\\time.txt`" ‘
    RunInteractively='false'
}

$Consumidor = Set-WmiInstance -Namespace root\Subscription -Class SMTPEventConsumer
-Arguments @{
        Name = "MiSubscripcion_13"
        ToLine = "to@mail.com"
        CcLine = "cc@mail.com"
        ReplyToLine = "reply@mail.com"
        SMTPServer = "smtp.mail.com"
        Subject = "WARNING: Proceso creado"
        Message = "WARNING: Nuevo proceso %ProcessName% creado."
}
```

Dependiendo del tipo de acción que se desee realizar, nos convendrá utilizar una clase u otra, sabiendo que si queremos realizar tareas complejas, las dos clases que permiten Scripting serían CommandLineEventConsumer y ActiveScriptEventConsumer. Si por el contrario lo que se desea realizar es generar algún tipo de alerta o generar ficheros de log, las tres restantes serían las más adecuadas.

En todo caso, para saber exactamente que datos vamos a tener disponibles en las cinco clases podemos consultar:

- **ActiveScriptEventConsumer:** https://msdn.microsoft.com/en-us/library/aa384749(v=vs.85).aspx
- **CommandLineEventConsumer:** https://msdn.microsoft.com/en-us/library/aa389231(v=vs.85).aspx
- **LogFileEventConsumer:** https://msdn.microsoft.com/en-us/library/aa392277(v=vs.85).aspx

- **NTEventLogEventConsumer:** https://msdn.microsoft.com/en-us/library/aa392715(v=vs.85).aspx
- **SMTPEventConsumer:** https://msdn.microsoft.com/en-us/library/aa393629(v=vs.85).aspx

C. Definición de instancia Enlace:

Una vez tenemos definido un filtro adecuado, y el correspondiente consumidor solo resta relacionarlos. Esto se realiza nuevamente con el cmdlet **Set-WmiInstance**.

> Ejemplos:

```
$Enlace = Set-WmiInstance -Namespace root\Subscription -Class __FilterToConsumerBinding
-Arguments @{
    Name = "MiSubscripcion_13"
    Filter = $Filtro
    Consumer = $Consumidor
}
```

D. Listado y eliminación de subscripciones persistentes:

En el caso de las subscripciones persistentes, es posible obtener un listado según si nos interesa el filtro, el consumidor o en enlace, en los tres casos con el cmdlet **Get-WmiObject**.

> Ejemplos:

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

La eliminación se realiza de igual forma, teniendo que eliminar sus tres componentes utilizando el cmdlet **Remove-WmiObject**.

7. Casos de uso

Es conocido que, desde el descubrimiento de Stuxnet, la utilización de Powershell y WMI han proliferado en los denominados **APT**.

A. Listado de subscripciones

> Desde un punto de vista de investigación, es útil conocer que subscripciones existen en un sistema para poder inspeccionar si sus acciones son benignas o maliciosas, por lo que, para un primer caso práctico y ocasionalmente útil en la vida real, intentemos crear dos funciones, una para listar las subscripciones temporales existentes en la sesión actual, y un segundo para listar las subscripciones persistentes existentes en el sistema.

Para el primer apartado, consistente en el listado de subscripciones temporales, teníamos el cmdlet **Get-EventSubscriber**. Este cmdlet, permite obtener información de una suscripción temporal ya sea por identificador u obtener un listado completo.

Por lo que en primera instancia la obtención del listado lo realizaremos con:

```
Get-EventSubscriber -Force
```

En esta primera aproximación solo hemos añadido el parámetro **-Force** para que nos dé un listado completo, incluyendo aquellas subscripciones definidas como ocultas.

Para añadir la posibilidad de localizar una subscripción con un identificador conocido, hemos de utilizar el parámetro **-SubscriptionId**

```
Get-EventSubscriber -Force -SubscriptionId $Identificador
```

Si la localización de la subscripción la deseamos realizar por el nombre de la misma, para ello tenemos el parámetro **-SourceIdentifier**

```
Get-EventSubscriber -Force -SourceIdentifier $Nombre
Get-EventSubscriber -Force | ?{ $_.SourceIdentifier -match $Nombre }
```

Finalmente, si queremos dar un poco más de versatilidad a nuestra función que al cmdlet original, podemos añadir un filtro adicional, que acote los resultados según algún patrón definido que se localice en el comando que se ejecuta al lanzarse el evento. Esto se realiza utilizando la cláusula genérica de Powershell **Where-Object**, por lo que finalmente nos quedaría una función similar a:

> **Solución: Listado subscripciones temporales**

```
function Get-SessionEvent {
    param(
        # Búsqueda por ID
        [int]    $Identificador,
        # Búsqueda por nombre
        [string] $Nombre,
        # Búsqueda por patron en comando
        [string] $Filtro
    )
    if ($Identificador) {
        $subscripciones = Get-EventSubscriber -Force -SubscriptionId $Identificador
    } ElseIf ($Nombre) {
        $subscripciones = Get-EventSubscriber -Force -SourceIdentifier $Nombre
    } Else {
        $subscripciones = Get-EventSubscriber -Force
    }
    $subscripciones | Where-Object { $_.Action.Command -match $Filtro }
}
```

Solventada esta primera parte, ahora afrontemos la segunda y última.Como ya sabemos, una subscripción persistente se define con tres objetos, filtro, consumidor y enlace.
Para listar las subscripciones efectivas, bastaría con localizar los enlaces, lo cual se podría realizar con:

```
Get-WmiObject -NameSpace ‘root\Subscription’ -Class '__FilterToConsumerBinding'
```

Aunque esto ya nos daría un listado de subscripciones, encapsulemos todo aquello que nos facilite el trabajo, o lo que en desarrollo se llama modularidad.

Ya que una subscripción se define por tres objetos, creemos una función para listar de forma individual estos tres tipos de objetos, y hagamos que la última función sea la que nos aporte toda la información posible aportando

opciones de filtrado por filtro, consumidor o enlace.

Desarrollo: Listado subscripciones persistentes - Filtros

```
function Get-WmiFilter {
    param(
        [string] $Nombre,
        [string] $Consulta,
        [string] $Namespace = 'root\subscription'
    )
    Get-WmiObject -NameSpace $Namespace -Class '__EventFilter' |
    Where-Object { $_.Name -match $Nombre } |
    Where-Object { $_.Query -match $Consulta }
}
```

La función anterior, en primer lugar, obtiene objetos de tipo **__EventFilter**, o lo que es lo mismo, filtros de subscripciones.

De los resultados obtenidos, quita todos aquellos en los que el nombre asignado al filtro no contenga el patrón dado, e igualmente, elimina aquellos resultados que no contengan el patrón definido para las cláusulas de la consulta. Como nota exclusiva de Powershell, el operador **-match** siempre retorna cierto si la variable no existe o está vacía, por lo que no habría que preocuparse de verificar la existencia de los parámetros.

Ahora enfoquemos la obtención de consumidores:

Desarrollo: Listado subscripciones persistentes - Consumidores

```
function Get-WmiConsumer {
    param(
        [string]   $Nombre,
        [ValidateSet(
            'ActiveScriptEventConsumer',
            'SMTPEventConsumer',
            'NTEventLogEventConsumer',
            'LogFileEventConsumer',
            'CommandLineEventConsumer'
        )][string] $TipoConsumidor,
        [string]   $Namespace = 'root\subscription'
    )
    Get-WmiObject -NameSpace $Namespace -Class '__EventConsumer' |
    Where-Object { $_.Name -match $Nombre} |
    Where-Object { $_.__CLASS -match $TipoConsumidor}
}
```

En esta función se permite seleccionar ya sea por el nombre asignado al consumidor o por el tipo de acción que realiza, que como recordamos tenía que ser de uno de los cinco tipos permitidos.

En primer lugar, selecciona todos los objetos consumidores utilizando la clase **__EventConsumer**, de estos, quita aquellos en que su nombre no coincida con el parámetro proporcionado, y finalmente realiza lo mismo con el nombre de la clase que define la acción.

El operador ValidateSet exclusivamente se encarga de que, si el usuario decide filtrar por el tipo de acción, ha de utilizar un nombre de clase válido.

Perfecto, ahora listemos los enlaces:

Desarrollo: Listado subscripciones persistentes - Enlaces

```
function Get-WmiBinding {
    param(
        [string] $Filtro,
        [string] $Consumidor,
        [string] $Namespace = 'root\subscription'
    )
    Get-WmiObject -NameSpace $Namespace -Class '__FilterToConsumerBinding' |
    Where-Object { $_.Consumer -match $Consumidor } |
    Where-Object { $_.Filter -match $Filtro }
}
```

Esta función vuelve a tener una estructura similar, inicialmente se seleccionan todos los objetos de tipo **_FilterToConsumerBinding** (enlaces), y posteriormente se aplican filtros para eliminar aquellas que no coincidan con el nombre del filtro o del consumidor.

En este punto, multitud de ejemplos en internet, paran y dejarían estas tres funciones como la forma de lista subscripciones, y sería totalmente correcto, ya que se tiene la opción de, individualmente obtener todo lo referente a una subscripción, aunque en tres pasos. Vamos a intentar definir una única función que ya retorne toda la información en un solo objeto.

Para ello nuevamente hemos de recordar que una subscripción siempre se forma a partir de tres elementos, y que el enlace por obligación ha de tener algún tipo de referencia al filtro y al consumidor que dan forma a la subscripción.

Para facilitar la programación, **WQL** tiene el operador ASSOCIATORS OF que busca objetos relacionados entre si, como pueden ser discos con particiones, particiones con unidades, directorios con unidades, o como ya suponemos.... Filtros con consumidores y enlaces. Es decir, dada una relación A -> B -> C conocidos A y B ASSOCIATORS OF nos retornará C, pudiéndose variar el orden de selección (propiedad transitiva).

Esto que puede parecer complejo, solo requiere cierta práctica, por lo que empecemos con un ejemplo similar a:

Solución: Listado subscripciones persistentes

```
function Get-PersistentEvent {
  param(
    [string] $Filtro,
    [string] $Consumidor,
    [string] $Namespace = 'root\subscription'
  )
  $events = @()
  # Enlazados
  Get-WmiBinding -Namespace $Namespace -Filter $Filter -Consumer $Consumer | %{
    $event = New-Object -TypeName PSObject

    $event | Add-Member -MemberType NoteProperty -Name 'Filter' -Value (Get-WmiObject -Namespace root/subscription -Query "ASSOCIATORS OF {$($_.Consumer)}")

    $event | Add-Member -MemberType NoteProperty -Name 'Consumer' -Value (Get-WmiObject -Namespace root/subscription -Query "ASSOCIATORS OF {$($_.Filter)}")

    $event | Add-Member -MemberType NoteProperty -Name 'Binding' -Value $_
    $events += $event
  }
```

```
  # Huerfanos
  Get-WmiFilter | ?{ $events.Filter -notcontains $_ } | %{
    $event = New-Object -TypeName PSObject
    $event | Add-Member -MemberType NoteProperty -Name 'Filter' -Value $_
    $events += $event
  }

  Get-WmiConsumer | ?{ $events.Consumer -notcontains $_ } | %{
    $event = New-Object -TypeName PSObject
    $event | Add-Member -MemberType NoteProperty -Name 'Consumer' -Value $_
    $events += $event
  }

  $events
}
```

Bien, expliquémoslo por secciones. En primer lugar, utilizamos nuestra propia función **Get-WmiBinding** para obtener todos los enlaces, aplicando ya los posibles filtrados que el usuario decida.
Por cada enlace existente, vamos a aislar sus componentes (objetos) utilizando ASSOCIATORS OF.

- **ASSOCIATORS OF {$($_.Consumer)}** nos retornará el objeto Filtro asociado entre enlace y consumidor.
- **ASSOCIATORS OF {$($_.Filter)}** nos retornará el objeto Consumidor asociado entre enlace y filtro.

Por lo que, con una sola llamada, tenemos los tres objetos que forman una subscripción persistente en un solo nuevo objeto propio (creado con PSObject).

Bien, pero esto es funcional en un caso ideal, es decir, en un sistema nada impide crear un filtro, o un consumidor, pero no realizar el enlace final entre estos, es decir, es posible que existan objetos huérfanos, por lo que, para listar también estos objetos para un posible análisis, podemos utilizar ahora si, las otras dos funciones que hemos pre-programado Get-WmiFilter y Get-WmiConsumer.

B. Monitorización de registro de Windows

> Una de las tareas que puede resultar más útil, es monitorizar posibles cambios en ramas concretas del registro de Windows, como pueden ser aquellas asociadas a autoarranque, asociaciones de ficheros, etc...
>
> Vamos a crear tres subscripciones temporales, en las que se monitorice cualquier modificación en:
>
> - **El valor de:**
> - **HKCR\textfile\shell\open\command\(default)**
> - **Claves en:**
> - **HKLM\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\FirewallRules**

Dado que la monitorización del registro de Windows, puede ser algo útil en tareas defensivas (detección de software malicioso), vamos a realizar un desarrollo por pasos.

En primer lugar, veamos que clases evento tenemos disponibles para monitorización de eventos, lo cual lo podemos obtener utilizando las funciones vistas en los primeros puntos de este documento función **Get-WmiEventClass**:

```
Get-WmiEventClass -Filter Registry
```

Con ello veremos que disponemos de tres clases, para monitorizar cambios en:

- **Valores: RegistryValueChangeEvent**
- **Claves: RegistryKeyChangeEvent**
- **Ramas: RegistryTreeChangeEvent**

Estas tres clases hacen referencia a eventos extrínsecos, lo que podemos averiguar con la función **Get-WmiExtrinsicEvent**:

```
Get-WmiExtrinsicEvent -Filter Registry
```

Utilizando ahora la función **Get-WmiClassProperties**, podemos ver que tenemos las propiedades:

- **Hive: Hive sobre la que se genera el evento.**
- **RootPath: Rama sobre la que se genera el evento.**
- **KeyPath: Clave sobre la que se genera el evento.**
- **ValueName: Nombre del valor sobre el que se genera el evento.**

Una vez conocidas las clases y propiedades disponibles para la monitorización del registro de Windows, y antes de dar forma a las tres consultas necesarias, hemos de tener en cuenta ciertas obligaciones referentes a la cláusula WHERE[4] la cual ha de contener cada una de las propiedades de la clase evento utilizada, o en caso contrario se obtendría un error.

Ahora ya estamos en disposición de definir las consultas WQL que se tendrán que utilizar para monitorizar el registro en los puntos que se solicitan:

```
SELECT * FROM RegistryValueChangeEvent WHERE Hive="HKEY_CLASSES_ROOT" AND KeyPath="textfile\\Shell\\Open\\Command" AND ValueName=""

SELECT * FROM RegistryKeyChangeEvent WHERE Hive="HKEY_LOCAL_MACHINE" AND
KeyPath="SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\\FirewallRules\\"
```

Con las consultas para los filtros definidas, nos resta la acción a realizar una vez generado el evento, lo cual ha de ser generar una alerta en el sistema de eventos de Windows. Esto se puede realizar utilizando el cmdlet **Write-EventLog**. Este cmdlet nos proporciona todo lo necesario para generar la alerta, la cual, como cualquier evento, tendrá que tener definidos los siguientes datos:

- **Log: Fichero de log donde almacenar la alerta.**
- **Fuente: Nombre de la fuente de origen del evento.**
- **Tipo: Tipo de evento (Informativo, alerta, error, etc..).**
- **ID: Numero identificativo del evento.**
- **Categoría: Clave para categorización del evento.**
- **Mensaje: Datos descriptivos del evento.**

Para no corromper logs existentes, vamos a crear una nueva fuente llamada PoshWMIEventing. Esto se realizaría con el cmdlet **New-EventLog** y su parámetro -Source:

```
$existeFuente = Get-EventLog -LogName Application -Source PoshWMIEventing-EA SilentlyContinue
if (! $existeFuente) {
    New-EventLog -LogName Application -Source PoshWMIEventing
}
Write-EventLog -LogName Application -Source PoshWMIEventing -EntryType Warning -Category 0 -EventID 1 -Message $Mensaje
```

En este punto solo faltaría dar forma al mensaje a incluir en el evento, lo cual lo podríamos hacer de forma parecida a esta:

```
$Datos = $EventArgs.NewEvent
$Mensaje =  "Hive = " + $Datos.Hive + "`r`n"
$Mensaje += "RootPath = " + $Datos.RootPath + "`r`n"
$Mensaje += "KeyPath = " + $Datos.KeyPath + "`r`n"
$Mensaje += "ValueName = " + $Datos.ValueName + "`r`n"
```

Pues solo quedaría unir lo anterior en un único script, pero antes de hacerlo, vemos que vamos a tener que repetir todas las acciones casi idénticamente dos veces, por lo que intentemos crear una función que contenga aquellas acciones repetitivas para facilitar la resolución del ejercicio.

> **Solución:** **Monitorización temporal de registro de Windows**

```
function New-WmiRegistryMonitor {
    param(
        [string]$Query
    )
    $Accion = {
        $existeFuente = Get-EventLog -LogName Application -Source PoshWMIEventing
        if (! $existeFuente) {
            New-EventLog -LogName Application -Source PoshWMIEventing
        }
        $Datos = $EventArgs.NewEvent
        $Mensaje = "Hive = " + $Datos.Hive + "`r`n"
        $Mensaje += "RootPath = " + $Datos.RootPath + "`r`n"
        $Mensaje += "KeyPath = " + $Datos.KeyPath + "`r`n"
        $Mensaje += "ValueName = " + $Datos.ValueName + "`r`n"
        Write-EventLog -LogName Application -Source PoshWMIEventing -EntryType Warning -Category 0 -EventID 1 -Message $Mensaje
    }
    $Argumentos = @{
        Query = $Query
        Action = $Accion
    }
    Register-WmiEvent @Argumentos
}

New-WmiRegistryMonitor -Query 'SELECT * FROM RegistryValueChangeEvent WHERE Hive="HKEY_CLASSES_ROOT" AND KeyPath="textfile\\Shell\\Open\\Command" AND ValueName=""'

New-WmiRegistryMonitor -Query 'SELECT * FROM RegistryKeyChangeEvent WHERE Hive="HKEY_LOCAL_MACHINE" AND
KeyPath="SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\\FirewallRules\\"'
```

C. Dispositivos de almacenamiento

> El control de nuevos dispositivos externos es un punto de monitorización habitual, ya sea para detección de dispositivos no controlados por listas blancas, como para búsqueda de información sensible.
> Por lo que vamos a crear una subscripción temporal, la cual detecte la inserción de dispositivos de almacenamiento USB externos. Todo nuevo USB externo detectado debe recorrerse y localizar todos los documentos de Microsoft Office (Word, Excel o PowerPoint), cifrando su contenido.

Este ejercicio tiene tres puntos básicos de análisis:

- Detección de nuevos dispositivos USB,
- Método de cifrado
- Gestión de clave de cifrado.

Para detectar nuevos USB tenemos múltiples formas, tres de estas serían utilizando las clases:

- **Win32_PnpEntity**
- **Win32_USBHub**
- **Win32_VolumeChangeEvent**

Antes de decidir cuál de estas nos puede ser más conveniente, hemos de tener en cuenta dos factores, que tipo de dispositivos son detectados, y facilidad de llegar desde el evento al sistema de ficheros.

En primer lugar, veamos que tipo de información tenemos respecto a los dispositivos USB ya existentes en el sistema, utilizando la clase **Win32_USBControllerDevice** y **Win32_PnpEntity**.

> **Desarrollo:** **Inventario de dispositivos USB**

```
function Get-USBInventory {
    Get-WmiObject -Query "SELECT * FROM Win32_USBControllerDevice" | ForEach-Object {
        $dId = ($_.Dependent).Split("=")[1].Replace('"', '')
        (Get-WmiObject -Query "SELECT * FROM Win32_PnPEntity WHERE DeviceID = '$dId'")
    }
}

Get-USBInventory | Export-CSV -Path Test.csv -UseCulture -Encoding UTF8 -NoTypeInformation
```

Una vez hemos visto a grandes rasgos los tipos de dispositivos USB existentes, veamos cómo se podría definir un filtro para las tres clases evento:

```
SELECT * FROM __InstanceCreationEvent WITHIN 20 WHERE TargetInstance ISA 'Win32_PnPEntity'
SELECT * FROM __InstanceCreationEvent WITHIN 20 WHERE TargetInstance ISA 'Win32_USBHub'
SELECT * FROM Win32_VolumeChangeEvent WHERE EventType = 2
```

Realizando pruebas con las tres clases, y múltiples tipos de dispositivos veremos que:

- **Win32_PnpEntity:** Detecta todo tipo de dispositivos USB (Sticks, HDD Externos, Teléfonos y tablets), pero no disponemos de un enlace directo con el sistema de ficheros.
- **Win32_USBHub:** No detecta HDD Externos (SCSI).
- **Win32_VolumeChangeEvent:** Detecta Sticks y HDD Externos, no detecta teléfonos o tablets, pero si nos da una forma directa de acceso al sistema de ficheros.

Vamos a utilizar Win32_VolumeChangeEvent, ya que como solución intermedia nos da acceso a nuevos dispositivos de almacenamiento USB con acceso al sistema de ficheros, sabiendo que los teléfonos o tablets no serán detectados.

La estructura, sea cual sea la clase utilizada será similar a las siguientes:

> **Desarrollo:** **Depuración de clases evento**

```
Register-WmiEvent -Namespace root\CIMv2 -Query "SELECT * FROM __InstanceCreationEvent WITHIN 30 WHERE TargetInstance ISA 'Win32_PnPEntity'" -Action {
    Write-Host "USB Detectado, utiliza variable `$evento para depurar"
    $global:evento = $event
}
```

```
Register-WmiEvent -Namespace root\CIMv2 -Query "SELECT * FROM __InstanceCreationEvent WITHIN 30 WHERE TargetInstance ISA 'Win32_USBHub'" -Action {
    Write-Host "USB Detectado, utiliza variable `$evento para depurar"
    $global:evento = $event
}

Register-WmiEvent -Namespace root\CIMv2 -Query "SELECT * FROM Win32_VolumeChangeEvent WHERE EventType = 2"  -Action {
    Write-Host "USB Detectado, utiliza variable `$evento para depurar"
    $global:evento = $event
}
```

Para el cifrado de ficheros, .NET nos ofrece múltiples posibilidades[5], para el ejercicio vamos a utilizar AES, aunque Rijndael sería también una opción sólida para el cifrado.

Finalmente, con respecto a la gestión de claves tenemos dos opciones habituales, gestionar la clave de forma local (en el propio Script), o gestionarla por recursos externos. Vamos a utilizar la más simple a nivel de desarrollo, que es su gestión local.

> **Solución: Cifrado de ficheros en dispositivos USB**

```
Register-WmiEvent -Namespace root\cimv2 `
-Query "SELECT * FROM Win32_VolumeChangeEvent WHERE EventType = 2"  `
-Action {
    # Punto de búsqueda
    $Unidad = $event.SourceArgs.NewEvent.DriveName
    # IV y Clave para cifrado
    $RNGCrypto = New-Object System.Security.Cryptography.RNGCryptoServiceProvider
    $key = New-Object Byte[] 32
    $RNGCrypto.GetBytes($key)
    $iv = New-Object Byte[] 16
    $RNGCrypto.GetBytes($iv)
    # Clave en fichero
    Set-Content -Value $key -LiteralPath "$Unidad/ScriptKiddie.101" -Encoding byte
    # Objeto criptográfico para cifrado
    $AES = New-Object System.Security.Cryptography.AesManaged
    $AES.Key = $key
    $AES.iv = $iv
    $AES.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7
    $Encryptor = $AES.CreateEncryptor()
    # Recorrido y cifrado
    $Extensiones = @("*.doc*", "*.ppt*", "*.xls*")
    Get-ChildItem -Path $Unidad -Recurse -Include $Extensiones | %{
        if ((! ($_ -is [System.IO.DirectoryInfo])) -and (! $_.IsReadOnly)) {
            $file = $_.FullName
            $bytes = Get-Content $file -Encoding byte
            $encryptedData = $Encryptor.TransformFinalBlock($bytes, 0, $bytes.Length)
            [byte[]] $fullData = $AES.IV + $encryptedData
            Remove-Item -Force -Path $file
            Set-Content -Value $fullData -Path "$file.101" -Encoding byte
            $bytes = $Null
        }
    }
}
```

D. Persistencia ante reinicios

A la hora de conseguir persistencia en APT's, una de las vías más habituales es utilizar eventos WMI.

Como ejemplo vamos a crear subscripciones persistentes que creen un fichero en el directorio C:\Windows\Temp conteniendo la fecha de ejecución.

Estas subscripciones han de cubrir cuatro casuísticas:

- Al arrancarse la máquina.
- Hora concreta del día.
- Al generarse el log de evento con ID 4624.
- Al iniciarse alguno de los siguientes navegadores:
  - Microsoft Edge
  - Internet Explorer
  - Firefox
  - Chrome

Para este ejercicio, la acción a realizar es simple, solo se ha de escribir en un fichero, lo que se puede hacer tal que:

```
Get-Date | Out-File -Append C:\Windows\Temp\Persistencia.txt
```

La dificultad puede radicar en generar una consulta WQL válida para los casos solicitados, por lo que analicémoslos individualmente.

- El caso más sencillo es el que ha de ejecutarse al iniciarse un proceso concreto. Como hemos visto varias veces a lo largo del documento, la clase **Win32_Process** puede utilizarse para monitorizar los estados de los procesos, por lo que repitiendo la consulta vista en ejemplos anteriores sería suficiente.
  SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_Process' AND Targetinstance.Name = 'PROCESO.EXE'

- El tercer caso propuesto, se basa en identificar un evento de log concreto, para identificar que clase o clases podrían tener relación con eventos de log, podemos utilizar la función **Get-WmiClasses** filtrando con -Filter LogEvent
  Get-WmiClasses -Filter LogEvent

  Con lo que veremos que la clase adecuada es **Win32_NTLogEvent** y por tanto la consulta WQL quedaría como:
  SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_NTLogEvent' AND Targetinstance.EventCode = 4624

- El segundo caso, requiere conocer la hora del sistema, por lo que esto se puede realizar con **Win32_LocalTime** y sus propiedades Hour y Minute.
  SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_LocalTime' AND TargetInstance.Hour = 23 AND TargetInstance.Minute = 59 GROUP WITHIN 5

- Por último, el primer caso propuesto requiere conocer cuando una máquina se arranca, esto puede realizarse también de varias formas, en base al evento de log con ID 6009, o en base al tiempo que una máquina lleva en marcha. Esto último inicialmente desconocemos que clase contiene este tipo de datos, pero podemos intentar buscarlo con la funcion **Get-WmiClassProperties**
  Get-WmiClassProperties -Class * -Filter SystemUpTime

  Lo que nos dirá que este dato podemos conocerlo a través de la case **Win32_PerfFormattedDate_PerfOS_System**, y por tanto las posibles consultas serían:
  SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_NTLogEvent' AND Targetinstance.EventCode = 6009
  SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 240 AND TargetInstance.SystemUpTime < 325

Solución: Persistencia ante reinicios

```
$filtro1 = Set-WmiInstance -Namespace root\subscription -Class __EventFilter -Arguments @{
    EventNamespace = "root\CIMv2"
    QueryLanguage = "WQL"
    Query = "SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_Process' AND (Targetinstance.Name = 'chrome.exe' OR Targetinstance.Name =
'firefox.exe' OR Targetinstance.Name = 'iexplore.exe' OR Targetinstance.Name = 'MicrosoftEdge.exe')"
}

$filtro2 = Set-WmiInstance -Namespace root\subscription -Class __EventFilter -Arguments @{
    EventNamespace = "root\CIMv2"
    QueryLanguage = "WQL"
    Query = "SELECT * FROM __InstanceCreationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_NTLogEvent' AND Targetinstance.EventCode = 4624"
}

$filtro3 = Set-WmiInstance -Namespace root\subscription -Class __EventFilter -Arguments @{
    EventNamespace = "root\CIMv2"
    QueryLanguage = "WQL"
    Query = "SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_LocalTime' AND TargetInstance.Hour = 23 AND TargetInstance.Minute = 59
GROUP WITHIN 5"
}

$filtro4 = Set-WmiInstance -Namespace root\subscription -Class __EventFilter -Arguments @{
    EventNamespace = "root\CIMv2"
    QueryLanguage = "WQL"
    Query = "SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 240
AND TargetInstance.SystemUpTime < 325"
}

$Consumidor = Set-WmiInstance -Namespace root\Subscription -Class CommandLineEventConsumer -Arguments @{
    CommandLineTemplate = 'C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe -C "Get-Date|Out-File -Append C:\\Windows\\Temp\\Persistencia.txt"'
    RunInteractively='false'
}

Set-WmiInstance -Namespace root\Subscription -Class __FilterToConsumerBinding -Arguments @{
    Filter = $Filtro1
    Consumer = $Consumidor
}

Set-WmiInstance -Namespace root\Subscription -Class __FilterToConsumerBinding -Arguments @{
    Filter = $Filtro2
    Consumer = $Consumidor
}

Set-WmiInstance -Namespace root\Subscription -Class __FilterToConsumerBinding -Arguments @{
    Filter = $Filtro3
    Consumer = $Consumidor
}

Set-WmiInstance -Namespace root\Subscription -Class __FilterToConsumerBinding -Arguments @{
    Filter = $Filtro4
    Consumer = $Consumidor
}
```

8. Referencias

[1] https://msdn.microsoft.com/en-us/library/windows/desktop/aa394606(v=vs.85).aspx
[2] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_automatic_variables?view=powershell-5.1
[3] https://msdn.microsoft.com/en-us/library/system.management.automation.pseventargs(v=vs.85).aspx
[4] https://msdn.microsoft.com/en-us/library/aa3la89756(VS.85).aspx
[5] https://msdn.microsoft.com/es-es/library/system.security.cryptography(v=vs.110).aspx