Put Some Power in Your Shell

SANS Blue Team Summit 2020 (V3)
Presented by Don Murdoch, GSE #99 by Regent University and the Institute for Cyber Security in Virginia Beach, VA

www.blueteamhandbook.com - Yes, there is an update forthcoming....

NOTE: Much of the session content is taken from BTHb:INRE V3, which will be released “RSN”.
Speaker Intro

- Don Murdoch, GSE #99
- Author, Blue Team Handbook (Look for an update after this talk!)
- Day Job: Run the Cyber Range at the Institute for Cyber Security in Virginia Beach, VA collocated with Regent University
  - Digital shenanigans, sand, sun, and ocean spray!
- > 20 years in IT
- > 17 years in InfoSec
- Pioneered the phrase “Wild, Wild West of Academic Computing”
- Dependent on spell check and Komodo Dragon coffee
Session Agenda

- Tools and Techniques for on system analysis using PowerShell
- Critical Metrics for BlueTeam / IR success
- Setting up for Remote Analysis - Good, Bad, Unattractive, and Workgroups (yes, you do have data islands...)
- Collecting data from remote machines
- Learning to dance and falling gracefully
- Survey of remote management scripting tools
During an investigation ... what type of information you need? And how do you get it?

- RDP, SneakerNet, remote access, or WinRM with PowerShell??
- Record keeping with system and date stamped output files?
- Machine / OS
- User, Group, Current logins
- Network Configuration
- AutoStart Points
- Open files, physical drives, and mappings
- Running processes and Services
  - Hey - how about the command line?
- Active network connections
  - Let's kick that up a notch - What process is listening on the network?
- File system changes
  - Since when? Alternate Data Streams?
- Open files on a given server
- Removable media
- Windows Event Logs
  - What? USB collection not Enabled? Got an answer for that!
- Sysmon
- Domain information
Tools: What You Need for PowerShell Success

- PowerShell 5.1+
  - Windows 10 RSAT Toolset: Use Ashley McGlone’s “Install the Active Directory PowerShell Module on Windows 10” from the PowerShell Gallery

- Think about a better editor than PS ISE
  - MSFT Code
  - NotePad++

- A writable shar for dropping off data is helpful
Self Don-cu-mentation!
(You will thank me one day...)

```powershell
$T = Get-Date -UFormat "%Y%m%d.%H%M%S"
Get-Service | Select ServiceName,displayName,Status | export-csv -path C:\$T.get_service.csv -NoTypeInformation

# How about adding the system name?
$T = Get-Date -UFormat "%Y%m%d.%H%M%S"
Get-Service | Select ServiceName,displayName,Status | export-csv -path C:\$env:COMPUTERNAME.$T.get_service.csv -NoTypeInformation
```

<table>
<thead>
<tr>
<th>Mode</th>
<th>LastWriteTime</th>
<th>Length</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>2/18/2020 1:49 PM</td>
<td>14218</td>
<td>20200218.134941.get_service.csv</td>
</tr>
<tr>
<td>-a</td>
<td>2/18/2020 1:50 PM</td>
<td>14218</td>
<td>DONWIN10ASUS.20200218.135032.get_service.csv</td>
</tr>
</tbody>
</table>

```powershell
PS C:\Users\dmurdoc> type C:\\20200218.134941.get_service.csv
"ServiceName","DisplayNam..."Stopped"
```
Chances are You Don’t Want the Defaults

Step One: Find the cmdlet that looks like it will work, try it out.

Step Two: Get the full property list.

Step Three: ‘Select’ the properties that best support the case and provide meaningful data.

As needed use Export-CSV, Out-GridView, .
A Dip Into Windows Management Instrumentation (WMI)

- Late 90’s: WMI is Microsoft’s implementation of web-based business management standards (WBEM), the common information model (CIM) and the Distributed Management Task Force (DMTF). First in NT4 SP4*.

- Variety of ways to get detailed data:
  - Get-CIMInstance
  - Get-WMIObject with a class reference

- Other Notes
  - [https://gist.github.com/JeskeCode/b1a0e6731acc3a548246](https://gist.github.com/JeskeCode/b1a0e6731acc3a548246) - a PS Script!
  - Sapien’s WMI Explorer is a well liked tool ([https://www.sapien.com/](https://www.sapien.com/), $60)
  - * Don offers sincere apologies for the stress created by this flashback.
WMIExplorer.PS1
(we have a script for that...)
Got Example? Why yes, yes we do!
(and lets chose something security posture relevant)

```powershell
Get-WmiObject -Class Win32_QuestionFixEngineering
```

<table>
<thead>
<tr>
<th>Source</th>
<th>Description</th>
<th>HotFixID</th>
<th>InstalledBy</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONWIN10ASUS</td>
<td>Update</td>
<td>KB4534130</td>
<td>NT AUTHORITY\SYSTEM</td>
</tr>
<tr>
<td>DONWIN10ASUS</td>
<td>Update</td>
<td>KB4100347</td>
<td>NT AUTHORITY\SYSTEM</td>
</tr>
</tbody>
</table>

```
powershell> Get-HotFix
```

<table>
<thead>
<tr>
<th>Source</th>
<th>Description</th>
<th>HotFixID</th>
<th>InstalledBy</th>
<th>InstalledOn</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONWIN10ASUS</td>
<td>Update</td>
<td>KB4534130</td>
<td>NT AUTHORITY\SYSTEM</td>
<td>2/12/2020 12:00:00 AM</td>
</tr>
<tr>
<td>DONWIN10ASUS</td>
<td>Update</td>
<td>KB4100347</td>
<td>NT AUTHORITY\SYSTEM</td>
<td>1/27/2020 12:00:00 AM</td>
</tr>
</tbody>
</table>
Like Data. Different Display Results. Think About Long Term Data Consumability.

Raw Data is often available

Many Cmdlets Preformat
Common Pattern: Improve and Format Output

Use Shorthand Notation `@{n e=}` to Adjust Each Property as data with `$_` is Returned

```powershell
get-wmiobject win32_logicaldisk | select Name, FileSystem, VolumeName, @{n="Size / GB"; e={[math]:trunc($(Name).size / 1GB)}}, @{n="Free GB"; e={[math]:Truncate($(Name).Freespace / 1GB)}} | Format-Table
```

<table>
<thead>
<tr>
<th>Name</th>
<th>FileSystem</th>
<th>VolumeName</th>
<th>Size / GB</th>
<th>Free GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:</td>
<td>NTFS</td>
<td>Local Disk Windows</td>
<td>464</td>
<td>184</td>
</tr>
<tr>
<td>E:</td>
<td>NTFS</td>
<td>Detection Lab Build</td>
<td>465</td>
<td>385</td>
</tr>
</tbody>
</table>
What are some key system properties?

```
Get-CimInstance Win32_OperatingSystem | Format-List *
Get-CimInstance -ClassName Win32_OperatingSystem |
  Select-Object -Property Build*,OSType,Name,CSName,ServicePack*
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildNumber</td>
<td>17134</td>
</tr>
<tr>
<td>BuildType</td>
<td>Multiprocessor Free</td>
</tr>
<tr>
<td>OSType</td>
<td>18</td>
</tr>
<tr>
<td>Name</td>
<td>Microsoft Windows 10 Enterprise\Device\Harddisk2\Partition2</td>
</tr>
<tr>
<td>CSName</td>
<td>DONWIN10ASUS</td>
</tr>
<tr>
<td>ServicePackMajorVersion</td>
<td>0</td>
</tr>
<tr>
<td>ServicePackMinorVersion</td>
<td>0</td>
</tr>
</tbody>
</table>
Users and Groups

```powershell
Get-WmiObject Win32_UserAccount -filter { LocalAccount="True" and Disabled="False" } | Select-Object Domain,Name,SID,Disabled,LoginScript
```

<table>
<thead>
<tr>
<th>Domain</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONWIN10ASUS</td>
<td>dmurdoc</td>
</tr>
<tr>
<td></td>
<td>S-1-5-21-2780683337-698146951-3618690366-1001</td>
</tr>
<tr>
<td></td>
<td>False</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```powershell
Get-WmiObject Win32_Group -filter "LocalAccount=True" | Select-Object Domain,Name
```

<table>
<thead>
<tr>
<th>Domain</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONWIN10ASUS</td>
<td>Access Control Assistance Operators</td>
</tr>
<tr>
<td>DONWIN10ASUS</td>
<td>Administrators</td>
</tr>
</tbody>
</table>
Recent files by type?

```powershell
Get-ChildItem C:\ -Include *.exe -Recurse -ErrorAction Continue | Where-Object {$_.creationtime -gt '02/25/2020'} | Select-Object -Property CreationTime, Name, DirectoryName, FullName, Extension, Length, IsReadOnly | export-csv c:\IR\exe_files.csv
```

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>#TYPE Selected.System.IO.FileInfo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CreationTime</td>
<td>Name</td>
<td>DirectoryName</td>
</tr>
<tr>
<td>3/1/2020 8:38</td>
<td>USBDevview.exe</td>
<td>C:\Data\SANS\SEC555_Tactical_SIEM\2019_DEC_CD1</td>
</tr>
<tr>
<td>2/25/2020 16:06</td>
<td>appvcleaner.exe</td>
<td>C:\Program Files\Common Files\microsoft shared\ClickToRun</td>
</tr>
<tr>
<td>2/25/2020 16:06</td>
<td>AppVShNotify.exe</td>
<td>C:\Program Files\Common Files\microsoft shared\ClickToRun</td>
</tr>
</tbody>
</table>

---

**INSTITUTE FOR CYBERSECURITY**
Registry contents?

```powershell
Remove-PSDrive M
New-PSDrive -Name "M" -PSProvider "Registry" -Root "HKLM:\SOFTWARE\Mozilla"
Get-ChildItem -recurse "M:" | select Name, ValueCount
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Used (GB)</th>
<th>Free (GB)</th>
<th>Provider</th>
<th>Root</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td></td>
<td></td>
<td>Registry</td>
<td>HKEY_LOCAL_MACHINE\SOFTWARE\Mozilla</td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td>HKEY_LOCAL_MACHINE\SOFTWARE\Mozilla\Firefox</td>
</tr>
<tr>
<td>ValueCount</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td>HKEY_LOCAL_MACHINE\SOFTWARE\Mozilla\Firefox\Extensions</td>
</tr>
<tr>
<td>ValueCount</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td>HKEY_LOCAL_MACHINE\SOFTWARE\Mozilla\Firefox\TaskBarIDs</td>
</tr>
<tr>
<td>ValueCount</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Current Logon Sessions

```
Get-CimInstance -ClassName Win32_LogonSession | select StartTime, AuthenticationPackage, LogonID, LogonType Format-Table
```

<table>
<thead>
<tr>
<th>StartTime</th>
<th>AuthenticationPackage</th>
<th>LogonID</th>
<th>LogonType</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/18/2020 12:37:51 PM NTLM</td>
<td>199976441</td>
<td>199976410</td>
<td>2</td>
</tr>
<tr>
<td>2/18/2020 12:37:51 PM NTLM</td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

Forky asks a question...
But Wait - Can’t I Easily Find the Currently Logged on User(s): Local System?

```powershell
Get-CimInstance -ClassName Win32_ComputerSystem | Select-object Name, UserName
```

<table>
<thead>
<tr>
<th>Name</th>
<th>UserName</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONWIN10ASUS</td>
<td>DONWIN10ASUS\dmurdoc</td>
</tr>
</tbody>
</table>
So Let's Connect the Dots!

```powershell
Get-WmiObject -Class Win32_LogonSession |
foreach {
    $ap = $_.AuthenticationPackage
    $ltype = $_.LogonType
    $data = $_
    $id = $data.__RELPATH -replace "\\", "\"
    $q = "ASSOCIATORS OF {$id} WHERE ResultClass = Win32_Account"
    Get-WmiObject -Query $q |
    select @N="User";E={$($_.Caption)}},
    @N="AuthPkg";E={$ap},
    @N="LogonType";E={$ltype},
    @N="LogonTime";E={$data.ConvertToDateTime($data.StartTime)}
}
```

<table>
<thead>
<tr>
<th>User</th>
<th>AuthPkg</th>
<th>LogonType</th>
<th>LogonTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>DONWIN10ASUS\dmurdoc</td>
<td>NTLM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DONWIN10ASUS\dmurdoc</td>
<td>NTLM</td>
<td></td>
<td>2 2/18/2020 12:37:51 PM</td>
</tr>
<tr>
<td>DONWIN10ASUS\dmurdoc</td>
<td>NTLM</td>
<td></td>
<td>2 2/18/2020 12:37:51 PM</td>
</tr>
</tbody>
</table>
Basic Process Information
(Build an output file, count the results, and show the first ten)

```
Get-WmiObject Win32_Process | 
select ProcessId, ParentProcessId, CreationDate, ProcessName, Name, ExecutablePath, CommandLine | 
export-csv -path C:\IR\procinfo.csv -NoTypeInformation

$data = Get-Content C:\IR\procinfo.csv
($data).count
$data | select -first 10
```
Dig a Little Deeper with Win32_Process

```
WmiObject Win32_Process | select ProcessId, ParentProcessId, CreationDate, ProcessName, Name, ExecutablePath, CommandLine

<table>
<thead>
<tr>
<th>ProcessId</th>
<th>11512</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentProcessId</td>
<td>876</td>
</tr>
<tr>
<td>CreationDate</td>
<td>20200218123751.339373-300</td>
</tr>
<tr>
<td>ProcessName</td>
<td>svchost.exe</td>
</tr>
<tr>
<td>Name</td>
<td>svchost.exe</td>
</tr>
<tr>
<td>ExecutablePath</td>
<td>c:\windows\system32\svchost.exe</td>
</tr>
<tr>
<td>CommandLine</td>
<td>c:\windows\system32\svchost.exe -k unistacksvcgroup -s CDPUserSvc</td>
</tr>
</tbody>
</table>
```
Process Relationship to Parent

```powershell
$IDtoParentIDH = $Null
$IDtoParentIDH = @{}
Get-WmiObject win32_process | ForEach {
    $IDtoNameH.add( $_.ProcessId, $_.ProcessName )
    $IDtoParentIDH.add( $_.ProcessId, $_.ParentProcessID )
    $IDtoOwnerH.add( $_.ProcessId, $_.getowner().user )
    $IDtoCmdLineH.add( $_.ProcessId, $_.CommandLine )
}
foreach ($prid in $IDtoNameH.GetEnumerator() | Sort Key)
{
    "Process ID: " + $prid.key
    "Process Name: " + $prid.value
    "Process Owner: " + $IDtoOwnerH[$prid.Key]
    "Process Cmd: " + $IDtoCmdLineH[$prid.name]
    "Parent ID: " + $IDtoParentIDH[$prid.Key]
    "Parent Owner: " + $IDtoOwnerH[ $IDtoParentIDH[$prid.Key] ]
    "Parent Cmd: " + $IDtoCmdLineH[ $IDtoParentIDH[$prid.Key] ]
}
```

Process ID: 124
Process Name: svchost.exe
Process Owner: SYSTEM
Process Cmd: C:\WINDOWS\system32\svchost.exe -k DcomLaunch -p
Parent ID: 876
Parent Owner: SYSTEM
Parent Cmd:
Always Check your Code!
Processes Part Two!

Not a MSFT app? How about running from temp?

```
Get-Process | Where { $_.company -notmatch "Microsoft" -and !$_.Company -and $_.path -and $_.path -match "Program Files" } | select name, path
```

```
NVDisplay.Container C:\Program Files\NVIDIA Corporation\Display.NvContainer\NVDisplay.Container.exe
NVDisplay.Container C:\Program Files\NVIDIA Corporation\Display.NvContainer\NVDisplay.Container.exe
vmware-hostd C:\Program Files (x86)\VMware\VMware Workstation\vmware-hostd.exe
```

```
Get-Process | Where { $_.path -match "temp" } | select starttime, name, path, company
```

```
<table>
<thead>
<tr>
<th>StartTime</th>
<th>Name</th>
<th>Path</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/18/2020 7:51:23 PM</td>
<td>procexp64</td>
<td>C:\Users\dmurdod\AppData\Local\Temp\procexp64.exe</td>
<td>Sysinternals - <a href="http://www.sysinternals.com">www.sysinternals.com</a></td>
</tr>
</tbody>
</table>
```
Network Addresses

```powershell
Get-NetIPAddress | select-object AddressFamily, IPAddress, InterfaceAlias | Format-Table
```

<table>
<thead>
<tr>
<th>IPv6</th>
<th>VMware Network Adapter VMnet8</th>
<th>VMware Network Adapter VMnet1</th>
</tr>
</thead>
<tbody>
<tr>
<td>fe80::a927:dad2:6560:b5ba%2</td>
<td>Ethernet</td>
<td></td>
</tr>
<tr>
<td>IPv6 fe80::b852:82a4:8cb7:98c%11</td>
<td>Loopback Pseudo-Interface 1</td>
<td></td>
</tr>
<tr>
<td>IPv6 fe80::88d6:146f:8ef4:d986%19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPv6 ::1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPv4 169.254.222.122</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Forky asks a question...
Extracting Data from Windows Event Logs
Here’s the story...

Older: Get-EventLog

New: Get-WinEvent

Properties Interface: Event specific extraction; requires you to know field order

XML Interface: Requires you to discover each event’s message schema

In either case - getting fact data requires you to learn positional field values
XML Based Approach
Part 1

```
1 # Prompts for creds
2 $cred = Get-Credential DonWin10Asus\dmurdoc
3 # Grab the events
4 $Event = Get-WinEvent -computerName DonWin10Asus -Credential $cred
5 -FilterHashtable @{Logname='Security';Id=4776} -MaxEvents 1
6 # View the event properties.
7 $Event | Format-List *
8 # View the array of message body values (prop names missing...)
9 # But the property names are missing.
10 $Event.Properties
11 # Convert the event to XML
12 $eventXML = [xml]$Event.ToXml()
13 # Drill down through the XML to the message goodness
14 # Ah ha! This is what we want.
15 $eventXML.Event.EventData.Data
16 # You have to index each data element to access it.
17 $eventXML.Event.EventData.Data[0].name
18 $eventXML.Event.EventData.Data[0].'#text'

Message : The computer attempted to validate the credentials for an account.

Authentication Package: MICROSOFT_AUTHENTICATION_PACKAGE_V1_0
Logon Account: dmurdoc
Source Workstation: DONWIN10ASUS
Error Code: 0x0

Id : 4776
Version : 0
Qualifiers :
Level : 0
Task : 14336
Opcode : 0
Keywords : -9214364837600034816
RecordId : 1204147
ProviderName : Microsoft-Windows-Security-Auditing
ProviderId : 54849625-5478-4994-a5ba-3e3b0328c30d
LogName : Security
```
XML Based Approach
Part 2

```powershell
# Prompt for creds
$cred = Get-Credential

# Grab the events from a DC
$Events = Get-WinEvent -ComputerName DonWin10Asus -Credential $cred
    -FilterHashtable @{Logname='Security';Id=4776}

# Parse out the event message data
ForEach ($Event in $Events) {
    # Convert the event to XML
    $eventXML = [xml]$Event.ToXml()
    # Iterate through each one of the XML message properties
    For ($i=0; $i -lt $eventXML.Event.EventData.Data.Count; $i++) {
        # Append these as object properties
        Add-Member -InputObject $Event -MemberType NoteProperty -Name $eventXML.Event.EventData.Data[$i].name
        -Value $eventXML.Event.EventData.Data[$i].'#text'
    }
}

# Example Output
$Events | Export-Csv .\events.csv
$Events | Select-Object * | Out-GridView
```
Output Please!
(Check out the Grid View!)

Source Workstation: DONWIN1...
Error Code: 0xC0000064
The computer attempted to validate th...

Authentication Package: MICROSOFT...
Logon Account: dmurdoch
Source Workstation: DONWIN1...
Error Code: 0xC0000064
The computer attempted to validate th...

<table>
<thead>
<tr>
<th>PackageName</th>
<th>TargetUserName</th>
<th>WorkstationName</th>
<th>Status</th>
<th>Id</th>
<th>Version</th>
<th>Qualifiers</th>
<th>Level</th>
<th>Task</th>
<th>Opcode</th>
<th>Keywords</th>
<th>RecordId</th>
</tr>
</thead>
<tbody>
<tr>
<td>MICROSOFT_</td>
<td>dmurdoch</td>
<td>DONWIN1</td>
<td>0</td>
<td>4,776</td>
<td>0</td>
<td>0</td>
<td>14,336</td>
<td>0</td>
<td>-9,214,364,837,6...</td>
<td>1,204,147</td>
<td></td>
</tr>
<tr>
<td>MICROSOFT_</td>
<td>dmurdoch</td>
<td>DONWIN1</td>
<td>0</td>
<td>4,776</td>
<td>0</td>
<td>0</td>
<td>14,336</td>
<td>0</td>
<td>-9,218,868,437,2...</td>
<td>1,204,143</td>
<td></td>
</tr>
<tr>
<td>MICROSOFT_</td>
<td>dmurdoch</td>
<td>DONWIN1</td>
<td>0</td>
<td>4,776</td>
<td>0</td>
<td>0</td>
<td>14,336</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOW I KNOW
OUT GRID KUNG FU
Properties? Yep, we got ‘em.

Most of the items wanted are baked into a ‘message’, which is an interpreted view of the data as processed by a DLL.

```powershell
get-WinEvent -FilterHashTable @{LogName='security'; id=4624} -MaxEvents 1 | Get-Member | format-list
```

<table>
<thead>
<tr>
<th>TypeName</th>
<th>System.Diagnostics.Eventing.Reader.EventLogRecord</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Message</td>
</tr>
<tr>
<td>MemberType</td>
<td>NoteProperty</td>
</tr>
<tr>
<td>Definition</td>
<td>string Message=An account was successfully logged on.</td>
</tr>
</tbody>
</table>

**Subject:**
- Security ID: S-1-5-18
- Account Name: DONWIN10ASUS$
- Account Domain: WORKGROUP
- Logon ID: 0x3E7

**Logon Information:**
- Logon Type: 5
- Restricted Admin Mode: -
- Virtual Account: No
- Elevated Token: Yes
Review the event of note:
4624 = authentication at the end node.
Fields are positional, exposed through ‘properties’ (Hey! You get to add your own useful column names!)

```
```

<table>
<thead>
<tr>
<th>TimeCreated</th>
<th>Target User Sid</th>
<th>Target User Name</th>
<th>Process Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/24/2020 5:16:36 PM S-1-5-18</td>
<td></td>
<td>SYSTEM</td>
<td>C:\windows\System32\services.exe</td>
</tr>
<tr>
<td>2/24/2020 5:10:36 PM S-1-5-18</td>
<td></td>
<td>SYSTEM</td>
<td>C:\windows\System32\services.exe</td>
</tr>
<tr>
<td>2/24/2020 5:01:36 PM S-1-5-18</td>
<td></td>
<td>SYSTEM</td>
<td>C:\windows\System32\services.exe</td>
</tr>
<tr>
<td>2/24/2020 4:58:13 PM S-1-5-21-2780683337-698146951-3618690366-1001</td>
<td></td>
<td>dmurdoc</td>
<td>-</td>
</tr>
<tr>
<td>2/24/2020 4:55:42 PM S-1-5-18</td>
<td></td>
<td>SYSTEM</td>
<td>C:\windows\System32\services.exe</td>
</tr>
</tbody>
</table>
Application: User Logins
Oh, and ... How many?

```
$logins = get-eventlog -logname security -instanceid 4624 |
where { $_.replacementstrings[5] -eq 'dmurdoc' } |
select-object timegenerated,@{name='username'; expression={$_.replacementstrings[5]}}
write-host "# Logins: $(logins.count)"
$logins | out-gridview
```

PS C:\\SYSTEM32> $logins =
Where { $_.ReplacementStrings[5] -eq 'dmurdoc' }
Select-Object TimeGenerated,@{Name="UserName"; Expression={$_.ReplacementStrings[5]}}
Write-Host "# Logins: $(logins_COUNT)"
$logins | Out-GridView

# Logins: 34
PS C:\\SYSTEM32>
Let's put these concepts together.

Sysmon?
Long Tail Analysis?
Yep.
The Code
(da Vinvi and Dan Brown didn’t write it though)

```perl
$Hash = @{}
(entries = Get-WinEvent -filterhashtable @{logname="Microsoft-Windows-Sysmon/Operational";id=1} |
  %{$_.Properties[4].Value}
  # group-object Value
  foreach ($l in $entries)
  {
    # write-output $l
    if ($Hash[$l] -eq $null ) {
      $Hash[$l] = 1;
    } else {
      $Hash[$l]++;
    }
  }
$Hash.getenumerator() | sort -Descending -Property value | ForEach-Object {
  $msg = '{0} {1}' -f $_.value, $_.Key
  write-output $msg
  
```
Output is in Long Tail Order
Acronyms: LTA, MFO, LFO (No UFO or MUFON)

How Long?
Use Get-Date!

Cattle mutilations are up.
We want to “Sweep the Enterprise”. Metric?

- Success - machine swept and we got meaningful results (this is the desired state)
- Failure - machine online, no access
- Failure - machine should be online but not responsive
- Failure - machine online, WinRM not responsive
- Failure - machine online, something else

We want to do this efficiently

We need to avoid some *serious* pitfalls
WinRM: What’s that?  
(Hint: PoSH Hotness, maybe 6-7 years ago...) 

- Several Reasons for using WinRM - IR focus
  - Rich Dataset available for Collection - PowerShell has access to WMI, COM, .NET, Windows API.
  - Analysis object-based pipeline. Oh, and “Export-CSV” will keep the Excel users really happy.
  - Performance (see next slide)

- Several Reasons for using WinRM - Operations focus
  - MSFT direction for past several years
  - Agentless
  - CoSt is expressed in “Sweat Equity”, which for the MBA’s in the crowd mean there is no CapEx, only OpEx which means you have an opportunity cost trade off to contend with

- What’s the catch?
  - There is op overhead
  - Security - if the adversary can gain elevated access, they can access machines that are running WinRM ... but we do have countermeasures....
Sweep the Enterprise - Enable WinRM
(Yes, you can collect from data islands)

- **Defaults**
  - WinRM service will be set to ‘auto start’, a listener is registered, and a Windows firewall “allow” rule enabled
  - Older than 2008 R2? Run “Enable-PSRemoting -Confirm:$FALSE” & reboot

- **Default Security:** WinRM uses Kerberos; connecting user must be a member of the local Administrators group
  - Standalone: *Will not work if the LAN is set to Public!*

```powershell
PS C:\WINDOWS\system32> Enable-PSRemoting -Confirm:$false
WinRM is already set up to receive requests on this computer.
WinRM has been updated for remote management.
WinRM firewall exception enabled.
Configured LocalAccountTokenFilterPolicy to grant administrative rights remotely to local users.
```
GPO (Briefly)

- **Computer Configuration > Policies > Administrative Templates: Policy definitions > Windows Components > Windows Remote Management (WinRM) > WinRM Service.**
  - Set Allow remote server management through WinRM to “Enabled” and IP source(s)

- **Preferences > Control Panel Settings > Services.**
  - Add WinRM, set as “Start Service”

- **Computer Configuration > Policies > Administrative Templates: Policy definitions > Network > Network Connections > Windows Firewall > Domain Profile.** Inbound rules include:
  - Windows Firewall: “Allow inbound remote administration exception,” with IP address into the field called “Allow unsolicited incoming messages from these IP addresses”
  - Windows Firewall: Allow ICMP exception, allow an inbound echo request

- **Example Walkthroughs:**
  - [https://support.infrasightlabs.com/article/how-to-enable-winrm-on-windows-servers-clients/](https://support.infrasightlabs.com/article/how-to-enable-winrm-on-windows-servers-clients/)
Troubleshooting

- On the machine itself, check for essential setup
  - Get-PSSessionConfiguration
  - Must be a local admin)
- If the service isn’t running...
  - Start it the PowerShell way: Set-Service -Name WinRM -ComputerName VICTIM -startup Automatic (and you may need to reboot)
  - Then Check status: Get-Service | Where-Object {$_._name -eq "WinRM"}
- Firewall refusing connections?
  - Linux: nc -z -w1 <IP or host name> 5985;echo $?
  - Windows: Test-WSMan -ComputerName <IP or host name>
- SSL via HTTPS mismatches?
- Is this thing on?
  - winrm quickconfig
  - Enable-PSRemoting
What is *Natively* Available?

CIMsessions: Uses an object locally that a remote system can connect to. Doesn’t use PSRemoting. Uses older WS-MAN or DCOM.
Commands that support ‘-ComputerName’

```powershell
$computers = get-content c:\temp\mycomputers.txt
$computers
get-ciminstance win32_useraccount -ComputerName $computers |
sort PSComputerName,Disabled,Name |
format-Table -groupby PSComputerName -property PSComputerName,Caption,Name,Disabled
```

<table>
<thead>
<tr>
<th>PSComputerName</th>
<th>Caption</th>
<th>Name</th>
<th>Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>vTrainee003</td>
<td>VTRAINEE003\sysAdmin</td>
<td>sysAdmin</td>
<td>False</td>
</tr>
<tr>
<td>vTrainee003</td>
<td>VTRAINEE003\win10-Template</td>
<td>win10-Template</td>
<td>False</td>
</tr>
<tr>
<td>vTrainee003</td>
<td>VTRAINEE003\Administrator</td>
<td>Administrator</td>
<td>True</td>
</tr>
<tr>
<td>vTrainee003</td>
<td>VTRAINEE003\DefaultAccount</td>
<td>DefaultAccount</td>
<td>True</td>
</tr>
<tr>
<td>vTrainee003</td>
<td>VTRAINEE003\defaultuser0</td>
<td>defaultuser0</td>
<td>True</td>
</tr>
<tr>
<td>vTrainee003</td>
<td>VTRAINEE003\Guest</td>
<td>Guest</td>
<td>True</td>
</tr>
<tr>
<td>Name</td>
<td>Synopsis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote</td>
<td>Describes how to run remote commands in Windows PowerShell.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote_Disconnected_Sessions</td>
<td>Explains how to disconnect from and reconnect to a PSSession</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote_FAQ</td>
<td>Contains questions and answers about running remote commands</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote_Jobs</td>
<td>Describes how to run background jobs on remote computers.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote_Output</td>
<td>Describes how to interpret and format the output of remote commands.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote_Requirements</td>
<td>Describes the system requirements and configuration requirements for</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote_Troubleshooting</td>
<td>Describes how to troubleshoot remote operations in Windows PowerShell.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote_Variables</td>
<td>Explains how to use local and remote variables in remote</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_CimSession</td>
<td>Describes a CimSession object and the difference between CIM sessions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_PSSessions</td>
<td>Describes Windows PowerShell sessions (PSSessions) and explains how to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_PSSession_Details</td>
<td>Provides detailed information about Windows PowerShell sessions and the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote_Disconnected_Sessions</td>
<td>Explains how to disconnect from and reconnect to a PSSession</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Session_Configurations</td>
<td>Describes session configurations, which determine the users who can</td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Session_Configuration_files</td>
<td>Describes session configuration files, which can be used in a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOPIC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>about_Remote_Requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SHORT DESCRIPTION**
Describes the system requirements and configuration requirements for running remote commands in Windows PowerShell.

**LONG DESCRIPTION**
This topic describes the system requirements, user requirements, and resource requirements for establishing remote connections and running remote commands in Windows PowerShell. It also provides instructions for configuring remote operations.
One to One Remoting
*(Hey, go check out system XYZ)*

- Enter-PSSession - ComputerName VICTIM
  - On that next system, run commands if they are local
  - You can use "localhost" -> check with ‘hostname’, don’t flip at the flashing cmd prompt
  - Prompt is updated with the target system name

- Exit-PSSession

- Alternate Cred:
  - $credvalue = Get-Credential -Credential domain\adminperson
  - PSSession - ComputerName VICTIM -Credential $credvalue

```
PS C:\WINDOWS\system32> Enter-PSSession -ComputerName localhost
[localhost]: PS C:\Users\dmurdoc\Documents> Get-Service
Status           Name                                      DisplayName
------           ------                                      ---------------
Running          AdobeARMservice                          Adobe Acrobat Update Service
Stopped          AJRouter                                  AllJoyn Router Service
```
One to Many: Invoke-Command

- ComputerName
  - Comma Separated list
  - ScriptBlock - What you want to run
  - FilePath - Hey, lets run this script
  - Credential
  - ThrottleLimit - Add more, or make list smaller

- By Default
  - Will communicate to 32 systems
  - Returns data only, not methods

- PSComputerName
  - Added to return output so you know which target provided information
Single Command, Exact Targets

```powershell
Get-ADDomain
Invoke-Command -ComputerName Trainer-c1, Trainer-C1 -ScriptBlock
{ Get-CimInstance -ClassName Win32_Bios }
```

SMBIOSBIOSVersion : 1.4.4
Manufacturer        : Dell Inc.
Name                : 1.4.4
SerialNumber        : 7BCGQK2
Version             : DELL - 1072009
PSComputerName      : Trainer-c1

SMBIOSBIOSVersion : 1.4.4
Manufacturer        : Dell Inc.
Name                : 1.4.4
SerialNumber        : 7BCGQK2
Version             : DELL - 1072009
PSComputerName      : Trainer-c1
CSV? Yes.

```
1 "ConfuzerName" | Out-File -FilePath c:\temp\systems.csv
2 Get-ADComputer -Filter 'Name -like "Trainer-C*"' |
3 Select-Object -ExpandProperty Name | Out-File -FilePath c:\temp\systems.csv -Append
4
5 -Invoke-Command -ComputerName ( Import-Csv -Path C:\temp\systems.csv |
6 Select-Object -ExpandProperty ConfuzerName ) -FilePath "C:\temp\EnableUSB_Log.psl"
```

```
PS C:\Users\trainer01> type C:\temp\systems.csv
ConfuzerName
TRAINER-C1
TRAINER-C2
TRAINER-C3
TRAINER-C4
```
Simple Filtered List from AD, Single Command

```
Get-ADComputer -Filter 'Name -like "Trainer-*"' | Select-Object -ExpandProperty Name | Out-File -FilePath c:\temp\systems.txt
Invoke-Command -ComputerName (Get-Content -Path C:\temp\systems.txt) ` -FilePath "C:\temp\Code_TO_run.ps1" | select PSComputerName, ID, Name | Format-Table
```
Simple List in a CSV, run a script
(hey - lets do something useful around USB’s)
Invoke Methods on Other Systems

```
Invoke-Command -Computer vTrainee003 -scriptblock ' {Get-WmiObject Win32_OperatingSystem | Invoke-WMIAction -Name Reboot }
```

Note: Doing this on every machine may cause some excitement. YMMV.
Sweep the Enterprise - Step One

- **Q-IR:** How many systems
  - Are present?
  - Have a DNS answer?
  - Logged in “recently”?

- **Q-PoSH:**
  - Get list of “enabled computers” and filter the list somehow?
  - Check the last login date
How Big is your AD?

**Round One:**
- How many users?
- How many groups?
- How many computers?
Improving The Data - Make it more “Actionable”

```
$hash_lastLogonTimestamp = @{Name="LastLogonTimestamp"; 
Expression={[(datetime)::FromFileTime($_.LastLogonTimestamp)]}}
$hash_pwdLastSet = @{Name="pwdLastSet";Expression={[(datetime)::FromFileTime($_.pwdLastSet)]}}
$DaysInactive = 30
$time = (Get-Date).Adddays(-($DaysInactive))
get-adcomputer -Filter {{(Name -like "*rainee*") -or (Name -like "*trainer*") } 
    -properties LastLogonTimestamp,pwdLastSet | 
    Sort-Object LastLogonTimestamp -descending | 
    select samaccountname, $hash_lastLogonTimestamp, hash_pwdLastSet
```

1600’s? Wow!
Sweep the Enterprise - Step Two

- **Q-IR:**
  - Are those systems responsive to reachable?
  - How do we define “reachable”?

- **Q-PoSH:**
  - Foreach system in AD, send (how many???) ICMP packets?
  - Are there errors?

- **A-PoSH:**
  - For 2251 systems, it takes 1 hr, 55 min, 20 sec. to send 2 ICMP packets for all systems that are online in AD. (Oh - wait - we forgot to count failures!)
  - ICMP Responsive:
    - 932 nonresponsive, 1318 responsive or ... 41% “failure”
    - IS THAT GOOD, BAD, or INDIFFERENT?
Sweep the Enterprise - How many respond to ICMP? (and what are the errors)?

- **Q-IR:**
  - How many responsive, known systems can run a PoSH script?
  - How can we incorporate what we’ve learned about response time and be more efficient?

```powershell
e$ipresult = Test-Connection -computername $pc.name -Count 1 -ErrorAction SilentlyContinue
if ($ipresult) {
    $iptest = "System is responsive"
    $status_Results = "Up"
    $status_IP = ($ipresult.IPV4Address).IPAddressToString
    $systemup = $systemup + 1
} else {
    $iptest = "System is non responsive"
    $status_Results = "Down"
    $status_IP = "X.X.X.X"
    $systemdown = $systemdown + 1
}
write-host "$pc.name -> $fqdn -> $status_IP"
```
Will WinRM Work?

```powershell
$investigation_target = "FredPC"
Test-WSMan -ComputerName $investigation_target
```

```
Test-WSMan : <f:WSManFault xmlns:f="http://schemas.microsoft.com/wbem/wsman/1/wsmanfault" Code="2150859046" Machine="RU211743.regent.edu"><f:Message>WinRM cannot complete the operation. Verify that the specified computer name is valid, that the computer is accessible over the network, and that a firewall exception for the WinRM service is enabled and allows access from this computer. By default, the WinRM firewall exception for public profiles limits access to remote computers within the same local subnet. </f:Message></f:WSManFault> At line:3 char:1 + Test-WSMan -ComputerName $investigation_target + ------------------------------------------------- + CategoryInfo    : Invalidoperation: (FredPC:String) [Test-WSMan], InvalidoperationException + FullyQualifiedErrorId : WsManError,Microsoft.WSMan.Management.TestWSmanCommand
```
Let's try that again ... When things go right

```powershell
$investigation_target = "env-share"

Try {
    Test-WSMan -ComputerName $investigation_target -ErrorAction Stop
}

Catch {
    "Error trying to connect to " + $investigation_target
    $Error
}
```
Detect when things to wrong: Use a STOP Error Action and a try/catch

Try-catch in Powershell is different to most other languages. There are 2 types of errors in Powershell, terminating and non-terminating. By default, a non-terminating error will not trigger try/catch handling. To force powershell to catch the error no matter what type it is, append -ErrorAction Stop to the command itself, like Test-WSMan or InvokeCommand.
Where to go From Here

- Kansa
- Jrotenberger/Powershell-IR-Scripts
- mgreen27/Invoke-LiveResponse
- Eric Conrad’s DeepBlueCLI - Mission specific analysis tool
- JavelinNetworks/IR-Tools - AD Specific analysis tools
- gfoss/PSRecon
Thank you!