# WINDOWS POWERSHELL LOGGING CHEAT SHEET - Win 7/Win 2008 or later

This "**Windows PowerShell Logging Cheat Sheet**" is intended to help you get started setting up basic and necessary PowerShell (Windows Management Framework) command and command line logging. This list includes some very common items that should be enabled, configured, gathered and harvested for any Log Management program. Start with these settings and add to it as you understand better what is in your logs and what you need.

Malware Archaeology

**Sponsored by:**

**LOG-MD**
Discover it

## DEFINITIONS:

ENABLE: Things you must do to enable logging to start collecting and keeping events.

CONFIGURE: Configuration that is needed to refine what events you will collect.

GATHER: Tools/Utilities that you can use locally on the system to set or gather log related information – AuditPol, WEvtUtil, Find, etc.

HARVEST: Events that you would want to harvest into some centralized Event log management solution like syslog, SIEM, Splunk, etc.

RESOURCES: Places to get information on PowerShell Logging

- PS 2,3,4 Command Line Logging - http://technet.microsoft.com/en-us/library/hh847796.aspx
- PowerShell Transcript information - https://technet.microsoft.com/en-us/library/hh849687.aspx
- PS 4 - https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html
- PS 4 & 5 - https://blogs.msdn.microsoft.com/powershell/2015/06/09/powershell-the-blue-team - KEY for PS 5
- https://www.blackhat.com/docs/us-14/materials/us-14-Kazanciyan-Investigating-Powershell-Attacks-WP.pdf
- http://learn-powershell.net/2014/08/26/more-new-stuff-in-powershell-v5-extra-powershell-auditing
- http://www.redblue.team/2016/01/powershell-traceless-threat-and-how-to.html?showComment=1464099315089#c3589963557794199352
- https://www.carbonblack.com/wp-content/uploads/2016/04/Cb-Powershell-Deep-Dive-A-United-Threat-Research-Report-1.pdf

## INFORMATION:

1. Why Enable and Configure PowerShell logging? PowerShell, which is found in the "Windows Management Framework" is the future way Microsoft will have us administer Windows. The Command line as we know it is going away and PowerShell will be taking over. Why is this important? PowerShell provides access to the .NET Framework which provides access to API calls that attackers can take advantage of and exploit and avoid Anti-Virus and other security controls in the process. PowerShell can be used to exploit a system with little noise or indicators in the logs unless properly enabled and configured to gather the PowerShell execution details. If you do not start enabling PowerShell logging options mentioned in this cheat sheet, attackers will be able to utilize and exploit your systems and do it quietly without additional file drops or noise generated by traditional malware and attacks. It is crucial to begin properly logging PowerShell to avoid this growing exploitation option. To understand what kind of PowerShell exploitation is available and being used, follow the following projects:
   - PowerSploit, PowerShell Empire, PowerTools, MetaSploit, Social Engineering Toolkit (SET) and PoshSec

# WINDOWS POWERSHELL LOGGING CHEAT SHEET - Win 7/Win 2008 or later

## INFORMATIONAL – SECURITY WARNING:

1. The ability to bypass your PowerShell controls is a concern and why it is crucial to alert on these bypass attempts as an indication that you are being attacked. If this behavior is occurring normally in your environment, work with your people to fix the issue, remove, and prohibit it from happening. Fortunately, if you follow the "***Windows Logging Cheat Sheet***" and enable "***Process Creation***" and the "***Command Line Logging***" registry tweak, you will see ***Event ID 4688*** where the "***Process Command Line***" shows the command executing the PowerShell bypass in many, if not most cases.
2. Execution Policy Bypass: The Execution Policy that you set can be bypassed by anyone with malicious intent. Alert on the execution of the following:
   a. –ExecutionPolicy bypass and/or
   b. –Ep bypass or –Exec bypass
   c. –ExecutionPolicy unrestricted
3. Profile Bypass: The profile(s) you set to configure PowerShell when each session is launched can also be bypassed. Alert on the execution of the following:
   a. –noprofile or -nop
4. These commands are often joined in malicious activity and can be in any form of the following:
   a. powershell.exe -executionpolicy bypass –noprofile –windowstyle hidden –file malicious.ps1
   b. powershell.exe -NonInteractive -WindowStyle Hidden -Ep bypass –nop –File "malicious.ps1"
   c. powershell.exe –e ZQBjAGgAbwAgACcAWQBvAHUAIABhAHIAZQAgAHAAdwBuAGUAZAAhACcA == (encoded)

## ENABLE:

1. LOCAL LOG SIZE: Increase the size of your local PowerShell logs. Don't worry, you have plenty of disk space, CPU is not an issue with today's systems.
   a. Applications and Services Logs – 'Windows PowerShell' log set to 500,000KB or larger
   b. Applications and Services Logs / Microsoft-Windows – 'PowerShell/Operational' log set to 500,000KB or larger
2. LOCAL SECURITY POLICY: No settings needed
3. GROUP POLICY: You can control all the PowerShell settings in Group Policy
   - ExecutionPolicy – must be set to RemoteSigned if using a default profile (profile.ps1) PS v2-4
   - ScriptBlock – Capture PowerShell execution details Event ID 4104 on PowerShell 5 Win 7, 2008 Server or later
   - ModuleLoad - Capture PowerShell execution details Event ID 4104 on PowerShell 5 Win 7, 2008 Server or later
   - Log script block execution start / stop events – Do NOT set, generates a lot of noise and too many log entries
4. REGISTRY SETTINGS:
   - HKCU/HKLM\SOFTWARE\Policies\Microsoft\Windows\PowerShell",REG_SZ,"ExecutionPolicy"
     - RemoteSigned
   - HKCU/HKLM\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell","Path"
   - HKCU/HKLM \SOFTWARE\Policies\Microsoft\Windows\PowerShell\
     - ModuleLogging, REG_DWORD, EnableModuleLogging, 1 — PowerShell 5
     - NEW ModuleLogging/ModuleNames REG_SZ, * * (value and data both *) — PowerShell 5
     - ScriptBlockLogging, REG_DWORD, EnableScriptBlockLogging, 1 — PowerShell 5
     - ScriptBlockLogging, REG_DWORD, EnableScriptBlockInvocationLogging, 1 — PowerShell 5
       - This is VERY noisy, do not set in most environments, or seriously test first (4105 & 4106)
     - Transcription, REG_DWORD, EnableInvocationHeader, 1 — PowerShell 5
     - Transcription, REG_DWORD, EnableTranscripting, 1 — PowerShell 5
     - Transcription, REG_SZ, OutputDirectory (Enter Path) — PowerShell 5

# WINDOWS POWERSHELL LOGGING CHEAT SHEET - Win 7/Win 2008 or later

**CONFIGURE:**

1. PowerShell Transcript: For PowerShell versions 2, 3, 4 & 5. You can record a PowerShell transcript of everything entered during a PowerShell session using a default profile. Transcripts will be recorded in one or more files depending on your settings. To make transcripts work for every user opening a PowerShell session, add it to the Default Profile (profile.ps1). Make sure the location of the transcript file is writeable by all users. It can be stored on a file share or local location, avoid using the default location in the Users directory to avoid easy guessing by attackers. This feature does not create any Windows log entries, but provides another avenue to capture command line execution.
   a. Start-transcript -path "<some_drive>:\<some_location>\PowerShell_transcript.txt" -force -noClobber –append
   b. Set whether you want to append the file, overwrite, etc.
   c. Be sure to rotate the logs if you do not overwrite
   d. The default location for the transaction logs are: $Home\My Documents\PowerShell_transcript.<time-stamp>.txt
2. You may also Start and Stop the transcripts as needed within your scripts with ***Start-Transcript*** or ***Stop-Transcript***

**CONFIGURE:**:For all versions of Windows

1. WEvtUtil: Use this utility to configure your log settings or configure them in Group Policy
   a. WevtUtil gl "Windows PowerShell" – List settings of the PowerShell Log
   b. WevtUtil sl "Windows PowerShell" /ms:512000000 – Set the PowerShell Log size to the number of bytes
   c. WevtUtil sl "Windows PowerShell" /rt:false – Overwrite as needed
   d. WevtUtil gl "Microsoft-Windows-PowerShell/Operational" – List settings of the PowerShell Log
   e. WevtUtil sl " Microsoft-Windows-PowerShell/Operational " /ms:512000000
   f. WevtUtil sl " Microsoft-Windows-PowerShell/Operational " /rt:false – Overwrite as needed
2. For more history in your logs, consider making the log size 1GB

**CONFIGURE:** For PowerShell 2, 3 and 4 (does NOT apply to PowerShell 5):

1. Default Profile: In order to capture the command line of a Windows PowerShell session for all users on the system, you will need to create a default profile. Create a file named '***profile.ps1***' and save it to the following location:

- C:\Windows\System32\WindowsPowerShell\v1.0 (yes, it is always 1.0 for any version of PowerShell)

With the following variable values:

| | |
|---|---|
| a. $LogCommandHealthEvent = $true | Command Line Details |
| b. $LogCommandLifecycleEvent = $true | Command Line Details |
| c. $LogPipelineExecutionDetails = $true | Module Loading (within scripts) |
| d. $PSVersionTable.PSVersion | Shows the version of PowerShell installed |

Every time a PowerShell session is launched, the commands entered in PowerShell will be recorded in the "***Windows PowerShell***" log. Be sure to watch ALL profile locations for new or modified ***profile.ps1*** used for persistence.

**NOT ALL POWERSHELL LOGGING IS EQUAL:**

PowerShell version 2 thru 4 on Windows 7 and 2008 Server is different in logging options and behavior than Windows 8.1, 10 and Server 2012. Options were added to PowerShell 4 and especially 5 that retired many things found in PS version 2 thru 4. So be aware of the settings you are using on what OS and PowerShell version.

# WINDOWS POWERSHELL LOGGING CHEAT SHEET - Win 7/Win 2008 or later

## CONFIGURE:

1. PowerShell Versions and OS: The ability to perform advanced logging of PowerShell is limited to certain operating systems and the versions of PowerShell used. Basic PowerShell logging is available for all versions of Windows 7, Server 2008 and above, but advanced auditing is limited to PowerShell 4 and 5. The following lists the OS, log(s), and Event ID's for each operating system and PowerShell version to monitor.
   - Windows 7 and Server 2008 and above:
     - PowerShell version 2 thru 4, "***Windows PowerShell***" log – Event ID's 400, 500, 501 and 800
   - Windows 8.1 and Server 2012 and above:
     - PowerShell version 3 and 4, "***Windows PowerShell***" log - Event ID's 400, 500, 501 and 800
     - ***"Microsoft-Windows-PowerShell/Operational"*** log – Event ID 4104
   - Windows 7 and Server 2008 and above:
     - PowerShell version 5, "***Windows PowerShell***" log - Event ID's 200, 400, 500 and 501
     - "***Microsoft-Windows-PowerShell/Operational***" log – Event ID 4100, 4103 and 4104

   ***Note:*** There are other 4105 & 4106 events, but they are of little value to security monitoring and VERY noisy!

## CONFIGURE:

1. REGISTRY AUDIT: To help you catch malicious activity trying to alter your PowerShell configuration, audit the registry where these settings are stored. Open Regedit and select the registry keys you want to monitor for changes.
   a. Right-Click a Key – Permissions – Advanced – Auditing – Add – EVERYONE – (check names), OK.
   b. Apply onto – THIS KEY AND SUBKEYS (or what you want)
   c. Select 'Set Value', 'Create Subkey', 'Create Link', 'Delete', 'Write DAC' & 'Write Owner' to start
2. POWERSHELL KEYS TO AUDIT:
   a. HKCU & HKLM\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell
      i. ExecutionPolicy
   b. HKLM\SYSTEM\CurrentControlSet\services\eventlog\Windows PowerShell
      i. MaxSize
      ii. Retention
   c. HKLM\SOFTWARE\Policies\Microsoft\Windows\PowerShell\
      i. All subkeys and values

## GATHER:

1. Reg.exe: Use this utility to query the registry and check the settings
   a. ***Changes to Services Keys***
      i. reg query "HKLM\System\CurrentControlSet\Services\eventlog\Windows PowerShell"
   b. ***Changes to PowerShell Policy***
      i. reg query "HKLM\Software\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell"
   c. Query a value of a Key
      i. reg query "HKLM\SOFTWARE\Policies\Microsoft\Windows\PowerShell"

# WINDOWS POWERSHELL LOGGING CHEAT SHEET - Win 7/Win 2008 or later

## GATHER:

1. WEvtUtil: Use this utility to query your logs
    a. WevtUtil qe “Windows PowerShell” – query the Security Log for events
        i. Lots of flags here so read help “WevtUtil -?”
        ii. /c:5 = Read 5 events
        iii. /rd:true = newest events first
        iv. /f:text = format text, also can do XML
    b. ***PowerShell executed*** - WevtUtil qe “Windows PowerShell” /q:"*[System[(EventID=501)]]" /c:5 /rd:true /f:text >Parsed\%computername%_PS_Cmds_Executed_Win7.log
2. Filtering Log Results: Use this method to filter lines within the logs
    a. ***PowerShellCommand Name executed*** - WevtUtil qe “Windows PowerShell” /q:"*[System[(EventID=500)]]" /c:5 /rd:true /f:text | find /I “CommandName”
    b. ***PowerShellCommand Line executed*** - WevtUtil qe “Windows PowerShell” /q:"*[System[(EventID=500)]]" /c:5 /rd:true /f:text | find /I “CommandLine”
    c. ***PowerShellCommand Line for Cmdlet and Scripts executed*** - WevtUtil qe "Windows PowerShell" /q:"*[System[(EventID=501)]]" /c:1000 /rd:true /f:text | findstr "Logged CommandLine Cmdlet Script"
3. For PowerShell 4 and 5 Event ID 4104. These logs are large in content
    a. ***Anything that is a “Get-“ call*** - WevtUtil qe "Microsoft-Windows-PowerShell/Operational" /q:"*[System[(EventID=4104)]]" /c:1000 /rd:true /f:text | findstr /i "Get-"
    b. ***Anything that is an “iex“ (invoke execution) call*** - WevtUtil qe "Microsoft-Windows-PowerShell/Operational" /q:"*[System[(EventID=4104)]]" /c:1000 /rd:true /f:text | findstr /i "iex"
4. Get-EventLog: Use this PowerShell module to query your logs. There too many options to list here, but here a couple to get you started.
    a. get-eventlog -logname “Windows PowerShell” -computername <your_systemname>
    b. get-eventlog -logname "Windows PowerShell" -computername <your_systemname> | where {$_.eventID -eq 400}

***Note:*** For EventID 400, look for ***HostApplication*** to see what executable called PowerShell.

## POWERSHELL VERSIONS:

NEW

1. PowerShell now comes in many forms and places. There is the engine that is installed on the operating system, such as version 2,3, 4, 5, and now version 6, but also a version that comes with Microsoft SQL Server. The following is a list of binaries that PowerShell uses and the version:
    a. PowerShell ver 2, 3, 4 & 5 – Found in “C:\Windows\System32\Windows PowerShell” (powershell.exe)
    b. PowerShell ver 6 which is installed as an application under “C:\Program Files” (pwsh.exe)
    c. PowerShell for SQL Server which will be found where you install MS SQL Server (sqlps.exe)
    d. A couple Dlls that can be called from a maliciously crafted binary “not powershell” (e.g. begin with System.Management.Automation*.dll, more on this below)

### HARVEST:

2. FIREWALL: Windows Filtering Platform - Watch for Inbound and Outbound connections – ***the Windows Firewall does not need to be used, logs will still be collected if enabled***
    a. This is the noisiest of all Events. Generating easily 9,000 - 10,000 events per hour per system
    b. Storage is required to utilize this event
    c. 5156 – Message=The Windows Filtering Platform has permitted a connection. Look for:
        i. Direction:, Source Address:, Source Port:, Destination Address: & Destination Port:
        ii. Specifically items where the 'Application Name' is ***PowerShell.exe***

### `HARVEST:

POWERSHELL EXPLOITATION: Monitoring PowerShell is much different than other logging due to what needs to be enabled and configured and what is logged and how PowerShell uses DLL's and uses API calls. You have to look for combination of suspicious calls to detect malicious behavior. Event IDs 4100, 4103 & 4104 will be your best bet to catch malicious activity in PS 5 or use Event ID 500 and 800 in PS 4 and lower and of course what initially executed on the command line with Event ID 4688 New Process Creation with Process Command Line enabled in all versions of Windows.

1. Monitor for these Dll's being called by an executable other than PowerShell.exe or PowerShell_ISE.exe (use Sysmon or AppLocker)
    a. System.Management.Automation.Dll or System.Management.Automation.ni.Dll
    b. System.Reflection.Dll
2. Monitor for calls to WMI, they are not necessarily malicious, but could indicate WMI is being used in an attack
    a. Invoke-WMIMethod
    b. Get-WMIObject
3. In order to detect PowerShell DLL's being called by something other than PowerShell.exe or PowerShell_ISE.exe, you should consider adding one of the following services to Windows to gather more intelligence:
    a. Sysmon – Sysinternals utility to record additional information about what Processes called what DLL's monitor all Images loading the PowerShell DLL's "***System.Management.Automation.Dll***", "***System.Management.Automation.ni.Dll***" or "***System.Reflection.Dll***"
        i. Event ID 7 ***ImageLoaded*** in the "***Microsoft-Windows-Sysmon/Operational***" log
        ii. https://technet.microsoft.com/en-us/sysinternals/sysmon
    b. Windows Logging Service (WLS) – A replacement syslog agent that records additional information like the modules loaded by a process.
        i. https://digirati82.com/wls-information/
        ii. http://energy.gov/sites/prod/files/cioprod/documents/Splunkified_-_the_Next_Evolution_of_Log_Analysis_-_Green_and_McCord.pdf

### MONITOR:

1. PROFILES: Locations that profile.ps1 can be stored should be monitored for new profiles or changes since these can be used for malicious persistence. These are all easily bypassed with the '-exec bypass' in attacks, so basically worthless today
    a. AllUsersAllHosts - %windir%\System32\WindowsPowerShell\v1.0\profile.ps1
    b. AllUsersAllHosts (WoW64) - %windir%\SysWOW64\WindowsPowerShell\v1.0\profile.ps1
    c. AllUsersCurrentHost - %windir%\System32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1
    d. AllUsersCurrentHost (ISE) - %windir%\System32\WindowsPowerShell\v1.0\Microsoft.PowerShellISE_profile.ps1
    e. AllUsersCurrentHost (WoW64) – %windir%\SysWOW64\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1
    f. AllUsersCurrentHost (ISE - WoW64) - %windir%\SysWOW64\WindowsPowerShell\v1.0\Microsoft.PowerShellISE_profile.ps1
    g. CurrentUserAllHosts - %homedrive%%homepath%\[My ]Documents\profile.ps1
    h. CurrentUserCurrentHost - %homedrive%%homepath%\[My ]Documents\Microsoft.PowerShell_profile.ps1
    i. CurrentUserCurrentHost (ISE) - %homedrive%%homepath%\[My ]Documents\Microsoft.PowerShellISE_profile.ps1

MONITOR:

6. POWERSHELL MODULES: The following is a list of PowerShell module calls that you should monitor for executing using either Sysmon, WLS or the enhanced module logging of PowerShell. This is by no means a complete list.

Any “***Set-***“, “***Get-***“, “***Invoke-***“, “***Out-***”, “***Write-***“ and all PowerShell arguments can be shortened like “***iex***”, “***ex***”, etc so be careful and look for these too. Then filter out your normal modules and look for malicious ones such as:

- Set-ExecutionPolicy, Set-MasterBootRecord
- Get-WMIObject, Get-GPPPassword, Get-Keystrokes, Get-TimedScreenshot, Get-VaultCredential, Get-ServiceUnquoted, Get-ServiceEXEPerms, Get-ServicePerms, Get-RegAlwaysInstallElevated, Get-RegAutoLogon, Get-UnattendedInstallFiles, Get-Webconfig, Get-ApplicationHost, Get-PassHashes, Get-LsaSecret, Get-Information, Get-PSADForestInfo, Get-KerberosPolicy, Get-PSADForestKRBTGTInfo, Get-PSADForestInfo, Get-KerberosPolicy
- Invoke-Command, Invoke-Expression, iex, Invoke-Shellcode, Invoke--Shellcode, Invoke-ShellcodeMSIL, Invoke-MimikatzWDigestDowngrade, Invoke-NinjaCopy, Invoke-CredentialInjection, Invoke-TokenManipulation, Invoke-CallbackIEX, Invoke-PSInject, Invoke-DllEncode, Invoke-ServiceUserAdd, Invoke-ServiceCMD, Invoke-ServiceStart, Invoke-ServiceStop, Invoke-ServiceEnable, Invoke-ServiceDisable, Invoke-FindDLLHijack, Invoke-FindPathHijack, Invoke-AllChecks, Invoke-MassCommand, Invoke-MassMimikatz, Invoke-MassSearch, Invoke-MassTemplate, Invoke-MassTokens, Invoke-ADSBackdoor, Invoke-CredentialsPhish, Invoke-BruteForce, Invoke-PowerShellIcmp, Invoke-PowerShellUdp, Invoke-PsGcatAgent, Invoke-PoshRatHttps, Invoke-PowerShellTcp, Invoke-PoshRatHttp, Invoke-PowerShellWmi, Invoke-PSGcat, Invoke-Encode, Invoke-Decode, Invoke-CreateCertificate, Invoke-NetworkRelay,
- EncodedCommand, New-ElevatedPersistenceOption, wsman, Enter-PSSession, DownloadString, DownloadFile
- Out-Word, Out-Excel, Out-Java, Out-Shortcut, Out-CHM, Out-HTA, Out-Minidump, HTTP-Backdoor, Find-AVSignature, DllInjection, ReflectivePEInjection, Base64, System.Reflection, System.Management
- Restore-ServiceEXE, Add-ScrnSaveBackdoor, Gupt-Backdoor, Execute-OnTime, DNS_TXT_Pwnage, Write-UserAddServiceBinary, Write-CMDServiceBinary, Write-UserAddMSI, Write-ServiceEXE, Write-ServiceEXECMD,
- Enable-DuplicateToken , Remove-Update, Execute-DNSTXT-Code, Download-Execute-PS, Execute-Command-MSSQL, Download_Execute, Copy-VSS, Check-VM, Create-MultipleSessions, Run-EXEonRemote, Port-Scan, Remove-PoshRat, TexttoEXE, Base64ToString, StringtoBase64, Do-Exfiltration, Parse_Keys, Add-Exfiltration, Add-Persistence, Remove-Persistence, Find-PSServiceAccounts, Discover-PSMSSQLServers, Discover-PSMSExchangeServers, Discover-PSInterestingServices, Discover-PSMSExchangeServers, Discover-PSInterestingServices
- Mimikatz, powercat, powersploit, PowershellEmpire, Payload, GetProcAddress,
- And any other commands found in any PowerShell exploit kits

## MONITOR:

3. POWERSHELL MODULES: The following is a list of PowerShell module calls from some other sources. This is by no means a complete list.

- Greg Foss of LogRhythm list of malicious PowerShell calls. This list will produce some normal noise
  - https://gist.github.com/gfoss/2b39d680badd2cad9d82
  - https://logrhythm.com/blog/powershell-command-line-logging/

Another way to look at detecting bad PowerShell Fu is to focus on a few known suspicious calls to cut down on the noise and trigger on things you should investigate such as;

- .Download – Look for downloads using PowerShell
- Net.WebClient – Look for downloads using PowerShell
- IEX – Short for invoke-expression
- Invoke-expression – Calls an expression
- Invoke-command – Calls a command
- ICM – Short for invoke command
- .invoke – will trigger on all invoke calls
- WMI anything – PowerShell being called by WMI or WMI using PowerShell

4. ENCODED MODULES: Encoding is one way PowerShell commands can be hidden from the user and certain logs. You should understand which Event ID's and fields are involved to collect and report on the right things as a part of your IR investigation or log management.

- Event ID 400 or 600 – All versions of PowerShell in the "***Windows PowerShell"*** log and the field 'HostApplication' will display the encoded bits used such as;
  - HostApplication=powershell.exe -EncodedCommand VwByAGkAdABlAC0ASABvAHMAdAAgAC0ATwBiAGoAZQBjAHQAIAAiAEgAZQBsAGwAbwAsACAAdwBvAHIAbABkACEAIgA7AA==
- If Command line logging is enabled, Event ID 4688 of the Security log will show you the full command that was executed with the encoded blob similar to above. To learn how to enable Command Line logging, read the "***Windows Logging Cheat Sheet***"
- Event ID 4104 for PowerShell 4 and 5 in the "***Microsoft-Windows-PowerShell/Operational***" log you will see un-encoded from the sample above;
  - Write-Host -Object "Hello, world!";

  If you enable ScriptBlockLogging as referenced on Page 2, PowerShell v4 and v5 will decode it for you in the 4104 events. For all versions of PowerShell , the encoded blurb will show up in the HostApplication field of Event ID's 400 and 600 of the "***Windows PowerShell***" log that you can then isolate and then un-encode. These are the events you will want to focus collecting.

  You can isolate the encoded portion and decode it with a module within your log management solution like Splunk offers with the Base64 Splunk App.

### HARVEST:

1. LOG CLEAR: Watch for log clear messages
    a. 104 – SYSTEM Log – The “Windows PowerShell” or “PowerShell Operational” log was cleared

### HARVEST:

1. REGISTRY: Monitor PS Keys for Add, Changes and Deletes. Setting auditing on the Specific keys is required (See the “***Windows Registry Auditing Cheat Sheet***”).
    a. 4657 – SECURITY log – A Registry value was modified

### HARVEST:

1. PROCESSES: Watch for a Process to start and call other processes
    a. 4688 – SECURITY Log – “***New Process Name***” (powershell.exe, PWSH.exe & SQLPS.exe), look for Creator Process ID to link what process launched what other processes
    b. 4688 – SECURITY Log – What “***Process Command Line***” was executed for any ‘powershell.exe’ ,’ PWSH.exe’ or ‘SQLPS.exe’ events
    c. Filter out normal events for your environment

### HARVEST:

1. WMI and PowerShell: Monitor for WMI calling PowerShell. This could indicate a remote attack using WMI.
    a. Security Log
    b. 4688 event
    c. The Parent process will be WmiPrvSE.exe
        i. The Child will be one of the PowerShell executables (powershell.exe, pwsh.exe, or sqlps.exe)

**Example:** LOG-MD-Pro Parent > Child report of WMImplant querying system information

| Parent | Child |
|---|---|
| C:\Windows\System32\wbem\WmiPrvSE.exe | C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe |

MONITOR:

5. SPLUNKAGE: The following are some ideas to catch suspicious PowerShell behavior in your logs. These examples are for Splunk, but you may be able to convert the logic in these to suit your log management solution.

Focus on a few known suspicious calls to cut down on the noise and trigger on things you should investigate such as;

- **Security Log (4688) - Look for 'Execution Policy bypass' and 'No Profile' executions.** The idea here is to look for execution bypasses. This can be obfuscated, but would get picked up by another query below by the use of odd characters and/or 'ticks'.
  ```
  index=windows source="WinEventLog:Security" (EventCode=4688) (powershell* AND -executionpolicy) OR (powershell* AND -ep) OR
  (powershell* AND bypass) OR (powershell* AND -noprofile) OR (powershell* AND -nop) NOT (“*\\some_expected_thing*”) | eval
  Message=split(Message,".") | eval Short_Message=mvindex(Message,0) | table _time, host, Account_Name, Process_Command_Line,
  New_Process_Name, New_Process_ID, Creator_Process_ID, Short_Message
  ```
- **Security Log (4688) - Look for Execution Policy bypass and No Profile executions – less noisy.** This removes a couple full name bypasses that are commonly used to catch the more suspicious shortened ones.
  ```
  index=windows source="WinEventLog:Security" (EventCode=4688) (powershell* AND -ep) OR (powershell* AND -nop) NOT
  (“*\\some_expected_thing*”) | eval Message=split(Message,".") | eval Short_Message=mvindex(Message,0) | table _time, host,
  Account_Name, Process_Command_Line, New_Process_Name, New_Process_ID, Creator_Process_ID, Short_Message
  ```
- **PowerShell Operational Log (4100) - Look for Execution Policy bypass and No Profile executions – less noisy.** This removes a couple full name bypasses that are commonly used to catch the more suspicious shortened ones.
  ```
  index=powershell source="WinEventLog:Microsoft-Windows-PowerShell/Operational" (EventCode=4100) (-ep AND bypass) OR (-exp AND
  bypass) OR (-exec AND bypass) OR ("-nop") NOT (*something_normal*)  | table host, ComputerName, User, Host_Version, PS_Version,
  Host_Application
  ```
- **PowerShell Operational Log (4104) PS v4-5- Look for PowerShell modules greater than 1000 characters**. The idea here is to trigger on large modules that may be malicious filtering our known good modules.
  ```
  index=powershell source="WinEventLog:Microsoft-Windows-PowerShell/Operational" EventCode=4104 NOT ("*Program
  Files\\SplunkUniversalForwarder\\etc\\apps*" OR "*SplunkUniversalForwarder\\bin\\splunk-powershell-common.ps1*" OR
  "*var\\log\\splunk\\splunk-powershell*" ) | eval Cmd_Length=len(Message) | where Cmd_Length > 1000 | table _time, host,
  Cmd_Length, Message
  ```
- **Security Log (4688) - Look for PowerShell using a large amount of odd characters (ticks ' and Percent %).** The idea here is to trigger on the use of ticks or special characters that are used to obfuscate or break keyword searches. Tweak to your needs or new attack methods.
  ```
  index=windows LogName="Security" EventCode=4688 NOT ("*some_expected_thing*") | eval Orig_Command=Process_Command_Line |
  eval Clean_Command_Line=Process_Command_Line | eval Obfuscations=Process_Command_Line | rex field=Obfuscations mode=sed
  "s/[a-zA-Z0-9]//g" | rex field=Clean_Command_Line mode=sed "s/[']//g" | eval Tick_Count = mvcount(split(Obfuscations,"'"))-1 | eval
  Pct_Count = mvcount(split(Obfuscations,"%"))-1 | table _time host, Orig_Command, Clean_Command_Line, Obfuscations, Tick_Count,
  Pct_Count | where Tick_Count > 2
  ```
- **Windows PowerShell Log (400) PS v2-5 - Look for PowerShell using a large amount of odd characters (e.g. ticks ' and Percent %).** The idea here is to trigger on the use of ticks or special characters that are used to obfuscate or break keyword searches. Tweak to your needs or new attack methods.
  ```
  index=powershell LogName="Windows Powershell" EventCode=400 (Net.WebClient) | eval MessageA=split(Message,"Details:") | Eval
  Short_Message=mvindex(MessageA,1) | eval Clean_Host_Application=Host_Application | eval Obfuscations=Host_Application
  | rex field=Obfuscations mode=sed "s/[a-zA-Z0-9]//g" | rex field=Clean_Host_Application mode=sed "s/[']//g" | eval Tick_Count =
  mvcount(split(Obfuscations,"'"))-1 | eval Pct_Count = mvcount(split(Obfuscations,"%"))-1 | table _time, host, ComputerName,
  Host_Application, Clean_Host_Application, Obfuscations, Tick_Count, Pct_Count | where Tick_Count > 2
  ```

MONITOR:

More examples to catch web calls using PowerShell, Base64 script blocks and using Sysmon to see PowerShell Dll attacks.

**Note:** For Splunk you may have to add field extractions to your transforms to get these field names to show up, or split and/or RegEx the Message or Details of the logs to create the field names. So these queries may not work in Splunk without that effort.

- **Security Log (4688) – Look for PowerShell web downloads.** The idea here is to trigger on the download requests. This can be obfuscated, but would get picked up by another query above by the use of odd characters and/or 'ticks'.
  ```
  index=windows LogName=Security EventCode=4688 (".Download" OR "Net.WebClient") | table _time, host, ComputerName,
  Account_Name, Account_Domain, New_Process_Name, Process_Command_Line
  ```
- **PowerShell Operational Log (4104) PS v4-5 – Look for PowerShell web downloads.** This can be obfuscated, but would get picked up by another query below by the use of odd characters and/or 'ticks'.
  ```
  index=powershell LogName="microsoft-windows-powershell/operational" EventCode=4104 (".Download" OR "Net.WebClient") NOT
  ("something_normal") | eval MessageA=split(Message,"):") | eval MessageB=mvindex(MessageA,1) | eval
  MessageB=split(MessageB,"ScriptBlock ID:") | eval Message_Block=mvindex(MessageB,0) | table _time, host, ComputerName, User,
  TaskCategory, Message_Block
  ```
- **Windows PowerShell Log (400) PS v2-5 – Look for PowerShell web downloads.** This can be obfuscated, but would get picked up by another query below by the use of odd characters and/or 'ticks'.
  ```
  index=powershell LogName="Windows Powershell" (EventCode=400) (.Download OR "Net.WebClient") NOT
  ("*\\some_expected_thing*") | eval MessageA=split(Message,"Details:") | Eval Short_Message=mvindex(MessageA,1) | table _time
  ComputerName, host, PS_Version, Engine_Version, Host_Application, Command_Line
  ```
- **Windows PowerShell Log (400) PS v2-5– Look for Base64 module data, isolate the Base64 for conversion.** The idea here is to look for encoded Base64 script blocks that may need to be decoded to know what they do.
  - **Note:** Splunk has a Base64 module that can decode the chunk if isolated, and this will isolate the base64 portion for you. There are also multiple ways (one = vs. two ==), but one equal will result in a lot of false positives.
  - **Note:** In a 4104 event the encoded script block will be decoded for you in PS v4 and V5 if the proper logging is enabled.

  ```
  index=powershell LogName="Windows Powershell" (EventCode=400 ) | eval MessageA=split(Message,"Details:") | Eval
  Short_Message=mvindex(MessageA,1) | rex field=Host_Application "(?<Base64_Data>.[a-zA-Z0-9]+={2})" | table _time ComputerName,
  host, PS_Version, Engine_Version, Host_Application, Base64_Data, Command_Line | where NOT isnull(Base64_Data)
  ```
- **Sysmon Log (7) - Look for PowerShell System.Management.Automation loading NOT by PowerShell.** The idea here is to look for 'Not PowerShell' type attacks that use a malicious binary to then call PowerShell Dll's and never launch PowerShell.exe of PowerShell_ISE.
  - **Note:** Sysmon is a free add-on service by Microsoft that creates enhanced logging.
    - https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon
  - **Note:** AppLocker in Audit mode can also be used monitor Dll loads of a system.

  ```
  index=Sysmon LogName=Microsoft-Windows-Sysmon/Operational EventCode=7 system.management.automation*.dll NOT
  (Image="*Windows\\System32\\WindowsPowerShell\\v1.0\\powershell*.exe") | eval Hashes=split(Hashes,"=") | eval
  SHA1_Hash=mvindex(Hashes,1) | dedup SHA1_Hash | table _time, host, Image, ImageLoaded, Signed, Signature, SHA1_Hash
  ```