PowerShell Logging

PowerShell Logging - Attacker Goals

Logging from an attacker's perspective:

1. A way to get caught
   a. Many of your actions will be logged. You must be aware of what your footprint is. Possible courses of action:
      i. Evade logging - perform actions that don’t get logged
      ii. Circumvent logging - prevent logging from occurring, ideally, in a way that also doesn’t create alerts.
2. A way to confuse defenders
   a. i.e. fill the logs with bogus data
3. A potential C2 channel
   a. Event logging offers a remote, push/pull model.

PowerShell Logging - Defender Goals

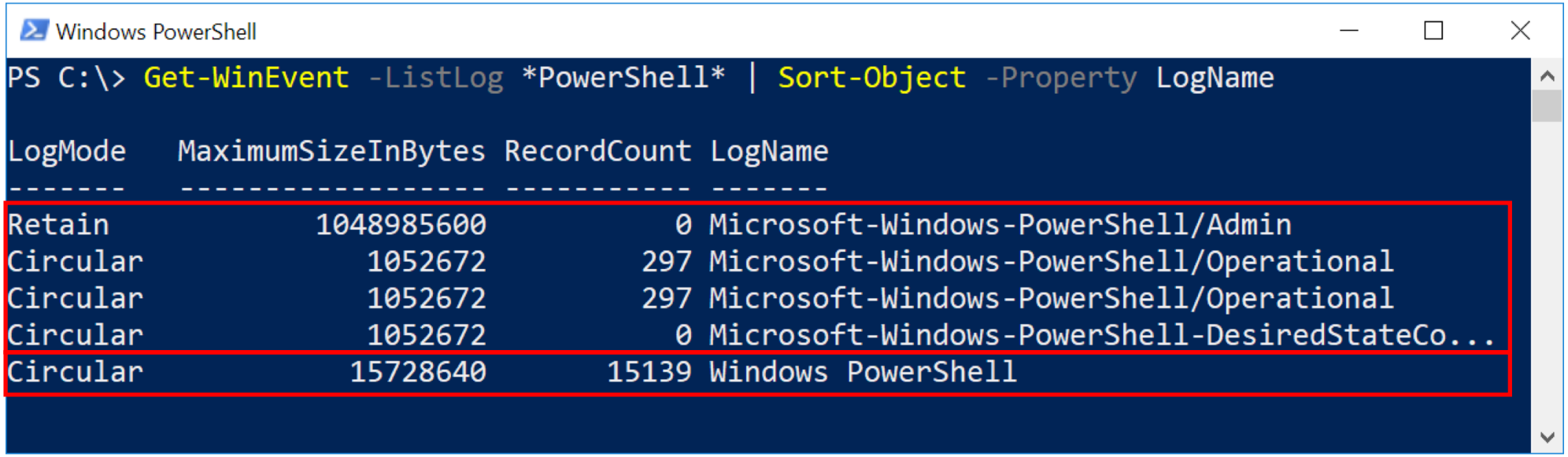
Logging from a defender's perspective:

1. Supply indicators of attack
   a. A single event log could supply enough relevant information to present evidence of an attack.
2. Supply attack context
   a. Events related to an alert supply much needed attack context - crucial for investigation and remediation.

Logs are meaningless though if they aren’t aggregated, forwarded, and looked at though.

PowerShell Logging

All PowerShell activity is logged to two event logs:

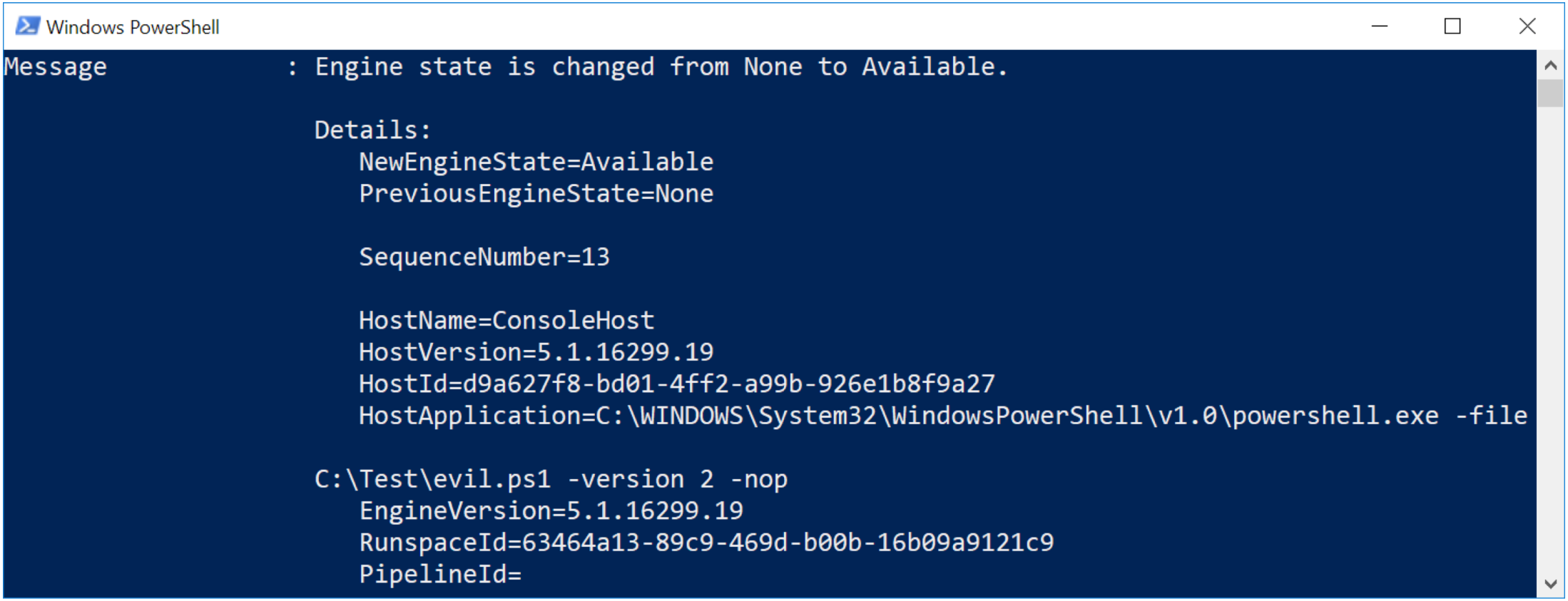


PowerShell Logging - Classic Log

Supports the following event IDs:

- 100-103 - Engine Health
- 200 - Command Health
- 300 - Provider Health
- 400-403 - Engine Lifecycle
- 500-502 - Command Lifecycle
- 600 - Provider Lifecycle
- 700 - Settings
- 800 - Pipeline Execution

PowerShell Logging - Classic Log - Engine Lifecycle Events (Event ID 400)



PowerShell Logging - Classic Log - Engine Lifecycle Events (Event ID 400)

Indicates when a PowerShell host process has started.

Valuable fields:

- HostApplication - PowerShell command line logging for free
- HostVersion - Useful for identifying PowerShell downgrade attacks
- HostName - The name of the PowerShell host. e.g. PSAttack uses a custom host name.

This log is not necessarily generated for all PowerShell hosts: e.g. Windows Troubleshooting Packs

PowerShell Logging - Classic Log - Pipeline Execution Events (Event ID 800)

- PowerShell “module logging” entries
- Available since PowerShell version 3
- Configured through GPO:
  - Administrative Templates/Windows Components/Windows PowerShell - Turn On Module Logging
  - [HKLM|HKCU]\SOFTWARE\Policies\Microsoft\Windows\PowerShell - EnableModuleLogging = 1
- EID events can be populated without module logging configured:
  - Any call to Add-Type
  - Any module loaded at runtime that sets LogPipelineExecutionDetails
    - see `Get-Help about_Eventlogs`

PowerShell Logging - Classic Log - Pipeline Execution Events (Event ID 800)

```
Details:
CommandInvocation(Add-Type): "Add-Type"
ParameterBinding(Add-Type): name="TypeDefinition"; value="        using System;
        using System.Runtime.InteropServices;
        using Microsoft.Win32.SafeHandles;

        namespace Crypto {
            public class NativeMethods {
                [DllImport("msvcrt.dll")]
                public static extern IntPtr memset(IntPtr dest, uint src, uint count);
```

Log Name:	Windows PowerShell		
Source:	PowerShell (PowerShell)	Logged:	10/24/2017 5:55:36 PM
Event ID:	800	Task Category:	Pipeline Execution Details
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	DESKTOP-AS6F42P

PowerShell Logging - Classic Log

- Attackers can write whatever they want to the event log (even remotely)
- This enables an attacker to craft their own seemingly legitimate logs to hamper analysis
- Or allow the event log to be used as a C2 channel

PowerShell Logging - Classic Log

- Writing a fake event log

```
$Arguments = @('Windows PowerShell', '.', 'PowerShell')
$Instance = New-Object -TypeName Diagnostics.EventInstance -ArgumentList 400, 4
$PowerShellEventLog = New-Object -TypeName Diagnostics.EventLog -ArgumentList $Arguments
$PowerShellEventLog.WriteEvent($Instance, @('Available', 'None', 'Fake entry!!!'))
```

PowerShell Logging - Classic Log

- Writing and retrieving arbitrary data

```
Write-EventLog -LogName 'Windows PowerShell' -Source PowerShell -Category 4 -EventId 1337 -RawData @(0,1,2,3) -Message ' '

Get-EventLog -LogName 'Windows PowerShell' -Source PowerShell -InstanceId 1337 | Select-Object -ExpandProperty Data
```

PowerShell Logging - Modern Log

Most importantly, this log captures scriptblock execution events

```
Get-WinEvent -LogName Microsoft-Windows-PowerShell/Operational -
FilterXPath '*[System[EventID=4104 and Level=3]]'
```

PowerShell Logging - Modern Log

Dump event schema: `perfview.exe /nogui userCommand DumpRegisteredManifest Microsoft-Windows-PowerShell`

```
<event value="4097" symbol="ConnectTobeusedwhenoperationisjustexecutingamethod_V1" version="1" task="Connect" op
<event value="4098" symbol="ConnectTobeusedwhenoperationisjustexecutingamethod4098_V1" version="1" task="Connect
<event value="4099" symbol="ConnectTobeusedwhenoperationisjustexecutingamethod4099_V1" version="1" task="Connect
<event value="4100" symbol="NoneTobeusedwhenanexceptionisraised_V1" version="1" task="None" opcode="Tobeusedwhen
<event value="4101" symbol="NoneTobeusedwhenanexceptionisraised4101_V1" version="1" task="None" opcode="Tobeused
<event value="4102" symbol="NoneTobeusedwhenanexceptionisraised4102_V1" version="1" task="None" opcode="Tobeused
<event value="4103" symbol="NoneTobeusedwhenanexceptionisraised4103_V1" version="1" task="None" opcode="Tobeused
<event value="4104" symbol="StartingCommandOncreatecalls_V1" version="1" task="StartingCommand" opcode="Oncreate
<event value="4105" symbol="StartingCommandOpen(async)_V1" version="1" task="StartingCommand" opcode="Open(async
<event value="4106" symbol="StoppingCommandClose(Async)_V1" version="1" task="StoppingCommand" opcode="Close(Asy
<event value="7937" symbol="NoneTobeusedwhenoperationisjustexecutingamethod_V1" version="1" task="None" opcode="
<event value="7938" symbol="NoneTobeusedwhenoperationisjustexecutingamethod7938_V1" version="1" task="None" opco
<event value="7939" symbol="NoneTobeusedwhenoperationisjustexecutingamethod7939_V1" version="1" task="None" opco
<event value="7940" symbol="NoneTobeusedwhenoperationisjustexecutingamethod7940_V1" version="1" task="None" opco
```

PowerShell Logging - Modern Log

ETW Tampering - i.e. cut off the event supply from the source

```
logman query providers | findstr PowerShell
logman query providers Microsoft-Windows-PowerShell
$OriginalProvider = Get-EtwTraceProvider -SessionName EventLog-Application -Guid
'{A0C1853B-5C40-4B15-8766-3CF1C58F985A}'
Remove-EtwTraceProvider -SessionName EventLog-Application -Guid '{A0C1853B-5C40-4B15-
8766-3CF1C58F985A}'
Add-EtwTraceProvider -SessionName EventLog-Application -Guid '{A0C1853B-5C40-4B15-8766-
3CF1C58F985A}' -MatchAnyKeyword ([UInt64] $OriginalProvider.MatchAnyKeyword)
```