Greater Visibility Through PowerShell Logging | Mandiant

By Mandiant

Published: 2016-02-11 · Archived: 2026-04-06 00:06:27 UTC

Written by: Matthew Dunwoody

UPDATE (Feb. 29): This post has been updated with new configuration recommendations due to the Feb. 24 rerelease of PowerShell 5, and now includes a link to a parsing script that users may find valuable.

Introduction

Mandiant is continuously investigating attacks that leverage PowerShell throughout all phases of the attack. A common issue we experience is a lack of available logging that adequately shows what actions the attacker performed using PowerShell. In those investigations, Mandiant routinely offers guidance on increasing PowerShell logging to provide investigators a detection mechanism for malicious activity and a historical record of how PowerShell was used on systems. This blog post details various PowerShell logging options and how they can help you obtain the visibility needed to better respond, investigate, and remediate attacks involving PowerShell.

Background

Attackers and developers of penetration-testing frameworks are increasingly leveraging Windows PowerShell to conduct their operations. PowerShell is an extremely powerful command environment and scripting language that is built in to Microsoft Windows. By default, PowerShell does not leave many artifacts of its execution in most Windows environments. The combination of impressive functionality and stealth has made attacks leveraging PowerShell a nightmare for enterprise security teams[1].

PowerShell 2.0, which comes installed on all Windows 7/2008 systems, provides very little evidence of attacker activity. The Windows event logs show that PowerShell executed, the start and end times of sessions, and whether the session executed locally or remotely (ConsoleHost or ServerRemoteHost). However, they reveal nothing about what was executed with PowerShell. Figure 1 shows an example of the event log messages recorded in the PowerShell 2.0 log Windows PowerShell.evtx.

```
Engine state is changed from None to Available.


Details:
             NewEngineState=Available
             PreviousEngineState=None

             SequenceNumber=9

             HostName=ConsoleHost
             HostVersion=2.0
             HostId=fbe80f36-d50a-4b69-bc1f-c40f16108018
             EngineVersion=2.0
             RunspaceId=7d119dd1-5b08-42d6-833b-880c16d2c222
             PipelineId=
             CommandName=
             CommandType=
             ScriptName=
             CommandPath=
             CommandLine=
```

Figure 1: PowerShell Session Start in PowerShell 2.0

Microsoft has been taking steps to improve the security transparency of PowerShell in recent versions. The most significant improvements, such as enhanced logging, were released in PowerShell version 5.0. This enhanced logging records executed PowerShell commands and scripts, de-obfuscated code, output, and transcripts of attacker activity. Enhanced PowerShell logging is an invaluable resource, both for enterprise monitoring and incident response.

PowerShell 5.0 is the current release for Windows 7/2008 R2 and above. Though many of the enhanced logging features of PowerShell 5.0 were backported to version 4.0, Mandiant recommends installing PowerShell 5.0 on all Windows platforms. PowerShell 5.0 includes features not available in 4.0, including suspicious script block logging.

Installation

Windows 10 does not require any software updates to support enhanced PowerShell logging.

For Windows 7/8.1/2008/2012, upgrading PowerShell to enable enhanced logging in PowerShell 5.0 (recommended) requires:

- .NET 4.5
- Windows Management Framework (WMF) 4.0 (Windows 7/2008 only)
- Windows Management Framework (WMF) 5.0

Windows 7 and 2008 R2 must be upgraded to Windows Management Framework (WMF) 4.0 prior to installing WMF 5.0.

Enabling enhanced logging in PowerShell 4.0 for Windows 7/8.1/2008/2012 requires:

» .NET 4.5
» Windows Management Framework (WMF) 4.0

» The appropriate WMF 4.0 update
- 8.1/2012 R2 – KB3000850
- 2012 – KB3119938
- 7/2008 R2 SP1 – KB3109118

Downloading these updates from Microsoft may require the completion of an automated request process.

Logging Configuration

Logging must be configured through Group Policy as follows:

Administrative Templates → Windows Components → Windows PowerShell

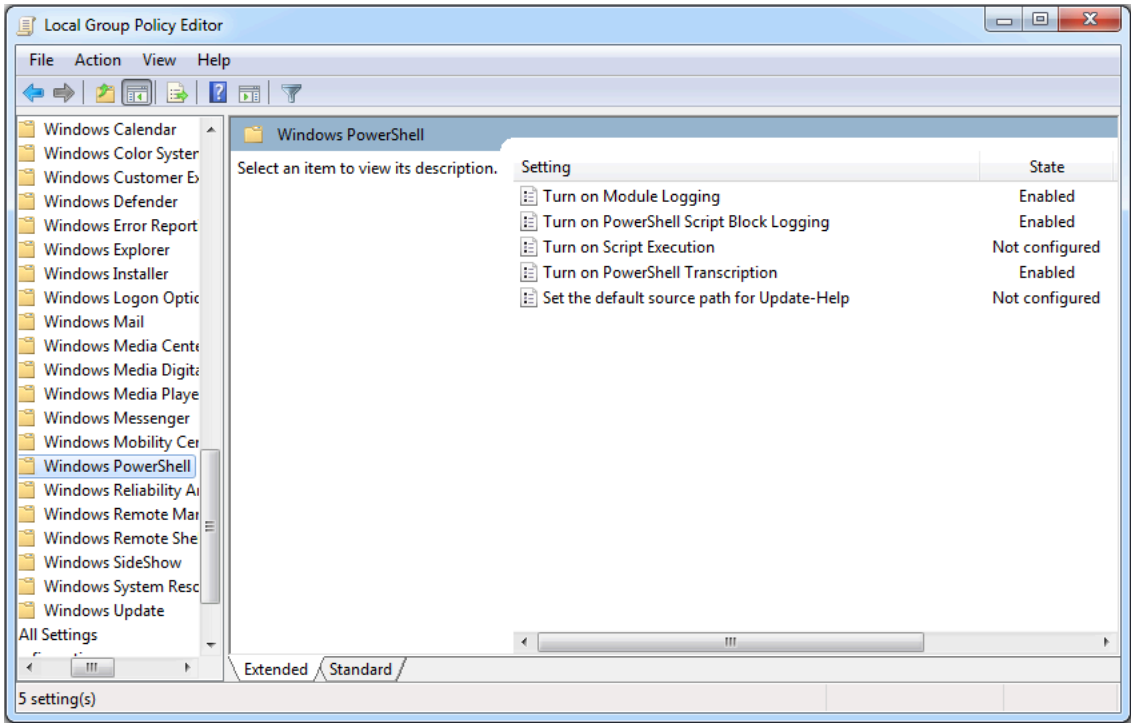

Figure 2: PowerShell configuration options

PowerShell supports three types of logging: module logging, script block logging, and transcription. PowerShell events are written to the PowerShell operational log Microsoft-Windows-PowerShell%4Operational.evtx.

Module Logging

Module logging records pipeline execution details as PowerShell executes, including variable initialization and command invocations. Module logging will record portions of scripts, some de-obfuscated code, and some data formatted for output. This logging will capture some details missed by other PowerShell logging sources, though it may not reliably capture the commands executed. Module logging has been available since PowerShell 3.0. Module logging events are written to Event ID (EID) 4103.

While module logging generates a large volume of events (the execution of the popular Invoke-Mimikatz script generated 2,285 events resulting in 7 MB of logs during testing), these events record valuable output not captured

in other sources.

To enable module logging:

1. In the "Windows PowerShell" GPO settings, set "Turn on Module Logging" to enabled.
2. In the "Options" pane, click the button to show Module Name.
3. In the Module Names window, enter ***** to record all modules.
   a. Optional: To log only specific modules, specify them here. (Note: this is not recommended.)
4. Click "OK" in the "Module Names" Window.
5. Click "OK" in the "Module Logging" Window.

Alternately, setting the following registry values will have the same effect:

» HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging → EnableModuleLogging = 1
» HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging \ModuleNames → * = *

Script Block Logging

Script block logging records blocks of code as they are executed by the PowerShell engine, thereby capturing the full contents of code executed by an attacker, including scripts and commands. Due to the nature of script block logging, it also records de-obfuscated code as it is executed. For example, in addition to recording the original obfuscated code, script block logging records the decoded commands passed with PowerShell's -EncodedCommand argument, as well as those obfuscated with XOR, Base64, ROT13, encryption, etc., in addition to the original obfuscated code. Script block logging will not record output from the executed code. Script block logging events are recorded in EID 4104. Script blocks exceeding the maximum length of an event log message are fragmented into multiple entries. A script is available to parse script block logs and reassemble fragmented blocks (see reference 5).

While not available in PowerShell 4.0, PowerShell 5.0 will automatically log code blocks if the block's contents match on a list of suspicious commands or scripting techniques, even if script block logging is not enabled. These suspicious blocks are logged at the "warning" level in EID 4104, unless script block logging is explicitly disabled. This feature ensures that some forensic data is logged for known-suspicious activity, even if logging is not enabled, but it is not considered to be a security feature by Microsoft. Enabling script block logging will capture all activity, not just blocks considered suspicious by the PowerShell process. This allows investigators to identify the full scope of attacker activity. The blocks that are not considered suspicious will also be logged to EID 4104, but with "verbose" or "information" levels.

Script block logging generates fewer events than module logging (Invoke-Mimikatz generated 116 events totaling 5 MB) and records valuable indicators for alerting in a SIEM or log monitoring platform.

Group Policy also offers an option to "Log script block execution start / stop events". This option records the start and stop of script blocks, by script block ID, in EIDs 4105 and 4106. This option may provide additional forensic information, as in the case of a PowerShell script executing over a long period, but it generates a prohibitively

large number of events (96,458 events totaling 50 MB per execution of Invoke-Mimikatz) and is not recommended for most environments.

To enable script block logging:

1. In the "Windows PowerShell" GPO settings, set "Turn on PowerShell Script Block Logging" to enabled.

Alternately, setting the following registry value will enable logging:

» HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging → EnableScriptBlockLogging = 1

Transcription

Transcription creates a unique record of every PowerShell session, including all input and output, exactly as it appears in the session. Transcripts are written to text files, broken out by user and session. Transcripts also contain timestamps and metadata for each command in order to aid analysis. However, transcription records only what appears in the PowerShell terminal, which will not include the contents of executed scripts or output written to other destinations such as the file system.

PowerShell transcripts are automatically named to prevent collisions, with names beginning with "PowerShell_transcript". By default, transcripts are written to the user's documents folder, but can be configured to any accessible location on the local system or on the network. The best practice is to write transcripts to a remote, write-only network share, where defenders can easily review the data and attackers cannot easily delete them (see reference 2 below). Transcripts are very storage-efficient (less than 6 KB per execution of Invoke-Mimikatz), easily compressed, and can be reviewed using standard tools like grep.

To enable transcription:

1. In the "Windows PowerShell" GPO settings, set "Turn on PowerShell Transcription" to enabled.
2. Check the "Include invocation headers" box, in order to record a timestamp for each command executed.
3. Optionally, set a centralized transcript output directory.

This directory should be a write-only, restricted network share that security personnel can access. If no output directory is specified, the transcript files will be created under the user's documents directory.

Alternately, setting the following registry values will enable logging:

» HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\Transcription → EnableTranscripting = 1
» HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\Transcription → EnableInvocationHeader = 1
» HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\Transcription → OutputDirectory = “” (Enter path. Empty = default)

Log Settings

Where possible, Mandiant recommends enabling all three log sources: module logging, script block logging and transcription. Each of these sources records unique data valuable to analyzing PowerShell activity. In environments where log sizes cannot be significantly increased, enabling script block logging and transcription will record most activity, while minimizing the amount of log data generated. At a minimum, script block logging should be enabled, in order to identify attacker commands and code execution.

Ideally, the size of the PowerShell event log Microsoft-Windows-PowerShell%4Operational.evtx should be increased to 1 GB (or as large as your organization will allow) in order to ensure that data is preserved for a reasonable period. PowerShell logging generates large volumes of data that quickly rolls the log (up to 1 MB per minute has been observed during typical admin or attacker activity).

The Windows Remote Management (WinRM) log, Microsoft-Windows-WinRM%4Operational.evtx, records inbound and outbound WinRM connections, including PowerShell remoting connections. The log captures the source (inbound connections) or destination (outbound connections), along with the username used to authenticate. This connection data can be valuable in tracking lateral movement using PowerShell remoting. Ideally, the WinRM log should be set to a sufficient size to store at least one year of data.

Due to the large number of events generated by PowerShell logging, organizations should carefully consider which events to forward to a log aggregator. In environments with PowerShell 5.0, organizations should consider, at a minimum, aggregating and monitoring suspicious script block logging events, EID 4104 with level "warning", in a SIEM or other log monitoring tool. These events provide the best opportunity to identify evidence of compromise while maintaining a minimal dataset.

Appendices

- Appendix A: Module Logging displays the contents of one module logging event generated during the execution of the Invoke-Mimikatz PowerShell script.
- Appendix B: Script Block Logging displays the contents of one script block logging event generated during the execution of the Invoke-Mimikatz script.
- Appendix C: PowerShell Transcription contains a sample transcript generated when executing Invoke-Mimikatz.

References

1. http://blogs.msdn.com/b/powershell/archive/2016/01/19/windows-management-framework-wmf-4-0-update-now-available-for-windows-server-2012-windows-server-2008-r2-sp1-and-windows-7-sp1.aspx
2. http://blogs.msdn.com/b/powershell/archive/2015/06/09/powershell-the-blue-team.aspx
3. https://www.fireeye.com/content/dam/fireeye-www/global/en/solutions/pdfs/wp-lazanciyan-investigating-powershell-attacks.pdf
4. https://blogs.msdn.microsoft.com/powershell/2016/02/24/windows-management-framework-wmf-5-0-rtm-packages-has-been-republished/
5. https://github.com/matthewdunwoody/block-parser

Special thanks to Lee Holmes and the Microsoft PowerShell team.

[1] See the Shmoocon 2016 presentation “No Easy Breach” by Mandiant consultants Matthew Dunwoody and Nick Carr for some additional context of our experience analyzing PowerShell attacks: https://archive.org/details/No_Easy_Breach.

Posted in

- Threat Intelligence
