PowerShell Logging

There are two types of logging enabled by default with PowerShell as shown below.

1. Engine and Provider Lifecycle Events
2. Suspicious Script Block Logging

In addition to the very limited default logging, you can choose to enable 3 additional log types as follows.

1. Script Block Logging
2. Module Logging
3. Transcription Logging

# Default Logging

First, we will look at the Engine and Provider Lifecycle events that are written to the “**Application and Services Logs 🡪 Windows PowerShell**” log. Open Event Viewer to this location to view these logs.

To open Event Viewer, do a search for “event viewer” in the search bar.

Graphical user interface, application

Description automatically generated

Table

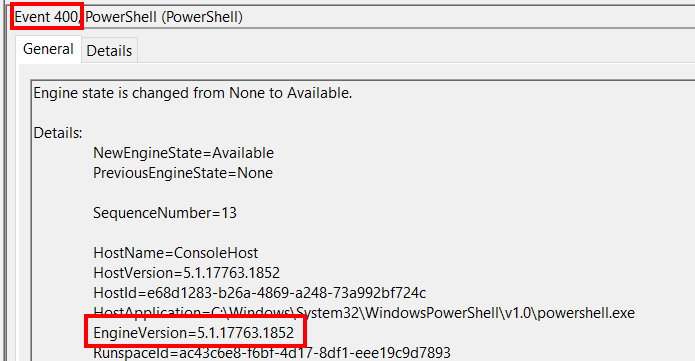
Description automatically generated

This log does not auto-update, so if you see the “**New events available**” message you will need to click the refresh button to load them (or press F5).

Graphical user interface

Description automatically generated with medium confidence

When you start PowerShell, an “Engine Lifecycle” event will be generated (event ID 400). This event will show the PowerShell version being used and can be used to identify a PowerShell downgrade attack. A PowerShell downgrade attack is when an older version of PowerShell is used in order to avoid the security mechanisms in place with later versions, like additional logging options.



The second type of logging that is enabled by default is suspicious script block logging. A script block is considered suspicious if it contains certain keywords like **Add-Type**, **GetField**, or **NonPublic**. Try running the SuspiciousScript.ps1 file provided on your lab VM. You’ll see event ID 800 which logs the details of that script appear in the default PowerShell log.

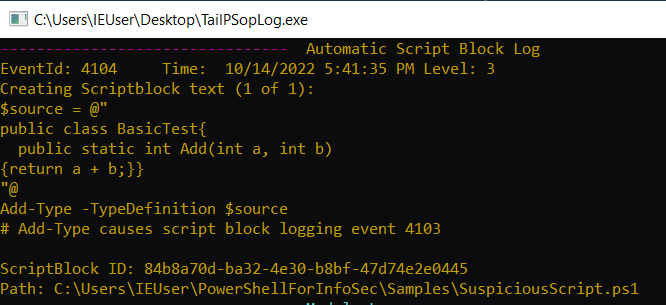
Graphical user interface, text, application, email

Description automatically generated

The SuspiciousScript uses the **Add-Type** command, which is considered suspicious. You can see the entire list of words considered suspicious by executing the following PowerShell command.

[ScriptBlock].GetField('signatures','NonPublic,Static').GetValue($null) | sort

We haven’t enabled any of the optional logging yet, but if we look at the **Microsoft\Windows\PowerShell\Operational** log we will see that these suspicious scripts are being logged there as well. I’ve included a tool on the desktop called **TailPSopLog** on the desktop of the lab VM that shows these events without having to refresh the view. Double click the **TailPSopLog** application and then run the Suspicious script again and watch for the suspicious events to show in the log.



Text

Description automatically generated

Note: You will only see the Dark Yellow script block log the first time it is run in each PowerShell session. You can start a new PowerShell window and then execute the suspicious script again to ensure that you see that log.

Try running one of the other sample scripts provided like, **AddNumbers.ps1**. Do you see anything about its execution in any of the logs?

# Optional Logging

Now let’s enable some of the optional logging and see what effect it has. The logging options can be set by modifying specific registry keys or by using Group Policy. I’ve included a tool for the class to be able to easily turn logging on and off. To run the tool, double click on the **LogMenu** shortcut from File Explorer and accept the UAC prompt by clicking **Yes**.

Graphical user interface, application, table

Description automatically generated

You now have an administrative PowerShell prompt where you can easily set the logging options. If you want to see the details of what the script does, review the Set-PSLogging.ps1 file in the Tools directory.

Graphical user interface, text

Description automatically generated

Choose option 1 to **Enable-AllReasonableLogging**.

Text

Description automatically generated

This will turn on script block, module, and transcription logging. It doesn’t turn on Script Block Invocation Logging because it is very noisy and not helpful. If you want to experiment with it to see, choose the **Enable-AllLogging** option and watch for the red and green colored events.

Note: Logging options are set at the time the PowerShell window opens so you must start a new PowerShell session to see the effects from the logging changes you make.

Now start a new PowerShell session and execute some commands and sample scripts. The color coding in the **TailPSopLog** tool will help clarify which events are module logs (blue) or script block logs (yellow). If you see a dark yellow log, that is a script block log with a level of **warning** that gets generated from *suspicious* commands.

Note: If you ever see the word **select** in the title bar of the **TailPSopLog** or the PowerShell window it means that output is paused. You won’t see any additional output in a window with **select** in the title bar until you click in the window and press the space bar. This will cause the output to continue to flow to the screen.

A screenshot of a computer

Description automatically generated with medium confidence

When you feel comfortable with what and how things are logged, use the **LogMenu** to experiment with different settings. Remember, you need to start a new PowerShell session for any logging settings to be reflected.

Note: Visual Studio code executes PowerShell commands in the background even when you aren’t activity using it. It is best to have VS Code closed when watching the logs to avoid unnecessary confusion.

Transcription logging is a different type of log than script block and module logging. Transcription logging doesn’t show up in a Windows event log and instead is written out to files. The directory for these files is configurable and the **LogMenu** tool sets it to the **C:\Users\IEUser\PSTranscripts** directory.

Graphical user interface, application, Word

Description automatically generated

In the transcripts folder you will see a folder for each day and within each folder you will find a text file for each PowerShell session that is created. A transcript is an “over the shoulder” version of logging where you see everything that the user entering commands sees, including the output to the screen.

Text, letter

Description automatically generated

In the example transcript above, you can see the effect of having double-clicked the **LogMenu** shortcut (the **Show-LoggingMenu.ps1** script was executed). You can also see the menu that was output to the screen.

Look at the transcripts for your sessions to get familiar with the type of information available to you from these logs.

This completes the introduction to PowerShell logging. In class we talked about ways that attackers can bypass logging. You’ll get a chance to do that in a later lab.