Because PowerShell is so powerful it is a major security risk to windows environments. Malware and other malicious threats can wreak havoc on your organizations. Securing PowerShell is by no means straight-forward I have saved my notes and resources for my fellow defenders. Below are notes for best practices. I have kept this as an ongoing document and separated this into 2 parts. Old is anything before year 2017, and New is anything post year 2017.

**OLD:**

Options to Locking Down Powershell Notes: Wndows 2008 R2 domain Powershell v 2.0

1. Use GPO to to force execution policy: Settings under Computer configuration, admin templates, windows components…
2. Use GPO to block script execution.

For machines not on the domain edit registry key: HKEY\_LOCAL\_MACHINE \Software\Policies\Microsoft\Windows\PowerShell

1. Only allow administrators or specific security groups to execute powershell.exe. (Only Allow who you want to allow to run powershell.)

Use GPO under comp config\windows settings\security settings\filesystem to modify rtfs permissions for —path to powershell.exe

%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe

%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell\_ise.exe

1. Add Default Deny Software Restriction policies with enforcement setting All software files except libraries scripts with All users except local admins setting
2. Set Software Restriction Policy default security level to disallowed and apply to all users except local admins
3. Add disallowed rule for \*.ps1 + powershell.exe + powershell\_ise.exe (redundant to number 3?
4. Adding powershell extensions to designated file types list to stop scripts running in IE

**NEW (some items may be redundant to Old section but it’s relevant):**

1. Use group policy set lockdown policy for PSLockDownPolicy And PowerShell: Settings below:

Constrained Language Mode.

Enable Constrained Language Mode:

[Environment]::SetEnvironmentVariable(‘\_\_PSLockdownPolicy‘, ‘4’, ‘Machine‘)

Enable via Group Policy:

Computer Configuration\Preferences\Windows Settings\Environment

1. Be updated with PowerShell 5.0.

PowerShell 5.0 comes with embedded security for the enterprise environment. PowerShell 5.0 With Applocker And Device Guard features :

* Script block logging.
* System-wide transcripts. System-wide transcription can be enabled via Group Policy
* Constrained PowerShell (same as number 1)
* Antimalware integration (Windows 10). The new Windows 10 Antimalware Scan Interface (AMSI) enables all the scripting engines (PowerShell, VBScript, and JScript) to request analysis of dynamic content: from a script file, typed commands at the command line, and even code downloaded and executed in memory. This enables scanning of PowerShell code before it is executed on the computer.

1. Logging PowerShell Activity

For those who have a Centralized Log Management/Aggregation (Splunk and ELK wtfpwns everything else I’ve used!) PowerShell logging can be enabled via Group Policy for PowerShell modules:

* Microsoft.PowerShell.\* (i.e., Microsoft.PowerShell.Management module)
* ActiveDirectory – Logs Active Directory cmdlet use.
* BITS Transfer – Logs use of Background Intelligent Transfer Service (BITS) cmdlets.
* CimCmdlets (2012R2/8.1) – Logs cmdlets that interface with Common Information Model (CIM).
* GroupPolicy – Logs Group Policy cmdlet use.
* Microsoft.WSMan.Management – Logs cmdlets that manage Web Services for Management (WS-Management) and Windows Remote Management (WinRM).
* NetAdapter/NetConnection – Logs Network-related cdmdlets.
* PSScheduledJob/ScheduledTasks (PSv5) – Logs cmdlets to manage scheduled jobs.
* ServerManager – Logs Server Manager cmdlet use.
* SmbShare – Logs Server Message Block (SMB) sharing activity.

With your logging system, alerts should be configured for known attack methods:

* Downloads via .Net (New-Object Net.WebClient).DownloadString)
* Invoke-Expression (and derivatives: “iex”)
* BITS activity
* Scheduled Task creation/deletion
* PowerShell Remoting

The best method to detect PowerShell attack code is to look for key indicators – code snippets required for the code to run correctly.

Example: Detecting Mimikatz (a widely-used tool for logged user credential capture)

Invoke-Mimikatz Event Log Keywords:

“System.Reflection.AssemblyName”

“System.Reflection.Emit.AssemblyBuilderAccess “

“System.Runtime.InteropServices.MarshalAsAttribute”

“TOKEN\_PRIVILEGES”

“SE\_PRIVILEGE\_ENABLED”

For obfuscated PowerShell, custom rules should be developed. For example:

* Look for lots of brackets { }.
* Look for lots of quote marks ‘ “

Both of these are heavily used in obfuscation techniques and usually are not used by legitimate software or normal administrators.

1. Remove PowerShell V.2

Microsoft has announced the that PowerShell V.2 will be deprecated. Since this is obsolete and not needed it should be removed. PowerShell V.2 has virtually no native logging or security capabilities and malicious code will remain stealth and undetected.

1. Just Enough Administration or JEA:

This is included with the latest update of Windows Management Framework 5.0 and 5.1, and is a security technology that helps organizations enforce information security by restricting IT administrative rights. JEA provides a practical, role-based approach to set up and automate restrictions for IT personnel, and reduces the risks associated with providing users with full administrative rights following the principle of least privilege.

JEA is implemented as a Windows PowerShell session endpoint (it requires PS remoting to be enabled), which includes a PowerShell Session Configuration file and one or more Role Capability files.

* PowerShell Session Configuration file. This file is used to specify who can connect to an endpoint. Users and security groups can be mapped to specific management roles. Those files are specific to each machine, so an access control per machine is available. They contain information of what will be the name of the JEA endpoint, which roles will be assigned and of course who will have access to this endpoint. These files are PowerShell data files ending in a .pssc extension
* Role Capability files. These files are used to specify what actions users in a particular role can perform. For example it can be restricted to use certain pre-selected cmdlets, functions and external programs making the use of custom potentially malicious cdmlets practically impossible. Examples of potentially dangerous commands that should be constrained, are ‘Start-Process’, ‘New-Service’, ‘Invokde-Item’ etc.
* Microsoft’s JEA Github provides great samples and resources. <https://github.com/PowerShell/JEA>

Another significant benefit of JEA is the actionable logging and reporting which available in the Windows event log format, since all operations performed through the JEA endpoint can be recorded (with transcripts and logs) and show who accessed the environment and when, and what changes were made.

1. Scripts Code Signing

If PowerShell scripts are used in an enterprise environment, code signing is another control that improves security posture, by ensuring authenticity and integrity. This feature, along with a defined Execution Policy or Group Policy as “AllSigned” or “RemoteSigned”, will permit only digitally signed scripts to run.