8

Implementing Enterprise Security

In this chapter, we cover the following recipes:

* Implementing Just Enough Administration (JEA)
* Examining the Application and Services logs
* Discovering Logon Events in Event Log
* Deploying PowerShell GP Policies
* Logging PowerShell Activity
* Configuring AD Default Password Policy
* Managing Windows Defender

# Introduction

Security within every organization is vital, and with the near-constant threats, more so now than ever. With today’s threats and challenges, you need to implement security into every aspect of your organization from physical security to the security of your network and computer infrastructure.

Since the earliest times, security-savvy folks have preached the gospel of Security in depth. Having as many layers and possible and realistic is just a good thing. As the theory goes – the bad guys have to defeat all your layers to defeat you, while you only need to hold one to stay safe.

PowerShell is a powerful tool for IT professionals wanting to be secure and stay secure. There is so much you can do with PowerShell to help your organization deploy excellent security over your network and computer infrastructure. In this chapter, we look at several ways to use PowerShell to improve your Windows infrastructure's security.

Just Enough Administration is a feature that enabled you to implement fine-grained administration, giving users just enough to enable power them to do their job and not more. A core objective of JEA is to reduce the number of users who are members of very high privilege groups, including the local Administrators, Domain Admins, and Enterprise Admins groups.

# Implementing Just Enough Administration (JEA)

Just Enough Administration, also known as JEA, is a security framework providing you with the ability to implement fine-grained administrative delegation. With JEA, you enable a user to have just enough administrative power to do their job, and no more. JEA is a more secure alternative to just adding users to the Domain Administrator or Enterprise Administrator groups.

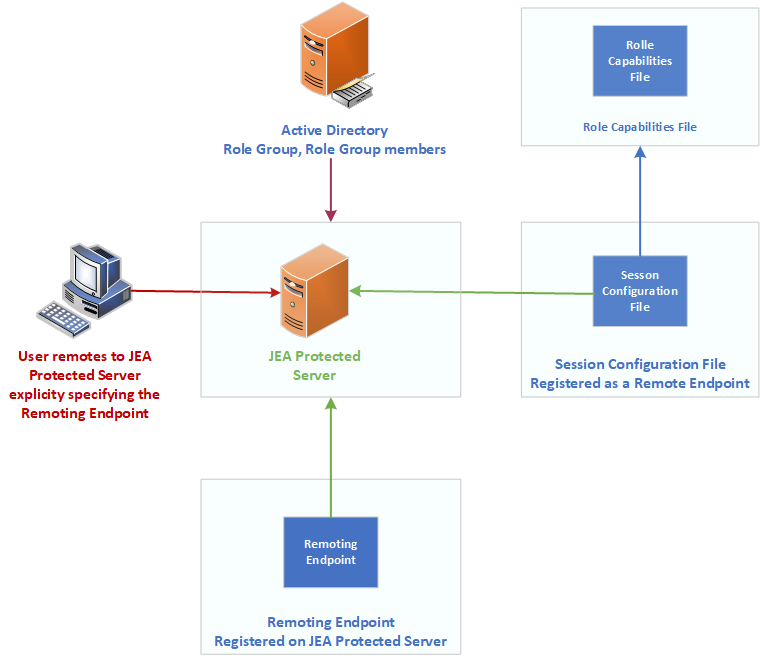
With JEA, you could, for example, enable a junior administrator the rights to access your domain controllers to administer the DNS Service on the DC. JEA allows you to constrain what the user can do on the protected server. For example, you could allow the user to stop and start the DNS Service (using Stop-Service and Start-Service) but no other services.

JEA makes use of three objects:

* JEA role capabilities file (.psrc) This file defines a role in terms of its capabilities. You would configure the JEA role RKDnsAdmins to define a limited set of cmdlets that the role has access to on the Domain Controller, namely those related to administering DNS on a DC.
* JEA session configuration file (.pssc). This file defines who can access a PowerShell remoting session and what they can do within the session. You could allow anyone in the RKDnsAdmins domain security group to access the server using a JEA endpoint. The session configuration file defines the actions allowed within the JEA session by reference to the role capabilities file. A JEA protected remoting session can only be used by certain people who can do whatever the role capabilities file dictates.
* A PowerShell remoting endpoint. Once you have the role capabilities and session configuration files created, you register the JEA endpoint to the server you are protecting with JEA

Once the JEA endpoint is registered, a user who is a member of the domain security group,   
RKDnsAdmins can use Invoke-Command or Enter-PsSession, specifying the remote server and the JEA-protected endpoint to access the protected server. Once inside the remoting session, the user can only do what the role capabilities file allows.

The following diagram shows the components of JEA:



1. Insert image B16762\_08\_01.png

## Getting Ready

This recipe uses DC1, a domain controller in the Reskit.Org domain on which you set up JEA for inbound connections. You installed DC1 as a domain controller and configured users, groups, and OUs, both in Chapter 5. You run the first part of this recipe on DC1.

You would typically use a client computer to access the DC to manage DNS in production. For this recipe, adding an extra client host is replaced by using DC1 to test JEA without requiring an additional host. Of course, in production, you should test JEA on a client host.

## How to do it...

1. Creating a transcripts folder

New-Item -Path C:\JEATranscripts -ItemType Directory |

  Out-Null

1. Creating a role capabilities folder

$JEACF = "C:\JEACapabilities"

New-Item -Path $JEACF -ItemType Directory |

  Out-Null

1. Creating a JEA session configuration folder

$SCF = 'C:\JEASessionConfiguration'

New-Item -Path $SCF -ItemType Directory |

  Out-Null

1. Creating DNSAdminsJEA as a global security group

$DNSGHT = @{

  Name          = 'DNSAdminsJEA'

  Description   = 'DNS Admins for JEA'

  GroupCategory = 'Security'

  GroupScope    = 'Global'

}

New-ADGroup @DNSGHT

Get-ADGroup -Identity 'DNSAdminsJEA' |

  Move-ADObject -TargetPath 'OU=IT, DC=Reskit, DC=Org'

1. Adding JerryG to the DNS Admins group

$ADGHT = @{

  Identity  = 'DNSAdminsJEA'

  Members   = 'JerryG'

}

Add-ADGroupMember @ADGHT

1. Creating a role capabilities file

$RCF = Join-Path -Path $JEACF -ChildPath "DnsAdmins.psrc"

$RCHT = @{

  Path            = $RCF

  Author          = 'Reskit Administration'

  CompanyName     = 'Reskit.Org'

  Description     = 'DnsAdminsJEA role capabilities'

  AliasDefinition = @{Name='gh';Value='Get-Help'}

  ModulesToImport = 'Microsoft.PowerShell.Core','DnsServer'

  VisibleCmdlets  = (@{ Name       = "Restart-Computer";

                        Parameters = @{Name = "ComputerName"}

                        ValidateSet = 'DC1, DC2'},

                       'DNSSERVER\\*',

                     @{ Name       = "Stop-Service";

                        Parameters = @{Name = "DNS"}},

                     @{ Name       = "Start-Service";

                        Parameters = @{Name = "DNS"}}

                     )

  VisibleExternalCommands = ('C:\Windows\System32\whoami.exe',

                             'C:\Windows\System32\ipconfig.exe')

  VisibleFunctions = 'Get-HW'

  FunctionDefinitions = @{

    Name = 'Get-HW'

    Scriptblock = {'Hello JEA World'}}

}

New-PSRoleCapabilityFile @RCHT

1. Creating a JEA session configuration file

$P   = Join-Path -Path $SCF -ChildPath 'DnsAdmins.pssc'

$RDHT = @{

  'DnsAdminsJEA' =

      @{'RoleCapabilityFiles' =

        'C:\JEACapabilities\DnsAdmins.psrc'}

}

$PSCHT= @{

  Author              = 'DoctorDNS@Gmail.Com'

  Description         = 'Session Definition for DnsAdminsJEA'

  SessionType         = 'RestrictedRemoteServer'   # ie JEA!

  Path                = $P       # Role Capabilties file

  RunAsVirtualAccount = $true

  TranscriptDirectory = 'C:\JeaTranscripts'

  RoleDefinitions     = $RDHT     # tk role mapping

}

New-PSSessionConfigurationFile @PSCHT

1. Testing the session configuration file

Test-PSSessionConfigurationFile -Path $P

1. Enabling remoting on DC1

Enable-PSRemoting -Force |

  Out-Null

1. Registering the JEA session configuration remoting endpoint

$SCHT = @{

  Path  = $P

  Name  = 'DnsAdminsJEA'

  Force =  $true

}

Register-PSSessionConfiguration @SCHT

1. Viewing remoting endpoints

Get-PSSessionConfiguration  |

  Format-Table -Property NAME, PSVERSION, Run\*Account

1. Verifying what the user can do

$SCHT = @{

  ConfigurationName = 'DnsAdminsJEA'

  Username          = 'Reskit\JerryG'

}

Get-PSSessionCapability  @SCHT |

  Sort-Object -Property Module

1. Creating credentials for user JerryG

$U    = 'JerryG@Reskit.Org'

$P    = ConvertTo-SecureString 'Pa$$w0rd' -AsPlainText -Force

$Cred = [PSCredential]::New($U,$P)

1. Defining three script blocks and an invocation splatting hash table

$SB1   = {Get-Command}

$SB2   = {Get-HW}

$SB3   = {Get-Command -Name  '\*-DNSSERVER\*'}

$ICMHT = @{

  ComputerName      = 'DC1.Reskit.Org'

  Credential        = $Cred

  ConfigurationName = 'DnsAdminsJEA'

}

1. Getting commands available within the JEA session

Invoke-Command -ScriptBlock $SB1 @ICMHT |

  Sort-Object -Property Module |

    Select-Object -First 15

1. Invoking a JEA-defined function in a JEA session As JerryG

Invoke-Command -ScriptBlock $SB2 @ICMHT

1. Get DNSServer commands available to JerryG

$C = Invoke-Command -ScriptBlock $SB3 @ICMHT

"$($C.Count) DNS commands available"

1. Examining the contents of the transcripts folder:

Get-ChildItem -Path $PSCHT.TranscriptDirectory

1. Examining a transcript

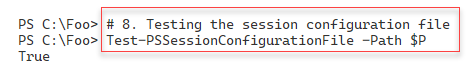
Get-ChildItem -Path $PSCHT.TranscriptDirectory |

  Select-Object -First 1  |

    Get-Content

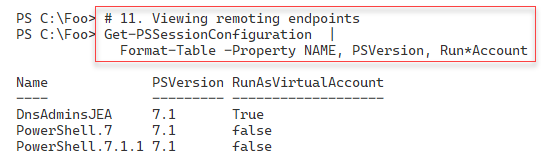
## How it works...

1. In step 1, you create a new folder which you use to hold JEA transcripts. In step 2, you create a folder to hold role capabilities files. In step 3, you create a folder to hold JEA session configuration files. These three steps produce no output.
2. In step 4, you create a global security group you use with JEA, and in step 5, you add the user JerryG to that group. Neither step produces output.
3. In step 6, you create a role capabilities file and store it in the role capabilities folder you created in step 2. In step 7, you create a JEA session configuration file, which you store in the folder you created in step 3. These two steps produce no output.
4. In step 8, you use the Test-PSSessionConfigurationFile command to test the session configuration file. The command produces the following output:



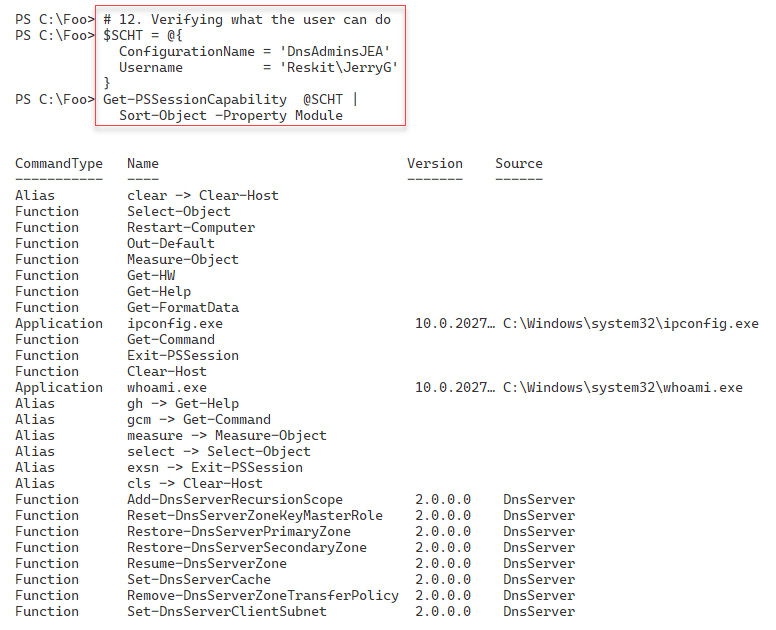
1. Insert image B16762\_08\_02.png
2. In step 9, you use the Enable-PSRemoting command to ensure that you have configured DC1 for WinRM remoting. This step creates no output.
3. In step 10, you complete the setup by registering a JEA session configuration remoting endpoint, producing no output.

In step 11, you use the Get-PSSessionConfiguration command to view the remoting endpoints available on DC1, with output like this:



1. Insert image B16762\_08\_03.png

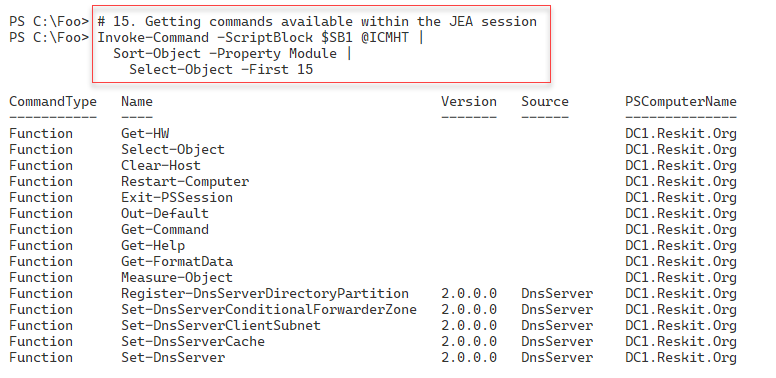
In step 12, you verify the commands that the JEA session configuration allows the user, JerryG, to access within the DNSAdminsJEA remoting endpoint, which produces the following output:



1. Insert image B16762\_08\_04.png

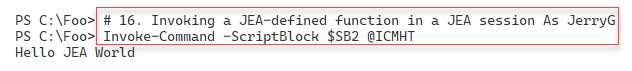
In step 13, you create a PowerShell credential object for JerryG. In step 14, you create a Windows credential object for JerryG@Reskit.Org. These two steps create no output.

In step 14, you crate three script blocks and an invocation hash table for use in later steps, producing no output. In step 15, you invoke the $SB1 script block you created in a inside a JEA session, with output (truncated) like this:



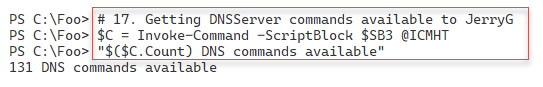
1. Insert image B16762\_08\_05.png

In step 16, you invoke the $SB2 script block to call the HW function defined in the



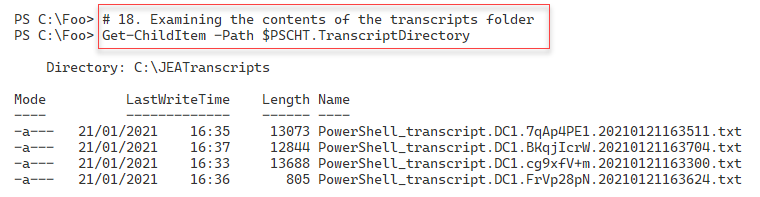
1. Insert image B16762\_08\_06.png

In step 17, you invoke the $SB2 script block which counts the number of commands available in the DNS server module which the user JerryG has permissions to use. The output is like this:



1. Insert image B16762\_08\_07.png

When you set up JEA, you indicated JEA should create a transcript for each JEA session. In step 18, you examing the transcripts in the transcript folder, with output like this.



1. Insert image B16762\_08\_08.png

In the final step, step 19, in this recipe, you examine the first transcript in the transcripts folder, with output (truncated for publishing) that should look like this:



1. Insert image B16762\_08\_09.png

## There's more...

In step 9, you use the Enable-PSRemoting command. This command ensures you enable WinRM remoting, and creates two standard PowerShell 7 remoting endpoints, in addition to the one you create in step 10.

In step 15, you run $SB1 inside a JEA session on DC1. This script invokes Get-Command to list all the commands available to any member of DNSAdminsJEA group. The output is truncated in the figure to take up a bit less space for publishing. The full output lists all the commands available.

In step 18, you examine the transcripts in the JEA transcripts folder. Depending on what you have done so far, you may see a different number of transcripts. Each transcript represents one use of a JEA session and contains full details of the commands that the user ran inside the session, which the user initiated the session, and when.

In the final step, step 19, you examine one of the JEA session transcripts. In the figure, you see the transcript generated by step 15. Depending on

# Examining Applications and Service Logs

Since the first version of Windows NT in 1993, anytime anything happens on a Windows, the component responsible writes details to an event log. In the earlier versions of Windows Server, there were four different Windows logs:

* Application - holds events related to software you have installed on the Server
* Security - holds events related to the security of your server
* Setup - holds events related to KB installation and events that occurred during installation
* System - holds events that relate to this system such as system start and system shut down.

In addition to these logs, other applications and features can add additional classic logs. You can see these classic logs by using the Windows Powershell Get-Eventlog cmdlet.

With Window Vista, Microsoft made some significant improvements to the event logging features. A substantial improvement is adding the Application and Services Logs. These contain over four hundred individual logs. These extra logs allow Windows components to write to application-specific logs rather than the System or Application classic event logs, making it easier to find the events on a given host. There are hundreds of these application and services logs that provide application-specific or service-specific event entries, but Windows does not enable all the logs by default. With PowerShell 7, you use the Get-WinEvent to work with all of the event logs, including these newer ones.

In this recipe, you examine the logs and how to get log event details.

## Getting Ready

You run this recipe on SRV1, a domain-joined Windows Server. You also need DC1, a domain controller in the Reskit.Org domain. You have installed PowerShell7 and VS code on each system.

## How to do it...

1. Registering PowerShell event log provider

& $PSHOME\RegisterManifest.ps1

1. Discovering classic event logs on SRV1

Get-EventLog -LogName \*

1. Discovering and measuring all event logs on this host

$Logs = Get-WinEvent -ListLog \*

"There are $($Logs.count) total event logs on SRV1"

1. Discovering and measuring all event logs on DC1

$SB1     = {Get-WinEvent -ListLog \*}

$LogsDC1 = Invoke-Command -ComputerName DC1 -ScriptBlock $SB1

"There are $($LogsDC1.count) total event logs on DC1"

1. Discovering log member details

$Logs | Get-Member

1. Measuring enabled logs on SRV1

$Logs |

  Where-Object IsEnabled |

    Measure-Object |

      Select-Object -Property Count

1. Measuring enabled logs on DC1

$LogsDC1 |

  Where-Object IsEnabled |

    Measure-Object |

      Select-Object -Property Count

1. Measuring Enabled logs that have records on SRV1

$Logs |

  Where-Object IsEnabled |

    Where-Object Recordcount -gt 0 |

      Measure-Object |

        Select-Object -Property Count

1. Discovering PowerShell related logs

$Logs |

  Where-Object LogName -match 'powershell'

1. Examining PowerShellCore event log

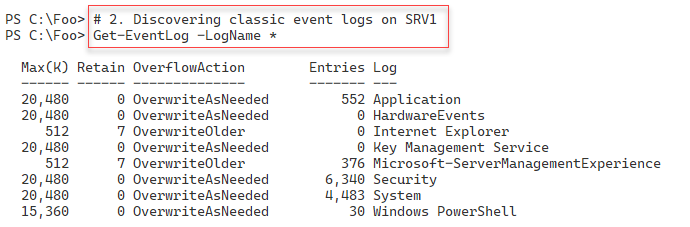
Get-Winevent -LogName 'PowerShellCore/Operational' |

  Select-Object -First 10

## How it works...

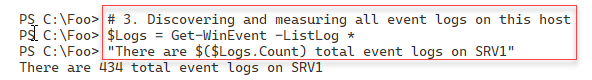
In step 1, you ensure that the Windows has had the PowerShell event log provider registered. This step creates not output.

In step 2, you use Get-EventLog to discover the classic event logs on SRV1, with output like this:



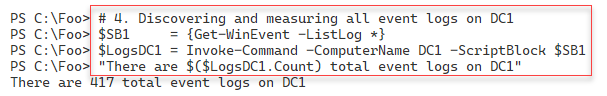
1. Insert image B16762\_08\_10.png

In step 3, you use the Get-WinEvent cmdlet to discover all of the event logs on SRV1. The output looks like this:



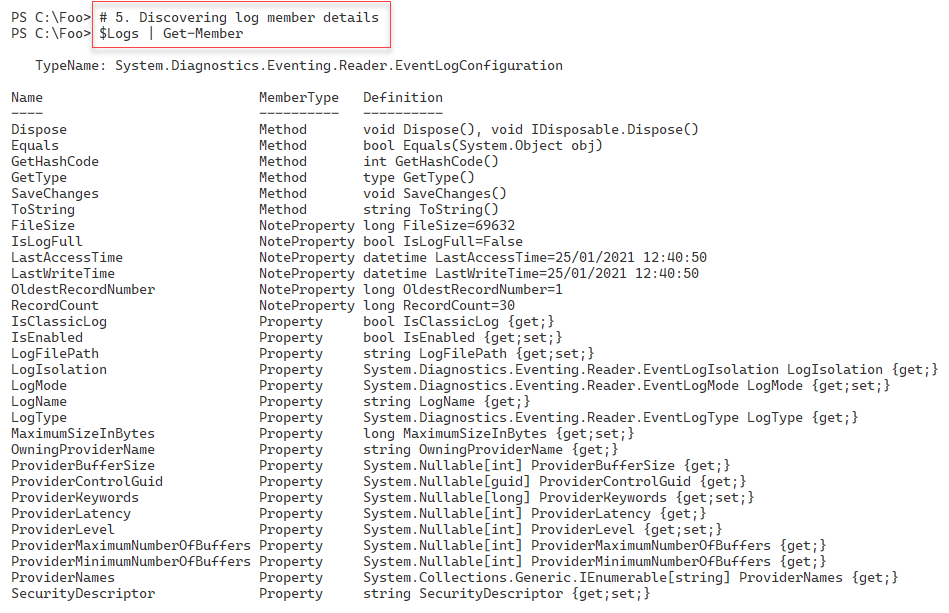
1. Insert image B16762\_08\_11.png

In step 4, you discover and measure the number of event logs on the domain controller DC1, with output that looks like this:



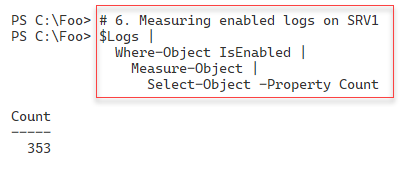
1. Insert image B16762\_08\_12.png

In step 5, you use Get-Member to discover the properties of event logs that you can make use of when querying. The output looks like this:



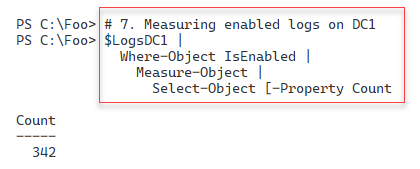
1. Insert image B16762\_08\_13.png

Windows does not enable all event logs by default. In step 6, you discover the enabled logs on SRV1. The output looks like this:



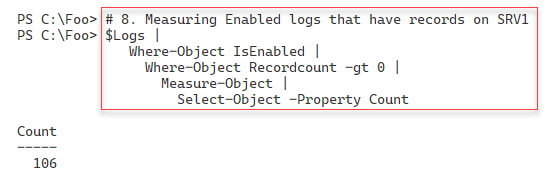
1. Insert image B16762\_08\_14.png

In step 7, you measure the enabeled event logs on DC1, with output like this:



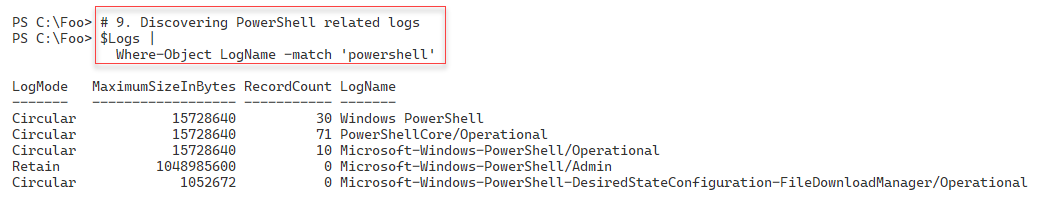
1. Insert image B16762\_08\_15.png

With step 8, you counbt the number of event logs that you have enabled on SRV1 and that have event log entries. The output looks like this:



1. Insert image B16762\_08\_16.png

In step 9, you discover which event logs relate to wither Windows PowerShell or PowerShell 7 (aka PowerShell Core). The output is as follows:



1. Insert image B16762\_08\_17.png

## There's more...

In step 3 and step 4, you get a count of the number of event logs on SRV1 and DC1. As you can see, the number of logs differs. Different Windows features and applications can add additional event logs for your use. In step 6 and step 7, you also see the number of enabled logs on both systems. And with step 8, you see how many enabled logs (on SRV1) actually contain event log entries.

In step 9, you see the event logs on SRV1 that relate to both Windows PowerShell and PowerShell 7. You examine the PowerShellcCore logs in more detaill in “Logging PowerShell Activity”.

# Discovering Logon Events in Event Log

Whenver you attempt to logon, whether you are successfull or not, Windows logs the attempt. These log events can help you determine who logged into a computer and when. This can be an important bit of information if you are tracking down an issue.

In Windows, there are several different logon types. You can read more detail in this article: https://docs.microsoft.com/previous-versions/windows/it-pro/windows-server-2003/  
cc787567(v=ws.10). Note that this document is somewhat outdated and has not been updated

A logon type of 2 indicates a local console logon (that is, logging on to a physical host), while a logon type of 10 indicates logon over RDP. Other logon types include service logon (type 5), Batch or scheduled task (type 4), and console unlock (type 7).

In this recipe, you use PowerShell to examine the Security event log and look at the logon events.

## Getting Ready

Specific stuff you need to do this recipe

## How to do it...

1. Getting security log events

$SecLog = Get-WinEvent -ListLog Security

"Security Event log entries:    [{0,10:N0}]" -f $Seclog.RecordCount

1. Getting all Windows Security log event details

$SecEvents = Get-WinEvent -LogName Security

"Found $($SecEvents.count) security events on DC1"

1. Examining Security event log event members

$SecEvents |

  Get-Member

1. Summarizing security events by event Id

$SecEvents |

  Sort-Object -Property Id |

    Group-Object -Property ID |

      Sort-Object -Property Name |

        Format-Table -Property Name, Count

1. Getting all successful logon events on DC1

$Logons = $SecEvents | Where-Object ID -eq 4624   # logon event

"Found $($Logons.Count) logon events on DC1"

1. Getting all failed logon events on DC1

$FLogons = $SecEvents | Where-Object ID -eq 4625   # failed logon event

"Found $($FLogons.Count) failed logon events on DC1"

1. Creating a summary array of successful logon events

$LogonEvents = @()

Foreach ($Logon in $Logons) {

  $XMLMSG = [xml] $Logon.ToXml()

  $Text = '#text'

  $HostName   = $XMLMSG.Event.EventData.data.$Text[1]

  $HostDomain = $XMLMSG.Event.EventData.data.$Text[2]

  $Account    = $XMLMSG.Event.EventData.data.$Text[5]

  $AcctDomain = $XMLMSG.Event.EventData.data.$Text[6]

  $LogonType  = $XMLMSG.Event.EventData.data.$Text[8]

  $LogonEvent = New-Object -Type PSCustomObject -Property @{

     Account   = "$AcctDomain\$Account"

     Host      = "$HostDomain\$Hostname"

     LogonType = $LogonType

     Time      = $Logon.TimeCreated

  }

  $LogonEvents += $logonEvent

}

1. Summarizing successful logon events on DC1

$LogonEvents |

  Group-Object -Property LogonType |

    Sort-Object -Property Name |

      Format-Table Name, Count

1. Creating a summary array of failed logon events on DC1

$FLogonEvents = @()

Foreach ($FLogon in $FLogons) {

  $XMLMSG = [xml] $FLogon.ToXml()

  $Text = '#text'

  $HostName   = $XMLMSG.Event.EventData.data.$Text[1]

  $HostDomain = $XMLMSG.Event.EventData.data.$Text[2]

  $Account    = $XMLMSG.Event.EventData.data.$Text[5]

  $AcctDomain = $XMLMSG.Event.EventData.data.$Text[6]

  $LogonType  = $XMLMSG.Event.EventData.data.$Text[8]

  $LogonEvent = New-Object -Type PSCustomObject -Property @{

     Account   = "$AcctDomain\$Account"

     Host      = "$HostDomain\$Hostname"

     LogonType = $LogonType

     Time      = $FLogon.TimeCreated

  }

  $FLogonEvents += $LogonEvent

}

1. Summarizing failed logon events on DC1

$FLogonEvents |

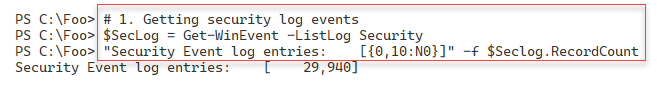
  Group-Object -Property Account |

    Sort-Object -Property Name |

      Format-Table Name, Count

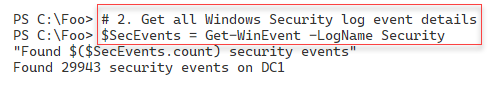
## How it works...

In step 1, you use the Get-WinEvent cmdlet to retrieve details about the security log on DC1. Then you display the number of events in the log. The output looks like this:



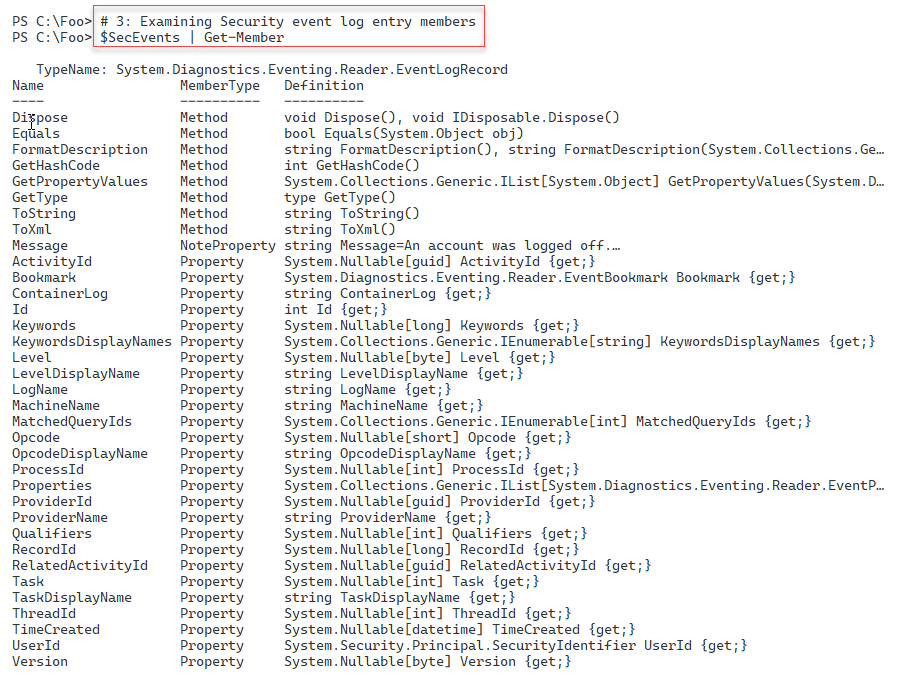
1. Insert image B16762\_08\_18.png

In step 2, you use Get-WinEvent to retrieve all events from the Security log and display a count of the events returned, with output like this:



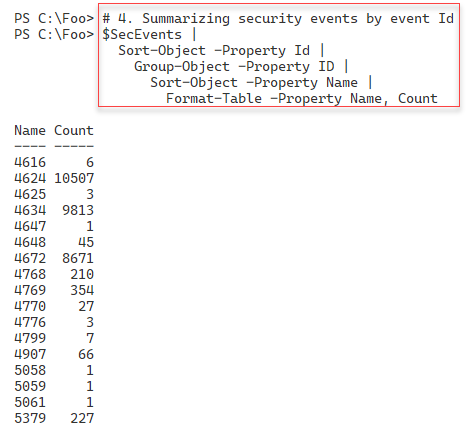
1. Insert image B16762\_08\_19.png

The Get-Winevent cmdlet returns objects of that contain indifidual event log entires. Eaach of of the type System.Diagnostics.Eventing.Reader.EventLogRecord. In step 3, you view the members of this .NET object class, with output like this:



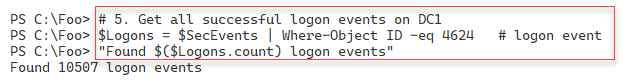
1. Insert image B16762\_08\_20.png

Once you have retrieved the events in the security log, you can examine the different security event types, held in the ID field of each log record. In step 4, you view, and count, the different event ids in the security log, which looks like this.



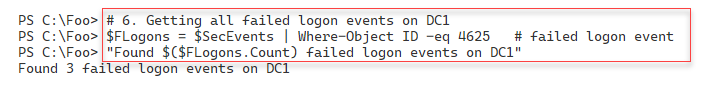
1. Insert image B16762\_08\_21.png

There are two related logon events you can track. Log entries with an event ID of 4624 represent successful logon events, while 4625 represents failed logons. In step 5, you get ALL the successful logon events, with output like this:



1. Insert image B16762\_08\_22.png

In step 6, you count the number of logon failures on DC1, which looks like this:



Screen shots for each step that generates one

## There's more...

In step 1, you retrieve a summary of the events in the security log and display the number of events in the log. In step 2, you retrieve and count the number of entries. As you can see in the figures above, the counts do not match. This is because, between the two steps, Windows logged additional events to the security log. The extra events are most likely events generated by background tasks or services. This minor discrepancy is not unexpected.

In step 3, you view the members of log event objects. You can discover more about the members of the class at https://docs.microsoft.com/dotnet/api/system.diagnostics.eventing.reader.eventlogrecord,

In step 6, you obtain unsuccessful logon events. In order to obtain unsuccessful logons, you need to ensure you have attempted to logon to DC1 but with invalid credentials. As you see in the output of step 10, there were two userids involved with the three unsuccessful logon attempts on DC1.

# sDeploying PowerShell Group Policies

<http://www.powershellcookbook.com/recipe/sOup/manage-powershell-security-in-an-enterprise>

This recipe, blah blah

## Getting Ready

Specific stuff you need to do this recipe

## How to do it...

1. Step by step

## How it works...

1. Screen shots for each step that generates one

## There's more...

1. Some things of interest in this recipe
2. Repeat the recipe structure

# Logging PowerShell Activity

This recipe, blah blah

https://www.digitalshadows.com/blog-and-research/powershell-security-best-practices/

## Getting Ready

Specific stuff you need to do this recipe

## How to do it...

1. Step by step

## How it works...

1. Screen shots for each step that generates one

## There's more...

1. Some things of interest in this recipe
2. Repeat the recipe structure

# Configuring AD Default Password Policy

This recipe, blah blah

<https://docs.microsoft.com/en-us/powershell/module/addsadministration/set-addefaultdomainpasswordpolicy?view=win10-ps>

## Getting Ready

Specific stuff you need to do this recipe

## How to do it...

1. Step by step

## How it works...

1. Screen shots for each step that generates one

## There's more...

1. Some things of interest in this recipe
2. Repeat the recipe structure

# Managing Windows Defender

This recipe, blah blah

https://docs.microsoft.com/en-us/powershell/module/defender/?view=win10-ps

## Getting Ready

Specific stuff you need to do this recipe

## How to do it...

1. Step by step

## How it works...

1. Screen shots for each step that generates one

## There's more...

1. Some things of interest in this recipe
2. Repeat the recipe structure