14

Troubleshooting Windows Server

In this chapter, we cover the following recipes:

* Checking network connectivity
* Using Best Practices Analyzer
* Managing event logs

# Introduction

Troubleshooting is the art and science of discovering the cause of some problem in your organization's computing estate and providing a solution that overcomes the problem. Troubleshooting encompasses a variety of tasks, many of which you can perform using PowerShell.

With applications and services increasingly being networked, network connectivity can be a problem in many organizations. In the first recipe, you look at some commands that can help you troubleshoot this area.

Microsoft has built a troubleshooting framework into both Windows 10 and into Server 2019. These troubleshoots enable common problems to be resolved by an IT pro just by running the troubleshooter.

Troubleshooting is not just what you do when an issue arises. It also involves being proactive to avoid small issues becoming major problems. Often, it also means ensuring that your systems and services are set up by way of accepted best practice. The Exchange and Office Communications Server (later known as Lync and Skype For Business) teams both produced Best Practice Analyzer (BPA) tools. These were applications that examined your Exchange or OCS (Lync/Skype for Business) environment and showed you places where you have not configured your application based on best practices. With Windows Server, many of the Windows features have their own BPA tools built around a common framework and are powered by PowerShell. The BPA tools can help you to ensure that the features installed on your Windows servers are operating according to best practices.

A great feature of Windows and Windows applications, roles, and services is the sheer amount of information that's logged. Windows NT (which is the basis for both Windows 10 and Windows Server 2019) initially came with a number of base event logs. In Windows Vista, Microsoft extended the amount of logging with the addition of application and service logs.

These event logs contain a wealth of additional information that is invaluable in terms of both troubleshooting after the fact and being proactive. It is certainly the case that getting information out of these logs is a bit like looking for a needle in a haystack. PowerShell has some great features for helping you find the information you need quickly and easily.

# Checking network connectivity

One of the first things you can do in terms of troubleshooting is to determine whether you have network connectivity between your hosts.

## Getting ready

This recipe uses servers in the Reskit.Org domain (DC1, DC2, SRV1, and SRV2)that you have previously installed. Run this recipe on SRV1.

## How to do it...

1. Use Test-Connection to test the connection to DC1:

Test-Connection -ComputerName DC1

1. Redo the test with a simple true/false return:

Test-Connection -ComputerName DC1 -Quiet

1. Test multiple systems at once:

Test-Connection -ComputerName 'DC1','DC2','SRV2' -Count 1

1. Test the connectivity for SMB traffic with DC1:

Test-NetConnection -ComputerName DC1 -CommonTCPPort SMB

1. Get a detailed connectivity check by using DC1 with HTTP:

$TNCHT = @{

ComputerName = 'DC1'

CommonTCPPort = 'HTTP'

InformationLevel = 'Detailed'

}

Test-NetConnection @TNCHT

1. Look for a particular port (that is, SMB on DC1):

Test-NetConnection -ComputerName DC1 -Port 445

1. Look for a host that does not exist:

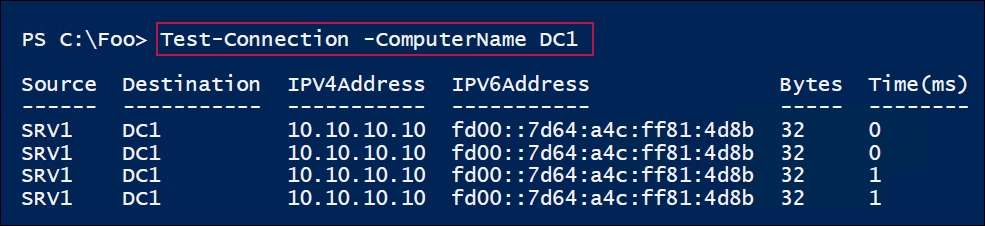
Test-NetConnection -ComputerName 10.10.10.123

1. Look for a host that exists but a port/application that does not exist:

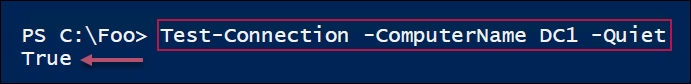
Test-NetConnection -ComputerName DC1 -PORT 9999

## How it works...

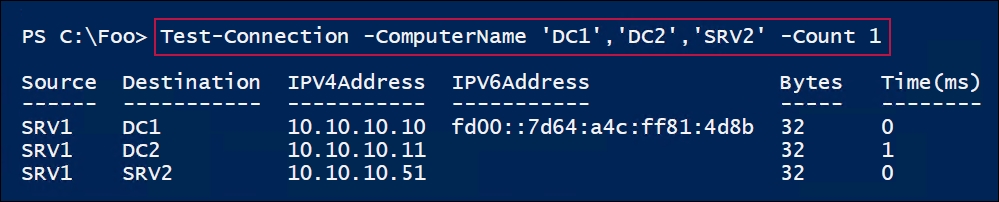
In step 1, you checked the network connectivity from SRV1 to DC1 using the Test-NetConnection cmdlet, which looks like this:



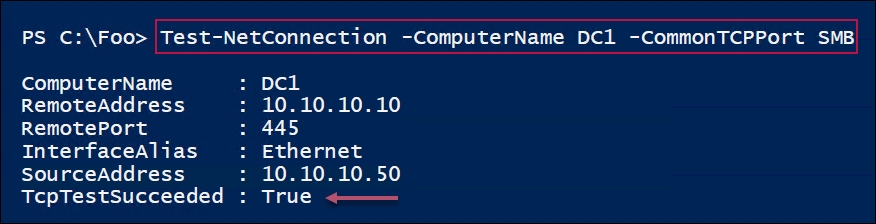
In step 2, you repeated this test using the -Quiet switch, which looks like this:



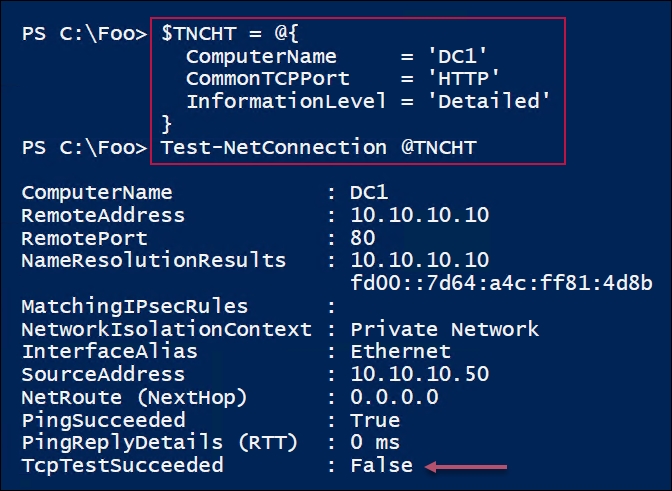
In step 3, you used Test-Connection to test connections from SRV1 to multiple remote servers, which looks like this:



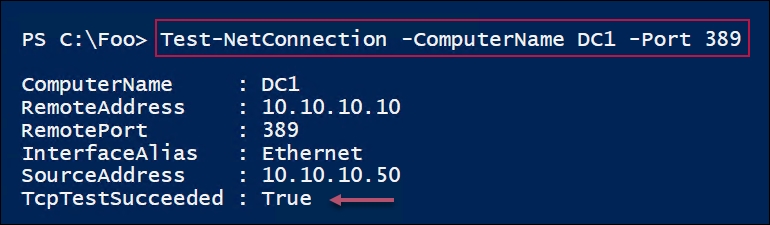
The Test-NetConnection cmdlet provides additional parameters. In step 4, you tested whether SRV1 can reach the SMB server service on DC1. SRV1 uses this port to download group policy details. The output looks like this:



You can also get a more detailed report of attempted connectivity. You can use the -InformationLevel parameter set to detailed so that you receive more information on the attempt to connect. In step 5, you checked whether SRV1 can create a HTTP connection with DC1, as follows:



In step 6, you used Test-NetConnection to test the connectivity to a numbered port, port 389 on DC1. Port 389 is the LDAP port that an AD client uses to talk to a domain controller. The output of this step looks like this:



## There's more...

In step 2, you used the -Quiet switch with Test-NetConnection. This directs the cmdlet to just return a true or false value after attempting to connect to the remote system. This can be useful in scripts where you only need to check whether a server is contactable.

# Using the Best Practices Analyzer

A best practice is a way of doing things that is considered by others (generally more experienced in the area) to provide the best result. For example, a best practice is to always have at least two domain controllers in case one goes down.

Following a best practice can both solve existing issues and avoid future ones, but a bit of common sense is needed to ensure that you are following the advice that is relevant for you and your organization. In a small test lab of a few VMs, having a second DC may not be needed.

The BPA is an automated tool that's built into Windows. With BPA, a best practice model is a set of specific guidelines for a single area. BPA reviews your infrastructure and points out areas where the environment is not compliant with the best practice model.

The Windows BPA framework provides PowerShell support for managing the BPA process. Windows and applications come with a number of BPA models, generally built by the relevant product group within the Windows Team. The PowerShell cmdlets let you find the BPA models, invoke them, and then view the results. Since not all BPA model guidelines are relevant for all situations, the BPA feature also lets you ignore specific recommendations that are not relevant to you.

## Getting ready

You run this recipe on SRV1, a server that was used in recipes earlier in this book. This recipe requires IIS (the web server feature) to be loaded. Refer to [Chapter 9](file:///C:\Users\siddh\OneDrive\Desktop\PEN\September%2022%20-%20Parvathy%20-%20Chapter%201-12%20-%20STY\9781789808537\OEBPS\ch09.html), Managing Windows Internet Information Server, the Installing IIS recipe for details on how to install the web server feature.

You also use DC1 in this recipe.

## How to do it...

1. Get all BPA models on SRV1:

Get-BpaModel |

Format-Table -Property Name, Id, LastScanTime -Wrap

1. Invoke a BPA model for the WebServer feature:

Invoke-BpaModel -ModelId Microsoft/Windows/WebServer

1. Get the results of the BPA run:

$Results = Get-BpaResult -ModelId Microsoft/Windows/webServer

1. Display how many tests/results are in the BPA model:

$Results.Count

1. How many errors and warnings were found?

$Errors = $Results | Where-Object Severity -eq 'Error'

$Warnings = $Results | Where-Object Severity -eq 'Warning'

"Errors found : {0}" -f $Errors.Count

"Warnings found : {0}" -f $Warnings.Count

1. Look at other BPA results:

$Results | Format-Table -Property Title, Compliance -Wrap

1. Use BPA remotely—what BPA models exist on DC1?

Invoke-Command -ComputerName DC1 -ScriptBlock {Get-BpaModel} |

Format-Table -Property Name, Id

1. Run BPA Analyzer on DC1:

$ModelId = 'Microsoft/Windows/DirectoryServices'

$SB = {Invoke-BpaModel -ModelId $using:ModelId}

Invoke-Command -ComputerName DC1 -ScriptBlock $SB

1. Get the results of the DirectoryServices BPA model from DC1:

$SB = {Get-BpaResult -ModelId Microsoft/Windows/DirectoryServices}

$RRESULTS = Invoke-Command -ComputerName DC1 -ScriptBlock $SB

1. Review the results returned from the scan:

"Total results returned: $($RResults.Count)"

$RResults | Group-Object SEVERITY |

Format-Table -Property Name, Count

1. View the error(s) from the scan:

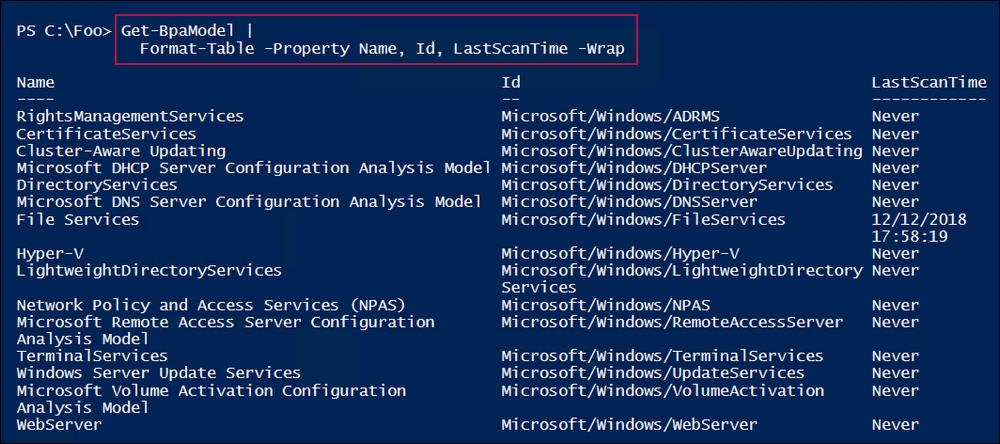
$RResults |

Where-Object Severity -EQ 'Error' |

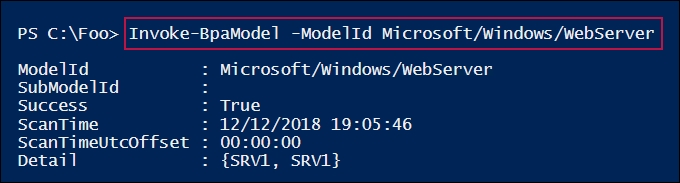
Format-List -Property Category,Problem,Impact,Resolution

## How it works...

In step 1, you obtained and displayed the details about the BPA models on the SRV1 host, which looks like this:

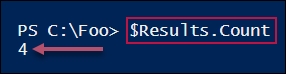


As IIS is installed on SRV1, in step 2, you ran the BPA model for the Windows WebServer feature. The output looks like this:

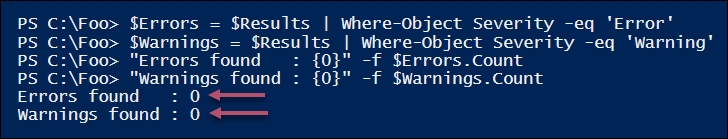


In step 3, you retrieved the results of the most recent invocation of the WebServer BPA model and stored it in $Results. This step produces no output.

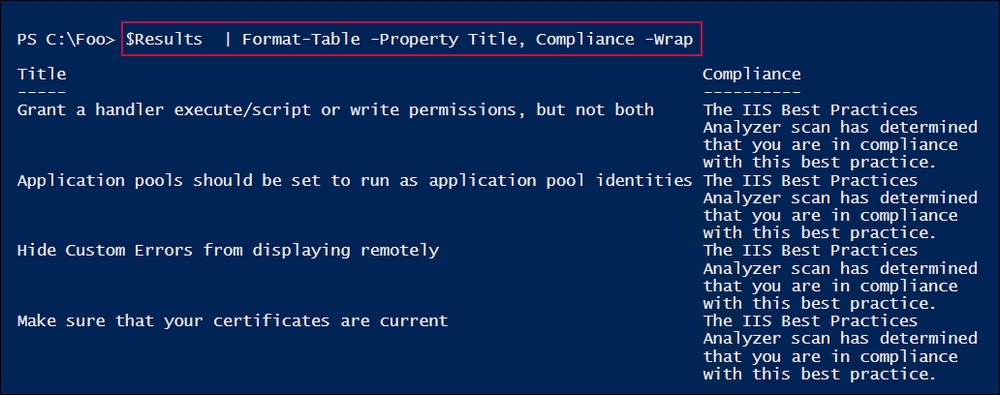
In step 4, you displayed a count of the number of BPA results returned from the WebServer BPA scan, which looks like this:



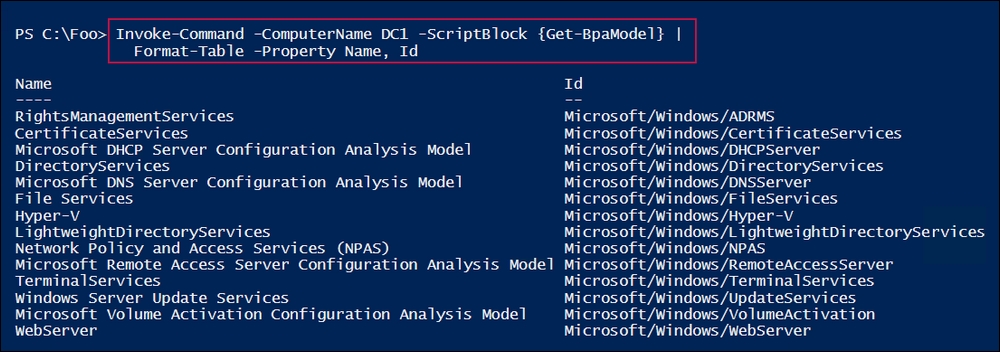
Next, in step 5, you counted and displayed the number of error or warning results that were returned by the BPA scan, as follows:



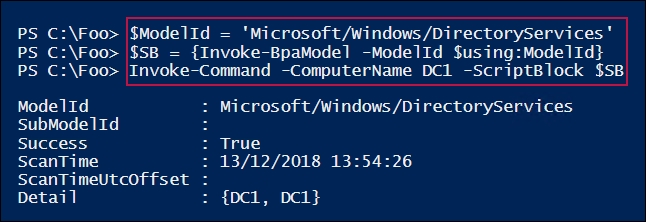
In step 6, you viewed the specific items that were tested by the WebService BPA model, and the compliance status of SRV1, which looks like this:



You can also use BPA models remotely. In step7, you viewed the BPA models on DC1, a domain controller, which looks like this:

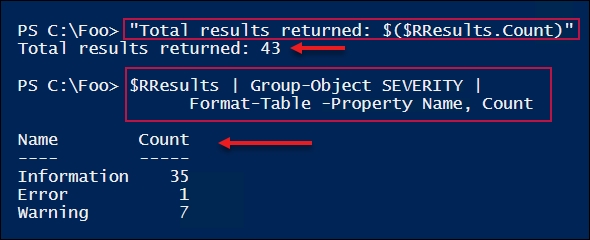


In step 8, you ran the DirectoryServices model remotely on DC1. The output looks like this:

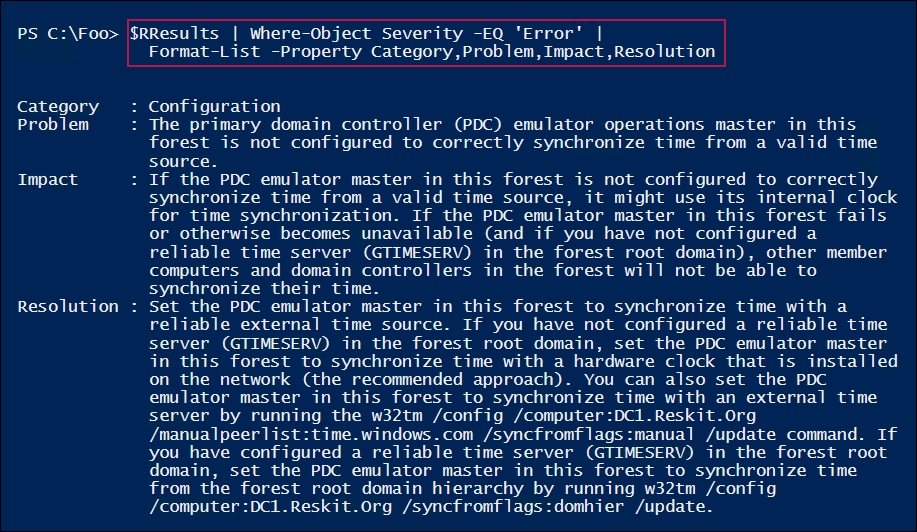


Having run the DirectoryServices BPA model on DC1, in step 9, you retrieved the results. This step produces no output.

In step 10, you examined the BPA results. You displayed the number of BPA results and what kinds of results the BPA scan of DC1 reveals, which looks like this:



The BPA results show one error and seven warnings out of 43 BPA checks on DC1. While you should investigate the warnings, you may find some of the BPA warnings can be ignored in your environment. The BPA error results should be prioritized. In our case, the error result, which you obtained in step 11, looks like this:



## There's more...

In step 1, you saw the BPA models on SRV1. Depending on which other features you added to SRV1, you may see more BPA models.

In step 4, you can observe that the + BPA model checks just four configuration settings for IIS. There are not a lot of BPA checks being done by this model. Other BPA models, such as the DirectoryServices model, which you used in step 9, are much more detailed.

In step 11, you can see that when you run the DirectoryServices BPA model on DC1, BPA reports an error. The error result object includes a description of the problem, the impact, and how to resolve the issue. In this case, the problem that was found was that there was a lack of time synchronization between your forest root DC (DC1.Reskit.Org) and an external (and reliable) time source. Since all hosts in your forest ultimately get their time settings from the forest root server, it is important that the forest root server is synchronized with a reliable, external time source.

# Managing event logs

Event logs are an important troubleshooting asset. Windows and Windows applications can log a significant amount of information that can be invaluable in both troubleshooting and in the day-to-day administration of Windows Server 2019.

Windows computers maintain a set of event logs that document events that occur on a given machine. Any time an event occurs, the application or service can log events that can then be used to help in the debugging process.

In Windows, there are two types of event logs: Windows logs and application and services logs. Windows logs began with Windows NT 3.1 and continue in Windows Server 2019 and are important components in troubleshooting and system monitoring.

Windows Vista added a new category of logs, application and services logs. These logs contain events that are within a single application, service, or other Windows component. Windows comes, by default, with a set of application and service logs—adding components such as new Windows features or roles often results in additional application and service logs.

These logs give you a great picture of what your system is actually doing. Additionally, you can also add new event logs and enable scripts to log events that occur while the script is running.

PowerShell provides you with several useful cmdlets to help you comb the event log looking for key events. The Get-EventLog enables you to get details of the logs that exist as well as retrieving log events from the Windows logs. With Get-WinEvent, you can examine both the classic Windows logs and the new application and services logs. You use both these cmdlets in this recipe.

## Getting ready

You run this recipe from SRV1, a domain joined server in the Reskit.Org domain. You also use the domain controller DC1. Reskit.Org in this recipe.

## How to do it...

1. Get the core event logs on SRV1:

Get-EventLog -LogName \*

1. Get the remote classic event logs from DC1:

Get-EventLog -LogName \* -ComputerName DC1

1. Clear the application log on DC1:

Clear-EventLog -LogName Application -ComputerName DC1

1. Look at the types of events on SRV1:

Get-EventLog -LogName Application |

Group-Object -Property EntryType |

Format-Table -Property Name, Count

1. Examine which area created the events in the application log:

Get-EventLog -LogName System |

Group-Object -Property Source |

Sort-Object -Property Count -Descending |

Select-Object -First 10 |

Format-Table -Property Name, Count

1. Examine all the event logs on SRV1:

$LocEventLogs = Get-WinEvent -ListLog \*

$LocEventLogs.Count

$LocEventLogs |

Sort-Object -Property RecordCount -Descending |

Select-Object -First 10

1. Examine all the event logs on DC1:

$RemEventLogs = Get-WinEvent -ListLog \* -ComputerName DC1

$RemEventLogs.count

$RemEventLogs |

Sort-Object -Property RecordCount -Descending |

Select-Object -First 10

1. Look at the WindowsUpdateClient operational event log on the localhost and discover the updates that the WU client has found:

$LN = 'Microsoft-Windows-WindowsUpdateClient/Operational'

$Updates = Get-WinEvent -LogName $LN |

Where-Object ID -EQ 41

$Out = foreach ($Update in $Updates) {

$HT = @{}

$HT.Time = [System.DateTime] $Update.TimeCreated

$HT.Update = ($Update.Properties | Select-Object -First 1).Value

New-Object -TypeName PSObject -Property $HT

}

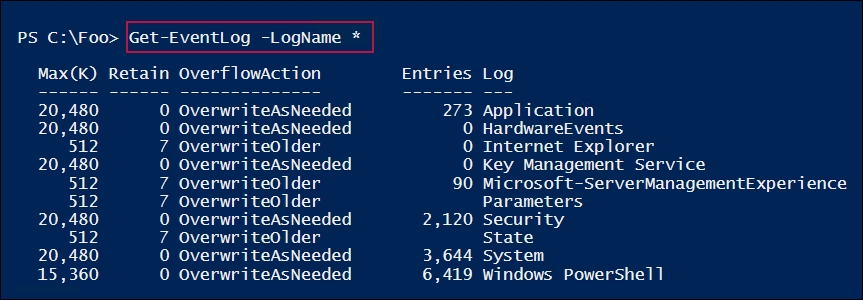
$Out | Sort -Property Time

Sort-Object -Property TimeCreated |

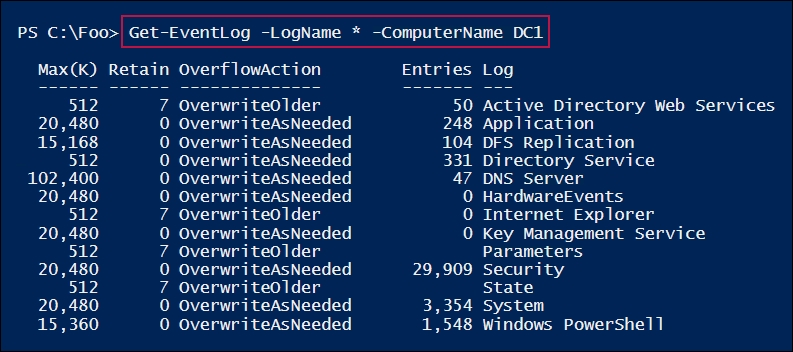
Format-Table -Wrap

## How it works...

In step 1, you used the Get-EventLog cmdlet to display the core event logs on SRV1, which looks like this:

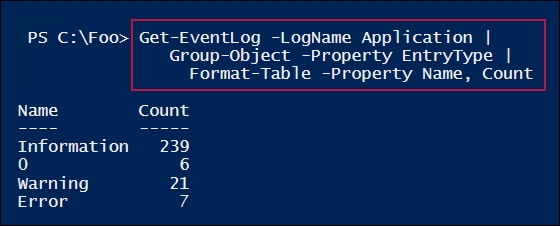


In step 2, you retrieved the core event logs remotely on DC1, which looks like this:

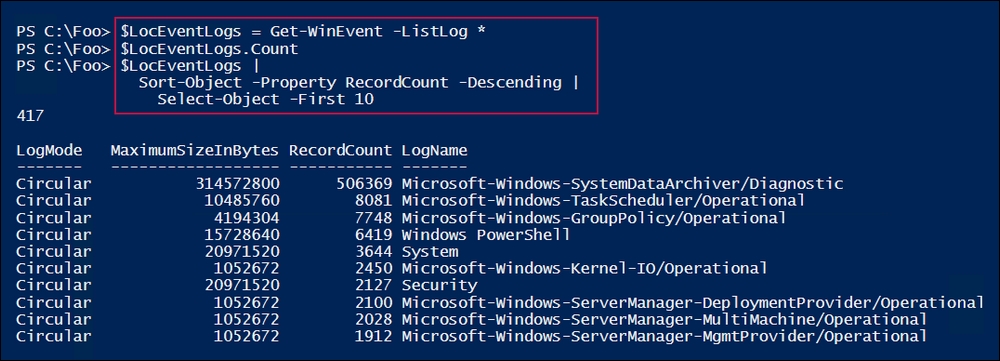


In step 3, you cleared the application log remotely on DC1. There is no output from this step.

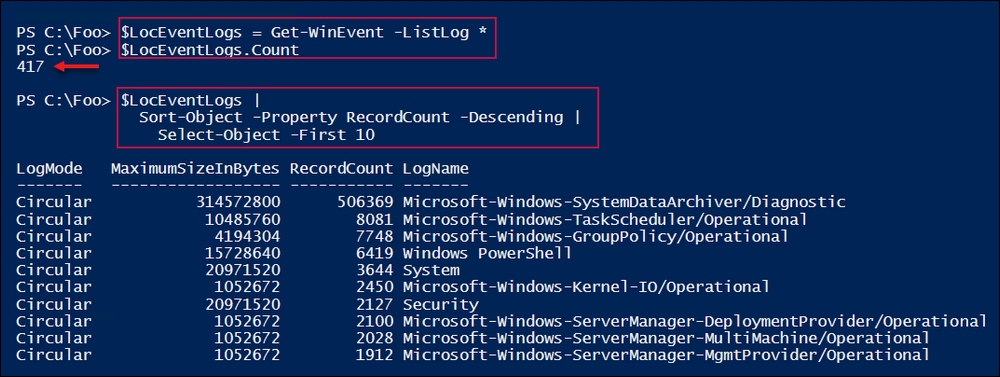
In step 4, you examined the event types that are in the event log on SRV1, which looks like this:



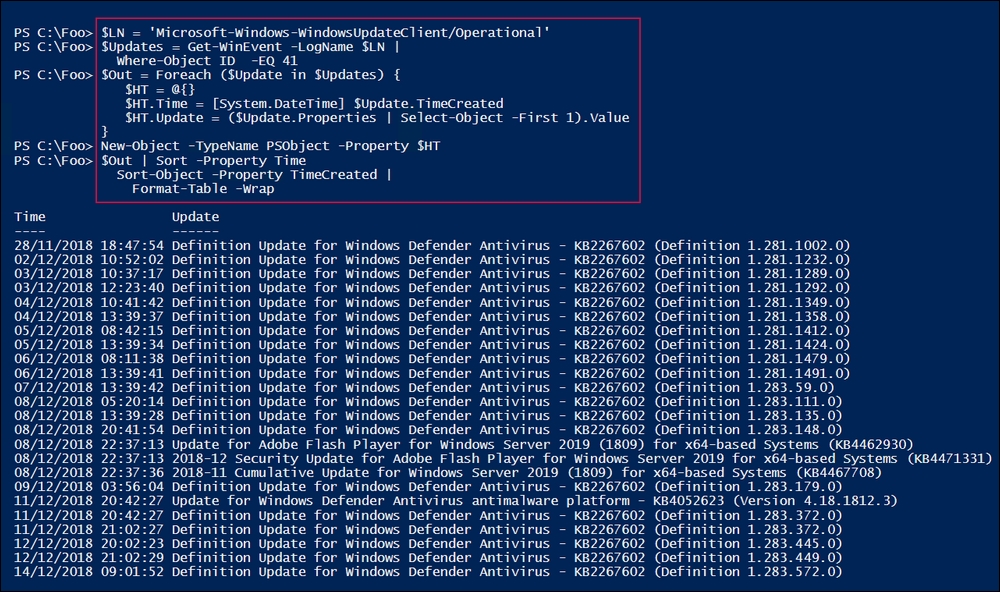
In step 5, you looked at the Windows event logs, using Get-EventLog to get all the different Windows components that are logging events to the System log. This step then displays the ten areas that are logging the most entries on SRV1, which looks like this:



In step 6, you displayed the total number of event logs found by Get-WinEvent, and then you listed the 10 busiest extended logs, as follows:



In step 8, you displayed the number of Windows updates that were discovered (and installed) on SRV1, which looks like this:



## There's more...

In step 1 and step 2, you examined the Windows event logs that exist on two systems (SRV1 and DC1). As you can see, the logs that are available differ—on DC1, you can see the Active Directory Web Services log, which does not exist on SRV1.

In step 3, you cleared the application log on DC1. As a best practice for event logs, you should only clear a log once you have copied the log elsewhere for safe keeping. Naturally, mileage varies on this point, since the vast majority of event log entries are not of very much use in day-to-day operations.

In step 4, you saw the different classifications of events, including one with a name of 0. In this case, the property containing the event log entry type is based on an enum, and this enum was not updated, so PowerShell was unable to display the entry name for this event log entry type.

In step 6 and step 7, you examined the service and application logs that exist on SRV1. These steps demonstrate how additional features or applications can result in additional event logs.

Step 8 showed you how to dive into a specific event in a specific event log. In this case, you examined the Software Update service's operational log to discover events with an event ID of 41. In general, when retrieving information from your event logs, you need to know which log and which event ID to look for.