6

Managing Active Directory

In this chapter, we cover the following recipes:

* Installing an Active Directory forest root domain
* Testing an AD installation
* Installing a replica domain controller
* Installing a child domain
* Creating and managing AD users and groups
* Managing Active Directory computers
* Adding/removing users using CSV files
* Creating a Group Policy Object for user Logon script
* Reporting on AD computers
* Reporting on AD users

# Introduction

A core component of almost all organizations’ IT infrastructure is Active Directory (AD). AD provides access control, user and system customization, and a wealth of directory and other services. Microsoft first introduced AD with Windows 2000 and has improved and expanded the product with each successive release of Windows Server.

Over the years, Microsoft has made “AD” more of a brand than a single feature. At the cores is Active Directory Domain Services (AD DS). There are four additional Windows Server features under the AD brand:

* **AD Certificate Services (AD-CS)** - this allows you to issue X.509 certificates for your organization. For an overview of AD-CS, see https://docs.microsoft.com/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831740(v=ws.11).
* **AD Federation Services (AD-FS)** - this feature enables you to federate identity with other organisations to facilitate interworking. You can find an overview of AD-FS at https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831502(v=ws.11).
* **AD Lightweight Directory Services (AD-LDS)** - This provides rich directory services for use by applications. You can find an overview of AD-LDS at https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831593(v=ws.11).
* **AD Rights Management Services (AD-RMS)** - RMS enables you to control the rights to document access to limit information leakage. For an overview of RMS, see https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831364(v=ws.11).

Note that the overview documents referred to above are older documents based on Windows Server 2012. At the time of writing, the documentation teams have not updated them fully to reflect the latest Windows Server version. The overview and the essential operation of these features remain mostly unchanged.

Active Directory’s domain service is complex, and there are a lot of moving parts. With AD, you have a logical structure consisting of Forests, Domains, Domain Trees and Organisational Units. You also have the physical structure, including Domain Controllers and global catalogs. There is also a replication mechanism to replicate objects across your domain.

A forest is a top-level container that houses domains. A forest is a security boundary, although you can set up cross-forest trusts to enable interworking between multiple forests. AD bases forest (and domain) names on DNS.

A domain is a collection of objects, including users, computers, policies, and much more. You create a forest by installing the forest’s first domain controller. In AD domains, trees are collections of domains that you group in a hierarchical structure. Most organisations use a single domain (and domain tree) within a single forest. Multiple domains in one or more domain trees are also supported, but best practice is to avoid them.

A domain controller (DC) is a Windows Server running AD and holding the objects for a given domain. ALL domains must have at least one DC, although best practice is always to have at least two. You install the AD DS service onto your server, then promote the server to be a DC.

The global catalog (GC) is a partial replica of objects from every domain in an object to enable searching. Exchange, for example, uses the GC heavily. You can have the GC service on some or all DCs in your forest. Generally, you install the GC facility while you promote a Windows Server to be a DC.

Using AD Domain Services (or AD) and PowerShell, you can deploy your domain controllers throughout your organization. Use the “Installing Active Directory and forest root domain” recipe to install a forest root domain controller and establish an AD Forest.

Installing features and services using PowerShell in a domain environment often uses remoting, which in turn requires authentication. From one machine, you use PowerShell remoting to perform operations on other systems. You need the correct credentials for those operations. Sometimes, you may encounter the well understood Kerberos double hop problem. In “Configuring CredSSP”, you install the Credential Security Subsystem Provider to manage the issue. For a better explanation of this issue see https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting/ps-remoting-second-hop?view=powershell-7.1.

Once you have a first DC in your Reskit.Org forest, you should add replica DC to ensure reliable domain operations. In “Installing a replica domain controller”, you add a second DC to your domain. In “Installing a child domain”, you extend the forest and add a child domain to your forest.

AD uses a database of objects that include Users, Computers, and Groups. In the “Managing AD users and groups” recipe, you create, move, and remove user and group objects and create and use Organizational Units (OUs). In “Managing Active Directory computers” you manage the computers in your Active Directory, including joining workgroup systems to the domain. In the “Adding/Removing users using CSV files” recipe, you add users to your AD using a comma-separated value file containing users’ details.

Group policy is another important feature of Active Directory. With Group Policy, you can define policies for users and computers that Windows applies automatically to the user and/or computer. In the “Creating a Group Policy Object for user logon script” recipe, you create a simple GPO and observe applying that policy.

Active Directory can make use of multiple DCs for both load balancing and fault tolerance. Such DCs must be synchronized whenever changes are made, and AD Replication performs that function. In “Managing AD replication” you examine tools to help you manage and troubleshoot replication.

In the recipes “Reporting on AD computers”, and “Reporting on AD Users” you examine the AD to find details on computers that have not started up or logged onto the domain. You also look at user accounts for users who are members of special security groups (such as enterprise administrators). These two recipes help you to keep your AD free of stale objects or objects that could represent a security risk.

Putting a user into a high privilege group is a practical approach to enabling users to perform administrative tasks. Doing so, however, can open up security issues by giving someone too much privilege. Just Enough Administration (JEA) is a mechanism that allows you to enable very fine-grained administration. You can allow a user to logon remotely but to perform a minimal set of actions. You could define a DNS Administrators group whose members can logon to a DNS server (such as you implemented on DC1) and manage DNS but nothing more. In “Implementing JEA”, you set up and deploy JEA.

# Installing an Active Directory forest root domain

You create an AD forest by creating your first domain controller. Installing Active Directory and DNS has always been reasonably straightforward. You can always use the Server Manager GUI, but using PowerShell is also straightforward.

To create a DC, you start with a system running Windows Server. You then add the AD DS services Windows feature to the server. Finally, you create your first DC, for example, a single domain controller, DC1.Reskit.Org, for the Reskit.Org domain.

## Getting Ready

You run this recipe on DC1, a workgroup server on which you have installed PowerShell 7 and VS Code.

## How to do it...

1. Install the AD Domain Services feature and management tools:

Install-WindowsFeature -Name AD-Domain-Services -IncludeManagementTools

1. Import the ADDSDeployment module:

Import-Module -Name ADDSDeployment

1. Examine the commands in the ADDSDeployment module:

Get-Command -Module ADDSDeployment

1. Create a secure password for Administrator:

$PSSHT = @{

  String      = ’Pa$$w0rd’

  AsPlainText = $true

  Force       = $true

}

$PSS = ConvertTo-SecureString @PSSHT

1. Test DC forest installation starting on DC1:

$FOTHT = @{

  DomainName           = ’Reskit.Org’

  InstallDNS           = $true

  NoRebootOnCompletion = $true

  SafeModeAdministratorPassword = $PSS

  ForestMode           = ’WinThreshold’

  DomainMOde           = ’WinThreshold’

}

Test-ADDSForestInstallation @FOTHT -WarningAction SilentlyContinue

1. Create forest root DC on DC1:

$ADHT = @{

  DomainName                    = ’Reskit.Org’

  SafeModeAdministratorPassword = $PSS

  InstallDNS                    = $true

  DomainMode                    = ’WinThreshold’

  ForestMode                    = ’WinThreshold’

  Force                         = $true

  NoRebootOnCompletion          = $true

  WarningAction                 = ’SilentlyContinue’

}

Install-ADDSForest @ADHT

1. Check key AD and related services:

Get-Service -Name DNS, Netlogon

1. Check DNS Zones:

Get-DnsServerZone

1. Restart DC1 to complete promotion:

Restart-Computer -Force

## How it works...

In step 1, you install the AD Domain Services feature. This feature enables you to deploy a server as a domain controller. The output of this command looks like this:

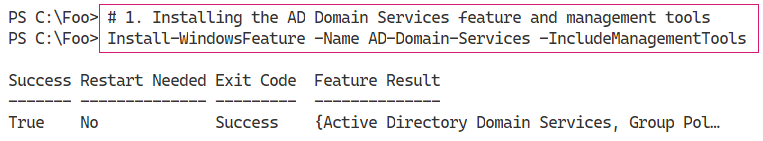


Figure 6.1: Installing the AD DS feature

1. Insert image B42024\_06\_01.png

In step 2, you manually import the ADDSDeployment module. Since this module is not supported natively by PowerShell 7, this step loads the module using the Windows PowerShell compatibility feature. The output of this command looks like this:

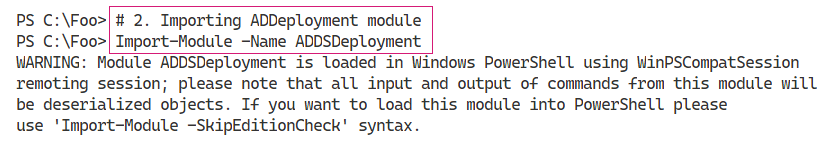


Figure 6.2: Importing the ADDeployment module

1. Insert image B42024\_06\_02.png

In step 3, you use the Get-Command cmdlet to discover the commands contained in the ADDSDeployment module, which looks like this:

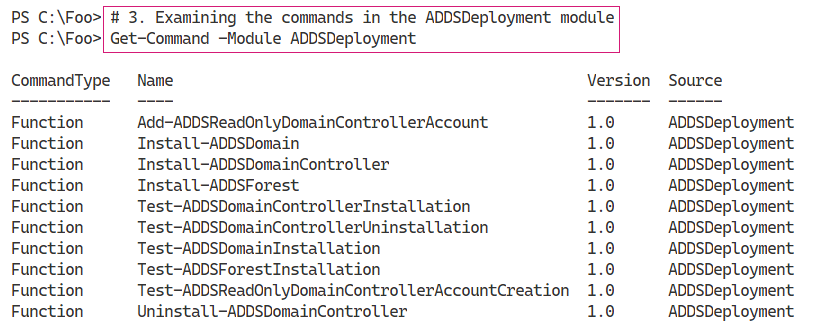


Figure 6.3: Examining commands in the ADDSDeployment module

1. Insert image B42024\_06\_03.png

With step 4, you create a secure string password to use as the Administrator password in the domain you are creating. This step produces no output.

Before you promote a server to be a DC, it’s useful to test to ensure that a promotion would be successful as far as possible. In step 5, you use the Test-ADDSForestInstallation command to check whether you can promote DC1 to be a DC in the Reskit.Org domain. The output of this command looks like this:

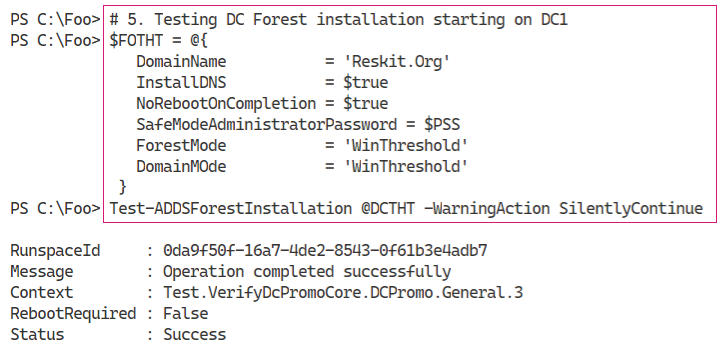


Figure 6.4: Testing DC forest installation

1. Insert image B42024\_06\_04.png

In step 6, you promote DC1 to be the first domain controller in a new domain, Reskit.Org. The output looks like this:

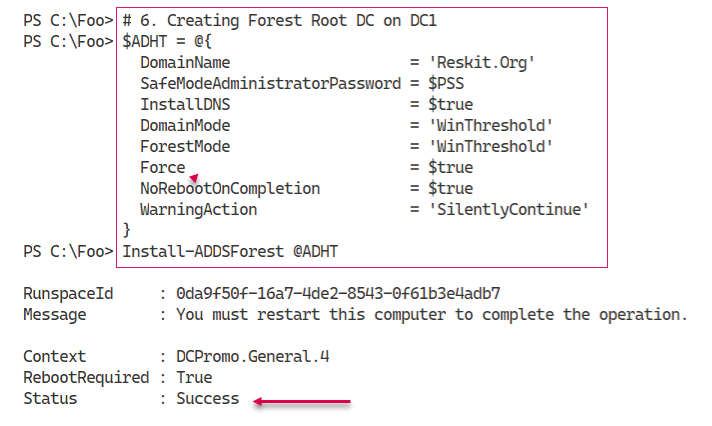


Figure 6.5: Creating forest root DC on DC1

1. Insert image B42024\_06\_05.png

After the promotion is complete, you can check the critical services required for Active Directory. Checking the Netlogon and DNS Services, which you do in step 7, should look like this:

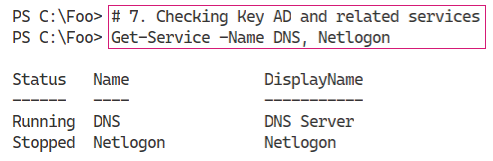


Figure 6.6: Checking the Netlogon and DNS services

1. Insert image B42024\_06\_06.png

When you promoted DC1, you also requested the promotion to install DNS DC1. In step 8, you check on the zones created by the DC promotion process, which looks like this:

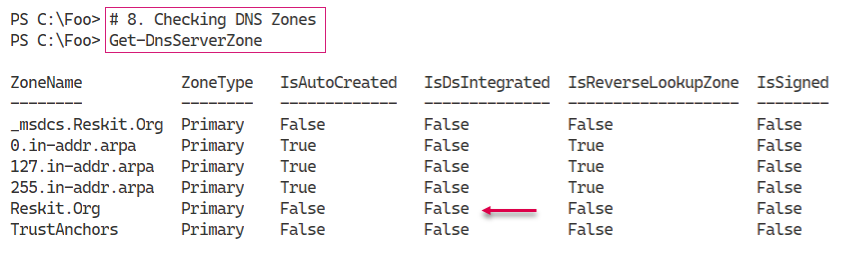


Figure 6.7: Checking DNS zones

1. Insert image B42024\_06\_07.png

To complete the promotion process, you need to reboot, which you do in step 9, generating no actual output.

## There’s more...

The cmdlets that enable you to promote a server to be a DC are not installed on a server system by default. Adding the Active Directory Domain Services Windows feature, in step 1, adds the necessary cmdlets to the system.

In step 6, you install AD and direct that a DNS server should also be installed - and you check for its presence in step 7. In step 8, you view the DNS zones created automatically by the promotion process. You specify the DNS domain name, Reskit.Org, using the DomainName parameter to Install‑ADDSForest. This DNS domain created but is, at this point, still a NON-AD integrated zone. Once you reboot the service, this zone should become AD-integrated (and set for secure updates only).

Once you complete the verification of a successful AD installation, you reboot the server. After the restart, there are further tests that you should run, as we show in the next recipe, “Testing an AD installation.”

# Testing an AD installation

In “Installing an Active Directory Forest Root Domain”, you installed AD on SRV1. In that recipe, you installed AD (without rebooting) and tested certain services. After the required reboot (which you completed at the end of the previous recipe), it is useful to check to ensure that your domain is fully up, running, and working correctly. In this recipe, you examine core aspects of the AD infrastructure on your first DC.

## Getting Ready

You run this recipe on DC1, the first domain controller in the Reskit.Org domain after you have promoted it to be a DC. Log on as Reskit\Administrator.

## How to do it...

1. Examine AD root directory Service Entry (DSE):

Get-ADRootDSE -Server DC1.Reskit.Org

1. View AD forest details:

Get-ADForest

1. View AD domain details:

Get-ADDomain

1. Check Netlogon, ADWS, and DNS services:

Get-Service NetLogon, ADWS, DNS

1. Get initial AD users:

Get-ADUser -Filter \* |

  Sort-Object -Property Name |

    Format-Table -Property Name, DistinguishedName

1. Get initial AD groups:

Get-ADGroup -Filter \*  |

  Sort-Object -Property Groupscope, Name |

    Format-Table -Property Name, GroupScope

1. Examine Enterprise Admins group membership:

Get-ADGroupMember -Identity ’Enterprise Admins’

1. Check DNS zones on DC1

Get-DnsServerZone -ComputerName DC1

1. Test domain name DNS resolution:

Resolve-DnsName -Name Reskit.Org

## How it works...

After you completed the installation of AD, and rebooted, in step 1, you examine the AD Directory Service Entry (DSE), which looks like this:

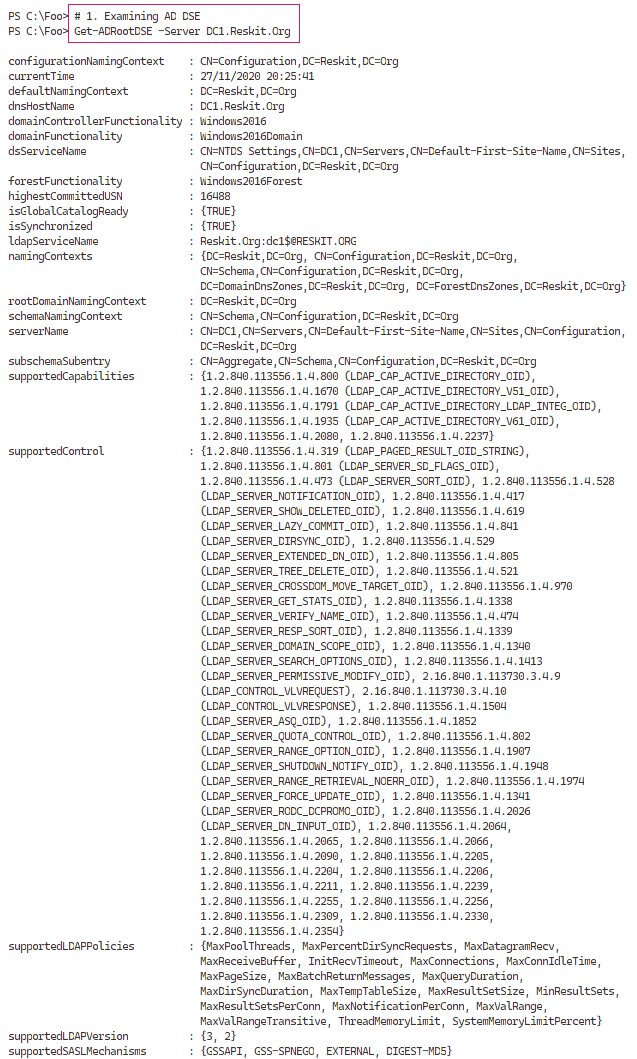


Figure 6.8: Examining the AD Directory Service Entry (DSE)

1. Insert image B42024\_06\_08.png

In step 2, you use the Get-ADForest command to review further information on the newly created forest, which looks like this:

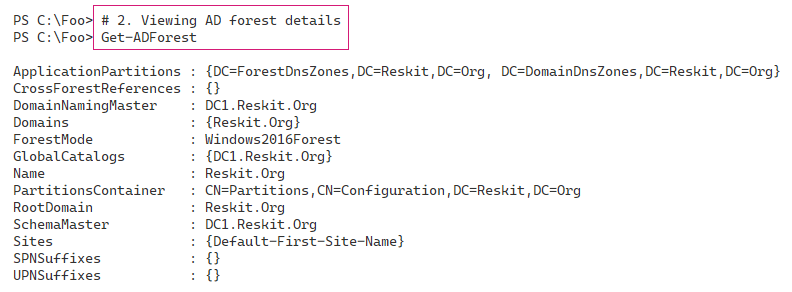


Figure 6.9: Viewing the AD forest details

1. Insert image B42024\_06\_09.png

In step 3, you use Get-ADDomain to return more information about the Reskit.Org domain, which looks like this:



Figure 6.10: Viewing AD domain details

Insert image B42024\_06\_10 .png

The AD services run within the Netlogon Windows Service. The ADWS service is a web service used by the PowerShell AD cmdlets to communicate with the AD. AD relies on DNS as a locator service enabling AD clients and DCs to find DCs. All three must be up and running for you to manage AD using Powershell. In step 4, you check all three services which look like this:

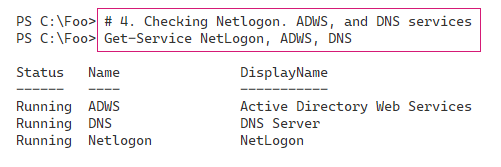


Figure 6.11: Checking the NetLogon, ADWS, and DNS services

Insert image B42024\_06\_11 .png

When you promote a Windows server to be a DC, the promotion process creates three users: Administrator, Guest and the krbtgt (Kerberos Ticket-Granting Ticket) user. You should never touch the krbtgt user and should usually leave the Guest user disabled. For added security, you can rename the Administrator user to something less easy to guess and create a new, very low privilege, user with the name Administrator.

In step 5, you examine the users in your newly created AD forest, which looks like this:

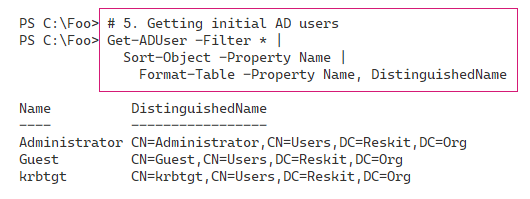


Figure 6.12: Getting initial AD users in the new AD forest

Insert image B42024\_06\_12 .png

The DC promotion process also creates several groups that can be useful. In step 6, you examine the different groups created (and their scope), which looks like this:

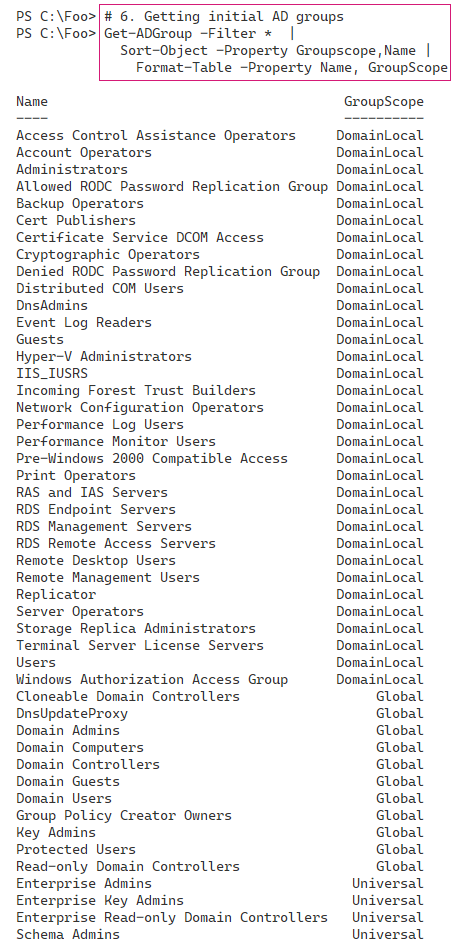


Figure 6.13: Examining initial AD groups and their scope

Insert image B42024\_06\_13 .png

The Enterprise Admin group is one with very high privilege. Members of this group can perform just about any operation across the domain - stop/start services, modify the AD, and access any file or folder on the domain or any domain-joined system. In step 7, you examine the initial members of this high-privilege group, which looks like this:

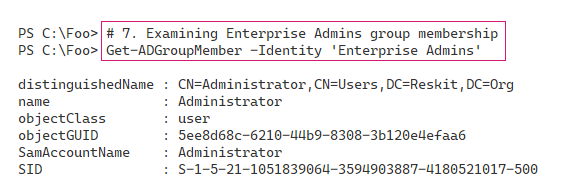


Figure 6.14: Examining Enterprise Admin group membership

Insert image B42024\_06\_14 .png

AD relies on DNS to enable an AD client or AD DC to find DCs. When you install a domain controller, you can also install the DNS service at the same time. When you install a DC with DNS, the AD Promotion process creates several DNS Zones in your newly created DNS service. In step 8, you examine the DNS Zones created on DC1, which looks like this:

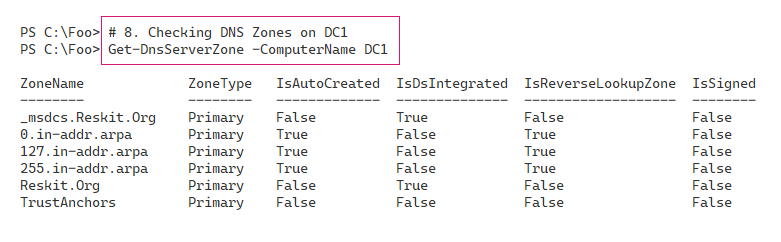


Figure 6.15: Checking the DNS Zones created on DC1

Insert image B42024\_06\_15 .png

Another good test of your newly promoted DC is to ensure you can resolve the DNS name of your new domain (Reskit.Org). In step 9, you use the Resolve-DnsName to check, which looks like this:

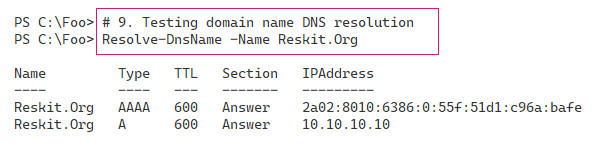


Figure 6.16: Testing domain name DNS resolution

Insert image B42024\_06\_16 .png

## There’s more

In step 1, you viewed the DSE for your domain. The DSE is a component of LDAP directories and contains information about your directory structure. The DSE is available without requiring authentication and contains much information about your AD forest. That information could help an attacker; thus, best practice is never to expose an AD DC to the Internet if you can help it. For a more detailed look at the Root DSE, see https://docs.microsoft.com/windows/win32/adschema/rootdse.

The ADWS service, which you investigate in step 4, implements a web service. The AD commands use this web service to get information from and make changes to your AD. If this service is not running, AD commands do not work. You should always check to ensure the service has started before proceeding to use the AD cmdlets.

In step 6, you saw the groups created by the promotion process. These groups have permissions associated and thus are useful. Before adding users to these groups, consider reviewing the group and determining (and possibly changing) the permissions and rights assigned to these groups.

# Installing a replica domain controller

In “Installing an Active Directory forest root domain”, you installed AD on DC1. If you have just one DC, then that DC is a single point of failure. In this scenario, when DC1 goes down, you cannot manage AD using the PowerShell cmdlets. It is always a best practice to install at least two DCs. If you are using VMs for your DCs, you should also ensure that each DC VM is on a separate virtualization host.

To add a second DC to your domain, you run Install-ADDSDomainController on another host, for example, DC2. This cmdlet is similar to Install-ADDSForest in terms of parameters. As with creating your first DC, it is useful to carry out some tests to ensure the second DC’s promotion can succeed.

In this recipe, you promote a host, DC2, to be the second DC in the Reskit.Org domain. Like creating your first DC, after you promote DC2 to be a DC, you need to reboot the server before processing. And after the reboot, it is useful to ensure the promotion process was successful.

## Getting Ready

You run this recipe on DC2, a domain-joined server on which you have installed PowerShell 7 and VS Code. You should log into DC2 as Reskit\Administrator, a member of the Enterprise Admins group.

## How to do it...

1. Import the ServerManager module:

Import-Module -Name ServerManager -WarningAction SilentlyContinue

1. Check DC1 can be resolved:

Resolve-DnsName -Name DC1.Reskit.Org -Type A

1. Test the network connection to DC1:

Test-NetConnection -ComputerName DC1.Reskit.Org -Port 445

Test-NetConnection -ComputerName DC1.Reskit.Org -Port 389

1. Add the AD DS features on DC2:

Install-WindowsFeature -Name AD-Domain-Services -IncludeManagementTools

1. Promote DC2 to be a DC:

Import-Module -Name ADDSDeployment -WarningAction SilentlyContinue

$URK    = ”Administrator@Reskit.Org”

$PW     = ’Pa$$w0rd’

$PSS    = ConvertTo-SecureString -String $PW -AsPlainText -Force

$CredRK = [PSCredential]::New($URK,$PSS)

$INSTALLHT = @{

  DomainName                    = ’Reskit.Org’

  SafeModeAdministratorPassword = $PSS

  SiteName                      = ’Default-First-Site-Name’

  NoRebootOnCompletion          = $true

  InstallDNS                    = $false

  Credential                    = $CredRK

  Force                         = $true

  }

Install-ADDSDomainController @INSTALLHT | Out-Null

1. Check the computer objects in AD:

Get-ADComputer -Filter \* |

  Format-Table DNSHostName, DistinguishedName

1. Reboot DC2 manually:

Restart-Computer -Force

1. Check DCs in Reskit.Org:

$SB = ’OU=Domain Controllers,DC=Reskit,DC=Org’

Get-ADComputer -Filter \* -SearchBase $SB |

  Format-Table -Property DNSHostName, Enabled

1. View Reskit.Org domain DCs:

Get-ADDomain |

  Format-Table -Property Forest, Name, Replica\*

## How it works...

In step 1, you import the ServerManager module, which creates no output. With step 2, you ensure that you can resolve your DC’s address, which is, at this point, the only DC in the Reskit.Org domain. The output looks like this:

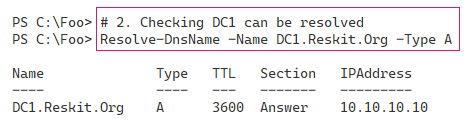


Figure 6.17: Checking DC1 can be resolved

1. Insert image B42024\_06\_17 .png

After confirming that you can resolve the IP address of DC1, in step 3, you check you can connect to two key ports on DC1 (445 and 389). The output looks like this:

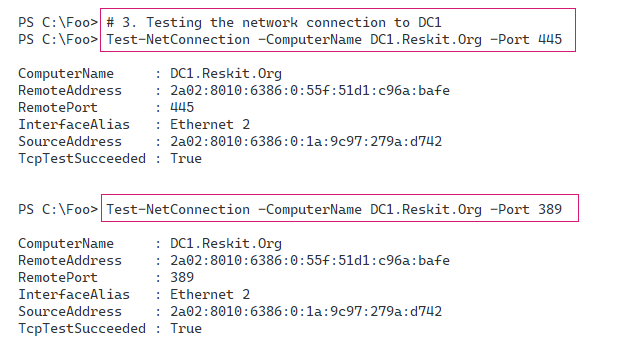


Figure 6.18: Testing the network connection to DC1

1. Insert image B42024\_06\_18 .png

As you saw, when promoting DC1 to be your first DC, you need to add the ADDSDeployment feature to DC2 before you can promote the DC. The output from step 4, looks like this|:

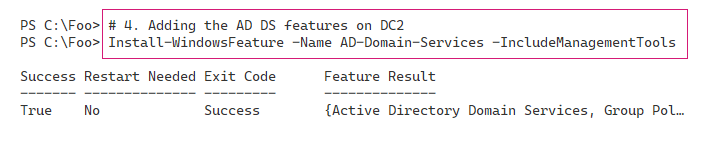


Figure 6.19: Adding the AD DS features on DC2

1. Insert image B42024\_06\_19 .png
2. With all the prerequisites in place, in step 5, you promote DC2 to be a DC and ensure that the DC does not reboot after this step completes. There is no output from this step.
3. In step 6, before rebooting DC2 (necessary to complete the promotion process), you check to see the computer objects now in the AD. This step helps you to ensure that DC2 is now (after the reboot) in the correct place in AD. The output of this step is like this:

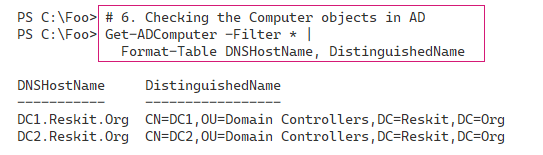


Figure 6.20: Checking the computer objects in the AD

1. Insert image B42024\_06\_20 .png
2. In step 7, you finalize the promoting process and reboot DC2. Once rebooted, you need to logon as the domain administrator (Reskit\Administrator). This step produces no output as such.

With step 8, you examine the hosts in the domain controllers OU, which looks like this:

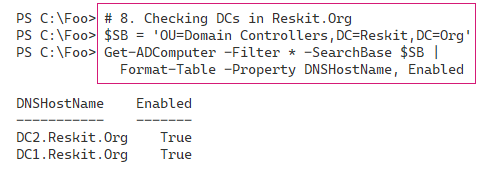


Figure 6.21: Checking DCs in Reskit.Org

1. Insert image B42024\_06\_21 .png

In the final step in this recipe, step 9, you use the Get-ADDomain cmdlet to check that DC1 and DC2 are DCs for this domain. The output of this step looks like this:

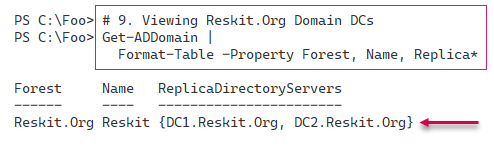


Figure 6.22: Viewing Reskit.Org domain DCs

1. Insert image B42024\_06\_22 .png

## There’s more...

In step 1, you import the ServerManager module manually. As you have seen in earlier recipes, this module is not supported natively with PowerShell 7. This step loads the module using the Windows PowerShell compatibility solution described earlier in the book.

In step 3, you check to ensure DC2 can connect to DC1 over ports 445 and 389. Windows uses TCP port 445 for SMB file sharing. The Group Policy Agent on each domain-joined host uses this port to retrieve Group Policy details from the SYSVOL share on DC1. LDAP uses port 389 to perform actions against a DC, such as adding a new user or checking group membership. Both ports need to be open and contactable if the promotion of DC2 is to be successful.

In step 6, you retrieve all computer accounts currently in your AD. By default, the AD DC promotion process ensures that this host’s computer account is now in the Domain Controllers OU. The location of a DC in an OU is essential as it enables your new DC to apply the group policy settings on this OU. If DC2 was a workgroup computer and not joined to the domain, the promotion process would create this account in the appropriate OU. If DC2 was a domain member, then the promotion process would move the computer account into the OU.

In this recipe, before promoting DC2 to be a DC, you check to ensure that the prerequisites are in place before the promotion. Then you ensure that things are correct after. In most cases, this is redundant checking, since most AD promotions work fine. Nevertheless, the additional checks ensure you discover and resolve any issues that might affect the promotion or cause the AD to not work correctly after a not-fully successful promotion.

# Installing a child domain

As you saw in “Installing a replica domain controller”, adding a DC to an existing domain is straightforward. So long as prerequisites like DNS are in place, the promotion process is simple.

An AD forest can contain more than one domain. This architecture has some value in terms of delegated control and some reduction in replication traffic. And like creating a replica DC, creating a child domain is simple, as you can see in this recipe.

Best practice calls for a contiguous namespace of domains, where the additional domain is a child of another existing domain. In this recipe, you create a child domain to the Reskit.Org domain you created earlier. You promote the server UKDC1 to be the first DC in a new child domain UK.Reskit.Org. The steps in this recipe are very similar to those in “Installing a replica directory controller“.

## Getting Ready

You run this recipe on UKDC1, a domain-joined server in the Reskit.Org domain on which you have installed PowerShell 7 and VS Code.

## How to do it...

1. Import the ServerManager module:

Import-Module -Name ServerManager -WarningAction SilentlyContinue

1. Check DC1 can be resolved:

Resolve-DnsName -Name DC1.Reskit.Org -Type A

1. Check network connection to DC1:

Test-NetConnection -ComputerName DC1.Reskit.Org -Port 445

Test-NetConnection -ComputerName DC1.Reskit.Org -Port 389

1. Add the AD DS features on UKDC1:

$Features = ’AD-Domain-Services’

Install-WindowsFeature -Name $Features -IncludeManagementTools

1. Create a credential and installation hash table:

Import-Module -Name ADDSDeployment -WarningAction SilentlyContinue

$URK    = ”Administrator@Reskit.Org”

$PW     = ’Pa$$w0rd’

$PSS    = ConvertTo-SecureString -String $PW -AsPlainText -Force

$CredRK = [PSCredential]::New($URK,$PSS)

$INSTALLHT    = @{

  NewDomainName                 = ’UK’

  ParentDomainName              = ’Reskit.Org’

  DomainType                    = ’ChildDomain’

  SafeModeAdministratorPassword = $PSS

  ReplicationSourceDC           = ’DC1.Reskit.Org’

  Credential                    = $CredRK

  SiteName                      = ’Default-First-Site-Name’

  InstallDNS                    = $false

  Force                         = $true

}

1. Install child domain:

Install-ADDSDomain @INSTALLHT

1. Look at the AD forest:

Get-ADForest -Server UKDC1.UK.Reskit.Org

1. Look at the UK domain:

Get-ADDomain -Server UKDC1.UK.Reskit.Org

## How it works...

In step 1, you load the Server Manager module. This module is not natively supported by PowerShell 7. This step, which produces no output, loads the module using the Windows PowerShell compatibility solution.

When you create a new domain and new domain controller (using UKDC1, for example), the server needs to contact the holder of the Domain Naming FSMO role, the DC responsible for the forest’s changes. In step 2, you check to ensure you can reach host, DC1, which looks like this:

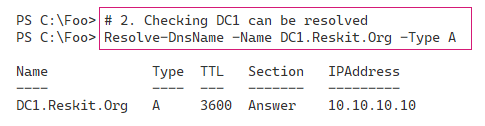


Figure 6.23: Checking DC1 can be resolved

1. Insert image B42024\_06\_23 .png

In step 3, you check that you can connect to DC1 on ports 445 and 389, both required for proper domain functioning. The output of this step looks like this:

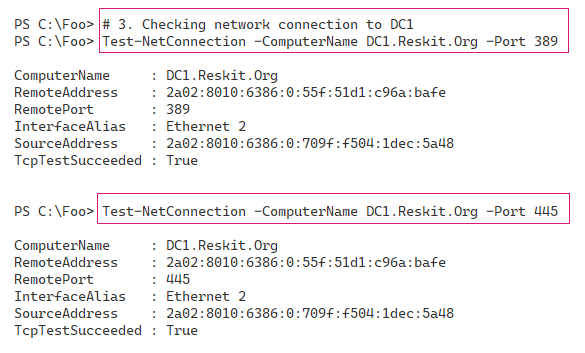


Figure 6.34: Checking network connection to DC1

1. Insert image B42024\_06\_24 .png

In step 4, you add the AD Domain Services feature to UKDC1. This feature is required to install the necessary tools you use to create the child domain. The output of this step looks like this:

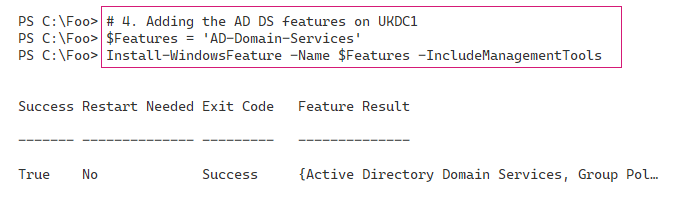


Figure 6.35: Adding the AD DS features on UKDC1

1. Insert image B42024\_06\_25 .png

In step 5, you build a hash table of parameters that you use in a later step to perform the promotion. In step 6, you use this hash table when calling Install-ADDSDomain to create a child domain. These two steps create no output. After the promotion has completed, your host reboots, log on as Reskit\Administrator.

Once you have logged in, you can check on the AD forest details from your newly promoted child domain DC. In step 7, you use Get-ADForest, with output like this:

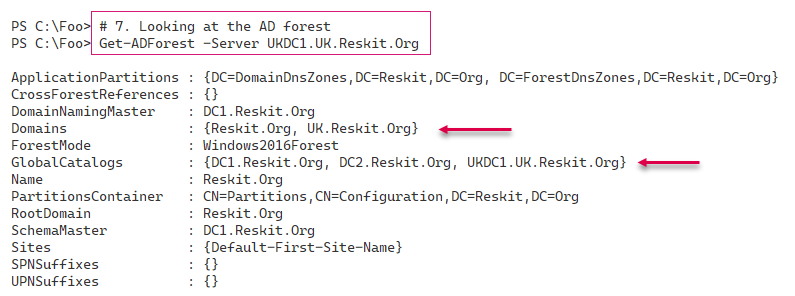


Figure 6.36: Looking at the AD forest details

1. Insert image B42024\_06\_26 .png

In step 8, you examine the domain details of the newly created child domain. By using Get‑ADDomain and targeting the new DC, you see output that looks like this:

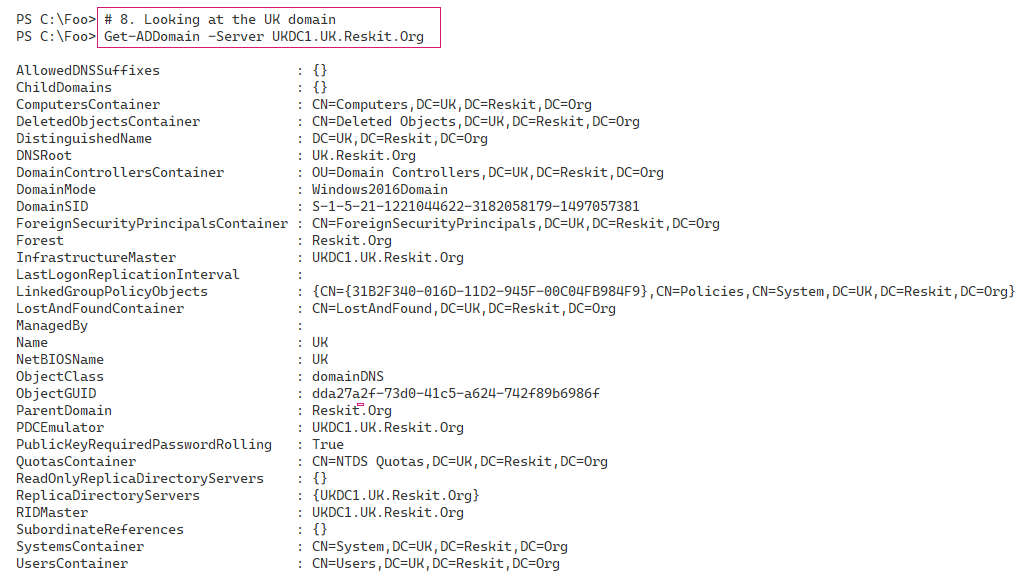


Figure 6.27: Looking at the UK domain details

1. Insert image B42024\_06\_27.png

## There’s more...

In step 2, you check that DC1, the domain naming FSMO role holder. Assuming you have installed the Reskit.Org domain and forest as shown in this chapter, that role holder is DC1. You could call Get‑ADForest and obtain the hostname from the DomainNamingMaster property.

In step 5, you promote UKDC1 to be the first DC in a new child domain, UK.Reskit.Org. Unlike in the two previous DC promotions (that is, promoting DC1 and DC2), in this step you allow Windows to reboot immediately after the promotion has complete. This step can take quite a while - potentially 10 minutes or more, so be patient.

In step 7, you use Get-ADForest to examine details of the forest as stored on UKDC1. As you can see in the figure, these details now show your new domain (UK.Reskit.Org) in the Domains property. Also, by default, you can see that UKDC1.UK.Reskit.Org is also a Global Catalog server.

# Creating and managing AD users and groups

After you have created your forest/domain and your domain controllers, you can begin to manage the core objects in AD, namely, users, groups, computers, and organizational units (OUs). User and computer accounts identify a specific user or computer. Windows uses these objects to enable the computer and the user to log on securely using passwords held in the AD.

AD Groups enable you to collect users and computers into a single (group) account that simplifies setting access controls on resources such as files or file shares. As you saw in “Testing an AD installation”, when you create a new forest, the AD promotion process creates many potentially useful groups.

Organizational Units (OUs) enable you to partition users, computers, and groups into separate container OUs. OUs provide you with essential roles in your AD. The first is role delegation. You can delegate the management of any OU (and child OUs) to be carried out by different groups. For example, you could create a top-level OU called UK in the Reskit.Org domain. You could then delegate permissions to the objects in this OU to a group, such as UKAdmins, enabling any member of that group to manage AD objects in, and below, the UK OU. You can delegate users in another group, such as the North America Admins group.

The second role played by OUs is to act as a target for group policy objects. You could create a group policy object for the IT team and apply it to the IT OU. You could create a separate OU and create GPOs that apply to only the computer and user objects in that OU. Thus, each user and computer in a given OU are configured based on the GPO.

In this recipe, you examine AD user and group objects. In a later recipe, “Managing AD computers”, you explore managing AD computers. And in “Creating Group Policy Objects”, you assign a Group Policy to an OU you create in this recipe.

## Getting Ready

You run this recipe on DC1, a domain controller in the Reskit.Org domain. You have previously workgroup server on which you have installed PowerShell 7 and VS Code on this host.

## How to do it...

1. Create a hash table for general user attributes:

$PW  = ’Pa$$w0rd’

$PSS = ConvertTo-SecureString -String $PW -AsPlainText -Force

$NewUserHT = @{}

$NewUserHT.AccountPassword       = $PSS

$NewUserHT.Enabled               = $true

$NewUserHT.PasswordNeverExpires  = $true

$NewUserHT.ChangePasswordAtLogon = $false

1. Create two new users:

# First user

$NewUserHT.SamAccountName    = ’ThomasL’

$NewUserHT.UserPrincipalName = ’thomasL@reskit.org’

$NewUserHT.Name              = ’ThomasL’

$NewUserHT.DisplayName       = ’Thomas Lee (IT)’

New-ADUser @NewUserHT

# Second user

$NewUserHT.SamAccountName    = ’RLT’

$NewUserHT.UserPrincipalName = ’rlt@reskit.org’

$NewUserHT.Name              = ’Rebecca Tanner’

$NewUserHT.DisplayName       = ’Rebecca Tanner (IT)’

New-ADUser @NewUserHT

1. Create an OU for IT:

$OUHT = @{

    Name        = ’IT’

    DisplayName = ’Reskit IT Team’

    Path        = ’DC=Reskit,DC=Org’

}

New-ADOrganizationalUnit @OUHT

1. Move users into the OU:

$MHT1 = @{

    Identity   = ’CN=ThomasL,CN=Users,DC=Reskit,DC=ORG’

    TargetPath = ’OU=IT,DC=Reskit,DC=Org’

}

Move-ADObject @MHT1

$MHT2 = @{

    Identity = ’CN=Rebecca Tanner, CN=Users, DC=Reskit, DC=ORG’

    TargetPath = ’OU=IT,DC=Reskit,DC=Org’

}

Move-ADObject @MHT2

1. Create a third user directly in the IT OU:

$NewUserHT.SamAccountName    = ’JerryG’

$NewUserHT.UserPrincipalName = ’jerryg@reskit.org’

$NewUserHT.Description       = ’Virtualization Team’

$NewUserHT.Name              = ’Jerry Garcia’

$NewUserHT.DisplayName       = ’Jerry Garcia (IT)’

$NewUserHT.Path              = ’OU=IT,DC=Reskit,DC=Org’

New-ADUser @NewUserHT

1. Add two users who get removed later:

# First user to be removed

$NewUserHT.SamAccountName    = ’TBR1’

$NewUserHT.UserPrincipalName = ’tbr@reskit.org’

$NewUserHT.Name              = ’TBR1’

$NewUserHT.DisplayName       = ’User to be removed’

$NewUserHT.Path              = ’OU=IT,DC=Reskit,DC=Org’

New-ADUser @NewUserHT

# Second user to be removed

$NewUserHT.SamAccountName     = ’TBR2’

$NewUserHT.UserPrincipalName  = ’tbr2@reskit.org’

$NewUserHT.Name               = ’TBR2’

New-ADUser @NewUserHT

1. View existing AD users:

Get-ADUser -Filter \*  -Property \*|

  Format-Table -Property Name, Displayname, SamAccountName

1. Removea user via a Get | Remove pattern:

Get-ADUser -Identity ’CN=TBR1,OU=IT,DC=Reskit,DC=Org’ |

    Remove-ADUser -Confirm:$false

1. Remove a user directly:

$RUHT = @{

  Identity = ’CN=TBR2,OU=IT,DC=Reskit,DC=Org’

  Confirm  = $false}

Remove-ADUser @RUHT

1. Update a user object:

$TLHT =@{

  Identity     = ’ThomasL’

  OfficePhone  = ’4416835420’

  Office       = ’Cookham HQ’

  EmailAddress = ’ThomasL@Reskit.Org’

  GivenName    = ’Thomas’

  Surname      = ’Lee’

  HomePage     = ’Https://tfl09.blogspot.com’

}

Set-ADUser @TLHT

1. View updated user:

Get-ADUser -Identity ThomasL -Properties \* |

  Format-Table -Property DisplayName,Name,Office,

                         OfficePhone,EmailAddress

1. Create a new domain local group:

$NGHT = @{

 Name        = ’IT Team’

 Path        = ’OU=IT,DC=Reskit,DC=org’

 Description = ’All members of the IT Team’

 GroupScope  = ’DomainLocal’

}

New-ADGroup @NGHT

1. Add all the users in the IT OU into the IT Team group:

$SB = ’OU=IT,DC=Reskit,DC=Org’

$ItUsers = Get-ADUser -Filter \* -SearchBase $SB

Add-ADGroupMember -Identity ’IT Team’  -Members $ItUsers

1. Display members of the IT Team group:

Get-ADGroupMember -Identity ’IT Team’ |

  Format-Table SamAccountName, DistinguishedName

## How it works...

You use the New-ADUser cmdlet to create a new AD user. Due to the amount of information you wish to hold for any user you create, the number of parameters you might need to pass to New-ADUser can make for very long code lines. To get around that, you create a hash table of parameters and parameter values and then use it to create the user. In step 1, you create such a hash table, which creates no output.

In step 2, you add to the hash table you created in the previous step and create two new AD users. This step creates no output.

With step 3, you create a new Organizational Unit, IT, which creates no output. You use this OU to collect user and computer objects for the IT Department of Reskit.Org. Next, in step 4, which also creates no output, you move the two users you created previously into the IT OU.

Rather than creating a user (which by default places the new user object in the Users container in your AD) and later moving it to an OU, you can create a new user object directly in an OU. In step 5, you create a third new user directly in the IT OU, which creates no output.

In step 6, which generates no output, you create two additional users which you use later in the recipe. With step 7, you use Get-ADUser to retrieve all the user objects in the Reskit.Org domain, which looks like this:

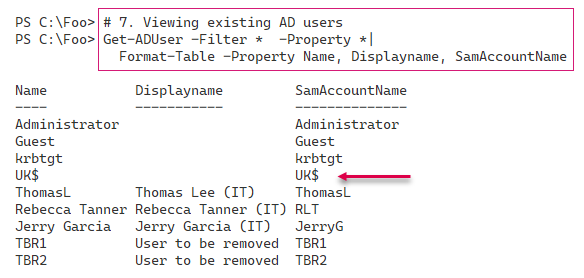


Figure 6.28: Viewing existing AD users

1. Insert image B42024\_06\_28.png

You can remove any AD user by invoking Remove-ADUser. There are two broad patterns for removing a user. The first pattern involves using Get-ADUser to get the user you want to remove and then piping that to Remove-ADUser. This pattern is appropriate for the command line where you should always verify you have the right user before deleting that user. The second, possibly more appropriate in scripts, is to remove the user in one operation (and dealing with any risks of removing the wrong user). In step 8, you use the Get | Remove pattern to get the user then remove it. In step 9, you remove the user directly by specifying the user’s identity when calling Remove-ADUser. Neither step 8 nor step 9 generate output.

In step 10, you update a user object with new/updated properties. This step generates no output. To see the changes, in step 11, you can use Get-ADUser and view the properties of the updated user, which looks like this:

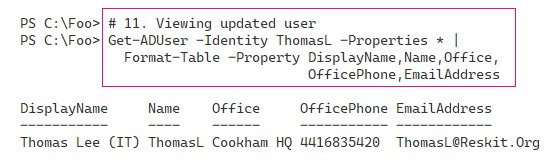


Figure 6.29: Viewing updated user

1. Insert image B42024\_06\_29.png

AD enables two types of group account - both of which can contain users and groups: Security and Distribution. You use distribution groups typically for email systems. Exchange, for example, uses them to implement distribution lists. You use security groups to assign permissions and rights. For example, if you allow a group to have access to a file or folder, then by default, all members of that group have that access permission. Using groups is a best practice when setting permissions and rights.

Security groups in AD have three scopes, which offer different features in terms of members and how you use them to assign permissions and rights. For more details on the different scopes, and AD security groups in general, see https://docs.microsoft.com/en-us/windows/security/identity-protection/access-control/active-directory-security-groups.

In step 12, you create a new domain local group, IT Team. In step 13, you add all the users in the IT OU to this IT Team group. Neither step produces output. In step 14, you get and display the members of the IT Team group, which looks like this:

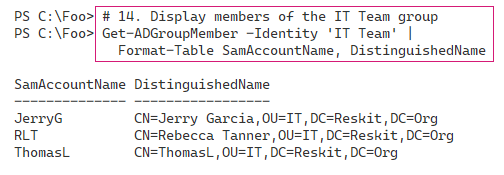


Figure 6.30: Displaying members of the IT Team group

1. Insert image B42024\_06\_30.png

## There’s more...

In step 7, you see the users in the Reskit.Org domain. Note the user UK$. This user relates to the child domain (UK.Reskit.Org). It is not, as such, an actual user. The $ character at the end of the username indicates it’s a hidden user but fundamental to supporting the child domain. Don’t be tempted to tidy up and remove this user - that would break the child domain structure.

As you see in this recipe, the cmdlets you use to add or modify a user, group, or OU create no output, which cuts down on the output you have to wade through. Some cmdlets and cmdlet sets would output details of the objects created, updated, or possibly deleted. Consider the lack of output for AD cmdlets as a feature.

# Managing Active Directory computers

AD computer objects represent domain-joined computers which can use the domain to authenticate user login. Before you can log in as a domain user, such as Reskit\JerryG, your computer must be a domain member. When a domain-joined computer starts up, it contacts a domain controller to authenticate itself. In effect, the computer logs into the domain and creates a secure channel to the DC. Once Windows establishes this secure channel, you can log a user. Under the covers, Windows uses the secure channel to negotiate the user logon.

In this recipe you

## Getting Ready

You run this recipe on DC1, a domain controller in the Reskit.Org domain. This recipe also uses SRV1 (a non-domain joined host you used in earlier chapters in this book) and UKDC1 (the DC in the UK.Reskit.Org domain). You should have PowerShell 7 and VS Code installed on each of these hosts.

## How to do it...

1. Get computers in the Reskit domain:

Get-ADComputer -Filter \* |

  Format-Table -Property Name, DistinguishedName

1. Gett computers in the UK domain:

Get-ADComputer -Filter \* -Server UKDC1 |

  Format-Table -Property Name, DistinguishedName

1. Create a new computer in the Reskit.Org domain:

$NCHT = @{

    Name                   = ’Wolf’

    DNSHostName            = ’Wolf.Reskit.Org’

    Description            = ’One for Jerry’

    Path                   = ’OU=IT,DC=Reskit,DC=Org’

}

New-ADComputer @NCHT

1. Create a credential object for SRV1:

$ASRV1    = ’SRV1\Administrator’

$PSRV1    = ’Pa$$w0rd’

$PSSRV1   = ConvertTo-SecureString -String $PSRV1 -AsPlainText -Force

$CredSRV1 = [pscredential]::New($ASRV1, $PSSRV1)

1. Create a script block to join SRV1:

$SB = {

  $ARK    = ’Reskit\Administrator’

  $PRK    = ’Pa$$w0rd’

  $PSRK   = ConvertTo-SecureString -String $PRK -AsPlainText -Force

  $CredRK = [pscredential]::New($ARK, $PSRK)

  $DJHT = @{

    DomainName  = ’Reskit.Org’

    OUPath      = ’OU=IT,DC=Reskit,DC=Org’

    Credential  = $CredRK

    Restart     = $false

 }

    Add-Computer @DJHT

}

1. Join the computer to the domain:

Invoke-Command -ComputerName SRV1 -Credential $CredSRV1 -ScriptBlock $SB

1. Restart SRV1:

Restart-Computer -ComputerName SRV1 -Credential $CredSRV1 -Force

1. View the resulting computer accounts for Reskit.Org:

Get-ADComputer -Filter \* -Properties DNSHostName,LastLogonDate |

  Format-Table -Property Name, DNSHostName, Enabled

## How it works...

In step 1, you use the Get-ADComputer cmdlet to obtain the computer objects defined, thus far, in the AD, which looks like this:

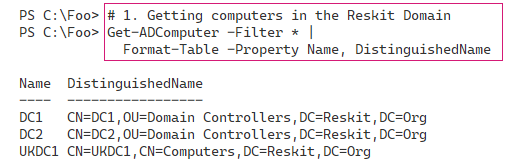


Figure 6.31: Getting computers in the Reskit domain

1. Insert image B42024\_06\_31.png

In step 2, you obtain the computers in the UK.Reskit.Org domain from that domain’s DC, UKDC1, which looks like this:

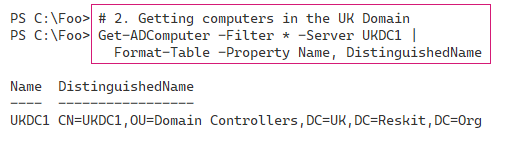


Figure 6.32: Getting computers in the UK domain

1. Insert image B42024\_06\_32.png

In step 3, you create a new computer in the Reskit domain, Wolf, or Wolf.Reskit.Org. This step creates no output.

To prepare to add SRV1 to the domain in a single operation, in step 4, you create a credential object for SRV1’s administrator user. You need this because SRV1 is a workgroup computer. In step 5, you create a script block that adds a computer to the Reskit.Org domain. Neither step produces any output.

In step 6, you invoke the script block on SRV1, adding the computer to the domain. The output of this step looks like this:

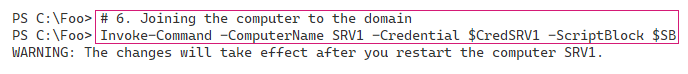


Figure 6.33: Joining the computer to the domain

1. Insert image B42024\_06\_33.png

To complete the process of joining SRV1 to the Reskit.Org domain, in step 7, you reboot SRV1 which creates no output.

After SRV1 has completed its reboot, in step 8, you view all the accounts now in the Reskit.Org domain, which looks like this:

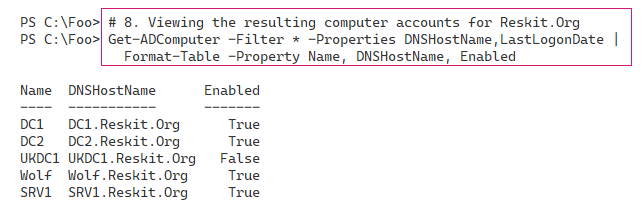


Figure 6.34: Viewing resulting computer accounts for Reskit.Org

1. Insert image B42024\_06\_34.png

## There’s more...

There are two broad ways of adding a computer to a domain. The first is to log on to the computer to be added and join the domain. To achieve this, you must have credentials for a user with permissions needed to add a computer to a domain (i.e. the domain administrator). You also need the credentials that enable you to log on to the system itself. Alternatively, you can create a computer object in advance, which is known as pre-staging. You need administrator credentials for this operation, but once pre-staged any user can join the computer to the domain.

In step 3, you pre-stage the Wolf computer. A user able to log on to Wolf could then use Add-Computer (or the GUI) to add the host to the domain. In step 4, you add a computer to a domain using domain administrator credentials.

# Adding/Removing users using CSV files

Spiceworks (https://www.spiceworks.com/) is an excellent site for IT Professionals to learn more and get their problems solved. Spiceworks has a busy PowerShell support forum which you can access at https://community.spiceworks.com/programming/powershell.

A frequently asked (and answered) question is: How do I add multiple users using an input file? This recipe does just that. You start with a CSV file containing details of the users you are going to add. Then you run this recipe to add the users.

This recipe uses a CSV file of users to add to AD, with a limited set of properties and values. In production, you would most likely extend the information contained in the CSV, based on your business needs and the information you want to store in AD.

## Getting Ready

You run this recipe on SRV1, a workgroup server on which you have installed PowerShell 7 and VS Code.

## How to do it...

1. Create a CSV file:

$CSVDATA = @’

Firstname, Initials, Lastname, UserPrincipalName, Alias, Description, Password

J, K,Smith, JKS, James, Data Team, Christmas42

Clair, B, Smith, CBS, Claire, Receptionist, Christmas42

Billy, Bob, JoeBob, BBJB, BillyBob, A Bob, Christmas42

Malcolm, Dudley, Duewrong, Malcolm, Malcolm, Mr Danger, Christmas42

‘@

$CSVDATA | Out-File -FilePath C:\Foo\Users.Csv

1. Import and display the CSV:

$Users = Import-CSV -Path C:\Foo\Users.Csv |

  Sort-Object  -Property Alias

$Users | Format-Table

1. Add the users using the CSV:

$Users |

  ForEach-Object -Parallel {

    $User = $\_

    #  Create a hash table of properties to set on created user

    $Prop = @{}

    #  Fill in values

    $Prop.GivenName         = $User.Firstname

    $Prop.Initials          = $User.Initials

    $Prop.Surname           = $User.Lastname

    $Prop.UserPrincipalName = $User.UserPrincipalName + ”@Reskit.Org”

    $Prop.Displayname       = $User.FirstName.Trim() + ” ” +

                              $User.LastName.Trim()

    $Prop.Description       = $User.Description

    $Prop.Name              = $User.Alias

    $PW = ConvertTo-SecureString -AsPlainText $User.Password -Force

    $Prop.AccountPassword   = $PW

    $Prop.ChangePasswordAtLogon = $true

    $Prop.Path                  = ’OU=IT,DC=Reskit,DC=ORG’

    $Prop.Enabled               = $true

    #  Now Create the User

    New-ADUser @Prop

    # Finally, Display User Created

    ”Created $($Prop.Name)”

}

1. Show all users in AD (Reskit.Org):

Get-ADUser -Filter \* |

  Format-Table -Property Name, UserPrincipalName

## How it works...

In step 1, which produces no output, you create a simple CSV file which you save to C:\Foo\Users.CSV.

In step 2, you import this newly created CSV file and display the information it contains, which looks like this:

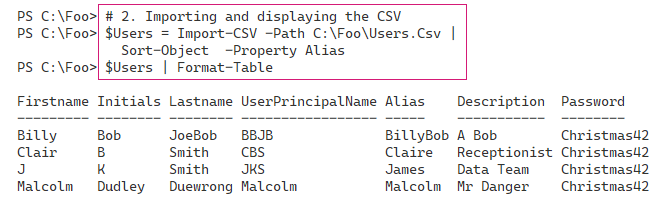


Figure 6.35: Importing and displaying the CSV file

1. Insert image B42024\_06\_35.png

In step 3, you add each user contained in the CSV into AD. You add the users, using New-ADUser which itself produces no output. This step adds some output to show what users you added, which looks like this:

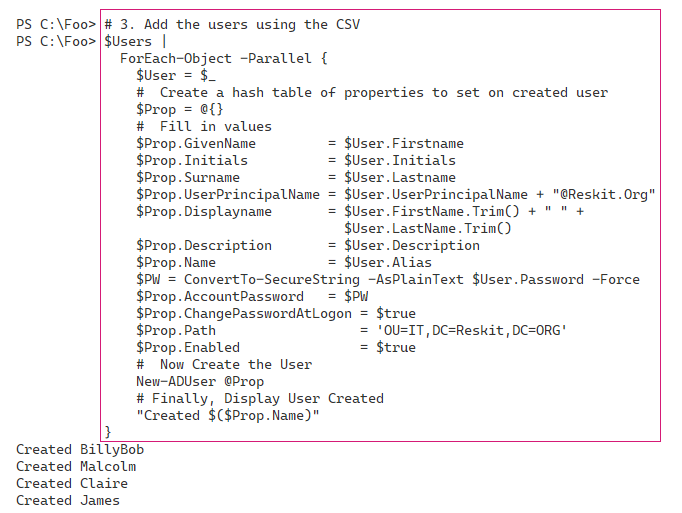


Figure 6.36: Adding users into AD using the CSV file

1. Insert image B42024\_06\_36.png

In the final step in this recipe, step 4, you use Get-ADUser to view all the users in the Reskit.Org domain. This step’s output looks like this:

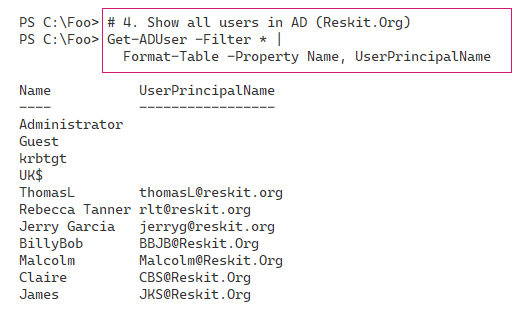


Figure 6.37: Viewing all users in the Reskit.Org domain

1. Insert image B42024\_06\_37.png

## There’s more...

In step 3, you add users based on the CSV. You add these users explicitly to the IT OU, using the parameter -Path (as specified in the $Prop hash table). In production, when adding users that could reside in different OUs, you should extend the CSV to include the distinguished name of the OU into which you wish to add each user.

# Creating Group Policy Objects

Group policy allows you to define computer and user configuration settings that ensure a system remains configured per policy. Each time a domain-joined computer starts up and each time a domain user logs on, the local group policy agent on your computer obtains the group policy settings from AD and ensures they are applied.

In this recipe, you begin by first creating a group policy object within the Active Directory. You then configure the GPO, for example, enabling computers in the IT organizational unit to use PowerShell scripts on those systems or set a specific screen saver. There are thousands of settings you can configure for a user or computer through group policy. Microsoft has created a spreadsheet that lists the policy settings which you can download from https://www.microsoft.com/en-us/download/101451. At the time of writing, the spreadsheet covers the Group Policy template files delivered with Windows 10 May 2020 Update (aka Windows 10 2004).

Once you configure your GPO object, you link the policy object to the OU you want to configure. You can also apply a GPO to the domain as a whole, to a specific AD site, or to an OU. You can also assign any GPO to multiple OUs which can simplify your OU design.

The configuration of a GPO typically results in Windows generating information that a host’s group policy agent (the code that applies the GPO objects) can access. This information tells the agent how to work. Settings made through administrative templates use registry settings inside Registry.POL files. The group policy agent obtains the policy details from the SYSVOL share on a domain controller and applies them whenever a user logs on or off or when a computer starts up or shuts down. The group policy module also provides the ability to create nice-looking reports describing the  
group policy object.

## Getting Ready

You run this recipe on DC1, a domain controller in the Reskit.Org domain. You created this DC in “Installing an Active Directory forest root domain” and after creating the IT OU.

## How to do it...

1. Create a Group Policy object:

$Pol = New-GPO -Name ITPolicy -Comment ”IT GPO” -Domain Reskit.Org

1. Ensure just computer settings are enabled:

$Pol.GpoStatus = ’UserSettingsDisabled’

1. Configure the policy with two registry-based settings:

$EPHT1= @{

  Name   = ’ITPolicy’

  Key    = ’HKLM\Software\Policies\Microsoft\Windows\PowerShell’

  ValueName = ’ExecutionPolicy’

  Value  = ’Unrestricted’

  Type   = ’String’

}

Set-GPRegistryValue @EPHT1 | Out-Null

$EPHT2= @{

  Name   = ’ITPolicy’

  Key    = ’HKLM\Software\Policies\Microsoft\Windows\PowerShell’

  ValueName = ’EnableScripts’

  Type   = ’DWord’

  Value  = 1

}

Set-GPRegistryValue @EPHT2 | Out-Null

1. Create a screen saver GPO:

$Pol2 = New-GPO -Name ’Screen Saver Time Out’

$Pol2.GpoStatus   = ’ComputerSettingsDisabled’

$Pol2.Description = ’15 minute timeout’

1. Set a Group Policy enforced registry value:

$EPHT3= @{

  Name   = ’Screen Saver Time Out’

  Key    = ’HKCU\Software\Policies\Microsoft\Windows\’+

              ’Control Panel\Desktop’

  ValueName = ’ScreenSaveTimeOut’

  Value  = 900

  Type   = ’DWord’

}

Set-GPRegistryValue @EPHT3 | Out-Null

1. Link both GPOs to the IT OU:

$GPLHT1 = @{

  Name     = ’ITPolicy’

  Target   = ’OU=IT,DC=Reskit,DC=org’

}

New-GPLink @GPLHT1 | Out-Null

$GPLHT2 = @{

  Name     = ’Screen Saver Time Out’

  Target   = ’OU=IT,DC=Reskit,DC=org’

}

New-GPLink @GPLHT2 | Out-Null

1. Display the GPOs in the domain:

Get-GPO -All -Domain Reskit.Org |

  Sort-Object -Property DisplayName |

    Format-Table -Property Displayname, Description, GpoStatus

1. Create and view a GPO Report:

$RPath = ’C:\Foo\GPOReport1.HTML’

Get-GPOReport -All -ReportType Html -Path $RPath

Invoke-Item -Path $RPath

1. Get the GPO report in XML format:

Get-GPOReport -All -ReportType XML -Path $RPath2

$XML = [xml] (Get-Content -Path $RPath2)

1. Create a simple GPO report:

$RPath2 = ’C:\Foo\GPOReport2.XML’

$FMTS = ”{0,-33}  {1,-30} {2,-10} {3}”

$FMTS -f ’Name’,’Linked To’, ’Enabled’, ’No Override’

$FMTS -f ’----’,’---------’, ’-------’, ’-----------’

$XML.report.GPO |

  Sort-Object -Property Name |

    ForEach-Object {

     $Gname = $\_.Name

     $SOM = $\_.linksto.SomPath

     $ENA = $\_.linksto.enabled

     $NOO = $\_.linksto.nooverride

     $FMTS -f $Gname, $SOM, $ENA, $NOO

   }

## How it works...

1. Note that, like many AD-related cmdlets, the cmdlets you use to manage Group Policy objects do not produce much output.
2. In step 1, you create a new GPO in the Reskit.Org domain. This step creates an empty GPO. This GPO is not yet linked to any OU thus does not get applied.
3. In step 2, you disable user settings, which allows the GPO client to ignore any user settings. Doing so can make the client GPO processing a bit faster.
4. In step 3, you set this GPO to have two specific registry-based values. When a computer starts up, the GPO processing on that client computer ensures that these two registry values are set on the client. During Group Policy refresh (which happens approximately every 2 hours) the value in the policy is enforced.
5. In step 4 and step 5, you create a new GPO and set a screen saver timeout of 900 seconds.
6. In step 6, you link the two GPOs to the IT Organizational unit. Until you link the GPOs to an OU (or to the domain or a domain site), GPO processing ignores the GPO.
7. In this recipe, step 1 through step 6 produce no output.

In step 7, you use Get-GPO to return information about all the GPOs in the domain, which looks like this:

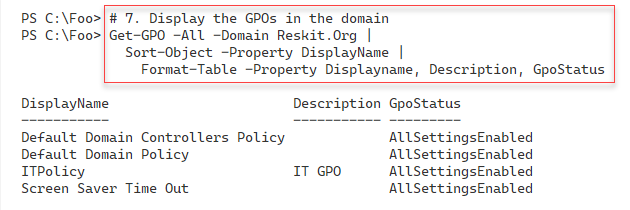


Figure 6.38: Displaying all the GPOs in the domain

1. Insert image B42024\_06\_38.png

In step 8, you generate and display a GPO report by using the Get-GPOReport command. The command produces no output, but by using Invoke-Command, you view the report in your default browser, which looks like this:

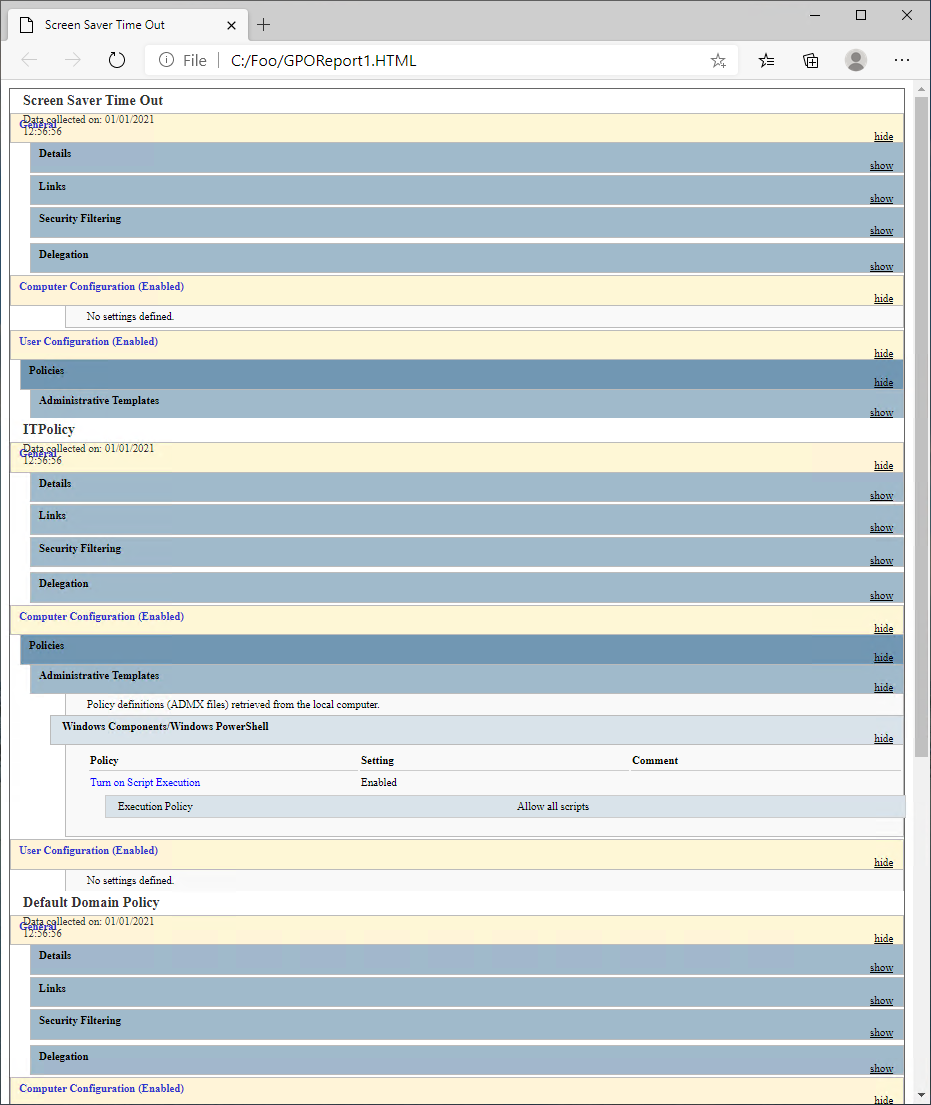


Figure 6.39: Viewing the GPO report in browser

1. Insert image B42024\_06\_39.png

In step 9 you use Get-GPOReport to return a report of all the GPOs in the domain in an XML format, which produces no output.

In step 10, you iterate through the returned XML and produce a simple report on GPOs and where they are linked, which looks like this:

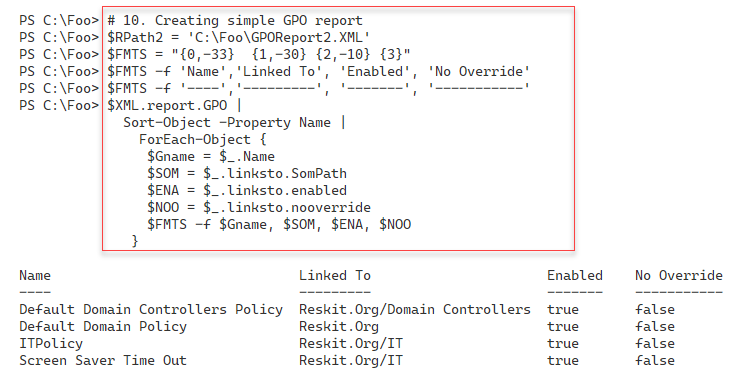


Figure 6.40: Creating a simple report on GPSs

1. Insert image B42024\_06\_40.png

## There’s more...

In step 8, you see the output from the Get-GPOReport cmdlet. At the time of writing, the Edge browser does not render the output as nicely as you might want. What may look like a bug in the graphic is just how Edge (again at the time of writing) renders this HTML document.

In step 9 and step 10, you create your mini-report. In step 9, you use the Get-GPOReport command to obtain a report of all GPOs in the domain returned as XML. In step 10, you report on the GPOs in the Reskit Domain using .NET composite string formatting with the -f operator.

Using .NET composite formatting enables you to use when the objects returned by a cmdlet are not in a form to be used directly with Format-Table. In step 9, for example, the XML returned contains details of the GPO links as a property which is actually an object with sub-properties. To create nice looking output, you create a format string you use to display each row of your report, including report header lines. You first use this format string to create the two report header lines for the report. Then, for each GPO in the returned list, you obtain the GPO name, which OU the GPO is linked to, whether it is enabled, and whether the GPO is non-overridable. Finally, you use the format string to output a single report line. In most cases, these custom reports are easier to read and contain only the information you deem useful.

The Format-Table and Format-List cmdlets are of most use when each object has simple properties. When an object has a property that is an object with properties, the format commands do not surface these (sub) properties. In this case, you must obtain the details manually of the GPO links for each GPO and generate your report lines based on those results. While the report layout works well for this specific set of GPOs, should you have GPOs with longer names, or linked to deeper organizational units, you may need to adjust the format string you set in Step 10.

# Managing AD replication

Active Directory uses a special database to support its operations. The database is a distributed, multi-master database with convergence—every DC in every domain stores this database in the file C:\Windows\NTDS\ntds.dit.

Every DC in any domain holds a complete copy of this database. If you add a new user or change a user’s office, that change occurs on just one DC (initially). AD replication makes the change in all database copies. In this way, the database remains consistent over time and across all DCs.

AD replication is based on partitions - a slice of the overall database. AD can replicate each partition separately. There are several partitions in AD:

* **Schema partition** - this holds the AD schema that defines each object that AD stores in the database. The schema also defines the properties of all these objects.
* **Configuration partition** - this holds the details of the structure of the domain.
* **Domain partition** - this partition, also known as the domain naming context, contains the objects relating to a domain (users, groups, OUs, etc.). The objects in this partition are defined based on the schema.
* **Application partition** - some applications, such as DNS, store objects in your AD, and rely on AD replication to replicate the values.

There are two types of replication: intra-site replication and inter-site replication. Intra-site replication happens between DCs in a given AD site, while inter-site replication occurs between different sites.

You can create different topologies for replication, including:

* **Ring** - each DC in a site has at least two inbound replication partners. When any change is made to any DC, that DC notifies its replication partners that it has a change. Those DCs can then replicate that change (if they have not seen the change before). AD by default ensures there are not more than three hops within the replication topology. If you have a large number of DCs (more than 7), AD automatically creates additional replication links to keep the hop count below 3.
* **Full mesh** - with this topology, all DCs replicate to all others. Full mesh replication keeps the database in sync with a minimum of replication delay, but can be more expensive in terms of bandwidth (and DC utilization). It is not scalable.
* **Hub and spoke** - You might use this approach in enormous organizations where a ‘spoke’ DC replicates with a central hub DC. The hub DC then replicates the change to all other spoke DCs in your organization. Hub and spoke can reduce replication for widely dispersed implementations.
* **Hybrid** - here you can combine any of these, based on business need.

By default, AD replication uses a ring topology. You can adopt different topologies if your business needs dictate, but this requires configuration.

In most smaller organizations (such as Reskit.Org), replication is set up and operates automagically. But for large and distributed organizations, replication can be quite complicated as you attempt to balance being totally up-to-date against the cost of geographical bandwidth. If you have DCs in several continents, you want to collect those changes and do then all at once, say every 4 hours. But that means the remote DC would have out of date information for a period. As a general rule of thumb, you should design replication so that it happens faster than a person can fly between your AD sites served by a DC. If you change your password, say in London, then as long as the changes occur within 12 hours, when you fly to say Brisbane, Australia, the Australian DCs contain the replicated password. Thus, you can log in with the new password immediately upon landing in Brisbane.

For more information on AD replication concepts, see https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/replication/active-directory-replication-concepts.

For any sizeable organization, design and planning of AD is vital - see https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/ad-ds-design-and-planning for more information on the planning and design work necessary.

Traditionally you use many Win32 console applications to manage and troubleshoot replication including repadmin.exe, which broadly replaces an earlier command replmon.exe. For some detail on the repadmin.exe command (and replication) see https://techcommunity.microsoft.com/t5/ask-the-directory-services-team/getting-over-replmon/ba-p/396687.

With the advent of PowerShell and the PowerShell active directory module, you can now perform many of the functions of repadmin.exe using PowerShell cmdlets. In this recipe, you examine some of the details of AD replication using PowerShell.

## Getting Ready

You run this recipe on DC1, a domain controller in the Reskit.Org domain. You should also have DC2 and UKDC1 available and online at the start of testing this recipe. You must have installed PowerShell 7 and VS Code on all these hosts.

## How to do it...

1. Check replication partners for DC1:

Get-ADReplicationPartnerMetadata -Target DC1.Reskit.Org   |

  Format-List -Property Server, PartnerType, Partner,

                        Partition, LastRep\*

1. Check AD replication partner metadata in the domain:

Get-ADReplicationPartnerMetadata -Target Reskit.Org -Scope Domain |

  Format-Table -Property Server, P\*Type,Last\*

1. Investigate group membership metadata:

$REPLHT = @{

  Object              = (Get-ADGroup -Identity ’IT Team’)

  Attribute           = ’Member’

  ShowAllLinkedValues = $true

  Server              = (Get-ADDomainController)

}

Get-ADReplicationAttributeMetadata @REPLHT |

  Format-Table -Property A\*NAME, A\*VALUE, \*TIME

1. Add two users to the group and remove one:

Add-ADGroupMember -Identity ”IT Team” -members Malcolm

Add-ADGroupMember -Identity ”IT Team” -members Claire

Remove-ADGroupMember -Identity ”IT Team” -members Claire -Confirm:$False

1. Check updated metadata:

Get-ADReplicationAttributeMetadata @REPLHT |

  Format-Table -Property A\*NAME, A\*VALUE, \*TIME

1. Create an initial replication failure report:

$DomainController = ’DC1’

$Report = [ordered] @{}

## Replication Partners ##

$ReplMeta =

    Get-ADReplicationPartnerMetadata -Target $DomainController

$Report.ReplicationPartners = $ReplMeta.Partner

$Report.LastReplication     = $ReplMeta.LastReplicationSuccess

## Replication Failures ##

$REPLF = Get-ADReplicationFailure -Target $DomainController

$Report.FailureCount  = $REPLF.FailureCount

$Report.FailureType   = $REPLF.FailureType

$Report.FirstFailure  = $REPLF.FirstFailureTime

$Report.LastFailure   = $REPLF.LastFailure

$Report

1. Simulate a connection issue:

Stop-Computer DC2  -Force

Start-Sleep -Seconds 30

1. Make a change to this AD:

Get-AdUser -identity BillyBob  |

  Set-AdUser -Office ”Cookham Office” -Server DC1

1. Using repadmin.exe to generate a status report

repadmin /replsummary

## How it works...

In step 1, you check to discover replication partners for DC1. With two DCs in the Reskit domain (DC1 and DC2), the output of this step looks like this:

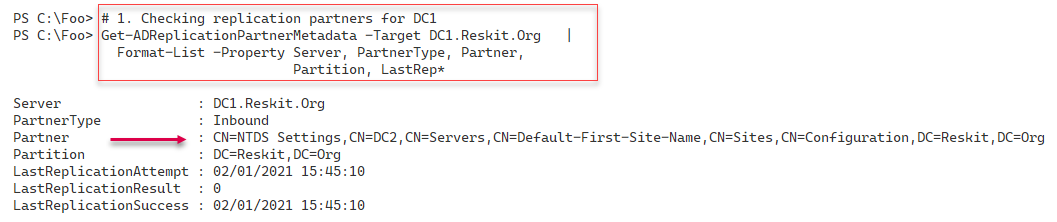


Figure 6.41:Checking replication partners for DC1

1. Insert image B42024\_06\_41.png

In step 2, you check on the replication partner metadata for the entire domain, which looks like this:

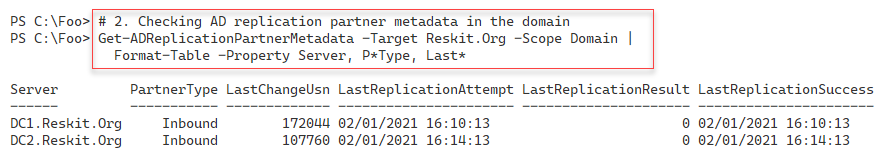


Figure 6.42: Checking replication partner metadata in the domain

1. Insert image B42024\_06\_42.png

In step 3, you examine the metadata for group membership. You take this step after creating the IT Team security group and populating it, at which point the output looks like this:

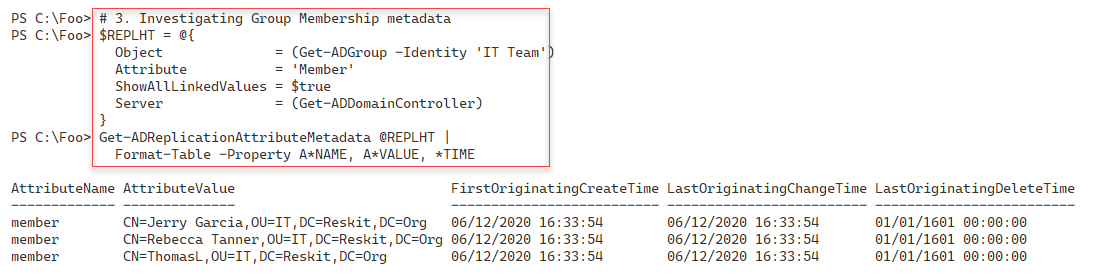


Figure 6.43: Examining group membership metadata

1. Insert image B42024\_06\_43.png

In step 4, which creates no output, you change this group by adding two members. Then, you remove one of them from the group. Under the covers, this step updates the group membership (three times), which generates replication traffic to replicate the group membership changes from DC1 to DC2.

With step 5, you re-examine the group membership metadata to see the effects of step 4, which looks like this:

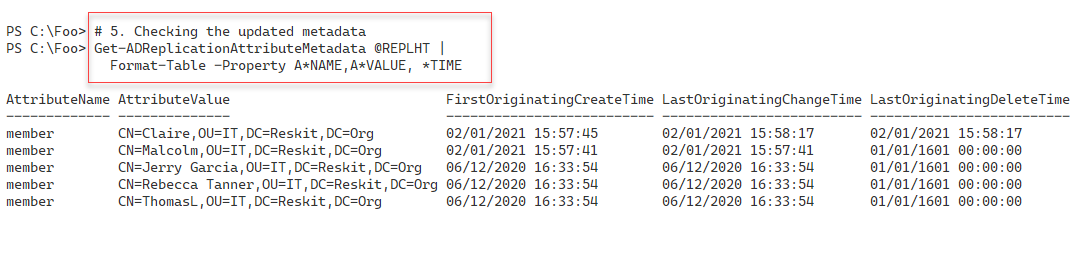


Figure 6.44: Checking the updated metadata

1. Insert image B42024\_06\_44.png

In step 6, you create a report of replication failures, with output like this:



Figure 6.45: Creating a report of replication failures

1. Insert image B42024\_06\_45.png

In step 7, you simulate a connection issue by stopping DC2 and waiting until the system has completed its shut down. In step 8, you make a change to a user (on DC1). These steps create no output, although step 7 results in AD attempting to replicate the changed user account to DC2.

In step 9, you use the repadmin.exe command to generate a replication summary report, which looks like this:

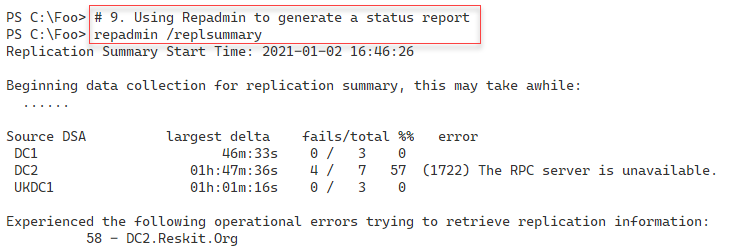


Figure 6.46: Generating a replication status report using repadmin

1. Insert image B42024\_06\_46.png

## There’s more...

In step 2, you discover the replication partners within the domain. In a domain with just two DCs, you would expect each DC to be an inbound replication partner to the other DC, and that is what the figure shows.

In step 3 through step 5, you examine the change in replication data before and after a group membership change. Using the Get-ADReplicationAttributeMetadata cmdlet is an excellent way to discover the specific changes to a security group’s membership, including removing a member. The information provided does not tell you who made the change or the system where the updated originated. But knowing that someone made a change, and when, are essential first steps.

In step 7, you create a potential replication issue by stopping DC2. In step 8, you generate a change to the AD on DC1, which would typically be replicated very quickly to DC2. But since DC2 is down, that replication cannot happen.

In step 9, you use the repadmin.exe console application to generate a replication summary showing the DC2 is not contactable from DC1. Sometimes, older tools, like repadmin.exe, produce better output than the cmdlets. For console usage and wanting a simple summary of replication, repadmin.exe is still a great tool.

# Reporting on AD computers

Monitoring the Active Directory is a necessary albeit time-consuming task. With larger numbers of users and computers to manage, you need all the help you can get, and PowerShell makes it easy to keep track of things.

A computer that has not logged on for an extended period could represent a security risk or could be a lost/stolen computer. It could also be a system that you have not rebooted after having appliedpatches and updates.

This recipe creates a report of computers that have not logged on, or that you have not rebooted for a while.

One challenge in developing scripts like this is creating meaningful test data. If you wish to generate a test report showing a system that has not logged in for over 6 months, you might have to wait for 6 months to get the necessary data. This recipe shows a way around that for testing purposes.

## Getting Ready

You run this recipe on DC1, a Domain Controller in the Reskit Domain on which you have installed PowerShell 7 and VS Code. Also, you should have completed earlier recipes that created some computer objects inside the AD.

## How to do it...

1. Create example computer accounts in the AD:

$NCHT1 = @{

    Name        = ’NLIComputer1\_1week’

    Description = ’Computer last logged in 1 week ago’

}

New-ADComputer @NCHT1

$NCHT2 = @{

  Name        = ’NLIComputer2\_1month’

  Description = ’Computer last logged in 1 week ago’

}

New-ADComputer @NCHT2

$NCHT3 = @{

  Name        = ’NLIComputer3\_6month’

  Description = ’Computer last logged in 1 week ago’

}

New-ADComputer @NCHT3

1. Create some constants for later comparison:

$OneWeekAgo   = (Get-Date).AddDays(-7)

$OneMonthAgo  = (Get-Date).AddMonths(-1)

$SixMonthsAgo = (Get-Date).AddMonths(-6)

1. Define a function to create sample data:

Function Get-RKComputers {

$ADComputers = Get-ADComputer -Filter \* -Properties LastLogonDate

$Computers = @()

foreach ($ADComputer in $ADComputers) {

  $Name = $ADComputer.Name

  # Real computers and last logon date

  if ($adComputer.name -NotMatch ”^NLI”) {

    $LLD = $ADComputer.LastLogonDate

  }

  Elseif ($ADComputer.Name -eq ”NLIComputer1\_1week”)  {

    $LLD = $OneWeekAgo.AddMinutes(-30)

  }

  Elseif ($ADComputer.Name -eq ”NLIComputer2\_1month”)  {

    $LLD = $OneMonthAgo.AddMinutes(-30)

  }

  Elseif ($ADComputer.Name -eq ”NLIComputer3\_6month”)  {

    $LLD = $SixMonthsAgo.AddMinutes(-30)

  }

  $Computers += [pscustomobject] @{

    Name = $Name

    LastLogonDate = $LLD

  }

}

$Computers

}

1. Build the report header:

$RKReport = ’’           # Start of report

$RKReport += ”\*\*\* Reskit.Org AD Daily AD Computer Report`n”

$RKReport += ”\*\*\* Generated [$(Get-Date)]`n”

$RKReport += ”\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*`n`n”

1. Get Computers in RK AD using Get-RKComputers:

$Computers = Get-RKComputers

1. Get computers that have never logged on:

$RKReport += ”Computers that have never logged on`n”

$RkReport += ”Name                    LastLogonDate`n”

$RkReport += ”----                    -------------`n”

$RKReport += Foreach($Computer in $Computers) {

  If ($null -eq $Computer.LastLogonDate) {

   ”{0,-22}  {1}  `n” -f $Computer.Name, ”Never”

  }

}

1. Report on computers that have not logged on in over 6 months:

$RKReport += ”`nComputers that havent logged in over 6 months`n”

$RkReport += ”Name                    LastLogonDate`n”

$RkReport += ”----                    -------------`n”

$RKReport +=

foreach($Computer in $Computers) {

  If (($Computer.LastLogonDate -lt $SixMonthsAgo) -and

      ($null -ne $Computer.LastLogonDate)) {

 (“`n{0,-23}  {1}  `n” -f $Computer.Name, $Computer.LastLogonDate).trim()

  }

}

1. Report on computer accounts that have not logged in for 1-6 months:

$RKReport += ”`n`nComputers that havent logged in 1-6 months`n”

$RkReport += ”Name                    LastLogonDate`n”

$RkReport += ”----                    -------------”

$RKReport +=

foreach($Computer in $Computers) {

  If (($Computer.LastLogonDate -ge $SixMonthsAgo) -and

     ($Computer.LastLogonDate -lt $OneMonthAgo) -and

       ($null -ne $Computer.LastLogonDate)) {

   ”`n{0,-22}  {1}  ” -f $Computer.Name, $Computer.LastLogonDate

  }

}

1. Report on computer accounts that have not logged in for the past 1 week to 1 month:

$RKReport += ”`n`nComputers that have between one week ”

$RKReport += ”and one month ago`n”

$RkReport += ”Name                    LastLogonDate`n”

$RkReport += ”----                    -------------”

$RKReport +=

foreach($Computer in $Computers) {

  If (($Computer.LastLogonDate -ge $OneMonthAgo) -and

     ($Computer.LastLogonDate -lt $OneWeekAgo) -and

       ($null -ne $Computer.LastLogonDate)) {

   ”`n{0,-22}  {1}  ” -f $Computer.Name, $Computer.LastLogonDate

  }

}

1. Display the report

$RKReport

## How it works...

1. In this recipe, all but the final step produce no output. Some of the steps exist to enable you to test the report that this recipe generates. Some of these steps might not be necessary for real life, as you already have enough real-life data to create a complete report.
2. In step 1, you create three AD computer accounts. You use these accounts to simulate computers that have not logged on for a while, thus enabling you to view a complete report. If you re-run this recipe or this step, adding these accounts produces errors since the accounts already exist. You could modify this step to check to see if the accounts exist before creating them.
3. In step 2, you create three time constants, representing the time 7 days ago, one month ago, and six months ago. This step enables you to test if a given user account has not longed on in that period.
4. In step 3, you create a new function, Get-RKComputers. This function returns a list of all the computer accounts in AD along with their last logon time.
5. In step 4, you begin the report by creating a report header.
6. In step 5, you call the Get-RKComputers and populate the $Computers array (all the computers available).

In step 6 through step 9, you add details to the report of computers who have never logged on, have not logged on for over 6 months, have not logged on in 1-6 months, and computers that have not logged on 1 week-1 month

In the final step, step 10, you display the report created by the earlier steps. This output of this step looks like this:

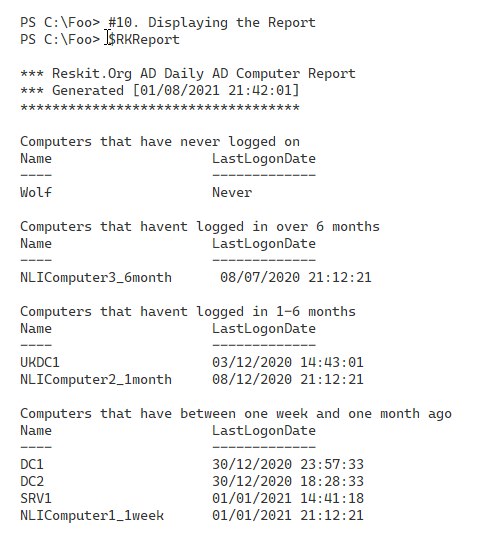


Figure 6.47: Displaying the report

1. Insert image B42024\_06\_47.png

## There’s more...

In step 3, you create a function to get the computer accounts in AD. This function returns an array of computer names and last logon date. In production, you might amend this function to return computer accounts from just certain OUs. You could also extend this function to test whether each computer is online by testing a network connection to the computer or checking to see if there is a DNS A record for the computer to detect stale computer accounts.

In step 5 through step 9, you create a report by adding text lines to the $RKReport variable. In doing so, you need to add CRLF characters before or after each line of text when you add each line to the report. Ensuring each line of the report begins in the right place can be challenging in creating reports using the technique shown by this recipe.

# Reporting on AD users

In the previous recipe, you created a report on computer accounts that may be of interest. User and group accounts are also worth tracking. If a user has not logged on for a reasonable period, the account could be a security risk. Likewise, a user with membership ina privileged account (for example, Enterprise Admins) could be used by an attacker. IT professionals know how easy it is just to put someone in a high privilege group than to set up more fine-grained permissions using something like Just Enough Administration (see “Implementing JEA”)

Regular reporting can help focus on accounts that could beusefully de-activated, removed from a security group, orremoved altogether.

In this recipe, you obtain all the accounts in the AD and examine potential security risks.

## Getting Ready

You run this recipe on DC1, a DC in the Reskit.Org domain, after running the recipes in this chapter. You should also have installed PowerShell 7 and VS Code on this host.

## How to do it...

1. Define a function Get-ReskitUser to return objects related to users in Reskit.Org domain:

Function Get-ReskitUser {

# Get PDC Emulator DC

$PrimaryDC = Get-ADDomainController -Discover -Service PrimaryDC

# Get Users

$ADUsers = Get-ADUser -Filter \* -Properties \* -Server $PrimaryDC

# Iterate through them and create $Userinfo hash table:

Foreach ($ADUser in $ADUsers) {

    # Create a userinfo HT

    $UserInfo = [Ordered] @{}

    $UserInfo.SamAccountname = $ADUser.SamAccountName

    $Userinfo.DisplayName    = $ADUser.DisplayName

    $UserInfo.Office         = $ADUser.Office

    $Userinfo.Enabled        = $ADUser.Enabled

    $userinfo.LastLogonDate  = $ADUser.LastLogonDate

    $UserInfo.ProfilePath    = $ADUser.ProfilePath

    $Userinfo.ScriptPath     = $ADUser.ScriptPath

    $UserInfo.BadPWDCount    = $ADUser.badPwdCount

    New-Object -TypeName PSObject -Property $UserInfo

    }

} # end of function

1. Get the users:

$RKUsers = Get-ReskitUser

1. Build the report header:

$RKReport = ’’  # first line of the report

$RkReport += ”\*\*\* Reskit.Org AD Report`n”

$RKReport += ”\*\*\* Generated [$(Get-Date)]`n”

$RKReport += ”\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*`n`n”

1. Report on disabled users:

$RkReport += ”\*\*\* Disabled Users`n”

$RKReport += $RKUsers |

    Where-Object {$\_.Enabled -NE $true} |

        Format-Table -Property SamAccountName, Displayname |

            Out-String

1. Report on users who have not recently logged on:

$OneWeekAgo = (Get-Date).AddDays(-7)

$RKReport += ”`n\*\*\* Users Not logged in since $OneWeekAgo`n”

$RkReport += $RKUsers |

    Where-Object {$\_.Enabled -and $\_.LastLogonDate -le $OneWeekAgo} |

        Sort-Object -Property LastlogonDate |

            Format-Table -Property SamAccountName,lastlogondate |

                Out-String

1. Discover users with a high number of invalid password attempts:

$RKReport += ”`n\*\*\* High Number of Bad Password Attempts`n”

$RKReport += $RKUsers | Where-Object BadPwdCount -ge 5 |

  Format-Table -Property SamAccountName, BadPwdCount |

    Out-String

1. Add another report header line for this part of the report and create an empty array of privileged users

$RKReport += ”`n\*\*\* Privileged  User Report`n”

$PUsers = @()

1. Query the Enterprise Admins/Domain Admins/Scheme Admins groups for   
   members and add them to the $Pusers array:

# Get Enterprise Admins group members

$Members = Get-ADGroupMember -Identity ’Enterprise Admins’ -Recursive |

    Sort-Object -Property Name

$PUsers += foreach ($Member in $Members) {

    Get-ADUser -Identity $Member.SID -Properties \* |

        Select-Object -Property Name,

               @{Name=‘Group’;expression={‘Enterprise Admins’}},

               whenCreated,LastlogonDate

}

# Get Domain Admins group members

$Members =

  Get-ADGroupMember -Identity ’Domain Admins’ -Recursive |

    Sort-Object -Property Name

$PUsers += Foreach ($Member in $Members)

    {Get-ADUser -Identity $member.SID -Properties \* |

        Select-Object -Property Name,

                @{Name=‘Group’;expression={‘Domain Admins’}},

                WhenCreated, Lastlogondate,SamAccountName

}

# Get Schema Admins members

$Members =

  Get-ADGroupMember -Identity ’Schema Admins’ -Recursive |

    Sort-Object Name

$PUsers += Foreach ($Member in $Members) {

    Get-ADUser -Identity $member.SID -Properties \* |

        Select-Object -Property Name,

            @{Name=‘Group’;expression={‘Schema Admins’}}, `

            WhenCreated, Lastlogondate,SamAccountName

}

1. Add the special users to the report:

$RKReport += $PUsers | Out-String

1. Display the final report:

$RKReport

## How it works...

1. In this recipe, all the steps except the last one produce no output. The steps create a report which you view in the final step.

In step 1, you create a function, Get-ReskitUser, which creates a set of user objects related to each of the users in your AD. In step 2, you use the function to populate an array, $RKUsers, containing users and necessary details needed for your report.

1. With step 3, you build the header for the report, and then in step 4, you build a report section on disabled users. You use step 4 to add details of disabled user accounts. In step 5, you add details of users that have not logged on within the last 7 days. In step 6, you add details of users who have had more than 5 unsuccessful login attempts.
2. The final section of the report lists users that are members of crucial AD groups. With step 7, you create a header for this section. Then with step 8, you add details of members of these groups.
3. Once these steps are complete, in step 10, you can view the output of the report, which looks like this:

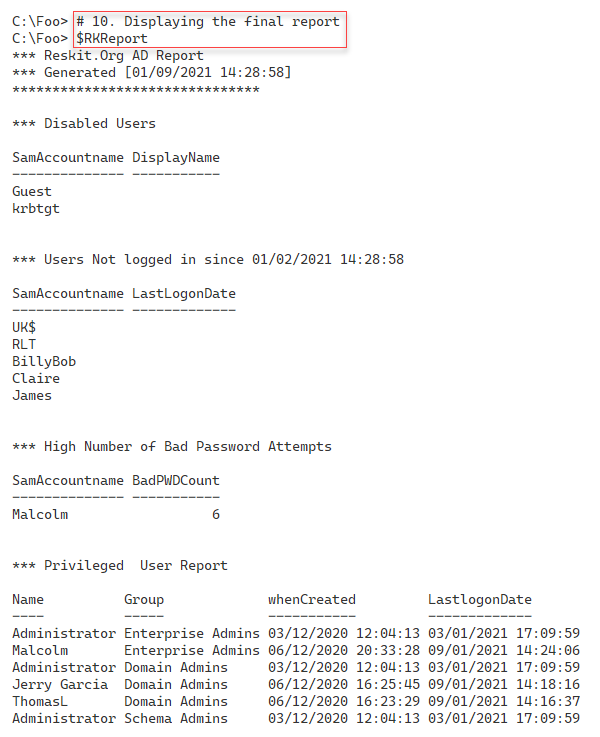


Figure 6.48: Displaying the final report

1. Insert image B42024\_06\_48.png

## There’s more...

In step 1, you create a function to retrieve users. This function allows you to reformat the properties as needed, improving the output in your report.

In step 4 through step 8, you build the report contents. To create the output you see in the above output, you have to log in to a host in the domain. Otherwise, these sections would be blank and contain no users.

In step 9, you see the final report. Note that the user Malcolm both had a high number of failed login attempts and has logged in at some point successfully. If you log in using a domain account and an incorrect password, AD rejects the login and increases the bad attempt count. Once you log in successfully, however, the bad logon count is zeroed.