Manage Windows Firewall with the command line

Article • 09/06/2024 •

Applies to: ✅ Windows 11, ✅ Windows 10, ✅ Windows Server 2025, ✅ Windows Server 2022, ✅ Windows Server 2019, ✅ Windows Server 2016

This article provides examples how to manage Windows Firewall with PowerShell and `netsh.exe`, which can be used to automate the management of Windows Firewall.

Set profile global defaults

Global defaults set the device behavior in a per-profile basis. Windows Firewall supports Domain, Private, and Public profiles.

Windows Firewall drops traffic that doesn't correspond to allowed unsolicited traffic, or traffic that is sent in response to a request by the device. If you find that the rules you create aren't enforced, you might need to enable Windows Firewall. Here's how to enable Windows Firewall on a local device:

PowerShell

```
PowerShell

Set-NetFirewallProfile -Profile Domain,Public,Private -Enabled True
```

Control Windows Firewall behavior

The global default settings can be defined through the command-line interface. These modifications are also available through the Windows Firewall console. The following scriptlets set the default inbound and outbound actions, specifies protected network connections, and disallows notifications to be displayed to the user when a program is blocked from receiving inbound connections. It allows unicast response to multicast or broadcast network traffic, and it specifies logging settings for troubleshooting.

PowerShell

```
PowerShell
```

```
Set-NetFirewallProfile -DefaultInboundAction Block -
DefaultOutboundAction Allow -NotifyOnListen False -
AllowUnicastResponseToMulticast True -LogFileName
%SystemRoot%\System32\LogFiles\Firewall\pfirewall.log
```

Disable Windows Firewall

Microsoft recommends that you don't disable Windows Firewall because you lose other benefits, such as the ability to use Internet Protocol security (IPsec) connection security rules, network protection from attacks that employ network fingerprinting, Windows Service Hardening, and boot time filters. Non-Microsoft firewall software can programmatically disable only the rule types of Windows Firewall that need to be disabled for compatibility. You shouldn't disable the firewall yourself for this purpose. If disabling Windows Firewall is required, don't disable it by stopping the Windows Firewall service (in the Services snap-in, the display name is Windows Defender Firewall and the service name is MpsSvc). Stopping the Windows Firewall service isn't supported by Microsoft and can cause problems, including:

- Start menu can stop working
- Modern applications can fail to install or update
- Activation of Windows via phone fails
- Application or OS incompatibilities that depend on Windows Firewall

The proper method to disable the Windows Firewall is to disable the Windows Firewall Profiles and leave the service running. Use the following procedure to turn off the firewall, or disable the Group Policy setting **Computer Configuration|Administrative Templates|Network|Network Connections|Windows Firewall|Domain Prolfile|Windows Firewall:Protect all network connections**. For more information, see Windows Firewall deployment guide. The following example disables Windows Firewall for all profiles.

PowerShell

PowerShell

```
Set-NetFirewallProfile -Profile Domain,Public,Private -Enabled False
```

Deploy basic firewall rules

This section provides scriptlet examples for creating, modifying, and deleting firewall rules.

Create firewall rules

Adding a firewall rule in Windows PowerShell looks a lot like it did in Netsh, but the parameters and values are specified differently. Here's an example of how to allow the Telnet application to listen on the network. This firewall rule is scoped to the local subnet by using a keyword instead of an IP address. Just like in Netsh, the rule is created on the local device, and it becomes effective immediately.

PowerShell

PowerShell

```
New-NetFirewallRule -DisplayName "Allow Inbound Telnet" -Direction
Inbound -Program %SystemRoot%\System32\tlntsvr.exe -RemoteAddress
LocalSubnet -Action Allow
```

The following scriptlet shows how to add a basic firewall rule that blocks outbound traffic from a specific application and local port to a Group Policy Object (GPO) in Active Directory. In Windows PowerShell, the policy store is specified as a parameter within the **New-NetFirewall** cmdlet. In Netsh, you must first specify the GPO that the commands in a Netsh session should modify. The commands you enter are run against the contents of the GPO, and the execution remains in effect until the Netsh session is ended or until another set store command is executed. Here, **domain.contoso.com** is the name of your Active Directory Domain Services (AD DS), and **gpo_name** is the name of the GPO that you want to modify. Quotation marks are required if there are any spaces in the GPO name.

PowerShell

PowerShell

```
New-NetFirewallRule -DisplayName "Block Outbound Telnet" -Direction
Outbound -Program %SystemRoot%\System32\tlntsvr.exe -Protocol TCP -
LocalPort 23 -Action Block -PolicyStore domain.contoso.com\gpo_name
```

GPO Caching

To reduce the burden on busy domain controllers, Windows PowerShell allows you to load a GPO to your local session, make all your changes in that session, and then save it back at all once. The following command performs the same actions as the previous example (by adding a Telnet rule to a GPO), but we do so by applying GPO caching in PowerShell. Changing the GPO by loading it onto your local session and using the *-GPOSession* parameter aren't supported in Netsh

PowerShell

```
$gpo = Open-NetGPO -PolicyStore domain.contoso.com\gpo_name
New-NetFirewallRule -DisplayName "Block Outbound Telnet" -Direction
Outbound -Program %SystemRoot%\System32\telnet.exe -Protocol TCP -
LocalPort 23 -Action Block -GPOSession $gpo
Save-NetGPO -GPOSession $gpo
```

This command doesn't batch your individual changes, it loads and saves the entire GPO at once. So if any other changes are made by other administrators, or in a different Windows PowerShell window, saving the GPO overwrites those changes.

Modify an existing firewall rule

When a rule is created, Netsh and Windows PowerShell allow you to change rule properties and influence, but the rule maintains its unique identifier (in Windows PowerShell, this identifier is specified with the *-Name* parameter). For example, you could have a rule **Allow Web 80** that enables TCP port 80 for inbound unsolicited traffic. You can change the rule to match a different remote IP address of a Web server whose traffic will be allowed by specifying the human-readable, localized name of the rule.

PowerShell

PowerShell

```
Set-NetFirewallRule -DisplayName "Allow Web 80" -RemoteAddress
192.168.0.2
```

Netsh requires you to provide the name of the rule for it to be changed and we don't have an alternate way of getting the firewall rule. In Windows PowerShell, you can query for the rule using its known properties. When you run `Get-NetFirewallRule`, you may notice that common conditions like addresses and ports don't appear. These conditions are represented in separate objects called Filters. As shown before, you can set all the conditions in New-NetFirewallRule and Set-NetFirewallRule. If you want to query for

firewall rules based on these fields (ports, addresses, security, interfaces, services), you'll need to get the filter objects themselves. You can change the remote endpoint of the **Allow Web 80** rule (as done previously) using filter objects. Using Windows PowerShell, you query by port using the port filter, then assuming other rules exist affecting the local port, you build with further queries until your desired rule is retrieved. In the following example, we assume the query returns a single firewall rule, which is then piped to the `Set-NetFirewallRule` cmdlet utilizing Windows PowerShell's ability to pipeline inputs.

PowerShell

```
Get-NetFirewallPortFilter | ?{$_.LocalPort -eq 80} | Get-NetFirewallRule
| ?{ $_.Direction -eq "Inbound" -and $_.Action -eq "Allow"} | Set-
NetFirewallRule -RemoteAddress 192.168.0.2
```

You can also query for rules using the wildcard character. The following example returns an array of firewall rules associated with a particular program. The elements of the array can be modified in subsequent `Set-NetFirewallRule` cmdlets.

PowerShell

```
Get-NetFirewallApplicationFilter -Program "*svchost*" | Get-
NetFirewallRule
```

Multiple rules in a group can be simultaneously modified when the associated group name is specified in a Set command. You can add firewall rules to specified management groups in order to manage multiple rules that share the same influences. In the following example, we add both inbound and outbound Telnet firewall rules to the group **Telnet Management**. In Windows PowerShell, group membership is specified when the rules are first created so we re-create the previous example rules. Adding rules to a custom rule group isn't possible in Netsh.

PowerShell

```
New-NetFirewallRule -DisplayName "Allow Inbound Telnet" -Direction
Inbound -Program %SystemRoot%\System32\tlntsvr.exe -RemoteAddress
LocalSubnet -Action Allow -Group "Telnet Management"
New-NetFirewallRule -DisplayName "Block Outbound Telnet" -Direction
Outbound -Program %SystemRoot%\System32\tlntsvr.exe -RemoteAddress
LocalSubnet -Action Allow -Group "Telnet Management"
```

If the group isn't specified at rule creation time, the rule can be added to the rule group using dot notation in Windows PowerShell. You can't specify the group using `Set-NetFirewallRule` since the command allows querying by rule group.

PowerShell

```
$rule = Get-NetFirewallRule -DisplayName "Allow Inbound Telnet"
$rule.Group = "Telnet Management"
$rule | Set-NetFirewallRule
```

With the help of the `Set` command, if the rule group name is specified, the group membership isn't modified but rather all rules of the group receive the same modifications indicated by the given parameters. The following scriptlet enables all rules in a predefined group containing remote management influencing firewall rules.

PowerShell

PowerShell

```
Set-NetFirewallRule -DisplayGroup "Windows Firewall Remote
Management" -Enabled True
```

There's also a separate `Enable-NetFirewallRule` cmdlet for enabling rules by group or by other properties of the rule.

PowerShell

```
Enable-NetFirewallRule -DisplayGroup "Windows Firewall Remote
Management" -Verbose
```

Delete a firewall rule

Rule objects can be disabled so that they're no longer active. In Windows PowerShell, the **Disable-NetFirewallRule** cmdlet will leave the rule on the system, but put it in a disabled state so the rule no longer is applied and impacts traffic. A disabled firewall rule can be re-enabled by **Enable-NetFirewallRule**. This cmdlet is different from the **Remove-NetFirewallRule**, which permanently removes the rule definition from the device. The following cmdlet deletes the specified existing firewall rule from the local policy store.

PowerShell

PowerShell

```
Remove-NetFirewallRule -DisplayName "Allow Web 80"
```

Like with other cmdlets, you can also query for rules to be removed. Here, all blocking firewall rules are deleted from the device.

PowerShell

```
Remove-NetFirewallRule -Action Block
```

It may be safer to query the rules with the **Get** command and save it in a variable, observe the rules to be affected, then pipe them to the **Remove** command, just as we did for the **Set** commands. The following example shows how you can view all the blocking firewall rules, and then delete the first four rules.

PowerShell

```
$x = Get-NetFirewallRule -Action Block
$x
$x[0-3] | Remove-NetFirewallRule
```

Manage remotely

Remote management using WinRM is enabled by default. The cmdlets that support the *CimSession* parameter use WinRM and can be managed remotely by default. The following example returns all firewall rules of the persistent store on a device named **RemoteDevice**.

PowerShell

```
Get-NetFirewallRule -CimSession RemoteDevice
```

We can perform any modifications or view rules on remote devices by using the *-CimSession* parameter. Here we remove a specific firewall rule from a remote device.

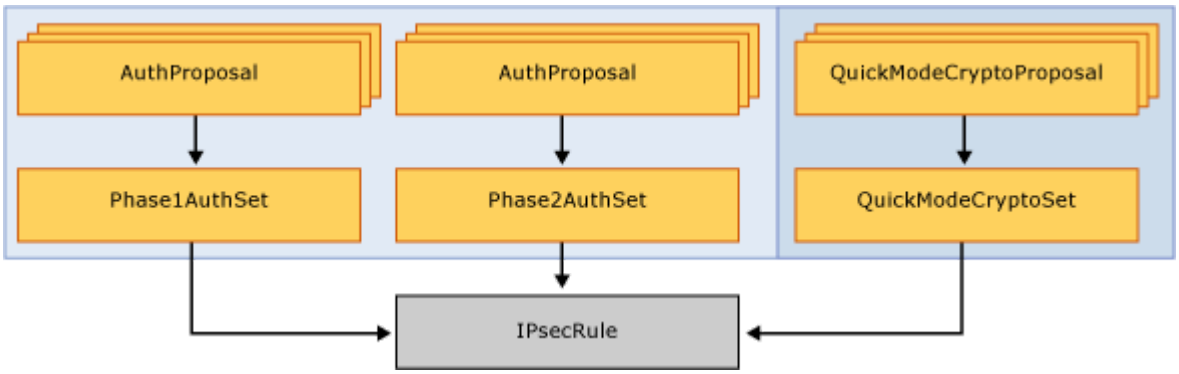
PowerShell

```
$RemoteSession = New-CimSession -ComputerName RemoteDevice
Remove-NetFirewallRule -DisplayName "AllowWeb80" -CimSession
$RemoteSession -Confirm
```

Deploy basic IPsec rule settings

An Internet Protocol security (IPsec) policy consists of rules that determine IPsec behavior. IPsec supports network-level peer authentication, data origin authentication, data

integrity, data confidentiality (encryption), and replay protection. Windows PowerShell can create powerful, complex IPsec policies like in Netsh and the Windows Firewall console. However, because Windows PowerShell is object-based rather than string token-based, configuration in Windows PowerShell offers greater control and flexibility. In Netsh, the authentication and cryptographic sets were specified as a list of comma-separated tokens in a specific format. In Windows PowerShell, rather than using default settings, you first create your desired authentication or cryptographic proposal objects and bundle them into lists in your preferred order. Then, you create one or more IPsec rules that reference these sets. The benefit of this model is that programmatic access to the information in the rules is much easier. See the following sections for clarifying examples.



Create IPsec rules

The following cmdlet creates basic IPsec transport mode rule in a Group Policy Object. An IPsec rule is simple to create; all that is required is the display name, and the remaining properties use default values. Inbound traffic is authenticated and integrity checked using the default quick mode and main mode settings. These default settings can be found in the console under Customize IPsec Defaults.

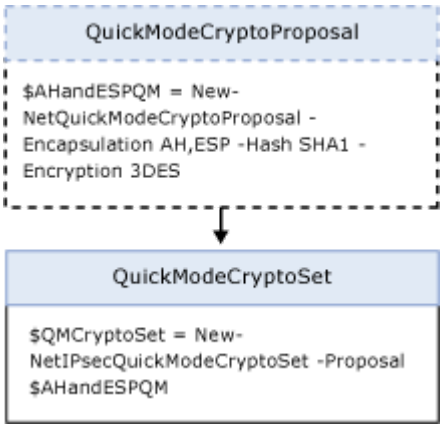
PowerShell

```
New-NetIPsecRule -DisplayName "Require Inbound Authentication" -
PolicyStore domain.contoso.com\gpo_name
```

Add custom authentication methods to an IPsec rule

If you want to create a custom set of quick-mode proposals that includes both AH and ESP in an IPsec rule object, you create the associated objects separately and link their associations. For more information about authentication methods, see Choosing the IPsec Protocol. You can then use the newly created custom quick-mode policies when you

create IPsec rules. The cryptography set object is linked to an IPsec rule object.



In this example, we build on the previously created IPsec rule by specifying a custom quick-mode crypto set. The final IPsec rule requires outbound traffic to be authenticated by the specified cryptography method.

PowerShell

PowerShell

```
$AHandESPQM = New-NetIPsecQuickModeCryptoProposal -Encapsulation
AH,ESP -AHHash SHA1 -ESPHash SHA1 -Encryption DES3
$QMCryptoSet = New-NetIPsecQuickModeCryptoSet -DisplayName
"ah:sha1+esp:sha1-des3" -Proposal $AHandESPQM -PolicyStore
domain.contoso.com\gpo_name
New-NetIPsecRule -DisplayName "Require Inbound Authentication" -
InboundSecurity Require -OutboundSecurity Request -
QuickModeCryptoSet $QMCryptoSet.Name -PolicyStore
domain.contoso.com\gpo_name
```

IKEv2 IPsec transport rules

A corporate network may need to secure communications with another agency. But, you discover the agency runs non-Windows operating systems and requires the use of the Internet Key Exchange Version 2 (IKEv2) standard. You can apply IKEv2 capabilities in Windows Server 2012 by specifying IKEv2 as the key module in an IPsec rule. This capability specification can only be done using computer certificate authentication and can't be used with phase-2 authentication.

PowerShell

```
New-NetIPsecRule -DisplayName "Require Inbound Authentication" -
InboundSecurity Require -OutboundSecurity Request -Phase1AuthSet
MyCertAuthSet -KeyModule IKEv2 -RemoteAddress $nonWindowsGateway
```

For more info about IKEv2, including scenarios, see Securing End-to-End IPsec Connections by Using IKEv2.

Copy an IPsec rule from one policy to another

Firewall and IPsec rules with the same rule properties can be duplicated to simplify the task of re-creating them within different policy stores. To copy the previously created rule from one policy store to another, the associated objects must also be copied separately. There's no need to copy associated firewall filters. You can query rules to be copied in the same way as other cmdlets. Copying individual rules is a task that isn't possible through the Netsh interface. Here's how you can accomplish it with Windows PowerShell.

PowerShell

```
$Rule = Get-NetIPsecRule -DisplayName "Require Inbound Authentication"
$Rule | Copy-NetIPsecRule -NewPolicyStore domain.costoso.com\new_gpo_name
$Rule | Copy-NetPhase1AuthSet -NewPolicyStore
domain.costoso.com\new_gpo_name
```

Handling Windows PowerShell errors

To handle errors in your Windows PowerShell scripts, you can use the *-ErrorAction* parameter. This parameter is especially useful with the **Remove** cmdlets. If you want to remove a particular rule, you'll notice that it fails if the rule isn't found. When rules are being removed, if the rule isn't already there, it's acceptable to ignore that error. In this case, you can do the following to suppress any "rule not found" errors during the remove operation.

PowerShell

```
Remove-NetFirewallRule -DisplayName "Contoso Messenger 98" -ErrorAction
SilentlyContinue
```

The use of wildcards can also suppress errors, but they could potentially match rules that you didn't intend to remove. These wildcards can be a useful shortcut, but should only be used if you know there aren't any extra rules that will be accidentally deleted. So the following cmdlet will also remove the rule, suppressing any "not found" errors.

PowerShell

```
Remove-NetFirewallRule -DisplayName "Contoso Messenger 98*"
```

When using wildcards, if you want to double-check the set of rules that is matched, you can use the *-WhatIf* parameter.

PowerShell

```
Remove-NetFirewallRule -DisplayName "Contoso Messenger 98*" -WhatIf
```

If you only want to delete some of the matched rules, you can use the *-Confirm* parameter to get a rule-by-rule confirmation prompt.

PowerShell

```
Remove-NetFirewallRule -DisplayName "Contoso Messenger 98*" -Confirm
```

You can also just perform the whole operation, displaying the name of each rule as the operation is performed.

PowerShell

```
Remove-NetFirewallRule -DisplayName "Contoso Messenger 98*" -Verbose
```

Monitor

The following Windows PowerShell commands are useful in the update cycle of a deployment phase. To allow you to view all the IPsec rules in a particular store, you can use the following commands. In Netsh, this command doesn't show rules where profile=domain,public or profile=domain,private. It only shows rules that have the single entry domain that is included in the rule. The following command examples will show the IPsec rules in all profiles.

PowerShell

PowerShell

```
Show-NetIPsecRule -PolicyStore ActiveStore
```

You can monitor main mode security associations for information such as which peers are currently connected to the device and which protection suite is used to form the security associations. Use the following cmdlet to view existing main mode rules and their security associations:

PowerShell

```
Get-NetIPsecMainModeSA
```

Find the source GPO of a rule

To view the properties of a particular rule or group of rules, you query for the rule. When a query returns fields that are specified as **NotConfigured**, you can determine which policy store a rule originates from. For objects that come from a GPO (the *-PolicyStoreSourceType* parameter is specified as **GroupPolicy** in the **Show** command), if *-TracePolicyStore* is passed, the name of the GPO is found and returned in the **PolicyStoreSource** field.

PowerShell

```
Get-NetIPsecRule -DisplayName "Require Inbound Authentication" -
TracePolicyStore
```

It's important to note that the revealed sources don't contain a domain name.

Deploy a basic domain isolation policy

IPsec can be used to isolate domain members from non-domain members. Domain isolation uses IPsec authentication to require that the domain-joined devices positively establish the identities of the communicating devices to improve security of an organization. One or more features of IPsec can be used to secure traffic with an IPsec rule object. To implement domain isolation on your network, the devices in the domain receive IPsec rules that block unsolicited inbound network traffic that isn't protected by IPsec. Here we create an IPsec rule that requires authentication by domain members. Through this authentication, you can isolate domain-joined devices from devices that aren't joined to a domain. In the following examples, Kerberos authentication is required for inbound traffic and requested for outbound traffic.

PowerShell

PowerShell

```
$kerbprop = New-NetIPsecAuthProposal -Machine -Kerberos
$Phase1AuthSet = New-NetIPsecPhase1AuthSet -DisplayName "Kerberos
Auth Phase1" -Proposal $kerbprop -PolicyStore
domain.contoso.com\domain_isolation
New-NetIPsecRule -DisplayName "Basic Domain Isolation Policy" -
Profile Domain -Phase1AuthSet $Phase1AuthSet.Name -InboundSecurity
Require -OutboundSecurity Request -PolicyStore
domain.contoso.com\domain_isolation
```

Configure IPsec tunnel mode

The following command creates an IPsec tunnel that routes traffic from a private network (192.168.0.0/16) through an interface on the local device (1.1.1.1) attached to a public network to a second device through its public interface (2.2.2.2) to another private network (192.157.0.0/16). All traffic through the tunnel is checked for integrity by using ESP/SHA1, and it's encrypted by using ESP/DES3.

PowerShell

PowerShell

```
$QMProposal = New-NetIPsecQuickModeCryptoProposal -Encapsulation ESP
-ESPHash SHA1 -Encryption DES3
$QMCryptoSet = New-NetIPsecQuickModeCryptoSet -DisplayName "esp:sha1-
des3" -Proposal $QMProposal
New-NetIPSecRule -DisplayName "Tunnel from HQ to Dallas Branch" -
Mode Tunnel -LocalAddress 192.168.0.0/16 -RemoteAddress
192.157.0.0/16 -LocalTunnelEndpoint 1.1.1.1 -
RemoteTunnelEndpoint 2.2.2.2 -InboundSecurity Require -
OutboundSecurity Require -QuickModeCryptoSet $QMCryptoSet.Name
```

Deploy secure firewall rules with IPsec

In situations where only secure traffic can be allowed through the Windows Firewall, a combination of manually configured firewall and IPsec rules are necessary. The firewall rules determine the level of security for allowed packets, and the underlying IPsec rules secure the traffic. The scenarios can be accomplished in Windows PowerShell and in Netsh, with many similarities in deployment.

Create a secure firewall rule (allow if secure)

Configuring firewalls rule to allow connections if they're secure requires the corresponding traffic to be authenticated and integrity protected, and then optionally encrypted by IPsec. The following example creates a firewall rule that requires traffic to be authenticated. The command permits inbound Telnet network traffic only if the connection from the remote device is authenticated by using a separate IPsec rule.

PowerShell

PowerShell

```
New-NetFirewallRule -DisplayName "Allow Authenticated Telnet" -
Direction Inbound -Program %SystemRoot%\System32\tlntsvr.exe -
Authentication Required -Action Allow
```

The following command creates an IPsec rule that requires a first (computer) authentication and then attempts an optional second (user) authentication. Creating this rule secures and allows the traffic through the firewall rule requirements for the messenger program.

PowerShell

PowerShell

```
$mkerbauthprop = New-NetIPsecAuthProposal -Machine -Kerberos
$mntlmauthprop = New-NetIPsecAuthProposal -Machine -NTLM
$P1Auth = New-NetIPsecPhase1AuthSet -DisplayName "Machine Auth" -
Proposal $mkerbauthprop,$mntlmauthprop
$ukerbauthprop = New-NetIPsecAuthProposal -User -Kerberos
$unentlmauthprop = New-NetIPsecAuthProposal -User -NTLM
$anonyauthprop = New-NetIPsecAuthProposal -Anonymous
$P2Auth = New-NetIPsecPhase2AuthSet -DisplayName "User Auth" -
Proposal $ukerbauthprop,$unentlmauthprop,$anonyauthprop
New-NetIPSecRule -DisplayName "Authenticate Both Computer and User" -
InboundSecurity Require -OutboundSecurity Require -Phase1AuthSet
$P1Auth.Name -Phase2AuthSet $P2Auth.Name
```

Isolate a server by requiring encryption and group membership

To improve the security of the devices in an organization, you can deploy domain isolation in which domain-members are restricted. They require authentication when communicating among each other and reject non-authenticated inbound connections. To

improve the security of servers with sensitive data, this data must be protected by allowing access only to a subset of devices within the enterprise domain. IPsec can provide this extra layer of protection by isolating the server. In server isolation, sensitive data access is restricted to users and devices with legitimate business need, and the data is additionally encrypted to prevent eavesdropping.

Create a firewall rule that requires group membership and encryption

To deploy server isolation, we layer a firewall rule that restricts traffic to authorized users or devices on the IPsec rule that enforces authentication. The following firewall rule allows Telnet traffic from user accounts that are members of a custom group called "Authorized to Access Server." This access can additionally be restricted based on the device, user, or both by specifying the restriction parameters. A Security Descriptor Definition Language (SDDL) string is created by extending a user or group's security identifier (SID). For more information about finding a group's SID, see: Finding the SID for a group account. Restricting access to a group allows administrations to extend strong authentication support through Windows Firewall and/or IPsec policies. The following example shows you how to create an SDDL string that represents security groups.

PowerShell

```
$user = new-object System.Security.Principal.NTAccount
("corp.contoso.com\Administrators")
$SIDofSecureUserGroup =
$user.Translate([System.Security.Principal.SecurityIdentifier]).Value
$secureUserGroup = "D:(A;;CC;;;$SIDofSecureUserGroup)"
```

By using the previous scriptlet, you can also get the SDDL string for a secure computer group as shown here:

PowerShell

```
$secureMachineGroup = "D:(A;;CC;;;$SIDofSecureMachineGroup)"
```

For more information about how to create security groups or how to determine the SDDL string, see Working with SIDs. Telnet is an application that doesn't provide encryption. This application can send data, such as names and passwords, over the network. This data can be intercepted by malicious users. If an administrator would like to allow the use of Telnet, but protect the traffic, a firewall rule that requires IPsec encryption can be created. This firewall rule is necessary so that the administrator can be certain that when this application is used, all of the traffic sent or received by this port is encrypted. If IPsec fails

to authorize the connection, no traffic is allowed from this application. In this example, we allow only authenticated and encrypted inbound Telnet traffic from a specified secure user group through the creation of the following firewall rule.

PowerShell

PowerShell

```
New-NetFirewallRule -DisplayName "Allow Encrypted Inbound Telnet to Group Members Only" -Program %SystemRoot%\System32\tlntsvr.exe -Protocol TCP -Direction Inbound -Action Allow -LocalPort 23 -Authentication Required -Encryption Required -RemoteUser $secureUserGroup -PolicyStore domain.contoso.com\Server_Isolation
```

Endpoint security enforcement

The previous example showed end to end security for a particular application. In situations where endpoint security is required for many applications, having a firewall rule per application can be cumbersome and difficult to manage. Authorization can override the per-rule basis and be done at the IPsec layer. In this example, we set the global IPsec setting to only allow transport mode traffic to come from an authorized user group with the following cmdlet. Consult the previous examples for working with security groups.

PowerShell

```
Set-NetFirewallSetting -RemoteMachineTransportAuthorizationList $secureMachineGroup
```

Create firewall rules that allow IPsec-protected network traffic (authenticated bypass)

Authenticated bypass allows traffic from a specified trusted device or user to override firewall block rules. This override is helpful when an administrator wants to use scanning servers to monitor and update devices without the need to use port-level exceptions. For more information, see How to enable authenticated firewall bypass. In this example, we assume that a blocking firewall rule exists. This example permits any network traffic on any port from any IP address to override the block rule, if the traffic is authenticated as originating from a device or user account that is a member of the specified device or user security group.

PowerShell

PowerShell

```
New-NetFirewallRule -DisplayName "Inbound Secure Bypass Rule" -
Direction Inbound -Authentication Required -OverrideBlockRules $true
-RemoteMachine $secureMachineGroup -RemoteUser $secureUserGroup -
PolicyStore domain.contoso.com\domain_isolation
```

Feedback

Was this page helpful? Yes No

Provide product feedback