Just Enough Administration

Published: 2014

For the latest information, please see

<http://blogs.msdn.com/powershell/>

and

<http://blogs.technet.com/privatecloud/>

[Introduction 1](#_Toc387143990)

[Problem Statement 2](#_Toc387143991)

[Just Enough Administration 2](#_Toc387143992)

[Solution Overview 3](#_Toc387143993)

[Technical Overview 5](#_Toc387143994)

[Windows PowerShell Desired State Configuration 5](#_Toc387143995)

[Windows PowerShell Constrained Endpoints 6](#_Toc387143996)

[JEA Toolkit 6](#_Toc387143997)

[Understanding use of Administrative Accounts 6](#_Toc387143998)

[Logging and Reporting 7](#_Toc387143999)

[How-To: Install JEA 8](#_Toc387144000)

[Load the xJEA resource 8](#_Toc387144001)

[Manual Installation 8](#_Toc387144002)

[Install using PSGet 8](#_Toc387144003)

[Install using NuGet 8](#_Toc387144004)

[Utilizing Windows a PowerShell DSC Pull Server 9](#_Toc387144005)

[Apply the DSC configuration 9](#_Toc387144006)

[JEAToolkit 9](#_Toc387144007)

[JEAEndPoint 10](#_Toc387144008)

[Example - DSC Script 11](#_Toc387144009)

[How-To: Create a Custom Proxy Function 12](#_Toc387144010)

[How-To: Access a JEA session 13](#_Toc387144011)

[Example – Connecting to Remote JEA 13](#_Toc387144012)

[Appendix 14](#_Toc387144013)

[User Context and Toolkit Security 14](#_Toc387144014)

[Language Mode 14](#_Toc387144015)

# Introduction

In the current world of Information Technology, protective measures do not stop at the network edge. Recent news reports based on security breach post-mortems indicate the need to protect assets using measures that reduce administrative access. While the principle of least privilege has always been known to IT Security professionals, there is a need in the industry for a standardized method of constructing an operator experience that reduces access with a more sophisticated level of granularity than what is available in many traditional access control models.

Just Enough Administration (JEA) is a solution designed to help protect Server systems. This is accomplished by allowing specific users to perform administrative tasks on servers without giving them administrator privileges and auditing all actions that these users performed. JEA is based on PowerShell constrained runspaces, a technology that is already being used to secure administrative tasks in environments such as Exchange online.

We are focusing on fortifying your existing environment allowing you to adopt at your own pace a solution that reduces administrator exposure and protects against lateral movement from breached machines

# Problem Statement

When organizations grant access to servers, a level of risk is assumed based on trust. Traditionally, this is accomplished by granting administrative rights to users that are entrusted with IT administration. Depending on the system, the granularity of access ranges from full control over every aspect of the system to more granular levels where only certain components can be changed. The access to perform specific tasks is grouped in to a definition of a role, but in many circumstances even the most well intentioned role based access control (RBAC) models still take on more risk than desired.

A good example would be a case where auditors need to be given access to the environment to verify information but should never be able to make changes. In many cases this requires unnecessary administrator privileged access in order for the auditor to gather information for business critical systems.

A second example would be a “front line” or “help desk” administrator who has been delegated responsibility to perform basic troubleshooting tasks on an instance of Microsoft SQL Server but should not have access to the server itself.

Another example of this scenario would be a server operator who is expected to apply the latest security updates according to what has been approved in Windows Server Update Services by a Senior IT person. This typically requires total control over the application server either directly or abstracted through a tool where the user could theoretically gain total control, because the changes being made are to the core operating system components. In addition, tasks may need to be completed before and after applying the updates such as starting and stopping services or rebooting the hardware so that updates to system files which are normally in use can apply during startup.

A final case is an operator for a service provider who is responsible for implementing changes on behalf of a tenant. This can be a very difficult model to properly implement without exposing access to tenant data. An example would be an operator who should complete tasks such as restarting a service during an outage or provision an additional capability, but should not be able to log in to the tenant environment at all.

And it goes on and on…

Today, it is common practice in each of these cases, to grant users with excessive administrative privileges in order to perform their job. This results in severe consequences when accounts are breached.

## Just Enough Administration

Just Enough Administration (JEA) is able to overcome many of these challenges through the following benefits:

* Users can only perform tasks that they are authorized to do as part of their role (using PowerShell constrained runspace)
* Users perform these tasks without being promoted to have administrator privilege on the server
* The tasks that users are allowed to perform and the user access to the servers are defined and managed from a central configuration server (using Desired State Configuration)
* Detailed logging occurs so that access and details of change within an environment are always documented.

# Solution Overview

Just Enough Administration (JEA) is delivered using built in capabilities in the Windows PowerShell scripting language. The functionality is delivered as a PowerShell Desired State Configuration resource so no new binaries are required to be installed. Although this solution is implemented in Windows, through the use of standards based management the technology can be applied to non-Windows systems as well.

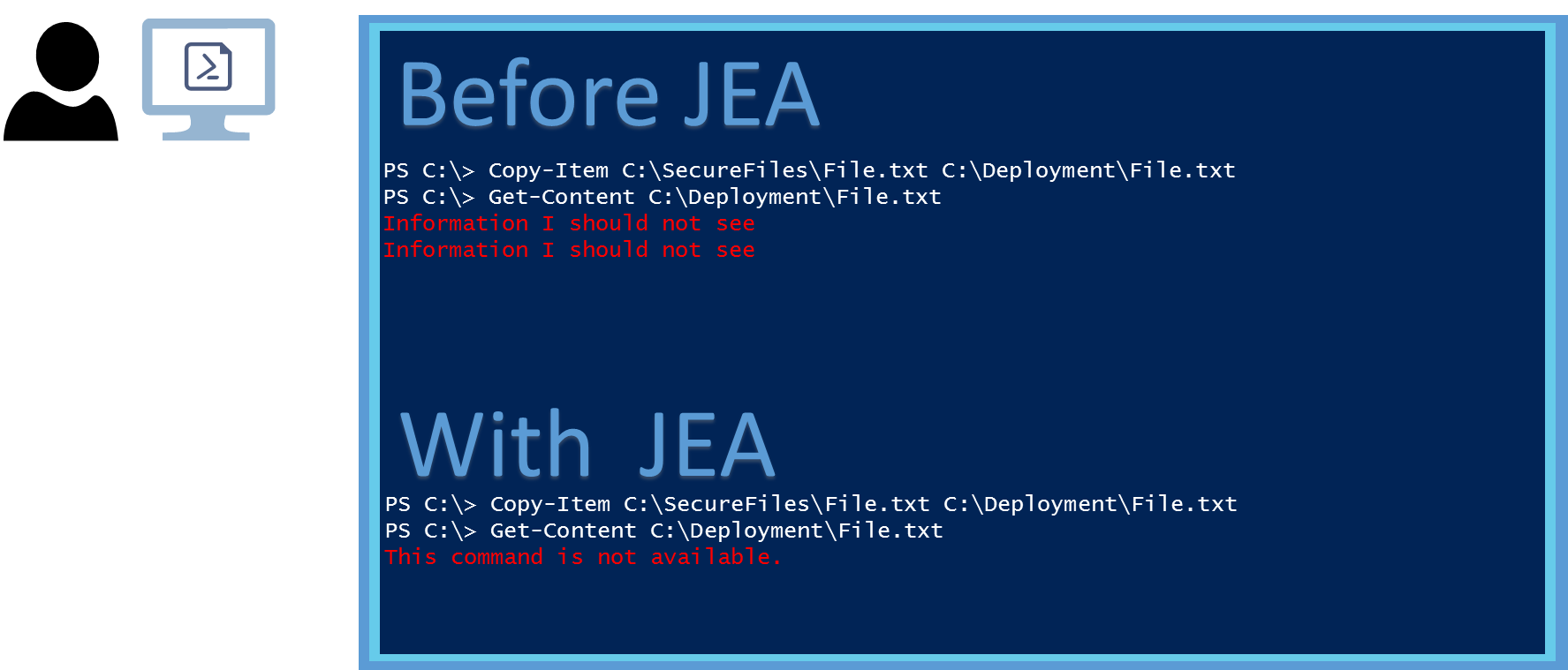
JEA is currently delivered as a Desired State Configuration Resource Kit module (xJEA) and required the Windows Management Framework (WMF) V5 Preview which only runs on the latest version of Windows client and server. This version will evolve rapidly based upon community feedback and no backward compatibility is guaranteed. The toolkit will be made available on previous versions of Windows concurrent with the availability of WMF V5 on those platforms.

The automation for JEA is delivered by Windows PowerShell Desired State Configuration (DSC). The core components of a JEA environment include:

* JEA Toolkit configuration – this represents a set of tasks (e.g.: auditor tasks, SQL admin tasks…) that a user can perform on the server. This is configured through the set of PowerShell cmdlets and parameters that the user can run in order to accomplish that task. For example: an “Auditor” JEA Toolkit will include the “Get-Events” and “Get-Logs” cmdlets. A “SQL admin” JEA Toolkit will include the “Stop-service SQL” and “Start-service SQL” cmdlets and parameters
* JEA Endpoint configuration – for each set of servers a JEA Endpoint represents an administration point for users on the Windows Server. A JEA Endpoint is composed from:
  + One or more JEA Toolkits (e.g.: “Auditor” toolkit)
  + An Access Control List (ACL) representing the users that can access the JEA Endpoint (e.g.: Auditors\_Security\_Group) – NOTE: these users should not be administrators on the server

For employees performing their role (e.g.: Auditor) JEA is exposed as a PowerShell session endpoint. Once connected, the JEA based environment functions as a normal PowerShell experience with only specific commands visible and only specific arguments accepted. Behind the scenes, on the server, the cmdlets are running in the context of a hidden local administrator account that is specifically configured for that JEA Endpoint.

If the employee (e.g.:Auditor) attempts to run an unauthorized command such as “cmd” they would receive an error in addition to their activity being logged. This creates a “blast container” around the administrator where even if malicious intent is present, the employees have limited impact. In addition, if the server is breached (e.g.: by malware) there are no administrator credentials that can be harvested to make lateral movement to other servers limiting the extent of the breach to that server.



Consider an administrator who is tasked with deploying a new web based application that includes privileged Human Resources information such as salaries for company executives. When the administrator connects to a JEA endpoint to complete work, he or she would be presented with options such as:

* New-WebApplication
* Set-WebContent
* Restart-WebApplication
* Test-WebApplication

Notice that the names of the commands align with the work that will be performed. No additional commands such as those commonly used to view data or copy the content to the administrator’s workstation are presented.

Further, the parameters for each command and the arguments that will be accepted can be restricted. Consider the following scenario using restrictions for the above commands and the benefit for each.

* New-WebApplication –Server (specific web servers)
  + Deploy only on the web servers where the application should be deployed and nowhere else.
* Set-WebContent –Source (secure location) –Destination (secure location)
  + Provision content from a secure location to a secure location without having a method to view the data or copy it to a less secure location.
* Restart-WebApplication –Application (specific application)
  + Perform maintenance task without having long term access to the application.
* Test-WebApplication –Application
  + Validate health of the application through synthetic transactions without having access to the application.

In each of the above examples, the command might actually support additional parameters but the properties of the PowerShell session for the JEA endpoint restrict those parameters. The session properties also validate the arguments. If the administrator attempted to do something malicious such as copying secure content to an insecure location not only would the properties of the session prevent the command from running, the details of the attempt would be written to audit logs.

# Technical Overview

JEA consists of Windows PowerShell Desired State Configuration resources and configuration scripts that together provide a simplified method of constructing and managing constrained endpoints adhering to prescribed boundaries.

Windows PowerShell Constrained Endpoints offer rich functionality that could be considered complex to configure and implement, especially at scale. JEA simplifies constrained endpoints to make the concept easier to implement and manage using the concept of toolkits.

In order to better understand how JEA works, first understand the core constructs that Windows PowerShell leverages to create the JEA environment.

*For specific examples of each solution, see the respective ‘How-To’ sections later in this document.*

## Windows PowerShell Desired State Configuration

Windows PowerShell Desired State Configuration (DSC) provides a method of using script files to apply, monitor, and/or maintain a configuration in declarative language. DSC scripts packaged as Windows PowerShell modules ready to be consumed are named DSC resources. A DSC resource consists of idempotent functions, meaning the script defines a configuration in such a way that it can be run multiple times on the same machine without error or undesired outcome.

To take advantage of a resource once it has been copied to a machine, a script is run to define a configuration. This script is known as the configuration script. The configuration script is organized in to human readable sections each representing a set of declarative statements used as parameters when running functions from each resource. The information is stored as a MOF file in Windows and applied to the machine.

In the following example, the ‘Configuration’ section defines a configuration that includes installation of the Hyper-V role on a Windows Server. Next the configuration is executed by calling ‘Example’ to create a MOF file. Finally, ‘Start-DSCConfiguration’ tells Windows to apply the configuration with verbose output and to wait for all changes to complete before exiting.

Configuration Example

{

Node 'localhost'

{

WindowsFeature Hyper-V

{

Ensure = 'Present'

Name = 'Hyper-V'

}

}

}

Example -OutputPath .\Example

Start-DscConfiguration -wait -force -verbose -path .\Example

For JEA, PowerShell DSC is used for installation and configuration of JEA Toolkits and Endpoints on a server using the same approach. Just as we see ‘WindowsFeature Hyper-V’ in the example, we would expect to see ‘JEAEndPoint Hyper-V’ if constructing an endpoint where an operator would performance maintenance on a Hyper-V server.

DSC has two modes – “Push” where an administrator remotely configures servers by specifying the “ComputerName” parameter when executing “Start-DSCConfiguration” from a local machine, and “Pull” where a server is configured to periodically pull its configuration from a central DSC server. Both models enable deployment at scan and can be used for applying JEA Endpoint configuration to servers.

## Windows PowerShell Constrained Endpoints

When a user opens PowerShell using the traditional method of clicking on the desktop icon, all modules and snap-ins available on that machine are visible. This can be demonstrated using the “Get-Command” cmdlet. A user opening a PowerShell 4.0 prompt will be greeted with over a thousand built-in commands.

The concept of a session was introduced with PowerShell remoting as the WinRM runspace a remote user connects to. Sessions are not limited to remote connections. A local user may also create and enter a PowerShell session. The remoting context lead to sessions being referred to as endpoints.

When a session is created, it is possible to apply restrictions. The basic method for restricting a session is through use of a PowerShell session configuration files or PowerShell session startup scripts. Startup scripts are the most flexible and allow for advanced configurations through the use Windows PowerShell script language. These capabilities include strict limits on which commands should be available including “Language Modes” as described in the Appendix of this document.

For JEA, PowerShell Constrained Endpoints provide a remotely accessible PowerShell session with defined characteristics that limit what functionality is available to the user. An example endpoint configuration is provided in the How-To section of this document.

## JEA Toolkit

A JEA Toolkit serves as a definition of the commands that should be available for a JEA endpoint. The definition can be set directly within a configuration script using a string value or through a CSV (comma separated values) file implemented in the configuration script using the Get-Content command. The value of authoring toolkits using a CSV file is to simplify the experience in to one that can be driven by any text editor or spreadsheet application.

The toolkit file is simply a flat text file containing comma separated values using a schema that is expected by the JEA initialization script. A simple method for editing the file is to open it in Microsoft Excel and add contents to each cell, then save it as a CSV file type when finished.

The schema is as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ModuleName | Name | Parameter | ValidateSet | ValidatePattern | ParameterType |
| The name of the module whose cmdlets should be available. | The cmdletthat should be available in the session | List a parameter that should be allowed. Repeat 1 row per parameter. Default = blank (all) | If a value is given for Parameter, optionally list a semicolon separated validation set to restrict a set of arguments. Default = blank (all) | If a value is given for Parameter, as an alternative to ValidateSet you can validate a Regular Expression pattern for arguments. Default = blank (all) | If a value is given for Parameter, you can set the Type to ensure only valid data is passed. |

An example toolkit configuration is provided in the How-To section of this document.

## Understanding use of Administrative Accounts

As was mentioned, when a user connects to a JEA Endpoint through a remote session, that user should not be an administrator on the server. Having said that, some (or all) of the cmdlets that the user will be performing will require administrator rights. For example, if a user needs to restart a service, this is an operation that normal users cannot do on a server.

The way this is addressed in the JEA Endpoint is that while the user is connecting to the server as a normal user, the JEA Endpoint then performs a “Run as” using a special local administrator account that is configured for that JEA Endpoint so that the user can perform the administrative operations (while still limited to a specific set of cmdlets)

This special local administrator account is automatically created by the JEA Endpoint DSC resource when the JEA Endpoint is configured and is only used by that JEA Endpoint.

Each JEA Endpoint on a server has its own local administrator account and these accounts have randomized passwords that are maintained by the JEA Endpoints. As these accounts are local to each server, a compromised server will not lead to a lateral movement attack on other servers.

One consequence of using a “Run as” is that the user that is connected to the JEA Endpoint is very limited when trying to perform off-server actions such as access to an SMB share (since the JEA Endpoint Run as account is local to the server). The users can still do off-server actions by providing their credentials. This behavior is by design so that users will not be able to take advantage of JEA to exploit areas that they do not have proper access to as well as the need for an appropriate audit trail when the users access off-server resources

## Logging and Reporting

All operations performed through the JEA Endpoint are recorded in the Windows Event Logs. When an end user logs in to a PowerShell Session, the process ID, RunSpaceID, and information from WinRM are written as Windows events. In addition, information about the WinRM session is written to the Windows Event Log.

In a centrally managed environment, these event logs provide a method of data correlation to determine who accessed the environment, when, and what changes were made. In order to form a complete picture, the data correlation will need to include details about the actions taken in the context of the JSA user account and the logged in operator.

# How-To: Install JEA

The JEA installation process includes the DSC resource ‘xJEA’ that is copied to the local machine’s Modules folder followed by a DSC Configuration script that creates and applies JEA Endpoints

This process is the same for both simple and complex environments and it is idempotent. This means you can define one endpoint or many endpoints in the same script and apply it to a server. To modify the server you only need to run the same script again with different settings.

## Load the xJEA resource

The first step is to load the xJEA resource on each server where an endpoint should be created. The following methods provide flexible options for loading the DSC resource xJEA.

## Manual Installation

To install JEA manually, first locate the xJEA DSC resource from the Windows PowerShell Desired State Configuration Resource Kit available online via <http://gallery.technet.microsoft.com/scriptcenter/DSC-Resource-Kit-All-c449312d>.

Extract the files from the downloaded archive (.zip) file and copy the folder in to the following location:  
'C:\Program Files\WindowsPowerShell\Modules'

The xJEA folder should appear in the root of the Modules directory. The contents of the xJEA folder should include an xJea.psd1 file and a folder named DSCResources.

## Install using Install-Module

The new Windows PowerShell Install-Module cmdlet (Available in the Windows Management May2014 Preview) simplifies the process of installation to a one-line execution when an Internet connection is available. The xJEA module is downloaded from the Config Gallery hosted by Microsoft Azure and copied in to the local Modules folder in the Program Files directory of the local machine. Currently, Install-Module must be executed in the context of a user account. This means the scenario for this method of installation would be an administrator who is logging in to the server and installing JEA.

To install and configure Install-Module on the local machine, copy and paste the command given at the Config Gallery website:

<http://go.microsoft.com/fwlink/?LinkID=398385&clcid=0x409>

Once PSGet is installed on the node, the command to load the xJEA resource is:  
Install-Module xJEA

## Install using NuGet

When installing xJEA on an endpoint server using an automated method such as Windows PowerShell DSC resources, Microsoft Azure VM Agent Custom Script Extension, or Windows Azure Pack VMRole Resource Extension, you can include a two-line PowerShell script and the process will complete without user interaction.

Invoke-WebRequest -Uri "http://nuget.org/nuget.exe" -OutFile "$env:TEMP\NuGet.exe"

&"$env:TEMP\NuGet.exe" install xJea -NoCache `  
 -Source ‘http://go.microsoft.com/fwlink/?LinkID=397631&clcid=0x409’ `

-ExcludeVersion -PackageSaveMode "nuspec" `

-OutputDirectory "$env:ProgramFiles\WindowsPowerShell\Modules"

## Utilizing Windows a PowerShell DSC Pull Server

A Windows PowerShell DSC Pull Server acts as a centralized solution for storing DSC resources and MOF files created by DSC configuration scripts, so that servers managed by DSC can be deployed in an automated fashion.

More information about configuration of a DSC Pull Server is available at the PowerShell blog:  
<http://blogs.msdn.com/b/powershell/archive/2013/11/21/powershell-dsc-resource-for-configuring-pull-server-environment.aspx>

To support delivery of the JEA resource, download the xJEA DSC resource and copy it in to the appropriate folder on the Pull Server:  
'C:\ProgramData\PSDSCPullServer\Modules'

Full details about how to configure the DSC server are available in the blog post linked above.

## Apply the DSC configuration

After the xJEA resource has been copied to the local machine, the second step is to create the JEA toolkits and endpoints. Both are defined in declarative language using a DSC configuration. To make it easy for testing, an example configuration script is included with the download for the Toolkit above. The critical components of the configuration script are the helper functions, the JEAToolKit resource, and the JEAEndPoint resource.

The basic sections of a DSC configuration for JEA include:

## JEAToolkit

The JEAToolkit section of a DSC configuration script defines what commands should be available inside a JEA session. The properties include Name, CommandSpecs, Applications, and Ensure. The information provided in CommandSpecs is parsed as comma separated values which makes content creation very simple in any text editor or spreadsheet application.

Importing the information in to a configuration script can be done either directly using a here-string value, or by importing information from a stored .csv file.

To create a new Toolkit – create a new JEAToolkit configuration in the script and set the properties, as in the example below. In this example, the toolkit is created using a CSV file for ease of authoring. The result would allow a connected user to run “Get-EventLog”, “Get-Content” with only the “-Name” parameter, “ipconfig”, and nothing else.

JeaToolkit ExampleToolkit

{

Name = "Auditor toolkit"

CommandSpecs = Get-Content “C:\AuditorToolkit\Toolkit.csv” -Raw

applications = "ipconfig"

}

This would load the contents of a CSV file that can be created using Microsoft Excel, or any text editor. An example of the content is given below. Once you have created this table in Excel, use “Save As” to change the type to “CSV (comma deliminted) (\*.csv)”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Name | Parameter | ValidateSet | ValidatePattern |
|  | Get-EventLog | -LogName | Application |  |
|  | Get-Content | -Path |  | C:\\logs\*.txt |
| SMBShare | Get-\* |  |  |  |

To create additional toolkits in the same configuration, repeat this section using a different name for each.

## JEAEndPoint

The JEAEndPoint section of a DSC configuration script defines the metadata properties for the session that will be created. The properties include Name, Toolkits, SecurityDescriptorSddl, Group, CleanAll, and Ensure. A definition for each property is given in the table below.

|  |  |
| --- | --- |
| Property | Description |
| Ensure | Defines whether the endpoint should be created or removed. |
| Name | Sets a name for the registered endpoint. This will be used by operators to choose which endpoint they should connect to. |
| Toolkits | The JEA toolkits that should be available from this endpoint. |
| SecurityDescriptorSddl | An SDDL that defines access for the session. |
| Group | The machine local group that the account created to host the session should be a member of. If not provided, defaults to ‘Administrators’ |
| CleanAll | Boolean value that when set to True will remove all endpoint configurations from the endpoint server. |

To create a new JEA EndPoint – create a new JEAEndPoint configuration in the DSC configuration script and set each of the properties. In this example, the endpoint will publish the toolkit defined in the example above.

JeaEndPoint AuditorEndPoint

{

Name = 'Auditor EndPoint'

Ensure = 'Present'

Toolkit = ‘AuditorToolkit’

DependsOn = '[JeaToolkit]AuditorToolkit'

}

It is also possible to define an endpoint which has multiple toolkits available.

JeaEndPoint AuditorEndPoint

{

Name = 'Auditor EndPoint'

Ensure = 'Present'

Toolkit = ‘AuditorToolkit’,’Auditor2Toolkit’,’Auditor3Toolkit’

DependsOn = '[JeaToolkit]AuditorToolkit', '[JeaToolkit]Auditor2Toolkit', '[JeaToolkit]Auditor3Toolkit'

}

To create additional endpoints in the same configuration, repeat this section using a different name for each.

This information is passed in to the JEAEndPoint resource that creates the actual Windows PowerShell Constrained Endpoint. A local account is automatically created named ‘JSA-<name’[[1]](#endnote-2) based on the Name property. A random password is generated and the account password is reset. An initialization script is created for the session and the session is registered on the host. The default Group and SDDL values permit anyone who is a local administrator on the host to connect to the session.

## Example - DSC Script

To test a fully functional configuration script, open the .ps1 file included in the root of the xJEA download. This ensures future improvements are always available in the Config Gallery source.

|  |
| --- |
| configuration Demo2  {  Import-DscResource -module xjea  xJeaToolKit Demo2Toolkit  {  Name = 'Demo2Toolkit'  CommandSpecs = @"  name,Parameter,ValidateSet,ValidatePattern  Get-Process  Get-Service  stop-process,name,calc;notepad  restart-service,name,,^A  "@  }  xJeaEndPoint Demo2EP  {  Name = 'Demo2EP'  Toolkit = 'Demo2Toolkit'  DependsOn = '[xJeaToolKit]Demo2Toolkit'  }  }  Demo2 -OutputPath C:\DSCDemo  Start-DscConfiguration -Path C:\DSCDemo -ComputerName localhost -Verbose -wait |

# How-To: Create a Custom Proxy Function

In addition to the security benefits of JEA, it is also possible to create custom proxy functions that simplify the operator experience. This is accomplished by authoring a PowerShell module (.psm1) and loading it on the endpoint server, and then surfacing the functions using the JEA toolkit. As an example, rather than giving an operator access to the native PowerShell cmdlet for Get-Content to read in the top 100 lines of a specific log file, it is possible to surface a command with a name that is friendlier to the context of the task such as ‘Get-LogFile’.

In this example, the following text would be saved as a .psm1 file and copied to the Modules folder on the endpoint server. The file creation can also be automated using the DSC resource ‘File’.

function Get-LogFile {

Get-Content -Path C:\Log0001.txt | Select -first 100

}

Next in the toolkit CSV file, make the following entry to make the command visible.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Name | Parameter | ValidateSet | ValidatePattern |
|  | Get-LogFile |  |  |  |

The operator now has a simplified experience with task-associated command names and is not expected to provide parameters or arguments, which means the domain knowledge required of where to find the log file or the file name is no longer required.

In circumstances where argument validation or pattern validation is complex, custom proxy functions can provide a method of simplifying the work effort in restricting what actions are available to the operator by surfacing the task that should be performed in a function that obfuscates that native commands. This reduces the risk of making an error when creating a toolkit and inadvertently exposing an unintended argument.

# How-To: Access a JEA session

When a user would like to connect to a JEA endpoint and perform work, the process is the same as connecting to any remote PowerShell session with one difference, they must specify the configuration name. This would correlate to the name of the JEAEndPoint specified in the DSC configuration script.

The latest details on PowerShell remoting are available in the online documentation.  
<http://technet.microsoft.com/en-us/library/dd347744.aspx>

In most cases the users will be connecting from their workstation experience and will prefer a “persistent session” meaning that the session should remain available even when they are not actively doing work. In a trusted environment such as when both the workstation and the JEA endpoint servers are all members of the same Active Directory environment, credentials will be passed automatically using Kerberos. In untrusted scenarios, credentials must be first stored and then provided when connecting.

## Example – Connecting to Remote JEA

$Session = New-PSSession -ConfigurationName JEAEndPointName -ComputerName Remote\_Computer\_Name

Enter-PSSession $Session

# Appendix

## User Context and Toolkit Security

When an operator connects to a JEA endpoint, the authentication to the PowerShell session validates they have access to connect. The endpoint configuration defines a ‘Run As’ account for the session. When the session is registered a local user is created with the name pattern “JSA-<name>” meaning JEA Session Account followed by a dash and then the name of the endpoint. A randomly generated secure password is set for the account and it is added to the machine local administrators group.

Upon entering the PS session, the operator will find their security context is not their own account, it is the context of the JSA account with limited visibility to commands which prevents restricted action. This is by design to create a ‘blast container’ where the operator can make approved changes to the local machine without their own account ever having permission.

To make changes to a remote server or an application on the local host that does not inherit permissions from the local machine groups, the operator will need to provide credentials in to the session.

It is important when constructing a new toolkit to understand this model and ensure no commands are ever made visible in a session that could be used by a malicious person to give themselves additional access.

## Language Mode

Windows PowerShell includes the ability to restrict the elements of the Windows PowerShell language available within a session. This is a built-in method for preventing access to common PowerShell tools which, while convenient and useful in everyday scripting and command execution, could be used by an administrator in a constrained session when attempting to “break out” of the restricted environment of give themselves more access that what they should have.

The scripts in the DSC resource xJEA automate the configuration of the PowerShell session and set the LanguageMode property to ‘**NoLanguage**’. This is the most restrictive setting available.

For the latest detailed information about Windows PowerShell Language Modes, see the following documentation:  
<http://technet.microsoft.com/en-us/library/dn433292.aspx>

1. JSA stands for Jea Session Account [↑](#endnote-ref-2)