Ivanti Patch for Windows® Servers

Guidelines for Creating Custom ITScripts

ivanti

Copyright and Trademarks

Document Information and Print History

Document number: N/A

Date	Version	Description
October 2011	VMware vCenter Protect Essentials Plus	Initial release of the **Guidelines for Creating Custom ITScripts** document.
September 2012	Vmware vCenter Protect 8.0.1	Update product name and version, update cover graphics.
May 2013	Shavlik Protect 9.0	Rebrand to Shavlik Protect. Add info about ESXi Hypervisors and Powershell modules.
April 2014	Shavlik Protect 9.1	Update Web links.
September 2015	Shavlik Protect 9.2	Add info about the ITScripts add-on license key for Shavlik Protect Standard users. Add ST-GetTargetOS function.
April 2017	Ivanti Patch for Windows® Servers 9.3	Rebrand to Ivanti, remove reference to threat management.

Table of Contents

This page intentionally left blank.

The document is designed for duplex printing.

OVERVIEW

Tip: To view an overview video on the ITScripts feature, click the video icon.

If you are an Ivanti Patch for Windows® Servers Advanced user, or if you are an Ivanti Patch for Windows® Servers Standard user and have purchased an ITScripts add-on license key, you can import your own scripts that will completely integrate into the product environment. User scripts can be executed against individual machines and groups of machines in exactly the same manner as executing the scripts provided with Ivanti Patch for Windows® Servers. Advantages to running scripts in Ivanti Patch for Windows® Servers include:

- Scripts execute against the machines and machine groups you have already defined in Ivanti Patch for Windows® Servers
- Use the machine and machine-group credentials you have already entered in Ivanti Patch for Windows® Servers
- Scripts execute in the background
- Script execution can be run immediately or scheduled to run in the future
- Scripts are executed in parallel against the target machines and usually complete in a fraction of the time that it would take to run them serially and you can control the level of parallelism
- Script output is captured to files that you can review at your convenience
- Status of script execution is displayed within Ivanti Patch for Windows® Servers
- You can open the result files directly from Ivanti Patch for Windows® Servers
- Your scripts can be parameterized, and different sets of parameters can be saved in ITScript Templates or provided when you start the script or schedule it for execution
- Scripts can use the PowerShell remoting features, allowing the broadest set of capabilities provided by Windows PowerShell
- Scripts can use PowerShell modules. Different scripts can import different versions of a given module.
- Only scripts signed by authorities that you trust will be allowed to execute
- Ivanti Patch for Windows® Servers also provides a number of PowerShell variables and functions that help simplify many common operations.

Most PowerShell scripts can be used in Ivanti Patch for Windows® Servers with only minor modifications. This document describes how to prepare your scripts and import them into Ivanti Patch for Windows® Servers.

VARIABLES AND FUNCTIONS

Ivanti Patch for Windows® Servers provides a set of PowerShell variables and functions to every script. The variable names all begin with "ST_". The function names all begin with "ST-". You can use these variables and functions in your scripts.

PowerShell variables can have different scopes:

- **Global**: Available to the current PowerShell session
- **Script**: Declared in the script (outside of functions) and local to that script
- **Private**: Declared in a function and local to that function
- **Local**: The current scope, which can be global, script, or private

The variables supplied with Ivanti Patch for Windows® Servers are declared at the global scope. Since each target machine is running in its own PowerShell session, this is equivalent to being declared in the scope of the script.

Run Variables

The following table lists the per-run variables. All machines will see the same value for these variables. Do not modify the values of variables marked as [Read-only].

Variable Name	Description
ST_OutputDirectory	The full path to the base output directory. A subdirectory will be created for the run. [Read-only]
ST_RunDirectory	The full path to the run output directory. [Read-only]
ST_RunErrorFile	The full path to the run error file. Errors that are not specific to a particular machine should be written to this file. This file will also contain messages about machines that could not be resolved to an IP address, and no machine-specific subfolder will be created for machines that could not be resolved. [Read-only]
ST_RunName	The name of the run specified in the user-interface when the run was initiated or scheduled, with the run time appended. The default is the name of the script or template. [Read-only]
ST_RunOnConsole	Set to $false if the script is using PowerShell remoting; otherwise $true. [Read-only]
ST_RunOutputFile	The full path to the standard run output file. All output that would appear on the console if run at a command prompt will be placed in this file. [Read-only]
ST_RunResult	This is a short run result that will appear in the Operations Monitor. It is limited to 100 characters in length. If you do not explicitly set this variable in your script, it will be set to indicate the number of machines that had errors and the number of machine resolution errors.

Machine Variables

The following table lists the per-machine variables. The value of these variables may be different for each target machine. You should not modify the values of variables marked as [Read-only].

Variable Name	Description
ST_ComputerName	The name of the target computer. [Read-only]
ST_Credential	A PSCredential object that contains the user name and SecureString password used to connect to the target machine. This variable can be passed to any command that supports the the Credential parameter. [Read-only]
ST_DomainName	The name of the target computer domain or workgroup. [Read-only]
ST_MachineDirectory	The full path to the machine directory. This will typically be the machine name, but if the machine name contains characters that are not allowed in folder names, then those characters will be replaced with underscore characters. [Read-only]
ST_MachineError	A Boolean value indicating that one or more errors were detected during the execution of this script targeting this specific machine.
ST_MachineErrorFile	The full path to the machine error file. If errors are detected during execution of the script, they will appear in this file. [Read-only]
ST_MachineOutputFile	The full path to the machine output file. Output generated by the script to the standard output will be captured in this file.
ST_MachineResult	This is a short machine result that will appear in the Operations Monitor. It is limited to 100 characters in length. If you do not explicitly set this variable in your script, it will be set to “Success” if no machine errors were detected, or to the error message of the first error encountered for that machine. Use the ST-SetMachineResult function to explicitly set the value of this variable.
ST_MachineResultSet	A Boolean value indicating that the script set the ST_MachineResult variable. Use the ST-SetMachineResult function to set both the ST_MachineResult variable and this variable.

Functions

Ivanti Patch for Windows® Servers provides a set of functions that are available to every script you create. They are listed in the following table.

Function Name	Description
ST-GetTargetOS	This function attempts to connect to the target system, query WMI and retrieve the Win32_OperatingSystem object.
ST-SetMachine Result	This function takes a string parameter. It sets the ST_MachineResult variable to the string passed in and set ST_MachineResultSet to $true.
ST-SetMachine Error	This function takes a string argument. It sets the ST_MachineResult variable to the string passed in. It also sets the ST_MachineResultSet and ST_MachineError variables to $true.
ST-SendMessage	This function takes a string parameter. The string will be displayed in the Operations Monitor. Use this function to monitor the progress of long-running scripts. When the script is complete the string in ST_MachineResult will be displayed in the operations monitor. Except during debug, you should avoid calling this method in scripts that execute quickly. The string will be truncated to 100 characters if a longer string is passed in.
ST-Create MachineDirectory	Normally, the machine directory is not created until output is written. If your script is creating files in the machine directory, you should invoke this function to create the directory before writing anything to the directory. Calling this function more than once will have no negative effect.
ST-CreateRun Directory	Normally, the run directory is not created until the run writes some output. If your script is creating files in the run directory, you should invoke this function to create the directory before writing anything to the directory. Calling this function more than once will have no negative effect.
ST-ComputerAnd Credential	This function returns appropriate values to use for the ComputerName and Credential parameters for the target machine. See the *Specifying ComputerName and Credential Parameters* section of this document for details.
ST-SubCC	This function takes a string parameter, which is a valid PowerShell command. This function substitutes "$ST_CC" in that command string with appropriate values to use for the ComputerName and Credential parameters for the command. It then executes the resulting command and returns the results of the execution. See the *Specifying ComputerName and Credential Parameters* section of this document for details.

TARGET TYPE

Ivanti Patch for Windows® Servers simplifies many IT functions, including patch management, asset management and script execution. The machines that are being managed are referred to as the "target machines," or "managed machines."

The ITScripts feature supports four different script execution environments. In the Script Catalog Manager, these are distinguished by their target type.

Console

Console scripts run on the console and do not target a group of managed machines. In the Script Catalog Manager, these have a target type of "Console" since the default target of script commands is the console (the local machine). These scripts are executed by selecting **Tools > Run console ITScripts**.

Any

These scripts target a group of managed machines without using PowerShell remoting. In the Script Catalog Manager, these have a target type of "Any" since the target machines generally do not require any special configuration. These scripts are executed from the home screen, Machine View, or Scan View. The commands in this type of script typically specify the target computer and credentials by using the **ComputerName** and **Credential** parameters on the command.

WinRM Remoting

These scripts target a group of managed machines using PowerShell remoting. In the Script Catalog Manager, these have a target type of "WinRM Remoting" and require that PowerShell remoting is configured on target machines. These scripts are executed from the home screen, Machine View, or Scan View. Because these scripts are executed on the target machine, the target of commands is typically the local machine, and therefore the **ComputerName** and **Credential** arguments do not need to be specified on the command.

ESXi Hypervisor

The script runs against an ESXi Server or a vCenter Server. This type of script may use VMware vSphere PowerCLI. VMware vSphere PowerCLI lets you automate all aspects of vSphere management, including network, storage, VM, guest OS and more. Scripts of this type only run against machine groups that contain ESXi servers. If the machine group contains any other machines, they will be ignored when this script executes.

SPECIFYING COMPUTERNAME AND CREDENTIAL PARAMETERS

When creating custom scripts, you specify the target type in the script metadata. Keep the target type in mind when writing the script. The primary thing to consider when writing the script will be whether you should include the ComputerName and Credential parameters on the commands. The ITScripts environment provides the computer name in the ST_ComputerName variable and the credential for that computer in the ST_Credential variable. For example, when writing a script with target type of "Any" you might specify this command:

```
Get-WmiObject win32_Share -ComputerName $ST_ComputerName -Credential
$ST_Credential
```

However, this command will fail if run against the local machine with the following error:

```
   Get-WmiObject : User credentials cannot be used for local
connections
```

In other words, this command will work if the target type is "Any" and the target machine is not the console, but it will fail in the following cases:

- Target type is 'Console'
- Target type is 'WinRM Remoting'
- Target type is 'Any' and the target is the console computer

To take these cases into account, your script may need to test whether it is using remoting, or if the target is the console machine. To simplify script creation, ITScripts provides two functions: **ST-ComputerAndCredential** and **ST-SubCC**.

ST-ComputerAndCredential

This function returns a string containing the appropriate string to use for the -ComputerName and -Credential parameters based on the type of script, whether it is executing on the console, and whether credentials were provided for the machine. It will return one of these strings:

```
"-ComputerName $ST_ComputerName –Credential $ST_Credential"
"-ComputerName $ST_ComputerName"
""
```

You can use this in your script command without concern for the environment you are executing in. For example:

```
$cc = ST-ComputerAndCredential
$shares = Invoke-Expression "Get-WmiObject win32_Share $cc"
```

Invoke-Expression is used in order to substitute the value of $cc before executing the command. The issue with this construct is that you must remember to escape special characters in your command. For example, the “$” and quote characters need to be escaped in the following command:

```
$ST_CC = ST-ComputerAndCredential
$registryObject = Invoke-Expression "Get-WmiObject -list
                  -namespace root\default $ST_CC |
                  where-object { `$_.name -eq `"StdRegProv`" }"
```

The following function makes this even simpler.

ST_SubCC

This function combines the two commands shown in the previous example. The first time it is invoked, it puts the result of ST-ComputerAndCredential into a PowerShell variable named ST_CC. It then passes the resulting command to Invoke-Expression. The thing to remember when using ST_SubCC is to place the command inside single quotes, not double quotes. The previous examples would be:

```
$shares = ST-SubCC ‘Get-WmiObject win32_Share $ST_CC’

$registryObject = ST-SubCC ‘Get-WmiObject -list
                  -namespace root\default $ST_CC |
                  where-object { $_.name -eq "StdRegProv" }’
```

UNSUPPORTED POWERSHELL COMMANDS

The Ivanti Patch for Windows® Servers scripting engine differs somewhat from running scripts in a PowerShell command window or in the PowerShell_ISE application. Each PowerShell application is a different “host” environment and supports different host options. The Ivanti Patch for Windows® Servers script engine does not currently implement the following host commands. Attempting to use these commands in your scripts will result in an error.

- Out-Host
- Read-Host
- Write-Host
- Out-GridView

Use Out-Default, Out-String, or Write-Output in place of Out-Host and Write-Host. Since Ivanti Patch for Windows® Servers scripts run in a background service, interactive scripts are not supported. Therefore, there is no substitute for Read-Host or Out-GridView.

OUTPUT

Output that is written to the standard output stream will be written to the machine output file. If errors are detected by the Ivanti Patch for Windows® Servers script engine, detailed error messages will be written to the machine error file.

We recommend that you explicitly format the output as you would like it to appear in the output file. If you do not explicitly format the output, the default formatter will be used. This will limit the fields that are output and the way in which they are presented.

It is particularly important to format output for scripts that are using PowerShell remoting. The default behavior for remote scripts is to serialize the script output objects on the target machine and deserialize then on the console. The deserialized objects are snapshots that have properties but no methods. You may get errors or unexpected results unless you explicitly format the output as strings.

There are several ways to format your output. Here are some of them.

- **Out-String:** Outputs a series of strings
- **Format-List:** Formats the output as a list of properties with each property on a new line
- **Format-Table:** Formats the object as a table
- **Format-Wide:** Formats objects as a wide table that displays only one property of each object
- **Select-Object:** Select the properties you want to output and optionally change the property names. This allows you fine control over what is output.
- **Export-CSV:** Save the output as a CSV file. You can output to a temporary file, then use Get-Content to write it to the standard output.

The following example retrieves a list of services, and then creates CSV output of the computer name, status, service name, and display name.

```
$fn = [io.path]::GetTempFileName()
Get-Service -ComputerName  "$ST_ComputerName" | select-
object
          @{Name="ComputerName";
Expression={$ST_ComputerName}}, Status, Name,
           DisplayName | export-csv "$fn" -notype
get-content $fn
remove-item $fn
```

PRE-EXECUTION AND POST-EXECUTION FUNCTIONS

The Ivanti Patch for Windows® Servers script engine executes the script in parallel against multiple target machines. Sometimes you may want to do some setup operations on the console computer prior to executing the script and/or some finalization operations after all machines are complete. To specify the setup operations, include a function named “ST-ConsoleBegin” within your script. This function will be called before the script is executed against any target machines. To specify finalization operations, include a function named “ST-ConsoleEnd” within your script. This function will be called after the main script is executed against all target machines.

These functions must take zero parameters and be formatted as follows:

```
function ST-ConsoleBegin
{
        #body of function indented from left margin
}
function ST-ConsoleEnd
{
        #body of function indented from left margin
}
```

The function declarations and the opening and closing braces must begin in column 1. All other lines of the functions should not begin in column 1.

These functions are executed on the console only. Therefore, it is generally not necessary to use the **ComputerName** or **Credential** parameters on any of the cmdlets invoked in these functions.

SCRIPT METADATA

In order for a script to be used in Ivanti Patch for Windows® Servers, it must contain metadata that uniquely identifies the script and describes its functionality and input parameters. This section will describe how to create that metadata.

Let's start with a simple one-line script that has one input parameter:

```
Get-Service $serviceName -ComputerName "$ST_ComputerName"
```

The complete script with the metadata that is required by Ivanti Patch for Windows® Servers is shown below. Some of the information in optional, and each element will be described in detail later.

```
<#
<stScript uid="d0d3c64c-19a3-4879-9396-ac8769d60c9c">
  <name>GetServices</name>
  <version>1.0.0.0</version>
  <author>My Company</author>
  <scriptType type="any" />
  <minEngineVersion>8.0.0.0</minEngineVersion>
  <modifiesTarget>false</modifiesTarget>
  <options>
    <option name="OutputMachineResults" value="true" />
    <option name="OutputRunResults" value="true" />
    <option name="CombineOutputFiles" value="false" />
    <option name="DeleteMachineFiles" value="false" />
  </options>
  <concurrency default="16" />
  <description>
    <category>Information</category>
    <purpose>Get the services on the target
             machine</purpose>
    <inputs>Wildcard of service name</inputs>
    <outputs>List of installed services</outputs>
  </description>
  <parameters>
    <parameter>
      <name>serviceName</name>
      <description>Wildcard service(s) to query
             </description>
      <default>”*”</default>
    </parameter>
  </parameters>
</stScript>
#>
Get-Service $serviceName -ComputerName "$ST_ComputerName"
```

Metadata Block

The metadata is an XML fragment inside a script comment.

The script comment delimiters are "<#" and "#>. For our purposes, these delimiters must be on lines by themselves and begin in column 1.

The root element of the XML document is "stScript". It has one attribute, the unique identifier of the script. This is a globally unique identifier (GUID). A GUID is specified by 32 hex characters in the format "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx". There are many standard tools for generating GUIDs.

The beginning and ending elements must be on lines by themselves, beginning in column 1. Therefore, all script used in Ivanti Patch for Windows® Servers should begin with these two lines (substituting a different GUID for each script):

```
<#
<stScript uid="d0d3c64c-19a3-4879-9396-ac8769d60c9c">
```

and end with these two lines:

```
</stScript>
#>
```

The other elements should appear in the order described below.

name

This required element provides the name of the script as it will appear within Ivanti Patch for Windows® Servers. The name must be between 1 and 100 characters. The name may contain spaces and special characters.

Example:

```
<name>GetServices</name>
```

version

This required element specifies the version of the script. The version is specified as four integer numbers separated by periods. When you modify a script, the newer version must have a version which is greater than any previous versions you have used.

Example:

```
<version>1.0.0.0</version>
```

author

This required element identifies the author of the script. It is customary to use either the author's name or the name of the company. This field must be between 1 and 256 characters and must not be blank.

Example:

```
<author>My Company</author>
```

scriptType

This optional element specifies the type of script execution. It has a required attribute, “type”, which must contain one of the following:

- **type=“any”** indicates that the script runs against one or more target machines, but does not require PowerShell remoting. This is the most common script type and is the default if the **scriptType** element is not specified.
- **type=”consoleOnly”** indicates that the script does not run against a set of target machines. This type of script cannot be run against machine groups or selected machines. Such a script will appear in the available scripts only when your select **Tools > Run console ITScripts** from the Ivanti Patch for Windows® Servers menu.
- **type=”remote”** indicates that the script runs against one or more target machines and requires PowerShell remoting to be available and configured on the target machines.
- **type=”esxServer”** indicates that the script runs against an ESXi Server or a vCenter Server. Scripts of this type run only against machine groups that contain ESXi Servers. If the machine group contains any other machines, they will be ignored when this script executes.

Example:

```
<scriptType type="any" />
```

minEngine Version

This optional element specifies the minimum ITScripts engine version required to execute this script. If not specified, the script should be written to run with any version of the ITScripts engine.

Example:

```
<minEngineVersion>8.0.0.0</minEngineVersion>
```

modifiesTarget

This optional element indicates whether the script makes any modifications to the target machines. If not specified, it defaults to “false”, suggesting that the script only gathers information but does not modify the target machines. This is for documentation only. Its value does not guarantee that the script does not modify the target machines.

Example:

```
<modifiesTarget>false</modifiesTarget>
```

options

This optional element and its four sub-elements tell the ITScripts engine which standard output files should be generated. By default, the engine will generate these standard output files:

- **runoutput.txt**: One file for the entire run if any run output is generated
- **runerror.txt**: One file for the entire run if errors were detected
- **machineoutput.txt**: One file for each target machine if any machine output is generated
- **machineerror.txt**: One file for each target machine if errors were detected

If you specify the following in the metadata, the runoutput.txt and runerror.txt files will not be generated:

```
<option name="OutputRunResults" value="false" />
```

If you specify the following in the metadata, the machineoutput.txt and machineerror.txt files will not be generated:

```
<option name="OutputMachineResults" value="true" />
```

If you'd like all the machineoutput.txt files for all machines to be concatenated into the runoutput.txt file, and all the machineerror.txt files concatenated into runerror.txt, then specify this option:

```
<option name="CombineOutputFiles" value="true" />
```

If you choose to combine the individual machine output and error files, then you can choose to delete the individual files by specifying the following:

```
<option name="DeleteMachineFiles" value="true" />
```

Example:

```
<options>
   <option name="OutputMachineResults" value="true" />
   <option name="OutputRunResults" value="true" />
   <option name="CombineOutputFiles" value="false" />
   <option name="DeleteMachineFiles" value="false" />
</options>
```

concurrency

This optional element specifies the default maximum concurrency level to use when a script is executed against a set of target machines. You can override this in Ivanti Patch for Windows® Servers by creating an ITScripts template and specifying a different concurrency value. If not specified, the default maximum concurrency is set to 32.

Example:

```
<concurrency default="16" />
```

description

The “description” element and its four sub-elements are all required. The information entered here will be displayed within Ivanti Patch for Windows® Servers.

category

The required “category” element is a child of the description element. The category helps in organizing your scripts. The category must be between 1 and 50 characters. Standard categories might include: “Configuration”, “Group Policy”, “Information”, “Maintenance”, “Monitoring”, and “Support”.

purpose

The required “purpose” element is a child of the description element. It must contain between 1 and 1024 characters. It should contain a concise and accurate description of what the script is used for.

inputs

The required “inputs” element is a child of the description element. It must contain between 1 and 1024 characters. This should accurately describe the input parameters required by the script. If there are no input parameters, simply specify “None”.

outputs

The required “outputs” element is a child of the description element. It must contain between 1 and 1024 characters. This should accurately describe the output generated by the script.

Example:

```
  <description>
    <category>Information</category>
    <purpose>Get the services on the target  machine</purpose>
    <inputs>Wildcard of service name</inputs>
    <outputs>List of installed services</outputs>
</description>
```

parameters

The “parameters” element must be specified if the script requires any input parameters. There must be one “parameter” child element for each input parameter. Each “parameter” element must have three child elements:

- **name:** The input parameter name with no leading ‘$’. The name must be the valid PowerShell variable name used in the script.
- **description:** This description will appear in Ivanti Patch for Windows® Servers.
- **default:** This element specifies the default parameter value that will be used for the parameter unless the user specifies an alternate value in a template or when starting or scheduling the script execution. If the default is a string, enclose it in apostrophes or quotes. Apostrophes allow the string to contain characters that have special meaning in PowerShell, such as the $ and quote. To embed such characters within quotes, precede them with the tic character (`).

Example:

```
<parameters>
   <parameter>
     <name>serviceName</name>
     <description>Wildcard service(s) to    find</description>
     <default>”*”</default>
   </parameter>
 </parameters>
```

SIGNING SCRIPTS

A script must be signed by an authority trusted by the console in order to be imported or executed by Ivanti Patch for Windows® Servers. All scripts provided with Ivanti Patch for Windows® Servers are signed by Ivanti. During the installation of Ivanti Patch for Windows® Servers, the Ivanti certificate is added to the trusted certificate store on the console machine.

You must sign scripts that you create before importing them into Ivanti Patch for Windows® Servers. To do this, you need a signing certificate. That certificate must be issued by an authority that is in the trust list for the console(s) that are going to execute the scripts you create.

This section will describe one way to create a signing certificate and add it to the trusted certificates on the console.

Creating a Self-signing Certificate Authority

This section will describe how to generate a certificate authority that can issue signing certificates. You can use this authority to generate a signing certificate that is used internally in your organization.

This method uses the MakeCert.exe utility from Microsoft. This tool is installed with Visual Studio and with the Windows SDK. See http://msdn.microsoft.com/en-us/library/ms229859.aspx for information on opening a Visual Studio or Windows SDK command prompt.

From a Visual Studio or Windows SDK command prompt, enter the following command:

Note: This command should be entered on a single line.

```
makecert -n "CN=PowerShell Local Certificate Root" -a sha1 -eku
1.3.6.1.5.5.7.3.3 -r -sv root.pvk root.cer -ss Root
-sr localMachine
```

You may substitute another string for the certificate name (“CN=xxx”) .

This command will:

1. Create a private key file named root.pvk (you can rename this on the –sv option).
2. Create the security certificate file named root.cer (you can rename this on the –sv option).
3. Add the certificate to the Root store (-ss Root) for the local machine (-sr localMachine).

You can use the Certificate snap-in for the Microsoft Management Console to inspect or delete certificates.

See http://msdn.microsoft.com/en-us/library/bfsktky3(v=VS.100).aspx for details of the MakeCert utility.

Creating a Signing Certificate

The next step is to create the signing certificate. To do so, enter the following at the command prompt:

```
makecert -pe -n "CN=PowerShell User" -ss MY -a sha1 -eku
1.3.6.1.5.5.7.3.3 -iv root.pvk -ic root1.cer
```

Again, you can substitute a different certificate name. The private key file name specified after the –iv argument must match the name following the –sv argument in the preceding section. This command will create an issuer's certificate file named root1.cer. You can specify a different name following the –ic argument.

This certificate is added to the store specified following the –ss argument. In this case, it is the user store. You can verify this using the Certificate snap-in for the Microsoft Management Console by looking under **Certificates – Current User > Personal > Certificates**.

You can also verify the certificate from within PowerShell by entering the following command:

```
Get-ChildItem cert:\CurrentUser\My -codesigning | Where-Object
{$_.Subject -match "CN=PowerShell User"}
```

If the certificate is found, it will display the thumbprint and subject of the certificate, similar to this:

```
Thumbprint                                Subject
----------                                -------
8249ABF7E1CE182DBE43D4FB522CC08C07907299  CN=PowerShell User
```

Signing a Script

This section will describe how to sign a script file from a PowerShell prompt or within the PowerShell ISE.

From the PowerShell command prompt, change to the directory containing your script file. In this example, it is named **test-script.ps1**. Then enter the following two commands:

Note: The first command should be entered on a single line. The **CN=PowerShell User** should match the name you specified when creating the signing certificate.

```
$cert = @(Get-ChildItem cert:\CurrentUser\My -codesigning |
Where-Object {$_.Subject -match "CN=PowerShell User"})[0]
```

```
Set-AuthenticodeSignature .\test-script.ps1 $cert
```

The first command will locate the certificate in the certificate store. The second command signs the script using that certificate.

Signing the script will add a block at the end of the script that looks similar to this:

```
# SIG # Begin signature block
# MIIEMwYJKoZIhvcNAQcCoIIEJDCCBCACAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQU4QyppfQY+5HviH7wuIBvvmRf
……
# SIG # End signature block
```

If you make any changes to the script, you will need to resign it.

Signing a Script Using a PFX File

If you'd like to sign scripts using the same signing certificate on multiple machines without installing the signing certificate on each machine, you can use a PFX (Personal Information Exchange) file.

To create a PFX file:

1) Run **CertMgr.exe**.
2) On the **Personal** tab, select the signing certificate.
3) Click **Export**.
4) On the first export wizard screen, click **Next**.
5) Select the **Personal Information Exchange** option.

   If there is an **Enable strong protection** option, choose it.
6) Enter a password when prompted.

   This will be required when using this pfx file to sign scripts.
7) When prompted, type a name for the pfx file.

To sign a script using the PFX file, enter the following commands in PowerShell:

```
$cert = Get-PfxCertificate mycert.pfx
Set-AuthenticodeSignature .\test-script.ps1 $cert
```

When prompted, supply the password you created in Step 6.

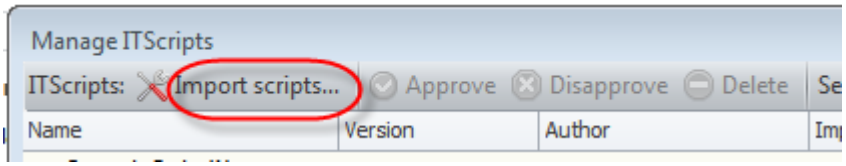
IMPORTING USER SCRIPTS IN IVANTI PATCH FOR WINDOWS® SERVERS

Once you've created a script with the required metadata and digital signature, you can import it into Ivanti Patch for Windows® Servers.

1. Select Manage > ITScripts.

   The **Manage IT Scripts** dialog will appear.

2. Click **Import scripts**.



3. Navigate to the signed script file and click **Open**.

   The script will be imported into Ivanti Patch for Windows® Servers.

4. Select the script in the list and click **Approve**.

   This script will now appear along with the approved Ivanti-provided scripts throughout the application.

Note: A console must trust the authority that issued the certificate in order to import or execute the script on that console. If you import user scripts on one console, they will appear on other consoles that are using the same database. However, if the other consoles don't trust the signer, they will not be able to execute the scripts.