This document includes basic information about Windows PowerShell and its role with SolarWinds SAM.
2 Configuring and Integrating PowerShell

Copyright© 1995-2012 SolarWinds. All rights reserved worldwide. No part of this document may be reproduced by any means nor modified, decompiled, disassembled, published or distributed, in whole or in part, or translated to any electronic medium or other means without the written consent of SolarWinds. All right, title and interest in and to the software and documentation are and shall remain the exclusive property of SolarWinds and its licensors. SolarWinds Orion™, SolarWinds Cirrus™, and SolarWinds Toolset™ are trademarks of SolarWinds and SolarWinds.net® and the SolarWinds logo are registered trademarks of SolarWinds All other trademarks contained in this document and in the Software are the property of their respective owners.

SOLARWINDS DISCLAIMS ALL WARRANTIES, CONDITIONS OR OTHER TERMS, EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, ON SOFTWARE AND DOCUMENTATION FURNISHED HEREUNDER INCLUDING WITHOUT LIMITATION THE WARRANTIES OF DESIGN, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL SOLARWINDS, ITS SUPPLIERS OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, WHETHER ARISING IN TORT, CONTRACT OR ANY OTHER LEGAL THEORY EVEN IF SOLARWINDS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Microsoft® and Windows 2000® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Graph Layout Toolkit and Graph Editor Toolkit © 1992 - 2001 Tom Sawyer Software, Oakland, California. All Rights Reserved.
Portions Copyright © ComponentOne, LLC 1991-2002. All Rights Reserved.
Document Revised 9/11/2013- DJR

SolarWinds does not provide customer support for any scripting language. For scripting support with SAM, the SolarWinds public community on thwack.com may be of assistance.
The Basics of PowerShell

The ability to employ PowerShell scripts within SAM is a powerful advantage for system administrators. This document merely provides an introduction to PowerShell, as well as its role with SAM.

Windows PowerShell is a command-line shell created for system administrators. PowerShell includes an interactive prompt and a scripting environment that can be used independently or in combination.

PowerShell is built on top of the .NET Framework Common Language Runtime (CLR) and the .NET Framework, and accepts and returns .NET Framework objects.

PowerShell also introduces the cmdlet. A cmdlet is a simple command that can manipulate objects in PowerShell. Cmdlets have a unique format -- a verb and noun separated by a dash (-), such as Get-Help. You can use each cmdlet separately or in combination to perform complex tasks. PowerShell includes more than one hundred cmdlets, and you can write your own.

PowerShell gives you access to the file system on the computer. In addition, PowerShell providers enable you to access other data stores, such as the registry, for example.

Things you should know about PowerShell:

- PowerShell does not process text. Instead, it processes objects based on the .NET Framework.
- PowerShell comes with a set of built-in commands with a consistent interface.
**PowerShell and SAM: Configuration and Usage**

In order to use PowerShell with SAM you must have PowerShell 2.0 installed on the SAM and target servers. PowerShell 2.0 can be found here: [http://support.microsoft.com/kb/968930](http://support.microsoft.com/kb/968930)

Once PowerShell is installed on the SAM and target servers, ensure that Windows Remote Management (WinRM) is properly configured and enabled on the SAM and target servers. To do this, follow these steps:

1. On the SAM server, open a command prompt as an Administrator. To do this, go to the **Start** menu and right-click the **Cmd.exe** and then select **Run as Administrator**.

2. Enter the following in the command prompt:
   ```
   winrm quickconfig -q
   winrm set winrm/config/client @{TrustedHosts="*"}
   ```
   **Note:** For Windows 2008 R2, use the following command to replace the italicized command directly above in Step 2:
   ```
   winrm set winrm/config/client '@{TrustedHosts="*"}'
   ```

3. On the target server, open a command prompt as an Administrator and enter the following:
   ```
   winrm quickconfig
   winrm set winrm/config/client @{TrustedHosts="IP_ADDRESS"}
   ```
   where **IP address** is the IP address of your SAM server. Once you have successfully completed these steps, PowerShell will be able to properly communicate with SAM.
Exchange 2010 Management Tools
Many SAM PowerShell components rely on Exchange Management Tools. Both the 2007 and 2010 editions are acceptable. Exchange 2010 Management Tools will be used for the following instructions.

When and where do Exchange Management Tools need to be installed for use with SAM?
Many SAM PowerShell components rely on Exchange Management Tools. Both the 2007 and 2010 editions are acceptable.

If you plan to monitor Exchange 2007 then you must use the Exchange 2007 Management Tools. If you plan to monitor Exchange 2010 then you will need to install the Exchange 2010 Management Tools.

Exchange Management Tools must be installed on the SAM server and any additional poller that will poll using PowerShell scripts. Any snap-ins that PowerShell may require must also be installed in the same location.

To ease PowerShell plug-in management in a multi-poller environment, you may prefer to assign nodes with PowerShell templates to a single polling engine.

Installation of Exchange Management Tools
To install Exchange Management Tools on your Windows computer you first need to configure the pre-requisite components.

1. Open the Control Panel, click on Programs and then click Turn Windows Features On or Off.
2. Enable the features shown below:

![Turn Windows features on or off window]

- Internet Information Services
- IIS & Windows Server Management
  - IIS Management
  - IIS Management Console
  - IIS Management Scripts and Tools
  - IIS Management Service
- Microsoft Message Queue ( MSMQ) Server

![Optional features]

- Web Receipt and Delivery
- .NET Framework 3.5 SP1
- Microsoft Message Queue (MSMQ) Server
- IIS Management Compatibility
- IIS Management Console
- IIS Management Scripts and Tools
- IIS Management Service
- IIS & Windows Server Management
  - IIS & Windows Server Management Compatibility
  - IIS & Windows Server Management Compatibility
  - IIS Management and IIS 6 configuration compatibility
  - IIS Management Console
  - IIS Management Scripts and Tools
3. Download the Exchange Server 2010 SP1 installation files and extract them to a temporary folder on your computer.

4. From that folder launch Setup.exe. (If your computer is missing the .NET Framework, visit Microsoft.com to download and install it. Additionally, Steps 1 and 2 of the Exchange Server 2010 SP1 installation will prompt you to install pre-requisites if they are missing, as shown below.)

5. If needed, install the pre-requisites for Exchange Server 2010 SP1. If not, click on Step 3 from the install screen and choose Install only languages from the DVD.

6. Choose language options for installing Exchange Server 2010 SP1 on Windows

7. Next, click on Step 4 from the install screen to begin the installation.
9. Click **Next** at the introduction page, then **accept the license agreement**.
10. Click **Next**, then choose your preference for **Error Reporting**.
11. Click **Next**.
12. At the **Installation Type** page, select **Custom Exchange Server Installation**, and also check the box to **Automatically install Windows Server roles and features required for Exchange Server**.
13. Click **Next**.
14. Select the **Management Tools** role and then click **Next**.

![Exchange Server 2010 Setup](image)

15. When the **Readiness Checks** have completed successfully, click **Install**.

![Exchange Server 2010 Setup](image)

After the install has completed, you can launch the Exchange Management Console from the **Start > All Programs > Microsoft Exchange Server 2010** menu.
64-bit vs. 32-bit

Using the 64-bit (x64) or 32-bit (x86) version of PowerShell depends on how your target server and applications are configured. Ideally, both machines should be running on the same platform to get the most out of PowerShell.

**Platform Scenarios**

For the most part, you will not encounter any differences between the 64-bit and 32-bit versions of PowerShell; however, each version of the shell, in some cases, can only load matching snap-ins. This means you should use the correct 64-bit or 32-bit version of any snap-ins you plan to use. Below are some guidelines to help you determine which version of PowerShell you should use for your particular monitoring environment:

- If SAM is installed on a 64-bit sever and polling a 64-bit machine, you should poll using the 64-bit version of PowerShell.
- If SAM is installed on a 64-bit sever and polling a 32-bit machine, you should poll using the 32-bit version of PowerShell.
- If SAM is installed on a 32-bit sever and polling a 32-bit machine, you should poll using the 32-bit version of PowerShell.
- If SAM is installed on a 32-bit server and polling a 64-bit machine, you may encounter some errors. You should install SAM on a 64-bit server if you need to poll 64-bit machines.

**Note:** There may be instances where installed software on a 64-bit machine may have added its own cmdlets into the 32-bit version of PowerShell. If this is the case, you should use 32-bit polling.

On Windows machines, you can determine the platform you are using by right-clicking **My Computer** and clicking **Properties**.

**Troubleshooting PowerShell Snap-In Compatibility**

The most common issue with PowerShell is the incompatibility of the snap-ins with the platform of PowerShell being used; meaning both PowerShell and its snap-ins should both be either 64-bit or 32-bit. When running your script in SAM, typical errors of this type resemble the following:

```
Script Output: ❌ Add-PSSnapin : No snapins have been registered for Windows PowerShell version 2.
At line1 char13
+ FullyQualifiedErrorId : AddPSSnapinRead,Microsoft.PowerShell.Commands.AddPSSnapinCommand
```

However, no evidence of a snap-in problem exists in PowerShell itself:

```
Name : Microsoft.WSMAN.Management
Version : 2.0
Description : This Windows PowerShell snap-in contains cmdlets (such as Get-WSManInstance and Set-WSManInstance) that are used by the Windows PowerShell host to manage WSMAN operations.
```

In order to diagnose and correct this problem, verify that the snap-ins are properly registered.

1. Find the following registry path:

   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\PowerShellSnapIns
Notice the ApplicationBase path is "C:\Program Files\Microsoft\Exchange Server\bin" not "C:\Program Files (x86)". The "C:\Program Files (x86)" path would suggest 32-bit PowerShell extensions are registered.

2. Open PowerShell though the GUI and check to see if the process is running in 32-bit mode. This is denoted by the "*32" next to the process name. Since this is a 64-bit server with the 64-bit Exchange Management installed, PowerShell should not have a "*32" next to the process name. It should look similar to the image below.

3. Open your 64-bit PowerShell session and run the following:

4. Run `get-PSSnapin -registered`. You should see the snap-in listed as shown below:

```
PS C:\Users\labuser> get-PSSnapin -registered
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

5. Now you should be able to execute a PowerShell command such as `Get-MailboxDatabase -Server {exchange server hostname}` and have it return results similar to the following:

```
PS C:\Users\labuser> Get-MailboxDatabase -Server lab-exc-2007-02
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Server</th>
<th>StorageGroup</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mailbox Database</td>
<td>LAB-Exc-2007-02</td>
<td>First Storage Group</td>
<td>False</td>
</tr>
</tbody>
</table>

6. Your snap-in module has properly been registered. SAM templates that use PowerShell will now work properly.
PowerShell Templates and Monitors

Many SAM templates contain component monitors that allow for the use of PowerShell scripts. An easy way to find a list of these templates is to navigate to the Manage Application Monitor Templates page and search for the word, “PowerShell”. This can be done from the SAM web console by navigating to Settings > SAM Settings > Manage Templates. The search text box is at the top-right of the screen.

Below is a sample list of the templates found when "PowerShell" is searched. To examine and edit a template, check the box next to the template name and then click Edit at the top of the list.

In this example, the Exchange 2007-2010 Mailbox Send and Receive Statistics with PowerShell template is used. This template tracks Exchange Mailbox Send/Receive statistics of Exchange 2007-2010 servers with the Mailbox role using PowerShell scripts.

The following screen appears once you have selected a template to edit, revealing the individual component monitors as well as details about the template:
The following documentation accompanies this template:

**Prerequisites:**

- PowerShell 2.0 and Exchange Management Tools 2007 or 2010 installed on the SAM server.
- The Exchange server must have an Exchange Mailbox role.
- The SAM server and the Exchange server must be in the same domain.

**Credentials:**

The credentials must be an Exchange Administrator (Organization Manager) account with at least view-only permissions.

**Note:** Before using this template, under the Advanced tree collapse [+], you should set the correct platform; either 32-bit or 64-bit, from the dropdown menu. The default is set to 32-bit.

**For all PowerShell component monitors:** You must specify the correct name of your Exchange user and server in the Script Arguments field of the corresponding PowerShell Monitor. If you fail to do this, the counter will return with an error of “Undefined” status.

**For example:** If the name of your Exchange server is exchange.mydomain.com, and the user you want to monitor is some.user@domain.sw, the value in the Script Arguments field should be the following: some.user@domain.sw,exchange.mydomain.com.

To see the names of your Exchange servers, run the following PowerShell command in the Exchange Management Shell: `Get-ExchangeServer`

To see the names of the users, run the following PowerShell command in Exchange Management Shell: `Get-Mailbox`

To examine and edit an individual PowerShell component monitor within the template, click the plus sign [+ ] to the left of the monitor. **For example:** *Number of items received by specific user during last month.*
The following details about the selected component monitor are revealed:

Using a PowerShell script, the monitor in this example is designed to return the number of items received by a specific user during the last month.

In order to use this monitor, you will need to change the Script Arguments field from the default example of, `user@domain.sw,server.domain.sw` to something that will suit your needs for your particular environment. You can do this by clicking the Edit button (highlighted above). You also have the ability to alter the pre-defined script that comes with PowerShell component monitors.

**Note:** Unless otherwise directed by the documentation, you should not need to edit pre-defined scripts.
Once you have changed the **Script Arguments** field, click **Submit** to begin using the component monitor within the template. The output for this script using the SAM monitor, *Number of items received by specific user during last month*, should be similar to the following illustration:

<table>
<thead>
<tr>
<th>COMPONENT NAME</th>
<th>STATISTIC</th>
<th>MESSAGE</th>
<th>RESPONSE TIME</th>
<th>PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of items received by specific user during last month</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Statistic</strong></td>
<td>9356</td>
<td>User <a href="mailto:Joe.Blow@Kokomo.com">Joe.Blow@Kokomo.com</a> received: 9356 items during last month</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The output for the script using only PowerShell should be similar to the following illustration:
PowerShell Code with SAM

*SolarWinds does not provide customer support for any scripting language. For scripting support with SAM, the SolarWinds public community on [thwack.com](http://thwack.com) may be of assistance.*

**Note:** Unless otherwise directed by the documentation, you should not need to edit pre-defined scripts.

To edit the default script for the *Number of items received by specific user during last month* monitor, click the **Edit** button for that monitor. Below is the default PowerShell script for this component monitor:

**Note:** Each statistic and message output pair of your script requires a unique identifier. A maximum of 10 output pairs can be monitored per script.

```powershell
$ErrorActionPreference = "silentlycontinue";
$address = $args.get(0);
$server = $args.get(1);
$Error.Clear();
if ( !$address )
{
    Write-Host "Message: Can't find "user_mailbox" argument. Check documentation.";
    exit 1;
}
if ( !$server )
{
    Write-Host "Message: Can't find "server" argument. Check documentation.";
    exit 1;
}
$t1 = Get-Date;
$t2 = $t1.AddMonths(-1);
$stat = (Get-MessageTrackingLog -Server $server -Recipients $address -EventID "Receive" -ResultSize "Unlimited" -Start $t2 -End $t1 | Measure-Object).Count;
if ($Error.Count -eq 0) {
    Write-Host "Message: User $address received: $stat items during last month";
    Write-Host "Statistic: $stat";
    Exit 0;
}
Write-Host "Message: $($Error[0])";
Exit 1;
```
Variables are used for storing information. In SAM, variables are prefixed with "$", as highlighted below. The following code snippet from the above code calculates a numerical value and then stores it in the variable $stat. In the illustration below, the variable's value is reported as 9356, as highlighted in the Statistic column's output.

```
$stat = (Get-MessageTrackingLog -Server $server -Recipients $address -EventID "Receive" -ResultSize "Unlimited" -Start $t2 -End $t1 | Measure-Object).Count;
```

Text and variables within quotes indicate information that may be visible to the user. When made visible, the variables in the message will be replaced with the values the variables store. The variables are highlighted below in both the code and the output:

```
Write-Host "Message: User $address received: $stat items during last month";
Write-Host "Statistic: $stat";
```

With these same lines of code, Message: and Statistic: refer to the columns where the information will be placed:

```
Write-Host "Message: User $address received: $stat items during last month";
Write-Host "Statistic: $stat";
```
**Scripts Must Report Status Through Exit Codes**

Scripts must report their status by exiting with the appropriate exit code. The exit code is used to report the status of the monitor, which is seen by the user through the interface. The following table explains the exit codes and their values:

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Up</td>
</tr>
<tr>
<td>1</td>
<td>Down</td>
</tr>
<tr>
<td>2</td>
<td>Warning</td>
</tr>
<tr>
<td>3</td>
<td>Critical</td>
</tr>
<tr>
<td>Any other value</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

The following code snippet shows proper usage of exit codes.

```powershell
if ($Error.Count -eq 0) {
    Write-Host "Message: User $address received: $stat items during last month";
    Write-Host "Statistic: $stat";
    Exit 0;
}
Write-Host "Message: @{$Error[0]}";
Exit 1;
```

The two exit codes in this example are conditional, meaning either one or the other will be triggered based on a certain outcome of the code provided. When `Exit 0;` (status of Up) is reported, the message and statistic are displayed and the monitor shows a status of Up. When `Exit 1;` (status of Down) is reported, the message and statistic are not displayed and the a status of Down is reported.

If you want to inform SolarWinds SAM that a PowerShell script reports an Up status, you would exit the script using `Exit 0;`.

**Scripts with Text Output**

Scripts report additional details by sending text to the script’s standard output. SAM supports multiple values returned by a script using the following format. There is a limit of 10 Statistic and Message pairs for the script. These can be placed anywhere in the script output. The Statistic and Message names you give must contain valid letters and/or numbers.

<table>
<thead>
<tr>
<th>Detail Type</th>
<th>Required</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
<td>Yes</td>
<td>A numeric value used to determine how the monitor compares to its set thresholds. This must be an integer value, (negative numbers are supported).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Statistic.Name1: 123</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Statistic.Name2: 456</td>
</tr>
<tr>
<td>Message</td>
<td>No</td>
<td>An error or information message to be displayed in the monitor status details. Note: Multi-line messages are supported. To use this functionality, print each line using a separate command. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Message.Name1: abc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Message.Name2: def</td>
</tr>
</tbody>
</table>
Remote Execution vs. Local Execution

When you select remote execution, SAM creates a remote session via WinRM on the target server. The script is then copied to the target server and run there.

When you select local execution, a local version of the PowerShell console will open. If you need to run specific non-default cmdlets for the target server, you should have the needed cmdlets installed on the SAM machine.

The following script can be used inside of SAM (local/remote execution modes) and outside of SAM in the PowerShell console. The script returns 0 and the Hostname of the SAM machine (SAM local execution) or the Hostname of the target machine (SAM remote execution). In the Windows PowerShell console, the script returns the local machine Hostname. If for some reason the script cannot get the hostname, it returns 1 and a message of "Host not found."

Script:

```powershell
$stat = $env:computername;
if ($stat -ne $null)
    {
        Write-Host "Statistic: 0";
        Write-Host "Message: $stat";
    }
else
    {
        Write-Host "Statistic: 1";
        Write-Host "Message: Host not found";
    }
```

To save this script as a PowerShell (ps1) file:

In Notepad, save the script as Get-Date.ps1 (You can rename an existing file from *.txt to *.ps1.)

To open the PowerShell console:

For Windows x64:

- For x64, navigate to Start > Accessories > Windows PowerShell > Windows PowerShell
- For x86, navigate to: Start > Accessories > Windows PowerShell > Windows PowerShell (x86)

For Windows x86:

- Navigate to Start > Accessories > Windows PowerShell > Windows PowerShell

To run this command via remote execution:

`invoke-command -ComputerName SOME_PC -Credential SOME_PC\SOME_USER -ScriptBlock { Get-Date }`

To run this command via local execution:

Name the file Get-Date.ps1 and run it by double-clicking it.
```powershell
$stat = $env:computername;
if ($stat -ne $null)
{
    Write-Host "Statistic: 0";
    Write-Host "Message: $stat";
}
else
{
    Write-Host "Statistic: 1";
    Write-Host "Message: Host not found";
}
```
Execute Scripts Remotely via Secure WinRM

Note: The steps outlined below are only for advanced users. Following the steps outlined on page 4 should be sufficient for most users and environments.

Aside from new remoting specific cmdlets with enhanced capabilities, PowerShell 2.0 also ships with full remoting functionality. It is possible to connect your local PowerShell session to a remote computer and execute commands just as if you were sitting in front of the server console. The technology to make this happen relies on WinRM 2.0, which is Microsoft’s latest implementation of the WS-Management Protocol, a SOAP-based protocol used to manage a variety of hardware devices. The theory behind this is that it will provide a shared way for differing systems to communicate with each other.

WinRM 2.0 communicates via HTTP, and so is likely to be firewall-friendly; it also listens on ports 5985 (default) and 5986 (encrypted), avoiding issues with locally installed IIS. Even though it uses HTTP for communication, security has still been considered; either NTLM or Kerberos are used for authentication, and if you wish to configure WinRM 2.0 to use SSL, that is possible too. A lot of the configuration can be carried out via new PowerShell cmdlets shipped with version 2.0.

Since PowerShell 2.0 and WinRM 2.0 go hand in hand, Microsoft has bundled them up together along with a new version of BITS (Background Intelligent Transfer Service) 4.0, making a single package known as the Windows Management Framework. Although this collection makes some sense if you know the background, there can be some confusion if an administrator is searching for the download of PowerShell 2.0, ends up at the homepage for the Windows Management Framework (http://support.microsoft.com/kb/968929) and wonders what the heck that is. Well, now you know!

The components of the Windows Management Framework are already installed with both Windows Server 2008 R2 and Windows 7, although WinRM 2.0 is not enabled by default on Windows 7. The components have also been made available for older OS versions, and you can download all of those from the homepage of the Windows Management Framework. Essentially, it is available for the various flavors of Windows Server 2008 and 2003, as well as the Windows Vista and XP client operating systems – although BITS 4.0 is not available for Windows Server 2003 or XP. This might sound like a muddle, but in short, it is possible to run remote PowerShell 2.0 sessions both to and from all these different operating systems.

PowerShell 2.0 Remoting Requirements

To enable PowerShell remoting, all computers participating in remote management should have the following software:

- Windows PowerShell 2.0
- NET framework 2.0 SP1 or later
- Windows Remote Management (WinRM) 2.0

All of the above are installed by default on Windows 7 and Windows Server 2008 R2. However, earlier versions of Windows will require you to download the updates from Microsoft website and install them yourself.


To be able run scripts and commands on remote computers, the user performing remote script execution must be:

- a member of the administrators group on the remote machine or
- should be able to provide administrator credentials at the time of remote execution or
• should have access the PS session configuration on the remote system

Also, on client OS versions of Windows such as Windows Vista and Windows 7, network location must be set either to Home or Work. WS-Management may not function properly if the network location for any of the network adapters is set to public.

**PowerShell 2.0 Remoting Configuration**

To get PowerShell remoting working we need to configure **Remoting Client** and **Remoting Server**. In this document **Remoting Client** is a SAM box (including additional pollers if there are any); The **Remoting Server** is any target node, which we are going to monitor with the use of the PowerShell monitor configured to execute scripts remotely via WinRM. As an example, we will discuss the remoting configuration procedure against Windows Server 2008 R2.

**Note:** Some commands (or steps) may be different if Windows Server 2003 is used.

**Configuration of the Remoting Server**

To turn your computer in to a **Remoting Server** that is accessible from other machines, the prerequisites for Remoting Server need to be in place:

• **SSL Certificate:** This is required if we are going to secure our WinRM connection.

• **Listener:** Inside WinRM, a listener needs to be set up that listens on the network port Windows PowerShell the Remoting Server uses to communicate.

• **Firewall Exception:** A firewall exception is needed that allows outside requests to reach the WinRM service;

• **WinRM Service:** This service receives requests from other computers and needs to be running.

**Create a Self-signed Certificate**

You will need an SSL certificate to use a secure WinRM connection. WinRM HTTPS requires a local computer Server Authentication Certificate with a CN matching the IP address that is not expired or revoked to be installed.

There are two well-known tools available to create self-signed certificates; **MakeCert.exe** and **SelfSSL.exe**. MakeCert.exe is for testing purposes only and comes with Visual Studio. SelfSSL.exe is a part of the **Internet Information Services (IIS) 6.0 Resource Kit Tools**. SelfSSL.exe will be used in this example. The following commands should be executed on the Remoting Server computer. This is the target node to be used with the SAM PowerShell monitor.
1. Download Internet Information Services (IIS) 6.0 Resource Kit and start setup:

2. Choose the Custom Setup option:
3. Select only the SelfSSL.exe tool (if you have no need for the additional components):

4. Open a Command Prompt as an administrator from the Start menu, (right-click):

5. Change the current location on SelfSSL install path. Typically the path is "C:\Program Files (x86)\IIS Resources\SelfSSL":
6. Enter the following command to create a self-signed certificate. Replace the parameters with actual values, as explained below. **Note:** Ignore the following possible error message: *Error opening metabase: 0x80040154.* This indicates that IIS 6.0 compatibility mode may not be installed.

```
selfssl.exe /N:CN=<Local Server IP Address> /V:<Certificate time to live in days> /P:<WinRM listener port> /T /Q
```

- `<Local Server IP Address>` – This is the IP address of the **Remoting Server** node. Use the IP address and not the computer name. SAM uses this IP address when the probes are run;

- `<Certificate time to live in days>` – This is the time interval, in days, for which the certificate remains valid;

- `<WinRM listener port>` – This is the port on which the HTTPS listener will be created. The default value for the WinRM HTTPS listener port is **5986**;

- `/T` – This option adds the self-signed certificate to the **Trusted Root Certificates** list;

- `/Q` – **Quiet mode.** This will prevent you from be prompted when SSL settings are overwritten.

**For example:**

```
selfssl.exe /N:CN=192.168.0.198 /V:3600 /P:5986 /T /Q
```
26 Configuring and Integrating PowerShell

7. Verify that certificate was created properly. Start the Management Console (MMC.exe):

8. Add the Certificates snap-in and verify that the recently created self-signed certificate is listed in both the “Personal” and “Trusted Root Certificate Authorities” storages.
9. Follow the red steps in the graphic, 1 through 4, to select the Certificates: snap-in

10. Once complete, you should have the following:
11. Verify that the certificate is in the *Personal* storage:

   ![Certificate Management Console](image1)

12. Verify that the certificate is in *Trusted Root Certificate Authorities* storage:

   ![Certificate Management Console](image2)

13. Open the created certificate by double-clicking it.

14. On the Details page, select and copy the Thumbprint field value:

   ![Certificate Details](image3)

15. Copy the values highlighted above to the clipboard. These copied values will be used in creating a *Listener* in the following section.
Create a WinRM HTTPS Listener

Create the Windows Remote Management Listener and bind it to the certificate using the following steps:

1. Open a Command Prompt as an administrator from the Start menu, (right-click):

2. The following command should be executed on the Remoting Server computer. This is the target node which will be used with the SAM PowerShell monitor. It is also the computer where the self-signed certificate was created in the previous section.

3. Enter the following command with the parameters replaced with actual values, as explained below:

   ```powershell
   winrm create winrm/config/Listener?Address=<IP Address used to bind listener>+Transport=HTTPS @{Hostname="<The name or IP of your remoting server>";CertificateThumbprint="<Paste from the previous step and remove the spaces>";Port="<Port number>"}
   ```

   - `<IP Address used to bind listener>` - To bind the certificate to the **Listener**, specify the Remoting Server's local IP address. You can use the wildcard, (*), symbol to allow listening on all available local addresses;

   - `<The name or IP of your remoting server>` - The Remoting Server's node name or IP address;

   - `<Paste from the previous step and remove the spaces>` - Paste the self-signed certificate thumbprint created in steps 13-14 of the previous section;

   - `<Port number>` - This is the port number for the **Listener**. You can specify the default WinRM HTTPS port of 5986.

   For example:

   ```powershell
   winrm create winrm/config/Listener?Address=IP:192.168.0.171+Transport=HTTPS @{Hostname="192.168.0.198";CertificateThumbprint="6aa47ed7356fb0f1e3b434850a7bb51ed40b0d3a";Port="5986"}
   ```
4. Once the command has been successfully executed, the output will look similar to the following illustration:

![Command Prompt output](image)

**Adding a Firewall Exception**

The following steps will create an in-bound exception for the Windows firewall using WinRM HTTPS port 5986.

1. Open a Command Prompt as an administrator from the **Start** menu, (right-click):

![Command Prompt](image)

2. Enter the following command with the parameters replaced with actual values, as explained below:

   ```
   netsh advfirewall firewall add rule name="<Rule name>" protocol=TCP dir=in localport=<Port number> action=allow
   ```

   - `<Rule name>` - This is the name of the rule shown in the Windows Firewall under **Advanced Security > Inbound Rules**;

   - `<Port number>` - This is the port number in use for the **Listener** that was created in the previous section.

   **For example:**

   ```
   netsh advfirewall firewall add rule name="WinRM via HTTPS - Open Port 5986" protocol=TCP dir=in localport=5986 action=allow
   ```
Remoting Client/SAM Computer Configuration

Import the self-signed certificate from the Remoting Server with the following steps:

1. On the Remoting Server, open the management console;

2. Add the Certificates snap-in;

3. Locate the created certificate and right-click on it then select All Tasks > Export…

4. The Certificate Export Wizard will be launched:

5. Select, No, do not export the private key, the click Next.
6. Select, DER binary X.509 (.CER), and then click **Next**.
7. Specify the file, and then click **Next**.

8. Click **Finish**.
Transferring the Certificate

1. Now you need to transfer the certificate. On the **Remoteing Client**, open the Management Console.

2. Add the Certificates snap-in.

3. Follow the red steps in the graphic, 1 through 4, to select the Certificates snap-in:
4. Click **OK** when done.

5. Locate **Trusted Root Certification Authorities**, right click on it, then navigate to **All Tasks > Import**.
6. The Certificate Import Wizard will be launched. Find the file to import and then click **Next**.

![Certificate Import Wizard](image1)

7. Select, **Place all Certificates in the following Store**. The store is "Trusted Root Certification Authorities."

![Certificate Import Wizard](image2)
8. Complete the wizard by clicking **Finish**.

9. Verify that certificate was imported successfully:
Allowing the Remoting Client to Communicate with any Remoting Host

If your machine is only acting as a client in a domain environment, meaning it is accessing other systems but not being accessed itself, you do not need to configure anything out of the ordinary.

The following minimum setup will only allow access to other systems that have enabled remoting. You will not be able to test remoting on your own computer.

To set up TrustedHosts without calling Enable-PSRemoting, you need to add a registry key and then temporarily run the WinRM service. The following steps address will set up TrustedHosts:

1. Run a Windows PowerShell console as an administrator (right-click):

2. Execute the following commands:

   ```powershell
   PS> Set-ItemProperty HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System LocalAccountTokenFilterPolicy 1 -Type DWord
   PS> Start-Service WinRM
   ```

3. Add the TrustedHosts entry:

   ```powershell
   PS> Set-Item WSMAN:\localhost\Client\TrustedHosts -Value * -Force
   ```

4. Turn off the WinRM service and revert the value of the LocalAccountTokenFilterPolicy registry entry:

   ```powershell
   PS> Stop-Service WinRM
   PS> Set-ItemProperty HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System LocalAccountTokenFilterPolicy 0 -Type DWord
   ```

Your client is now able to access other systems remotely using Windows PowerShell remoting and is no longer limited to Kerberos and machines inside a domain.
Setup Windows PowerShell monitor in SAM
Create and test the SAM PowerShell monitor using the following steps:

1. From the SAM web console, navigate to: Settings > SAM Settings > Manage Templates.
2. Search for "PowerShell."
3. Select a template by checking the box next to the template name, and then clicking Edit.
4. Click the [+] to the left of a monitor name to reveal its details:
5. You must choose Remote Host as the execution mode.
6. Check the Use HTTPS Protocol setting.
7. The Port Number value, (the default value is 5986 for WinRM secure connection), should correlate with the port on which the Listener was created on the Remoting Server.

8. Enter a simple script in the monitor for testing by clicking Edit, for example:

   ```powershell
   write "Statistic.RemoteMachineName: 0";
   write "Message.RemoteMachineName: $env:computername";
   exit 0;
   ```

   **Note:** This script is printing the computer name (the machine on which this is executed).
9. Define the output columns for the script:

10. View the script output:
11. Create an application based on our template and assign it to the **Remoting Server** target node:

![Application Details](image)

12. Examine the script output. Done correctly, you should see the name of the **Remote Server** node in the output, not the SAM server, (i.e. the script was executed remotely).

   You now have a PowerShell monitor configured to execute scripts remotely via secure WinRM connection.
Additional Information

