PowerShell remoting over SSH

09/30/20195 minutes to read

In this article

Overview

PowerShell remoting normally uses WinRM for connection negotiation and data transport. SSH is now available for Linux and Windows platforms and allows true multiplatform PowerShell remoting.

WinRM provides a robust hosting model for PowerShell remote sessions. SSH-based remoting doesn't currently support remote endpoint configuration and Just Enough Administration (JEA).

SSH remoting lets you do basic PowerShell session remoting between Windows and Linux computers. SSH remoting creates a PowerShell host process on the target computer as an SSH subsystem. Eventually we'll implement a general hosting model, similar to WinRM, to support endpoint configuration and JEA.

The `New-PSSession`, `Enter-PSSession`, and `Invoke-Command` cmdlets now have a new parameter set to support this new remoting connection.

Copy

```
[-HostName <string>]  [-UserName <string>]
[-KeyFilePath <string>]
```

To create a remote session, you specify the target computer with the `HostName` parameter and provide the user name with `UserName`. When running the cmdlets interactively, you're prompted for a password. You can also, use SSH key authentication using a private key file with the `KeyFilePath` parameter.

General setup information

PowerShell 6 or higher, and SSH must be installed on all computers. Install both the SSH client (`ssh.exe`) and server (`sshd.exe`) so that you can remote to and from the computers. OpenSSH for Windows is now available in Windows 10 build 1809 and Windows Server 2019. For more information, see Manage Windows with OpenSSH. For Linux, install SSH, including sshd server, that's appropriate for your platform. You also need to install PowerShell from GitHub to get the SSH remoting feature. The SSH server must be configured to create an SSH subsystem to host a PowerShell process on the remote computer. And, you must enable **password** or **key-based**authentication.

Set up on a Windows computer

1. Install the latest version of PowerShell, see Installing PowerShell Core on Windows.

   You can confirm that PowerShell has SSH remoting support by listing the `New-PSSession` parameter sets. You'll notice there are parameter set names that begin with **SSH**. Those parameter sets include **SSH** parameters.

   PowerShell

   Copy

   ```
   (Get-Command New-PSSession).ParameterSets.Name
   ```

   Output

   Copy

   ```
   Name
   ----
   SSHHost
   SSHHostHashParam
   ```

2. Install the latest Win32 OpenSSH. For installation instructions, see Getting started with OpenSSH.

   Note

   If you want to set PowerShell as the default shell for OpenSSH, see **Configuring Windows for OpenSSH**.

3. Edit the `sshd_config` file located at `$env:ProgramData\ssh`.

   Make sure password authentication is enabled:

   Copy

```
PasswordAuthentication yes
```

Create the SSH subsystem that hosts a PowerShell process on the remote computer:

Copy

```
Subsystem powershell c:/progra~1/power-
shell/6/pwsh.exe -sshs -NoLogo -NoProfile
```

Note

You must use the 8.3 short name for any file paths that contain spaces. There's a bug in OpenSSH for Windows that prevents spaces from working in subsystem executable paths. For more information, see this **GitHub issue**.

The 8.3 short name for the `Program Files` folder in Windows is usually `Progra~1`. However, you can use the following command to make sure:

PowerShell

Copy

```
Get-CimInstance Win32_Directory -Fil-
ter 'Name="C:\\Program Files"' |
  Select-Object EightDotThreeFileName
```

Output

Copy

```
EightDotThreeFileName
---------------------
c:\progra~1
```

Optionally, enable key authentication:

Copy

```
PubkeyAuthentication yes
```

For more information, see Managing OpenSSH Keys.

4. Restart the **sshd** service.

   PowerShell

   Copy

   ```
   Restart-Service sshd
   ```

5. Add the path where OpenSSH is installed to your Path environment variable. For example, `C:\Program Files\OpenSSH\`. This entry allows for the `ssh.exe` to be found.

Set up on an Ubuntu 16.04 Linux computer

1. Install the latest version of PowerShell, seeInstalling PowerShell Core on Linux.

2. Install Ubuntu OpenSSH Server.

   bash

   Copy

   ```
   sudo apt install openssh-client
   sudo apt install openssh-server
   ```

3. Edit the `sshd_config` file at location `/etc/ssh`.

   Make sure password authentication is enabled:

   Copy

   ```
   PasswordAuthentication yes
   ```

   Add a PowerShell subsystem entry:

   Copy

   ```
   Subsystem powershell /usr/bin/pwsh -sshs
   -NoLogo -NoProfile
   ```

   Optionally, enable key authentication:

   Copy

   ```
   PubkeyAuthentication yes
   ```

4. Restart the **sshd** service.

   bash

   Copy

   ```
   sudo service sshd restart
   ```

Set up on a macOS computer

1. Install the latest version of PowerShell, see Installing PowerShell Core on macOS.

Make sure SSH Remoting is enabled by following these steps:

a. Open `System Preferences`.
b. Click on `Sharing`.
c. Check `Remote Login` to set `Remote Login: On`.
d. Allow access to the appropriate users.

2. Edit the `sshd_config` file at location `/private/etc/ssh/sshd_config`.

   Use a text editor such as **nano**:

   bash

   Copy

   ```
   sudo nano /private/etc/ssh/sshd_config
   ```

   Make sure password authentication is enabled:

   Copy

   ```
   PasswordAuthentication yes
   ```

   Add a PowerShell subsystem entry:

   Copy

   ```
   Subsystem powershell /usr/local/bin/pwsh
   -sshs -NoLogo -NoProfile
   ```

   Optionally, enable key authentication:

   Copy

   ```
   PubkeyAuthentication yes
   ```

3. Restart the **sshd** service.

bash

Copy

```
sudo launchctl stop com.openssh.sshd
sudo launchctl start com.openssh.sshd
```

Authentication

PowerShell remoting over SSH relies on the authentication exchange between the SSH client and SSH service and doesn't implement any authentication schemes itself. The result is that any configured authentication schemes including multi-factor authentication are handled by SSH and independent of PowerShell. For example, you can configure the SSH service to require public key authentication and a one-time password for added security. Configuration of multi-factor authentication is outside the scope of this documentation. Refer to documentation for SSH on how to correctly configure multi-factor authentication and validate it works outside of PowerShell before attempting to use it with PowerShell remoting.

PowerShell remoting example

The easiest way to test remoting is to try it on a single computer. In this example, we create a remote session back to the same Linux computer. We're using PowerShell cmdlets interactively so we see prompts from SSH asking to verify the host computer and prompting for a password. You can do the same thing on a Windows computer to ensure remoting is working. Then, remote between computers by changing the host name.

PowerShell

Copy

```
#
# Linux to Linux
#
$session = New-PSSession -HostName UbuntuVM1
-UserName TestUser
```

Output

Copy

```
The authenticity of host 'UbuntuVM1
(9.129.17.107)' cannot be established.
ECDSA key fingerprint is SHA256:2kCbnhT2d-
UE6WCGgVJ8Hyfu1z2wE4lifaJXLO7QJy0Y.
Are you sure you want to continue connecting
(yes/no)?
TestUser@UbuntuVM1s password:
```

PowerShell

Copy

```
$session
```

Output

Copy

```
 Id Name   ComputerName    ComputerType
State    ConfigurationName     Availability
 -- ----   ------------    ------------    --
---    -----------------     ------------
  1 SSH1   UbuntuVM1       RemoteMachine
Opened   DefaultShell             Available
```

PowerShell

Copy

```
Enter-PSSession $session
```

Output

Copy

```
[UbuntuVM1]: PS /home/TestUser> uname -a
Linux TestUser-UbuntuVM1 4.2.0-42-generic
49~16.04.1-Ubuntu SMP Wed Jun 29 20:22:11 UTC
2016 x86_64 x86_64 x86_64 GNU/Linux

[UbuntuVM1]: PS /home/TestUser> Exit-
PSSession
```

PowerShell

Copy

```
Invoke-Command $session -ScriptBlock { Get-
Process powershell }
```

Output

Copy

```
Handles  NPM(K)    PM(K)      WS(K)
CPU(s)     Id  SI ProcessName
PSComputerName
-------  ------    -----      -----     -----
-     --  -- -----------                  -
-------------
      0       0        0         19
3.23  10635 635 powershell
UbuntuVM1
      0       0        0         21
4.92  11033 017 powershell
UbuntuVM1
      0       0        0         20
3.07  11076 076 powershell
UbuntuVM1
```

PowerShell

Copy

```
#
# Linux to Windows
#
Enter-PSSession -HostName WinVM1 -UserName
PTestName
```

Output

Copy

```
PTestName@WinVM1s password:
```

PowerShell

Copy

```
[WinVM1]: PS C:\Users\PTestName\Documents>
cmd /c ver
```

Output

Copy

```
Microsoft Windows [Version 10.0.10586]
```

PowerShell

Copy

```
#
# Windows to Windows
#
C:\Users\PSUser\Documents>pwsh.exe
```

Output

Copy

```
PowerShell
Copyright (c) Microsoft Corporation. All
rights reserved.
```

PowerShell

Copy

```
$session = New-PSSession -HostName WinVM2 -
UserName PSRemoteUser
```

Output

Copy

```
The authenticity of host 'WinVM2
(10.13.37.3)' can't be established.
ECDSA key fingerprint is SHA256:kSU6slAROy-
QVMEynVIXAdxSiZpwDBigpAF/TXjjWjmw.
Are you sure you want to continue connecting
(yes/no)?
Warning: Permanently added 'Win-
VM2,10.13.37.3' (ECDSA) to the list of known
hosts.
PSRemoteUser@WinVM2's password:
```

PowerShell

Copy

```
$session
```

Output

Copy

```
 Id Name            ComputerName    Computer-
Type    State         ConfigurationName
Availability
 -- ----            ------------    ---------
```

```
---    -----          -----------------     --
----------
  1 SSH1            WinVM2          RemoteMa-
chine   Opened        DefaultShell
Available
```

PowerShell

Copy

```
Enter-PSSession -Session $session
```

Output

Copy

```
[WinVM2]: PS C:\Users\PSRemoteUser\Documents>
$PSVersionTable

Name                           Value
----                           -----
PSEdition                      Core
PSCompatibleVersions           {1.0, 2.0,
3.0, 4.0...}
SerializationVersion           1.1.0.1
BuildVersion                   3.0.0.0
CLRVersion
PSVersion                      6.0.0-alpha
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
GitCommitId                    v6.0.0-
alpha.17


[WinVM2]: PS C:\Users\PSRemoteUser\Documents>
```

Known issues

The **sudo** command doesn't work in a remote session to a Linux computer.

See also

Installing PowerShell Core on Linux

Installing PowerShell Core on macOS

Installing PowerShell Core on Windows

Manage Windows with OpenSSH

Managing OpenSSH Keys

Ubuntu SSH