PowerShell Remoting

Note: For the “PowerShell Remoting” and “Just Enough Admin (JEA)” labs you will need a second lab virtual machine. You can create the second virtual machine by selection option 2 during the setup described [here](https://onedrive.live.com/view.aspx?resid=5EFC811CDEC9D7F0!9867&ithint=file%2cdocx&authkey=!AFmEtJQN-wdNXmA).

PowerShell remoting makes executing commands on remote systems easy. On Windows, PowerShell remoting uses the WinRM protocol and always encrypts the messages sent across the network. Windows Servers accept PowerShell remoting connections by default while Windows Clients do not. To allow PowerShell remoting connections on Windows Clients you can use the **Enable-PSRemoting** cmdlet.

Because our lab VMs aren’t joined to an Active Directory domain, there is one extra command we must run before we can connect to the Remote VM using PowerShell remoting. The following command adds the remote lab vm to a list of trusted hosts to allow the connection.

Set-Item WSMan:localhost\client\trustedhosts -value "PS4I-Remote" -Force

Enter the command above in an administrative PowerShell prompt on your Main lab VM.



Now let’s create a PowerShell Remoting session and use it interactively.



In the commands above, we created a new session and stored it in a variable called **$sess**. We then “entered” the session making it as if we were sitting in front of a PowerShell session on the Remote VM. Anything we enter from this prompt, such as the hostname command, will be executed on the Remote VM.

Experiment by running some commands in the session. For example, you could make a new file on the desktop called **hello-remote.txt**. Does it show up on your current desktop (the main lab VM) or on the Remote VM where the session is executing?



When you are done playing with the remote session interactively, use the **Exit-PSSession** command to return to your normal (local) PowerShell session.



We can execute any command on the remote machine from a non-interactive prompt as well (one that does not require human interaction or typing). For this we use the **Invoke-Command** cmdlet and specify the session we want it to run in. Here we are printing out the **hello-remote.txt** file we created on the desktop of the remote machine.



This is a very convenient way to configure remote computers without having to make manual connections like a remote desktop connection for example.

In the examples above, we were able to easily create the PowerShell Remoting session without specifying credentials because our current **IEUser** exists on both the local and remote virtual machines and is configured with the same password. If that were not the case, or we preferred to connect as a different user, we can specify alternate credentials to use for the connection.

Use the **Get-Credential** command from the main VM and assign it to a variable called **$cred** so that we can refer to it later. We will use the credentials for the **RemoteMgmtUser** that was added to the remote VM by the lab setup script you ran initially. The password is **Passw0rd!**



Now if we enter a PowerShell remoting session by providing this credential object, we will connect as the **RemoteMgmtUser** instead of our current **IEUser** user.



Any commands executed using this session will be executed from the context of the **RemoteMgmtUser**. If we need a script to be able to use the credentials of this alternate user, we could save the password to disk as encrypted text.



We used the following command to save an encrypted version of the password to a file called **cred.txt**.

$cred.Password | ConvertFrom-SecureString | Out-File cred.txt

The actual clear text password can only be recovered from the file by the user that encrypted it (**IEUser** in this case) and from the same machine where it was encrypted. Even if an attacker can steal the **cred.txt** file, they won’t be able to use it from another computer, or even as a different user on the same computer.

We can “rehydrate” these credentials so that an unattended script can use them as follows.

$secureString = Get-Content cred.txt | ConvertTo-SecureString

$cred = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList RemoteMgmtUser,$secureString



In this case, we were able to read the credentials from the **cred.txt** file without needing to type any username or password interactively, which is useful for unattended scripts. Now we can use our credential variable to execute any commands we would like against one or many remote machines.



Excellent, you’ve been able to learn about how to safely use credentials and easily execute commands on remote computers through the use of PowerShell Remoting.