Name Click here to enter name ID Click here to enter id

Introduction to Remoting

Exercise 3.3

# Instructions

Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

There is a lab network of virtual machines set up for this course, and an account has been created for you to log in and use them. This exercise briefly introduces some of PowerShell’s *remoting* features.

# Requirements

* SSH client (There is already a command-line OpenSSH client available in current versions of Windows, macOS, and Linux)
* Internet access

# Setup

Use SSH as a remote access tool to connect to the lab:  
Computer: **cit361-lab.cit.byui.edu**Port: either **22** or **443** *(Important: for SSH, 443 is a* nonstandard *transport layer port number.)*  
Username: The mailbox portion of your BYU-I email address. It usually consists of three letters and five numbers. (Example: If your BYU-I email is **mer23079@byui.edu** you would use **mer23079** for your username.)  
Password: Your I-Number.

If you are using OpenSSH from a CLI, the parameters above can be specified like this:  
**ssh mer23079@cit361-lab.cit.byui.edu**   
If that doesn’t work (or you don’t get prompted for your password), try the non-standard port:  
**ssh -p 443 mer23079@cit361-lab.cit.byui.edu**   
If you successfully type the correct password when prompted, you will see a prompt like this:  
PS C:\Users\mer23079>

The SSH service of **cit361-lab.cit.byui.edu** is *Internet-facing*, to facilitate remote access from the Internet by students. It connects you to a *bastion host* (or *jump-server*), which means that after you log into it, you can then remotely access additional virtual machines in the lab which are *not* Internet-facing.

# Task 1—Cmdlet Based Remoting

Any cmdlet that has a **-ComputerName** parameter is capable of remoting. In this task you will use cmdlet based remoting to get information from remote hosts.

## Steps

1. After using SSH to log in, launch Windows PowerShell (not PowerShell Core):   
    **powershell**   
   Then get a list of commands that support the -ComputerName parameter:  
    **Get-Command -ParameterName ComputerName**
   1. Pipe the result into the **Measure-Object** cmdlet, to count how many commands support that parameter. How many? Click or tap here to enter text. This list is likely incomplete; it might only show cmdlets from the source named Microsoft.PowerShell.Core.
   2. Let’s execute a simple command that activates another source:   
      **Get-Location**   
      Then use up-arrow to find the **Get-Command -ParameterName ComputerName** command in your command history, and execute it again. Which source provides **-ComputerName** supportive cmdlets that weren’t included in step 1.1? Click or tap here to enter text. (*Hint: by default, Get-Command displays its results in table format. If your terminal window is too narrow, you might not see the* Source *property’s column of the table. Try*  
       **Get-Command -ParameterName ComputerName | Select-Object Name,Source**  
      *to see just those two properties.*)
   3. What are the **-ComputerName** supporting commands that start with the verb **Get**? Figure out a command that lists these for you, then use the **-join** operator to combine them all into one string separated by commas. Copy your result here: Click or tap here to enter text.
2. Notice that the Get-Process cmdlet supports the **-ComputerName** parameter. Use its convenient alias **gps** to count the processes on the lab VM you logged into:   
   gps|Measure
   1. How many processes are running on the *local* lab machine (the bastion host)? Click or tap here to enter text.
3. Now get a list of the processes on a remote machine. To do that, you need to get a PowerShell process with elevated privileges. Enter:  
   **Enter-PSSession localhost -ConfigurationName AdminShell**  
   Then enter:  
   gps -ComputerName DC|Measure
   1. How many processes are running on the lab virtual machine named DC? Click or tap here to enter text.
4. You can also list *services*. (The cmdlet is **Get-Service**, and it too has an alias, **gsv**.)
   1. How many services are running on the local machine? Click or tap here to enter text.
   2. How many services are running on DC? Click or tap here to enter text.
   3. What command line did you use to find the number of services running on DC? Click or tap here to enter text.
5. Exit the privileged PowerShell process and return to your own PowerShell account prompt:  
   **exit**

# Task 2—PowerShell Remoting

When using cmdlet-based remoting, the cmdlet uses whatever mechanism the cmdlet’s developer implemented to perform the remoting. Some of the advantages to cmdlet-based remoting include flexibility and speed. A drawback is that you need to configure whatever underlying remoting technique that each cmdlet uses. Such configuration may include firewall rules, permissions, etc. Also, you can only remote the cmdlets that were specifically designed for remoting.

PowerShell’s **Invoke-Command** cmdlet provides a PowerShell based remoting infrastructure. It allows you to launch a remote PowerShell session and run *any* script on it you wish. The script actually runs locally on the remote machine. It just needs a secure method of remoting, so that it can deliver the script to the target. It does have at least one drawback: **Invoke-Command** must launch a new instance of PowerShell on the remote machine, which is processing overhead that makes script execution take longer on the remote host than it would locally.

## Steps

First, let’s evaluate an expression on a local host, then on a remote machine.

1. The **$Env:COMPUTERNAME** variable contains the name of the computer PowerShell is running on. Evaluate it:  
   $env:computername
   1. What is the name of the computer you are connected to? Click or tap here to enter text.
2. Now evaluate the same expression on DC using the **Invoke-Command** cmdlet. Use curly-braces to put the expression inside a script block, like this:   
   Invoke-Command -ComputerName DC -ScriptBlock {$env:COMPUTERNAME}
   1. What was returned? Click or tap here to enter text.

Next, let’s try remoting a different command in a script block.

1. View all of the environment variables on DC:   
   Invoke-Command -ComputerName DC -ScriptBlock {dir env:}
   1. What is DC’s Processor\_Architecture? Click or tap here to enter text.
   2. What is the Number\_of\_processors on DC? Click or tap here to enter text.
2. There’s another remote VM named slaygore. Try the same for that host:   
   Invoke-Command -ComputerName slaygore -ScriptBlock {dir env:}
   1. What is slaygore’s Number\_of\_processors? Click or tap here to enter text.
   2. What is the UserDomain on slaygore? Click or tap here to enter text.

# Task 3—Examine Host-based Firewall Configuration for Remoting

Much of textbook chapter three is concerned with managing and securing remoting mechanisms. You will *not* change settings for remoting on **cit361-lab.cit.byui.edu** lab VMs, since they are shared resources, and you don’t want to risk breaking remoting capabilities that your classmates depend on. However, we can look at some of the remote configurations there, such as firewall rules.

## Steps

1. You need elevated privilege again for this step:  
   **Enter-PSSession localhost -ConfigurationName AdminShell**  
     
   View the firewall rules for WinRM remoting:  
   **Get-NetFirewallRule winrm\***
2. Look at the rule **WINRM-HTTP-In-TCP-NoScope**:  
   **Get-NetFirewallRule -Name WINRM-HTTP-In-Tcp-NoScope**
   1. From the description, what port number is being used?  
      Click or tap here to enter text.
3. Exit the privileged PowerShell process and return to your own PowerShell account prompt:  
   **exit**

When you have completed these tasks, go ahead and type **exit** *again* to close Windows Powershell, then **exit** *a third time* to close your SSH remote-access connection.

# Deliverable

Upload this document with completed answers to I-Learn Canvas.