PowerShell Remoting

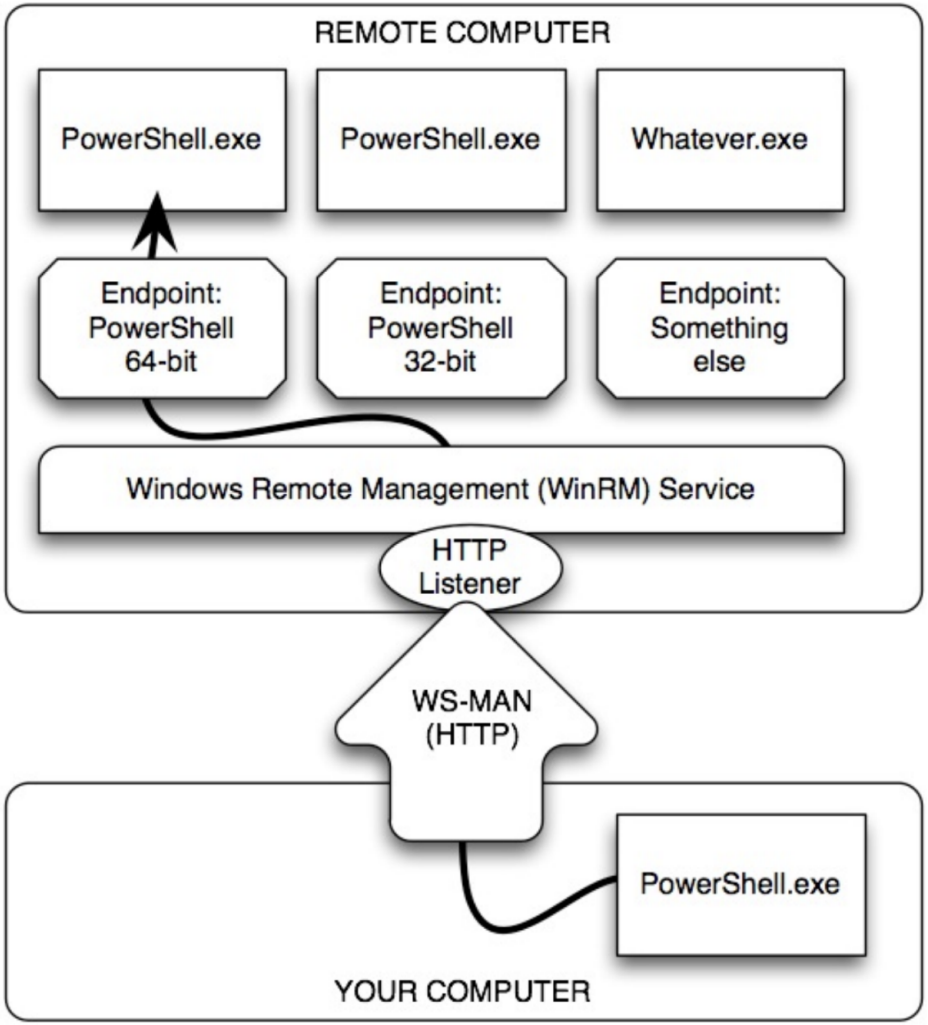
You want to run what, where?

PowerShell Remoting Introduction

- A protocol that allows running PowerShell commands on a single or multiple remote systems
- First introduced with PowerShell v2
- Based on the Simple Object Access Protocol
- Firewall friendly (uses one port)
  - 5985 for HTTP
  - 5986 for HTTPS
- Provides temporary or persistent (PSSessions) connections

PSRP Architecture

- PowerShell Remoting Protocol [1]
  - Encodes .NET objects prior to sending sending them over WinRM
- Windows Remote Management [2]
  - Microsoft Implementation of WS-Management
- WS-Management [3]
  - Protocol to provide consistency and interoperability for management across many types of devices and operating systems



[1]https://msdn.microsoft.com/en-us/library/dd357801.aspx
[2]https://msdn.microsoft.com/en-us/library/aa384426(v=vs.85).aspx
[3]https://msdn.microsoft.com/en-us/library/aa384470(v=vs.85).aspx

PSRP Security

- Traffic is Encrypted by Default (per-session AES-256 symmetric key)
- Kerberos Authentication by Default
  - Provides mutual authentication
  - Must specify the computer name of the remote system (not the IP Address)
- Significantly less overhead than other remote admin protocols
  - Remote Desktop Protocol
- Network Authentication
  - Credentials are not passed to remote system (no mimikatz)

https://docs.microsoft.com/en-us/powershell/scripting/setup/winrmsecurity?view=powershell-5.1

Enabling PowerShell Remoting

The Enable-PSRemoting cmdlet performs the following step:

1. Start WinRM Service
2. Set WinRM Service Startup Type to Automatic
3. Create WinRM Listener (HTTP and/or HTTPS)
4. Allow WinRM requests through local firewall
   - HTTP - 5985
   - HTTPS - 5986

PSRP ACLs

- The ACL for each PowerShell remote endpoint can be set
- By default, access is granted to:
  - NT AUTHORITY\INTERACTIVE
  - Administrators
  - Remote Management Users

```
PS C:\WINDOWS\system32> Get-PSSessionConfiguration | Select-Object -ExcludeProperty Permission


Name          : microsoft.powershell
PSVersion     : 5.1
StartupScript :
RunAsUser     :
Permission    : NT AUTHORITY\INTERACTIVE AccessAllowed, BUILTIN\Administrators AccessAllowed, BUILTIN\Remote Management Users
                AccessAllowed

Name          : microsoft.powershell.workflow
PSVersion     : 5.1
StartupScript :
RunAsUser     :
Permission    : BUILTIN\Administrators AccessAllowed, BUILTIN\Remote Management Users AccessAllowed

Name          : microsoft.powershell32
PSVersion     : 5.1
StartupScript :
RunAsUser     :
Permission    : NT AUTHORITY\INTERACTIVE AccessAllowed, BUILTIN\Administrators AccessAllowed, BUILTIN\Remote Management Users
                AccessAllowed
```

WinRM Listeners

- HTTP vs. HTTPS
  - WinRM is encrypted by default (both HTTP and HTTPS)
    - Must specify the ComputerName (not IP Address) to use Kerberos
  - HTTPS adds server identification for non-domain systems
    - Kerberos Authentication handles server identification transparently

```
ERROR:  The WinRM client cannot process the request. If the
authentication scheme is different from Kerberos, or if the client
computer is not joined to a domain, then HTTPS transport must be used
or the destination machine must be added to the TrustedHosts
configuration setting.
```

- IPv(4/6) Filter
  - This value specifies the local interface(s) that will accept PSRP requests
  - Typically set to * (all interfaces)

Connecting to Non-domain Systems

- By default, PS Remoting is limited to systems that meet the following criteria:
  - Use Kerberos Authentication
  - Domain joined
- This limitation is in place to guarantee mutual authentication
- PowerShell wants to use HTTPS instead of HTTP to connect
- You can explicitly trust a system by setting the TrustedHosts value
  - Ex. `Set-Item WSMan:\localhost\client\TrustedHosts -Value 192.168.1.10`
  - The TrustedHosts value accepts wildcards like *.specterops.io
- For more information check out about_remote_troubleshooting

Test-WSMan

- PowerShell's utility for testing Windows Remote Management
  - Sends a WinRM identification request to the local or remote machine
  - If WinRM is configured, returns service details such as:
    - WS-Management Identity Schema
    - Protocol Version
    - Product Vendor
    - Product Version
  - Should be your first troubleshooting step

```
PS C:\> Test-WSMan -ComputerName 10.1.20.123


wsmid           : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor   : Microsoft Corporation
ProductVersion  : OS: 0.0.0 SP: 0.0 Stack: 3.0
```

PSSessions

- Persistent PowerShell Remoting connection to a computer
- Limits overhead of each remote connection
  - Authentication
  - Session Standup
- Commands ran in the same session can share data (maintain state)
- Use the **New-PSSession** cmdlet to create a PSSession

```
PS C:\> $s = New-PSSession -ComputerName 10.1.10.25 -Credential localadmin
PS C:\> $s

 Id Name            ComputerName    ComputerType    State         ConfigurationName     Availability
 -- ----            ------------    ------------    -----         -----------------     ------------
  1 WinRM1          10.1.10.25      RemoteMachine   Opened        Microsoft.PowerShell     Available


PS C:\> Enter-PSSession -Session $s
[10.1.10.25]: PS C:\Users\localadmin\Documents> $env:COMPUTERNAME
zeus
```

Direct Remoting (1:1)

- Remote shell experience via the Enter-PSSession cmdlet
- Provides remote command line (PowerShell) access
- Requires less resources than Remote Desktop Protocol
- Prompt changes to `[<hostname>]: PS C:\>`
- Works with PSSessions or -ComputerName

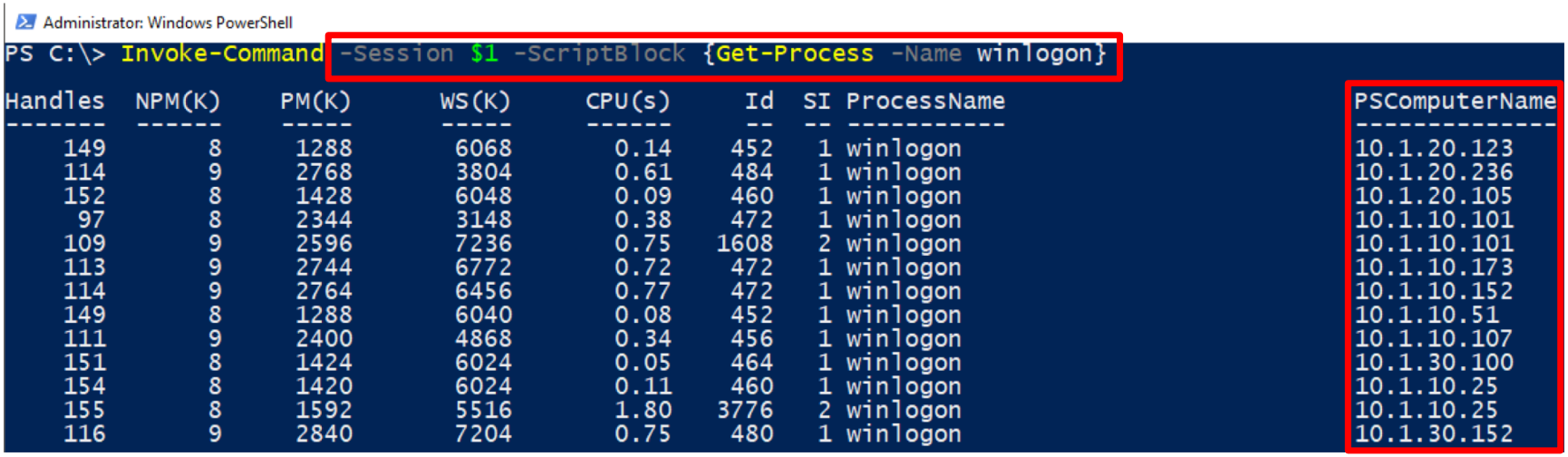
```
PS C:\> Enter-PSSession -ComputerName 10.1.10.25 -Credential localadmin
[10.1.10.25]: PS C:\Users\localadmin\Documents> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id  SI ProcessName
-------  ------    -----      ----- -----   ------     --  -- -----------
     44       5     1488       3800 ...79       0.02     76   0 conhost
     46       5     1528       3956 ...80       0.38   5360   0 conhost
    296      14     1680       3828     48       5.19    372   0 csrss
    176      11     1884      10824     49       1.00    424   1 csrss
    175      10     1732       9156     47      39.05   2328   2 csrss
    337      32    17688      23888 ...05      53.61   1384   0 dfsrs
    130      11     2012       5436     29       0.17   1880   0 dfssvc
  10284    5081    97656      97812 ...92      96.55   1436   0 dns
```

Exercise 1/2

- Use **Enter-PSSession** and the **-ComputerName** parameter to get a remote PowerShell on a system and run some commands
- Create a persistent session
  - `$Cred = Get-Credential`
  - `New-PSSession -ComputerName REMOTING -Credential $Cred`
- Enter persistent session and run a command
  - `$s = Get-PSSession`
  - `Enter-PSSession -Session $s`
  - `$proc = Get-Process`
- Exit and re-enter session
  - `exit`
  - `Enter-PSSession`
  - `$proc`

One to Many Remoting

- Execute a script or scriptblock across many systems
- Threaded by default (32 concurrent runspaces by default)
  - Number of default connections can be set with -ThrottleLimit
- Adds ‘PSComputerName’ field to output instances



CIM Sessions

- Available for PowerShell v3 and later
- Allows for WMI over WinRM (not PSRP)
- Can create reusable sessions to reduce authentication overhead

```
PS C:\> $s = New-CimSession -ComputerName localhost
PS C:\> Get-CimInstance -CimSession $s -ClassName Win32_Process

ProcessId Name                         HandleCount WorkingSetSize VirtualSize   PSComputerName
--------- ----                         ----------- -------------- -----------   --------------
0         System Idle Process          0           8192           65536         localhost
4         System                       2463        77824          3653632       localhost
48        Secure System                0           2744320        0             localhost
324       smss.exe                     52          946176         2199029895168 localhost
432       csrss.exe                    514         6397952        2199089668096 localhost
512       wininit.exe                  145         6524928        2199078645760 localhost
520       csrss.exe                    260         5058560        2199074684928 localhost
588       winlogon.exe                 244         16207872       2199109509120 localhost
660       services.exe                 596         9560064        2199053832192 localhost
668       lsass.exe                    1265        16941056       2199075364864 localhost
764       svchost.exe                  70          3784704        2199047487488 localhost
```

-ComputerName

- -ComputerName does not mean PS Remoting is used
- Many cmdlets (Get-Process or Get-WmiObject) use DCOM or RPC to execute queries on remote systems
  - This can cause issues with host/network firewalls
- The following cmdlets are built on PSRemoting:
  - *-PSSession
  - Invoke-Command
  - *-Cim*
  - Copy-Item
  - Get-Command -ParameterName CimSession

Local vs Remote Processing

- Important to keep in mind where filtering is being performed
- Filter as much as possible on the remote machine
  - If you scan 1,000 endpoints for currently running powershell processes, then filter on the remote machine instead of returning all processes over the network
- Methods on returned objects may be limited
  - Data returned from PowerShell remoting are deserialized snapshots of what was on the remote computer at the time of the command

Filter Remotely

```
PS C:\> Measure-Command {$result = Invoke-Command -Session $s -ScriptBlock {Get-Process -Name lsass}}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 1
Milliseconds      : 834
Ticks             : 18340946
TotalDays         : 2.12279467592593E-05
TotalHours        : 0.000509470722222222
TotalMinutes      : 0.0305682433333333
TotalSeconds      : 1.8340946
TotalMilliseconds : 1834.0946


PS C:\> Measure-Command {$result = Invoke-Command -Session $s -ScriptBlock {Get-Process} | Where-Object {$_.Name -eq 'lsass'}}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 8
Milliseconds      : 22
Ticks             : 80220752
TotalDays         : 9.28480925925926E-05
TotalHours        : 0.00222835422222222
TotalMinutes      : 0.133701253333333
TotalSeconds      : 8.0220752
TotalMilliseconds : 8022.0752
```

Execute Methods Remotely

```
PS C:\> Invoke-Command -Session $s[0] -ScriptBlock {Get-Process -Name calc}

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName                                 PSComputerName
-------  ------    -----      -----     ------      --  -- -----------                                 --------------
    154      10     4724       8676       0.06    3644   0 calc                                        10.1.20.236


PS C:\> Invoke-Command -Session $s[0] -ScriptBlock {Get-Process -Name calc} | Stop-Process
Stop-Process : Cannot find a process with the process identifier 3644.
At line:1 char:71
+ ... d -Session $s[0] -ScriptBlock {Get-Process -Name calc} | Stop-Process
+                                                             ~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (3644:Int32) [Stop-Process], ProcessCommandException
    + FullyQualifiedErrorId : NoProcessFoundForGivenId,Microsoft.PowerShell.Commands.StopProcessCommand

PS C:\> Invoke-Command -Session $s[0] -ScriptBlock {Get-Process -Name calc | Stop-Process}
PS C:\> Invoke-Command -Session $s[0] -ScriptBlock {Get-Process -Name calc}
Cannot find a process with the name "calc". Verify the process name and call the cmdlet again.
    + CategoryInfo          : ObjectNotFound: (calc:String) [Get-Process], ProcessCommandException
    + FullyQualifiedErrorId : NoProcessFoundForGivenName,Microsoft.PowerShell.Commands.GetProcessCommand
    + PSComputerName        : 10.1.20.236
```

Executing Scripts Remotely

- Invoke-Command has a -Filename parameter
  - Passes a local script to a remote system and executes it
    - If your script defines a function, the function must be called if you want it to execute
  - The script is written to disk (temp directory), executed, and deleted

```
PS C:\> Get-Content C:\Users\tester\Desktop\foo.ps1
function foo
{
    Write-Host "foo executed"
}

foo
PS C:\> $s[3]

 Id Name            ComputerName    ComputerType    State         ConfigurationName     Availability
 -- ----            ------------    ------------    -----         -----------------     ------------
  5 WinRM5           10.1.10.101     RemoteMachine   Opened        Microsoft.PowerShell     Available


PS C:\> Invoke-Command -Session $s[3] -FilePath C:\Users\tester\Desktop\foo.ps1
foo executed
```

Executing Functions Remotely

- PS Remoting can pass a locally defined function to a remote system
- Can not resolve additional function dependencies
- Call a local function with `${function:foo}` syntax

```
PS C:\> function foo{Write-Host "foo executed"}
PS C:\> $s[0]

 Id Name            ComputerName    ComputerType    State         ConfigurationName     Availability
 -- ----            ------------    ------------    -----         -----------------     ------------
  2 WinRM2          10.1.20.236     RemoteMachine   Opened        Microsoft.PowerShell     Available


PS C:\> foo
foo executed
PS C:\> Invoke-Command -Session $s[0] -ScriptBlock ${function:foo}
foo executed
```

```
PS C:\> function foo{bar}
PS C:\> function bar{Write-Host "bar executed"}
PS C:\> Invoke-Command -Session $s[0] -ScriptBlock ${function:foo}
The term 'bar' is not recognized as the name of a cmdlet, function, script file, or operable program.
the name, or if a path was included, verify that the path is correct and try again.
    + CategoryInfo          : ObjectNotFound: (bar:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
    + PSComputerName        : 10.1.20.236
```


http://mikefrobbins.com/2014/03/27/run-a-local-powershell-function-against-a-remote-computer-with-powershell-remoting/

Nested Functions for Remoting

```
PS C:\> function foo
>> {
>>     function bar
>>     {
>>         Write-Host "bar executed"
>>     }
>>
>>     Write-Host "foo executed"
>>     bar
>> }
PS C:\> foo
foo executed
bar executed
PS C:\> Invoke-Command -Session $s[0] -ScriptBlock ${function:foo}
foo executed
bar executed
```

Exercise 2/2

- Gather a process listing from a remote systems (non-interactively)
- Create a local function and run on a remote system
  - `function foo {Get-Process}`
- Create two local functions, one that calls the other, and run them on a remote system
  - Example:
    - `function foo {bar}`
    - `function bar {Get-Process}`