Hey, Scripting Guy! Blog

Learn about Windows PowerShell

Creating a Port Scanner with Windows PowerShell

ScriptingGuy1 19 Mar 2014 12:01 AM 8

Summary: Microsoft Scripting Guy, Ed Wilson, talks about creating a port scanner with Windows PowerShell.

Microsoft Scripting Guy, Ed Wilson, is here. The other day, I needed to access my printer. Unfortunately, after several networking changes, I did not remember the IP address of my printer. However, I did know that the printer sets up a web server. It was this web server that I needed to access so I could make some changes to the way the printer was handling default forms.

But dude, I did not know the IP address, and I did not want to have to rummage around in my network documentation to find the particular printer in question. So what to do? I figured it would be easier to write a port scanner. By using Windows PowerShell, this was actually a pretty simple task. I simply needed to look for a device listening on Port 80.

Create a range of IP addresses

First I will admit that my solution is not the cleanest solution possible. I will also admit it is not the fastest solution possible. I think my solution might simply be useful to show something that I could do. And like I said in my introduction, it met my need at the time. Perhaps one of the cool things about Windows PowerShell is that it permits this sort of ad-hoc solution to be easily created.

So, I needed to create a range of IP addresses. Actually, the only range I needed to create was a range of host addresses for a class A network. Therefore, I used the range operator and created an array of numbers that go from 1 to 254. As I mentioned earlier, I am only interested in Port 80, so I assign that as a value to the variable **$port**. My class A network address is 192.168.0, so I assign that as a value to the **$net** variable. This is shown here:

```
$port = 80

$net = "192.168.0"

$range = 1..254
```

Note I highly recommend using meaningful variables when you write a script—even a quick script such as this. Using the Hungarian Notation is not required, but I think that calling a port **$port** certainly is. It makes the script easier to read and understand.

Create the IP addresses

Now, I am going to walk through my array of numbers (1..254) and create a whole bunch of IP addresses. Of course, I will create them one at a time. After an IP address is created, I ping the address to see if it responds. If it does, I will attempt a connection to Port 80 to see if I find a web server for my printer. But first the IP address…

To walk through the array, I use the **ForEach** command. Following the word **ForEach**, I create a new variable, **$r**, to hold the individual number from the array. Therefore, the first time through, **$r** will be equal to 1.

I then open a script block, and create my IP address, and store it in the variable **$ip**. There are two parts to my IP address, the first part is my network address, 192.168.0, the second part is the individual number I get from my **$range** of numbers. I want to put them together in order, first the **$net** address followed by a period, and then the number (**$r**).

I use a format specifier to do this. **"{0}.{1}"** tells Windows PowerShell to take the first value following the **–F** and put it in position {0}. In this case, it is the value that is stored in the **$net** variable. Next we have the "**.**" And then the second thing to substitute, which goes into position {1}. This value will be the second value following the **–F**, which is the value stored in the **$r** variable. Here is the script that does all this:

```
foreach ($r in $range)

{

 $ip = "{0}.{1}" -F $net,$r
```

Ping…

After I have created my IP address, it is time to ping the remote computer. I use the **Test-Connection** cmdlet to do this. I am using **–Quiet** mode, and therefore it will return **$true** or **$false** depending on whether it receives a return. I specify a buffer size of 32, and I want to send one ping only. Then I specify the destination as the IP address stored in the **$IP** variable. The command is shown here:

```
if(Test-Connection -BufferSize 32 -Count 1 -Quiet -ComputerName $ip)

  {
```

If I get a return, I attempt to make a connection to Port 80. To do this, I use a .NET Framework class, TcpClient from the System.Net.Sockets assembly. The TcpClient class lets me specify a port number and an IP address. I store any returned socket in the **$socket** variable. This command is shown here:

```
{

    $socket = new-object System.Net.Sockets.TcpClient($ip, $port)
```

Did we make a connection?

Now I need to see if I made a connection to Port 80. To do this, I can check the **Connected** property. If I am connected to the remote socket, I print a string that the device at that IP address is listening to Port 80. I then close the connection. This script is shown here:

```
If($socket.Connected)

      {

       "$ip listening to port $port"

       $socket.Close() }

       }

}
```

That is it. I did not need to check anything else, nor do I need an **ELSE** condition. I was simply looking for devices listening to Port 80. Interestingly enough, I found out that one of my switches had also set up a web server. This, I am afraid to admit, I did not know. So this also becomes a useful network security technique.

Here is the completed script—not much to it, but it worked for me.

```
# -----------------------------------------------------------------------------

# Script: PoshPortScanner.ps1

# Author: ed wilson, msft

# Date: 02/19/2014 15:17:33

# Keywords: Security, Networking, Tcp/IP, Monitoring

# comments: This script scans a range of IP addresses for web servers listening

# to port 80. It is a useful audit tool, because there are lots of software and

# devices that setup web servers for management, but that do not necessarily

# inform about them.

#

# -----------------------------------------------------------------------------

$port = 80

$net = "192.168.0"

$range = 1..254

foreach ($r in $range)

{

 $ip = "{0}.{1}" -F $net,$r

 if(Test-Connection -BufferSize 32 -Count 1 -Quiet -ComputerName $ip)

   {

    $socket = new-object System.Net.Sockets.TcpClient($ip, $port)

    If($socket.Connected)

      {

       "$ip listening to port $port"
```

```
            $socket.Close() }

             }

        }
```

Join me tomorrow when I will talk about more cool Windows PowerShell stuff.

I invite you to follow me on Twitter and Facebook. If you have any questions, send email to me at scripter@microsoft.com, or post your questions on the Official Scripting Guys Forum. See you tomorrow. Until then, peace.

Ed Wilson, Microsoft Scripting Guy

Tweet 47 Like 3 Share Save this on Delicious

Comments

19 Mar 2014 9:42 AM

M.T.Nielsen - mni@systemhosting.dk

$scope = '192.168.0'
$port = 80
$result = 1..254 | % { Test-NetConnection ("{0}.$_" -f $scope) -Port $port }

Takes a while to run though, due to there being no -Timeout parameter.

19 Mar 2014 10:50 AM

ed wilson

@M.T.Nielsen Test-NetConnection does make it easy to do this sort of thing. It is from the NetTCPIP module, which I believe was only introduced beginning with Windows 8.1. My version, while being a bit more complicated, is compatible back to PowerShell 2.0 systems.

19 Mar 2014 11:12 AM

AlexP

@M.T.Nielsen

Test-NetConnection is a custom function, as far as I'm aware.

@Ed, thanks for a nice post, learned about format specifiers :-) - still new to PS

19 Mar 2014 12:00 PM

M.T.Nielsen - mni@systemhosting.dk

@ed

It's absolutely a Powershell 4 cmdlet, but with that said it's still very relevant to the article.

Speaking of relevant, here's the WMF4.0 download: http://www.microsoft.com/en-us/download/details.aspx?id=40855

19 Mar 2014 1:50 PM

ed wilson

@M.T.Nielsen you are right it is definitely applicable to this article. Thank you for sharing it.

22 Mar 2014 12:59 AM

Arne

192.168.0.0/24 is a class C network, not a class A network

Jeffrey S. Patton 23 Sep 2014 6:26 PM

I made one of these a while ago, just to add some fuel ;-)

http://gallery.technet.microsoft.com/New-SubnetSweepps1-67bee5b7