



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

## Interested in learning more?

Check out the list of upcoming events offering  
"Network Monitoring and Threat Detection In-Depth (Security 503)"  
at <http://www.giac.org/registration/gcia>

# Intrusion Analysis Using Windows PowerShell

*GIAC (GCIA) Gold Certification*

Author: Michael J. Weeks, mweeks9989@gmail.com

Advisor: Angel Alonso-Parrizas

Accepted: May 29th 2014

## Abstract

Microsoft has continually evolved its technology and has introduced some tools that can be used for intrusion analysis. The Windows Advanced Firewall and custom Windows Event Logs are some examples but this paper focuses on a quantum leap forward: PowerShell. Many Analysts must use Windows as their main platform for analysis, and with PowerShell alone, they can perform many of their daily duties. PowerShell is not just an administration language: it can also perform regular expression pattern matching, check the integrity of network monitoring, parse and analyze security events and almost limitless potential other uses. In this paper, we will dive into a few of the many techniques and capabilities of these technologies.

## 1. Introduction

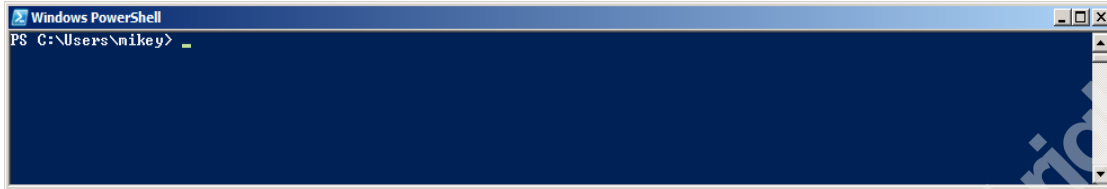
Microsoft during the late 90s and through the turn of the millennium was not held in high regard in terms to security. Microsoft stopped all development in 2002, and Bill Gates ushered in an era of what he called “Trustworthy Computing” (Callahan, 2014). He defined Trustworthy Computing as “computing that is as available, reliable and secure as electricity, water services and telephony” (Gates, 2012). These efforts have gone a long way to make Microsoft the largest client-side operating system on the planet (w3Schools.com, 2014). It is likely that an intrusion detection Analyst will be using a type of Microsoft Windows Operating System as his main workstation, as well as analyzing Microsoft Systems. In the past many tools were downloaded, with a focus on Linux power tools in order to properly perform analysis. Microsoft now provides tools as Microsoft Event Viewer and Windows Firewall, eliminating the need for other tools. Hence, intrusion analysis can be performed without downloading a lot of tools. One of the best tools that Microsoft created is PowerShell. PowerShell is full-featured scripting language that was built by Microsoft as an administration language on the .NET Framework. (TechNet, 2013). PowerShell offers much more than its predecessors: it has the ability to run all of the classic cmd.exe commands like the net.exe and netsh.exe and has all the com objects built-in so VB scripts can be upgraded to the superior PowerShell.

PowerShell as an analysis language can use the administrative capability to perform monitoring tasks of some of the other Microsoft Security technologies such as: Microsoft Windows Firewall, Active Directory, and Windows Event Logs. In order to take advantage of the monitoring capability of PowerShell, an Analyst will need to learn how to script and use programmatic logic, which in PowerShell is not difficult, although some of the nuances can be complicated.

### 2.1. PowerShell – working in the Shell

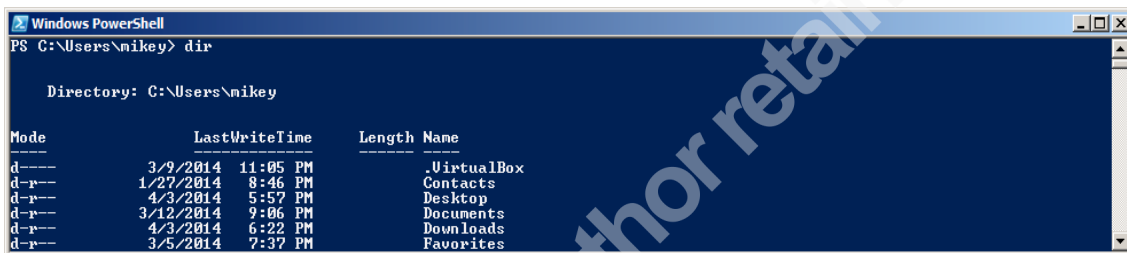
The first thing to learn when using a new tool is how to get familiar with it. If the system is at least Windows 7 Professional (and at the time of this paper they should be at

a minimum), PowerShell comes pre-installed and is similar to the start > run “cmd” but instead type “powershell”.



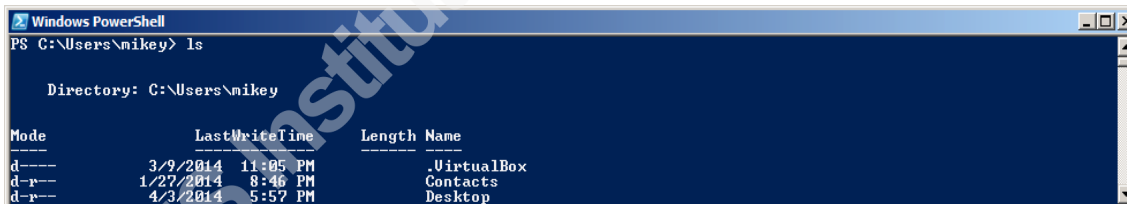
**Figure 1. - The PowerShell**

The blue PowerShell window appears (Figure. 1) this is the new and improved shell that Microsoft has provided.



**Figure 2. - dir**

Upon entering “dir” and a directory listing appears, it looks different than the old cmd.exe output (Figure 2.).



**Figure 3. - ls**

Also entering “ls” – will provide the same output. This is an alias for the Get-ChildItem PowerShell cmdlet (Figure 3. and 4).



**Figure 4. – Get-Alias ls**

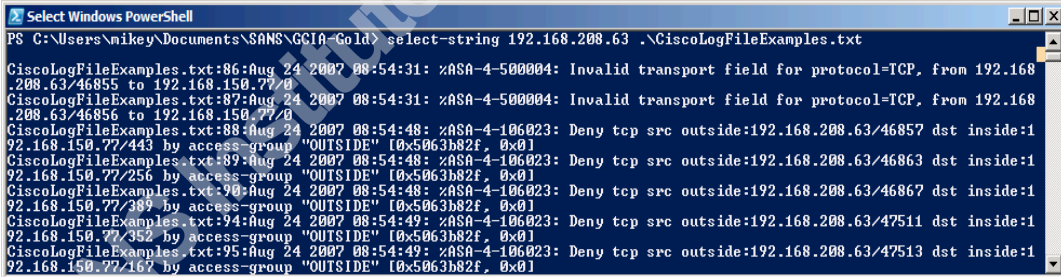
## 2.2. Log Analysis with PowerShell

PowerShell guides such as the official guide from Microsoft [Windows PowerShell First Steps](#), (Wilson 2013) or [PowerShell deep Dive](#) (Hicks et al, 2013) show that the `get-alias` cmdlet can provide and set an alias for any cmdlet in the PowerShell environment. PowerShell uses a noun-verb pair for its naming of the .net based commands. `Get-Command` when ran shows what cmdlets are available. `Get-Help` can be run with a tremendous amount of options against any other cmdlet (the `-example` is particularly helpful). After reviewing the cmdlets, the `Select-String` cmdlet should appear very interesting, the first line in the DESCRIPTION section reads: “The `Select-String` cmdlet searches for text and text patterns in input strings and files. You can use it like `Grep` in UNIX and `Findstr` in Windows.” To test, look at any syslog file and identify the specific regular expression. In order to evaluate the capabilities of PowerShell log analysis, the syslog examples from Cisco are downloaded:

<http://www.cisco.com/web/about/security/intelligence/identify-incidents-via-syslog.html>

One of the IP addresses in the file is 192.168.208.63 by running: **Select-String 192.168.208.63 .\CiscoLogFileExamples.txt** the following results are displayed (Figure

5).



```

PS C:\Users\mikey\Documents\SANS\GCIA-Gold> select-string 192.168.208.63 .\CiscoLogFileExamples.txt
CiscoLogFileExamples.txt:86:Aug 24 2007 08:54:31: %ASA-4-500004: Invalid transport field for protocol=TCP, from 192.168
.208.63/46855 to 192.168.150.77/0
CiscoLogFileExamples.txt:87:Aug 24 2007 08:54:31: %ASA-4-500004: Invalid transport field for protocol=TCP, from 192.168
.208.63/46856 to 192.168.150.77/0
CiscoLogFileExamples.txt:88:Aug 24 2007 08:54:48: %ASA-4-106023: Deny tcp src outside:192.168.208.63/46857 dst inside:1
92.168.150.77/443 by access-group "OUTSIDE" [0x5063b82f, 0x0]
CiscoLogFileExamples.txt:89:Aug 24 2007 08:54:48: %ASA-4-106023: Deny tcp src outside:192.168.208.63/46863 dst inside:1
92.168.150.77/256 by access-group "OUTSIDE" [0x5063b82f, 0x0]
CiscoLogFileExamples.txt:90:Aug 24 2007 08:54:48: %ASA-4-106023: Deny tcp src outside:192.168.208.63/46867 dst inside:1
92.168.150.77/389 by access-group "OUTSIDE" [0x5063b82f, 0x0]
CiscoLogFileExamples.txt:94:Aug 24 2007 08:54:49: %ASA-4-106023: Deny tcp src outside:192.168.208.63/47511 dst inside:1
92.168.150.77/352 by access-group "OUTSIDE" [0x5063b82f, 0x0]
CiscoLogFileExamples.txt:95:Aug 24 2007 08:54:49: %ASA-4-106023: Deny tcp src outside:192.168.208.63/47513 dst inside:1
92.168.150.77/167 by access-group "OUTSIDE" [0x5063b82f, 0x0]

```

**Figure 5. – Select-String**

In (Figure 5.) the output looks like the \*nix command, ‘`grep`’ but with some extra data. The name of the file and line number is output to the shell. This data may be interesting in some cases, but to obtain just the matching line the output must be piped to a command that selects the line object (Figure 6.).

```

Windows PowerShell
PS C:\Users\mikey\Documents\SANS\GCIA-Gold> select-string 192.168.208.63 .\CiscoLogFileExamples.txt | select line
Line
Aug 24 2007 08:54:31: xASA-4-500004: Invalid transport field for protocol=TCP, from 192.168.208.63/46855 to 192.168...
Aug 24 2007 08:54:31: xASA-4-500004: Invalid transport field for protocol=TCP, from 192.168.208.63/46856 to 192.168...
Aug 24 2007 08:54:48: xASA-4-106023: Deny tcp src outside:192.168.208.63/46857 dst inside:192.168.150.77/443 by acce...
Aug 24 2007 08:54:48: xASA-4-106023: Deny tcp src outside:192.168.208.63/46863 dst inside:192.168.150.77/256 by acce...
Aug 24 2007 08:54:48: xASA-4-106023: Deny tcp src outside:192.168.208.63/46867 dst inside:192.168.150.77/389 by acce...
Aug 24 2007 08:54:49: xASA-4-106023: Deny tcp src outside:192.168.208.63/47511 dst inside:192.168.150.77/352 by acce...
Aug 24 2007 08:54:49: xASA-4-106023: Deny tcp src outside:192.168.208.63/47513 dst inside:192.168.150.77/167 by acce...
Aug 24 2007 08:54:49: xASA-4-106023: Deny tcp src outside:192.168.208.63/47517 dst inside:192.168.150.77/210 by acce...
Aug 24 2007 08:54:49: xASA-4-106023: Deny tcp src outside:192.168.208.63/47522 dst inside:192.168.150.77/281 by acce...
Aug 24 2007 09:02:37: xASA-4-106023: Deny tcp src outside:192.168.208.63/51584 dst inside:192.168.150.77/239 by acce...
Aug 24 2007 09:02:37: xASA-4-106023: Deny tcp src outside:192.168.208.63/51585 dst inside:192.168.150.77/288 by acce...
Aug 24 2007 09:02:37: xASA-4-106023: Deny tcp src outside:192.168.208.63/51586 dst inside:192.168.150.77/147 by acce...

```

**Figure 6. – Select-String select line**

When extracted the object identified in the Select-String Cmdlet matches the select line in the pattern search. To see how many connections are made when analyzing a single host, the output from that can be piped to another command: Measure-Object (Figure 7.).

```

Windows PowerShell
PS C:\Users\mikey\Documents\SANS\GCIA-Gold> select-string 192.168.208.63 .\CiscoLogFileExamples.txt | select line | Measure-Object
Count      : 89
Average    :
Sum        :
Maximum    :
Minimum    :
Property   :
PS C:\Users\mikey\Documents\SANS\GCIA-Gold>

```

**Figure 7. – Select-String measure line**

The command is able to obtain count information as well as average, sum, max, min, and other property information. The following command (Figure 8.) was taken from Command Line Kung-Fu, (Williams, 2011), it will select all IP addresses in the file expand the matches property, select the value, get unique values and measure the output.

```

Windows PowerShell
PS C:\Users\mikey\Documents\SANS\GCIA-Gold> select-string "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select -ExpandProperty value | Sort-Object -Unique | Measure-Object
Count      : 23
Average    :
Sum        :
Maximum    :
Minimum    :
Property   :
PS C:\Users\mikey\Documents\SANS\GCIA-Gold>

```

**Figure 8. – Select-String sort and measure output count**

Removing Measure-Object shows all the individual IPs instead of just the count of the IP addresses (Figure 9.). The Measure-Object command counts the IP addresses.

```

Windows PowerShell
PS C:\Users\nikey\Documents\SANS\GCIA-Gold> select-string "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select -ExpandProperty value | Sort-Object -Unique
172.16.1.0
172.16.1.14
172.16.1.182
172.16.1.2
172.16.1.208
172.16.1.219
172.16.1.229
172.16.1.26
172.16.1.42
172.16.1.49
172.16.1.72
172.16.1.92
172.16.1.95

```

**Figure 9. – Select-String sort -uniq**

In order to determine which IP addresses have the most communication the last commands are removed to determine the value of the matches (Figure 9.). Then the group command is issued on the piped output to group all the IP addresses (value), and then sort the objects by using the alias for Sort-Object: **sort count -des**. This sorts the IP addresses in a descending pattern as well as count and deliver the output to the shell (Figure 10.).

```

Windows PowerShell
PS C:\Users\nikey\Documents\SANS\GCIA-Gold> select-string "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select value | group value | sort count -des
Count Name Group
-----
89 192.168.208.63 {<@Value=192.168.208.63>, @<Value=192.168.208.63>, @<Value=192.168.208.63>, @<Value=...
6 172.16.1.92 {<@Value=172.16.1.92>, @<Value=172.16.1.92>, @<Value=172.16.1.92>, @<Value=172.16.1.92>...
6 172.16.1.14 {<@Value=172.16.1.14>, @<Value=172.16.1.14>, @<Value=172.16.1.14>, @<Value=172.16.1....
4 172.16.1.49 {<@Value=172.16.1.49>, @<Value=172.16.1.49>
2 172.16.1.49 {<@Value=172.16.1.49>, @<Value=172.16.1.49>
1 192.168.150.70 {<@Value=192.168.150.70>
1 192.168.20.55 {<@Value=192.168.20.55>
1 172.16.1.2 {<@Value=172.16.1.2>
1 172.16.1.72 {<@Value=172.16.1.72>
1 192.168.28.68 {<@Value=192.168.28.68>
1 192.168.8.21 {<@Value=192.168.8.21>
1 192.168.16.231 {<@Value=192.168.16.231>
1 192.168.18.13 {<@Value=192.168.18.13>
1 192.168.11.31 {<@Value=192.168.11.31>

```

**Figure 10. Select-String Count IPs**

With this type of analysis, it is easy to find the top talkers in a log file. Here is the command for ease of use in the future:

```

PS :> select-string "\b(?:\d{1,3}\.){3}\d{1,3}\b" .\CiscoLogFileExamples.txt | select -ExpandProperty matches | select value | group value | sort count -des.
(Williams, 2011)

```

### 2.3. Windows Firewall Analysis with PowerShell

A bane to windows systems analysts is the difficulty of centralizing Windows Firewall logs to a central repository for review. A central log repository could be configured for the Windows Advanced Firewall using the legacy command netsh.exe, PowerShell baked in logic, some .Net and some Windows Administration. The first step to this is to find where the logs are on a windows system.

Author Name, email@address

```
netsh advfirewall show allprofiles | Select-String FileName | select -ExpandProperty line | Select-String "%systemroot%.+\.log" | select -ExpandProperty matches | select -ExpandProperty value | sort -uniq
```

This will get the setting for logs in the windows firewall which should be enabled in GPO policy for analysis. The command shows that the Firewall log is at: %systemroot%\system32\LogFiles\Firewall\pfirewall.log, in order to open the file PowerShell will need to be run with administrative privileges. First step is to get the above command into a variable using script logic. Thankfully PowerShell has a built-in integrated scripting environment, PowerShell.ise.

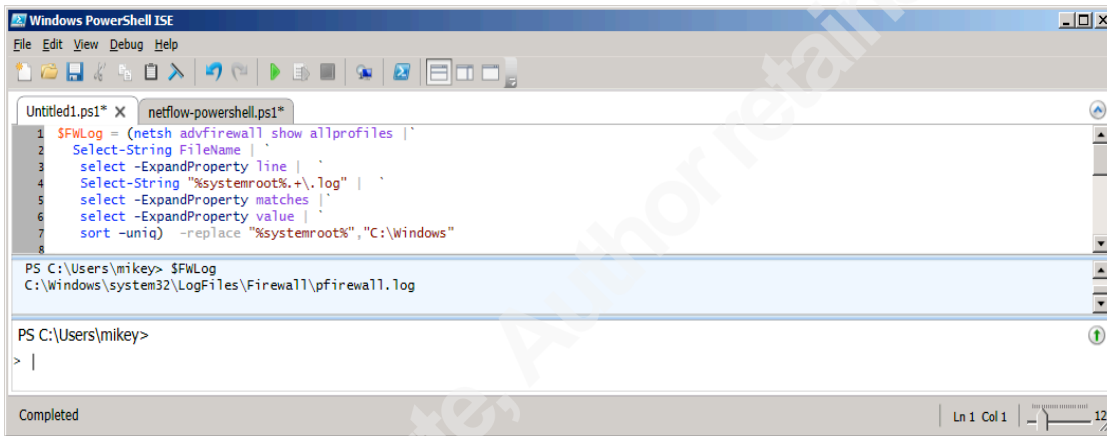


Figure 11. – Firewall Logger

Then finally run the above command (Figure 10.) and get the connections count. The following type of information will be displayed (Figure 12.):

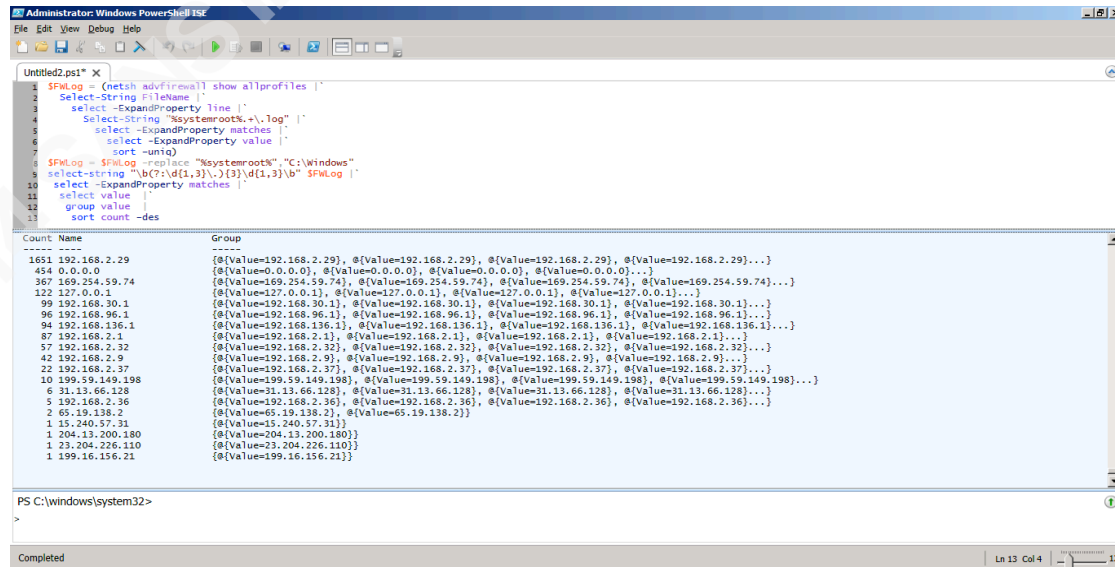
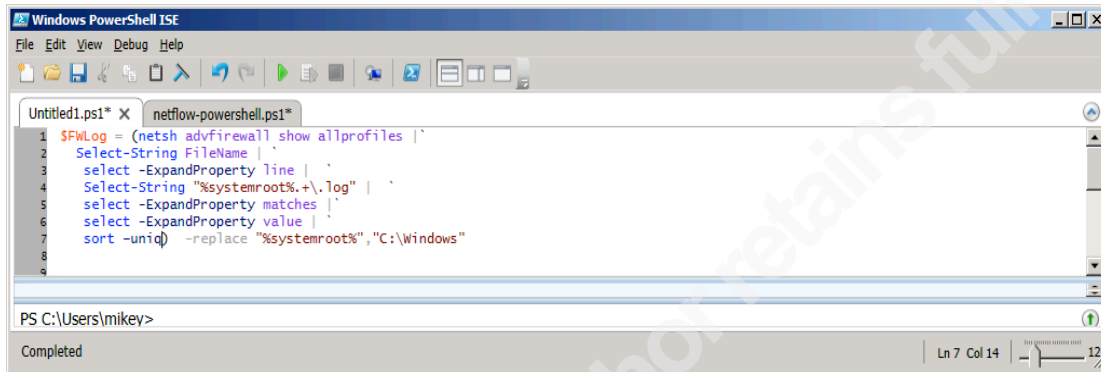


Figure 12. – Firewall Analysis

Through this information, the level of connections, the default gateway and all connections to and from the system can all be extrapolated. Now in order to solve the central logging problem, where on the network to send data needs to be determined. Using the assumption that he chooses “[\\secureshare\logs\](#)”, changing this command is simple. To send this system’s logs to the share drive with the following format: (Date)-(Hostname)-FWLogs.log, run the same log-name identifier (Figure 13.):

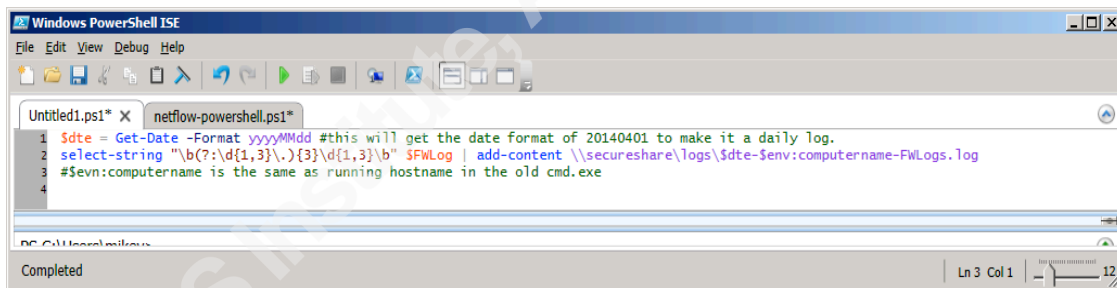


```

Windows PowerShell ISE
File Edit View Debug Help
Untitled1.ps1* X netflow-powershell.ps1*
1 $FWLog = (netsh advfirewall show allprofiles |
2 Select-String FileName |
3 select -ExpandProperty line |
4 Select-String "%systemroot%.+\.log" |
5 select -ExpandProperty matches |
6 select -ExpandProperty value |
7 sort -uniq) -replace "%systemroot%", "C:\Windows"
8
PS C:\Users\mikey>
Completed Ln 7 Col 14

```

**Figure 13. – Get Firewall Logs**



```

Windows PowerShell ISE
File Edit View Debug Help
Untitled1.ps1* X netflow-powershell.ps1*
1 $dte = Get-Date -Format yyyyMMdd #this will get the date format of 20140401 to make it a daily log.
2 select-string "\b(?:[d{1,3}\.]{3}[d{1,3}]b" $FWLog | add-content \\secureshare\logs\$dte-Env:computername-FWLogs.log
3 #Env:computername is the same as running hostname in the old cmd.exe
4
PS C:\Users\mikey>
Completed Ln 3 Col 1

```

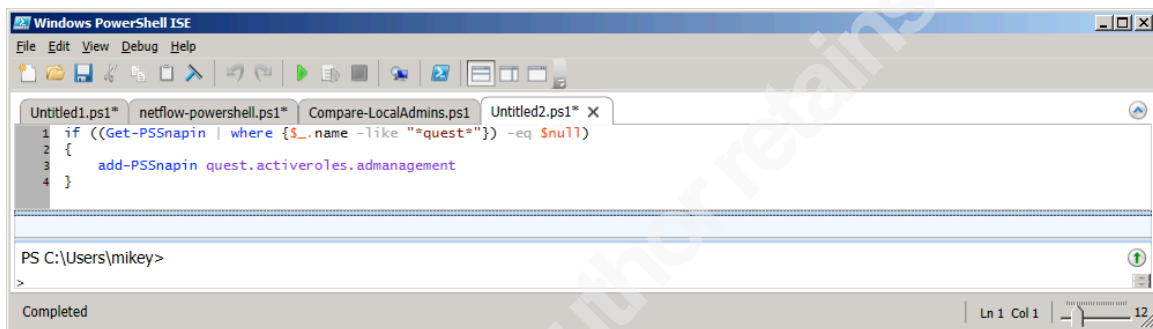
**Figure 14. – Send to Central Log**

With the log file cleaned up (Figure 14.), now it is simply another selection of what is wanted and sending that to the log file. Either a schedule task can be created across the domain using GPO, the invoke-command cmdlet, or create scheduled tasks on critical systems.

## 2.4. PowerShell Modules

PowerShell has tremendous baked in functionality, but when additional capability is needed, there are the PowerShell Modules and PowerShell Community Extensions. These modules are written to add functionality to PowerShell and a very useful module is the Quest Active Directory cmdlets, by Quest (Dell) Software. The cmdlets can be

downloaded at <http://www.quest.com/powershell/activeroles-server.aspx>. After installation, the Quest Active Directory Shell can be started from the start menu or run the following code: `Add-PSSnapin quest.activeroles.admanagement`. This will run the PowerShell snapin if it is not added already (if it has, the console will just output an error), this is especially good for scripting. Scripts using this module are very good for doing monitoring of critical security groups such as “Domain Admin”. To monitor a Security Group, add the code below (Figure 15) into the top of the script after installing Quest Active Directory cmdlets.



```

1 if ((Get-PSSnapin | where {$_.name -like "*quest*"} -eq $null)
2 {
3     add-PSSnapin quest.activeroles.admanagement
4 }

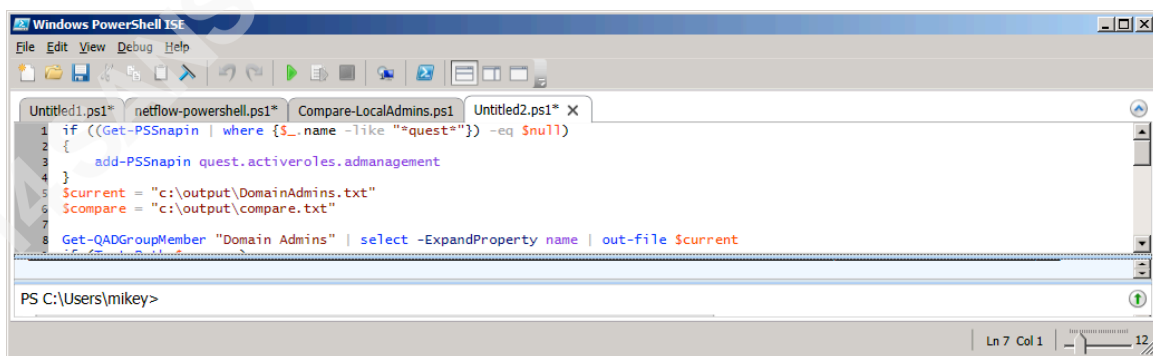
```

PS C:\Users\mikey>

Completed

**Figure 15. – Import Module**

Monitoring of the “Domain Admins” account should be simple now. By creating variables for two files and piping the output of the `Get-QADGroupMember` command into the `$current` variable through the `Out-File` cmdlet to get a current group membership of the monitored account (Figure 16).



```

1 if ((Get-PSSnapin | where {$_.name -like "*quest*"} -eq $null)
2 {
3     add-PSSnapin quest.activeroles.admanagement
4 }
5 $current = "c:\output\DomainAdmins.txt"
6 $compare = "c:\output\compare.txt"
7
8 Get-QADGroupMember "Domain Admins" | select -ExpandProperty name | out-file $current

```

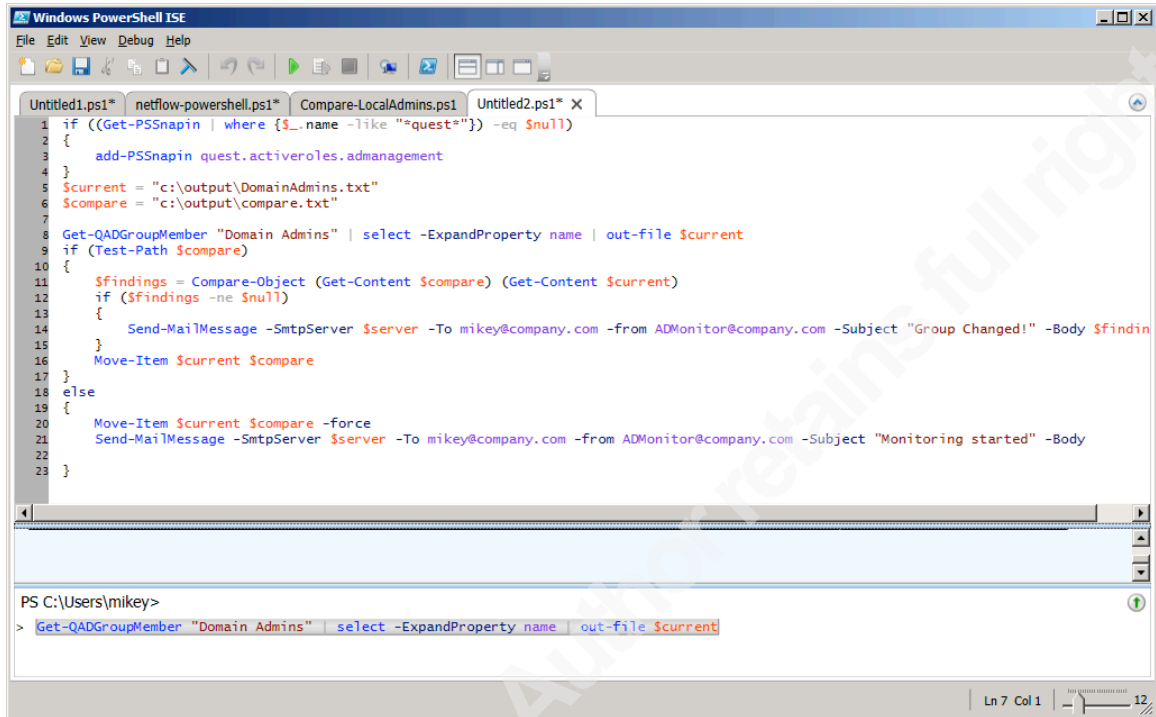
PS C:\Users\mikey>

**Figure 16. – Compare Admins**

Next the logic has to be created for the files. The logic tests if the compare path exists: the difference between the current and compare files, if there are any, a mail message must be sent. Then the current file must be moved to the compare file for the

Author Name, email@address

next run. If the compare file does not exist, the file must be moved to the compare file and a message will be sent stating that monitoring has started. (Figure 17.)



```

1 if ((Get-PSSnapin | where {$_.name -like "quest*"}) -eq $null)
2 {
3     add-PSSnapin quest.activeroles.admanagement
4 }
5 $current = "c:\output\DomainAdmins.txt"
6 $compare = "c:\output\compare.txt"
7
8 Get-QADGroupMember "Domain Admins" | select -ExpandProperty name | out-file $current
9 if (Test-Path $compare)
10 {
11     $findings = Compare-Object (Get-Content $compare) (Get-Content $current)
12     if ($findings -ne $null)
13     {
14         Send-MailMessage -SmtpServer $server -To mikey@company.com -from ADMonitor@company.com -Subject "Group Changed!" -Body $findings
15     }
16     Move-Item $current $compare
17 }
18 else
19 {
20     Move-Item $current $compare -force
21     Send-MailMessage -SmtpServer $server -To mikey@company.com -from ADMonitor@company.com -Subject "Monitoring started" -Body
22 }
23 }

```

PS C:\Users\mikey>

```

> Get-QADGroupMember "Domain Admins" | select -ExpandProperty name | out-file $current

```

Ln 7 Col 1 | 12

**Figure 17. – Alert for Admin Changes**

The next step is to make this a recurring activity. There are multiple ways to do so, such as creating scheduled tasks for Windows as well as coding it into the script. This can be run in a continuous loop with a while loop (Figure 18.). The sleep at the end says wait 100 seconds, and this can be manipulated more if needed (Figure 18.).

```

1 if ((Get-PSSnapin | where {$_.name -like "quest*"}) -eq $null)
2 {
3     add-PSSnapin quest.activeroles.admanagement
4 }
5
6 while (1)
7 {
8     $current = "c:\output\DomainAdmins.txt"
9     $compare = "c:\output\compare.txt"
10
11     Get-QADGroupMember "Domain Admins" | select -ExpandProperty name | out-file $current
12     if (Test-Path $compare)
13     {
14         $findings = Compare-Object (Get-Content $compare) (Get-Content $current)
15         if ($findings -ne $null)
16         {
17             Send-MailMessage -SmtpServer $server -To mikey@company.com -from ADMonitor@company.com -Subject "Group Changed!" -Body $findings
18         }
19         Move-Item $current $compare
20     }
21     else
22     {
23         Move-Item $current $compare -force
24         Send-MailMessage -SmtpServer $server -To mikey@company.com -from ADMonitor@company.com -Subject "Monitoring started" -Body
25     }
26
27     Sleep 100|
28 }

```

PS C:\Users\mikey>  
>

Completed | Ln 27 Col 14 | 12

**Figure 18. – Add sleep**

If more groups need to be analyzed, then the script can be parameterized and functionalized.

#### 2.4.1. Monitoring Other Security Groups

By functionalizing (Figure 19.) the monitoring script code can be deployed in a myriad of ways.

```

1 function ADMonitor ($interim,$group) {
2     if ((Get-PSSnapin | where {$_.name -like "*quest*"}) -eq $null)
3     {
4         add-PSSnapin quest.activeroles.admanagement
5     }
6     if ((Get-QADGroup $group) -ne $null)
7     {
8         while (1)
9         {
10            $current = "c:\output\$group-new.txt"
11            $compare = "c:\output\$group-old.txt"
12
13            Get-QADGroupMember $group | select -ExpandProperty name | out-file $current
14            if (Test-Path $compare)
15            {
16                $findings = Compare-Object (Get-Content $compare) (Get-Content $current)
17                if ($findings -ne $null)
18                {
19                    Send-MailMessage -SmtpServer $server `
20                        -To mikey@company.com `
21                        -from ADMonitor@company.com `
22                        -Subject "Group Changed!" `
23                        -Body $findings
24                }
25                Move-Item $current $compare
26            }
27            else
28            {
29                Move-Item $current $compare -force
30                Send-MailMessage -SmtpServer $server `
31                    -To mikey@company.com `
32                    -from ADMonitor@company.com `
33                    -Subject "Monitoring started for $group"
34            }
35            Sleep $interim
36        }
37    }
38    else
39    {
40        exit
41    }
42 }
43 }

```

PS C:\Users\mikey>

Completed | Ln 5 Col 5 | 12

**Figure 19 – Security Group Monitoring**

Now run this code (Figure 19.) in a shell and kick off monitoring as needed.

```

PS:> Start-Job {ADMonitor -interim 100 -group "Domain Admins"}
PS:> Start-Job {ADMonitor -interim 100 -group "Enterprise Admins"}
PS:> Start-Job {ADMonitor -interim 100 -group "Schema Admins"}

```

The scripts run as a service and can be made to monitor multiple things. The framework can even be done to monitor locked accounts, using the `get-qaduser -locked` switch. Adding another script to the startup command creates a full feature interactive application server that can be monitored.

#### 2.4.2. PowerShell as a Syslog Server

Knowing an account is locked out is not as important as knowing why. This only works with monitoring security events and the best PowerShell code for monitoring event logs from a central location comes from Robert Sheldon, from the (Window IT Pro Article, 2008). The actual script was well designed, in that, there is not much need to change the code except for the SMTP information for the mail alerts (code Attached). The content of the referenced .txt and .csv files will need to be modified to indicate which systems need to be monitored and what security events you would like to pull from the systems.

```
Monitored_computers.txt:  
Domaincontrollersystemname1  
Domaincontrollersystemname2  
AllotherDCs
```

```
Alert_events.csv:  
Source,ID  
Microsoft-Windows-Security-Auditing,539  
Microsoft-Windows-Security-Auditing,644  
Microsoft-Windows-Security-Auditing,4740
```

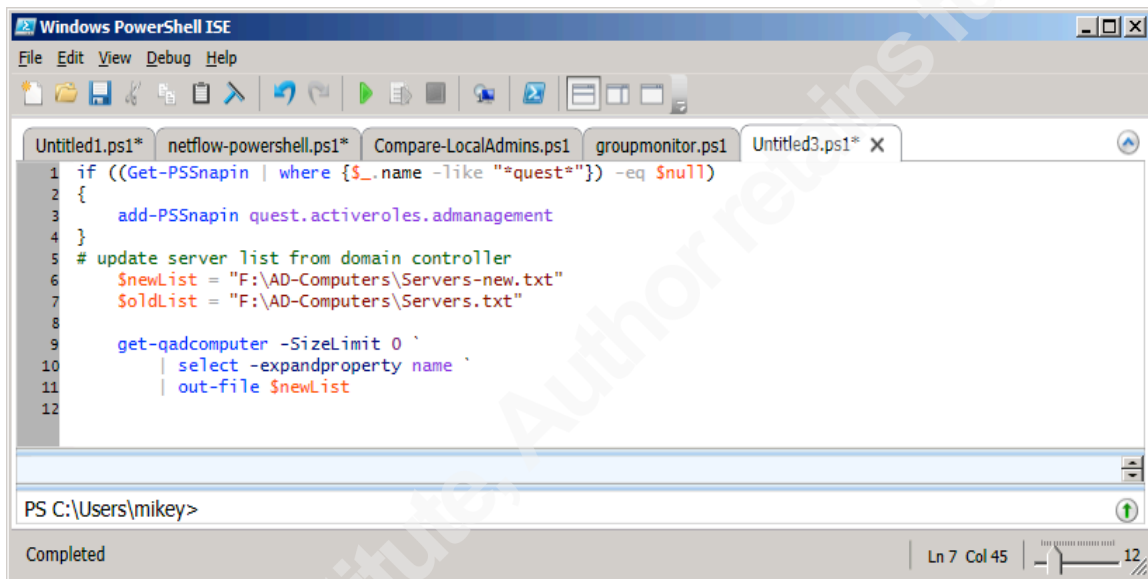
If the path is outlined for these in the “logmonitor” script and has the correct email information, an email will be sent with the log information every time an account locks out containing the system information and the reason that an account was locked out.

#### 2.4.3. Active Directory Monitoring

Now the ability to determine when security groups change, why security accounts get locked out, obtain network connection logs across the domain, and process the logs

Author Name, email@address

through multiple processes including regular expression pattern matching are all possible. Monitoring Domain Admin accounts is critical but monitoring local admin accounts can catch an attacker earlier in the exploitation cycle. This activity is one of the basic things an interactive attacker will do after he exploits a machine. The same techniques as in the previous examples can be utilized to detect this activity. An up to date list of systems needs to be acquired. This can also be done using the quest active directory cmdlets (Figure 20).



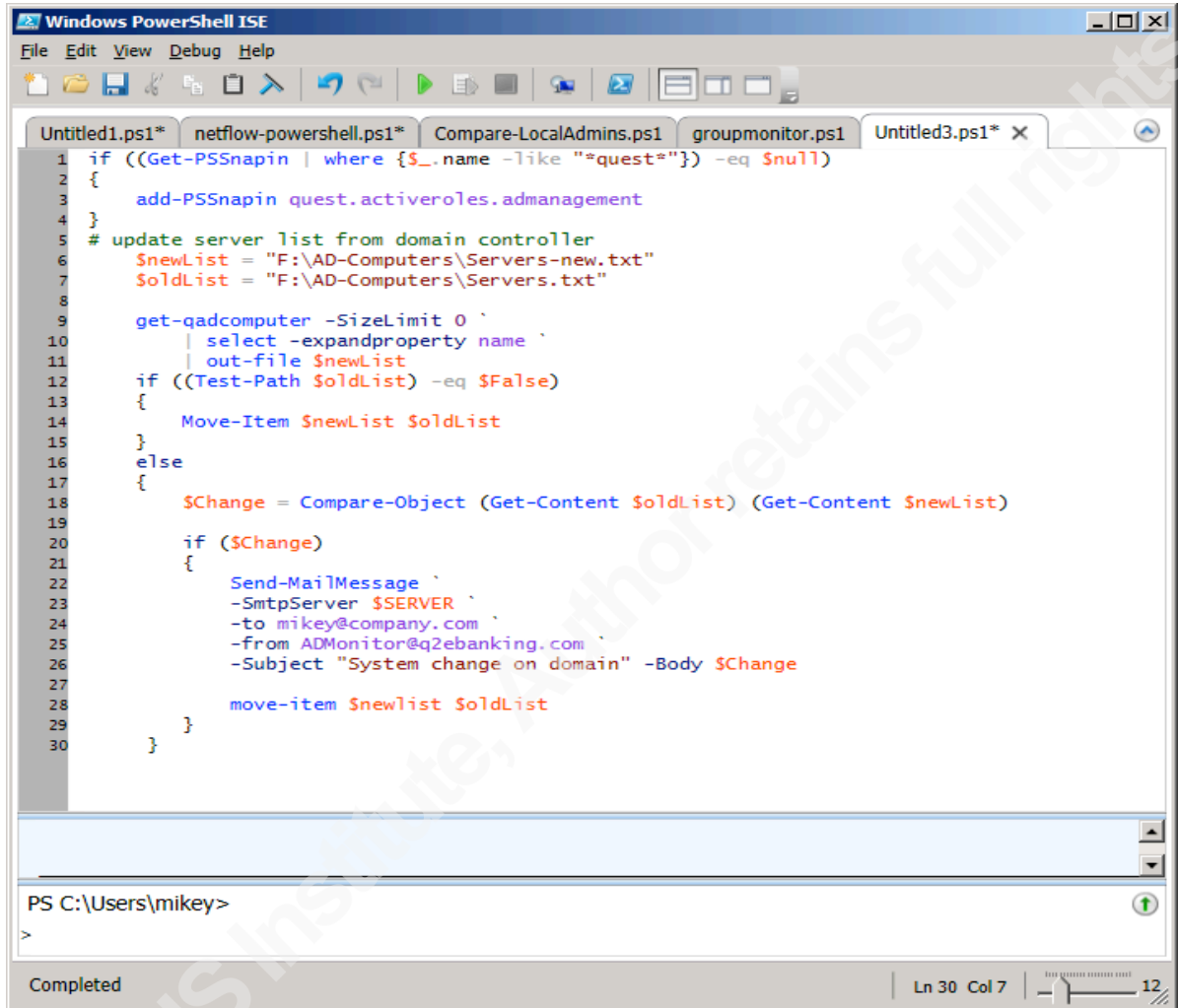
```

Windows PowerShell ISE
File Edit View Debug Help
Untitled1.ps1* netflow-powershell.ps1* Compare-LocalAdmins.ps1 groupmonitor.ps1 Untitled3.ps1* X
1 if ((Get-PSSnapin | where {$_.name -like "*quest*"} -eq $null)
2 {
3     add-PSSnapin quest.activeroles.admanagement
4 }
5 # update server list from domain controller
6 $newList = "F:\AD-Computers\Servers-new.txt"
7 $oldList = "F:\AD-Computers\Servers.txt"
8
9 get-qadcomputer -SizeLimit 0 `
10 | select -expandproperty name `
11 | out-file $newList
12
PS C:\Users\mikey>
Completed Ln 7 Col 45 12

```

Figure 20. – Get servers

This will generate a list systems on a domain as well as set up alerting (Figure 21):



```

1 if ((Get-PSSnapin | where {$_.name -like "*quest*"} -eq $null)
2 {
3     add-PSSnapin quest.activeroles.admanagement
4 }
5 # update server list from domain controller
6 $newList = "F:\AD-Computers\Servers-new.txt"
7 $oldList = "F:\AD-Computers\Servers.txt"
8
9 get-qadcomputer -SizeLimit 0 `
10 | select -expandproperty name `
11 | out-file $newList
12 if ((Test-Path $oldList) -eq $False)
13 {
14     Move-Item $newList $oldList
15 }
16 else
17 {
18     $Change = Compare-Object (Get-Content $oldList) (Get-Content $newList)
19
20     if ($Change)
21     {
22         Send-MailMessage `
23             -SmtpServer $SERVER `
24             -to mikey@company.com `
25             -from ADMonitor@q2ebanking.com `
26             -Subject "System change on domain" -Body $Change
27
28         move-item $newlist $oldList
29     }
30 }

```

PS C:\Users\mikey>

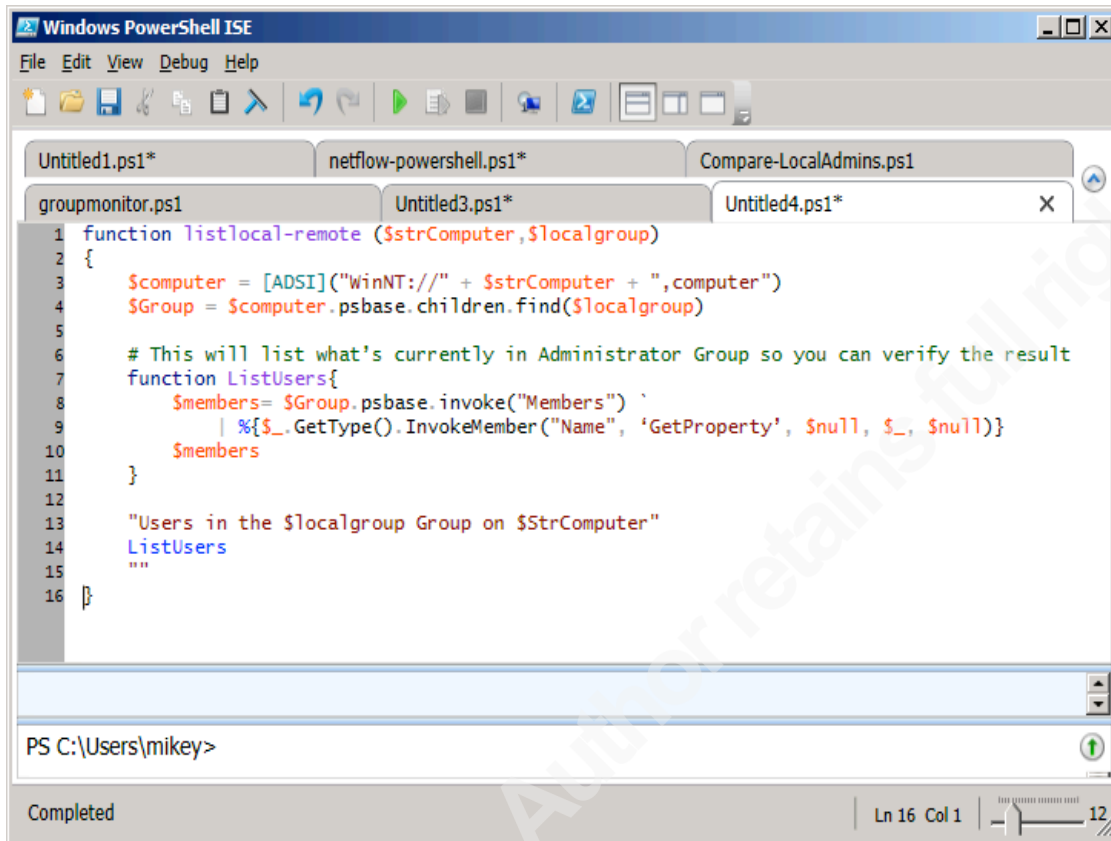
Completed | Ln 30 Col 7 | 12

Figure 21. – Monitor Systems added/removed AD

## 2.5. Local Security Group Monitoring

The next task is getting a list of administrators from a local group on a remote system. There are multiple methods, but a useful technique using the Active Directory Service Inquiry Object was outlined in 2008 on the Microsoft Blog Scripting Guys. The following function using the [ADSI] objects will list members of local security groups on a system (ScriptingGuy1, 2008) (Figure 22):

Author Name, email@address



```

1 function listlocal-remote ($strComputer,$localgroup)
2 {
3     $computer = [ADSI]("WinNT://" + $strComputer + ",computer")
4     $Group = $computer.psbase.children.find($localgroup)
5
6     # This will list what's currently in Administrator Group so you can verify the result
7     function ListUsers{
8         $members= $Group.psbase.invoke("Members") `
9             | %{$_.GetType().InvokeMember("Name", 'GetProperty', $null, $_, $null)}
10        $members
11    }
12
13    "Users in the $localgroup Group on $strComputer"
14    ListUsers
15    ""
16 }

```

PS C:\Users\mikey>

Completed | Ln 16 Col 1 | 12

**Figure 22. – [ADSI] – get local users**

By running:

```
PS > listlocal-remote -strComputer $systemnameorIP -localgroup Administrators
```

A list of local admins from a remote system can be gathered. Using the same logic as earlier to compare old and new output from files detecting changes can be possible (Figure 23):

```

48
49 #Alright - let's get some local admins
50 $servers = Get-Content $oldList
51 foreach ($server in $servers)
52 {
53     $newAdmins = "F:\AD-Computers\LA\$server-localadmins-new.txt"
54     $oldAdmins = "F:\AD-Computers\LA\$server-localadmins.txt"
55
56
57     listlocal-remote -strComputer $server -localgroup Administrators | out-file $newAdmins
58
59     if ((test-path $oldAdmins) -eq $false)
60     {
61         Move-Item $newAdmins $oldAdmins
62     }
63     elseif ((Test-Path $newAdmins) -eq $false)
64     {
65         Out-Null
66     }
67     else
68     {
69         $LChange = Compare-Object -old $oldAdmins -new $newAdmins -msg $server
70         if ($LChange)
71         {
72             Send-MailMessage `
73                 -smtpServer $server `
74                 -to mikey@company.com `
75                 -from LAMonitor@company.com `
76                 -Subject "Local Administrators account changed on $server" -Body $LChange
77         }
78         Move-Item $newAdmins $oldAdmins
79     }
80 }
81

```

PS C:\Users\mikey>

Completed | Ln 65 Col 21 | 12

**Figure 23. – Get local admins script and compare**

The full script is attached in Appendix A, which will notify via a specified email when a local admin group changes on a system on the domain. A startup task or a sleep with the start-job/service can be created to keep the monitoring script running. This will ensure that when an admin security group changes a notification is sent to track down why and when it changed.

### 3. Conclusion

Windows environments have historically fell short in the automation world because of the lack of a full-featured scripting language. Now this disparity has been reduced with the advent of PowerShell. PowerShell was designed as an administrative language however it has tremendous capability in regards to Security scripting and monitoring. The only limit that exists is the Analyst's imagination.

Author Name, email@address

In this paper, it was demonstrated how PowerShell shell commands can be used to analyze logs and windows systems using several cmdlets as well as how to create some monitoring that has been traditionally lacking in Windows Systems. Modules to expand its capability such as the Quest Active Directory cmdlets and some techniques to compare changes from one moment to another was demonstrated. The critical task monitoring of monitoring privileged accounts across the domain was proved possible.

The method for centralizing logs across the domain from the Windows firewall in order to get the proper logs into one place for the network connections for analysis was outlined. This coupled with monitoring of other logs is critical for analysis and detection.

Historically intrusion analysts have depended on tools for the identification and interpretation of this type of information, and there are lots of tools out there that will do this type of monitoring/analysis. They all are expensive, and for a small to medium sized shop that needs to monitor this information, Microsoft has provided the tools to monitor and extrapolate this type of information. The only additional task is the dedication and the will to accomplish the task.

## 4. References

- Callahan ,John, (2002), “Microsoft briefly stopped Windows development in February 2002 to focus on security”, NEOWIN, Retrieved From:  
<http://www.neowin.net/news/microsoft-briefly-stopped-windows-development-in-february-2002-to-focus-on-security>
- Dell Software [SOFTWARE], (2014), Quest Active Directory Cmdlets, Retrieved from:  
<http://www.quest.com/powershell/activeroles-server.aspx>
- Dumont, Cody, (2012), “Auditing Windows Environment xml-output security OSSAMS”, SANS Reading Room, Retrieved from:  
<http://www.sans.org/reading-room/whitepapers/auditing/auditing-windows-environments-powershell-xml-output-windows-security-ossams-33854>
- Gates, Bill, (2012), “Memo from Bill Gates”, Retrieved from:  
<http://www.microsoft.com/en-us/news/features/2012/jan12/gatesmemo.aspx>
- Goerlich, Wolfgang, (2013), “Incident Management in PowerShell: Identification”, PosHSEC, Retrieved from: <http://www.poshsec.com/2013/06/incident-management-in-powershell-identification/>,
- Hartman, Kenneth G., (2014), “PowerShell Script to Log Network Connections”, Retrieved from: <http://www.kennethghartman.com/log-connections-powershell-script/>
- Hicks, et al (2013), PowerShell Deep Dives, Manning Publication, Manning Publications, Co., 464 pages ISBN: 9781617291319
- Roric, (2012) “Central Monitor (PowerShell Event Log Email Reporter)”, Spiceworks.com, Retrieved from:  
<http://community.spiceworks.com/scripts/show/1714-central-monitor-powershell-event-log-email-reporter>
- ScriptingGuy1, (12 Mar 2008), Hey, Scripting Guy! How Can I Use Windows PowerShell to Add a Domain User to a Local Group?, blogs.technet.com, Retrieved from:  
<http://blogs.technet.com/b/heyscriptingguy/archive/2008/03/11/how-can-i-use-windows-powershell-to-add-a-domain-user-to-a-local-group.aspx>

Author Name, email@address

Unknown, (2013), Microsoft TechNet, Retrieved from: <http://technet.microsoft.com/en-us/library/bb978526.aspx>

Unknown, (2014), “Identifying Incidents Using Firewall and Cisco IOS Router Syslog Events”,

Retrieved form: <http://www.cisco.com/web/about/security/intelligence/identify-incidents-via-syslog.html>

Unknown, (2014), “OS Platform Statistics, What is the Trend in Operation Systems usage”, w3schools, Retrieved from:

[http://www.w3schools.com/browsers/browsers\\_os.asp](http://www.w3schools.com/browsers/browsers_os.asp),

Sheldon, Robert, (2008), “PowerShell Makes Security Log Access Easy”, Windows IT Pro, Retrieved from: <http://windowsitpro.com/powershell/powershell-makes-security-log-access-easy>,

Swift, David, (2011), “A Process for Continuous Improvement Using Log Analysis”, SANS Reading Room, Retrieved from: <http://www.sans.org/reading-room/whitepapers/awareness/process-continuous-improvement-log-analysis-33824>

Williams, Cory, (2011), “Count Uniq”, Command Line Kung Fu, <http://blog.commandlinekungfu.com/search?q=count+uniq>

Wilson, Ed (2013), Windows PowerShell 3.0 First Steps, Microsoft Press, p. 223

Author Name, email@address

## 5. Appendix A

Log Monitor:

<http://community.spiceworks.com/scripts/show/1714-central-monitor-powershell-event-log-email-reporter> Central Monitor (PowerShell Event Log Email Reporter)  
Roric Dec 13, 2012 at 11:29 AM

```
#####START SCRIPT#####
param([switch]$ShowEvents = $false,[switch]$NoEmail = $false,[switch]$useinstanceid = $false)

$log = "Security"
$hist_file = $log + "_loghist.xml"
$seed_depth = 100

#run interval in minutes - set to zero for runonce, "C" for 0 delay continuous loop.
$run_interval = 5

$emailFrom = "SecurityLogMonitor@company.com"
$emailTo = "SecurityAlerts@company.com"
$emailTo2 = "L1ITHelpdesk@company.com"
$emailSubject = "SecurityAlert"

$SMTPServer = "server.mail.com"
$SMTPAuthUsername = "username"
$SMTPAuthPassword = "password"

$computers = @(get-content F:\output\monitored_computers.txt)
$event_list = @{}
Import-Csv "F:\output>alert_events.csv" |% {$event_list[$_source + '#' + $_id] = 1}
#see if we have a history file to use, if not create an empty $histlog
if (Test-Path $hist_file){$loghist = Import-Clixml $hist_file}
else {$loghist = @{}}
$timer = [System.Diagnostics.Stopwatch]::StartNew()
function send_email {
$mailmessage = New-Object system.net.mail.mailmessage
$mailmessage.from = ($emailfrom)
$mailmessage.To.add($emailto)
$mailmessage.To.Add($emailto2)
$mailmessage.Subject = $emailsubject
$mailmessage.Body = $emailbody
$mailmessage.IsBodyHTML = $true
$SMTPClient = New-Object Net.Mail.SmtpClient($SMTPServer, 25)
$SMTPClient.Credentials = New-Object System.Net.NetworkCredential("$SMTPAuthUsername", "$SMTPAuthPassword")
$SMTPClient.Send($mailmessage)
}
#START OF RUN PASS
$run_pass = {

$emailBody = "Log monitor found monitored events. `n"

$computers |%{
$timer.reset()
$timer.start()

Write-Host "Started processing $($_) "

#Get the index number of the last log entry
$index = (Get-EventLog -ComputerName $_ -LogName $log -newest 1).index
```

Author Name, email@address

```

#If we have a history entry calculate number of events to retrieve
# if we don't have an event history, use the $Seed_depth to do initial seeding
if ($loghist[$_]){ $n = $index - $loghist[$_] }
else { $n = $Seed_depth }

if ($n -lt 0){
  Write-Host "Log index changed since last run. The log may have been cleared. Re-seeding index."
  $events_found = $true
  $EmailBody += "`n Possible Log Reset $($_) `n Event Index reset detected by Log Monitor `n" | ConvertTo-Html
  $n = $Seed_depth
}

Write-Host "Processing $($n) events."

#get the log entries

if ($useinstanceid){
  $log_hits = Get-EventLog -ComputerName $_ -LogName $log -Newest $n |
  ? {$Sevent_list[$_.source + "#" + $_.instanceid]}
}

else {$log_hits = Get-EventLog -ComputerName $_ -LogName $log -Newest $n |
? {$Sevent_list[$_.source + "#" + $_.eventid]}
}

#save the current index to $loghist for the next pass
$loghist[$_] = $index

#report number of alert events found and how long it took to do it
if ($log_hits){
  $events_found = $true
  $hits = $log_hits.count
  $EmailBody += "`n Alert Events on server $($_) `n"
  $log_hits | % {
    $Emailbody += "<br><br>"
    $Emailbody += $_ | select MachineName,EventID,Message | ConvertTo-Html
    $Emailbody += "<br><br>"
  }
}
else {$hits = 0}
$duration = ($timer.elapsed).totalseconds
write-host "Found $($hits) alert events in $($duration) seconds."
"-"*60
" "

if ($ShowEvents){ $log_hits | fl | Out-String | ? {$_.} }
}

#save the history file to disk for next script run
$loghist | export-clixml $hist_file

#Send email if there were any monitored events found
if ($events_found -and -not $NoEmail){ send_email }
}

#END OF RUN PASS

Write-Host "`n($"*60)"
Write-Host "Log monitor started at $(get-date)"
Write-Host "$($"*60)`n"

#run the first pass
$start_pass = Get-Date
&$run_pass

#If $run_interval is set, calculate how long to sleep before the next pass
while ($run_interval -gt 0){
if ($run_interval -eq "C"){ &$run_pass }
else{
  $last_run = (Get-Date) - $start_pass
  $sleep_time = ([TimeSpan]::FromMinutes($run_interval) - $last_run).totalseconds
}
}

```

Author Name, email@address

```
Write-Host "`n$( "*" * 10) Sleeping for $($sleep_time) seconds `n"

#sleep, and then start the next pass
Start-Sleep -seconds $sleep_time
$start_pass = Get-Date
&$run_pass
}
}

#####END SCRIPT#####
```

Code written for this paper:

Get top ten talkers in a log file:

```
#####Start Script#####

select-string "\b(?:\d{1,3}\.){3}\d{1,3}\b" .logs.txt |
select -ExpandProperty matches |
select value | `
group value | `
sort count -des

#####End Script#####
```

Get Windows Firewall Logs and pipe to central location:

```
#####Start Script#####

$date = Get-Date -Format yyyyMMdd
$FWLog = (netsh advfirewall show allprofiles |
Select-String FileName |
select -ExpandProperty line |
Select-String "%systemroot%.+\.log" |
select -ExpandProperty matches |
select -ExpandProperty value |
sort -uniq)
$FWLog = $FWLog -replace "%systemroot%", "C:\Windows"
select-string "\b(?:\d{1,3}\.){3}\d{1,3}\b" $FWLog |
add-content "\\secureshare\logs\$date-$env:computername-FWLogs.log"

#####End Script#####
```

Change Local admin (in case you don't want to just monitor)

```
#####Start Script #####

# Example:
# ChangeLA -strComputer mweeks -username mweeks -add
# ChangeLA -strComputer mweeks -username mweeks -remove
# changeLA -strComputer mweeks

Function ChangeLA([string]$strComputer, [switch]$add, [switch]$remove, [string]$username)
{
    $DOMAIN = $env:USERDNSDOMAIN
    if ($strComputer -eq "")
    {
```

Author Name, email@address

```

    "Please specify a computername, IP or system name with ps:> changeLA -strComputer test-lt"
    break
}

$computer = [ADSI]("WinNT://" + $strComputer + ",computer")
#$computer.name

#$strComputer

$Group = $computer.psbases.children.find("administrators")
#$Group.name

# This will list what's currently in Administrator Group so you can verify the result
function ListAdministrators ($members= $Group.psbases.invoke("Members") | % {$_.GetType().InvokeMember("Name",
'GetProperty', $null, $_, $null)}
$members)

"Current Administrators on $strComputer"
ListAdministrators
" `n"
if ($username -eq "")
{
    out-null
}
else
{
    if ($remove -ne $false)
    {
        $Group.Remove("WinNT://" + $domain + "/" + $username)
        "Administrators after script:"
        ListAdministrators
        "`n"
    }

    elseif ($add -ne $false)
    {
        $Group.add("WinNT://" + $domain + "/" + $username)
        "Administrators after script:"
        ListAdministrators
        "`n"
    }
    else
    {
        "what do you want me to do? please specify with changeLA -strComputer test-lt -username test -add"
    }
}

}
$username = ""
$strComputer = ""
$add = $false
$remove = $false
}

#####End Script #####

```

ADMonitor – Monitor any Security group in AD and shoot an email – will have to change email code

```

#####Start Script #####

function ADMonitor ($interim,$group) {
    if ((Get-PSSnapin | where {$_.name -like "*quest*"} -eq $null)
    {

```

Author Name, email@address

```

add-PSSnapin quest.activeroles.admanagement
}
if ((Get-QADGroup $group) -ne $null)
{
  while (1)
  {
    $current = "c:\output\$group-new.txt"
    $compare = "c:\output\$group-old.txt"

    Get-QADGroupMember $group | select -ExpandProperty name | out-file $current
    if (Test-Path $compare)
    {
      $findings = Compare-Object (Get-Content $compare) (Get-Content $current)
      if ($findings -ne $null)
      {
        Send-MailMessage -SmtpServer $server `
          -To mikey@company.com `
          -from ADMonitor@company.com `
          -Subject "Group Changed!" `
          -Body $findings
      }
      Move-Item $current $compare
    }
    else
    {
      Move-Item $current $compare -force
      Send-MailMessage -SmtpServer $server `
        -To mikey@company.com `
        -from ADMonitor@company.com `
        -Subject "Monitoring started for $group"
    }

    Sleep $interim
  }
}
else
{
  exit
}
}

#####END SCRIPT #####

```

Author Name, email@address

LocalAdminMonitor – monitor across the domain and all changes to the domain.

```
#####START SCRIPT #####
if ((Get-PSSnapin | where {$_.name -like "*quest*"}) -eq $null)
{
    add-PSSnapin quest.activeroles.admanagement
}
# update server list from domain controller
$newList = "F:\AD-Computers\Servers-new.txt"
$soldList = "F:\AD-Computers\Servers.txt"

get-qadcomputer -SizeLimit 0 `
| select -expandproperty name `
| out-file $newList
if ((Test-Path $soldList) -eq $false)
{
    Move-Item $newList $soldList
}
else
{
    $Change = Compare-Object (Get-Content $soldList) (Get-Content $newList)

    if ($Change)
    {
        Send-MailMessage `
        -SmtpServer $SERVER `
        -to mikey@company.com `
        -from ADMonitor@q2ebanking.com `
        -Subject "System change on domain" -Body $Change

        move-item $newlist $soldList
    }
}
#function to acquire local admins:
function listlocal-remote ($strComputer,$localgroup)
{
    $computer = [ADSI]("WinNT://" + $strComputer + ",computer")
    $Group = $computer.psbase.children.find($localgroup)

    # This will list what's currently in Administrator Group so you can verify the result
    function ListUsers{
        $members= $Group.psbase.invoke("Members") `
        | %{$_.GetType().InvokeMember("Name", 'GetProperty', $null, $_, $null)}
        $members
    }

    "Users in the $localgroup Group on $strComputer"
    ListUsers
    ""
}
#Alright - let's get some local admins
$servers = Get-Content $soldList
foreach ($server in $servers)
{
    $newAdmins = "F:\AD-Computers\LAs\$server-localadmins-new.txt"
    $soldAdmins = "F:\AD-Computers\LAs\$server-localadmins.txt"

    listlocal-remote -strComputer $server -localgroup Administrators | out-file $newAdmins

    if ((test-path $soldAdmins) -eq $false)
    {
        Move-Item $newAdmins $soldAdmins
    }
    elseif ((Test-Path $newAdmins) -eq $false)
    {

```

Author Name, email@address

```
Out-Null
}
else
{
    $LChange = Compare-Object -old $oldAdmins -new $newAdmins -msg $server
    if ($LChange)
    {
        Send-MailMessage `
            -SmtpServer $server `
            -to mikey@company.com `
            -from LAMonitor@company.com `
            -Subject "Local Administrators account changed on $server" -Body $LChange

    }
    Move-Item $newAdmins $oldAdmins
}
}
#####END SCRIPT #####
```

Author Name, email@address