**DEFENDERS HANDBOOK**

Contents

[Networking 4](#_Toc149221645)

[Common Ports 4](#_Toc149221646)

[TCP/IP Header 5](#_Toc149221647)

[IPv6 7](#_Toc149221648)

[OSI vs TCP/IP Model 9](#_Toc149221649)

[Cisco 11](#_Toc149221650)

[Commands 11](#_Toc149221651)

[Router information 12](#_Toc149221652)

[Access Control List 15](#_Toc149221653)

[Juniper 16](#_Toc149221654)

[Juniper hardware CLI – Junos. 16](#_Toc149221655)

[PFSense (empty) 28](#_Toc149221656)

[Host 29](#_Toc149221657)

[Windows 29](#_Toc149221658)

[PowerShell 29](#_Toc149221659)

[Group Policy Object 36](#_Toc149221660)

[Active Directory 36](#_Toc149221661)

[Linux 46](#_Toc149221662)

[Bash 49](#_Toc149221663)

[Incident Response 52](#_Toc149221664)

[IR / HUNT Flowchart 52](#_Toc149221665)

[DCI Mission Phases 52](#_Toc149221666)

[Mission Planning Phase: 52](#_Toc149221667)

[Survey Phase: 53](#_Toc149221668)

[Secure Phase: 54](#_Toc149221669)

[Protect Phase: 55](#_Toc149221670)

[Recover Phase: 56](#_Toc149221671)

[Initial DCI investigation methodology 56](#_Toc149221672)

[Actions On 57](#_Toc149221673)

[Suspicious Process and/or File 57](#_Toc149221674)

[Suspicious Traffic 58](#_Toc149221675)

[Suspicious E-Mail 59](#_Toc149221676)

[Malicious Binary/Process 59](#_Toc149221677)

[Malicious Network Activity 60](#_Toc149221678)

[Malicious E-Mail 61](#_Toc149221679)

[Linux commands for IR 62](#_Toc149221680)

[Vulnerability Assessment (empty) 71](#_Toc149221681)

[Important Registry Keys 71](#_Toc149221682)

[Scripts 71](#_Toc149221683)

[Remote connections 71](#_Toc149221684)

[Enumeration Script 71](#_Toc149221685)

[Baseline Script 76](#_Toc149221686)

[Baseline Light Windows Host 81](#_Toc149221687)

[Baseline Wrapper 82](#_Toc149221688)

[AD Account collection 82](#_Toc149221689)

[Linux Host Enumeration 83](#_Toc149221690)

[Compare Script 100](#_Toc149221691)

[Compare File Hash 100](#_Toc149221692)

[Portable Executable Injection 100](#_Toc149221693)

[Hunt Script 101](#_Toc149221694)

[Disable Weak SSL 105](#_Toc149221695)

[DNS Zone Transfer Alert 110](#_Toc149221696)

[Unusual User Sign-In Alert 110](#_Toc149221697)

[Watch Port Scanning 114](#_Toc149221698)

[Set Attack Surface Reduction Rule 115](#_Toc149221699)

[Enable Audit Policy for CA 116](#_Toc149221700)

[Enable Auto Enrolment Logging 117](#_Toc149221701)

[Get Users and Cert Count 117](#_Toc149221702)

[SMB Encryption 118](#_Toc149221703)

[Get Servers Features 119](#_Toc149221704)

[Update Hosts file 120](#_Toc149221705)

[Group Managed Service Accounts 124](#_Toc149221706)

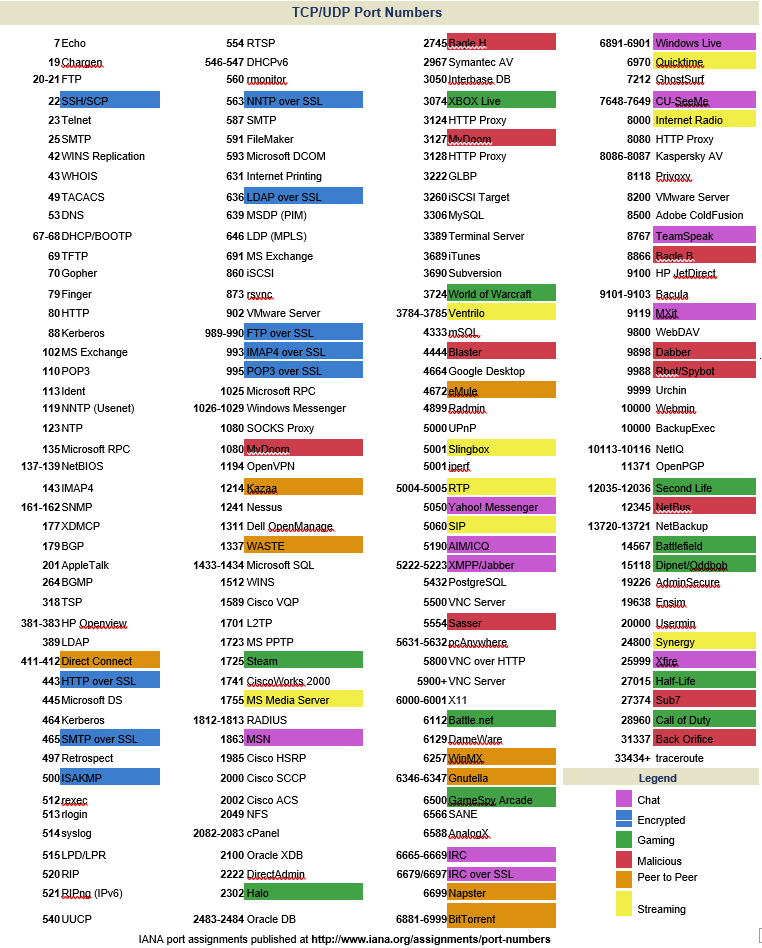
[Get Service Privileges 125](#_Toc149221707)

[Hashing Script 125](#_Toc149221708)

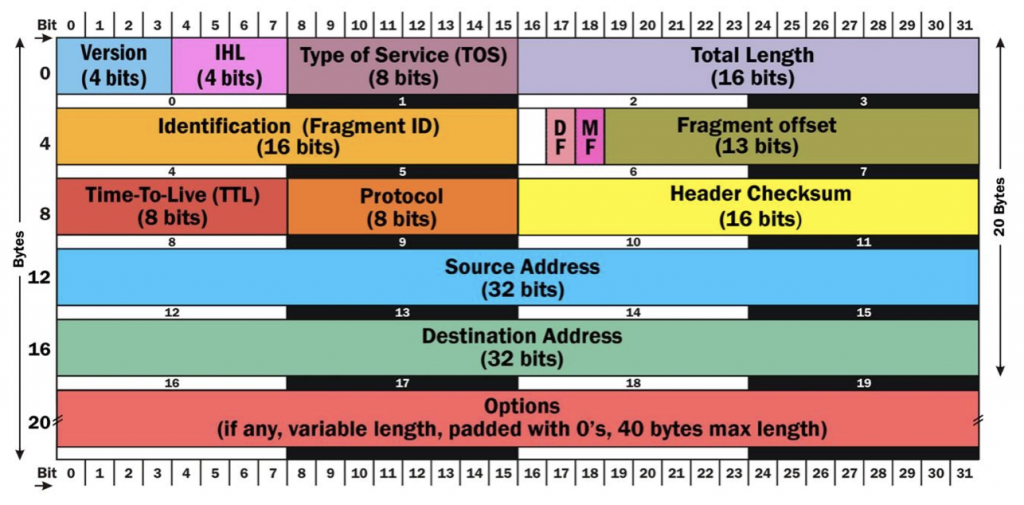
[Other documents 126](#_Toc149221709)

# Networking

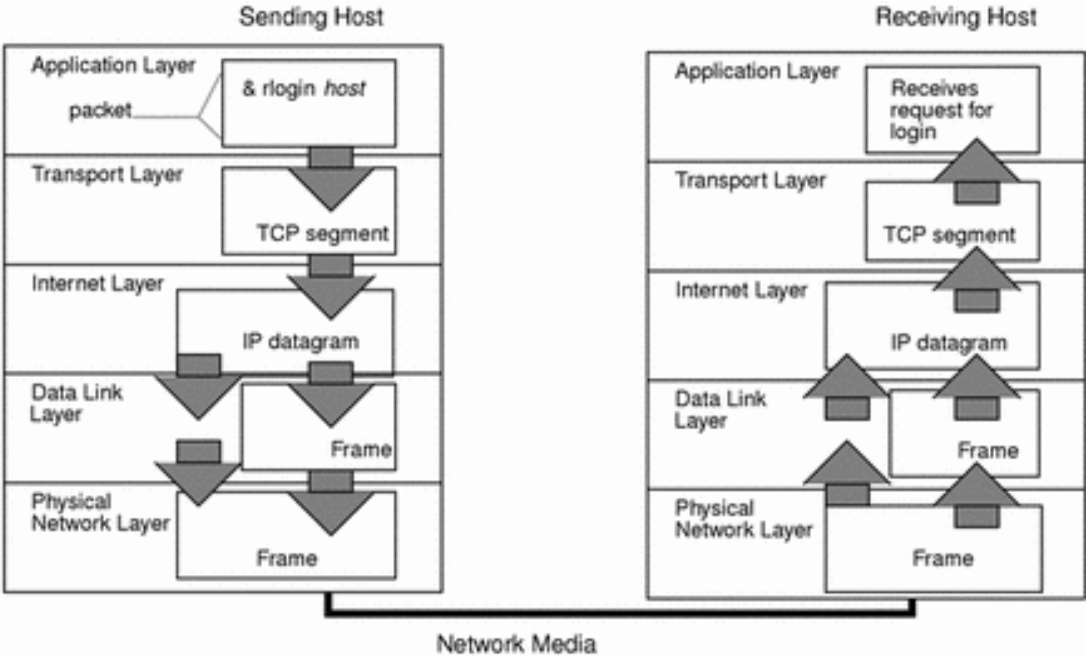
## Common Ports



# TCP/IP Header



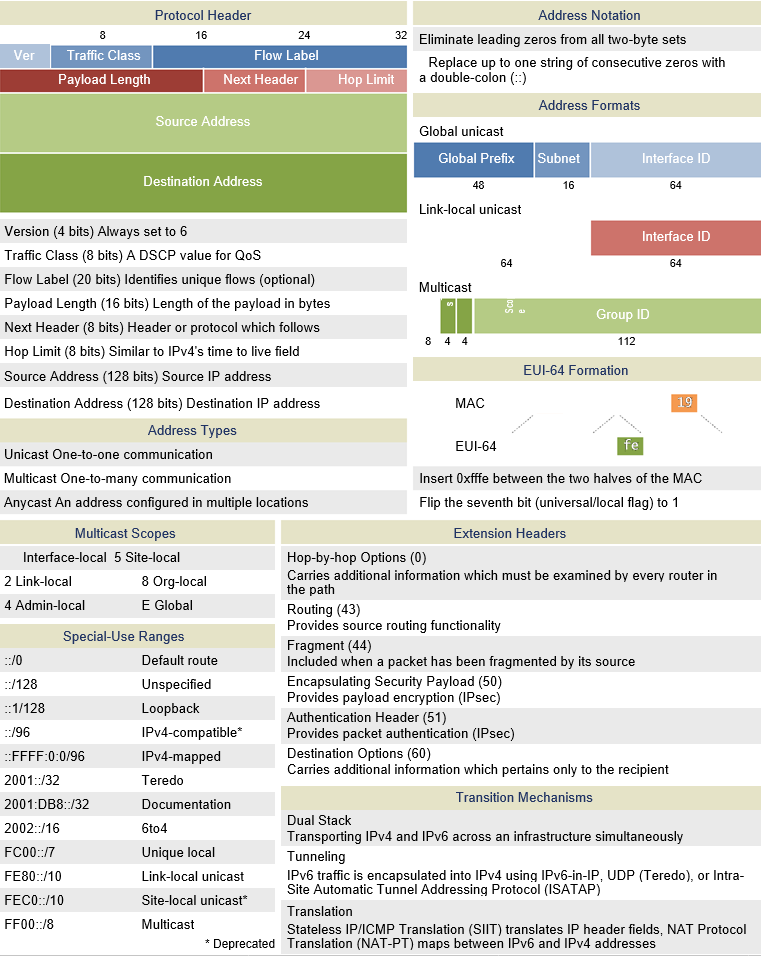
**H** **ow a packet travels through the TCP/IP Stack**

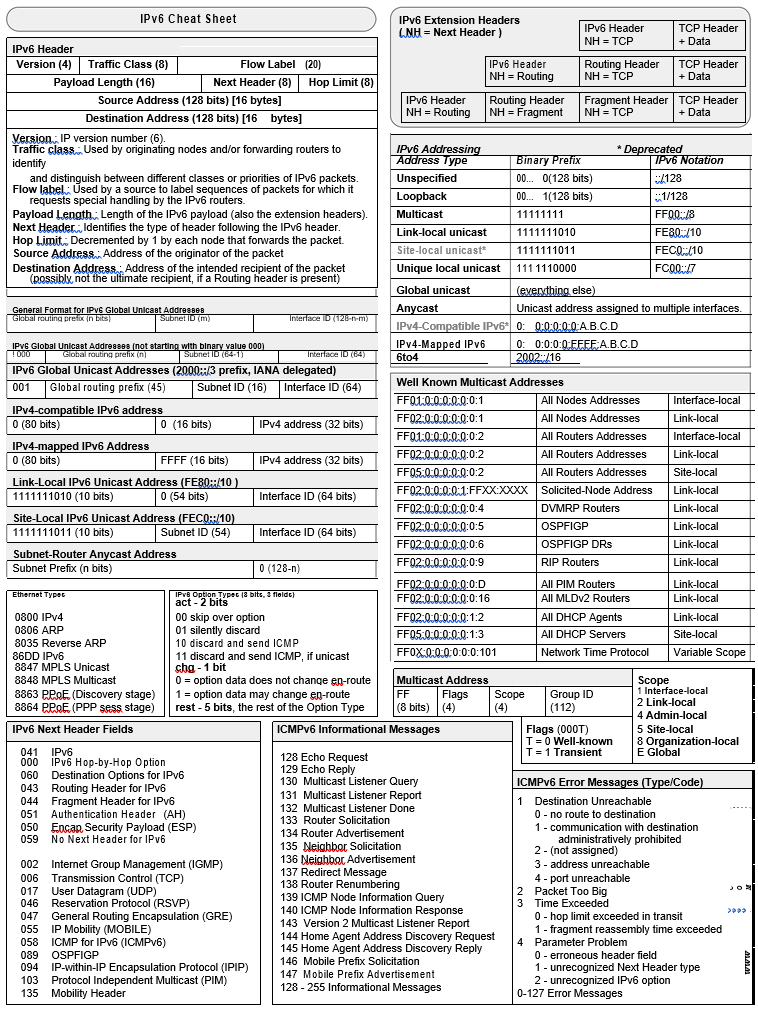


IPv4 Subnetting

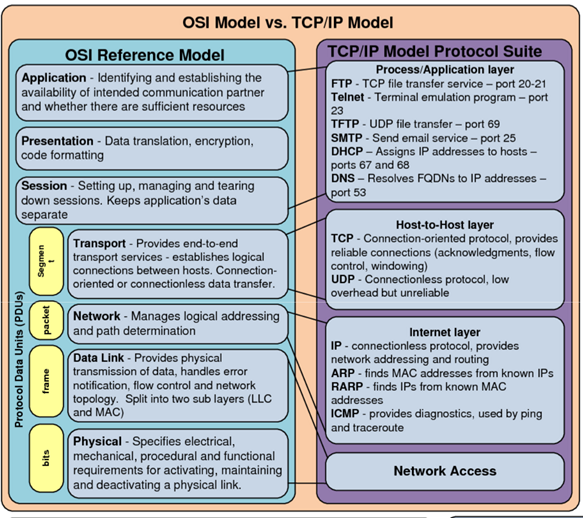


## IPv6





## OSI vs TCP/IP Model



OSI Protocol levels

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Layer | OSI protocols | | Responsibilities | | Scope | TCP/IP |
|  |  | Model |
|  |  |  |  |  |  |  |  |
| # |  | Name |  |  |  |  |  |  |
|  |  |  |  | NNTP, SIP, SSI, DNS, FTP, |  |  |  |  |
|  |  |  |  | Gopher, HTTP, NFS, NTP, |  |  |  |  |
|  |  |  |  | DHCP, SMPP, SMTP, |  |  |  |  |
|  |  |  | FTAM, X.400, | SNMP, Telnet, RIP, BGP, |  |  |  |  |
|  |  |  | X.500, DAP, ROSE, | BOOTP, TFTP, POP3, | User Applications |  |  |  |
|  | 7 | Application | RTSE, ACSE | IMAP, | Services | User Data | Application Data |  |
|  |  |  | ISO/IEC 8823, |  |  |  |  | Application |
|  |  |  |  |  |  |  |  |
|  |  |  | X.226, ISO/IEC 9576- | MIME, SSL, TLS, XDR, |  |  |  |  |
|  | 6 | Presentation | 1, X.236 | Shells and Redirectors |  |  |  |  |
|  |  |  |  | Sockets. Session |  |  |  |  |
|  |  |  | ISO/IEC 8327, | establishment in TCP, SIP, | Session Establishment, |  | Sessions between |  |
|  |  |  | X.225, ISO/IEC 9548- | RTP, NetBIOS, RPC, | Management and |  | local or remote |  |
|  | 5 | Session | 1, X.235 | Named Pipes | Termination | Sessions | devices |  |
|  |  |  |  |  | Process-Level Addressing; |  |  |  |
|  |  |  |  |  | Multiplexing/Demultiplex |  |  |  |
|  |  |  |  |  | ing; Connections; |  |  | Transport |
|  |  |  |  |  | Segmentation and |  |  |
|  |  |  |  |  |  |  | (TCP) |
|  |  |  | ISO/IEC 8073, TP0, |  | Reassembly; |  |  |
|  |  |  |  |  |  |  |
|  |  |  | TP1, TP2, TP3, TP4 |  | Acknowledgments and |  |  |  |
|  |  |  | (X.224), |  | Retransmissions; Flow | Datagrams/ |  |  |
|  | 4 | Transport | ISO/IEC 8602, X.234 | TCP, UDP, SCTP | Control | Segments |  |  |
|  |  |  |  | IP, IPsec, ICMP, IGMP, |  |  |  |  |
|  |  |  |  | OSPF, IPv6; IP NAT; IPsec; |  |  |  |  |
|  |  |  | ISO/IEC 8208, X.25 | Mobile IP; ICMP; IPX; |  |  |  | Internet |
|  |  |  | (PLP), ISO/IEC 8878, | DLC; PLP; Routing |  |  |  | (IP) |
|  |  |  | X.223, ISO/IEC 8473- | protocols such as RIP and |  |  |  |  |
|  | 3 | Network | 1, CLNP X.233. | BGP |  |  |  |  |
|  |  |  | ISO/IEC 7666, X.25 |  |  |  | Low-level data |  |
|  |  |  | (LAPB), Token Bus, |  |  |  | messages |  |
|  |  |  | X.222, ISO/IEC 8802- |  |  |  | between local |  |
|  | 2 | Data Link | 2 LLC Type 1 and 2 | PPP, SLIP, PPTP, L2TP |  |  | devices | Network |
|  |  |  |  |  | Encoding and Signaling; |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | X.25 (X.21bis, |  | Physical Data |  | Electrical or light |  |
|  |  |  | EIA/TIA-232, |  | Transmission; Hardware |  | signals sent |  |
|  |  |  | EIA/TIA-449, EIA- |  | Specifications; Topology |  | between local |  |
|  | 1 | Physical | 530, G.703) |  | and Design | Bits | devices |  |
|  |  |  |  |  |  |  |  |  |

## Cisco

### Commands

*Below are basic Cisco commands for configuring, securing and troubleshooting Cisco network devices.*

**Basic configuration commands**

# **Enable** - Logs you into enable mode, which is also known as user exec mode or privileged mode.

# **configure terminal -** Logs you into configuration mode.

# **Interface***fastethernet/number (e.g. interface g/0/0/0) -* Enters interface configuration mode for the specified fast Ethernet interface.

# **reload** - An exec mode command that reboots a Cisco switch or router.

# **hostname***name -* Sets a host name to the current Cisco network device.

# **copy running-config startup-config -** An enable mode command that saves the active config, replacing the startup config when a Cisco network device initializes.

**# ip address***ip-address mask -* Assigns an IP address and a subnet mask.

# **show running-config -** An enable mode command that displays the current configuration.

# **show ip interface brief** - Displays the usability status of interfaces that are configured for IP.

**Troubleshooting commands**

# **ping** *ip address* - Used in enable mode to diagnose basic network connectivity.

# **show mac address-table** - Displays the MAC address table.

# **show cdp neighbor** - Lists summary information about each neighbor connected to this device; the “detail” option lists detailed information about each neighbor.

# **show interfaces** - Displays detailed information about interface status, settings and counters.

# **show interface status** - Displays the interface line status.

# **show interfaces switchport -** Displays a large variety of configuration settings and current operational status, including VLAN trunking details.

# **show vlan brief** - Lists each VLAN and all interfaces assigned to that VLAN but does not include trunks.

**Monitoring and logging commands**

**# logging***ip address -* Configures the IP address of the host that will receive the system logging (syslog) messages.

**# Logging trap level** - Used in configuration mode to limit messages that are logged to the syslog servers based on severity. Specify the number or name of the desired severity level at which messages should be logged.

**# show logging** - Enable mode command that displays the state of system logging (syslog) and the contents of the standard system logging buffer.

**# terminal monitor** - An enable mode command that tells Cisco IOS to send a copy of all syslog messages, including debug messages, to the Telnet or SSH user who issues this command.

**Security commands (including ACL configuration)**

**# password** *pass-value -* Lists the password that is required if the login command (with no other parameters) is conﬁgured.

# **username***name***password***pass-value -* A global command that deﬁnes one of possibly multiple user names and associated passwords used for user authentication. It is used when the login local line conﬁguration command has been used.

# **enable password***pass-value -* A configuration mode command that deﬁnes the password required when using the **enable** command.

# **enable secret***pass-value -* A configuration mode command that sets this Cisco device password that is required for any user to enter enable mode.

# **service password-encryption -** A configuration mode command that directs the Cisco IOS software to encrypt the passwords, CHAP secrets, and similar data saved in its configuration file.

# **show port security***[interface interface-id] -* Displays information about security options configured on the interface.

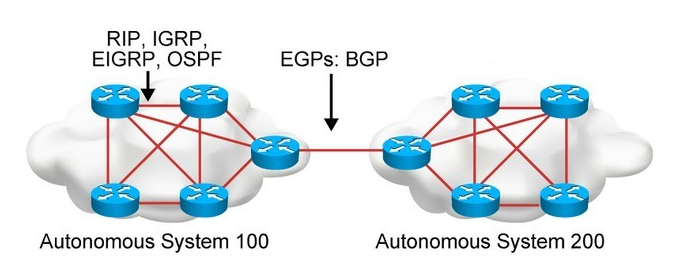
### Router information

**Routing – Static / dynamic / interior / exterior**

Both static and dynamic routing are useful in different network environments. In static routing, the path is manually updated. The administration distance, by default, is 1 on a static route. Making this the quickest routing option for packets to be forwarded by.

Dynamic routing requires more effort to configure. However, once the protocol is configured the routing path will automatically update if the network changes.

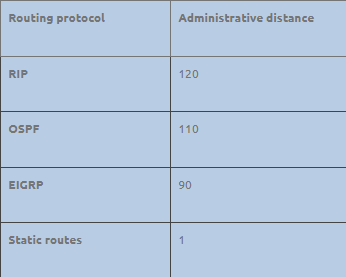
A static floating point route can be added, even when a dynamic routing protocol is configured. This is a backup route that has an admin distance of over 110 (if OSPF is configured). If the dynamic routing protocol is down – the static route will be used as a backup.



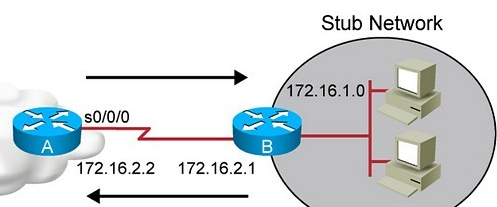
Above shows 2 autonomous system networks with a connecting BGP dynamic route. This topology demonstrates interior dynamic protocols and exterior dynamic protocols. Within an autonomous system, interior protocols are used. Outside of an Autonomous system, exterior protocols are used – border gateway protocol (BGP).

An Autonomous System (AS) is **a group of networks under a single administrative control which could be an Internet Service Provider (ISP)** or a large Enterprise Organization. An Interior Gateway Protocol (IGP) refers to a routing protocol that handles routing within a single autonomous system.

Administrative distance



Routing – Static



The topology above is a network with two branches – Router A and Router B.

*The Stub network behind router B is the 172.16.1.0/24 network.*

*The network between router A and B is the 172.16.2.0/24 network.*

In order to configure static routing in Cisco IOS routers, the command to use.

In this scenario, Router B knows both the 172.16.2.0/24 network and the 72.16.1.0/24 network. This is because router B is directly connected to both. However, Router A is not directly connected to the network behind Router B – 172.16.1.0/24.

Router A will need to be routed to the 172.16.1.0/24 network. A static route will be applied. Below are two different methods in which to apply a static route. If both are implemented, this this will be considered a fully specified static route.

**RouterA(config)# ip route 172.16.1.0 255.255.255.0 172.162.1**

Or

**RouterA(config)# ip route 172.16.1.0 255.255.255.0 s0/0/0**

You can also add an admin distance if you were configuring a floating static point route.

Routing – Floating point static

A floating static point route is used as a redundancy if the dynamic protocol goes down. If OSPF is implemented, then an administration distance of 121 should be implemented into the floating point static route configuration.

**RouterA(config)# ip route 172.16.1.0 255.255.255.0 172.162.1 111**

Or

**RouterA(config)# ip route 172.16.1.0 255.255.255.0 s0/0/0 111** As seen above, all that was added was the administration distance at the end of the syntax (111, 1 + OSPF distance).

Routing – Interior gateway protocols

Interior Gateway Protocols (IGPs) can be classified as two types:

* Distance vector routing protocols – RIPv2, IGRP
* Link-state routing protocols – OSPF, ISIS

Distance vector protocols work best in situations where:

* The network is simple and flat and does not require a special hierarchical design.
* The administrators do not have enough knowledge to configure and troubleshoot link-state protocols.
* Specific types of networks, such as hub-and-spoke networks, are being implemented.
* Worst-case convergence times in a network are not a concern

Link-state protocols work best in situations where:

* The network design is hierarchical, usually occurring in large networks.
* The administrators have a good knowledge of the implemented link-state routing protocol.
* Fast convergence of the network is crucial.

### Access Control List

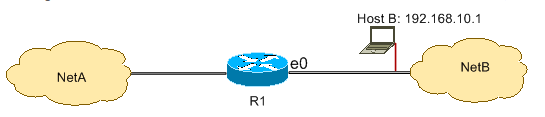
**Access control list –** *An ACL is a permit or deny rule that details what can or can’t enter an interface on a layer 3 switch or router. When a packet enters or leaves the interface, it will be tested against each ACL rule until a match is found. If not match is found, then it will be denied.* ***(If blocking a certain IP address is required, implement an ACL)***

There are two types of access control lists – **standard** and **extended.**

Standard ACL: Numeric range 1-99

Extended ACL: Numeric range 100-199

Below is how to configure an ACL to deny a specific host from gaining access to your network. This is a standard ACL because the selected number – 1, is within the standard ACL numeric range.



This configuration denies all packets from host 192.168.10.1/32 through Ethernet 0 on R1 and permits everything else. You must use the command **access list 1 permit any** to explicitly permit everything else because there is an implicit deny all clause with every ACL.

R1# hostname R1

R1# configure terminal

R1# interface ethernet0

R1# ip access-group 1 in

R1# access-list 1 deny host 192.168.10.1

R1# access-list 1 permit any - *Note, this implies that all other traffic not coming from 192.168.10.1 is permitted.*

## Juniper

### Juniper hardware CLI – Junos.

Juniper Junos is the software or the network operating system used in Juniper Networks hardware systems. It is an operating system that is used in Juniper's routing, switching and security devices.

**--Navigating the different modes.**

Juniper hardware have 3 different modes.

* Linux shell
* operation mode – denoted by >
* configure mode – denoted by #

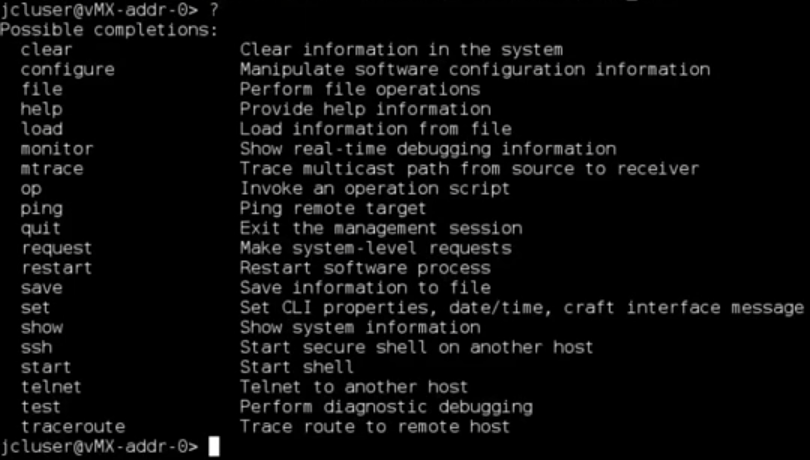
**Note, you cannot do any configuration in Linux shell mode or operation mode. Only in configuration mode.**

**Operations mode.**

If you SSH into a Juniper device and find yourself on the Linux shell mode – type; **CLI**. This will bring you to the operations mode.

Below the user jcluster is in operations mode and the routers name is MX-addr-0.



In operations mode, question mark (?) will bring up all available show commands.

To see the latest committed active configuration, type

**> show configuration**

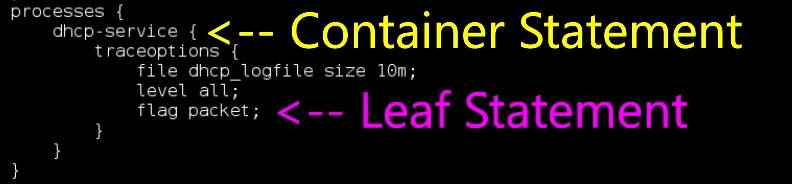
Be aware the output will be different from a Cisco IOS router. The output contains indentation and curly braces. Each line is called a statement – there are two types of statements.

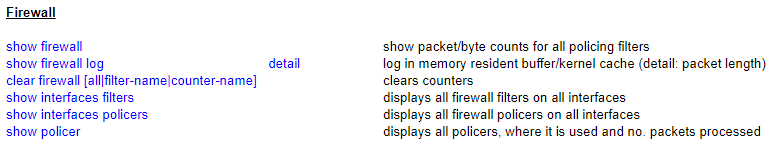
* Container statement – groups other statements together
* leaf statement - is contained within the container statement

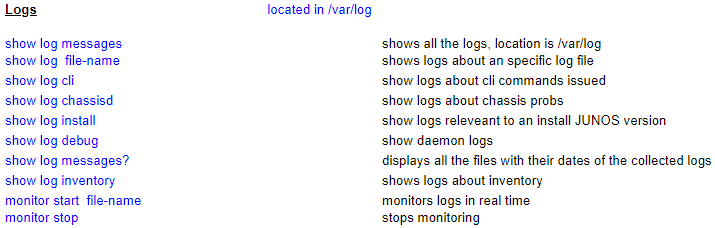
Below are the current processes, within the processes is DHCP services (container statement).

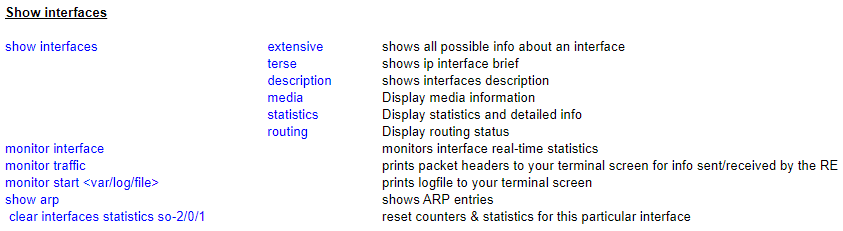
Within that container statement is another one – trace options (container statement).

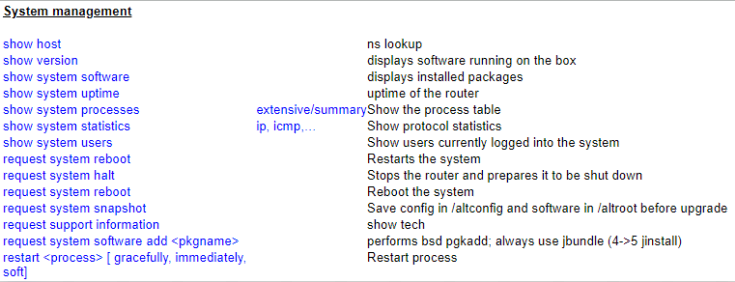
Contained within the trace options container statement is 3 leaf statements. This is where you can configure DHCP services by editing the leaf statements. Think of a container statement as a Linux configuration file.



**Show commands –**

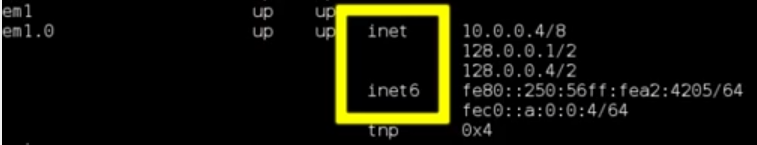






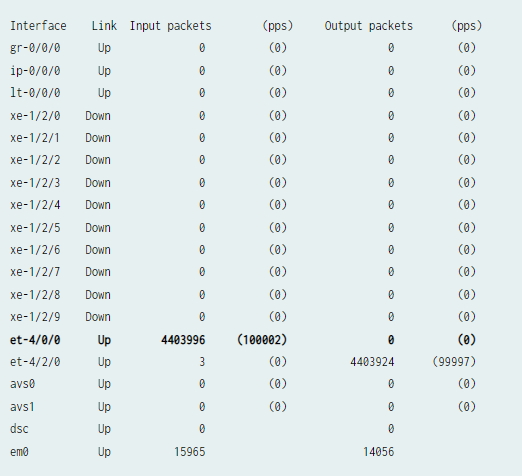
To show the interface brief, type

**> show interfaces terse**

Below is the output. Interface em1.0 is up and is configured with an IPV4 (inet) and IPV6 (inet6) addresses. Em is a management interface.

**To monitor traffic on all the interfaces on your device, type**

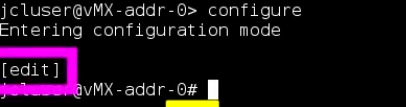
****

****

**Configuration mode**

**To enter configuration mode, type**

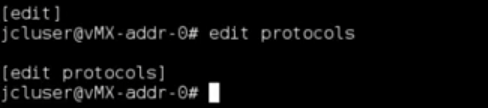
> configured

Below is a User going from operations mode to configuration mode. Note, the banner in the square purple box -This shows the part of the hierarchy you're in. **Edit is the root of the hierarchy.**

**To change to another directory, type**

#edit {directory} *– below the protocols directory was navigated to.*

*Note – edit changed to edit protocols showing what the current directory is.*

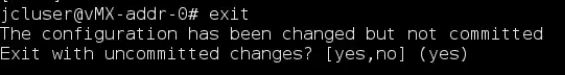


**To go back to last directory, type**

#top

**To exit configuration mode, type**

#exit



**If you had made configuration changes and want to save them, type**

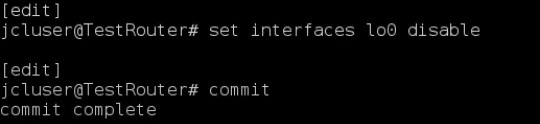
#commit

**Or**

# Commit and-quit

**-Interface configuration**

**To disable a specific interface, type**

# set interfaces lo0 disable

**This command will disable the loop back interface lo0. Then commit will save that configuration.**

**To enable the interface again, you will need to add a leaf statement to the interfaces container to remove that disable statement.**

**To do this you will need to use the delete command.**

# delete interfaces lo0 disable



**Now the leaf statement – disable, was removed from the interfaces container.**

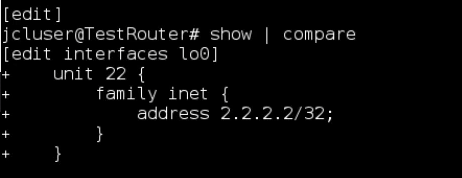
**To set an IP address to an interface, and you know exactly what interface you are going to apply IP address to, type**

# set interfaces lo0 unit 22 family inet address 2.2.2.2/32

**The above command has**

**- Set an ipv4 address 2.2.2.2/32 to lo0 (loop back interface), more specifically to the sub-interface 22(known as a logical interface). Using an inet(IPV4) address.**

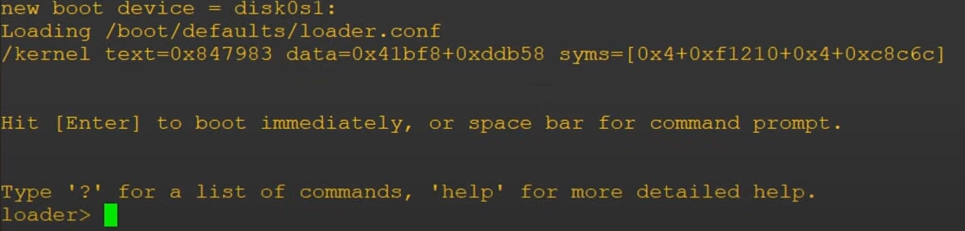
**As you can see in the container statement, interface lo0 unit 22 now has a IPV4 address.**

****

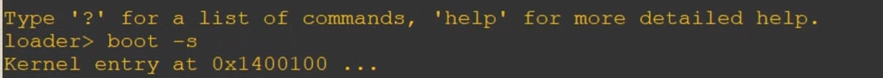
**Password recovery on Juniper switch or router**

Do the following steps if you’re unable to access switch or router due to of password failure.

1. Connect to the console port
2. Reboot the switch/router
3. Enter the boot loader
4. Boot into single-user mode
5. set the root password

During reboot, you will be prompted to press space bar – this will give you access to the boot loader. The boot loader comes with very limited CLI commands, only for use in emergency situations such as password recovery.

In the boot loader, type >boot -s.

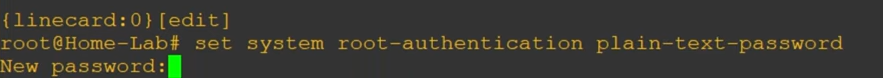
This will boot into single user mode (This is similar to safe mode on windows)

Once single user mode is enabled, type /bin/sh: recovery.

****

Now the User is in the Junos CLI, because recovery mode was used there was no need to enter a password. From CLI enter configuration mode. Type > configuration.

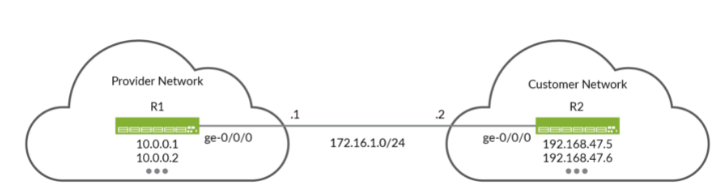
Once in the configuration mode, type # set system root-authentication plain-text-password.

This will prompt the User to enter a new password.

Static routing

Static routing is often used when the complexity of a dynamic routing protocol is not desired. A route that does not frequently change, and for which there is only one (or very few) paths to the destination, is a good candidate for static routing.

Below is an example of a static route from device-R1 to device-R2.

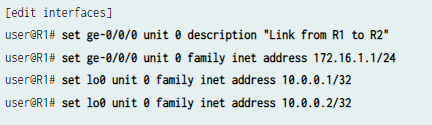


Loopback interfaces will also be configured, these loopback addresses provide remote destinations to ping, so you can verify the IPv4 static routing works properly.

R1 configuration

From configuration mode, set a hostname for R1.

Configure all interfaces – including loopback interfaces.



Define the static route to the customer’s prefix on the R1 device. Be sure to specify the R2 end of the point-to-point link as the next hop for the static route.



Commit your changes to save configuration.

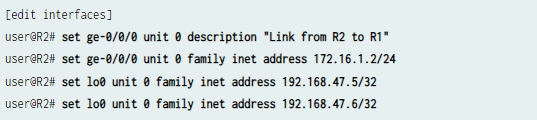


R2 configuration

From configuration mode, set R2 hostname.



Configure all interfaces – including loopback interfaces.



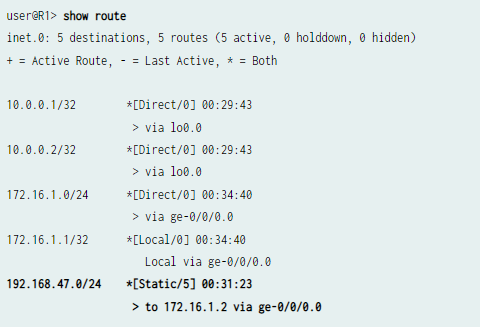
Define the IPv4 static default route on the R2 device. Be sure to specify the R1 end of the point-to-point link as the next hop for the static route.

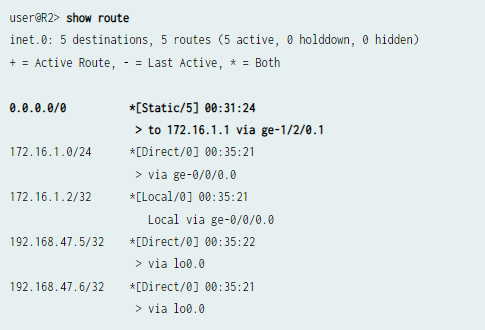


Commit your changes.



To see the changes that you have committed.





## PFSense (empty)

# Host

## Windows

### PowerShell

**RUNAS**

Starting with PowerShell 4.0, we can specify that a script requires administrative privileges by including a #Requires statement with the -RunAsAdministrator switch parameter.#Requires -RunAsAdministrator

**Run a script on a remote computer**

****

1. invoke-command -computername machine1, machine2 - filepath c:\Script\script.ps1

**Remotely shut down another machine after one minute**

1. Start-Sleep 60; Restart-Computer –Force – ComputerName TARGETMACHINE

**Install an MSI package on a remote computer**

1. (Get-WMIObject -ComputerName TARGETMACHINE -List | Where-Object -FilterScript {$\_.Name -eq "Win32\_Product"}).Install(\\MACHINEWHEREMSIRESI DES\path\package.msi)

**Upgrade an installed application with an MSI-based application upgrade package**

1. (Get-WmiObject -Class Win32\_Product -ComputerName

. -Filter

"Name='name\_of\_app\_to\_be\_upgraded'").Upgrade(\\MAC HINEWHEREMSIRESIDES\path\upgrade\_package.msi)

**Remove an MSI package from the current computer**

1. (Get-WmiObject -Class Win32\_Product -Filter "Name='product\_to\_remove'" -ComputerName . ).Uninstall()

**Collecting information**

**Get information about the make and model of a computer**

Get-WmiObject -Class Win32\_ComputerSystem

**Get information about the BIOS of the current computer**

Get-WmiObject -Class Win32\_BIOS -ComputerName .

**List installed hotfixes (QFEs, or Windows Update files)**

Get-WmiObject -Class Win32\_QuickFixEngineering - ComputerName .

**Get the username of the person currently logged on to a computer**

1. Get-WmiObject -Class Win32\_ComputerSystem - Property UserName -ComputerName .

**Find just the names of installed applications on the current computer**

1. Get-WmiObject -Class Win32\_Product -ComputerName. | Format-Wide -Column 1
2. **Get IP addresses assigned to the current computer**
3. -- Get-WmiObject -Class Win32\_NetworkAdapterConfiguration -Filter
4. IPEnabled=TRUE -ComputerName . | Format-Table - Property IPAddress
5. **Get a more detailed IP configuration report for the current machine**
6. -- Get-WmiObject -Class
7. Win32\_NetworkAdapterConfiguration -Filter IPEnabled=TRUE -ComputerName . | Select-Object - Property [a-z]\* -ExcludeProperty IPX\*,WINS\*
8. **To find network cards with DHCP enabled on the current computer**
9. -- Get-WmiObject -Class
10. Win32\_NetworkAdapterConfiguration -Filter "DHCPEnabled=true" -ComputerName .
11. **Enable DHCP on all network adapters on the current computer**
12. -- Get-WmiObject -Class
13. Win32\_NetworkAdapterConfiguration -Filter
14. IPEnabled=true -ComputerName . | ForEach-Object -
15. Process {$\_.EnableDHCP()}
16. **Navigate the Windows Registry like the file system** -- cd hkcu:
17. **Search recursively for a certain string within files** -- dir –r | select string "searchforthis"
18. **Find the five processes using the most memory** -- ps | sort –p ws | select –last 5
19. **Cycle a service (stop, and then restart it) like DHCP** -- Restart-Service DHCP
20. **List all items within a folder** -- Get-ChildItem – Force
21. **Recurse over a series of directories or folders** -- Get-ChildItem –Force c:\directory –Recurse
22. **Remove all files within a directory without being prompted for each** -- **Remove-Item C:\tobedeleted –**
23. **Recurse**
24. **Restart the current computer --** (Get-WmiObject -Class Win32\_OperatingSystem -ComputerName .).Win32Shutdown(2)

**Set-ExecutionPolicy**

Although you can create and execute PowerShell scripts, Microsoft has disabled scripting by default in an effort to prevent malicious code from executing in a PowerShell environment. You can use the Set-ExecutionPolicy command to control the level of security surrounding PowerShell scripts. Four levels of security are available to you:

**Restricted** -- Restricted is the default execution policy and locks PowerShell down so that commands can be entered only interactively. PowerShell scripts are not allowed to run.

**All Signed** -- If the execution policy is set to All Signed then scripts will be allowed to run, but only if they are signed by a trusted publisher.

**Remote Signed** -- If the execution policy is set to Remote Signed, any PowerShell scripts that have been locally created will be allowed to run. Scripts created remotely are allowed to run only if they are signed by a trusted publisher.

**Unrestricted** -- As the name implies, Unrestricted removes all restrictions from the execution policy.

You can set an execution policy by entering the Set-ExecutionPolicy command followed by the name of the policy. For example, if you wanted to allow scripts to run in an unrestricted manner you could type:

**Set-ExecutionPolicy Unrestricted**

**Get-ExecutionPolicy**

If you're working on an unfamiliar server, you'll need to know what execution policy is in use before you attempt to run a script. You can find out by using the Get-ExecutionPolicy command.

**Get-Service**

The Get-Service command provides a list of all of the services that are installed on the system. If you are interested in a specific service you can append the -Name switch and the name of the service (wildcards are permitted) When you do, Windows will show you the service's state.

**Export-CSV**

Just as you can create an HTML report based on PowerShell data, you can also export data from PowerShell into a CSV file that you can open using Microsoft Excel. The syntax is similar to that of converting a command's output to HTML. At a minimum, you must provide an output filename. For example, to export the list of system services to a CSV file, you could use the following command:

**Get-Service | Export-CSV c:\service.csv**

**Select-Object**

If you tried using the command above, you know that there were numerous properties included in the CSV file. It's often helpful to narrow things down by including only the properties you are really interested in. This is where the Select-Object command comes into play. The Select-Object command allows you to specify specific properties for inclusion. For example, to create a CSV file containing the name of each system service and its status, you could use the following command:

**Get-Service | Select-Object Name, Status | Export-CSV c:\service.csv**

**Get-Process**

Just as you can use the Get-Service command to display a list of all of the system services, you can use the Get-Process command to display a list of all of the processes that are currently running on the system.

**Stop-Process**

Sometimes, a process will freeze up. When this happens, you can use the Get-Process command to get the name or the process ID for the process that has stopped responding. You can then terminate the process by using the Stop-Process command. You can terminate a process based on its name or on its process ID. For example, you could terminate Notepad by using:

**Stop-Process -Name notepad**

**Stop-Process -ID 2668**

**PowerShell Active Directory**

**Reset a User Password**

Let's start with a typical IT pro task: resetting a user's password. We can easily accomplish this by using the Set-ADAccountPassword cmdlet. The tricky part is that the new password must be specified as a secure string: a piece of text that's encrypted and stored in memory for the duration of your PowerShell session. So first, we'll create a variable with the new password:

**PS C:\> $new=Read-Host "Enter the new password" - AsSecureString**

Next, we'll enter the new password:

**PS C:\>**

Now we can retrieve the account (using the samAccountname is best) and provide the new password. Here's the change for user Jack Frost:

**PS C:\> Set-ADAccountPassword jfrost -NewPassword $new**

Unfortunately, there's a bug with this cmdlet: -Passthru, - Whatif, and -Confirm don't work. If you prefer a one-line approach, try this:

**PS C:\> Set-ADAccountPassword jfrost -NewPassword**

(ConvertTo-SecureString -AsPlainText -String "P@ssw0rd1z3" -force)

Finally, I need Jack to change his password at his next logon, so I'll modify the account by using Set-ADUser:

**PS C:\> Set-ADUser jfrost -ChangePasswordAtLogon $True**

The command doesn't write to the pipeline or console unless you use -True. But I can verify success by retrieving the username via the Get-ADUser cmdlet and specifying the PasswordExpired property, shown in Figure 2.

**Disable and Enable a User Account**

Next, let's disable an account. We'll continue to pick on Jack Frost. This code takes advantage of the

-Whatif parameter, which you can find on many cmdlets that change things, to verify my command without running it:

**PS C:\> Disable-ADAccount jfrost -whatif**

What if: Performing operation "Set" on Target "CN=Jack Frost,

**OU=staff,OU=Testing,DC=GLOBOMANTICS,DC=local"**.

Now to do the deed for real:

**PS C:\> Disable-ADAccount jfrost**

When the time comes to enable the account, can you guess the cmdlet name?

**PS C:\> Enable-ADAccount jfrost**

These cmdlets can be used in a pipelined expression to enable or disable as many accounts as you need. For example, this code disables all user accounts in the Sales department:

**PS C:\> get-aduser -filter "department -eq 'sales'" | disable-adaccount**

**Unlock a User Account**

Now, Jack has locked himself out after trying to use his new password. Rather than dig through the GUI to find his account, I can unlock it by using this simple command:

**PS C:\> Unlock-ADAccount jfrost**

**Delete a User Account**

Deleting 1 or 100 user accounts is easy with the Remove-ADUser cmdlet. I don't want to delete Jack Frost, but if I did, I could use this code:

**PS C:\> Remove-ADUser jfrost -whatif**

What if: Performing operation "Remove" on Target

**"CN=Jack Frost,OU=staff,OU=Testing,DC=GLOBOMANTICS,DC=local".**

Or I could pipe in a bunch of users and delete them with one simple command:

**PS C:\> get-aduser -filter "enabled -eq 'false'"-property WhenChanged -SearchBase "OU=Employees, DC=Globomantics,DC=Local" | where {$\_.WhenChanged -le (Get-Date).AddDays(-180)} | Remove-ADuser -whatif**

This one-line command would find and delete all disabled accounts in the Employees organizational unit (OU) that haven't been changed in at least 180 days.

**Add Members to a Group**

Let's add Jack Frost to the Chicago IT group:

**PS C:\> add-adgroupmember "chicago IT" -Members jfrost**

It's that simple. You can just as easily add hundreds of users to a group, although doing so is a bit more awkward than I would like:

**PS C:\> Add-ADGroupMember "Chicago Employees" – member (get-aduser -filter "city -eq 'Chicago'")**

I used a parenthetical pipelined expression to find all users with a City property of Chicago. The code in the parentheses is executed and the resulting objects are piped to the -Member parameter. Each user object is then added to the Chicago Employees group. It doesn't matter whether there are 5 or 500 users; updating group membership takes only a few seconds This expression could also be written using ForEach-Object, which might be easier to follow.

**PS C:\> Get-ADUser -filter "city -eq 'Chicago'" | foreach {Add-ADGroupMember "Chicago Employees" -Member $\_}**

**Enumerate Members of a Group**

You might want to see who belongs to a given group. For example, you should periodically find out who belongs to the Domain Admins group:

**PS C:\> Get-ADGroupMember "Domain Admins"**

The cmdlet writes an AD object for each member to the pipeline. But what about nested groups? My Chicago All Users group is a collection of nested groups. To get a list of all user accounts, all I need to do is use the -Recursive parameter:

**PS C:\> Get-ADGroupMember "Chicago All Users" -Recursive | Select DistinguishedName**

**Disable a Computer Account**

Perhaps when you find those inactive or obsolete accounts, you'd like to disable them. Easy enough. We'll use the same cmdlet that we use with user accounts. You can specify it by using the account's samAccountname:

**PS C:\> Disable-ADAccount -Identity "chi-srv01$" -whatif**

What if: Performing operation "Set" on Target "CN=CHI-SRV01,

CN=Computers,DC=GLOBOMANTICS,DC=local".

Or you can use a pipelined expression:

**PS C:\> get-adcomputer "chi-srv01" | Disable-ADAccount**

I can also take my code to find obsolete accounts and disable all those accounts:

**PS C:\> get-adcomputer -filter "Passwordlastset** -lt '1/1/2012'" -properties \*| Disable-ADAccount

**Find Computers by Type**

The last task that I'm often asked about is finding computer accounts by type, such as servers or laptops. This requires a little creative thinking on your part. There's nothing in AD that distinguishes a server from a client, other than the OS. If you have a laptop or desktop running Windows Server 2008, you'll need to get extra creative.

You need to filter computer accounts based on the OS. It might be helpful to get a list of those OSs first:

**PS C:\> Get-ADComputer -Filter \* -Properties OperatingSystem | Select OperatingSystem -unique | Sort OperatingSystem**

I want to find all the computers that have a server OS:

**PS C:\> Get-ADComputer -Filter "OperatingSystem -like '\*Server\*'" -properties OperatingSystem,OperatingSystem ServicePack | Select Name,Op\* | format-list**

As with the other AD Get cmdlets, you can fine-tune your search parameters and limit your query to a specific OU if necessary. All the expressions that I've shown you can be integrated into larger PowerShell expressions. For example, you can sort, group, filter, export to a comma-separated value (CSV), or build and email an HTML report, all from PowerShell and all without writing a single PowerShell script! In fact, here's a bonus: a user password-age report, saved as an HTML file:

**PS C:\> Get-ADUser -Filter "Enabled -eq 'True' -AND PasswordNeverExpires -eq 'False'" -Properties**

**PasswordLastSet,PasswordNeverExpires,PasswordExpired | Select DistinguishedName,Name,pass\*,@{Name="PasswordAge"**

**;**

**Expression={(Get-Date)-$\_.PasswordLastSet}} |sort**

**PasswordAge -Descending | ConvertTo-Html -Title**

**"Password Age Report" | Out-File c:\Work\pwage.htm**

### Group Policy Object

A Group Policy Object (GPO) is a virtual collection of policy settings. A GPO has a unique name, such as a GUID.

Group Policy settings are contained in a GPO. A GPO can represent policy settings in the file system and in the Active Directory. GPO settings are evaluated by clients using the hierarchical nature of Active Directory.

Type of Group Policy Objects (GPOs)

When learning about GPOs, there are three main types that you should be aware of:

**Local Group Policy Objects**

Local group policy objects exist by default on all Windows computers and are utilized when IT admins need to apply policy settings to a single Windows computer or user. These types of GPO's only apply to local computers and to the users that log on to that computer on-site.

**Non-local Group Policy Objects**

Unlike local GPO's, non-local group policy objects require your Windows computers and users to be linked to Active Directory objects, sites, domains, or organizational units. This means that non-local GPO's can apply to one or more Windows computers and users.

**Starter Group Policy Objects**

Starter GPO's are templates for group policy settings. These templates enable IT, administrators, to pre-configure a group of settings that represent a baseline for any future policy to be created.

### Active Directory

Add-ADCentralAccessPolicyMember

Adds central access rules to a central access policy in Active Directory.

Add-ADComputerServiceAccount

Adds one or more service accounts to an Active Directory computer.

Add-ADDomainControllerPasswordReplicationPolicy

Adds users, computers, and groups to the allowed or denied list of a read-only domain controller password replication policy.

Add-ADFineGrainedPasswordPolicySubject

Applies a fine-grained password policy to one more users and groups.

Add-ADGroupMember

Adds one or more members to an Active Directory group.

Add-ADPrincipalGroupMembership

Adds a member to one or more Active Directory groups.

Add-ADResourcePropertyListMember

Adds one or more resource properties to a resource property list in Active Directory.

Clear-ADAccountExpiration

Clears the expiration date for an Active Directory account.

Clear-ADClaimTransformLink

Removes a claims transformation from being applied to one or more cross-forest trust relationships in Active Directory.

Disable-ADAccount

Disables an Active Directory account.

Disable-ADOptionalFeature

Disables an Active Directory optional feature.

Enable-ADAccount

Enables an Active Directory account.

Enable-ADOptionalFeature

Enables an Active Directory optional feature.

Get-ADAccountAuthorizationGroup

Gets the accounts token group information.

Get-ADAccountResultantPasswordReplicationPolicy

Gets the resultant password replication policy for an Active Directory account.

Get-ADAuthenticationPolicy

Gets one or more Active Directory Domain Services authentication policies.

Get-ADAuthenticationPolicySilo

Gets one or more Active Directory Domain Services authentication policy silos.

Get-ADCentralAccessPolicy

Retrieves central access policies from Active Directory.

Get-ADCentralAccessRule

Retrieves central access rules from Active Directory.

Get-ADClaimTransformPolicy

Returns one or more Active Directory claim transform objects based on a specified filter.

Get-ADClaimType

Returns a claim type from Active Directory.

Get-ADComputer

Gets one or more Active Directory computers.

Get-ADComputerServiceAccount

Gets the service accounts hosted by a computer.

Get-ADDCCloningExcludedApplicationList

Gets a list of installed programs and services present on this domain controller that are not in the default or user defined inclusion list.

Get-ADDefaultDomainPasswordPolicy

Gets the default password policy for an Active Directory domain.

Get-ADDomain

Gets an Active Directory domain.

Get-ADDomainController

Gets one or more Active Directory domain controllers based on discoverable services criteria, search parameters or by providing a domain controller identifier, such as the NetBIOS name.

Get-ADDomainControllerPasswordReplicationPolicy

Gets the members of the allowed list or denied list of a read-only domain controller's password replication policy.

Get-ADDomainControllerPasswordReplicationPolicyUsage

Gets the Active Directory accounts that are authenticated by a read-only domain controller or that are in the revealed list of the domain controller.

Get-ADFineGrainedPasswordPolicy

Gets one or more Active Directory fine-grained password policies.

Get-ADFineGrainedPasswordPolicySubject

Gets the users and groups to which a fine-grained password policy is applied.

Get-ADForest

Gets an Active Directory forest.

Get-ADGroup

Gets one or more Active Directory groups.

Get-ADGroupMember

Gets the members of an Active Directory group.

Get-ADObject

Gets one or more Active Directory objects.

Get-ADOptionalFeature

Gets one or more Active Directory optional features.

Get-ADOrganizationalUnit

Gets one or more Active Directory organizational units.

Get-ADPrincipalGroupMembership

Gets the Active Directory groups that have a specified user, computer, group, or service account.

Get-ADReplicationAttributeMetadata

Gets the replication metadata for one or more Active Directory replication partners.

Get-ADReplicationConnection

Returns a specific Active Directory replication connection or a set of AD replication connection objects based on a specified filter.

Get-ADReplicationFailure

Returns a collection of data describing an Active Directory replication failure.

Get-ADReplicationPartnerMetadata

Returns the replication metadata for a set of one or more replication partners.

Get-ADReplicationQueueOperation

Returns the contents of the replication queue for a specified server.

Get-ADReplicationSite

Returns a specific Active Directory replication site or a set of replication site objects based on a specified filter.

Get-ADReplicationSiteLink

Returns a specific Active Directory site link or a set of site links based on a specified filter.

Get-ADReplicationSiteLinkBridge

Gets a specific Active Directory site link bridge or a set of site link bridge objects based on a specified filter.

Get-ADReplicationSubnet

Gets one or more Active Directory subnets.

Get-ADReplicationUpToDatenessVectorTable

Displays the highest Update Sequence Number (USN) for the specified domain controller.

Get-ADResourceProperty

Gets one or more resource properties.

Get-ADResourcePropertyList

Gets resource property lists from Active Directory.

Get-ADResourcePropertyValueType

Gets a resource property value type from Active Directory.

Get-ADRootDSE

Gets the root of a directory server information tree.

Get-ADServiceAccount

Gets one or more Active Directory managed service accounts or group managed service accounts.

Get-ADTrust

Gets all trusted domain objects in the directory.

Get-ADUser

Gets one or more Active Directory users.

Get-ADUserResultantPasswordPolicy

Gets the resultant password policy for a user.

Grant-ADAuthenticationPolicySiloAccess

Grants permission to join an authentication policy silo.

Install-ADServiceAccount

Installs an Active Directory managed service account on a computer or caches a group managed service account on a computer.

Move-ADDirectoryServer

Moves a directory server in Active Directory to a new site.

Move-ADDirectoryServerOperationMasterRole

Moves operation master roles to an Active Directory directory server.

Move-ADObject

Moves an Active Directory object or a container of objects to a different container or domain.

New-ADAuthenticationPolicy

Creates an Active Directory Domain Services authentication policy object.

New-ADAuthenticationPolicySilo

Creates an Active Directory Domain Services authentication policy silo object.

New-ADCentralAccessPolicy

Creates a new central access policy in Active Directory containing a set of central access rules.

New-ADCentralAccessRule

Creates a central access rule in Active Directory.

New-ADClaimTransformPolicy

Creates a new claim transformation policy object in Active Directory.

New-ADClaimType

Creates a new claim type in Active Directory.

New-ADComputer

Creates a new Active Directory computer object.

New-ADDCCloneConfigFile

Performs prerequisite checks for cloning a domain controller and generates a clone configuration file if all checks succeed.

New-ADFineGrainedPasswordPolicy

Creates a new Active Directory fine-grained password policy.

New-ADGroup

Creates an Active Directory group.

New-ADObject

Creates an Active Directory object.

New-ADOrganizationalUnit

Creates an Active Directory organizational unit.

New-ADReplicationSite

Creates an Active Directory replication site in the directory.

New-ADReplicationSiteLink

Creates a new Active Directory site link for in managing replication.

New-ADReplicationSiteLinkBridge

Creates a site link bridge in Active Directory for replication.

New-ADReplicationSubnet

Creates an Active Directory replication subnet object.

New-ADResourceProperty

Creates a resource property in Active Directory.

New-ADResourcePropertyList

Creates a resource property list in Active Directory.

New-ADServiceAccount

Creates a new Active Directory managed service account or group managed service account object.

New-ADUser

Creates an Active Directory user.

Remove-ADAuthenticationPolicy

Removes an Active Directory Domain Services authentication policy object.

Remove-ADAuthenticationPolicySilo

Removes an Active Directory Domain Services authentication policy silo object.

Remove-ADCentralAccessPolicy

Removes a central access policy from Active Directory.

Remove-ADCentralAccessPolicyMember

Removes central access rules from a central access policy in Active Directory.

Remove-ADCentralAccessRule

Removes a central access rule from Active Directory.

Remove-ADClaimTransformPolicy

Removes a claim transformation policy object from Active Directory.

Remove-ADClaimType

Removes a claim type from Active Directory.

Remove-ADComputer

Removes an Active Directory computer.

Remove-ADComputerServiceAccount

Removes one or more service accounts from a computer.

Remove-ADDomainControllerPasswordReplicationPolicy

Removes users, computers, and groups from the allowed or denied list of a read-only domain controller password replication policy.

Remove-ADFineGrainedPasswordPolicy

Removes an Active Directory fine-grained password policy.

Remove-ADFineGrainedPasswordPolicySubject

Removes one or more users from a fine-grained password policy.

Remove-ADGroup

Removes an Active Directory group.

Remove-ADGroupMember

Removes one or more members from an Active Directory group.

Remove-ADObject

Removes an Active Directory object.

Remove-ADOrganizationalUnit

Removes an Active Directory organizational unit.

Remove-ADPrincipalGroupMembership

Removes a member from one or more Active Directory groups.

Remove-ADReplicationSite

Deletes the specified replication site object from Active Directory.

Remove-ADReplicationSiteLink

Deletes an Active Directory site link used to manage replication.

Remove-ADReplicationSiteLinkBridge

Deletes a replication site link bridge from Active Directory.

Remove-ADReplicationSubnet

Deletes the specified Active Directory replication subnet object from the directory.

Remove-ADResourceProperty

Removes a resource property from Active Directory.

Remove-ADResourcePropertyList

Removes one or more resource property lists from Active Directory.

Remove-ADResourcePropertyListMember

Removes one or more resource properties from a resource property list in Active Directory.

Remove-ADServiceAccount

Removes an Active Directory managed service account or group managed service account object.

Remove-ADUser

Removes an Active Directory user.

Rename-ADObject

Changes the name of an Active Directory object.

Reset-ADServiceAccountPassword

Resets the password for a standalone managed service account.

Restore-ADObject

Restores an Active Directory object.

Revoke-ADAuthenticationPolicySiloAccess

Revokes membership in an authentication policy silo for the specified account.

Search-ADAccount

Gets Active Directory user, computer, or service accounts.

Set-ADAccountAuthenticationPolicySilo

Modifies the authentication policy or authentication policy silo of an account.

Set-ADAccountControl

Modifies user account control (UAC) values for an Active Directory account.

Set-ADAccountExpiration

Sets the expiration date for an Active Directory account.

Set-ADAccountPassword

Modifies the password of an Active Directory account.

Set-ADAuthenticationPolicy

Modifies an Active Directory Domain Services authentication policy object.

Set-ADAuthenticationPolicySilo

Modifies an Active Directory Domain Services authentication policy silo object.

Set-ADCentralAccessPolicy

Modifies a central access policy in Active Directory.

Set-ADCentralAccessRule

Modifies a central access rule in Active Directory.

Set-ADClaimTransformLink

Applies a claims transformation to one or more cross-forest trust relationships in Active Directory.

Set-ADClaimTransformPolicy

Sets the properties of a claims transformation policy in Active Directory.

Set-ADClaimType

Modify a claim type in Active Directory.

Set-ADComputer

Modifies an Active Directory computer object.

Set-ADDefaultDomainPasswordPolicy

Modifies the default password policy for an Active Directory domain.

Set-ADDomain

Modifies an Active Directory domain.

Set-ADDomainMode

Sets the domain mode for an Active Directory domain.

Set-ADFineGrainedPasswordPolicy

Modifies an Active Directory fine-grained password policy.

Set-ADForest

Modifies an Active Directory forest.

Set-ADForestMode

Sets the forest mode for an Active Directory forest.

Set-ADGroup

Modifies an Active Directory group.

Set-ADObject

Modifies an Active Directory object.

Set-ADOrganizationalUnit

Modifies an Active Directory organizational unit.

Set-ADReplicationConnection

Sets properties on Active Directory replication connections.

Set-ADReplicationSite

Sets the replication properties for an Active Directory site.

Set-ADReplicationSiteLink

Sets the properties for an Active Directory site link.

Set-ADReplicationSiteLinkBridge

Sets the properties of a replication site link bridge in Active Directory.

Set-ADReplicationSubnet

Sets the properties of an Active Directory replication subnet object.

Set-ADResourceProperty

Modifies a resource property in Active Directory.

Set-ADResourcePropertyList

Modifies a resource property list in Active Directory.

Set-ADServiceAccount

Modifies an Active Directory managed service account or group managed service account object.

Set-ADUser

Modifies an Active Directory user.

Show-ADAuthenticationPolicyExpression

Displays the Edit Access Control Conditions window update or create security descriptor definition language (SDDL) security descriptors.

Sync-ADObject

Replicates a single object between any two domain controllers that have partitions in common.

Test-ADServiceAccount

Tests a managed service account from a computer.

Uninstall-ADServiceAccount

Uninstalls an Active Directory managed service account from a computer or removes a cached group managed service account from a computer.

Unlock-ADAccount

Unlocks an Active Directory account.

## Linux

**System**

uname -a =>Display linux system information

uname -r =>Display kernel release information

uptime =>Show how long the system has been running + load

hostname =>Show system host name

hostname -i =>Display the IP address of the host

last reboot =>Show system reboot history

date =>Show the current date and time

cal =>Show this month calendar

w =>Display who is online

whoami =>Who you are logged in as

finger user =>Display information about user

**HARDWARE**

dmesg =>Detected hardware and boot messages

cat /proc/cpuinfo =>CPU model

cat /proc/meminfo =>Hardware memory

cat /proc/interrupts =>Lists the number of interrupts per CPU per I/O device

lshw =>Displays information on hardware configuration of the system

lsblk =>Displays block device related information in Linux

free -m =>Used and free memory (-m for MB)

lspci -tv =>Show PCI devices

lsusb -tv =>Show USB devices

dmidecode =>Show hardware info from the BIOS

hdparm -i /dev/sda =>Show info about disk sda

hdparm -tT /dev/sda =>Do a read speed test on disk sda

badblocks -s /dev/sda =>Test for unreadable blocks on disk sda

**USERS**

id =>Show the active user id with login and group

last =>Show last logins on the system

who =>Show who is logged on the system

groupadd admin =>Add group "admin"

useradd -c "Sam" =>g admin -m sam #Create user "sam"

userdel sam =>Delete user sam

adduser sam =>Add user "sam"

usermod =>Modify user information

chgrp => Changes a users group

**File Commands**

ls –al =>Display all information about files/ directories

pwd =>Show the path of current directory

mkdir directory-name =>Create a directory

rm file-name =>Delete file

rm -r directory-nam =>Delete directory recursively

rm -f file-name =>Forcefully remove file

rm -rf directory-name =>Forcefully remove directory recursively

cp file1 file2 =>Copy file1 to file2

cp -r dir1 dir2 =>Copy dir1 to dir2, create dir2 if it doesn’t exist

mv file1 file2 =>Rename source to dest / move source to directory

ln –s /path/to/file-name link-name #Create symbolic link to file-name touch file =>Create or update file

cat > file =>Place standard input into file

more file =>Output contents of file

head file =>Output first 10 lines of file

tail file =>Output last 10 lines of file

tail -f file =>Output contents of file as it grows starting with the last 10 lines

gpg -c file =>Encrypt file

gpg file.gpg =>Decrypt file

wc =>print the number of bytes, words, and lines in files

xargs =>Execute command lines from standard input

**Process Related**

ps =>Display your currently active processes

ps aux | grep 'telnet' =>Find all process id related to telnet process

pmap =>Memory map of process

top =>Display all running processes

kill pid =>Kill process with mentioned pid id

killall proc =>Kill all processes named proc

pkill process-name =>Send signal to a process with its name

bg =>Resumes suspended jobs without bringing them to foreground

fg =>Brings the most recent job to foreground

fg n =>Brings job n to the foreground

File Permission Related

chmod octal file-name =>Change the permissions of file to octal Example

chmod 777 /data/test.c =>Set rwx permission for owner, group, world

chmod 755 /data/test.c =>Set rwx permission for owner,rx for group and world

chown owner-user file =>Change owner of the file

chown owner-user:owner-group file-name =>Change owner and group owner of the file

chown owner-user:owner-group directory =>Change owner and group owner of the directory

**Network**

ip addr show =>Display all network interfaces and IP address

ip address add 192.168.0.1 dev eth0 =>Set IP address

ethtool eth0 =>Linux tool to show ethernet status

mii-tool eth0 =>Linux tool to show ethernet status

ping host =>Send echo request to test connection

whois domain =>Get who is information for domain

dig domain =>Get DNS information for domain

dig -x host =>Reverse lookup host

host google.com =>Lookup DNS ip address for the name

hostname –i =>Lookup local ip address

wget file =>Download file

netstat -tupl =>Listing all active listening ports

**Compression / Archives**

tar cf home.tar home =>Create tar named home.tar containing home/

tar xf file.tar =>Extract the files from file.tar

tar czf file.tar.gz files =>Create a tar with gzip compression

gzip file =>Compress file and renames it to file.gz

**Install Package**

rpm -i pkgname.rpm =>Install rpm based package

rpm -e pkgname =>Remove package

**Install from Source**

./configure

make

make install

Search

grep pattern files =>Search for pattern in files

grep -r pattern dir =>Search recursively for pattern in dir

locate file =>Find all instances of file

find /home/tom -name 'index\*' =>Find files names that start with "index"

find /home -size +10000k =>Find files larger than 10000k in /home

**Login (SSH AND TELNET)**

ssh user@host =>Connect to host as user

ssh -p port user@host =>Connect to host using specific port

telnet host =>Connect to the system using telnet port

**File Transfer**

sftp 192.16875.2 =>Connect remote host

scp

scp file.txt server2:/tmp =>Secure copy file.txt to remote host /tmp

rsync folder

rsync -a /home/apps /backup/ =>Synchronize source to destination

**Disk usage**

df –h =>Show free space on mounted filesystems

df -i =>Show free inodes on mounted filesystems

fdisk -l =>Show disks partitions sizes and types

du -ah =>Display disk usage in human readable form

du -sh =>Display total disk usage on the current directory

findmnt =>Displays target mount point for all filesystem

mount device-path mount-point =>Mount a device

**Directory Traverse**

cd .. =>To go up one level of the directory tree

cd =>Go to $HOME directory

cd /test =>Change to /test directory

### Bash

The UNIX shell program interprets user commands, which are either directly entered by the user, or which can be read from a file called the shell script or shell program. Shell scripts are interpreted, not compiled. The shell reads commands from the script line per line and searches for those commands on the system (see Section 1.2), while a compiler converts a program into machine readable form, an executable file - which may then be used in a shell script.

Apart from passing commands to the kernel, the main task of a shell is providing a user environment, which can be configured individually using shell resource configuration files.

**Shell types**

Just like people know different languages and dialects, your UNIX system will usually offer a variety of shell types:

sh or Bourne Shell: the original shell still used on UNIX systems and in UNIX-related environments. This is the basic shell, a small program with few features. While this is not the standard shell, it is still available on every Linux system for compatibility with UNIX programs.

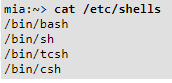
bash or Bourne Again shell: the standard GNU shell, intuitive and flexible. Probably most advisable for beginning users while being at the same time a powerful tool for the advanced and professional user. On Linux, bash is the standard shell for common users. This shell is a so-called superset of the Bourne shell, a set of add-ons and plug-ins. This means that the Bourne Again shell is compatible with the Bourne shell: commands that work in sh, also work in bash. However, the reverse is not always the case. All examples and exercises in this book use bash.

csh or C shell: the syntax of this shell resembles that of the C programming language. Sometimes asked for by programmers.

tcsh or TENEX C shell: a superset of the common C shell, enhancing user-friendliness and speed. That is why some also call it the Turbo C shell.

ksh or the Korn shell: sometimes appreciated by people with a UNIX background. A superset of the Bourne shell; with standard configuration a nightmare for beginning users.

The file /etc/shells gives an overview of known shells on a Linux system:



Your default shell is set in the /etc/passwd file, like this line for user mia:



To switch from one shell to another, just enter the name of the new shell in the active terminal. The system finds the directory where the name occurs using the PATH settings, and since a shell is an executable file (program), the current shell activates it and it gets executed. A new prompt is usually shown, because each shell has its typical appearance:



**Executing commands**

Bash determines the type of program that is to be executed. Normal programs are system commands that exist in compiled form on your system. When such a program is executed, a new process is created because Bash makes an exact copy of itself. This child process has the same environment as its parent, only the process ID number is different. This procedure is called forking.

After the forking process, the address space of the child process is overwritten with the new process data. This is done through an exec call to the system.

The fork-and-exec mechanism thus switches an old command with a new, while the environment in which the new program is executed remains the same, including configuration of input and output devices, environment variables and priority. This mechanism is used to create all UNIX processes, so it also applies to the Linux operating system. Even the first process, init, with process ID 1, is forked during the boot procedure in the so-called bootstrapping procedure.

**Shell built-in commands**

Built-in commands are contained within the shell itself. When the name of a built-in command is used as the first word of a simple command, the shell executes the command directly, without creating a new process. Built-in commands are necessary to implement functionality impossible or inconvenient to obtain with separate utilities.

Bash supports 3 types of built-in commands:

Bourne Shell built-ins:

:, ., break, cd, continue, eval, exec, exit, export, getopts, hash, pwd, readonly, return, set, shift, test, [, times, trap, umask and unset.

Bash built-in commands:

alias, bind, builtin, command, declare, echo, enable, help, let, local, logout, printf, read, shopt, type, typeset, ulimit and unalias.

Special built-in commands:

When Bash is executing in POSIX mode, the special built-ins differ from other built-in commands in three respects:

Special built-ins are found before shell functions during command lookup.

If a special built-in returns an error status, a non-interactive shell exits.

Assignment statements preceding the command stay in effect in the shell environment after the command completes.

The POSIX special built-ins are :, ., break, continue, eval, exec, exit, export, readonly, return, set, shift, trap and unset.

Most of these built-ins will be discussed in the next chapters. For those commands for which this is not the case, we refer to the Info pages.

**Executing programs from a script**

When the program being executed is a shell script, bash will create a new bash process using a fork. This subshell reads the lines from the shell script one line at a time. Commands on each line are read, interpreted and executed as if they would have come directly from the keyboard.

While the subshell processes each line of the script, the parent shell waits for its child process to finish. When there are no more lines in the shell script to read, the subshell terminates. The parent shell awakes and displays a new prompt.

# Incident Response

## IR / HUNT Flowchart

Host

Network

Event Occurs

Determine outcome

DCOT

CTI

False Positive

True Positive

Activate IR

White List

Report

Close

Host

Network

Determine cause

Posture of Host

Escalate

Remediate

Close

Report / CTI

Close

## DCI Mission Phases

### Mission Planning Phase:

The Host Countermeasures (HC) role which forms part of the Detection and Counter Intrusion (DCI) team, works to gather information to develop an accurate network picture, obtain network access and implement host sensors. The information gathered is used to develop the defended asset list (DAL) and from potential threats to the mission.

Obtain and analyse documentation (network diagrams/configs/inventories) from the LND

Develop and present technical questionaries to the local network defender (LND) to gather network and mission overview and develop DAL

Obtain administrative credentials for network access (and timeframes for use)

Obtain list of active/inactive/privilege type of users on the network, ensure no user has only an admin account

Identify and adhere to the LND’s change management process.

Define and obtain pre-approved actions

Domain admin access

Admin credentials for specific servers/services/applications

Actions on specific malware threats (delete/disable user account)

Develop the baseline checklist prioritising key assets to the mission

Liaise with cyber threat intelligence (CTI) on the probable threat and obtain signatures/artifacts and TTP’s of threat actors

Identify mission, map mission to mission critical systems

Identify mission specific training and implement a training plan

### Survey Phase:

The survey phase involves developing a more precise network picture and understanding which is often due to incomplete documentation and diagrams. This includes placing network/host sensors and analysing data from logs. An accurate network picture is then developed by the DCI team. Host countermeasures will analyse hosts to identify key terrain – cyber (KT-C) in tandem with provided intelligence. This then leads to development of the risk mitigation plan (RMP) and the countermeasures

Obtain permission to operate/connect for all team members/tools/equipment

Protect own assets.

Develop a sensor placement plan

Analyse hosts and identify risks and recommendations for hardening to CPT HQ for RMP development.

Continuous logging and change management

Analysis of mission, continuous mapping of mission to mission critical systems

Check backups of configurations, i.e webservers for quick recovery (consult LND)

Passive scanning of network

Packet captures, identify hosts/servers/endpoints

Draw/Update diagram in Visio/Word for team reference

Continually monitor for suspicious behaviour and report to CPT HQ/LND

Active scanning of network

Roll out Host agents/HIDS for scalable analysis

Probe hosts/servers for services/ports.

Periodically scan network segments for new hosts.

Baseline/obtain host registry entries, user accounts, processes, network connections, file hashes, cron jobs, web/sec log files and store in secure area for future reference.

Analyse enumerated host data, comparing to baseline data

Identify vulnerabilities (i.e Nessus/Open vas) & misconfigurations

Draw/Update diagrams in Visio/Word for team reference

Implement/review preapproved actions as required

Monitor HIDS, GRR, Kibana for suspicious host activity

Continually monitor for suspicious behaviour and report to CPT HQ/LND

Investigate/act/report on suspicious behaviour as required

### Secure Phase:

During the previous two phases (Plan & Secure) the DCOT identified risks and mitigations which were forwarded to DCOT HQ to develop the risk mitigation plan. This phase involves the implementation of hardening measures that have been granted approval. Furthermore, the mission defence plan (MDP) is developed in this phase which includes more targeted recommendations to protecting the mission critical systems.

Host hardening in accordance with approved measures.

Confirmation that sensors are functioning correctly (HID’s, HIP’s) IAW CTI advice.

Confirm log and network capture forwarding is functional and relevant to CTI.

Confirm relevant Dashboards are implanted on Kibana.

Conduct incident response and reporting as anomalies arise.

Conduct triage takings as directed by Host Countermeasures team lead, DCOT HQ and CTI.

Document, report and follow change management process meticulously.

Respond to updated threat picture from CTI, passing hardening recommendations and mitigations to DCI HQ

Attempt to identify threats and TTP’s iaw pyramid of pain diagram in the next phase of this SOP

### Protect Phase:

This phase is a more refined detection and counter intrusion phase in defending the network. It involves protecting the network in a targeted manner in line with the MDP and RMP developed in the previous phases. The tasks from the previous phase are still relevant and in play, which involves the continuous monitoring and triaging of active threats on the network.

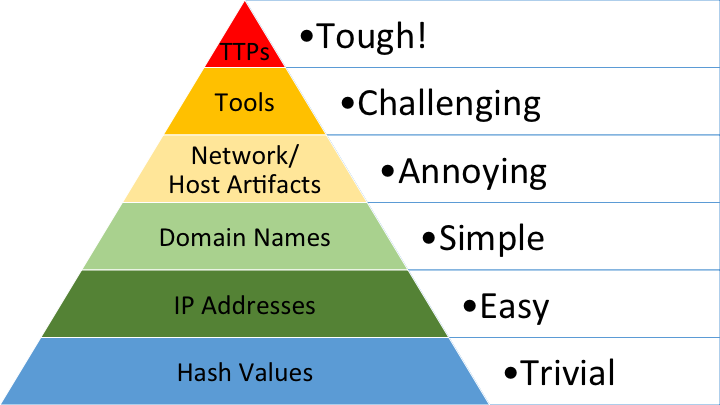
Active hunting/analysis of anomalies and incident response/triage

Host hardening in accordance with approved measures.

Targeted hunting (mission critical systems) and targeted triage.

Continuous Logging and change management.

Identifying and sharing artefacts:  
Utilising David Bianco’s pyramid of pain, the team should share artefacts with other team members as soon as possible, starting with the low hanging fruit from the base of the pyramid to the top. Progress to identification of TTP’s should be made in collaboration with CTI team and usage of the Mitre ATT&CK Enterprise matrix for Windows and Linux



Preapproved actions and the threat will determine the response of the team members depending on the artefact. Further data gathering and analysis may be required with recommendations of mitigations passed to HQ with the impact to the systems. Consideration will be given to implement further mitigations.

### Recover Phase:

The recovery phase of the CPT identifies residual risk to the defended network and hands responsibility back to the local network defenders and SO.

Removal of implemented sensors/forwarders as required

Removal of DCOT credentials supplied

Provide contact details

Provide training recommendations to the LND’s on remaining systems

## Initial DCI investigation methodology

It has previously been identified that a decisive lack of methodology has been employed by DCI operators when investigating potentially compromised machines. To mitigate this and aid in the information flow to DCOT team leaders the following methodology (below) should be employed. This will provide team leaders with clarity as to what has been conducted by the DCI operator in the initial triage of a machine.

**DCI initial survey procedure:**

This methodology is best employed on a system in which nefarious traffic has been identified by an IDS/SIEM.

1. **Conduct analysis of incoming and outgoing connections** to the machine being investigated (netstat -ano). Particular attention should be made to ports being used and a trust but verify mindset should be employed.
2. **Conduct analysis of processes running** and identify if anything nefarious may be running (tasklist, ps -l). If a connection in the previous step is irregular, attribute the PID(s) to the connection.
3. If a nefarious process is identified, **confirm with an LND what the process is**. In the event an LND cannot attribute the process to a system further analysis is required.
4. If the LND cannot identify the process, run process monitor tool (sysinternals) to **view any subprocess’s or child items** of the process. Resist the urge to kill the process as this will hinder the larger picture for investigating the issue.
5. If the above steps have not rendered any success in IOC discovery, compare a known good baseline to the current system status. This can be achieved with the use of the powershell cmdlet file compare or a script. The script will display any discrepancies or changes to the baseline but analysts should pay attention to new registry keys, scheduled tasks & cron jobs.
6. At this stage you should be able to confidently brief the relevant team lead on your findings.

If this procedure has not rendered any results deeper analysis will be required to find IOC’s.

## Actions On

### Suspicious Process and/or File

**If it is connecting to an IP, internal or external**

-Determine the IP/PID relationship using available process listing and IP activity tools native or otherwise.

**If it is running exclusively in memory**

-Dump/Scrape memory and analyse finding using memory analysis tools. Determine if an existing process has been injected into.

**Location of any source binary**

-Locate source binary of process by leveraging native administrative tools or scripts.

**Method of Persistence**

-Inspect all known locations that allow executables to be run without user interaction. Analyse system settings and configurations for anomalies that would allow an unknown or undocumented executable to run without user interaction.

**Enumerate Binary Hash**

-Hash binary using available tools and/or methods. Match hash to known malicious malware using open source intelligence.

**If the machine requires isolating**

-Using network sniffers/binary analysis methods, and/or open source intelligence, determine severity of binaries effects and whether the machine may be removed from the network.

**Purpose/Activity of any malicious process/Binary**

-Use binary analysis tools, techniques and open source intelligence to determine purpose of malicious binary and how it achieves its purpose.

**Any effects or changes made to Blue Force hosts/network**

-Compare machines, network settings, configurations and directory listings with images previously obtained during baselining activities. Analyse any local reporting of uncommon system behaviour and determine its origin.

### Suspicious Traffic

**The foreign IP address(es)**

-In conjunction with CTI Operators and Local Defenders investigate the malicious IP address and determine if it has a history on the network or anywhere else. Consult Open source intelligence.

**The local IP address(es)**

-Determine the location and use of the Blue Force machine(s) being attacked via RFI’s, Documentation or Local Defenders.

**The function of the local machine**

-Determine whether the Blue Force Machine being attacked is a work station or a server hosting a critical service. Determine effects of losing the machine.

**The nature of the traffic being observed**

-Determine whether information is being extracted, if a beacon is being observed, Active Intruder or a Port Scan etc. Analyse with traffic analysis tools and investigate historical traffic if applicable.

**The source process and/or binary of the network activity**

-After determine that a Blue Force machine has been compromised, in conjunction with Host Operators hunt for binary IAW Binary/Process methodology.

### Suspicious E-Mail

**The Sender**

-Research senders address using open source intelligence and other available means to develop an intelligence picture.

**The Recipient**

-Determine whether recipient was randomly targeted or phished with intent. Determine whether recipient has compromised themselves and their workstation by poor practice.

**If malware is being delivered via a malicious link**

**-**Determine whether recipient has clicked the link and conduct Process/Binary and/or Network Traffic Actions as necessary. Report link to CTI Operators for further investigation.

**If malware is being delivered via a malicious binary**

**-**Determine whether recipient has clicked the link and conduct Process/Binary and/or Network Traffic Actions as necessary. Report binary to CTI Operators for further investigation

**If an attempt of social engineering is being attempted**

**-**In the absence of a suspicious link or binary, is the sender attempting to impersonate another in an attempt to leverage information from the target.

**Intended effect on Blue Force hosts/Network**

-In conjunction with CTI Operators and any information gained from link or binary exploitation, determine intended effect on victim machine/network

### Malicious Binary/Process

**In the event of identifying a potentially suspicious process or file CPT Operators will determine where applicable or possible, the following.**

-If it is connecting to an IP, internal or external

-If it is running exclusively in memory

-Location of any source binary

-The method of persistence

-The malicious binaries hash

-If the machine requires isolating

-The purpose/activity of any malicious binary

-Any effects or changes made to Blue Force hosts/network

**If any or all of the above have been determined the appropriate CPT Operator in conjunction with LND’s and System Owners will remediate enemy effects where possible by carrying out the following actions.**

-Report Incident through Chain-of-Command

-Reversing and/or addressing any changes made to Blue Force machines/network by the malicious binary or process

-Ending of process and/or removal of binary

-Add any artifacts to known IOC’s

-Determine how Blue Force machine was initially compromised

-Refer any relevant network traffic to Network Operators

-Report all actions taken and artifacts collected to relevant personnel and CTI Operators.

**(Further detail of Methodology outlined in Annex)**

**Determine and Implement Mitigation strategies in conjunction with Local Defenders and System Owners, Including but not limited to**

**-**Add any IOC/hash to any commercial AV implemented

-Implementation of an IPS, IDS or both

-Creation or updating OF IDS/IPS rules

-Black and Whitelisting

-Altering system and network policies

-Updating or patching of vulnerable services, application or Operating Systems

-Justification of excessive, vulnerable or unused services

### Malicious Network Activity

**In the event of identifying potentially suspicious network traffic the CPT Operator will determine where applicable or possible, the following.**

The foreign IP address(es)

The local IP address(es)

The function of the local machine

The nature of the traffic being observed

The source process and/or binary of the network activity

**If any or all of the above have been determined, the appropriate CPT Operator in conjunction with Local Defenders and System Owners will remediate enemy effects where possible by carrying out the following actions.**

-Report Incident through Chain-of-Command

-Inspecting previous traffic captures for any history of this activity

-Reversing and/or addressing any changes made to Blue Force hosts, segments and/or servers

-Determine scope, sensitivity and/or nature of any information leak if applicable

-Refer any relevant information to host operators for IOC hunt

-Report all actions taken and artifacts collected to relevant personnel and CTI Operators.

**(Further detail of Methodology outlined in Annex)**

**Determine and Implement Mitigation strategies in conjunction with Local Defenders and System Owners, Including but not limited to**

**-**Add any IOC/hash to any commercial AV implemented

-Implementation of an IPS, IDS or both

-Creation or updating OF IDS/IPS rules

-Black and Whitelisting

-Altering system and network policies

-Updating or patching of vulnerable services, application or Operating Systems

-Justification of excessive, vulnerable or unused services

### Malicious E-Mail

**In the event of identifying a potentially suspicious e-mail(s) CPT Operators will determine where applicable or possible, the following.**

-The sender

-The recipient

-Intended effect on Blue Force hosts/network

-If malware is being delivered via a malicious link

-If malware is being delivered via a malicious binary

-If malware is being delivered via any other means

**If any or all of the above have been determined the appropriate CPT Operator in conjunction with LND’s and System Owners will remediate enemy effects where possible by carrying out the following actions.**

-Report Incident through Chain-of-Command

-If malware has been delivered, refer to process file Actions-On

-Add any artifacts to known IOC’s

-Report all actions taken and artifacts collected to relevant personnel and CTI Operators.

**(Further detail of Methodology outlined in Annex)**

**Determine and Implement Mitigation strategies in conjunction with Local Defenders and System Owners, Including but not limited to**

**-**Add any IOC/hash to any commercial AV implemented

-Implementation of an IPS, IDS or both

-Creation or updating OF IDS/IPS rules

-Black and Whitelisting

-Altering system and network policies

-Updating or patching of vulnerable services, application or Operating Systems

-Justification of excessive, vulnerable or unused services

## Linux commands for IR

To identify whether there is an account in your system that may be suspect. The below cat command usually **fetches all information about user accounts**.

Cat /etc/passwd

The group file displays the information of the group used by the user.

Cat /etc/group

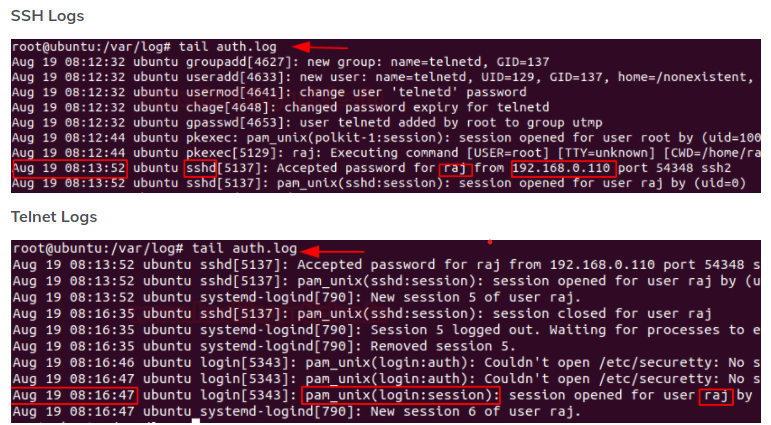
If you want to display information about user and group privileges.

Cat etc/sudoers

To view history of commands that were typed.

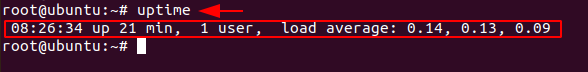
History

To identify any recent SSH or telnet connections in the system. Go to /var/log/ directory then type,

Tail auth.log

To know whether your system has been running overtime or see how long the server has been running for. The following command will also show how many users have been logged on, and the load average of the system.

Uptime



To check detailed information about the ram, memory space available, buffers and swap on the system.

Cat /proc/meminfo

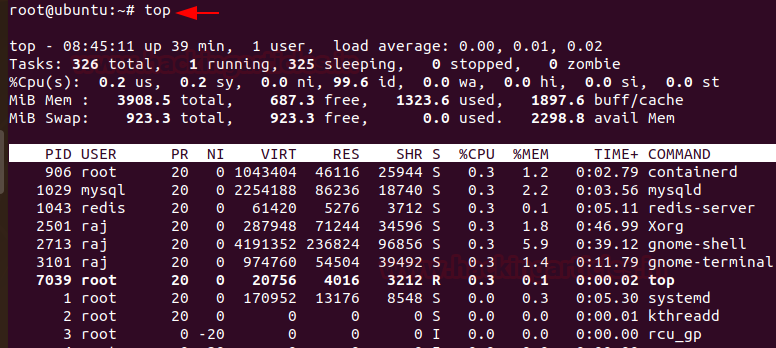
To check if there are any unknown mounts on your system.

Cat /proc/mounts

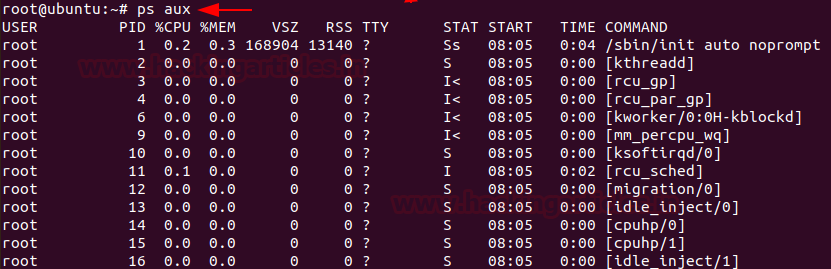
To get a snapshot of all the running processes including the associated PID, TTY and CMD.

ps

To get a dynamic and real time visual of all processes running and the associated PID.

Top

You can use ps aux to get more in-depth information about your running processes.

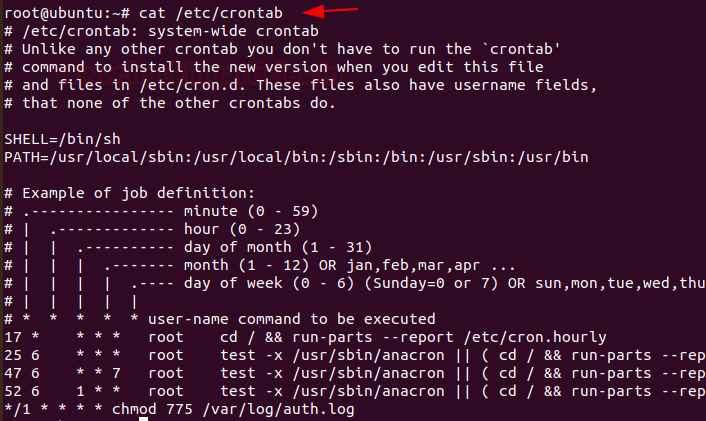
Ps aux.

To view all services running on the system.

Service --status-all

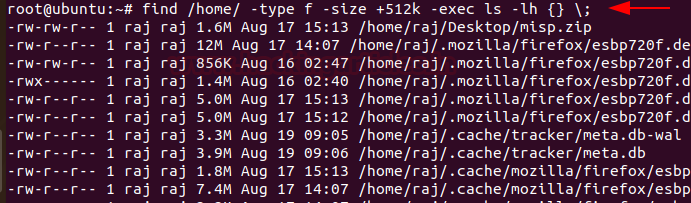
To look for any suspect scheduled tasks or jobs.

Cat /etc/crontab



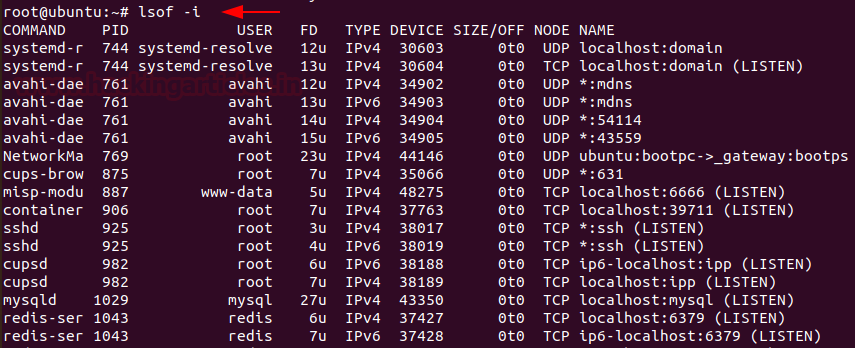
To identify any overy large files in your system and their permissions.

Find /home/ -type f –size +512k –exec ls –lh {} /;

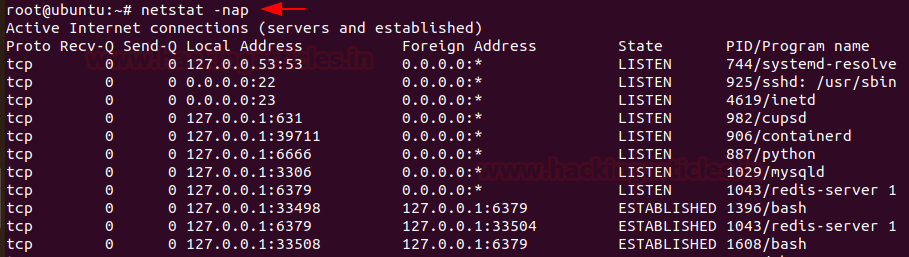


To list all the processes that are listening to ports with their PID.

Lsof –i



To display all the listening ports in the network.

Netstat -nap

To view the reports of the most recent user login.

Lastlog



To see all network interfaces in the system.

Ifconfig

To see if any keys have been added – attackers add keys to maintain SSH access.

/.ssh/authorized\_keys

Attackers can add code here to have it executed when initd restarts.

/etc/inittab

Directory names starting with . – (full stop) this is a technique used by attackers to hide directories.

Common malicious startup script locations.

* /etc/inittab
* /etc/init.d
* /etc/rc.d
* /etc/init.conf
* /etc/init

Common services and their log locations.

* /var/log/messages
* /var/log/auth.log
* /var/log/kern.log
* /var/log/cron.log
* /var/log/secure
* /var/log/mysqld.log
* /var/log/yum.log

**IP tables**

IP tables is a command line firewall utility that uses policy chains to allow or block traffic. When a connection tries to establish itself on your system, ip tables looks for a rule in its list to match it to. If it doesn’t find one, it resorts to default action.

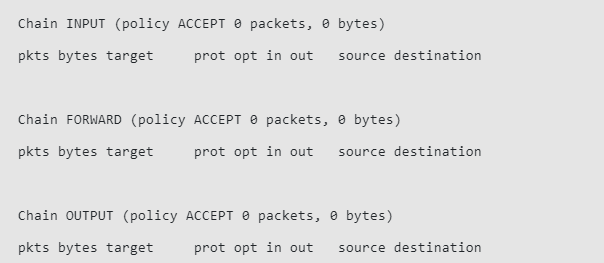
**Step 1 – install IP tables**

Ip tables comes pre-installed in most Linux distributions. However, if you don’t have it installed.

**Step 2 – check the status of your current IP tables configuration by running**



Here, the**-L** option is used to list all the rules, and **-v** is for showing the info in a more detailed format. Below is the example output:



You will now have the Linux firewall installed. At this point, you can notice that all chains are set to **ACCEPT** and have no rules. This is not secure since any packet can come through without filtering.

**Step 3 – Defining chain rules**

Defining a rule means appending it to the chain. To do this, you need to insert the **-A** option (**Append**) right after the iptables command, like so:



It will alert iptables that you are adding new rules to a chain. Then, you can combine the command with other options, such as:

* **-i** (**interface**) — the network interface whose traffic you want to filter, such as eth0, lo, ppp0, etc.
* **-p** (**protocol**) — the network protocol where your filtering process takes place. It can be either **tcp**, **udp**, **udplite**, **icmp**, **sctp**, **icmpv6**, and so on. Alternatively, you can type **all** to choose every protocol.
* **-s** (**source**) — the address from which traffic comes from. You can add a hostname or IP address.
* **–dport** (**destination port**) — the destination port number of a protocol, such as **22** (**SSH**), **443** (**https**), etc.
* **-j** (**target**) — the target name (**ACCEPT**, **DROP**, **RETURN**). You need to insert this every time you make a new rule.

#### **Filtering Packets Based on Source**

Iptables allows you to filter packets based on an IP address or a range of IP addresses. You need to specify it after the**-s** option. For example, to accept packets from **192.168.1.3**, the command would be:

You can also reject packets from a specific IP address by replacing the**ACCEPT** target with **DROP**.

#### **Filtering packets by a range of IP addresses**

If you want to drop packets from a range of IP addresses, you have to use the **-m** option and **iprange** module. Then, specify the IP address range with **–src-range**. Remember, a hyphen should separate the range of ip addresses without space, like this:

#### **Dropping all other traffic**

It is crucial to use the **DROP** target for all other traffic after defining **–dport** rules. This will prevent an unauthorized connection from accessing the server via other open ports. To achieve this, simply type:

**Step 4 – Save the changes**

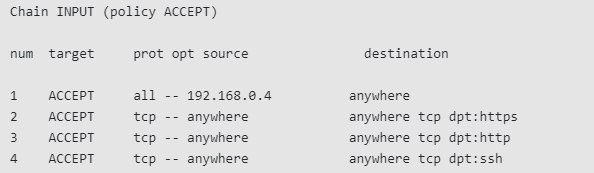
The iptables rules that we have created are saved in memory. That means we have to redefine them on reboot. To make these changes persistent after restarting the server, you can use this command:

It will save the current rules on the system configuration file, which will be used to reconfigure the tables every time the server reboots.

Deleting rules

If you want to remove all rules and start with a clean slate, you can use the **-F** option (**flush**):

This command erases all current rules. However, to delete a specific rule, you must use the -D option. First, you need to see all the available rules by entering the following command:

Then you will get a list of rules with the line number.

To delete a rule, insert the corresponding chain and the number from the list. Let’s say for this iptables tutorial, we want to get rid of **rule number three** of the **INPUT** chain. The command should be:

# Vulnerability Assessment (empty)

## Important Registry Keys

LSASS as Protected Process

Windows Registry Editor Version 5.00

[HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa]

"RunAsPPL"=dword:00000001

Disable SMBv1

Windows Registry Editor Version 5.00

[HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters]

"SMB1"=dword:00000000

Encrypt Folder context Menu

Windows Registry Editor Version 5.00

[HKEY\_CURRENT\_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced]

"EncryptionContextMenu"=dword:00000001

Disable CRL Checking on SSTP VPN Client

Windows Registry Editor Version 5.00

[HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\services\SstpSvc\Parameters]

"NoCertRevocationCheck"=dword:00000001

Disable Cert Propagation

Windows Registry Editor Version 5.00

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Policies\Microsoft\Windows\CertProp]

"CertPropEnabled"=dword:00000000

# Scripts

## Remote connections

cd "\\env-share\Trainees\_Images\CPT Gold\CRS" #move to directory of scripts and psexec

$IP = Read-host -Prompt 'Input DC ip address' #DC ip address

$creds = get-credential #asks for credentials and stores them

set-item WSMAN:\localhost\Client\TrustedHosts -Value '\*' -Force

$script = Read-host -Prompt 'What Script would you like to run? i.e Winenum.ps1'

$s = get-content -raw $script

Invoke-Command $IP -Credential ($creds) -Scriptblock {Invoke-expression $using:s} | export-csv "$IP $script.csv"

## Enumeration Script

#Main Menu Function, add options as needed.

function Show-Menu

{

param (

[string]$Title = 'Forensic Scripts'

)

Clear-Host

Write-host `n

Write-Host -f Green "================ $Title ================"

Write-Host `n

Write-Host -f DarkCyan "1: Ping Sweep Script."

Write-Host -f DarkCyan "2: File Search and Hash."

Write-Host -f DarkCyan "3: Enum Script."

Write-Host -f DarkCyan "4: Remote Host Processes."

Write-Host -f DarkCyan "5: Remote Host Connections."

Write-Host -f DarkCyan "6: Remote Host Local Users/Groups."

Write-Host -f DarkCyan "7: Remote Host Services."

Write-Host -f DarkCyan "8: Remote Host Scheduled Tasks."

Write-Host -f DarkCyan "9: PsSession to remote host."

Write-Host -f DarkCyan "10: Kill Process on remote host."

Write-Host -f Red `n "Q: Press 'Q' to quit."

}

#Functions

Function Ping-Sweep {

Clear-Host

Write-Host `n

$iprange=Read-Host "Please enter IP range to Scan (eg. 10.10.10 or 192.168.0)"

$ping = new-object System.Net.NetworkInformation.Ping

$pingselection = Read-Host "Would you like to save Ping Results to a file? (Y or N)"

switch ($pingselection)

{

'Y' {

'Your file will be saved to Your Desktop as PingResults.txt'

'Scan Running, Please wait....'

1..254 |%{$ping.send("$iprange.$\_",1) |where status -eq Success|

%{ "{0}" -f $\_.Address}} | Out-File -Append -FilePath $Env:USERPROFILE\Desktop\PingResults.txt

} 'N' {

'Your results will be displayed below'

1..254 |%{$ping.send("$iprange.$\_",1) |where status -eq Success|

%{ "{0}" -f $\_.Address}}

}

}}

Function SearchAnd-HashFile {

Clear-Host

Write-Host `n

$filecheckselection = Read-Host "Is the file on a remote host? (Y or N)"

switch ($filecheckselection)

{

'Y' {

$remotehost = Read-Host "Please enter the remote host IP"

$THip = [string]$remotehost

Set-Item WSMan:\localhost\Client\TrustedHosts -Value $THip -Force

$filename = Read-Host "Please enter the filename (eg. hack.exe)"

$creds = Get-Credential -Message "Please enter valid username and password"

'Scan Running, Please wait....'

Invoke-Command -ComputerName $remotehost -Credential $creds -ScriptBlock {

$f = Get-ChildItem -Path C:\ -Include $using:filename -Force -Recurse -ErrorAction SilentlyContinue

Clear-Host

Write-Host "FilePath and MD5 Results for [$using:filename]:"

Get-FileHash -Path $f -Algorithm MD5 | Select-Object path,hash | fl

}

} 'N' {

$filename = Read-Host "Please enter the filename (eg. hack.exe)"

'Scan Running, Please wait....'

$f = Get-ChildItem -Path C:\ -Include $filename -Force -Recurse -ErrorAction SilentlyContinue

Clear-Host

Write-Host "FilePath and MD5 Results for [$filename]:"

Get-FileHash -Path $f -Algorithm MD5 | Select-Object path,hash | fl

}

} }

Function Get-Enum {

## Questions for user - INPUT REQUIRED ##

Clear-Host

# Set-Variable -Name basicenu -Value $null

$basicenu = Read-Host 'Do you wish to identify IPCONFIG, NETSTAT, PROCESSES, SERVICES, LOCAL USERS / GROUPS & REGISTRY KEYS? (Y or N)'

# Set-Variable -Name basicenuAD -Value $null

$basicenuAD = Read-Host 'Do you wish to identify ADUSERs & key DOMAIN ADMIN GROUPS? (Y or N)'

# Set-Variable -Name HostIPs -Value $null

$HostIPs = Read-Host 'List ALL remote IP addresses you wish to run this against? (Eg: 192.168.1.10,192.168.1.20)'

$THip = [string]$HostIPs

Set-Item WSMan:\localhost\Client\TrustedHosts -Value $THip -Force

$HostIPs = $HostIPs -replace '[,]',"`n"

# Set-Variable -Name creds -Value $null

$creds = Get-Credential

# Set-Variable -Name date -Value $null

$date = Get-Date -Format "dd/MM/yyyy-HH:mm:ss"

## Basic Enumeration Commands ##

Switch ($basicenu) {

## YES, run the following commands ##

'Y' {

$HostIPs | % {

Invoke-Command -Credential $creds -ComputerName $\_ -ScriptBlock {

Write-Output "-------------------------------------- $using:\_ Machine --------------------------------------"

## All ipconfig settings ##

Write-Output "-----IPCONFIG-----"

ipconfig /all

## All TCP & UDP connections

Write-Output "`n`n-----NETSTAT-----"

netstat -ano

## All local processes ##

Write-Output "`n`n-----PROCESSES-----"

Get-Process

## All local services and states ##

Write-Output "`n`n-----SERVICES-----"

Get-Service | Select-Object Status,Name

## All local user accounts & whether or not they are enabled/disabled ##

Write-Output "`n`n-----LOCAL USERS-----"

Set-Variable -Name PSversionU -Value $null

$PSversionU= ((Get-Host).Version).Major

IF ($PSversionU -gt 2) {

Get-LocalUser | Select-Object Name,Enabled

}

ELSE {

wmic useraccount list brief

}

## Who is in the local Administrators Group ##

Write-Output "`n`n-----LOCAL ADMIN GROUP-----"

Set-Variable -Name PSversionG -Value $null

Set-Variable -Name LocalGroup -Value $null

$PSversionG = ((Get-Host).Version).Major

IF ($PSversionG -gt 2) {

$LocalGroup = (Get-LocalGroup).Name

$LocalGroup | % {

Get-LocalGroupMember -Group $\_

}

}

ELSE {

wmic path win32\_groupuser

}

Write-Output "`n`n-----REGISTRY KEYS-----"

## Creates an array containing all possible Run Key locations ##

Set-Variable -Name RunKeys -Value $null

$RunKeys = "HKLM:\Software\Microsoft\Windows\CurrentVersion\Run\",

"HKLM:\Software\Microsoft\Windows\CurrentVersion\RunOnce\",

"HKLM:\Software\Microsoft\Windows\CurrentVersion\RunServices\",

"HKLM:\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce\",

"HKCU:\Software\Microsoft\Windows\CurrentVersion\Run\",

"HKCU:\Software\Microsoft\Windows\CurrentVersion\RunOnce\",

"HKCU:\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup\",

"HKU:\.Default\Software\Microsoft\Windows\CurrentVersion\Run\",

"HKU:\.Default\Software\Microsoft\Windows\CurrentVersion\RunOnce\",

"HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Winlogon",

"HKLM:\System\CurrentControlSet\Services\",

"HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit",

"HKLM:\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\run\",

"HKCU:\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\run\ "

## Disable standard error output = can create problems within your Keys variable (will fill it up with RED SHIT) ##

$ErrorActionPreference = 'silentlycontinue'

$RunKeys | ForEach {echo "`n$\_" ((Get-Item -Path $\_) | Format-Table)}

} | Out-File -FilePath "$env:USERPROFILE\Desktop\1.txt"

}

}

## NO, do not run the basic enumeration commands ##

'N' {

}

}

Switch ($basicenuAD) {

'Y' {

$HostIPs | % {

Invoke-Command -Credential $creds -ComputerName $\_ -ScriptBlock {

## Who are the Active Directory Users in the system ##

## NOTE: This could be a very large output depending upon the size of the organisation ##

Write-Output "`n`n-----AD USERS-----"

Get-ADUser -Filter \* | Select-Object Name,Enabled

## Who are the Active Directory Administrators group ##

Write-Output "`n`n-----AD GRP MBR's ADMINS-----"

Get-ADGroupMember -Identity 'Administrators' | Select-Object name,objectClass

## Who are the Active Directory Domain Administrators group ##

Write-Output "`n`n-----AD GRP MBR's DOMAIN ADMINS-----"

Get-ADGroupMember -Identity 'Domain Admins' | Select-Object name,objectClass

## Who are the Active Directory Schema Administrators group ##

Write-Output "`n`n-----AD GRP MBR's SCHEMA ADMINS-----"

Get-ADGroupMember -Identity 'Schema Admins' | Select-Object name,objectClass

## Who are the Active Directory Enterprise Administrators group ##

Write-Output "`n`n-----AD GRP MBR's ENTERPRISE ADMINS-----"

Get-ADGroupMember -Identity 'Enterprise Admins' | Select-Object name,objectClass

} | Out-File -Append "$env:USERPROFILE\Desktop\1.txt"

}

}

## NO, do not run the AD enumeration commands ##

'N' {

}

}

}

Function Kill-RemoteProcess {

Clear-Host

Write-Host `n

$processip=Read-Host "Please enter the host IP address"

$THip = [string]$processip

Set-Item WSMan:\localhost\Client\TrustedHosts -Value $THip -Force

$processname = Read-Host "Please enter the name of the process to kill (wildcards accepted)"

$creds = Get-Credential -Message "Please enter valid username and password"

$processselection = Read-Host "Ending a critical process can cause the host to crash. Do you want to continue? (Y or N)"

switch ($pingselection)

{

'Y' {

'Kill Process Running, See below for processes killed....'

Invoke-Command -ComputerName $processip -Credential $creds -ScriptBlock {

$p = Get-Process | where Name -Like "$using:processname"

Stop-Process -InputObject $p

Get-Process | Where-Object {$\_.HasExited} }

} 'N' {

}

}

}

#Main Menu Loop which will only quit when 'q' is entered

do

{

Show-Menu

Write-Host `n

$selection = Read-Host "Please make a selection"

switch ($selection)

{

'1' {

Clear-Host

Ping-Sweep

} '2' {

Clear-Host

SearchAnd-HashFile

} '3' {

Clear-Host

Get-Enum

} '4' {

Clear-Host

Write-Host `n

$hostip=Read-Host "Please enter IP of remote host"

$THip = [string]$hostip

Set-Item WSMan:\localhost\Client\TrustedHosts -Value $THip -Force

$creds = Get-Credential -Message "Please enter valid Credentials"

Invoke-Command -Credential $creds -ComputerName $hostip -ScriptBlock {

Get-Process }

} '5' {

Clear-Host

Write-Host `n

$hostip=Read-Host "Please enter IP of remote host"

$THip = [string]$hostip

Set-Item WSMan:\localhost\Client\TrustedHosts -Value $THip -Force

$creds = Get-Credential -Message "Please enter valid Credentials"

Invoke-Command -Credential $creds -ComputerName $hostip -ScriptBlock {

Get-NetTCPConnection }

} '6' {

Clear-Host

Write-Host `n

$hostip=Read-Host "Please enter IP of remote host"

$THip = [string]$hostip

Set-Item WSMan:\localhost\Client\TrustedHosts -Value $THip -Force

$creds = Get-Credential -Message "Please enter valid Credentials"

Invoke-Command -Credential $creds -ComputerName $hostip -ScriptBlock {

Get-LocalUser | Select-Object Name,Enabled

Get-LocalGroupMember -Group Administrators }

} '7' {

Clear-Host

Write-Host `n

$hostip=Read-Host "Please enter IP of remote host"

$THip = [string]$hostip

Set-Item WSMan:\localhost\Client\TrustedHosts -Value $THip -Force

$creds = Get-Credential -Message "Please enter valid Credentials"

Invoke-Command -Credential $creds -ComputerName $hostip -ScriptBlock {

Get-Service | Select-Object Status,Name }

} '8' {

Clear-Host

Write-Host `n

$hostip=Read-Host "Please enter IP of remote host"

$THip = [string]$hostip

Set-Item WSMan:\localhost\Client\TrustedHosts -Value $THip -Force

$creds = Get-Credential -Message "Please enter valid Credentials"

Invoke-Command -Credential $creds -ComputerName $hostip -ScriptBlock {

Get-ScheduledTask }

} '9' {

Clear-Host

Write-Host `n

$hostip=Read-Host "Please enter IP of remote host"

$THip = [string]$hostip

Set-Item WSMan:\localhost\Client\TrustedHosts -Value $THip -Force

$creds = Get-Credential -Message "Please enter valid Credentials"

Enter-PSSession -ComputerName $hostip -Credential $creds

} '10' {

Clear-Host

Kill-RemoteProcess

}

}

pause

}

until ($selection -eq 'q')

## Baseline Script

<##############################################################################

.SYNOPSIS

Creates a folder with files which capture the current OS state.

.DESCRIPTION

Creates a folder filled with CSV, XML and TXT files which capture

the current operational state of the computer, such as running

processes, services, user accounts, audit policies, shared folders,

networking settings, and more. These files can be used for threat

hunting, auditing, compliance, and troubleshooting purposes.

The output folder will be named after the local host and the current

date and time, e.g., .\COMPUTERNAME-Year-Month-Day-Hour-Minute.

The commands creating the files are simple and can be edited by

those without advanced scripting skills. The files produced also

can be compressed, copied, analyzed and compared without expensive

forensics or analysis tools, such as Notepad++ or WinMerge.

Script requires PowerShell 3.0, Windows 7, Server 2008, or later,

and must be run with administrative privileges.

Most commands are built into PowerShell 3.0 and later, but some

tools will need to be installed first in order to use them, such

as AUTORUNSC.EXE (http://www.microsoft.com/sysinternals/) and

SHA256DEEP.EXE (http://md5deep.sourceforge.net), which are not

required, but very useful for baselines. Be aware, though, that

producing thousands of file hashes may require a long time.

.PARAMETER OutputParentFolder

Optional path to the parent folder under which a new subfolder will

be created to hold the baseline files. This is not the path to

the output folder itself, which will be automatically created, but

to its parent folder. Defaults to $PWD, the present directory.

Write access permission is required to the output folder.

.PARAMETER TextFileOutput

Forces all output files to be flat TXT files instead of XML.

.PARAMETER Verbose

Show progress information as the script is running.

.NOTES

Version: 4.5

Updated: 26.Oct.2017

Author: Enclave Consulting LLC (http://www.sans.org/sec505)

Legal: 0BSD.

Requires -Version 3.0

##############################################################################>

[CmdletBinding()]

Param ([String] $OutputParentFolder = ($Pwd.Path), [Switch] $TextFileOutput)

# Verbose start time:

$StartTime = Get-Date

Write-Verbose -Message ("Started: " + (Get-Date -Format 'F'))

#.DESCRIPTION

# Helper function to write output as XML (default) or as TXT (with -TextFileOutput).

# Almost every command below pipes into this function.

function WriteOut ($FileName)

{

if ($TextFileOutput)

{

Write-Verbose -Message ("Writing to " + ($FileName + ".txt"))

$Input | Format-List \* | Out-File -Encoding UTF8 -FilePath ($FileName + ".txt")

}

else

{

Write-Verbose -Message ("Writing to " + ($FileName + ".xml"))

$Input | Export-Clixml -Encoding UTF8 -Path ($FileName + ".xml")

}

}

# Confirm that the destination PARENT folder exists:

if (-not (Test-Path -Path $OutputParentFolder -PathType Container))

{

Write-Error -Message "$OutputParentFolder does not exist or is not accessible, exiting."

Exit

}

# If this script is run with File Explorer, the present working

# directory becomes C:\Windows\System32, which is not good, so

# disallow $env:SystemRoot or anything underneath it:

if ( $OutputParentFolder -like ($env:SystemRoot + '\*') )

{

Write-Error -Message "Output folder cannot be under $Env:SystemRoot, and script must be run from within a command shell, exiting."

Exit

}

# Record present directory in order to switch back to it later,

# and attempt to switch into $OutputParentFolder now:

$PresentDirectory = $Pwd

cd $OutputParentFolder

if (-not $?){ Write-Error -Message "Could not switch into $OutputParentFolder, exiting." ; Exit }

# Set FOLDER variable to contain output files. The format will look

# like "COMPUTERNAME-2018-06-05-11-03" (computername-year-month-day-hour-minute).

$OutputFolder = $env:COMPUTERNAME + "-" + (Get-Date -Format 'yyyy-MM-dd-hh-mm')

Write-Verbose -Message "Creating $(Join-Path -Path $OutputParentFolder -ChildPath $OutputFolder)"

# Create the $Folder in the present working directory and switch into it:

mkdir $OutputFolder | out-null

if (-not $?){ Write-Error -Message "Could not create $OutputFolder, exiting." ; Exit }

cd $OutputFolder

if ($pwd.Path -ne (Join-Path -Path $OutputParentFolder -ChildPath $OutputFolder))

{ Write-Error -Message "Could not switch into $OutputFolder, exiting." ; Exit }

###############################################################################

#

# Create README.TXT file to identify this computer and baseline record.

# More text will be appended to the end of this file later in this script.

#

###############################################################################

$ReadmeText = @"

\*SYSTEM CONFIGURATION BASELINE

\*Computer: $env:COMPUTERNAME

\*HostName: $(hostname.exe)

\*Box-Date: $(Get-Date -Format 'F')

\*UTC-Date: $(Get-Date -Format 'U')

\*ZuluDate: $(Get-Date -Format 'u')

\*PVersion: $($PSVersionTable.PSVersion.ToString())

\*UserName: $env:USERNAME

\*User-Dom: $env:USERDOMAIN

"@

$ReadmeText | Out-File -Encoding UTF8 -FilePath .\README.TXT -Force

if (-not $?)

{ Write-Error -Message "Could not write to README.TXT, exiting." ; Exit }

else

{ Write-Verbose -Message "Created README.TXT" }

###############################################################################

#

# Now run whatever commands you wish to capture operational state data.

# Please add more commands and use additional tools too, always piping

# the output of any command into the WriteOut function (defined above).

#

###############################################################################

# Computer System

Get-CimInstance -ClassName Win32\_ComputerSystem | WriteOut -FileName ComputerSystem

# BIOS

Get-CimInstance -ClassName Win32\_BIOS | WriteOut -FileName BIOS

# Environment Variables

dir env:\ | WriteOut -FileName Environment-Variables

# Users

Get-CimInstance -ClassName Win32\_UserAccount | WriteOut -FileName Users

# Groups

Get-CimInstance -ClassName Win32\_Group | WriteOut -FileName Groups

# Group Members

Get-CimInstance -ClassName Win32\_GroupUser | WriteOut -FileName Group-Members

# Password And Lockout Policies

net.exe accounts | Out-File -FilePath Password-And-Lockout-Policies.txt

# Local Audit Policy

auditpol.exe /get /category:\* | Out-File -FilePath Audit-Policy.txt

# SECEDIT Security Policy Export

secedit.exe /export /cfg SecEdit-Security-Policy.txt | out-null

# Shared Folders

Get-SmbShare | WriteOut -FileName Shared-Folders

# Networking Configuration

Get-NetAdapter -IncludeHidden | WriteOut -FileName Network-Adapters

Get-NetIPAddress | WriteOut -FileName Network-IPaddresses

Get-NetTCPConnection -State Listen | Sort LocalPort | WriteOut -FileName Network-TCP-Listening-Ports

Get-NetUDPEndpoint | Sort LocalPort | WriteOut -FileName Network-UDP-Listening-Ports

Get-NetRoute | WriteOut -FileName Network-Route-Table

nbtstat.exe -n | Out-File -FilePath Network-NbtStat.txt

netsh.exe winsock show catalog | Out-File -FilePath Network-WinSock.txt

Get-DnsClientNrptPolicy -Effective | WriteOut -FileName Name-Resolution-Policy-Table

# Windows Firewall and IPSec

Get-NetConnectionProfile | WriteOut -FileName Network-Connection-Profiles

Get-NetFirewallProfile | WriteOut -FileName Network-Firewall-Profiles

Get-NetFirewallRule | WriteOut -FileName Network-Firewall-Rules

Get-NetIPsecRule | WriteOut -FileName Network-IPSec-Rules

netsh.exe advfirewall export Network-Firewall-Export.wfw | out-null

# Processes

Get-Process -IncludeUserName | WriteOut -FileName Processes

# Drivers

Get-CimInstance -ClassName Win32\_SystemDriver | WriteOut -FileName Drivers

# DirectX Diagnostics

dxdiag.exe /whql:off /64bit /t dxdiag.txt

# Services

Get-Service | WriteOut -FileName Services

# Registry Exports (add more as you wish)

Write-Verbose -Message "Writing to registry files: \*.reg"

reg.exe export hklm\system\CurrentControlSet Registry-CurrentControlSet.reg /y | out-null

reg.exe export hklm\software\microsoft\windows\currentversion Registry-WindowsCurrentVersion.reg /y | out-null

# Generate an MSINFO32.EXE report, which includes lots of misc info

Write-Verbose -Message "Writing to MSINFO32-Report.txt"

msinfo32.exe /report MSINFO32-Report.txt

# Hidden Files and Folders

dir -Path c:\ -Hidden -Recurse -ErrorAction SilentlyContinue | Select-Object FullName,Length,Mode,CreationTime,LastAccessTime,LastWriteTime | Export-Csv -Path FileSystem-Hidden-Files.csv

# Non-Hidden Files and Folders

dir -Path c:\ -Recurse -ErrorAction SilentlyContinue | Select-Object FullName,Length,Mode,CreationTime,LastAccessTime,LastWriteTime | Export-Csv -Path FileSystem-Files.csv

# NTFS Permissions And Integrity Labels

# This file can reach 100's of MB in size, so

# we'll limit this example to just System32:

icacls.exe c:\windows\system32 /t /c /q 2>$null | Out-File -FilePath FileSystem-NTFS-Permissions.txt

###############################################################################

#

# The following commands require that various tools be installed and in the

# PATH, since they are not installed by default. Uncomment the lines after

# installing the tools.

#

###############################################################################

# Sysinternals AutoRuns; not in the PATH by default even when

# installed; get from microsoft.com/sysinternals

######### autorunsc.exe -accepteula -a -c | Out-File -FilePath AutoRuns.csv

# SHA256 File Hashes

# Takes a long time! Requires lots of space!

# Add more paths as you wish of course, this is just to get started.

# sha256deep.exe is used instead of Get-FileHash because it's faster.

######### sha256deep.exe -s "c:\\*" | Out-File -FilePath Hashes-C.txt

######### sha256deep.exe -s "d:\\*" | Out-File -FilePath Hashes-D.txt

######### sha256deep.exe -s -r ($env:PROGRAMFILES + "\\*") | Out-File -FilePath Hashes-ProgramFiles.txt

######### sha256deep.exe -s -r ($env:SYSTEMROOT + "\\*") | Out-File -FilePath Hashes-SystemRoot.txt

###############################################################################

#

# Record baseline metadata to README.TXT and Baseline-File-Hashes.csv:

#

###############################################################################

# Save info about the baseline output files to README.TXT:

'\*Finished: ' + $(Get-Date -Format 'u') | Out-File -Encoding UTF8 -Append -FilePath README.TXT

"-" \* 50 | Out-File -Encoding UTF8 -Append -FilePath README.TXT

dir | select Name,Length,LastWriteTime | Out-File -Encoding UTF8 -Append -FilePath README.TXT

# Save hashes and full paths to the baseline files to a CSV:

if (Get-Command -Name Get-FileHash -ErrorAction SilentlyContinue)

{

$hashes = dir -File | Get-FileHash -Algorithm SHA256 -ErrorAction SilentlyContinue

$hashes | Export-Csv -Path Baseline-File-Hashes.csv -Force #cannot directly pipe

}

###############################################################################

#

# Perform final tasks, such as writing to an event log, cleaning up temp files,

# compressing the folder into an archive, moving the archive into a shared folder,

# etc. This can also be done in an external wrapper script run as a scheduled task.

#

###############################################################################

# Delete any leftover temp files? What about the hashes list? (del \*.tmp)

# Set read-only bit on files created? (attrib.exe +R \*.txt)

# Write to the event log about the baseline process? (write-eventlog)

###############################################################################

#

# THIS COMMAND MUST BE LAST: Go back to the original working directory:

#

###############################################################################

Write-Verbose -Message "Saved files to $(Join-Path -Path $OutputParentFolder -ChildPath $OutputFolder)"

Write-Verbose -Message ("Finished: " + (Get-Date -Format 'F'))

$seconds = New-TimeSpan -Start $StartTime -End (Get-Date) | Select -ExpandProperty TotalSeconds

Write-Verbose -Message "Total run time = $seconds seconds"

cd $PresentDirectory

## Baseline Light Windows Host

#script to Windows Host Baseline Light

pingsweep (CSS)

#The will give us live hosts and we will use this data to remote into the live host for light baselining

$subnet = "192.168.10."

$start = 1

$stop = 255

While ($start -le $stop)

$ip="$subnet$start"

#write-host $ip

$result=test-connection $ip

remoting

Invoking

#system info

Host Name

OS Name

OS Version

Hotfixs

ipconfig

get local users # this will give us name of the local users, their password expiry date, last logon time, Account Expires, usermay chnage password

netstat -ano / Get-NetTCPConnection

processz

reg run key

schedule tasks

services that start up

hostfile

log entries user creation and prv esc

## Baseline Wrapper

<# ###########################################################

This is a simple wrapper for Baseline.ps1. Use this to

delete prior baselines older than 21 days (or whatever

number you choose) and then create a new baseline.

Edit the $SnapshotDir variable to change the path; the

New-Baseline.ps1 script must be in this folder. Edit the

shared folder path at the bottom, if you want to move the

zip to a centralized locations for all baseline zips.

To keep it simple, this script includes no error handling

or logging, so these should be added in real life.

########################################################### #>

# Local folder where the New-Baseline.ps1 script is placed and

# where the actual baseline data folders will be kept:

$BaselineDir = 'C:\Data\Baselines'

# Make that directory the current:

cd $BaselineDir

# Get a datetime object representing 21 days ago, or

# edit the number to change the number of days:

$DaysAgo = (get-date).AddDays(-21)

# Delete any old zip files:

dir -Path (Join-Path -Path $BaselineDir -ChildPath '\*.zip') |

where { $\_.LastWriteTime -lt $DaysAgo } |

Where { $\_.Name -match '^.+\-20\d\d\-\d+\-\d+\-\d+\-\d+\.zip$' } |

del -force

# Create a new baseline folder in current dir:

.\New-Baseline.ps1 -Verbose

# Get all baseline folders in current directory:

$baselinefolders = dir -Directory | Where { $\_.Name -match '^.+\-20\d\d\-\d+\-\d+\-\d+\-\d+$' }

# Compress each folder into a zip with the same name:

$baselinefolders | foreach { Compress-Archive -Path $\_.FullName -DestinationPath $\_.Name -CompressionLevel Optimal }

# Delete any baseline folders:

$baselinefolders | del -Recurse -Force

# Move all zips into a centralized share:

# dir \*.zip | Move-Item -Force -Destination \\server\share

## AD Account collection

$creds=get-credential

$DCIP = "192.168.200.1"

set-item WSMAN:\localhost\Client\TrustedHosts -Value '\*' -Force

enter-pssession -credential Administrator -computername $DCIP

#gather baseline admin list on in AD (ensure import module active directory is available)

import-module activeDirectory

get-adgroupmember "Domain Admins"

get-adgroupmember "Server Admins"

get-adgroupmember "Workstation Admins"

get-adgroupmember "Administrators" -recursive

<## user query-lastlogin time and actual name

net user User082

import-module activeDirectory

get-aduser User082

#GWMI win32\_useraccount

#get-localuser | getmember

#get-localuser | select Name,Lastlogon,SID

#>

exit

clear-item WSMAN:\localhost\Client\TrustedHosts

## Linux Host Enumeration

#!/bin/bash

##### (Cosmetic) Colour output

red=$(tput setaf 1) # Issues/Errors

green=$(tput setaf 2) # Success

yellow=$(tput setaf 3) # Warnings/Information

blue=$(tput setaf 4) # Heading

bold=$(tput bold setaf 7) # Highlight

reset=$(tput setaf 7) # Norma

# Quick Linux Local Enumeration Script

# v1.0

# Logs output to enum.log

(

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##\n"

printf "$red"

printf "$blue## $red Linux Version\n"

printf "$blue"

printf "##\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n$reset"

/bin/cat /etc/issue

printf "\n"

/bin/cat /etc/\*-release

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red Kernel Info"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/uname -ar

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red Network Info"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/cat /etc/sysconfig/network

printf "\n"

/bin/cat /etc/resolv.conf

printf "\n"

/bin/cat /etc/sysconfig/network-scripts/ifcfg-eth0

printf "\n"

/sbin/ip a

printf "\n"

/sbin/ifconfig

printf "\n"

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red File System Info"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/df -h

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red Mounted File Systems with Pretty Output"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/df -h

mount | column -t

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red /etc/fstab File Contents"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/cat /etc/fstab

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red /etc/passwd File Contents"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/cat /etc/passwd

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red /etc/passwd File Contents"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/cat /etc/shadow

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red /etc/group File Contents"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/cat /etc/group

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red /etc/sudoers File Contents"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/cat /etc/sudoers

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red root owned SUID Files"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/usr/bin/find / -user root -perm -4000 -print 2>/dev/null

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red root owned GUID Files"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/usr/bin/find / -group root -perm -2000 -print 2>/dev/null

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red World Writable Directories"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/usr/bin/find / -perm -222 -type d 2>/dev/null

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##\n"

printf "$red"

printf "$blue## $red SUID Owned by any user\n"

printf "$blue"

printf "##\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n$reset"

/usr/bin/find / -perm -4000 -o -perm -2000 -print 2>/dev/null

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##\n"

printf "$red"

printf "$blue## $red Files with no owner\n"

printf "$blue"

printf "##\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n$reset"

/usr/bin/find / -nouser -print 2>/dev/null

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##\n"

printf "$red"

printf "$blue## $red Files with no group\n"

printf "$blue"

printf "##\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n$reset"

/usr/bin/find / -nogroup -print 2>/dev/null

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red World Writable Files"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/usr/bin/find / -type f -perm 0777 2>/dev/null

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red Files Owned by Current User"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/usr/bin/find / -user $(whoami) 2>/dev/null

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red /home and /root Permissions"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/ls -ahlR /home/

/bin/ls -ahlR /root/

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red Logged on Users"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/usr/bin/w

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red Last Logged on Users"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/usr/bin/last

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red Processes Running as root"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

/bin/ps -ef | /bin/grep "[r]oot"

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red Installed Packages for RHEL / Debian Based Systems"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

# Enumarate CentOS / Ubuntu Boxes

# This is not a great way of ID'ing a box, but I'm being lazy

printf "\n"

/usr/bin/dpkg -l

printf "\n"

/usr/bin/rpm -qa

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red CentOS / RHEL Services that start at Boot"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

chkconfig --list | grep $(runlevel | awk '{ print $2}'):on

printf "\n"

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "##"

printf "\n"

printf "$red"

printf "$blue## $red List of init Scripts aka System Services"

printf "\n"

printf "$blue"

printf "##"

printf "\n"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "\n"

printf "$reset"

ls /etc/init.d/

printf "$blue"

printf '%\*s\n' "${COLUMNS:-$(tput cols)}" '' | tr ' ' '#'

printf "$reset"

printf "\n More Linux enumeration commands can be found at: $blue https://highon.coffee/docs/linux-commands \n"

printf "\n $red So long, and thanks for all the fish... \n $reset"

printf "\n"

# outputs to enum.log in current dir

) 2>&1 | /usr/bin/tee enum.log

## Compare Script

#compare files

$fileA = "C:\Enum\240900Jun2020"

$fileB = "C:\Enum\241600Jun2020"

$fileC = "C:\Enum\231600Jun2020"

if(Compare-Object -ReferenceObject $(Get-Content $fileA) -DifferenceObject $(Get-Content $fileB))

{"files are different"}

Else {"Files are the same"}

Compare-Object -ReferenceObject $(Get-Content $fileA) -DifferenceObject $(Get-Content $fileC)

## Compare File Hash

#Get then compare File Hashes

$fileA = "C:\fso\myfile.txt"

$fileB = "C:\fso\CopyOfmyfile.txt"

$fileC = "C:\fso\changedMyFile.txt"

if((Get-FileHash $fileA).hash -ne (Get-FileHash $fileC).hash)

{"files are different"}

Else {"Files are the same"}

## Portable Executable Injection

# Start a notepad process. This will be our injection target

$NotepadProcess = Invoke-CimMethod -ClassName Win32\_Process -MethodName Create -Arguments @{ CommandLine = 'notepad.exe' }

# Compile and load Windows API functions that will be needed to inject into another process

Add-Type -TypeDefinition @'

using System;

using System.Runtime.InteropServices;

namespace Win32 {

public class NativeMethods {

[DllImport("kernel32.dll", SetLastError = true)] public static extern IntPtr OpenProcess(int processAccess, bool bInheritHandle, int processId);

[DllImport("kernel32.dll", SetLastError = true)] public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, int flAllocationType, int flProtect);

[DllImport("kernel32.dll", SetLastError = true)] public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize, ref uint lpNumberOfBytesWritten);

[DllImport("kernel32.dll", SetLastError = true)] public static extern bool CloseHandle(IntPtr hObject);

[DllImport("ntdll.dll")] public static extern int RtlCreateUserThread(IntPtr ProcessHandle, IntPtr SecurityDescriptor, bool CreateSuspended, IntPtr StackZeroBits, IntPtr StackReserved, IntPtr StackCommit, IntPtr StartAddress, IntPtr StartParameter, ref IntPtr ThreadHandle, ref IntPtr ClientID);

}

}

'@

# Benign shellcode to inject into notepad - 0x90 (NOP), 0x90 (NOP), 0x90 (NOP), 0xC3 (RET)

[Byte[]] $Code = @(0x90, 0x90, 0x90, 0xC3)

# Obtain a handle to the open notepad process

$TargetProcessHandle = [Win32.NativeMethods]::OpenProcess(0x001FFFFF, $False, $NotepadProcess.ProcessId)

# Allocate read/write/execute memory in the notepad process for the shellcode

$TargetProcessBaseAddress = [Win32.NativeMethods]::VirtualAllocEx($TargetProcessHandle, [IntPtr]::Zero, $Code.Length, 0x3000, 0x40)

# Write the shellcode buffer to the notepad process

$BytesWritten = 0

$null = [Win32.NativeMethods]::WriteProcessMemory($TargetProcessHandle, $TargetProcessBaseAddress, $Code, $Code.Length, [Ref] $BytesWritten)

# Execute the shellcode in the notepad process

[IntPtr] $RemoteThreadHandle = [IntPtr]::Zero

[IntPtr] $CliendId = [IntPtr]::Zero

$null = [Win32.NativeMethods]::RtlCreateUserThread($TargetProcessHandle, [IntPtr]::Zero, $False, [IntPtr]::Zero, [IntPtr]::Zero, [IntPtr]::Zero, $TargetProcessBaseAddress, [IntPtr]::Zero, [Ref] $RemoteThreadHandle, [Ref] $CliendId)

# Cleanup

$null = [Win32.NativeMethods]::CloseHandle($RemoteThreadHandle)

$null = [Win32.NativeMethods]::CloseHandle($TargetProcessHandle)

Stop-Process -Id $NotepadProcess.ProcessId

## Hunt Script

# Description

# This script is meant to be used to help determine if a computer has been compromised.

# It checks the following items

# 1.) Displays the top 20 heaviest processes. Make sure they are all legit.

# 2.) If the hosts file has been altered the IP Addresses are displayed. The functino then requires the admin to enter the IP Addresses manually. This will close any open connections and prevent any more connections to the discovered IP Addresses.

# 3.) If an altered start page is configured it will be shown to the admin who will need to remove the setting.

# 4.) Checks local machine and current user registry for any previously unknown applications and shows the unknown apps to the admin. The admin should verify these applications are safe.

# 5.) Make sure no proxy settings have been configured/altered.

Function Search-ForCompromise {

[CmdletBinding()]

param(

[Parameter(

Mandatory=$false,

ValueFromPipeline=$True,

ValueFromPipelineByPropertyName=$True,

HelpMessage="Enter The hostname of the remote computer you want to check."

)] # End Parameter

[string[]]$ComputerName

) # End Param

BEGIN

{

# ControlAPpListFile is a list of known applications and should not cause any alarm.

$ControlAppListFile = 'K:\Configs\AppList.csv'

#ControlCUApplistFile is a list of the current users installed applications and is used as a reference

$ControlCUAppListFile = 'K:\Configs\CUAppList.csv'

# ControlHostsFile should be a copy of C:\Windows\system32\Drivers\etc\hosts If this file is ever edited we want to know it has been changed

$ControlHostsFile = 'K:\Configs\hosts'

# This variable is used for mapping the network location as a drive in order to update the files in the network locations

$NetworkShareLoationsAbove = '\\networkshare\files$'

New-PsDrive -Name K -PSProvider FileSystem -Root $NetworkShareLoationsAbove -Description 'Temporary drive mapping for Search-ForCompromise' -Scope Global -Persist -Credential (Get-Credential -Message "Enter crednetial to map drive")

} # End BEGIN

PROCESS

{

If (!($ComputerName))

{

Write-Host "Finding the heaviest running processes....`n" -ForegroundColor 'Cyan'

Get-Process | Sort-Object -Property 'CPU' -Descending | Select-Object -First 20

Read-Host "`nAbove is a list of the top heaviest processes currently running. Take note of anything unusual. Press Enter to continue"

Write-Host "`nDetermining whether or not the hosts file has been altered...." -ForegroundColor 'Cyan'

$DifferenceObject = Get-Content -Path "C:\Windows\system32\Drivers\etc\hosts"

$ReferenceObject = Get-Content -Path $ControlHostsFile

If (Compare-Object -ReferenceObject $ReferenceObject -DifferenceObject $DifferenceObject)

{

$DifferenceObject

Write-Host 'Hosts file has been altered. Take note of any IP Addresses and break their connections by completing the next steps. If the IP Addresses are not malicious enter 0 as the next value' -ForegroundColor 'Red'

[int]$NumberOfBad = Read-Host 'How many Bad IP Addresses have been added to the hosts file? Enter 0 for none. Example: 2'

[array]$IPAddressesToBlock = Read-Host "Enter the IP Addresses you wish to block through the Windows Firewall. Use a comma to separate multiple values. Example: '1.1.1.1','1.1.1.2'"

For ([int]$i = 1; $i -le $NumberOfBad; $i++)

{

Function Block-BadGuy {

[CmdletBinding()]

param(

[Parameter(

Mandatory=$True,

Position=0,

HelpMessage="Enter an IP Address that was added to the hosts file listed in the above output."

)] # End Parameter

[string[]]$IPaddress

) # End Param

If ($IPAddress)

{

ForEach ($IpAddr in $IPaddress)

{

New-NetFirewallRule -Name "Deny Inbound Connections to $IPAddress" -DisplayName "Deny Inbound Connections from $IpAddr" -Enabled True -Direction Inbound -Protocol ANY -Action Block -Profile ANY -RemoteAddress $IpAddr

New-NetFirewallRule -Name "Deny Outbound Connections to $IPAddress" -DisplayName "Deny Outbound Connections from $IpAddr" -Enabled True -Direction Outbound -Protocol ANY -Action Block -Profile ANY -RemoteAddress $IpAddr

Write-Verbose 'New Firewall rules added to block inbound and outbound connections to the malicious IP Address.'

$BadGuyProcessIDs = Get-NetTCPConnection -RemoteAddress $IpAddr | Select-Object -Property 'OwningProcess'

Foreach ($ProcessId in $BadGuyProcessIDs)

{

Stop-Process -Id $ProcessId -Force -PassThru

Write-Host "Above are the processes that were stopped which connected to the remote address.`nFirewall rules have been added to block anymore connections to those addresses." -ForegroundColor 'Cyan'

} # End Foreach

} # End ForEach

} # End If bad guy IP response

Else

{

Write-Warning "No IP Address was entered."

} # End Else

} # End Function Block-BadGuy

Block-BadGuy -IpAddress $IPAddressesToBlock -Verbose

} # End for loop

} # End if for finding an altered hosts file

Else

{

Write-Host 'Hosts file has not been altered. Moving on to next check.....' -ForegroundColor 'Green'

} # End Else

Write-Host "Checking for altered Internet Explorer homepage..." -ForegroundColor 'Cyan'

If (Get-Childitem -Path "HKCU:\software\Microsoft\Internet Explorer\Main\Start Page Redirect=\*")

{

Write-Host 'Internet Explorer start page redirect found. Make sure it is not malicious.' -ForegroundColor 'Red'

Pause

} # End if for finding start page redirect

# Checks local machine registry

$LMAppRef = Import-Csv -Path $ControlAppListFile

$LMAppDiff = Get-ChildItem -Path 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\' | Select-Object -Property PSChildName

If ($LMApplist = Compare-Object -DifferenceObject $LMAppDiff -ReferenceObject $LMAppRef -Property PsChildName | Where-Object -Property SideIndicator -like "<=" | Select-Object -Property PSChildName )

{

$LMApplist

Write-Warning 'This is a list of previously unrecorded Application Processes. Check these results to find any possibly malicous applications.'

$LMApplist | Export-Csv -Path $ControlAppListFile -Append

} # End If AppList

Else

{

Write-Host 'No previously unknown application services were found under Local Machine.' -ForegroundColor 'Green'

} # End Else

Write-Host "Checking current user registry for installed applications" -ForegroundColor 'Cyan'

$CUAppRef = Import-Csv -Path $ControlCUAppListFile

$CUAppDiff = Get-ChildItem -Path 'HKCU:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\' | Select-Object -Property PSChildName

If ($Applist = Compare-Object -DifferenceObject $CUAppDiff -ReferenceObject $CUAppRef -Property PsChildName | Where-Object -Property SideIndicator -like "<=" | Select-Object -Property PSChildName )

{

$CUApplist

Write-Host 'This is a list of previously unrecorded Application Processes. Check these results to find any possibly malicous applications.' -ForegroundColor 'Yellow'

$CUApplist | Export-Csv -Path $ControlCUAppListFile -Append

} # End if AppList

Else

{

Write-Host 'No previously unknown application services were found under Current User.'

} # End Else

Write-Host "Checking Proxy configuration" -ForegroundColor 'Cyan'

If (Get-ChildItem -Path 'HKCU:\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Proxy\*')

{

Write-Host 'Proxy settings have been configured. This may mean trouble.' -ForegroundColor 'Red'

} # End If

Else

{

Write-Host 'No proxy settings detected.' -ForegroundColor 'Green'

} # End Else

Write-Host 'Checking for Alternate Data Streams...' -ForegroundColor 'Cyan'

$ADSFiles = Get-ChildItem -Path 'C:\' -Recurse | ForEach-Object { Get-Item $\_.FullName -Stream \* } | Where-Object { ($\_.Stream -ne ':$Data') -and ($\_.Stream -ne 'Zone.Identifier') }

If ($ADSFiles)

{

ForEach ($ADSFile in $ADSFiles)

{

$ADSFilePath = $ADSFile.FileName

$ADSFileNameStream1,$ADSFileNameStream2 = ($ADSFilePath.PSChildName).Split(':')

If ($ADSFileNameStream2)

{

$DeleteOrKeep = Read-Host "Would you like to delete the Alternate Dat Stream from this file? Enter y to delete and leave blank to keep."

If ($DeleteOrKeep -like 'y')

{

Remove-Item –Path { $ADSFilePath } –Stream { $ADSFileNameStream2 }

} # End If to delete ADS

} # End If

} # End ForEach

} # End If ADS

} # End If not ComputerName

} # End PROCESS

END

{

Remove-PSDrive -Name K -PSProvider FileSystem -Scope Global -Force

} # End END

} # End Function Search-ForCompromise

## Disable Weak SSL

#Description

#This script has the option to disable the weak ciphers such as TipleDES, RC4, and disabled null. SSL 2.0, 3.0, TLS 1.0 are disabled in another option. TLSv1.2, TLSv1.3 and AES 256 get enabled and NULL encryption is always disabled with this script.

Function Disable-WeakSSL {

[CmdletBinding()]

param(

[Parameter(

Mandatory=$False,

ValueFromPipeline=$False)] # End Parameter

[Switch][Bool]$CISBenchmarkRecommendations,

[Parameter(

Mandatory=$False,

ValueFromPipeline=$False)] # End Parameter

[Switch][Bool]$WeakTLSCipherSuites,

[Parameter(

Mandatory=$False,

ValueFromPipeline=$False)] # End Parameter

[Switch][Bool]$TripleDES,

[Parameter(

Mandatory=$False,

ValueFromPipeline=$False)] # End Parameter

[Switch][Bool]$RC4,

[Parameter(

Mandatory=$False,

ValueFromPipeline=$False)] # End Parameter

[Switch][Bool]$AES128,

[Parameter(

Mandatory=$False,

ValueFromPipeline=$False)] # End Parameter

[Switch][Bool]$SSLv2,

[Parameter(

Mandatory=$False,

ValueFromPipeline=$False)] # End Parameter

[Switch][Bool]$SSLv3,

[Parameter(

Mandatory=$False,

ValueFromPipeline=$False)] # End Parameter

[Switch][Bool]$TLSv1,

[Parameter(

Mandatory=$False,

ValueFromPipeline=$False)] # End Parameter

[Switch][Bool]$TLSv11

) # End param

If (($WeakTLSCipherSuites.IsPresent) -or ($CISBenchmarkRecommendations.IsPresent))

{

Write-Verbose "Disabling Weak TLS Ciphers"

Disable-TlsCipherSuite -Name "TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_DHE\_DSS\_WITH\_AES\_256\_CBC\_SHA256" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_DHE\_DSS\_WITH\_AES\_128\_CBC\_SHA256" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_DHE\_DSS\_WITH\_AES\_256\_CBC\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_DHE\_DSS\_WITH\_AES\_128\_CBC\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_RC4\_128\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_RC4\_128\_MD5" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_NULL\_SHA256" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_RSA\_WITH\_NULL\_SHA" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_PSK\_WITH\_AES\_256\_GCM\_SHA384" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_PSK\_WITH\_AES\_128\_GCM\_SHA256" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_PSK\_WITH\_AES\_256\_CBC\_SHA384" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_PSK\_WITH\_AES\_128\_CBC\_SHA256" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_PSK\_WITH\_NULL\_SHA384" -ErrorAction SilentlyContinue | Out-Null

Disable-TlsCipherSuite -Name "TLS\_PSK\_WITH\_NULL\_SHA256" -ErrorAction SilentlyContinue | Out-Null

Write-Output "=============================================================================================="

Write-Output "| A LIST OF ALLOWED TLS CIPHER SUITES ARE BELOW |"

Write-Output "=============================================================================================="

Get-TlsCipherSuite | Format-Table -Property Name,Certificate,Exchange,Hash

} # End If WeakCiphers

Write-Verbose "Disabling NULL Ciphers"

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\NULL' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\NULL' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

If (($TripleDES.IsPresent) -or ($CISBenchmarkRecommendations.IsPresent))

{

Write-Verbose "Disabling DES Ciphers"

(Get-Item -Path 'HKLM:\').OpenSubKey('SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers', $true).CreateSubKey('DES 56/56')

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\DES 56/56' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

(Get-Item -Path 'HKLM:\').OpenSubKey('SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers', $true).CreateSubKey('Triple DES 168/168')

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\Triple DES 168/168' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

} # End If

If (($RC4.IsPresent) -or ($CISBenchmarkRecommendations.IsPresent))

{

Write-Verbose "Disabling RC4 ciphers"

(Get-Item -Path 'HKLM:\').OpenSubKey('SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers', $true).CreateSubKey('RC4 40/128')

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\RC4 40/128' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

(Get-Item -Path 'HKLM:\').OpenSubKey('SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers', $true).CreateSubKey('RC4 56/128')

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\RC4 56/128' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

(Get-Item -Path 'HKLM:\').OpenSubKey('SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers', $true).CreateSubKey('RC4 64/128')

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\RC4 64/128' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

(Get-Item -Path 'HKLM:\').OpenSubKey('SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers', $true).CreateSubKey('RC4 128/128')

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\RC4 128/128' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

} # End If

If (($AES128.IsPresent) -or ($CISBenchmarkRecommendations.IsPresent))

{

Write-Verbose "Disabling AES 128/128"

(Get-Item -Path 'HKLM:\').OpenSubKey('SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers', $true).CreateSubKey('AES 128/128')

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\AES 128/128' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

} # End If

Write-Verbose "Enabling AES 256/256"

(Get-Item -Path 'HKLM:\').OpenSubKey('SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers', $true).CreateSubKey('AES 256/256')

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\AES 256/256' -Name 'Enabled' -Value '1' -PropertyType 'DWord' -Force | Out-Null

If (($SSLv2.IsPresent) -or ($CISBenchmarkRecommendations.IsPresent))

{

Write-Verbose "Disabling SSL 2.0"

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 2.0\Server' -Force | Out-Null

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 2.0\Client' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 2.0\Server' -name 'Enabled' -value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 2.0\Client' -name 'Enabled' -value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 2.0\Server' -name 'DisabledByDefault' -value '1' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 2.0\Client' -name 'DisabledByDefault' -value '1' -PropertyType 'DWord' -Force | Out-Null

} # End If

If (($SSLv3.IsPresent) -or ($CISBenchmarkRecommendations.IsPresent))

{

Write-Verbose "Disabling SSL 3.0"

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 3.0\Server' -Force | Out-Null

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 3.0\Client' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 3.0\Server' -Name 'Enabled' -value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 3.0\Client' -Name 'Enabled' -value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 3.0\Server' -Name 'DisabledByDefault' -Value '1' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 3.0\Client' -Name 'DisabledByDefault' -Value '1' -PropertyType 'DWord' -Force | Out-Null

} # End If

If (($TLSv1.IsPresent) -or ($CISBenchmarkRecommendations.IsPresent))

{

Write-Verbose "Disabling TLS v1.0"

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.0\Server' -Force | Out-Null

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.0\Client' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.0\Server' -Name 'Enabled' -value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.0\Client' -Name 'Enabled' -value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.0\Server' -Name 'DisabledByDefault' -Value '1' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.0\Client' -Name 'DisabledByDefault' -Value '1' -PropertyType 'DWord' -Force | Out-Null

} # End If

If (($TLSv11.IsPresent) -or ($CISBenchmarkRecommendations.IsPresent))

{

Write-Verbose "Disabling TLS v1.1"

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.1\Server' -Force | Out-Null

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.1\Client' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.1\Server' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.1\Client' -Name 'Enabled' -Value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.1\Server' -Name 'DisabledByDefault' -Value '1' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.1\Client' -Name 'DisabledByDefault' -Value '1' -PropertyType 'DWord' -Force | Out-Null

} # End If

Write-Verbose "Enabling TLS 1.2"

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.2\Server' -Force | Out-Null

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.2\Client' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.2\Server' -Name 'Enabled' -Value '1' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.2\Client' -Name 'Enabled' -Value '1' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.2\Server' -Name 'DisabledByDefault' -Value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.2\Client' -Name 'DisabledByDefault' -Value '0' -PropertyType 'DWord' -Force | Out-Null

Write-Verbose "Enabling TLS 1.3"

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.3\Server' -Force | Out-Null

New-Item -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.3\Client' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.3\Server' -Name 'Enabled' -Value '1' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.3\Client' -Name 'Enabled' -Value '1' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.3\Server' -Name 'DisabledByDefault' -Value '0' -PropertyType 'DWord' -Force | Out-Null

New-ItemProperty -Path 'HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\TLS 1.3\Client' -Name 'DisabledByDefault' -Value '0' -PropertyType 'DWord' -Force | Out-Null

} # End Function Disable-WeakSSL

## DNS Zone Transfer Alert

# This is an alert that should be run in response to DNS Server event ID 6001 to alert IT admins when a DNS zone transfer occurs

$Event = Get-WinEvent -FilterHashtable @{LogName='DNS Server';ID='6001'} -MaxEvents 1

If ($Event -like $Null)

{

exit

} # End If

Else

{

$MailBody = $Event.Message + "`r`n`t" + $Event.TimeGenerated | Format-List -Property \* | Out-String

Send-Mailmessage -From "alerts@osbornepro.com" -To "me@osbornepro.com" -Subject "DNS Zone Transfer Occured" -Body $MailBody -SmtpServer mail.smtp2go.com

} # End Else

## Unusual User Sign-In Alert

# This script is used to check for users who have signed into devices that are outside their normally assigend devices

#===========================================================================

# REQUIREMENTS

#===========================================================================

# - This will require a CSV file containing a ComputerName header and Name header

# - The script needs to be run on a domain controller logging Event ID 4624

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

# Csv file containing the headers ComputerName and Name

$CsvInformation = Import-Csv -Path "$env:USERPROFILE\Documents\UserComputerList.csv" -Delimiter ','

$UserList = $CsvInformation | Select-Object -Property Name -Unique

# Who should receive the email alerts

$SmtpServer = 'smtp.outlook.com'

$AlertEmail = 'alertingemail@domain.com'

# Array of Shared Computer Names is for excluding computers that may be shared such as conference room computers that may be signed into

$SharedComputerIPs = @('10.0.1.1','10.0.2.2','10.0.3.3')

# Regex used for filtering event log

[regex]$Ipv4Regex = ‘\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b’

# Primary Domain Controller and DNS Server

$PDC = ([ADSI]”LDAP://RootDSE”).dnshostname

$FinalResult = @()

Function Get-UserSid

{

[CmdletBinding()]

param(

[Parameter(Mandatory = $True,

Position = 0,

ValueFromPipeline=$True,

ValueFromPipelineByPropertyName=$True,

HelpMessage = "Enter a SamAccountName for the user profile. Example: OsbornePro\rob.osborne"

)] # End Parameter

[string[]]$SamAccountName) # End param

$ObjUser = New-Object -TypeNaem System.Security.Principal.NTAccount($SamAccountName)

$ObjSID = $ObjUser.Translate([System.Security.Principal.SecurityIdentifier])

If (!($Null -eq $ObjSID))

{

$ObjSID.Value

} # End If

Else

{

Write-Output "[X] SID Lookup failed."

} # End Else

} # End Function Get-UserSid

ForEach ($Assignment in $UserList)

{

Write-Host "[\*] Getting SamAccountName and SID values..." -ForegroundColor 'Cyan'

$SamAccountName = ($Assignment.Name).Replace(' ','.')

$SID = Get-UserSid -SamAccountName $SamAccountName

$Name = $Assignment.Name

Write-Host "[\*] Getting computers assigned to $SamAccountName" -ForegroundColor 'Cyan'

$ResolveTheseComputerNames = $CsvInformation | Where-Object -Property 'Name' -like $Name | Select-Object -ExpandProperty 'ComputerName'

Write-Host "[\*]Translating computernames to Ip Addresses for searching the event logs." -ForegroundColor 'Cyan'

$SearchIP = @()

ForEach ($Device in $ResolveTheseCOmputerNames)

{

$Ipv4Address = (Resolve-DnsName -Name $Device -Server $PDC -Type A -ErrorAction SilentlyContinue).IPAddress

If ($Ipv4Address)

{

$SearchIP += $Ipv4Address

} # End If

} # End ForEach

$ComputerAssignments = @()

$ComputerAssignments = $SharedComputerIPs + $SearchIP

Write-Host "[\*] Getting log on events for $SamAccountName. Please wait..." -ForegroundColor 'Cyan'

[array]$UserLogonEvents = @()

# This event checks the last 24 hours (86400000)

[array]$UserLogonEvents = Get-WinEvent -LogName "Security" -FilterXPath "\*[System[EventID=4624 and TimeCreated[timediff(@SystemTime) <= 86400000]] and EventData[Data[@Name='TargetUserName']=`'$SamAccountName`']]" -ErrorAction SilentlyContinue

[array]$EventLoggedInIps = @()

# Selects one of each IP address found that was accessed

[array]$EventLoggedInIps = $UserLogonEvents.Message -Split "`n" | Select-String -Pattern $Ipv4Regex | Select-Object -Unique

[System.Collections.ArrayList]$UnusualSignInIps = @()

[System.Collections.ArrayList]$UnusualSignInHostname = @()

# Comapres the assigned computers to signed in devices

ForEach ($EventIp in $EventLoggedInIps)

{

$CompareValue = ($EventIp | Out-String).Replace('Source Network Address: ','').Trim()

# BELOW SWITCH OPTIONS SHOULD BE SET TO MATCH SUBNETS IN YOUR ENVIRONMENT THAT ARE ON WIFI OR VPN THAT CHANGE ##############################################################

Switch -Wildcard ($CompareValue)

{

"10.0.0.\*" {

$DhcpResolvedHost = Invoke-Command -HideComputerName "DHCPserver01.$env:USERDNSDOMAIN" -ScriptBlock {Get-DhcpServerv4Lease -ComputerName localhost -ScopeID '10.0.0.0'}; $SingleHost = $DhcpResolvedHost.Where({[IPAddress]$\_.Ipaddress -like $CompareValue})

}

"10.1.0.\*" {

$DhcpResolvedHost = Invoke-Command -HideComputerName "DHCPserver02.$env:USERDNSDOMAIN" -ScriptBlock {Get-DhcpServerv4Lease -ComputerName localhost -ScopeID '10.1.0.0'};; $SingleHost = $DhcpResolvedHost.Where({[IPAddress]$\_.Ipaddress -like $CompareValue})

}

"10.2.0.\*" {

$DhcpResolvedHost = Invoke-Command -HideComputerName "Dhcpserver03.$env:USERDNSDOMAIN" -ScriptBlock {Get-DhcpServerv4Lease -ComputerName localhost -IPAddress -ScopeID '10.2.0.0'}; $SingleHost = $DhcpResolvedHost.Where({[IPAddress]$\_.Ipaddress -like $CompareValue})

}

"10.3.0.\*" {

$DhcpResolvedHost = Invoke-Command -HideComputerName "Dhcpserver04.$env:USERDNSDOMAIN" -ScriptBlock {Get-DhcpServerv4Lease -ComputerName localhost -ScopeID '10.3.0.0'}; $SingleHost = $DhcpResolvedHost.Where({[IPAddress]$\_.Ipaddress -like $CompareValue})

}

Default {

Remove-Variable -Name DhcpResolvedHost -ErrorAction SilentlyContinue

}

} # End Switch

If ($Null -eq $SingleHost)

{

Try

{

$DnsCheck = ((Resolve-DnsName -Name $CompareValue -Server "$env:COMPUTERNAME.usav.org" -DnssecOk -ErrorAction SilentlyContinue).NameHost).Replace(".usav.org","")

If ($ResolveTheseComputerNames -contains $DnsCheck)

{

$ComputerAssignments += ($CompareValue)

} # End If

} # End Try

Catch

{

Write-Host "[\*] Could not resolve $CompareValue to an hostname" -ForegroundColor Cyan

} # End Catch

If (($ComputerAssignments -notcontains $CompareValue) -and ($CompareValue -notlike "10.10.10.\*")) # 10.10.10.\* can be used to exclude VPN subnets or whatever

{

$UnusualSignInIps += ($CompareValue)

$UnusualSignInHostname += ((Resolve-DnsName -Name $CompareValue -Server "$env:COMPUTERNAME.usav.org" -DnssecOk -ErrorAction SilentlyContinue).NameHost).Replace(".usav.org","")

} # End If

} # End If

Else

{

If ($ResolveTheseComputerNames -notcontains $SingleHost.Hostname.Replace("$env:USERDNSDOMAIN",""))

{

$UnusualSignInIps += $SingleHost.IPAddress

$UnusualSignInHostname += $SingleHost.Hostname.Replace("$env:USERDNSDOMAIN","")

} # End If

} # End Else

Remove-Variable -Name SingleHost,DhcpResolvedHost

} # End ForEach

If ($UnusualSignInIps)

{

$Obj = New-Object -TypeName PSObject -Property @{User=$SamAccountName; SID=$SID; IPv4Location="$UnusualSignInIps";Hostnames="$UnusualSignInHostname"}

$FinalResult += $Obj

} # End If

Else

{

Write-Host "[\*] No unexpected logon events found for $SamAccountName" -ForegroundColor 'Green'

} # End Else

} # End ForEach

# Build Email to send final results to inform admins

$Css = @"

<style>

table {

font-family: verdana,arial,sans-serif;

font-size:11px;

color:#333333;

border-width: 1px;

border-color: #666666;

border-collapse: collapse;

}

th {

border-width: 1px;

padding: 8px;

border-style: solid;

border-color: #666666;

background-color: #dedede;

}

td {

border-width: 1px;

padding: 8px;

border-style: solid;

border-color: #666666;

background-color: #ffffff;

}

</style>

"@ # End CSS

$PreContent = "<Title>NOTIFICATION: Unusual Sign In: $env:COMPUTERNAME</Title>"

$NoteLine = "This Message was Sent on $(Get-Date -Format 'MM/dd/yyyy HH:mm:ss')"

$PostContent = "<br><p><font size='2'><i>$NoteLine</i></font>"

$MailBody = $FinalResult | ConvertTo-Html -Head $Css -PostContent $PostContent -PreContent $PreContent -Body "<br>The below table contains information on users who have signed into devices they are not assigned in the last 24 hours<br><br><hr><br><br>" | Out-String

Send-MailMessage -From $AlertEmail -To $AlertEmail -Subject "Unusual Login Occurred" -BodyAsHtml -Body "$MailBody" -SmtpServer $SmtpServer

## Watch Port Scanning

# This script can be used to monitor for internal port scans on domain and/or private and/or public networks using the firewall logs.

# After the permissions are set correctly on the directory your firewall logs are stored in you may need to restart the device to apply them.

# This can be used to receive an email alert when port scans happen as well as automatically blacklist the ip address performing the port scan

# on the localhost.

# SET THESE VALUES TO RECEIVE EMAIL ALERTS WHEN DEFINING THE -EmailAlert SWITCH PARMETER

$To = "alertme@osbornepro.com"

$From = "do-not-reply@osbornepro.com"

$SmtpServer = "mail.smtp2go.com"

# This examples test to ensure the current user is a member of the local Administrators group.

Function Test-Admin {

[CmdletBinding()]

param() # End param

Write-Verbose "Verifying permissions"

$IsAdmin = ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole] "Administrator")

If ($IsAdmin)

{

Write-Verbose "Permissions verified, continuing execution"

} # End If

Else

{

Throw "[x] Insufficient permissions detected. Run this cmdlet in an adminsitrative prompt."

} # End Else

} # End Function Test-Admin

#This cmdlet tests to make sure the files do not already exist before creating them.

#The default value creates the appropriately named firewall log files in C:\Windows\System32\logfiles\Firewall directory.

Function New-FirewallLogFile

{

[CmdletBinding()]

param (

[Parameter(

Mandatory=$False,

ValueFromPipeline=$True,

ValueFromPipelineByPropertyName=$True,

HelpMessage="[H] Define the directory location to save firewall logs too. `n[E] EXAMPLE: C:\Windows\System32\LogFiles\Firewall"

)] # End Parameter

[String]$Path = "C:\Windows\System32\LogFiles\Firewall"

) # End param

BEGIN

{

Test-Admin

$FirewallLogFiles = "$Path\domainfw.log","$Path\domainfw.log.old","$Path\privatefw.log","$Path\privatefw.log.old","$Path\publicfw.log","$Path\publicfw.log.old","$Path"

New-Item -Path $Path -ItemType Directory -Force -ErrorAction SilentlyContinue | Out-Null

} # End BEGIN

PROCESS

{

Write-Output "[\*] Creating firewall log files in $Path"

New-Item -Path $FirewallLogFiles -Type File -Force -ErrorAction SilentlyContinue | Out-Null

Write-Output "[\*] Setting permissions on the log files created"

$Acl = Get-Acl -Path $FirewallLogFiles

$Acl.SetAccessRuleProtection($True, $False)

$PermittedUsers = @('NT AUTHORITY\SYSTEM', 'BUILTIN\Administrators', 'BUILTIN\Network Configuration Operators', 'NT SERVICE\MpsSvc', 'USAV\sour.pell')

ForEach ($User in $PermittedUsers)

{

$Permission = $User, 'FullControl', 'Allow'

$AccessRule = New-Object -TypeName System.Security.AccessControl.FileSystemAccessRule -ArgumentList $Permission

$Acl.AddAccessRule($AccessRule)

} # End ForEach

} # End PROCESS

END

{

$Acl.SetOwner((New-Object -TypeName System.Security.Principal.NTAccount('BUILTIN\Administrators')))

$Acl | Set-Acl -Path $FirewallLogFiles

} # End END

} # End Function New-FirewallLog

## Set Attack Surface Reduction Rule

<#

.SYNOPSIS

Set an action for all Attack Surface Reduction (ASR) rules.

.NOTES

ASR rules are dangerous to enable because of false positives, see:

https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/overview-attack-surface-reduction

In the Windows Defender operations log, look for these event ID numbers:

5007 Settings are changed

1121 Rule blocked something

1122 Rule triggered in audit mode, nothing blocked

#>

# The ASR rules in Windows 10 v1909:

$RulesASR = @{

'7674ba52-37eb-4a4f-a9a1-f0f9a1619a2c' = 'Block Adobe Reader from creating child processes'

'D4F940AB-401B-4EFC-AADC-AD5F3C50688A' = 'Block all Office applications from creating child processes'

'9e6c4e1f-7d60-472f-ba1a-a39ef669e4b2' = 'Block credential stealing from the Windows local security authority subsystem (lsass.exe)'

'BE9BA2D9-53EA-4CDC-84E5-9B1EEEE46550' = 'Block executable content from email client and webmail'

'01443614-cd74-433a-b99e-2ecdc07bfc25' = 'Block executable files from running unless they meet a prevalence, age, or trusted list criteria'

'5BEB7EFE-FD9A-4556-801D-275E5FFC04CC' = 'Block execution of potentially obfuscated scripts'

'D3E037E1-3EB8-44C8-A917-57927947596D' = 'Block JavaScript or VBScript from launching downloaded executable content'

'3B576869-A4EC-4529-8536-B80A7769E899' = 'Block Office applications from creating executable content'

'75668C1F-73B5-4CF0-BB93-3ECF5CB7CC84' = 'Block Office applications from injecting code into other processes'

'26190899-1602-49e8-8b27-eb1d0a1ce869' = 'Block Office communication applications from creating child processes'

'e6db77e5-3df2-4cf1-b95a-636979351e5b' = 'Block persistence through WMI event subscription'

'd1e49aac-8f56-4280-b9ba-993a6d77406c' = 'Block process creations originating from PSExec and WMI commands'

'b2b3f03d-6a65-4f7b-a9c7-1c7ef74a9ba4' = 'Block untrusted and unsigned processes that run from USB'

'92E97FA1-2EDF-4476-BDD6-9DD0B4DDDC7B' = 'Block Win32 API calls from Office macro'

'c1db55ab-c21a-4637-bb3f-a12568109d35' = 'Use advanced protection against ransomware'

}

###################################################

#

# WARNING! DO NOT SET $ACTION TO 'Enabled'

# UNLESS YOU HAVE CREATED A SNAPSHOT OR

# CHECKPOINT OF YOUR VIRTUAL MACHINE FIRST!

#

###################################################

# Choose your desired setting for all of the above rules.

# Options: AuditMode, Enabled, Disabled

$Action = 'AuditMode'

# Need to create an array of your chosen $Action that is exactly

# the same size as $RulesASR:

[String[]] $ActionArray = @()

1..$($RulesASR.Count) | ForEach { $ActionArray += $Action }

# Sanity check:

if ($RulesASR.Count -ne $ActionArray.Count)

{ throw "These two arrays must be the same size" }

# With one command, all existing ASR rules, if any, are overwritten with an

# array of the rules you wish to have ($RulesASR.Keys) matched with a same-sized

# array with the desired $Action for each of these rules:

Set-MpPreference -AttackSurfaceReductionRules\_Ids $RulesASR.Keys -AttackSurfaceReductionRules\_Actions $ActionArray

# Set-MpPreference overwrites the existing rules. To only add a rule,

# use Add-MpPreference instead.

## Enable Audit Policy for CA

# Enable logging audit policy for Certification Services:

auditpol.exe /set /subcategory:"Certification Services" /success:enable /failure:enable

# Enable auditing on the properties of the CA (Auditing tab):

certutil.exe -setreg CA\AuditFilter 127

## Enable Auto Enrolment Logging

####################################################################

#.SYNOPSIS

# Enable enhanced certificate auto-enrollment event logging.

#

#.DESCRIPTION

# Enable enhanced certificate auto-enrollment event logging. Writes

# to the Application event log, Event Source = AutoEnrollment, with

# various event ID numbers: 2, 3, 20, 27, 28, 29 for successful

# events; and 7, 13, 15 and 16 for errors.

#

#.NOTES

# Errors and problems are logged by default even without

# this enhanced logging being enabled.

####################################################################

# User Cert Auto-Enrollment

reg.exe add 'HKCU\Software\Microsoft\Cryptography\Autoenrollment' /v AEEventLogLevel /t REG\_DWORD /d 0

# Machine Cert Auto-Enrollment

reg.exe add 'HKLM\Software\Microsoft\Cryptography\Autoenrollment' /v AEEventLogLevel /t REG\_DWORD /d 0

# Delete the above values (not keys) to disable this logging.

## Get Users and Cert Count

########################################################################################

#.Synopsis

# Outputs paths to users in AD with more than one certificate.

#

#.Description

# Script outputs the paths to user accounts in Active Directory which have more

# than one certificate, plus the count of total certificates in AD for each user.

# This is useful when cleaning up a PKI deployment where duplicate certificates

# were not suppressed properly through template options and credentials roaming.

# Leave $SearchRoot empty to search entire domain, or enter the

# DN path to an OU, e.g., "LDAP://OU=There,DC=Company,DC=Net".

# Does not return users with zero certificates because of SizeLimit

# issues which limit the results returned to 1000 by default:

# http://msdn.microsoft.com/en-us/library/system.directoryservices.directorysearcher.sizelimit.aspx

#

#.Notes

# Author: Jason Fossen, Enclave Consulting (http://www.sans.org/windows-security/)

# Version: 1.1

# Updated: 24.Nov.2012

# Legal: 0BSD.

########################################################################################

param ($SearchRoot = "")

function DumpUsersAndCertCount ( $SearchRoot = "" )

{

$DirectoryEntry = new-object System.DirectoryServices.DirectoryEntry -arg $SearchRoot

$DirectorySearcher = new-object System.DirectoryServices.DirectorySearcher -arg $DirectoryEntry

$DirectorySearcher.PropertiesToLoad.Add("userCertificate") | out-null

$DirectorySearcher.Filter = "(&(objectClass=user)(objectCategory=person)(userCertificate=\*))"

$SearchResultCollection = $DirectorySearcher.FindAll()

$SearchResultCollection | ForEach{$\_.Properties} | ForEach {

$obj = new-Object system.Management.Automation.PSObject

$path = $\_.adspath[0]

$certs = $\_.usercertificate

$certcount = $certs.count

add-member -inputobject $obj -membertype NoteProperty -name NumberOfCertificates -value $certcount

add-member -inputobject $obj -membertype NoteProperty -name UserPath -value $path

$obj

}

$SearchResultCollection.Dispose()

}

DumpUsersAndCertCount -SearchRoot $SearchRoot

## SMB Encryption

#################################################################

# Example PowerShell commands for managing SMB encryption.

# Server 2012, Windows 8 and later support SMB encryption.

#################################################################

# To require SMB encryption for the entire server (all shares):

Set-SmbServerConfiguration -EncryptData $True -Force

Set-SmbServerConfiguration -RejectUnencryptedAccess $True -Force

# To only enable SMB encryption, but not require it:

Set-SmbServerConfiguration -EncryptData $True -Force

Set-SmbServerConfiguration -RejectUnencryptedAccess $False -Force

# To see the current SMB settings for the entire server:

Get-SmbServerConfiguration | Select EncryptData,RejectUnencryptedAccess

# To require SMB encryption for one shared folder only:

Set-SmbShare -Name <ShareName> -EncryptData $True -Force

# To require SMB encryption for all shared folders individually:

Get-SmbShare | Set-SmbShare -EncryptData $True -Force

# To list which shared folders do or do not require SMB encryption:

Get-SmbShare | Select EncryptData,Name,Path

# To see what version of SMB is being used (run on server, not client):

Get-SmbSession | Select ClientComputerName,ClientUserName,Dialect

# To disable inbound SMBv1 support:

Set-SmbServerConfiguration -EnableSMB1Protocol $False -Force

# Notes:

# You cannot mark the ADMIN$, IPC$ or C$ shares as encrypted,

# but, if the server as a whole has EncryptData = $True, then

# SMB traffic to these shares will be encrypted anyway. Other

# hidden shares (with a $ at the end of the share name) may

# be encrypted.

#

# You can mark the SYSVOL and NETLOGON shares as encrypted

# on domain controllers, just make sure all domain-joined

# clients are Windows 8, Server 2012, or later.

#

# If a particular shared folder has EncryptData set to $False,

# access to that share will still be encrypted if encryption is

# enabled for the server as a whole. When in doubt, enable

# encryption for the server as a whole and also mark every

# individual shared folder as encrypted.

#

# If RejectUnencryptedAccess is set to $True on the server,

# which is the default, then SMBv1 clients will be rejected

# even if EnableSMB1Protocol is set to $True. So, this means

# that if RejectUnencryptedAccess is set to $False on the server,

# and EnableSMB1Protocol is set to $True, then SMBv1 clients

# will be allowed access. The most secure configuration is to

# disable SMBv1 and also reject all unencrypted access.

## Get Servers Features

#########################################################

#.SYNOPSIS

# Get roles and features for every server in the domain.

#

#.DESCRIPTION

# This is a starter script to query AD for servers and then

# inventory the roles and features on each server. Save

# the inventory with Export-CliXml. Add error handling

# and other polish to use in production. Script only queries

# Windows Server, not client OS machines.

#

#.PARAMETER SearchBase

# Distinguished name of Active Directory container where

# search for computer accounts for servers should begin.

# Defaults to the entire domain of which the local

# computer is a member.

#########################################################

Param ( [String] $SearchBase )

# Import the Active Directory module if necessary.

# Install the Remote Server Administration Tools (RSAT)

# to make the module available on non-server OS's.

Import-Module -Name ActiveDirectory

# Search entire local domain by default.

if ($SearchBase.Length -eq 0) { $SearchBase = (Get-AdDomain -Current LocalComputer).DistinguishedName }

# Find computer accounts with the word "Server" in their operating system property.

$Servers = Get-AdComputer -LdapFilter '(OperatingSystem=\*Server\*)' -SearchBase $SearchBase

# Function returns a hashtable of feature names and their install states.

Function Get-FeaturesHashTable ( $ComputerName )

{

$Table = @{}

Get-WindowsFeature -ComputerName $ComputerName | ForEach { $Table.Add($\_.Name,$\_.Installed) }

$Table

}

# Connect to each server discovered in AD and query its roles and features.

ForEach ($Server in $Servers)

{

$Output = ($Output = "") | Select-Object ComputerName,DnsHostName,Date,Features

$Output.ComputerName = $Server.Name

$Output.DnsHostName = $Server.DnsHostName

$Output.Date = Get-Date

$FeaturesTable = Get-FeaturesHashTable -ComputerName $Server.DnsHostName

$Output.Features = $FeaturesTable

$Output

$Output = $null

}

## Update Hosts file

####################################################################################

#.Synopsis

# Adds domain names to the HOSTS file for blocking. Find the HOSTS

# file at $env:systemroot\system32\drivers\etc\hosts

#

#.Description

# The HOSTS text file is typically used for name resolution before any

# DNS queries are performed. This script can import the names from one

# or multiple text files into the local HOSTS file for blocking correct

# resolution of those names. The input file(s) can be on the local drive,

# in shared folders, or on HTTP servers. By default, names will resolve

# to "0.0.0.0", but a different IP address can be specified. You must be

# a member of the local Administrators group to use the script (or at

# least be granted NTFS write access to the HOSTS file itself).

#

#.Parameter FilePathOrURL

# Path to a file which contains the FQDNs and domain names to sinkhole.

# File can have blank lines, comment lines (# or ;), multiple FQDNs or

# domains per line (space- or comma-delimited), and can be a HOSTS file

# with IP addresses too (addresses and localhost entires will be ignored).

# You can also include wildcards to input multiple files (e.g., "bad\*.txt"),

# use a UNC path to an SMB shared folder (e.g., "\\server\share\file.txt")

# or an HTTP path; in fact, you can mix all these path together into one

# long space-delimited string for this parameter. The parameter will

# default to "http://www.malwaredomainlist.com/hostslist/hosts.txt" if

# left unspecified.

#

#.Parameter SinkholeIP

# The IP address to which sinkholed names will resolve. Default is 0.0.0.0.

#

#.Parameter BlockWPAD

# Web Proxy Automatic Discovery (WPAD) protocol resolves the "wpad" name

# to an HTTP server with a browser configuration file. WPAD is vulnerable

# to DNS spoofing attacks. Adding this switch will cause the hosts file

# to resolve "wpad" and "wpad.your.domain" to 0.0.0.0, which will block

# the DNS method of using WPAD (but not the DHCP method). This will also

# break any browser's Internet access that requires WPAD to function, so

# do not use this switch unless you know exactly how WPAD works.

#

#.Parameter AddDuplicateWWW

# When adding names to the HOSTS file, if any name does not begin with

# "www.", then this switch will add that name twice to the HOSTS file:

# the original unaltered name and a second copy with "www." prepended.

# Many browsers will automatically prepend "www." to any name which

# results in an error or cannot be resolved correctly.

#

#.Parameter ResetToDefaultHostsFile

# Will erase the HOSTS file and add only two entries back to it:

# 127.0.0.1 localhost

# ::1 localhost

#

#.Parameter EditHostsFile

# Opens the HOSTS file in Notepad.exe.

#

#.Parameter ShowHostNameCount

# Displays a count of the number of names in the HOSTS file, not

# including the localhost entries.

#

#.Parameter ShowHostsFilePath

# As specified in the registry, displays the path to the folder which

# contains the hosts file. Malware and hackers change this sometimes.

# Prints a warning if it is not the default path.

#

#.Example

# Update-HostsFile.ps1

#

# Adds all the names from www.MalwareDomainList.com to your HOSTS file

# and makes them all resolve to "0.0.0.0".

#

#.Example

# Update-HostsFile.ps1 -FilePathOrURL "c:\folder\file.txt \\server\ `

# share\remotefile.txt http://www.malwaredomainlist.com/hostslist `

# /hosts.txt" -SinkHoleIP "10.1.1.1"

#

# Sinkholes all the names listed in file.txt, remotefile.txt, and all the

# names from the URL shown, then makes them all resolve to "10.1.1.1".

# Notice that each path is separated by a space character.

#

#.Example

# Update-HostsFile.ps1 -ResetToDefaultHostsFile

#

# Erase the HOSTS file and then add back only the following:

# 127.0.0.1 localhost

# ::1 localhost

#

#.Notes

# Author: Jason Fossen, Enclave Consulting LLC (http://www.sans.org/sec505)

# Version: 1.5.2

# Updated: 17.Jul.2017

# Legal: 0BSD.

#Requires -Version 2.0

####################################################################################

Param ($FilePathOrURL = "http://mirror1.malwaredomains.com/files/justdomains",

[String] $SinkholeIP = "0.0.0.0",

[Switch] $ResetToDefaultHostsFile,

[Switch] $AddDuplicateWWW,

[Switch] $EditHostsFile,

[Switch] $BlockWPAD,

[Switch] $ShowHostNameCount,

[Switch] $ShowHostsFilePath)

filter extract-text ($RegularExpression)

{

select-string -inputobject $\_ -pattern $regularexpression -allmatches |

select-object -expandproperty matches |

foreach {

if ($\_.groups.count -le 1) { if ($\_.value){ $\_.value } }

else

{

$submatches = select-object -input $\_ -expandproperty groups

$submatches[1..($submatches.count - 1)] | foreach { if ($\_.value){ $\_.value } }

}

}

}

function update-hostsfile

{

Param ($FilePathOrURL = "http://mirror1.malwaredomains.com/files/justdomains",

[String] $SinkholeIP = "0.0.0.0",

[Switch] $ResetToDefaultHostsFile,

[Switch] $AddDuplicateWWW,

[Switch] $EditHostsFile,

[Switch] $BlockWPAD,

[Switch] $ShowHostNameCount,

[Switch] $ShowHostsFilePath)

$HostsFilePath = "$env:systemroot\system32\drivers\etc\hosts"

$webclient = new-object System.Net.WebClient

$names = @() #Array of names to add to HOSTS file.

# Check for common help switches and show help text.

if (($FilePathOrURL -ne $null) -and ($FilePathOrURL.GetType().Name -eq "String") -and ($FilePathOrURL -match "/\?|/help|-help|--h|--help"))

{

If ($Host.Version.Major -ge 2) { get-help -full .\update-hostsfile.ps1 }

Else {"`nPlease read this script's header in Notepad for the help information."}

return

}

# Confirm PowerShell 2.0 or later.

If ($Host.Version.Major -lt 2) { "This script requires PowerShell 2.0 or later.`nDownload the latest version from http://www.microsoft.com/powershell`n" ; return }

#Edit hosts file in notepad?

if ($EditHostsFile) { notepad.exe $HostsFilePath ; return }

#Show path to hosts file? Sometimes modified by malware.

if ($ShowHostsFilePath)

{

$folder = (Get-ItemProperty -Path HKLM:\SYSTEM\CurrentControlSet\services\Tcpip\Parameters -Name DataBasePath).DataBasePath

"`nAs defined in the registry, the hosts file folder is:`n`n `t $folder `n"

if ($folder -eq "$env:SystemRoot\System32\drivers\etc")

{ "This is the default folder path.`n" }

else

{ "CAUTION! THIS IS NOT THE DEFAULT LOCATION! POSSIBLE MALWARE INDICATOR! `n" }

return

}

#Show count of names in hosts file? Does not include localhost entries, but will include non-sinkhole names.

if ($ShowHostNameCount)

{ "`nCount of names in hosts file = " + ($(get-content $HostsFilePath) -split " " |

where { $\_ -notmatch '\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}|\:|localhost|^\s\*$|^\#' }).count ; "`n" ; return }

#Check if we're just resetting the hosts file to the default localhosts.

if ($ResetToDefaultHostsFile)

{

#Sometimes a full CRLF newline (0x0D,0x0A) is not appended

#unless we do these lines separately (weird).

"127.0.0.1 localhost" | set-content $HostsFilePath -force

"::1 localhost" | add-content $HostsFilePath -force

if (-not $?) { "Error writing to hosts file!" }

return

}

#Get one or more space-delimited input files, split up into the $names array.

#If the path contains spaces, use the 8.3 DOS short name, e.g., PROGRA~1.

$FilePathOrURL = @(( $FilePathOrURL -split '\s+' ) | foreach { $\_.trim() } | where { $\_.length -gt 1 } )

if ($FilePathOrURL.Count -eq 0) { "No valid paths to input files, quitting." ; return }

$FilePathOrURL | foreach { `

if ($\_ -like "http\*")

{

$string = $webclient.DownloadString("$\_")

# If an error, wait two seconds, then try a second time.

if (-not $?) { start-sleep -seconds 2 ; $string = $webclient.DownloadString("$\_") }

#If an error again or anything html-ish looking is returned, give up.

if ($string.contains("</") -or -not $?)

{

"`nDownload failure: " + $\_ + "`n"

"Hosts file will not be modified."

}

else

{

"`nDownloaded: " + $\_ + "`n"

$names += $string -split '[\r\n]+'

}

}

else

{

if (test-path $\_)

{

$names += get-content -path $\_

if ($?) { "`nImported: " + $\_ + "`n" }

else { "`nImport failure: " + $\_ + "`n" }

}

else

{ "`nInvalid path: " + $\_ + "`n" }

}

}

#Confirm that at least one line was imported.

if ($names.count -lt 1) { "`nNothing to add to the hosts file, quitting.`n" ; return }

#Remove anything after a comment char (# or ;) on each line, including the comment char.

$names = $names | foreach { if ($\_.IndexOfAny("#;") -ne -1) { $\_.Remove($\_.IndexOfAny("#;")) } else { $\_ } }

#Split the IPs from the names; maybe this file has no IPs in it.

$names = $names -split '[\,\s]+'

#Extract FQDNs from any http URLs.

$names = $names | foreach { if ($\_ -match '^http\:\/\/.+') { $\_ | extract-text 'http[s]\*\:\/\/([a-z0-9\.\-]+\.[a-z0-9\.\-]+)'} else { $\_ } }

#Filter out blanks, IPv4/IPv6 addresses, localhost, anything without at least one period, anything with two periods in a row, then remove duplicates

$names = $names | foreach {$\_.Trim()} | where { $\_.Length -gt 1 } | where { $\_ -notmatch '\.\.|\:|^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$|^localhost$' } | where { $\_ -match '^.+\..+' } | sort-object -unique

#If requested, add "www.\*" entries for names which don't begin with "www" already.

if ($AddDuplicateWWW){ $names = $names += ($names | where {$\_ -notmatch '^www\.'} | foreach {"www." + $\_ }) }

#In the hosts file, each line should have a max of nine hostnames, ending with a CRLF.

#Lookup performance is maximized by having nine hostnames per line and by

#resolving to 0.0.0.0 instead of 127.0.0.1, especially if you're listening on TCP/80.

$size = $names.count - 1

for ($i = 0 ; $i -lt $size ; $i = $i + 9)

{ $output += "$SinkholeIP " + ($names[$i..$($i + 8)] -join " ") + [char]13 + [char]10 }

#Sometimes a full CRLF newline (0x0D,0x0A) is not appended

#unless we do these two lines separately (weird). The all-zeros

#line is added for the sake of tools doing reverse lookups.

"127.0.0.1 localhost" | set-content $HostsFilePath -force

"::1 localhost" | add-content $HostsFilePath -force

"0.0.0.0 zero.zero.zero.zero" | add-content $HostsFilePath -force

if (-not $?) { "Error writing to hosts file!`n" ; return }

#WPAD

$wpadline = "0.0.0.0 wpad" #Should this be $SinkholeIP instead?

if (Get-Command -Name Get-DnsClientGlobalSetting -ErrorAction SilentlyContinue)

{

Get-DnsClientGlobalSetting | Select -ExpandProperty SuffixSearchList | ForEach { $wpadline = $wpadline + (' wpad.' + $\_) }

}

elseif ($env:userdnsdomain -ne $null) #Will be null when run as System

{

$wpadline = $wpadline + (' wpad.' + ($env:userdnsdomain).tolower() )

}

if ($blockwpad) { $wpadline | add-content $HostsFilePath -force }

#Write the rest:

$output | add-content $HostsFilePath -force

}

# Main (need to splat all this...)

if ($ResetToDefaultHostsFile) { update-hostsfile -ResetToDefaultHostsFile }

elseif ($EditHostsFile) { update-hostsfile -EditHostsFile }

elseif ($ShowHostNameCount) { update-hostsfile -ShowHostnameCount }

elseif ($ShowHostsFilePath) { update-hostsfile -ShowHostsFilePath }

else

{

if ($BlockWPAD -and $AddDuplicateWWW) { update-hostsfile -FilePathOrURL $FilePathOrURL -SinkholeIP $SinkholeIP -BlockWPAD -AddDuplicateWWW }

elseif ($AddDuplicateWWW) { update-hostsfile -FilePathOrURL $FilePathOrURL -SinkholeIP $SinkholeIP -AddDuplicateWWW }

elseif ($BlockWPAD) { update-hostsfile -FilePathOrURL $FilePathOrURL -SinkholeIP $SinkholeIP -BlockWPAD }

else {update-hostsfile -FilePathOrURL $FilePathOrURL -SinkholeIP $SinkholeIP }

}

## Group Managed Service Accounts

####################################################################################

# Group Managed Service Accounts can be used on Server 2012, Windows 8, and later.

# GMSAs can be used across multiple machines and for scheduled jobs.

# Requires at least one controller to be running Server 2012 or later.

# Requires the schema to be upgraded to Server 2012 or later.

# Scheduled tasks which use GMSAs must be created and edited in PowerShell.

# When used with SQL Server, it must be SQL Server 2012 or later.

# For more information see:

# http://blogs.technet.com/b/askpfeplat/archive/2012/12/17/windows-server-2012-group-managed-service-accounts.aspx

####################################################################################

# Step 1: Create the Key Distribution Service (KDS) root key:

Add-KdsRootKey -EffectiveTime ((Get-Date).AddHours(-10))

# Note that in real life you would execute the following command

# to create the KDS root key and then wait 10 hours:

# Add-KdsRootKey -EffectiveImmediately

# This only needs to be done once per domain.

# Step 2: Choose or create a computer group whose member computers

# will be allowed to use the GMSA, e.g., "Domain Controllers".

# Step 3: Create the GMSA and specify the computer group:

New-ADServiceAccount -name TestingGMSA `

-DNSHostName testinggmsa.testing.local `

-PrincipalsAllowedToRetrieveManagedPassword "Domain Controllers"

# Step 4: Install and text GMSA on a computer in the group:

Install-AdServiceAccount TestingGMSA

Test-AdServiceAccount TestingGMSA

# Step 5a: Use the GMSA for a service account. Make sure to

# specify the username as "testing\TestingGMSA$", i.e., the name

# of the domain, backslash, then the name of the of the GMSA account

# with a dollar sign ($) at the end. Leave the password blank.

# Step 5b: Or use the GMSA for a scheduled task. This must be done in

# PowerShell since Microsoft couldn't be bothered with updating the

# Task Scheduler graphical tool to support creating or editing tasks

# which use GMSA accounts (very nice). Don't forget to grant the

# GMSA account the "Log on as a batch job" right also.

$Principal = New-ScheduledTaskPrincipal -UserID testing\TestingGMSA$ -LogonType Password

$Action = New-ScheduledTaskAction "C:\Folder\Script.ps1"

$Trigger = New-ScheduledTaskTrigger -At 2:00 -Daily

Register-ScheduledTask GmsaTaskName -Action $Action -Trigger $Trigger -Principal $Principal

# Note that '-LogonType Password' should be entered as shown literally, do not replace the

# 'Password' with another string which might be the password in the first command above.

# That's it! But be aware that if the GMSA account is added to Domain Admins or another

# high-powered group, anyone who is a member of the local Administrators group on an

# authorized computer can create a scheduled task using the GMSA account. Also, we do not

# yet have the details about how the GMSA account's password or hash is stored in AD or

# transmitted over the network to the computers permitted to use the GMSA.

## Get Service Privileges

# List explicitly-granted privileges for services.

Param ($ServiceName = '\*')

function Get-ServicePrivileges ($ServiceName = '\*')

{

Get-Service -Name $ServiceName | ForEach `

{

$output = '' | Select Name,DisplayName,Privileges

$output.Name = $\_.Name

$output.DisplayName = $\_.DisplayName

$privs = sc.exe qprivs $\_.Name | select-string -NotMatch '\_NAME|SUCCESS' | Out-String

$privs = $privs.Replace('PRIVILEGES','')

$privs = $privs.Replace(':',' ')

$privs = $privs -Replace '\s+',' '

$output.Privileges = $privs.Trim()

$output

}

}

Get-ServicePrivileges -ServiceName $ServiceName

## Hashing Script

# save the old Information preference

$OldInfoPref = $InformationPreference

# enable Information display

$InformationPreference = 'Continue'

$TargetDir = Read-Host 'Please enter the full path to the target Directory '

$FileList = Get-ChildItem -LiteralPath $TargetDir -Recurse -File

$Counter = 0

$Results = foreach ($FL\_Item in $FileList)

{

$Counter ++

Write-Information ('Processing file {0} of {1} ...' -f $Counter, $FileList.Count)

# this will \_silently\_ skip files that are locked for whatever reason

$FileHash = Get-FileHash -LiteralPath $FL\_Item.FullName -Algorithm MD5 -ErrorAction SilentlyContinue

# if this is empty, then the "else" block will show "\_\_Error\_\_" in the Hash column

if ($FileHash)

{

[PSCustomObject]@{

Hash = $FileHash.Hash

Path = $FileHash.Path

}

}

else

{

[PSCustomObject]@{

Hash = '\_\_Error\_\_'

Path = $FL\_Item.FullName

}

} # end >> if ($FileHash)

} # end >> foreach ($FL\_Item in $FileList)

# on screen display

$Results

# send to CSV file

$Results |

Export-Csv -LiteralPath "$env:TEMP\FileHashListing.csv" -NoTypeInformation

# restore Information preference

$InformationPreference = $OldInfoPref

## Other documents