CYB631 Automating Information Security with Python and Shell Scripting

Lab 1: PowerShell Basics and Gathering Host Information

Your Name: **VAIBHAV AVINASH MAYEKAR**

Goals: This lab will guide you through basic PowerShell commands and use them to gather host information from Windows 10 OS.

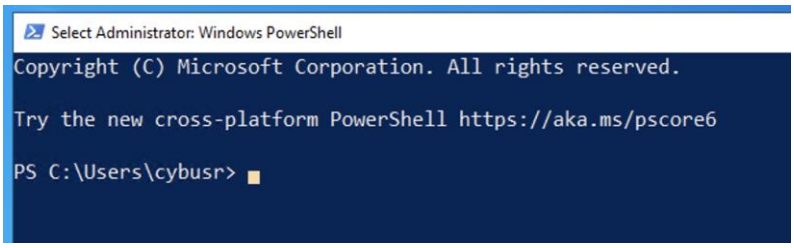
Readings and References: The contents of this lab are adapted from Part I, Chapter 18.1, and Chapter 24 of the Windows PowerShell Cookbook by Lee Holmes.

Exercises:

[Exercise I: Starting with PowerShell and Simple PS Commands]

1. Launch Windows PowerShell. Click Start->Run, and then type PowerShell.
2. Alternatively, you can also use Windows PowerShell ISE. Click Start->Run, and then type PowerShell ISE and run as an administrator.

   Note: you might need to open PowerShell and run as an administrator first and then open PowerShell ISE by typing "powershell ise" in the command line.
3. Windows 10 environment is needed for the lab. A PowerShell prompt window opens with a command prompt like the one below. All the commands in this lab should be typed and run after PowerShell command prompt.



4. Move to your working directory.

   cd \\pace.edu\shares\users\your_user_id\Desktop

5. Let us simplify the command prompt.

   function Prompt { "PS > " }

6. Exam login user.

   whoami

7. Review files under your current directory.

   dir

8. Examine your current directory. What is your current directory?
   C:\Users\mayek\OneDrive\Desktop\python shell scripting

   pwd

9. We can also use a PowerShell cmdlet to obtain the same information.

   get-location

10. We will try **pushd** and **popd** commands. The command **pushd** stores the current directory to a stack for use later by **popd**, and then changes to the specified directory. The command **popd** changes the current directory to the most recent directory stored by **pushd**.

    pushd .

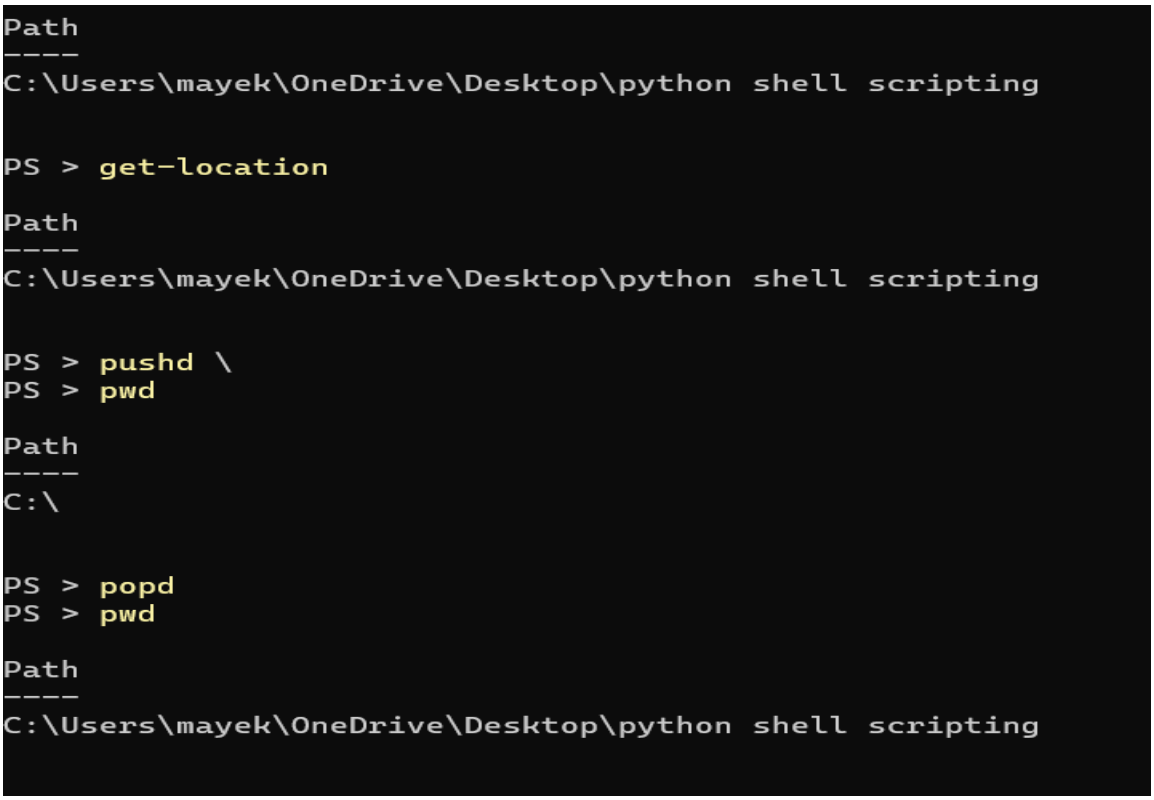
    pwd

    cd ..

    popd

    pwd

11. Provide a screenshot of the results from the last step.



12. Let us create a folder to store all of the files for this class. First of all, make sure that you are in your home directory (such as C:\Users\cybusr). Create the folder and go into the folder. All of the files you created will be stored here.

    mkdir cyb631

    dir

    cd cyb631

13. The folder should be empty now. Let us create a new file in the folder using PowerShell cmdlet **new-item**.

    new-item -path . -name "test1.txt" -itemtype "file" -value "Hello World."

14. Now, you should be able to see the new file test1.txt under cyb631 folder. What is the content of this file? **Hello World**

    dir

    cat test1.txt

15. We will store the file name in a variable and then later use it in the command.

$filename=”C:\Users\cybusr\cyb631\test1.txt”

16. To see the value of the variable to make sure that it has the value you want.

    get-variable filename

17. Let change test1.txt to a different file attribute. The current attribute is a (archive). Let us change it to r (read-only).

    dir

    attrib +R $filename

    dir

18. You should see that the mode is listed differently. Paste a screenshot of your results after the change of attribute.

```
PS > attrib +R $filename
Path not found - C:\Users\mayek\OneDrive\Desktop\python shell scriptin
g\cyb631\ C:\Users\mayek\OneDrive\Desktop\python shell
    scripting
PS > dir


    Directory: C:\Users\mayek\OneDrive\Desktop\python shell
    scripting\cyb631


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-ar---         9/13/2023   1:07 PM             12 test1.txt
```

19. Try changing the content of the file test1.txt. Are you able to do so? **NO**

20. Now, change the attribute back to where it was. Are you able to change the content of the file now? **YES**

    attrib -R $filename

[Exercise II: Simple Object Operation]

21. Let us create a simple command below. The result is an object with a string.
    "Hello" + " World"
22. We can assign the object to a variable and then find out what properties and members are associated with it.
    $s1 = "Hello" + " World"
    get-variable s1
    $s1| get-member

23. From the results above, you can tell that this class has only one property- **length.** Let see the value of it. What is the result? **11**
    $s1.Length
24. We can also try one of the methods associated with this class. **ToLower** method lowers every character to the lower case.
    $s1.ToLower()

[Exercise III: Gather Process Information]

We will investigate processes running on the system and gather information about them.

25. Now, let us find out what processes are running on the machine.
    get-process
26. It is a very long list. Let us find only the processes that starts with "w" and uses its alias gps.
    gps -n w*
27. Now, **open a notepad** and find the information about the process for notepad.
    gps -n notepad*
28. Let us store the information in a variable **prs**.
    $prs= gps -n notepad*
29. Stop the notepad process by killing it. You will find an "Access Denied" error message. This is because you did not have enough privilege.
    $prs.kill()
30. Now, open another PowerShell but run ad the Administrator. Repeat step 29 to 31. You should be able to kill the process. If successful, what happened to the notepad window that you opened earlier? **CLOSED**
31. Connect several cmdlet together to analyze the results from **get-process**. This will show you a table with processes that have their handles >=500.
    get-process |
    where-object { $_.handles -ge 500 } |
    sort-object handles |
    format-table handles, name, description -auto

32. Paste the screenshot of your results above.

```
PS > get-process |where-object { $_.handles -ge 500 } |sort-object handles |format-table handles, name, description -auto

Handles Name                                Description
------- ----                                -----------
    503 svchost
    505 uihost                              McAfee WebAdv...
    518 AMDRSServ                           Radeon Settin...
    519 nvcontainer
    525 LenovoVantage-(LenovoServiceBridgeAddin)
    532 svchost
    533 ModuleCoreService
    534 RuntimeBroker                       Runtime Broker
    541 dllhost                             COM Surrogate
    543 svchost
    544 chrome                              Google Chrome
    547 svchost
    548 svchost
    552 chrome                              Google Chrome
    559 chrome                              Google Chrome
    564 svchost
    565 Microsoft.SharePoint                Microsoft Sha...
    574 ctfmon
    590 svchost
    604 NVIDIA Web Helper                   NVIDIA Web He...
    605 svchost
    608 LockApp                             LockApp.exe
    619 ApplicationFrameHost                Application F...
    621 RuntimeBroker                       Runtime Broker
    636 McUICnt                             McAfee
```

33. Now modify the statement above to show processes that have identifiers (id) larger or equal to 10000. Paste the screenshot of your results.

```
PS > get-process |where-object { $_.handles -ge 10000 } |sort-object
handles |format-table handles, name, description -auto

Handles Name     Description
------- ----     -----------
  16586 MMSSHOST
```

[Exercise IV: Useful PowerShell Commands]

There are commands to discover what you can use under PowerShell. Also, please refer to this list, https://devblogs.microsoft.com/scripting/table-of-basic-powershell-commands/ , for commands to use.

34. To see all the command available under PowerShell
    get-command
35. To see all the options available for the cmdlet.
    get-help get-process

36. To see all the members associated with an object returned by a cmdlet, such as **get-process** class.
    get-process | get-member

[Exercise V: First PowerShell Script]

We will create a simple PowerShell script.

37. Our goal is to sum up the handles in the selected processes.
    get-process -n n*
38. Paste a screenshot of your results above.

```
PS > get-process -n n*

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessNam
                                                        e
-------  ------    -----      -----     ------      --  -- -----------
    829      54    22616      64708       0.03   22392   2 Nahimic3
   2938      32    42140      52352       0.14   16996   2 nahimic...
   5569      26     9788      30024              4740   0 Nahimic...
   1645      25    11664       5824       4.44   24732   2 Nahimic...
   2770      22     7372       9176       1.89   15812   2 Nahimic...
    228      15     4872      12356               9652   0 NisSrv
    739     246    14768      43436               4792   0 nvconta...
    519      47    17012      37068       4.09   13260   2 nvconta...
    365      19     8592      21896               2608   0 NVDispl...
    781      35    50944      62204              16376   2 NVDispl...
    604      78    36688      35828       0.34   22160   2 NVIDIA ...
```

39. We will first run commands separately and then put them together as a script.
    $hcount=0
40. Sum up the handles in the process being selected.
    foreach($process in get-process -n n*) {$hcount+=$process.handles}
41. Show the result.
    $hcount
42. What is your result from above? **17125**
43. Let us create our first PowerShell script using PowerShell ISE.
    powershell ise
44. You can use the editor provided by PowerShell ISE to create a new PowerShell script. Every PowerShell script has a .ps1 extension. Please enter the three commands for counting process handles (from step 42 to 44).
45. Run the script1.ps1. Most likely, you will find that an error message "… script1.ps1 cannot be loaded because running scripts is disabled on this system…."
    .\script1.ps1
46. You cannot run the script because the security policy of the system is blocking scripting. To enable it, we would like to make sure that the policy is not wide open for other malicious scripts from the Internet. To see the current policy,

Get-ExecutionPolicy

What is the current Execution Policy? **_Restricted_____**

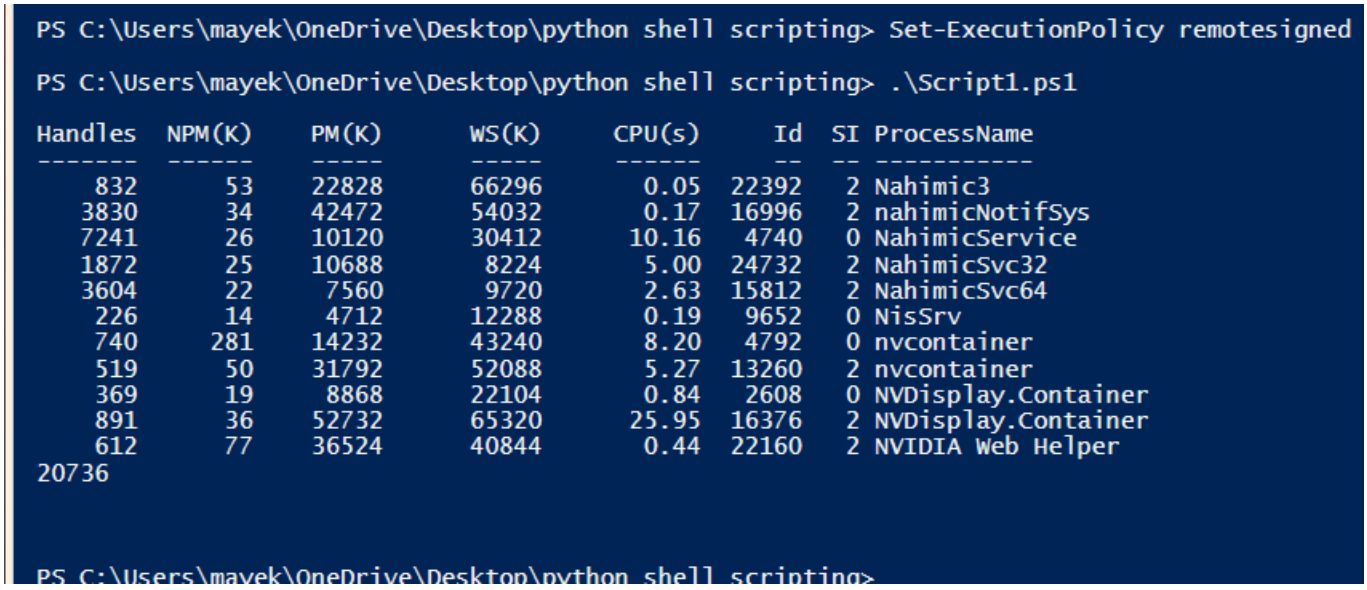
47. Now, open a new PowerShell ISE and run as an administrator. Make sure that you navigate to the directory where your script is. Change the executive policy to RemoteSigned.

    cd C:\Users\cybusr\cyb631

    Set-ExecutionPolicy remotesigned

48. Now, you can run the script. Paste a screenshot showing that you run the script and the results.

    .\script1.ps1



[Exercise VI: Develop Your Own PowerShell Script – CPU Time]

49. It is important for a system administrator/security analyst to know what processes consume the most CPU resources. Write a PowerShell script, **cputime.ps1**, to achieve the following goal.
    a. Sort processes by CPU time used.
    b. Output the process ID and process names of top five processes that consume the most CPU time.

50. Submit cputime.ps1 in additional to the lab1 report.

51. Paste a screenshot of running the program and shows its results.

```
PS C:\Users\mayek\OneDrive\Desktop\python shell scripting> .\cputime.ps1

ProcessID ProcessName             CPUTime
--------- -----------             -------
        4 System                890.265625
    16492 Discord                 759.21875
     4288 MsMpEng               644.140625
    24588 dwm                     552.875
    24664 Microsoft.SharePoint 334.640625
```