Lab: 2 – Analyzing Logs and Other Administrators' Tasks

Vijaysingh Mahendrasingh Puwar

Seidenberg School of Computer Science and Information Systems

CYB631 – Automating Information Security with Python and Shell Scripting

Professor Alex Tsekhansky

September 12, 2025

Contents

Summary

This lab introduced me to more advanced PowerShell features by building upon the basics from Lab 1. I began with object operations, arrays, and hashtables, which reinforced the idea that PowerShell treats all data as objects with properties and methods. I then explored file and directory management, script parameterization, and the process of converting a simple script into a reusable module.

In the middle portion of the lab, I analyzed event logs using both Get-EventLog and Get-WinEvent, which demonstrated the importance of log data for security monitoring and troubleshooting. I also worked with services, learning how to list, filter, and sort them by dependency count, which highlighted how service management can directly impact system stability and security.

The final task was the development of a complete administrative script, sys_admin.ps1. This script automatically generates a system report including timestamp, machine and user details, and event log summaries. When the -ShowService parameter is used, it also lists the top five running services sorted by dependent count. Creating this script helped me understand how individual PowerShell concepts can be combined into a practical automation tool for system administration and host security.

Overall, this lab was valuable because it connected foundational scripting skills with real-world administrative and security tasks. It demonstrated how PowerShell can be leveraged not only to automate routine system checks but also to support cybersecurity monitoring and incident response.

EXERCISES: [ENVIRONMENT: STARTING WITH POWERSHELL ISE]

Step 1 — Launching Windows PowerShell ISE

For this lab, we will be working within a **Windows 10 environment**, since many PowerShell features and cmdlets are designed for Windows systems. To begin, launch the **Windows PowerShell Integrated Scripting Environment (ISE)**, which provides both a command-line shell and a script editor in one interface.

- To open it, click on the **Start Menu**, then choose **Run** (or press Windows + R on your keyboard).
- In the Run dialog box, type: “PowerShell ISE” and press ENTER

This will start the PowerShell ISE application, which provides color-coded syntax highlighting, script pane, and console pane. Using the ISE instead of the regular PowerShell console makes it easier to edit, debug, and save scripts for later use.

Step 2 — Running PowerShell ISE as Administrator and Preparing the Environment

Many administrative tasks in this lab require elevated privileges. Therefore, you must open PowerShell ISE with **Run as Administrator**. This ensures that you have the necessary rights to execute commands that affect system policies, directories, and services.

Once the ISE is open with administrative privileges, navigate to the directory where your lab scripts are stored. In this case, the shared lab directory is:

C:\Users\Administrator\Desktop\CYB631Labs\lab2

Use the cd command to set this as your working directory: cd C:\Users\Administrator\Desktop\CYB631Labs\lab2

Your console prompt should confirm the path, similar to: PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>

EXERCISE I: BASIC OBJECT OPERATIONS

Step 3 — Storing the Value of a Cmdlet Output in a Variable

In PowerShell, almost everything is treated as an **object**. Objects contain both data (properties) and actions (methods). One of the most useful examples is the **Get-Date** cmdlet, which returns a DateTime object representing the current date and time.

Command sequence:

```
$today = Get-Date
$today.Date
$today.DayOfWeek
$today.DayOfYear
$today.ToLongDateString()
```

Explanation:

$today = Get-Date assigns the current date and time to a variable named $today.

$today.Date shows only the date portion (year, month, and day) without the time.

$today.DayOfWeek outputs the current day (e.g., Tuesday).

$today.DayOfYear outputs the numeric day within the year (e.g., 254 = September 10, 2025).

$today.ToLongDateString() formats the date in a more descriptive way (e.g., *Wednesday, September 10, 2025*).

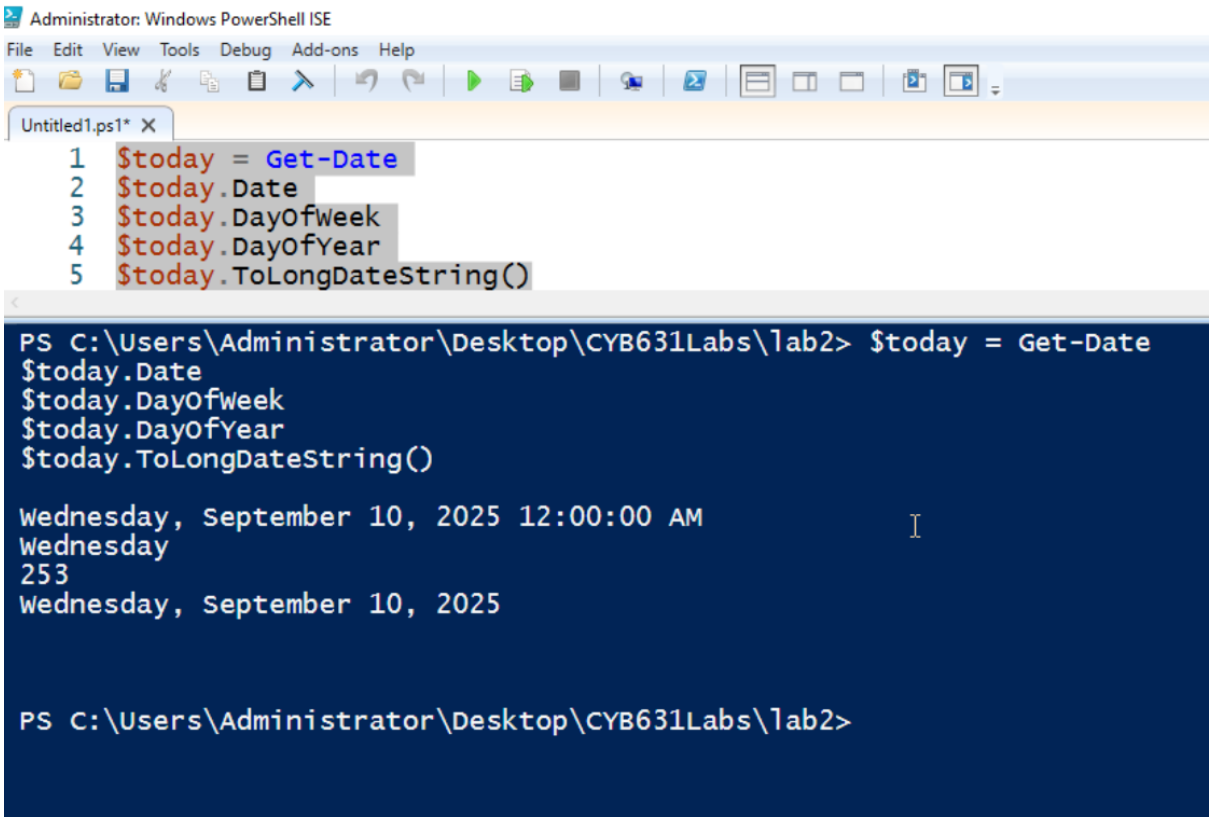

Figure 1

Step 4 — Using Static Members of a Class

PowerShell also allows you to directly access static properties and methods of .NET classes without creating an object. The [System.DateTime] class provides several useful static members.

Command sequence:

[System.DateTime]::Today

[System.DateTime]::DaysInMonth(2023, 9)

Explanation:

[System.DateTime]::Today outputs the current date (at midnight), ignoring the time of day.

[System.DateTime]::DaysInMonth(2023, 9) calculates the number of days in a given month (September 2023 = 30 days).

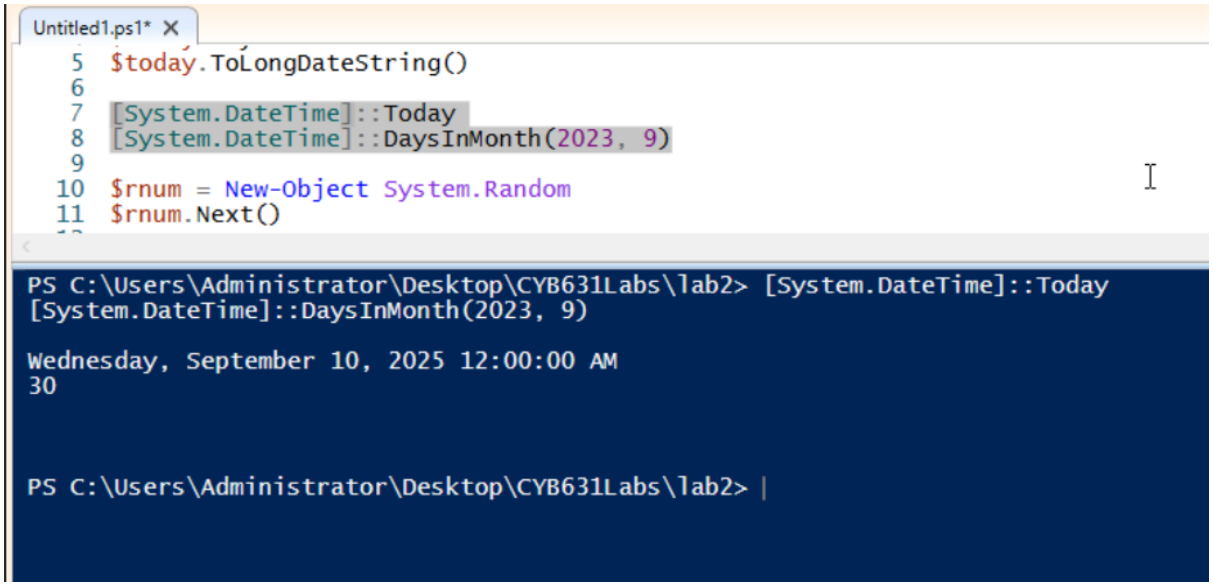

Figure 2

Step 5 — Creating an Instance of a .NET Class

Sometimes you need to create an object yourself rather than relying on cmdlets. For example, the System.Random class generates random numbers.

Command sequence:

$rnum = New-Object System.Random

$rnum.Next()

Explanation:

$rnum = New-Object System.Random creates a new instance of the Random class.

$rnum.Next() generates a random integer. Each time you run it, you'll get a different number.

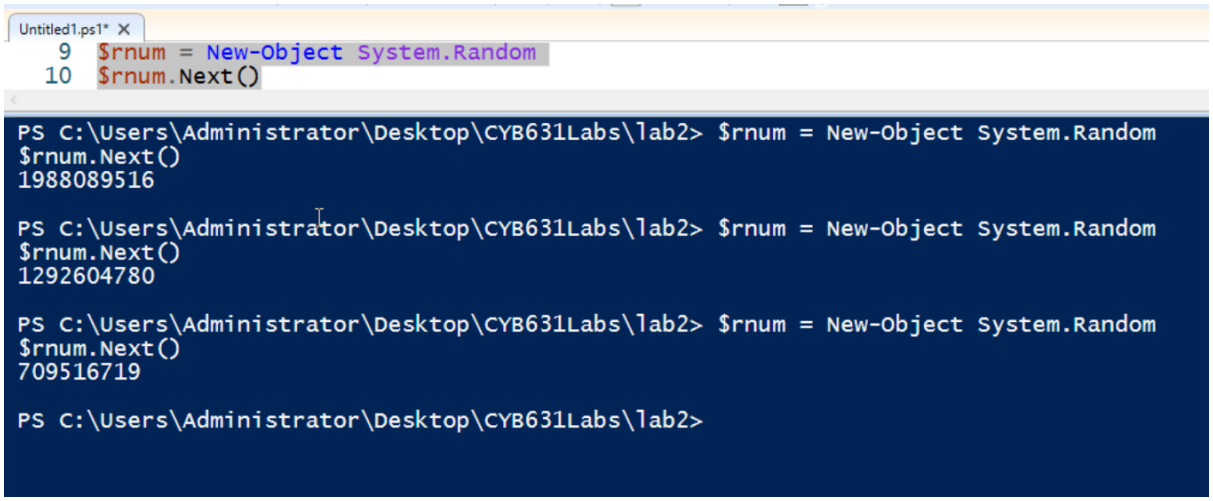

Figure 3

Step 6 — Screenshots of Results

The Screenshot results are provided into Fig 2, 3 and 4 in the above following Steps.

Exercise II: List, Array, and Hashtables

Step 7 — Creating a Simple Array with Mixed Items

In PowerShell, an array is a collection of items stored in a single variable. Arrays can hold values of different types (numbers, strings, etc.).

Command sequence:

$myarray = 1, 2, 3, "apple", "banana"

$myarray

Explanation:

$myarray is assigned a collection containing three integers and two strings.

Typing $myarray prints the contents of the array: 1 2 3 apple banana.

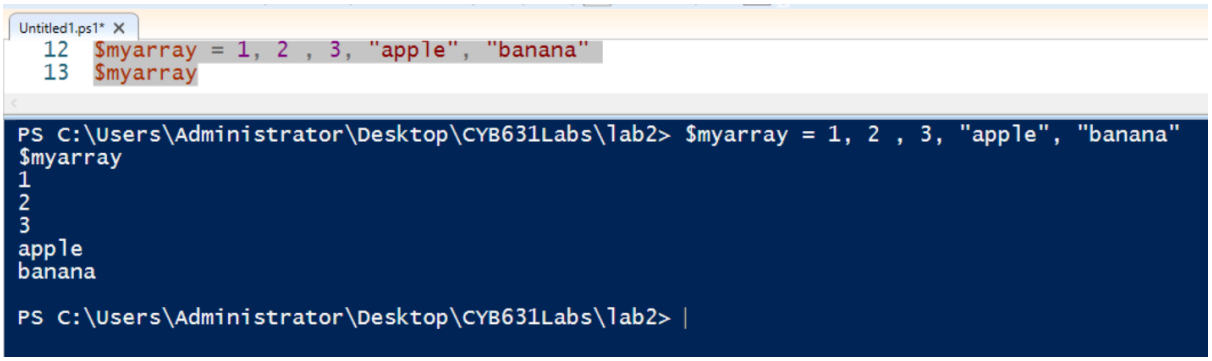

Figure 4

Step 8 — Creating an Array of Fixed Size and Assigning Values

PowerShell allows you to explicitly create an array of a certain type and size.

Command sequence:

$myarray = New-Object string[] 10

$myarray[5] = "apple"

$myarray[2] = "pear"

$myarray

Explanation:

- New-Object string[] 10 creates an array with **10 slots**, all initially empty (null).
- $myarray[5] = "apple" assigns the value "apple" to the 6th element (arrays are 0-indexed).
- $myarray[2] = "pear" assigns "pear" to the 3rd element.
- Printing $myarray displays the 10-element array, with "pear" in position 2, "apple" in position 5, and the rest blank.

Step 9 — Result of Step 8

The output of Step 8 shows:

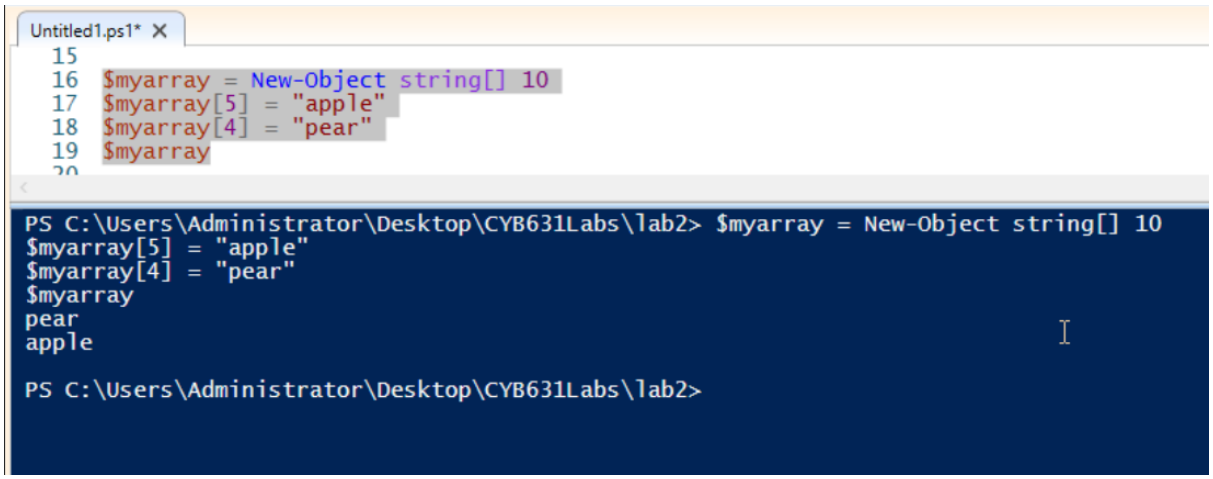

Figure 5

Step 10 — Creating a Multidimensional Array

Arrays can also hold nested arrays, effectively creating a multidimensional structure.

Command sequence:

- $arr2 = @((1,2,3,4), (5,6,7,8))
- $arr2[0][1]

Explanation:

- $arr2 is assigned two sub-arrays: (1,2,3,4) and (5,6,7,8).
- $arr2[0][1] accesses the **first sub-array** (1,2,3,4) and then the **second element** of that sub-array, which is 2.

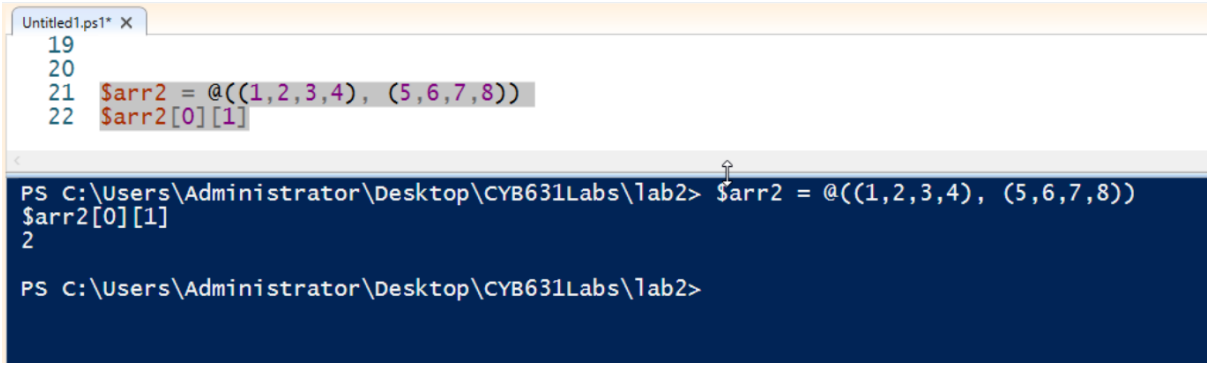

Figure 6

Step 11 — Value of $arr2[1][1]

- $arr2[1] = the second sub-array (5,6,7,8).
- $arr2[1][1] = the second element of that sub-array, which is **6**.
- **Answer:** The value is 6.

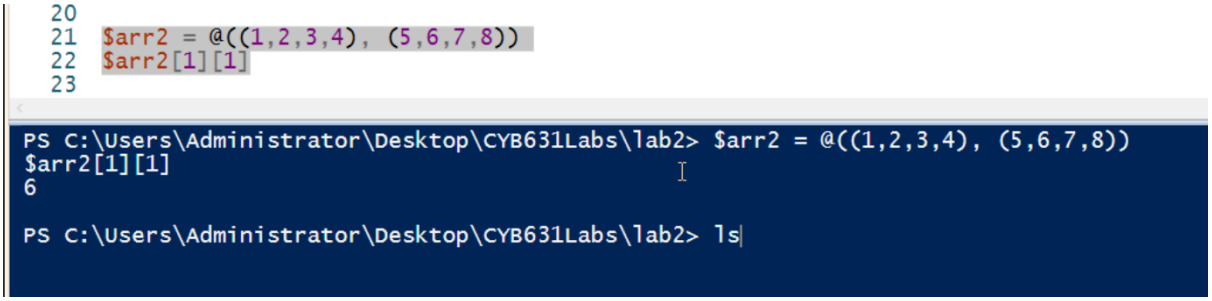

Figure 7

Step 12 — Sorting an Array of Strings

PowerShell can sort string arrays using the built-in .NET array methods.

Command sequence:

- $list = "watermelon","pear","banana","apple"
- [Array]::Sort($list)

- $list

Explanation:

- $list is initially unsorted: watermelon pear banana apple.
- [Array]::Sort($list) sorts the items in ascending (alphabetical) order.
- Printing $list after sorting shows: apple banana pear watermelon.

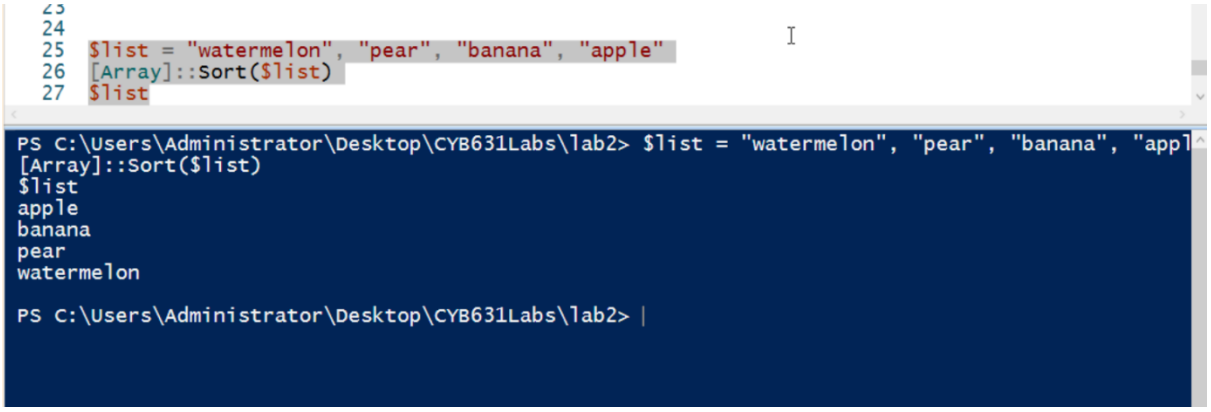

Figure 8

Step 13 — Result of Step 12

Answer: The sorted array is:

apple

banana

pear

watermelon

Also you can refer to Fig 8.

Step 14 — Creating and Using a Hashtable

A hashtable is a collection of key-value pairs, useful for quick lookups.

Command sequence:

- $htable = @{}
- $htable["Mary"] = 5

- $htable["John"] = 7
- $htable["Eric"] = 6
- $htable

Explanation:

- $htable = @{} initializes an empty hashtable.
- Each line assigns a value to a key ("Mary"=5, "John"=7, "Eric"=6).
- Printing $htable displays the key-value pairs.

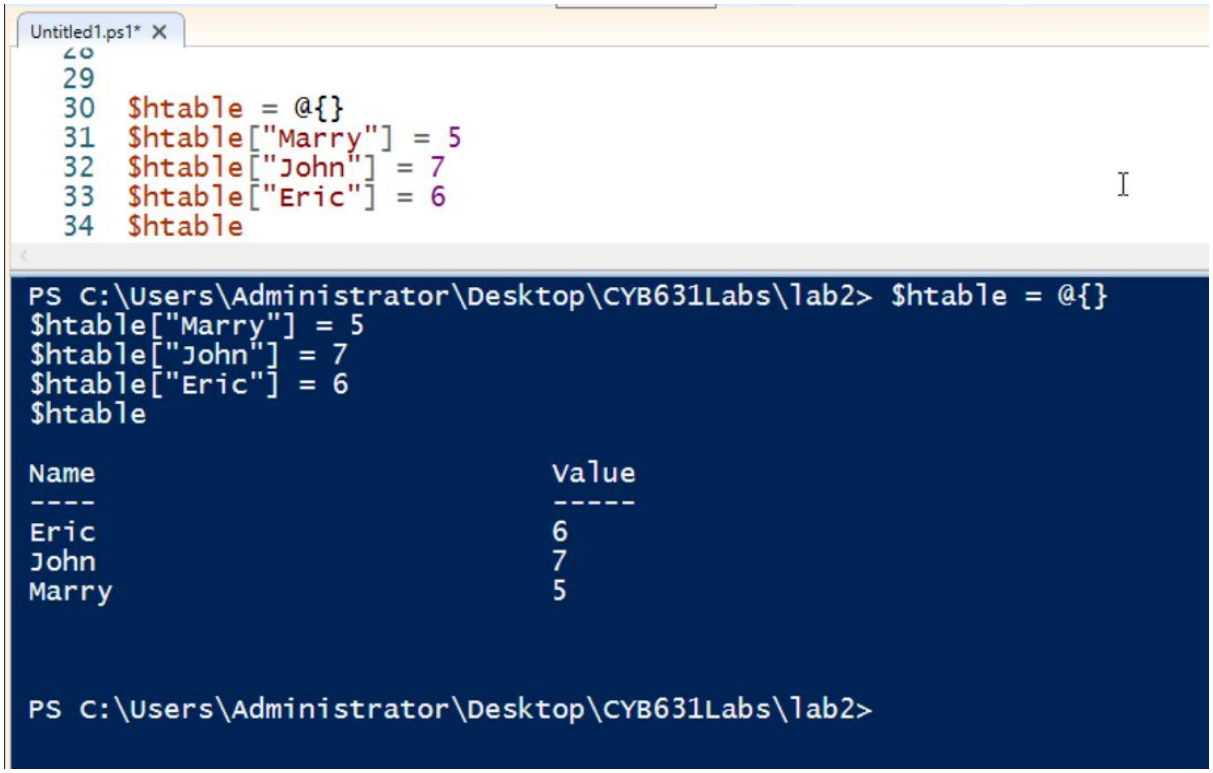

Figure 9

Step 15 — Paste Results

All the screenshots are shared in there respective steps.

EXERCISE III: FILES AND DIRECTORIES

Step 16 —Examine what files are in the current directory

In PowerShell, the **Get-ChildItem** cmdlet is used to display the contents of the current directory. This includes files and subdirectories, along with key details such as file size and last modification date.

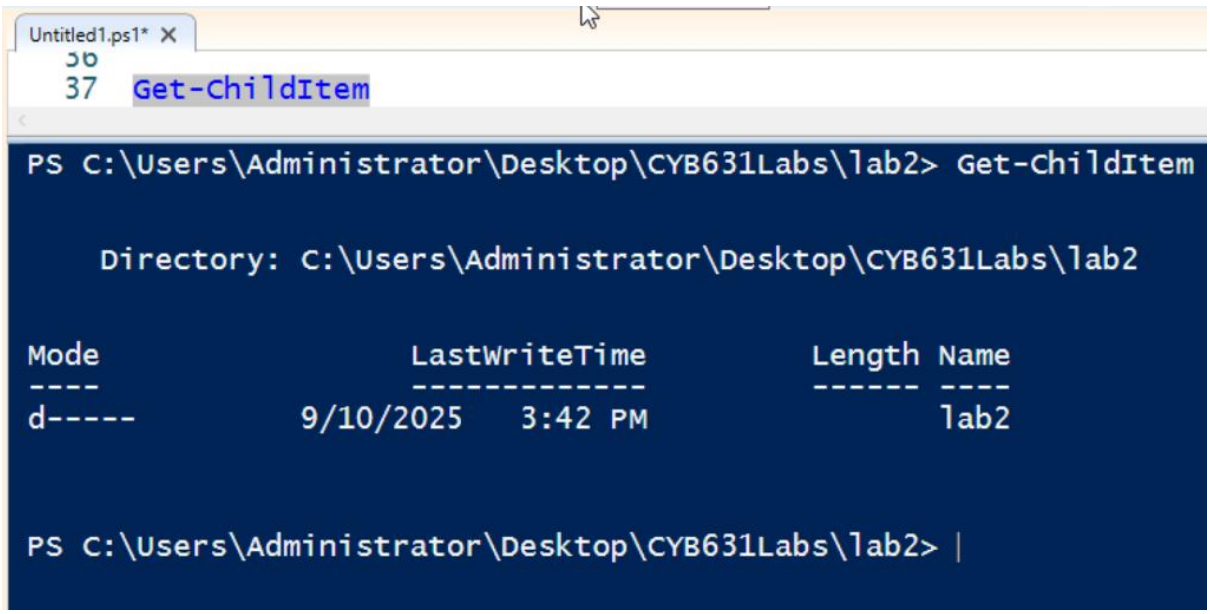

Figure 10

Step 18 — Comparing Two Files Using Compare-Object

The **Compare-Object** cmdlet highlights differences between two objects or files. In this step, we create two text files with slightly different contents and then compare them.

```
"Ten Apples Up on Top!" > .\file1.txt
"Ten Apples Up on Top*&^!" > .\file2.txt
$c1 = Get-Content .\file1.txt
$c2 = Get-Content .\file2.txt
Compare-Object $c1 $c2
```

Step 19 — Find Files Modified in the Past 10 Days and Sort by Length

PowerShell can filter files by their **LastWriteTime** property to show recently modified items. Sorting by file length helps quickly identify which files are larger.

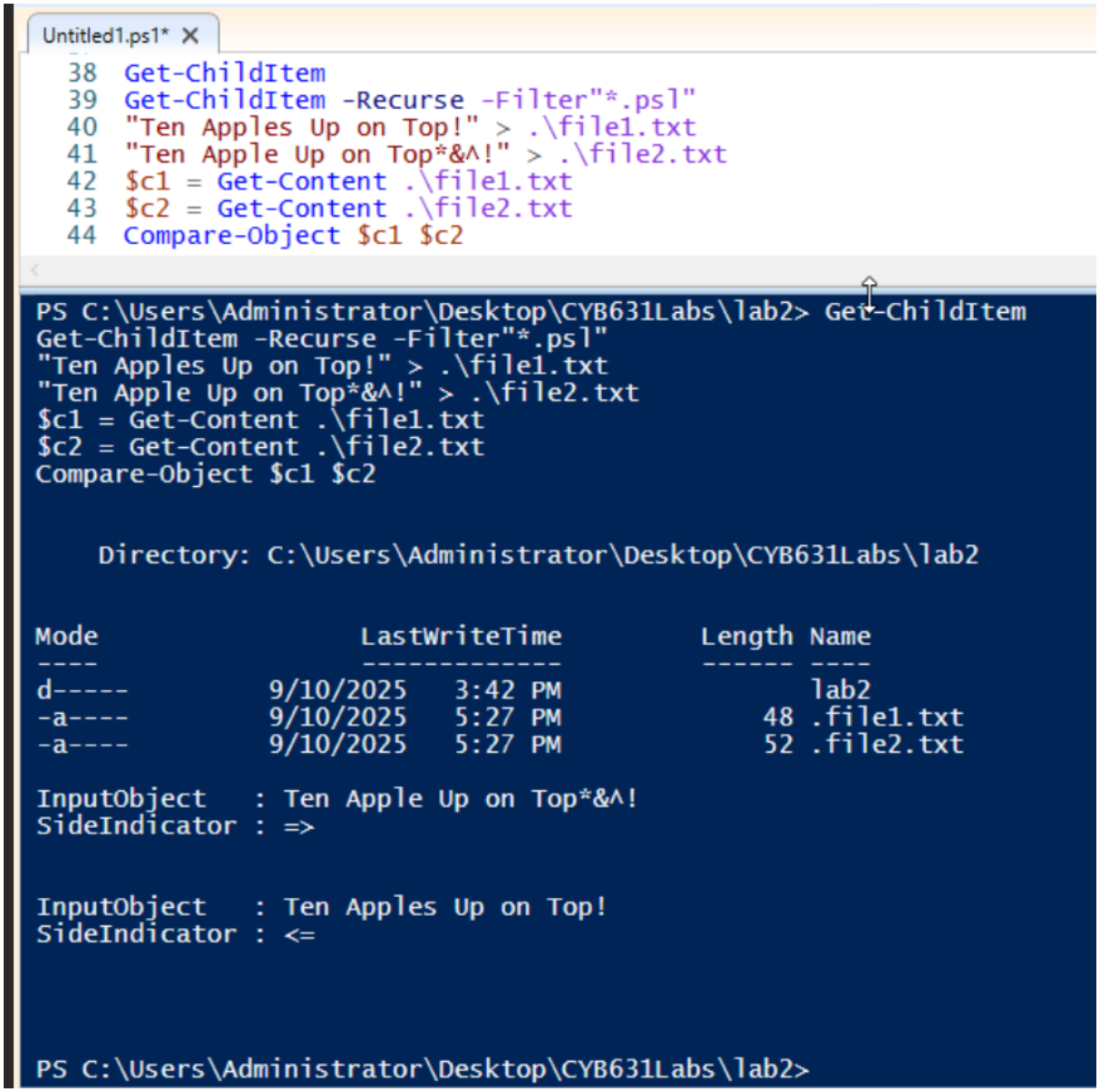

Figure 11

Step 20 — Screenshot

All the screenshot in the respective steps.

EXERCISE IV: SET PARAMETER FOR CALCULATING DISK USAGE

21 — Open and Review Get-DiskUsage.ps1

This includes files and subdirectories, along with key details such as file size and last

This step introduces a PowerShell script that reports directory sizes and demonstrates how to **accept an input parameter** (a switch) in a script.

powershell_ise .\lab2\Get-DiskUsage.ps1

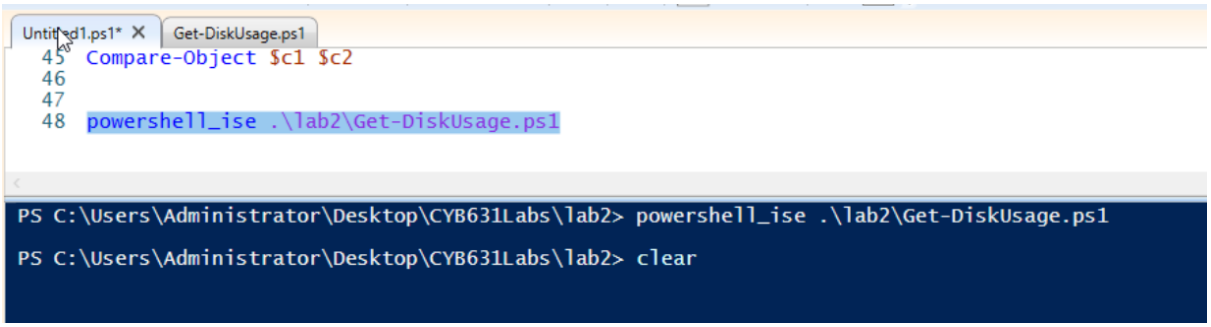

Figure 12

22 — Preparing the Script for Execution

Since the script came from Brightspace, it may be blocked by Windows security settings. Before running it, you must ensure scripts can execute.

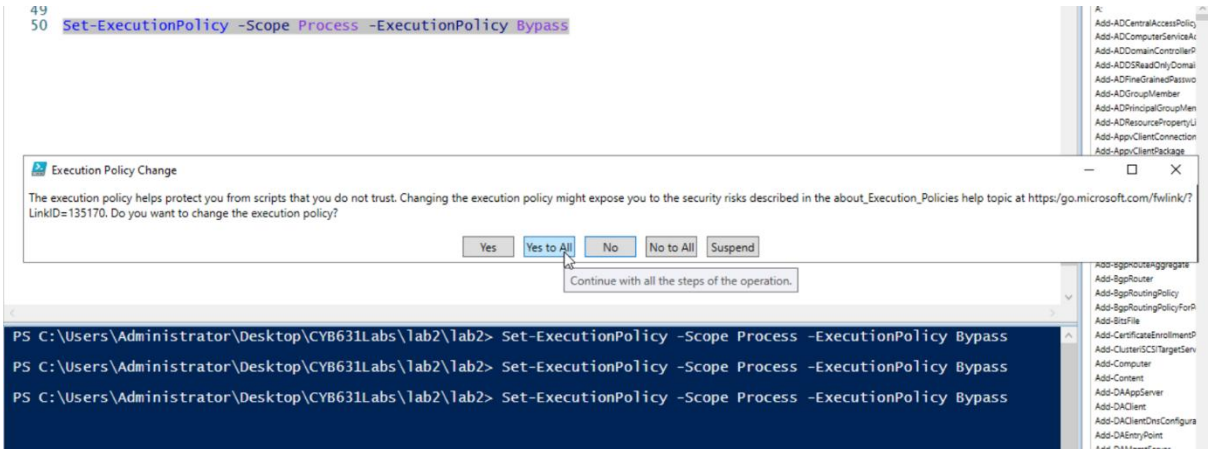

Figure 13

Step 23 — Creating Test Directories and Files

To test the script, create a folder structure with a few text files. This setup ensures there is meaningful data for the disk-usage report.

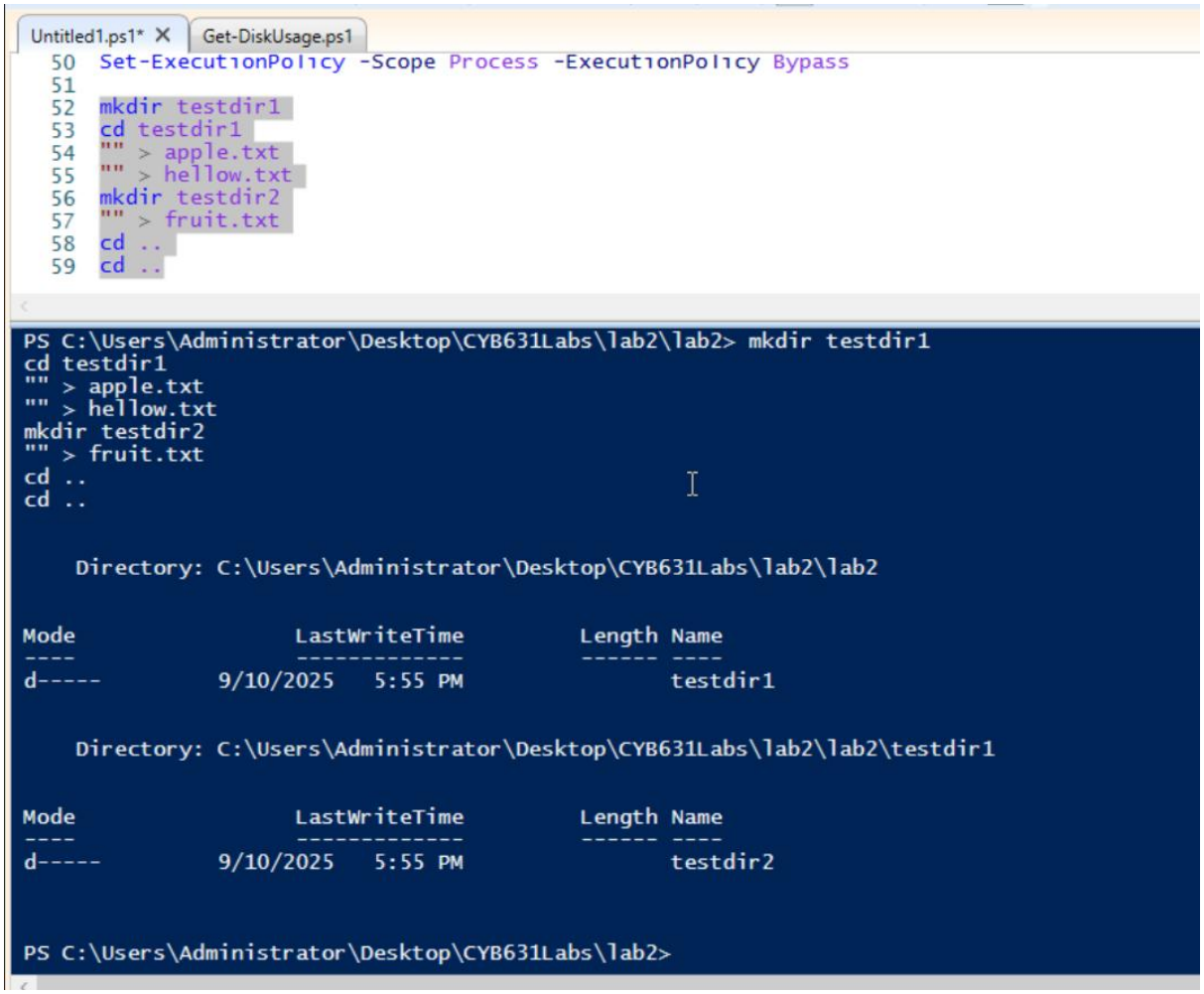

Figure 14

Step 24 — Running the Script With and Without the Parameter

Now runing the script twice — once normally, and once with the -IncludeSubdirectories switch.

```
61
62  .\Get-DiskUsage.ps1
63  .\Get-DiskUsage.ps1 -IncludeSubdirectories
```

```
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> .\Get-DiskUsage.ps1
.\Get-DiskUsage.ps1 -IncludeSubdirectories

FullName                                                              Size
--------                                                              ----
C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2\testdir1             18
C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2\testdir1\testdir2
                                                                         18


PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2>
```

Figure 15

Step 25 — Discussing the Differences in the Results

- **Without -IncludeSubdirectories:** The script reports folder sizes independently. The size of testdir1 only includes its direct files.
- **With -IncludeSubdirectories:** The script rolls up sizes to include all nested subfolders. This is why testdir1 appears larger if more files were inside testdir2.

Answer: The difference is that the -IncludeSubdirectories parameter provides a **recursive calculation of folder size**, while the default execution only measures **non-recursive sizes**.

Step 26 — Writing the showtoday.ps1 Script

We now create a new PowerShell script that demonstrates how to accept and use parameters. This script will display today's date. If the -ShowWeek parameter is included, only the weekday will be shown; otherwise, full date information will be displayed.

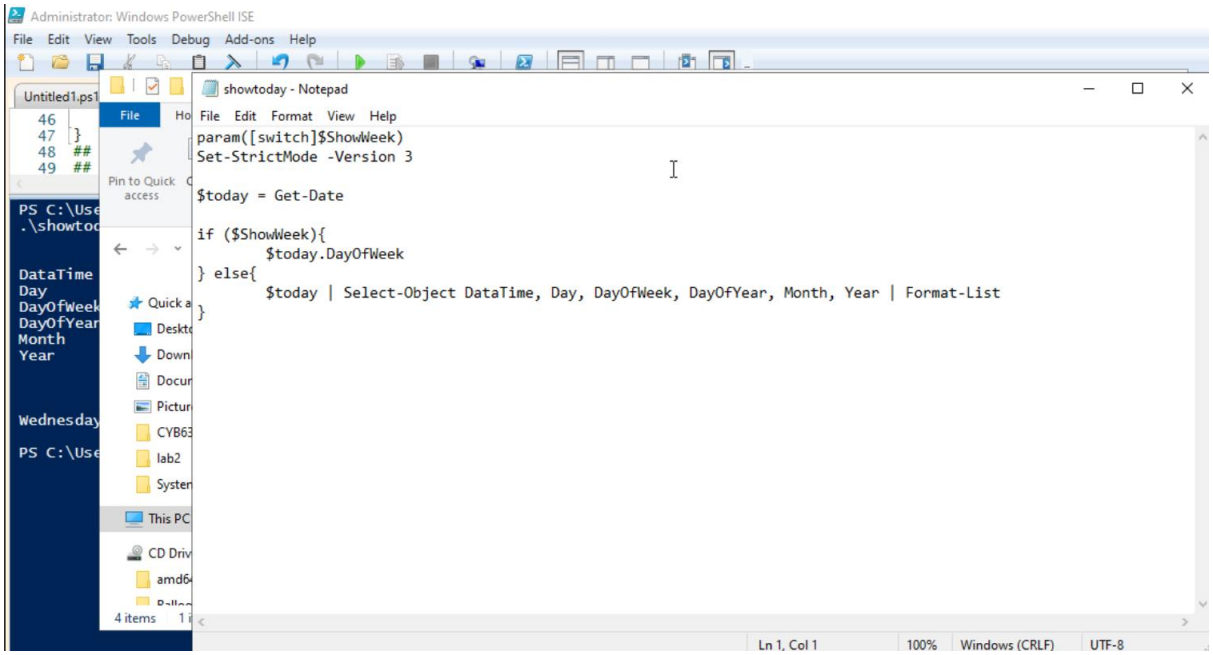

Figure 16

Step 27-28 — Screenshot of showtoday.ps1- Running showtoday.ps1 in Both Modes

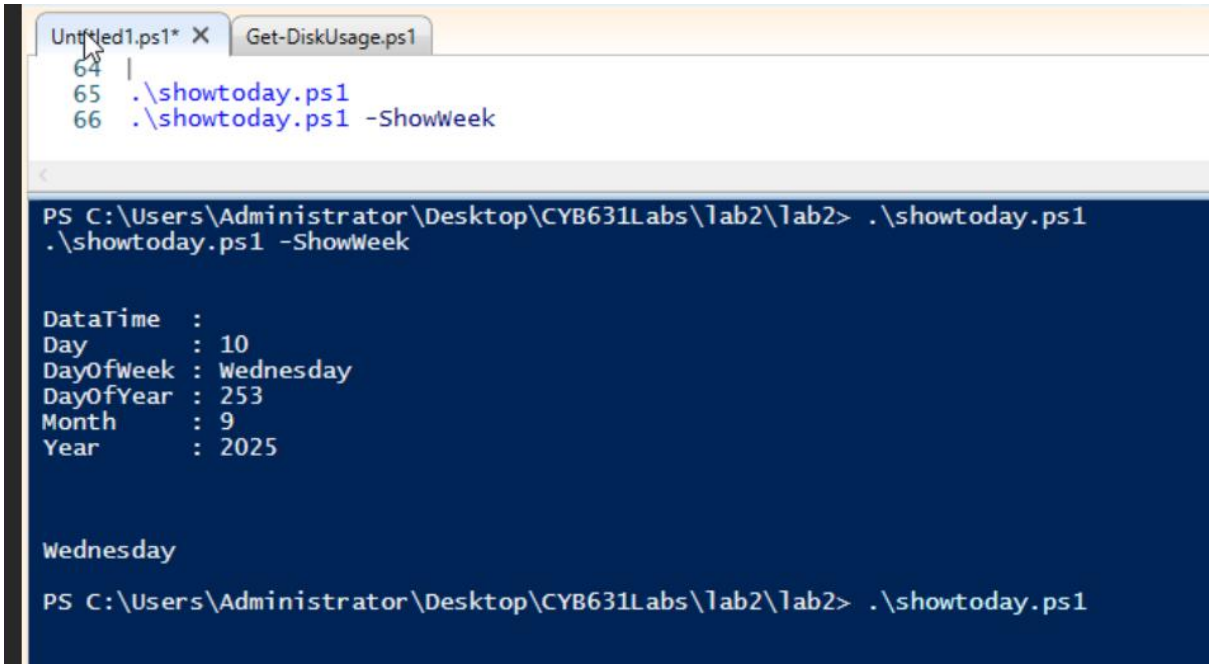

Figure 17

EXERCISE V: PACKAGE COMMANDS IN A MODULE

Step 29 — Understanding PowerShell Modules

PowerShell allows scripts and functions to be packaged into reusable **modules**. Once a script is turned into a module, it can be imported and used just like built-in cmdlets. This step introduces the process of converting your showtoday.ps1 script into a reusable PowerShell module.

Step 30 — Wrapping the Script Inside a Function

To convert a script into a module, the code must be wrapped inside a **function**. In this case, we will use the function name Show-Today.

Explanation:

- Wrapping the logic in a function makes the script callable by name (Show-Today).
- The functionality remains the same: with the -ShowWeek switch, only the day of the week is shown; without it, the full date details are displayed.

Step 31 — Saving the File as a Module

After editing the script, save it as Show-Today.psm1.

- The .psm1 extension indicates that this file is a **PowerShell module file**.
- Store it temporarily in your Lab 2 working directory before moving it to the module path.

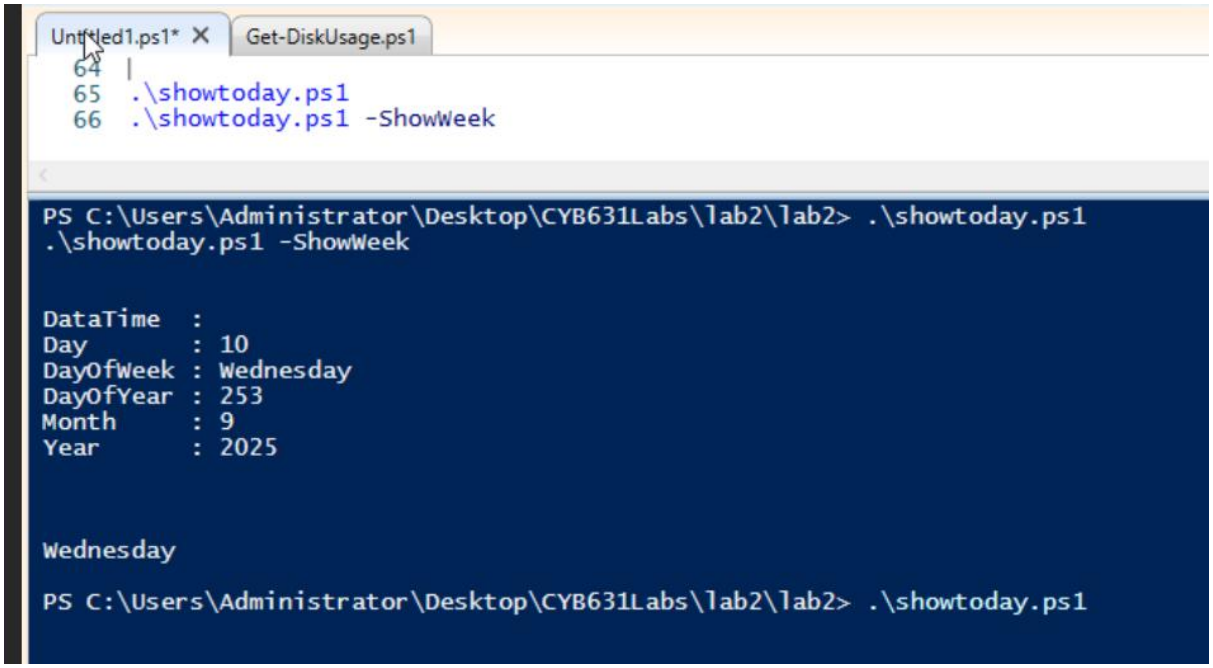

Figure 18

Step 32 — Checking the Module Path

To determine where PowerShell expects user modules to be stored, check the environment variable $env:PSModulePath.

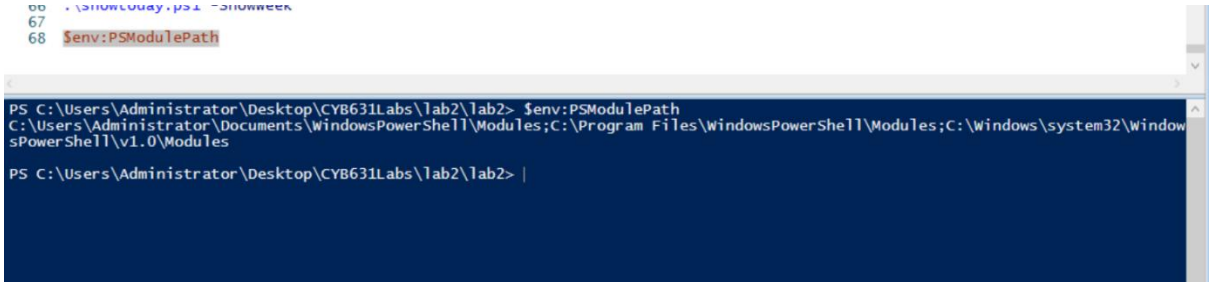

Figure 19

33 — Identifying the User Module Directory

The $env:PSModulePath environment variable shows all directories where PowerShell looks for modules. Running the command reveals multiple paths separated by semicolons.

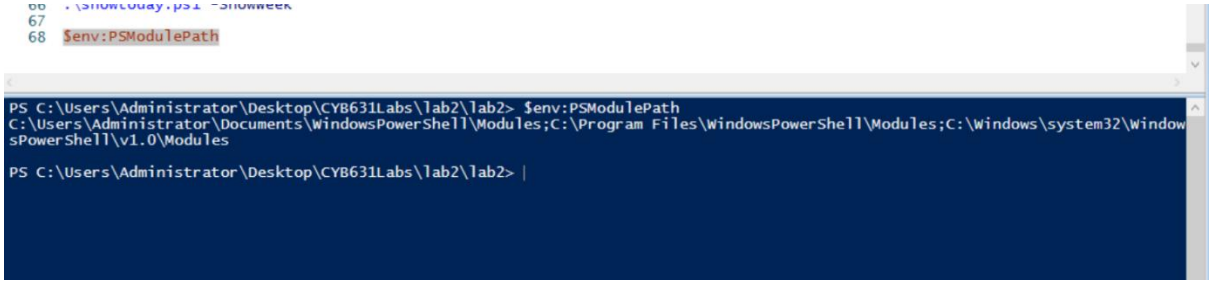

Figure 20

Step 34 — Creating a Subdirectory for the Module

Each module must reside in its own folder named after the module. Therefore, inside the modules directory, create a subfolder called **Show-Today**.

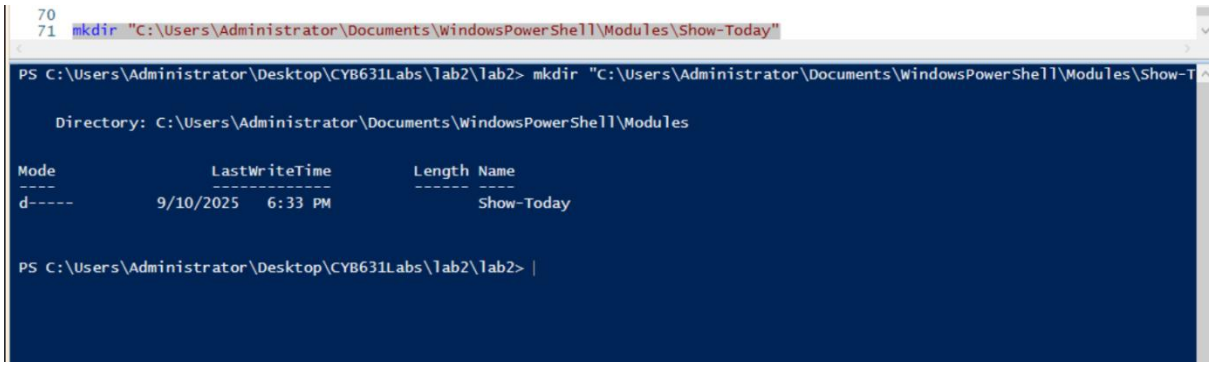

Figure 21

Step 35 — Moving the Module File

Move your Show-Today.psm1 file (created earlier) from the Lab 2 working directory into the new Show-Today module folder.

mv .\Show-Today.psm1 "C:\Users\Administrator\Documents\WindowsPowerShell\Modules\Show-Today"

Step 36 — Importing the Show-Today Module

After moving the file to the module directory, you can import it as a module: Import-Module Show-Today

Step 37 — Running the Show-Today Cmdlet

Once *Show-Today* is imported as a module, you can run it like any other PowerShell cmdlet:

Show-Today

Show-Today -ShowWeek

Step 38 — Verifying the Module Execution

Paste a screenshot of the above commands and their results to show that the module ran successfully.

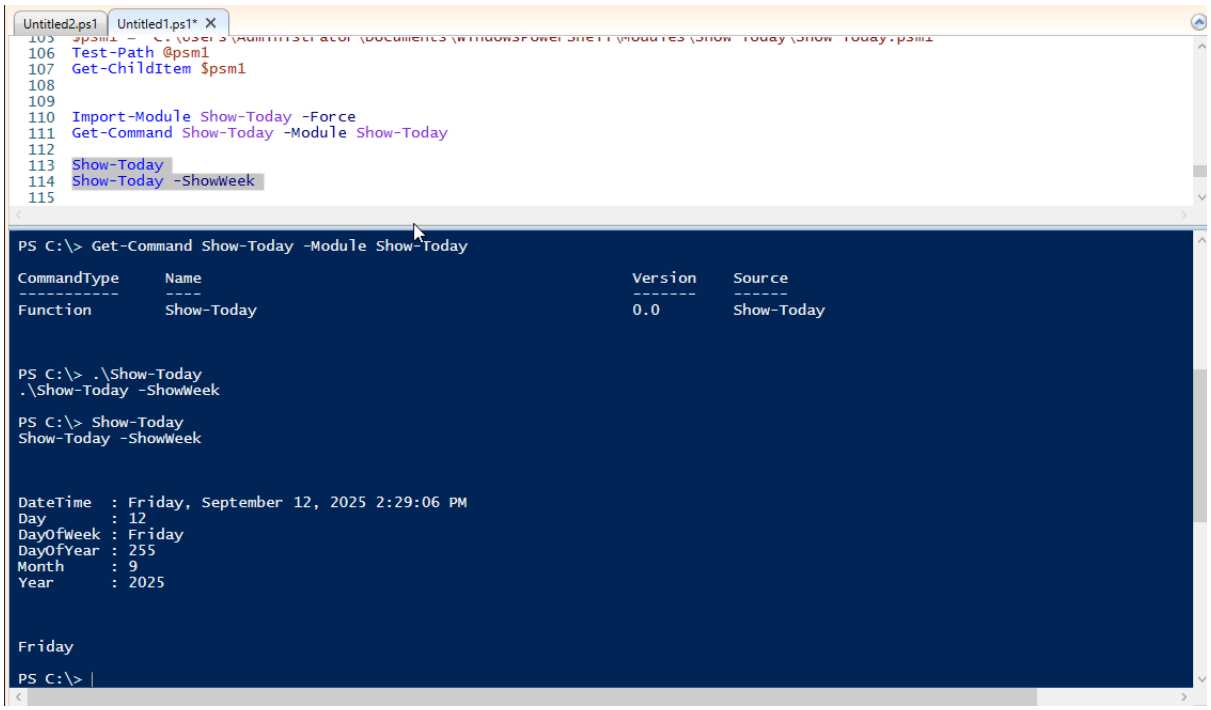

Figure 22: Finally, I made it :)

Exercise VI: Event Logs

Step 39 — Listing Event Logs

To determine the event logs available on the system, I ran:

Get-EventLog -List

Get-EventLog -List | Select-Object LogDisplayName, MaximumKilobytes

Observed result: Displayed all event logs along with their maximum sizes. This helps identify which logs exist and how much storage is allocated for each.

```
118
119 Get-EventLog -List
120 Get-EventLog -List | Select-Object LogDisplayName, MaximumKilobytes
```

```
PS C:\> Get-EventLog -List
Get-EventLog -List | Select-Object LogDisplayName, MaximumKilobytes

  Max(K) Retain OverflowAction       Entries Log
  ------ ------ --------------       ------- ---
     512      7 OverwriteOlder             0 Active Directory Web Services
  20,480      0 OverwriteAsNeeded     30,978 Application
  15,168      0 OverwriteAsNeeded        670 DFS Replication
     512      7 OverwriteOlder             0 Directory Service
  20,480      0 OverwriteAsNeeded          0 HardwareEvents
     512      7 OverwriteOlder             0 Internet Explorer
  20,480      0 OverwriteAsNeeded          0 Key Management Service
  20,480      0 OverwriteAsNeeded     26,290 Security
  20,480      0 OverwriteAsNeeded     64,244 System
  15,360      0 OverwriteAsNeeded      3,264 Windows PowerShell

LogDisplayName   : Active Directory Web Services
MaximumKilobytes : 512


LogDisplayName   : Application
MaximumKilobytes : 20480


LogDisplayName   : DFS Replication
MaximumKilobytes : 15168


LogDisplayName   : Directory Service
MaximumKilobytes : 512


LogDisplayName   : Hardware Events
MaximumKilobytes : 20480


LogDisplayName   : Internet Explorer
MaximumKilobytes : 512


LogDisplayName   : Key Management Service
MaximumKilobytes : 20480


LogDisplayName   : Security
MaximumKilobytes : 20480


LogDisplayName   : System
MaximumKilobytes : 20480


LogDisplayName   : Windows PowerShell
MaximumKilobytes : 15360



PS C:\>
```

Figure 23

Step 40 — Determining Application and Service Logs

I used Get-WinEvent to enumerate application and service logs:

Get-WinEvent -ListLog * | Select-Object LogName, RecordCount

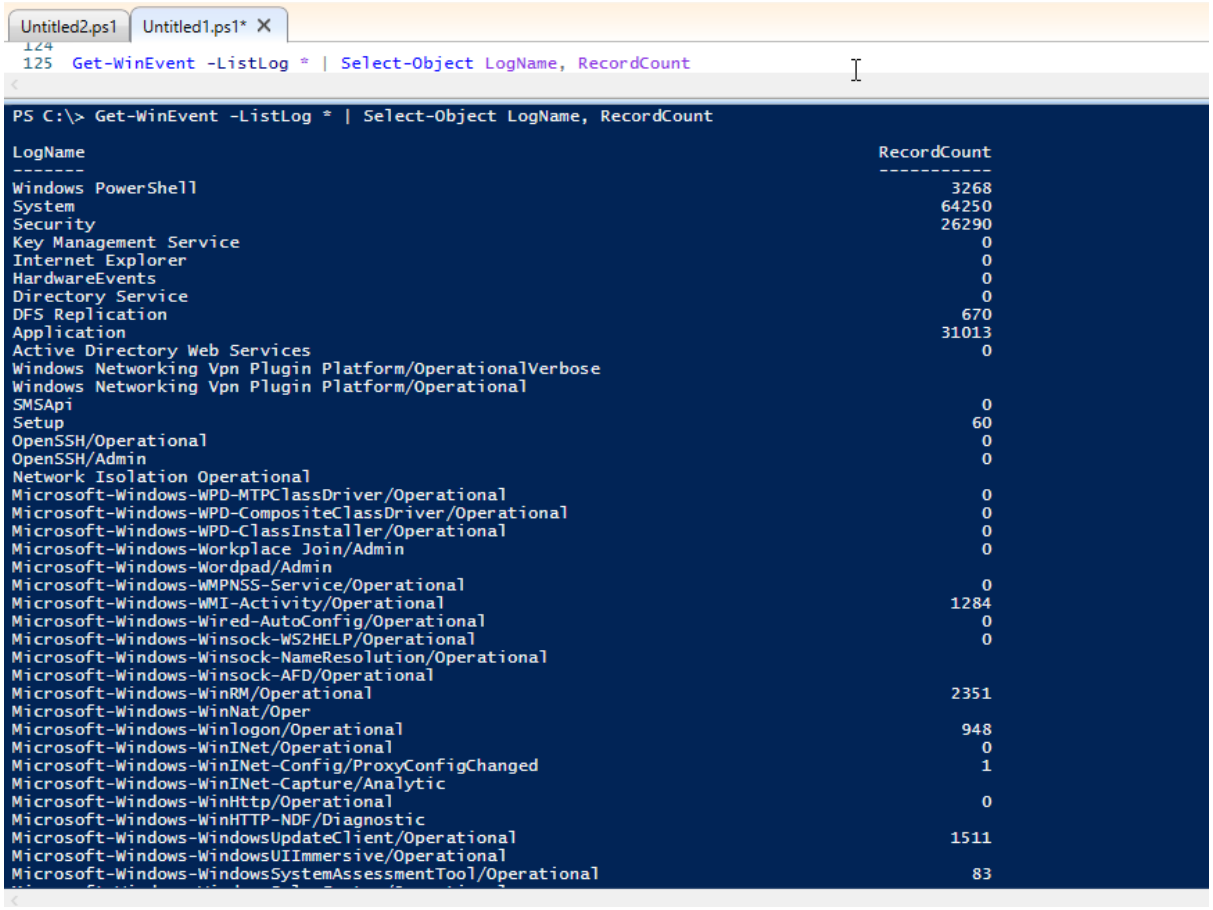

Figure 24

Step 41 — Obtaining Recent Security Log Entries

To view the 10 most recent security log entries:

Get-EventLog Security -Newest 10 | Format-Table Index, Source, Message -AutoSize

```
131  Get-EventLog Security -Newest 10 | Format-Table Index, Source, Message -AutoSize
```

```
PS C:\> Get-EventLog Security -Newest 10 | Format-Table Index, Source, Message -AutoSize

Index Source                                Message
----- ------                                -------
85743 Microsoft-Windows-Security-Auditing Special privileges assigned to new logon....
85742 Microsoft-Windows-Security-Auditing An account was successfully logged on....
85741 Microsoft-Windows-Security-Auditing Special privileges assigned to new logon....
85740 Microsoft-Windows-Security-Auditing An account was successfully logged on....
85739 Microsoft-Windows-Security-Auditing Special privileges assigned to new logon....
85738 Microsoft-Windows-Security-Auditing An account was successfully logged on....
85737 Microsoft-Windows-Security-Auditing Credential Manager credentials were read....
85736 Microsoft-Windows-Security-Auditing Credential Manager credentials were read....
85735 Microsoft-Windows-Security-Auditing Credential Manager credentials were read....
85734 Microsoft-Windows-Security-Auditing Credential Manager credentials were read....


PS C:\>
```

Figure 25

Step 42 — Filtering Recent System Events

To search the most recent 100 system events for specific text (example: “service”):

Get-EventLog System -Newest 100 | Where-Object { $_.Message -match "service" }

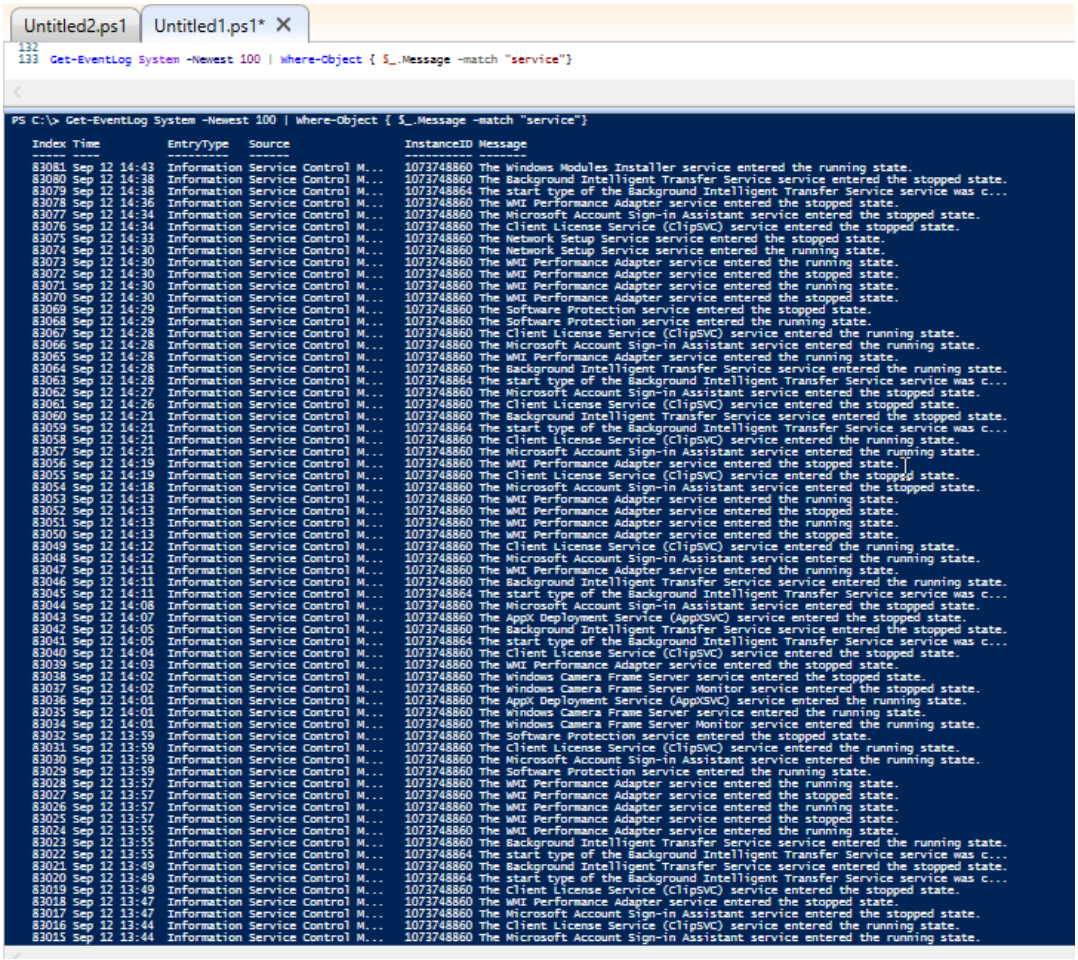

Figure 26

Step 43 — Listing Today's System Events

The following PowerShell command lists all system events generated **today**: Get-EventLog System | Where-Object { $_.TimeGenerated -ge (Get-Date).Date }

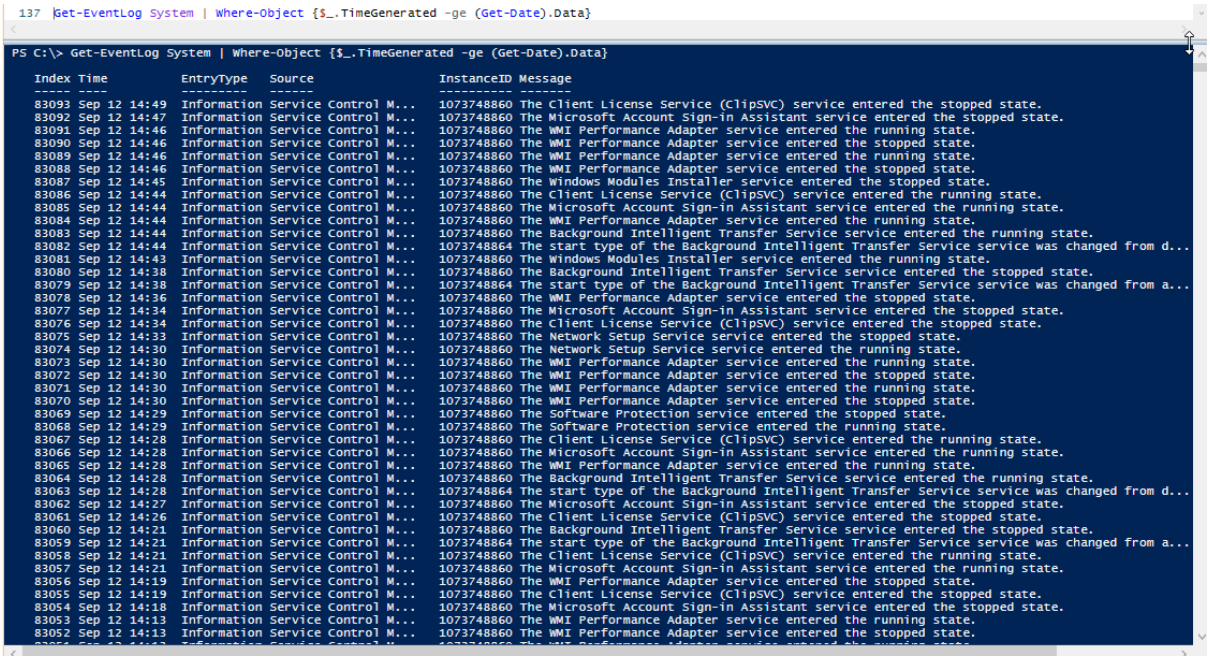

Figure 27

Step 44 — Importance of Analyzing Logs for Host Security

Analyzing logs is critical for host security because:

- Logs provide visibility into **suspicious or unauthorized activity** (failed logins, privilege escalation).
- They help detect **malware infections, misconfigurations, or hardware failures**.
- Logs serve as **forensic evidence** during incident response and investigations.
- Regular log analysis improves compliance with **security frameworks and regulations** (NIST, ISO 27001, SOC 2).

In short: Without monitoring logs, malicious activity could go undetected, making systems vulnerable to compromise.

Exercise VII: System Service

Step 45 — Listing Running Services

To view only the services currently running on the system, I used: Get-Service | Where-Object { $_.Status -eq "Running" }

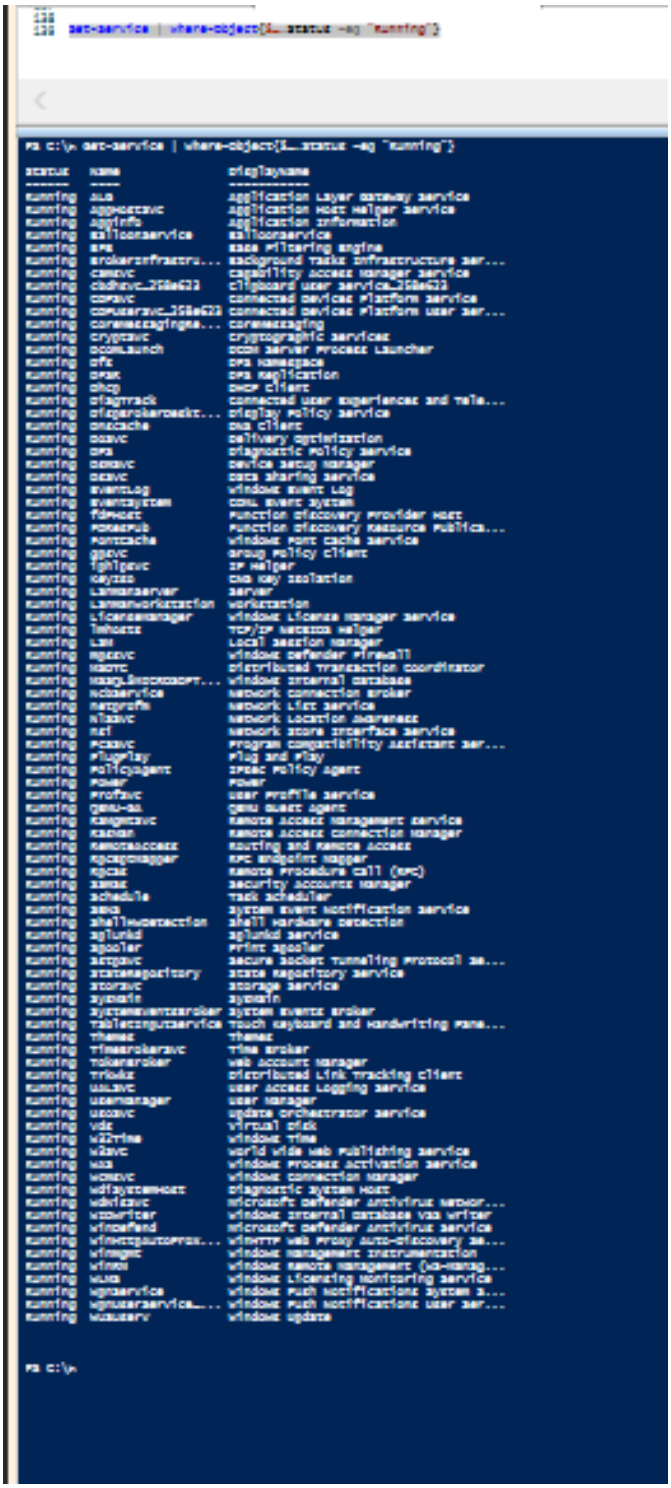

Figure 28

Step 46 — Sorting Services by Dependency Count

I then ran the same command but sorted the results based on how many services depend on each:

```
141
142 Get-Service | Where-Object {$_.Status -eq "Running"} |Sort-Object -Descending {$_.DependentServices.Count}
```

```
PS C:\> Get-Service | Where-Object {$_.Status -eq "Running"} |Sort-Object -Descending {$_.DependentServices.Count}

Status   Name               DisplayName
------   ----               -----------
Running  DcomLaunch         DCOM Server Process Launcher
Running  RpcEptMapper       RPC Endpoint Mapper
Running  RpcSs              Remote Procedure Call (RPC)
Running  nsi                Network Store Interface Service
Running  BFE                Base Filtering Engine
Running  BrokerInfrastru... Background Tasks Infrastructure Ser...
Running  Dhcp               DHCP Client
Running  Dnscache           DNS Client
Running  SamSs              Security Accounts Manager
Running  ProfSvc            User Profile Service
Running  EventSystem        COM+ Event System
Running  EventLog           Windows Event Log
Running  SstpSvc            Secure Socket Tunneling Protocol Se...
Running  LanmanWorkstation  Workstation
Running  RasMan             Remote Access Connection Manager
Running  WIDWriter          Windows Internal Database VSS Writer
Running  CryptSvc           Cryptographic Services
Running  NlaSvc             Network Location Awareness
Running  WinHttpAutoProx... WinHTTP Web Proxy Auto-Discovery Se...
Running  Winmgmt            Windows Management Instrumentation
Running  RemoteAccess       Routing and Remote Access
Running  KeyIso             CNG Key Isolation
Running  netprofm           Network List Service
Running  LanmanServer       Server
Running  NcbService         Network Connection Broker
Running  MSSQL$MICROSOFT... Windows Internal Database
Running  iphlpsvc           IP Helper
Running  SENS               System Event Notification Service
Running  SystemEventsBroker System Events Broker
Running  UserManager        User Manager
Running  StateRepository    State Repository Service
Running  WAS                Windows Process Activation Service
Running  fdPHost            Function Discovery Provider Host
Running  StorSvc            Storage Service
Running  SysMain            SysMain
Running  Spooler            Print Spooler
```

Figure 29

Step 46 — Sorting Services by Dependency Count

All screenshot provided in above two steps

Step 48 — Importance of Knowing Running Services for Host Security

It is important for host security to know what services are running because:

- **Attack Surface Reduction**: Each running service is a potential entry point. Disabling unnecessary ones minimizes risk.
- **Malware Detection**: Attackers often install rogue services that persist after reboot. Monitoring reveals anomalies.
- **Performance & Stability**: Resource-heavy or misconfigured services can degrade system performance and availability.

- **Privilege & Access Control**: Some services run with elevated privileges; identifying them helps ensure they are hardened and monitored.
- **Compliance & Forensics**: Auditing services is required for many standards (e.g., NIST, ISO 27001) and helps during incident investigations.

In summary: Actively monitoring services ensures that only legitimate, necessary, and secure services are running, thereby reducing risks and supporting overall host hardening.

Exercise VIII: Develop your system admin script

Step 49 — Developing the sys_admin.ps1 Script

I created a new PowerShell script named **sys_admin.ps1**. This script was designed for system administrators to collect host information and produce a report. The script includes the following functionality:

- Records the date and time of the report.
- Accepts a **-ShowService** parameter. If included, it lists the top five running services sorted by the number of dependent services. If omitted, this section is skipped.
- In both cases, it summarizes entries from the Security event log, grouped and sorted by source name, showing the top five sources.

The script saves its output as a text file inside the Reports directory, with filenames based on the current timestamp.

Step 50 — sys_admin.ps1 Script File

I saved the script as **sys_admin.ps1** in the Scripts directory of my Lab 2 folder. A screenshot of the script file in the PowerShell ISE/VS Code editor is shown in Figure XX.

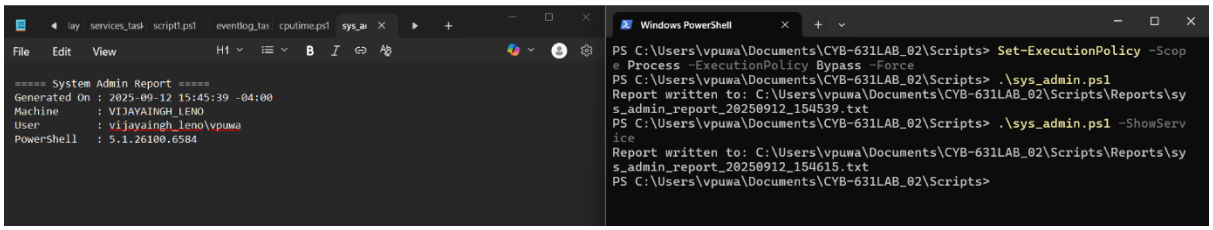

Figure 30

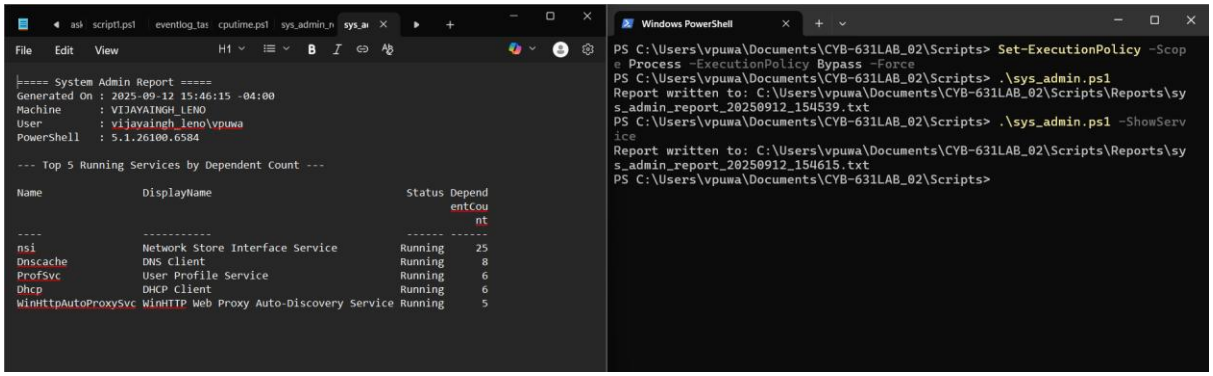

Figure 31

Step 51 — Running sys_admin.ps1

I executed the script in two ways:

1. Without parameters:
2. .\sys_admin.ps1

This generated a report with timestamp, machine information, and Security log summary.

Example header from the report:

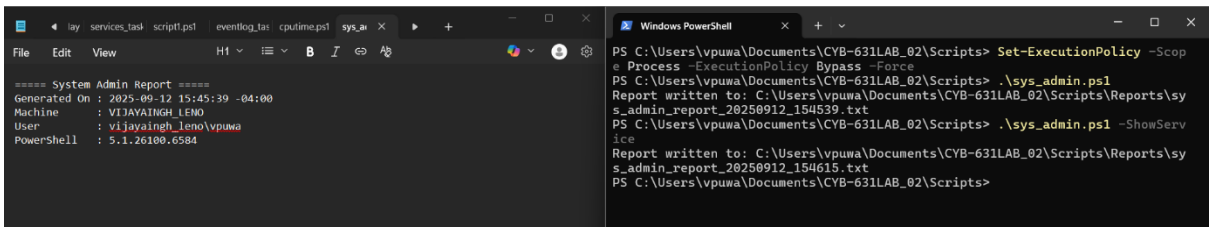

Figure 32

2. With the **-ShowService** parameter:
3. .\sys_admin.ps1 -ShowService

This generated the same header along with the **Top 5 Running Services by Dependent Count** table:

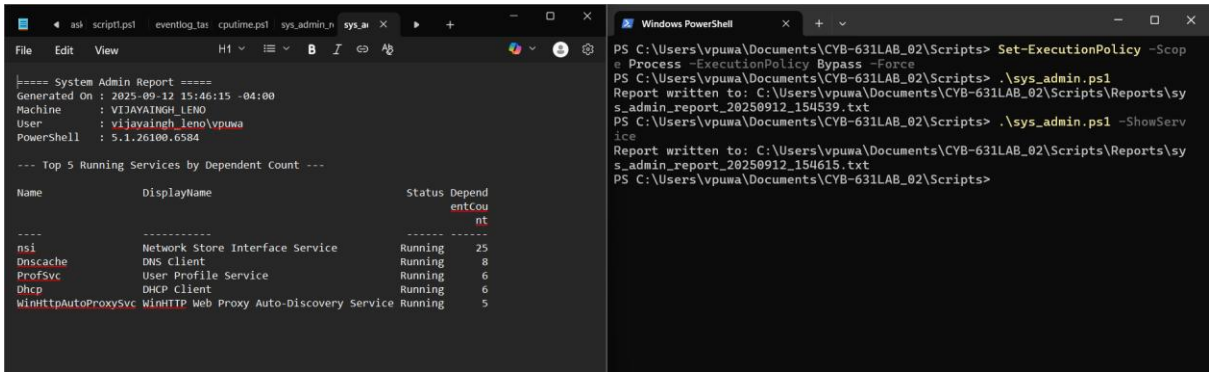
File Edit View
===== System Admin Report =====
Generated On : 2025-09-12 15:46:15 -04:00
Machine : VIJAYAINGH_LENO
User : vijayaingh_leno\vpuwa
PowerShell : 5.1.26100.6584
--- Top 5 Running Services by Dependent Count ---
Name DisplayName Status DependentCount
nsi Network Store Interface Service Running 25
Dnscache DNS Client Running 8
ProfSvc User Profile Service Running 6
Dhcp DHCP Client Running 6
WinHttpAutoProxySvc WinHTTP Web Proxy Auto-Discovery Service Running 5
Windows PowerShell
PS C:\Users\vpuwa\Documents\CYB-631LAB_02\Scripts> Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force
PS C:\Users\vpuwa\Documents\CYB-631LAB_02\Scripts> .\sys_admin.ps1
Report written to: C:\Users\vpuwa\Documents\CYB-631LAB_02\Scripts\Reports\sys_admin_report_20250912_154539.txt
PS C:\Users\vpuwa\Documents\CYB-631LAB_02\Scripts> .\sys_admin.ps1 -ShowService
Report written to: C:\Users\vpuwa\Documents\CYB-631LAB_02\Scripts\Reports\sys_admin_report_20250912_154615.txt
PS C:\Users\vpuwa\Documents\CYB-631LAB_02\Scripts>

Figure 33

Lab and Class Reflection

1. What did I like about this lab?

I appreciated that this lab combined multiple aspects of PowerShell into a structured workflow. It started with object operations and arrays, then moved into modules, event logs, and finally developing a complete administrative script. The progression showed me how small building blocks can be combined into powerful tools. Writing and running my own sys_admin.ps1 felt very practical and connected directly to real system administration tasks.

2. What challenges did I encounter?

The main challenges were handling execution policies and accessing the Security event log. Initially, I received errors due to restricted script execution and insufficient privileges when trying to read Security logs. I resolved these by using a **process-scoped policy bypass** for testing and running PowerShell with administrative privileges. Another challenge was dealing with null values for DependentServices.Count, which required rewriting the script logic to avoid runtime errors.

3. Suggestions for improving the class:

It would be helpful if the lab instructions included more examples directly related to **cybersecurity use cases**, such as analyzing failed login attempts or automating log correlation. Including short troubleshooting notes (e.g., why Security log access fails without Admin rights) would save time for students new to PowerShell. Additional practice on error handling in scripts would also reinforce the idea of writing resilient automation tools.

4. Turning off virtual machines:

As instructed, I shut down the Windows virtual machine after completing the lab to free up server resources.

References

Holmes, L. (2021). *Windows PowerShell cookbook* (4th ed.). O'Reilly Media.

Microsoft. (2023, June 12). *Table of basic PowerShell commands*. Microsoft DevBlogs. https://devblogs.microsoft.com/scripting/table-of-basic-powershell-commands/

Pace University. (2025). *CYB 631: Automating Information Security with Python and Shell Scripting – Lab 2: Analyzing Logs and Other Administrators' Tasks* [Course handout]. Department of Computer Science, Seidenberg School of CSIS.