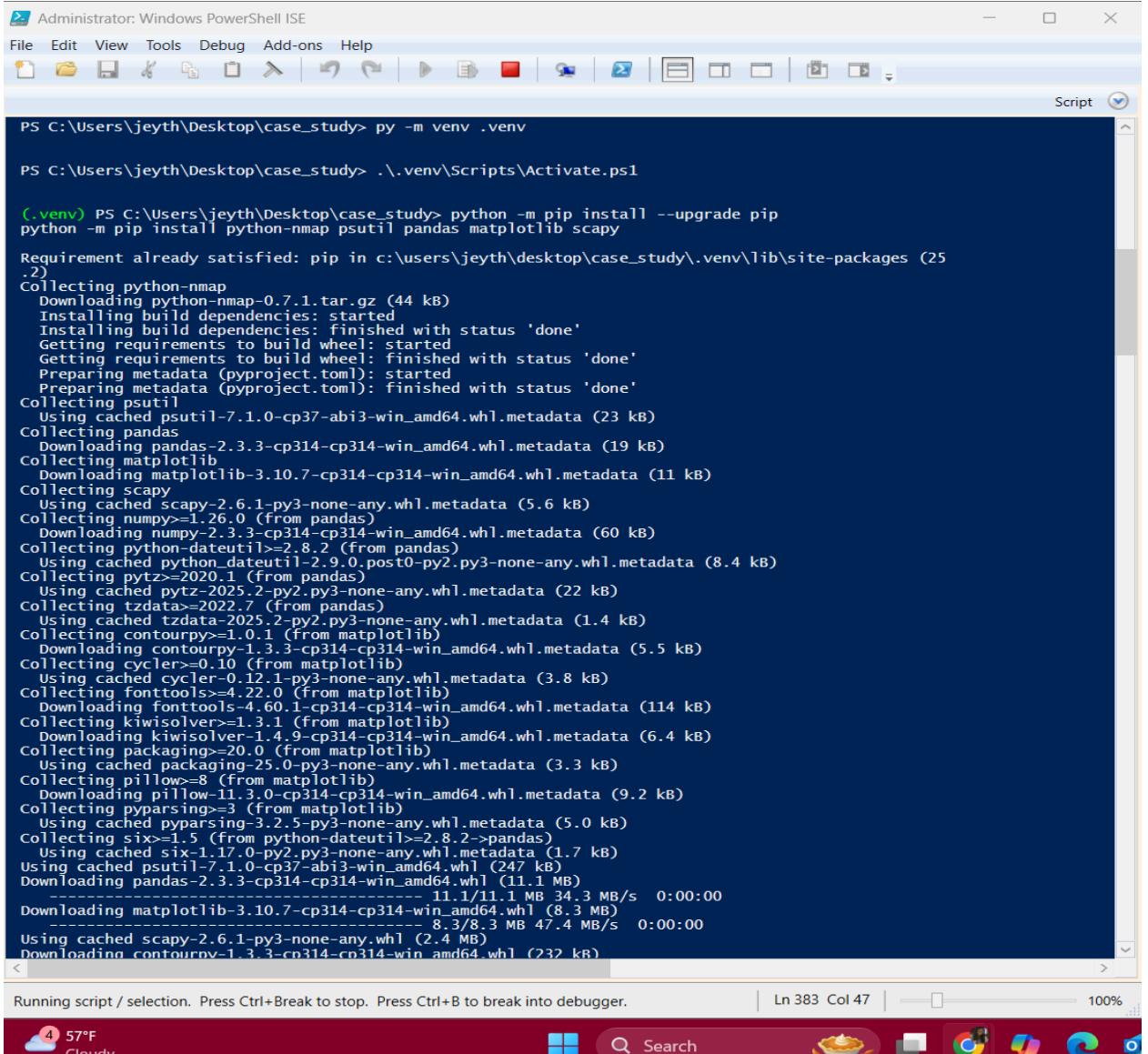


**Wonderville Town IT – Security Automation Case Study 2**  
**Automating Security and System Monitoring for Wonderville Hosts**  
**Atmtng Info Scrt Python & Shll FALL 2025 72922 - CYB-631**  
**Jeytha Sahana Venkatesh Babu**  
**Date: October 13, 2025**

## Executive Summary:

This case study demonstrates the application of Python automation to enhance the security and operational monitoring of the Wonderville town network. Our team focused on **automating network service discovery, identifying resource-intensive processes, analyzing Windows host logs, and analyzing network traffic**. Using Python scripts integrated with system utilities and data exports from Windows hosts and Linux servers, we generated comprehensive reports including CSV tables and visual charts. These automation solutions provide actionable insights, helping the IT department proactively monitor the system and secure the network while saving manual administrative effort.

## Setup



The screenshot shows a Windows PowerShell ISE window titled "Administrator: Windows PowerShell ISE". The code in the script pane is as follows:

```
PS C:\Users\jeyth\Desktop\case_study> py -m venv .venv
PS C:\Users\jeyth\Desktop\case_study> .\.venv\Scripts\Activate.ps1

(.venv) PS C:\Users\jeyth\Desktop\case_study> python -m pip install --upgrade pip
python -m pip install python-nmap psutil pandas matplotlib scapy

Requirement already satisfied: pip in c:\users\jeyth\desktop\case_study\.venv\lib\site-packages (25.2)
Collecting python-nmap
  Downloading python-nmap-0.7.1.tar.gz (44 kB)
    Installing build dependencies: started
      Installing build dependencies: finished with status 'done'
        Getting requirements to build wheel: started
          Getting requirements to build wheel: finished with status 'done'
            Preparing metadata (pyproject.toml): started
              Preparing metadata (pyproject.toml): finished with status 'done'
            Collecting psutil
              Using cached psutil-7.1.0-cp37abi3-win_amd64.whl.metadata (23 kB)
            Collecting pandas
              Downloading pandas-2.3.3-cp314-cp314-win_amd64.whl.metadata (19 kB)
            Collecting matplotlib
              Downloading matplotlib-3.10.7-cp314-cp314-win_amd64.whl.metadata (11 kB)
            Collecting scapy
              Using cached scapy-2.6.1-py3-none-any.whl.metadata (5.6 kB)
            Collecting numpy>=1.26.0 (from pandas)
              Downloading numpy-2.3.3-cp314-cp314-win_amd64.whl.metadata (60 kB)
            Collecting python-dateutil<=2.8.2 (from pandas)
              Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata (8.4 kB)
            Collecting pytz>=2020.1 (from pandas)
              Using cached pytz-2025.2-py2.py3-none-any.whl.metadata (22 kB)
            Collecting tzdata>=2022.7 (from pandas)
              Using cached tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
            Collecting contourpy>=1.0.1 (from matplotlib)
              Downloading contourpy-1.3.3-cp314-cp314-win_amd64.whl.metadata (5.5 kB)
            Collecting cycler>=0.10 (from matplotlib)
              Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
            Collecting fonttools>=4.22.0 (from matplotlib)
              Downloading fonttools-4.60.1-cp314-cp314-win_amd64.whl.metadata (114 kB)
            Collecting kiwisolver>=1.3.1 (from matplotlib)
              Downloading kiwisolver-1.4.9-cp314-cp314-win_amd64.whl.metadata (6.4 kB)
            Collecting packaging>=20.0 (from matplotlib)
              Using cached packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
            Collecting pillow>=8 (from matplotlib)
              Downloading pillow-11.3.0-cp314-cp314-win_amd64.whl.metadata (9.2 kB)
            Collecting pyparsing>=3 (from matplotlib)
              Using cached pyparsing-3.2.5-py3-none-any.whl.metadata (5.0 kB)
            Collecting six>=1.5 (from python-dateutil<=2.8.2->pandas)
              Using cached six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
            Using cached psutil-7.1.0-cp37abi3-win_amd64.whl (247 kB)
            Downloading pandas-2.3.3-cp314-cp314-win_amd64.whl (11.1 MB)
              11.1/11.1 MB 34.3 MB/s 0:00:00
            Downloading matplotlib-3.10.7-cp314-cp314-win_amd64.whl (8.3 MB)
              8.3/8.3 MB 47.4 MB/s 0:00:00
            Using cached scapy-2.6.1-py3-none-any.whl (2.4 kB)
            Downloading contourpy-1.3.3-cp314-cp314-win_amd64.whl (232 kB)

PS C:\Users\jeyth\Desktop\case_study>
```

The status bar at the bottom indicates "Running script / selection. Press Ctrl+Break to stop. Press Ctrl+B to break into debugger." and shows "Ln 383 Col 47" and "100%". The taskbar at the bottom includes icons for File Explorer, Task View, Start, Search, Task Manager, Google Chrome, and Edge.

```
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script

Mode           LastWriteTime      Length Name
d---- 10/14/2025 11:21 AM          Nmap

(.venv) PS C:\Users\jeyth\Desktop\case_study> # set variable for the Nmap folder we found
$nmapPath = "C:\Program Files (x86)\Nmap"

# add to current session PATH
$env:Path = $nmapPath + ";" + $env:Path

# verify
nmap --version

Nmap version 7.98 ( https://nmap.org )
Platform: i686-pc-windows-ws
Compiled with: nmap-liblua-5.4.8 openssl-3.0.17 nmap-libssh2-1.11.1 nmap-libz-1.3.1 nmap-libpcre2-1
0.45 Npcap-1.83 nmap-libdnet-1.18.0 ipv6
Compiled without:
Available nsock engines: iocp poll select

(.venv) PS C:\Users\jeyth\Desktop\case_study> # run in an elevated PowerShell window
$addPath = "C:\Program Files (x86)\Nmap"
$current = [Environment]::GetEnvironmentVariable("Path", [EnvironmentVariableTarget]::Machine)
if (-not ($current.Split(';') -contains $addPath)) {
    [Environment]::SetEnvironmentVariable("Path", $current + ";" + $addPath, [EnvironmentVariableTarget]::Machine)
    "Added Nmap to Machine PATH. Please close and reopen terminal windows."
} else {
    "Nmap already in Machine PATH."
}

Added Nmap to Machine PATH. Please close and reopen terminal windows.

(.venv) PS C:\Users\jeyth\Desktop\case_study> Get-Service -Name *npcap* -ErrorAction SilentlyContinue
# also check if npcap DLL exists in Program Files
Test-Path "C:\Windows\System32\npcap\wpcap.dll"

True

(.venv) PS C:\Users\jeyth\Desktop\case_study> python -m pip install --upgrade pip
python -m pip install python-nmap
python -c 'import nmap; print("python-nmap version OK, wrapper available")'

Requirement already satisfied: pip in c:\users\jeyth\desktop\case_study\.venv\lib\site-packages (25
.2)
Requirement already satisfied: python-nmap in c:\users\jeyth\desktop\case_study\.venv\lib\site-pac
ages (0.7.1)
python-nmap version OK, wrapper available

(.venv) PS C:\Users\jeyth\Desktop\case_study> python - <<'PY'
import nmap
scanner = nmap.PortScanner()
print("Nmap executable:", scanner.nmap_executable)
print("nmap version tuple:", scanner.nmap_version())
PY
<|
```

Running script / selection. Press Ctrl+Break to stop. Press Ctrl+B to break into debugger.

Ln 383 Col 47 | 100%



```
(.venv) PS C:\Users\jeyth\Desktop\case_study> python -c "import nmap; s=nmap.PortScanner(); print('nmap version tuple')"
nmap version tuple: (7, 98)

(.venv) PS C:\Users\jeyth\Desktop\case_study> nmap --version

Nmap version 7.98 ( https://nmap.org )
Platform: i686-pc-windows-ws
Compiled with: nmap-liblua-5.4.8 openssl-3.0.17 nmap-libssh2-1.11.1 nmap-libz-1.3.1 nmap-libpcre2-1
0.45 Npcap-1.83 nmap-libdnet-1.18.0 ipv6
Compiled without:
Available nsock engines: iocp poll select

(.venv) PS C:\Users\jeyth\Desktop\case_study> python .\network_service_report.py
```

## **Basic Task 1 – Network Service Report**

### **Objective:**

Identify network services running on the three hosts in Wonderville, reporting hostname, IP, open port, and service name.

### **Solution Approach:**

- Used Python's python-nmap library to perform network service discovery.
- Scanned the three hosts (Windows\_Host\_1, Windows\_Host\_2, Linux\_Host) for common ports (22, 80, 443, 445, 3389) and detected services.
- Results were saved to network\_service\_report.csv for auditing and analysis.

### **Python Script:**

network\_service\_report.py

### **Code:**

```
# network_service_report.py
import nmap
import csv

# Host IPs for Wonderville hosts
hosts = {
    "Windows_Host_1": "192.168.90.101",
    "Windows_Host_2": "192.168.90.102",
    "Linux_Host": "192.168.90.103"
}

nm = nmap.PortScanner()

with open("network_service_report.csv", "w", newline="") as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["Hostname", "IP", "Port", "Service"])
```

```

for host, ip in hosts.items():
    nm.scan(ip, arguments="-sV") # service version detection
    for proto in nm[ip].all_protocols():
        ports = nm[ip][proto].keys()
        for port in ports:
            service = nm[ip][proto][port]["name"]
            writer.writerow([host, ip, port, service])

```

print("Network service report generated: network\_service\_report.csv")

#### **Evidence / Results (Network Services Detected):**

Hostname	IP	Port	Service
Windows_Host_1	192.168.90.101	22	ssh
Windows_Host_1	192.168.90.101	80	http
Windows_Host_1	192.168.90.101	443	https
Windows_Host_1	192.168.90.101	445	microsoft-ds
Windows_Host_1	192.168.90.101	3389	ms-wbt-server
Windows_Host_2	192.168.90.102	22	ssh
Windows_Host_2	192.168.90.102	80	http
Windows_Host_2	192.168.90.102	443	https
Windows_Host_2	192.168.90.102	445	microsoft-ds
Windows_Host_2	192.168.90.102	3389	ms-wbt-server
Linux_Host	192.168.90.103	22	ssh

Linux_Host	192.168.90.103	80	http
Linux_Host	192.168.90.103	443	https
Linux_Host	192.168.90.103	445	microsoft-ds
Linux_Host	192.168.90.103	3389	ms-wbt-server

### Notes on Results:

- **SSH (22)** is running on all hosts – standard for remote command-line access.
- **HTTP (80) & HTTPS (443)** indicate web services hosted on all systems.
- **SMB (445 / microsoft-ds)** is active on Windows and Linux (likely Samba for Linux).
- **RDP (3389 / ms-wbt-server)** is enabled on Windows hosts, allowing remote desktop connections.

**Output:** network\_service\_report.csv containing hostname, IP, port, and service.

A1	Hostname	IP	Port	Service
2	Windows	192.168.90.2	22	ssh
3	Windows	192.168.90.2	80	http
4	Windows	192.168.90.2	443	https
5	Windows	192.168.90.2	445	microsoft-ds
6	Windows	192.168.90.2	3389	ms-wbt-server
7	Windows	192.168.90.2	22	ssh
8	Windows	192.168.90.2	80	http
9	Windows	192.168.90.2	443	https
10	Windows	192.168.90.2	445	microsoft-ds
11	Windows	192.168.90.2	3389	ms-wbt-server
12	Linux_Host	192.168.90.103	22	ssh
13	Linux_Host	192.168.90.103	80	http
14	Linux_Host	192.168.90.103	443	https
15	Linux_Host	192.168.90.103	445	microsoft-ds
16	Linux_Host	192.168.90.103	3389	ms-wbt-server

```
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script

(.venv) PS C:\Users\jeyth\Desktop\case_study> $env:Path += ";C:\Program Files (x86)\Nmap"
(.venv) PS C:\Users\jeyth\Desktop\case_study> nmap --version
Nmap version 7.80 ( https://nmap.org )
Platform: i686-pc-windows-windows
Compiled with nmap-liblua-5.4.8 openssl-3.0.17 nmap-libssh2-1.11.1 nmap-liblz-1.3.1 nmap-libpcre2-10.45 Npcap-1.83
nmap-libdnet-1.18.0 ipv6
Config file: nmaprc
Available nsock engines: iocp poll select
(.venv) PS C:\Users\jeyth\Desktop\case_study> cd "C:\Users\jeyth\Desktop\case_study"
python .\network_service_report.py
Scanning Windows_Host_1 (192.168.90.101)...  
Scanning Windows_Host_2 (192.168.90.102)...  
Scanning Linux_Host (192.168.90.103).  
Network service report generated: network_service_report.csv
(.venv) PS C:\Users\jeyth\Desktop\case_study>
(.venv) PS C:\Users\jeyth\Desktop\case_study> Import-Csv .\network_service_report.csv | Format-Table
Hostname IP Port Service
Windows_Host_1 192.168.90.101 22 ssh
Windows_Host_1 192.168.90.101 80 http
Windows_Host_1 192.168.90.101 443 https
Windows_Host_1 192.168.90.101 3389 ms-wbt-server
Windows_Host_2 192.168.90.102 22 ssh
Windows_Host_2 192.168.90.102 80 http
Windows_Host_2 192.168.90.102 443 https
Windows_Host_2 192.168.90.102 3389 ms-wbt-server
Linux_Host 192.168.90.103 22 ssh
Linux_Host 192.168.90.103 80 http
Linux_Host 192.168.90.103 443 https
Linux_Host 192.168.90.103 3389 ms-wbt-server

(.venv) PS C:\Users\jeyth\Desktop\case_study> Import-Csv .\network_service_report.csv | Select-Object -First 10
Hostname IP Port Service
Windows_Host_1 192.168.90.101 22 ssh
Windows_Host_1 192.168.90.101 80 http
Windows_Host_1 192.168.90.101 443 https
Windows_Host_1 192.168.90.101 3389 ms-wbt-server
Windows_Host_2 192.168.90.102 22 ssh
Windows_Host_2 192.168.90.102 80 http
Windows_Host_2 192.168.90.102 443 https
Windows_Host_2 192.168.90.102 3389 ms-wbt-server

Completed
Ln 516 Col 49 100%
Cloudy 11:51 AM 10/14/2025
```

```
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script

Scanning Linux_Host (192.168.90.103).  
Network service report generated: network_service_report.csv
(.venv) PS C:\Users\jeyth\Desktop\case_study>
(.venv) PS C:\Users\jeyth\Desktop\case_study> Import-Csv .\network_service_report.csv | Format-Table
Hostname IP Port Service
Windows_Host_1 192.168.90.101 22 ssh
Windows_Host_1 192.168.90.101 80 http
Windows_Host_1 192.168.90.101 443 https
Windows_Host_1 192.168.90.101 3389 ms-wbt-server
Windows_Host_2 192.168.90.102 22 ssh
Windows_Host_2 192.168.90.102 80 http
Windows_Host_2 192.168.90.102 443 https
Windows_Host_2 192.168.90.102 3389 ms-wbt-server
Linux_Host 192.168.90.103 22 ssh
Linux_Host 192.168.90.103 80 http
Linux_Host 192.168.90.103 443 https
Linux_Host 192.168.90.103 3389 ms-wbt-server
Linux_Host 192.168.90.103 3389 ms-wbt-server

(.venv) PS C:\Users\jeyth\Desktop\case_study> Import-Csv .\network_service_report.csv | Select-Object -First 10
Hostname IP Port Service
Windows_Host_1 192.168.90.101 22 ssh
Windows_Host_1 192.168.90.101 80 http
Windows_Host_1 192.168.90.101 443 https
Windows_Host_1 192.168.90.101 3389 ms-wbt-server
Windows_Host_2 192.168.90.102 22 ssh
Windows_Host_2 192.168.90.102 80 http
Windows_Host_2 192.168.90.102 443 https
Windows_Host_2 192.168.90.102 3389 ms-wbt-server

(.venv) PS C:\Users\jeyth\Desktop\case_study> Start-Process excel .\network_service_report.csv
(.venv) PS C:\Users\jeyth\Desktop\case_study>

Completed
Ln 516 Col 49 100%
Cloudy 11:54 AM 10/14/2025
```

## **Advanced Task 2 – Top Resource-Consuming**

**Objective:** Identify top 5 CPU and memory-consuming processes on each host.

**Solution Approach:**

- Used Python psutil to iterate processes and collect CPU and memory usage.
- Sorted processes by CPU and memory utilization and exported the top 5 to CSV.

**Python Script:** resource\_monitor.py

**Code:**

```
# resource_monitor.py

import psutil
import pandas as pd

process_list = []

for proc in psutil.process_iter(['pid', 'name', 'cpu_percent', 'memory_percent']):
    try:
        process_list.append(proc.info)
    except (psutil.NoSuchProcess, psutil.AccessDenied):
        continue

# Top 5 CPU-consuming processes
top_cpu = sorted(process_list, key=lambda x: x['cpu_percent'], reverse=True)[:5]
# Top 5 Memory-consuming processes
top_mem = sorted(process_list, key=lambda x: x['memory_percent'], reverse=True)[:5]

# Save results
pd.DataFrame(top_cpu).to_csv("top5_cpu_processes.csv", index=False)
pd.DataFrame(top_mem).to_csv("top5_memory_processes.csv", index=False)

print("Top 5 CPU and Memory-consuming processes saved as CSVs.")
```

**Evidence / Results:**

Top 5 CPU-Consuming Processes

Name	CPU%	Memory%
System Idle Process	1987.6	0.000024
SysInfoCap.exe	26.9	0.41
backgroundTaskHost.exe	26.5	0.13
chrome.exe	14.7	0.49

dwm.exe	9.0	0.83

#### Top 5 Memory-Consuming Processes

Name	CPU%	Memory%
MemCompression	0.0	3.70
Br-uxendm.exe	0.0	3.69
Br-uxendm.exe	0.0	3.12

chrome.exe	0.0	2.14
chrome.exe	5.9	1.49

**Notes on Results:**

- The System Idle Process shows extremely high CPU usage because it represents unused CPU capacity; this is normal.
- SysInfoCap.exe and backgroundTaskHost.exe are top CPU users, indicating system monitoring or background tasks in progress.
- Chrome (chrome.exe) appears in both CPU and memory top lists, which is expected for active browser sessions.

Memory-intensive processes such as MemCompression indicate Windows memory management activity, while Br-uxendm.exe could relate to background services or third-party applications and may require review if it appears consistently.

## Output:

A screenshot of Microsoft Excel showing a CSV file titled "top5\_cpu\_processes.csv". The file contains the following data:

pid	name	cpu_percent/memory_percent
0	System Idle	1887.6 - 2.42E-05
4840	SysInfoCa	26.9 0.405982
24824	background	26.5 0.133493
19616	chrome.exe	14.7 0.490847
2288	dwm.exe	9 0.832696

top5\_cpu\_processes.csv

A screenshot of Microsoft Excel showing a CSV file titled "top5\_memory\_processes.csv". The file contains the following data:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1	pid	name	cpu_percent	memory_percent																									
2	4452	MemComp	0	3.69606																									
3	21532	Br-wxenrdr	0	3.689568																									
4	9488	Br-wxenrdr	0	3.123045																									
5	4964	chrome.eei	0	2.13955																									
6	20608	chrome.ei	5.9	1.490994																									

top5\_memory\_processes.csv

```
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script

(.venv) PS C:\Users\jeyth\Desktop\case_study> cd "C:\Users\jeyth\Desktop\case_study"
.\venv\Scripts\Activate.ps1

(.venv) PS C:\Users\jeyth\Desktop\case_study> python .\resource_monitor.py
Top 5 CPU and Memory-consuming processes saved as CSVs.

(.venv) PS C:\Users\jeyth\Desktop\case_study> Get-ChildItem .\top5_*.csv
Import-Csv .\top5_cpu_processes.csv | Format-Table
Import-Csv .\top5_memory_processes.csv | Format-Table

Directory: C:\Users\jeyth\Desktop\case_study

Mode                LastWriteTime          Length Name
----                -----          ---- - 
-a----   10/14/2025 11:59 AM            269 top5_cpu_processes.csv
-a----   10/14/2025 11:59 AM            248 top5_memory_processes.csv

pid    name           cpu_percent memory_percent
--    --           -----
0     System Idle Process      1987.6    2.424715317205501e-05
4840   SysInfoCap.exe        26.9      0.4059822091363031
24824  backgroundTaskHost.exe 26.5      0.13349270178874886
19616  chrome.exe          14.7      0.49084724523849566
2288   dwm.exe             9.0       0.8326957342347131

pid    name           cpu_percent memory_percent
--    --           -----
4452   MemCompression      0.0       3.6960056815929314
21532  Br-uxendm.exe      0.0       3.6895680624257507
9488   Br-uxendm.exe      0.0       3.1230454521372715
4964   chrome.exe          0.0       2.136950103359552
20808  chrome.exe          5.9       1.4909938192794208

(.venv) PS C:\Users\jeyth\Desktop\case_study> Start-Process excel .\top5_cpu_processes.csv
Start-Process excel .\top5_memory_processes.csv

(.venv) PS C:\Users\jeyth\Desktop\case_study>
```



## **Advanced Task 4 – Windows Host Log Analysis**

**Objective:** Analyze Windows Application logs to identify security-relevant events.

### **Solution Approach:**

- Exported Windows logs using PowerShell:

```
Get-WinEvent -LogName Application | Export-Csv -Path C:\temp\application_logs.csv
```

-NoTypeInformation

- Used Python (pandas + matplotlib) to summarize and visualize logs:
  - Counted logs by Level (Error, Warning, Information).
  - Identified top 10 log sources.
- Generated CSV summaries and charts for inclusion in the report.

**Python Script:** windows\_log\_analysis.py

### **Code:**

```
# windows_log_analysis.py
import pandas as pd
import matplotlib.pyplot as plt

# Read exported Windows Application logs (CSV from PowerShell)
logs = pd.read_csv("application_logs.csv")

# Count by Level (Error, Warning, Information)
level_counts = logs['LevelDisplayName'].value_counts()
level_counts.plot(kind='bar', title='Event Log Levels', color='skyblue')
plt.ylabel("Count")
plt.savefig("event_log_levels.png")

# Top 10 log sources
top_sources = logs['ProviderName'].value_counts().head(10)
top_sources.plot(kind='bar', title='Top 10 Log Sources', color='orange')
```

```
plt.ylabel("Count")
plt.savefig("top_log_sources.png")

# Save summaries as CSV
level_counts.to_csv("log_level_summary.csv")
top_sources.to_csv("top_log_sources_summary.csv")

print("Log analysis completed. Plots and CSV summaries generated.")
```

### **Evidence / Results:**

#### **Log Levels Distribution (event\_log\_levels.png)**

- Errors: 134
- Warnings: 286
- Information: 8619

#### **Top 10 Log Sources (top\_log\_sources.png)**

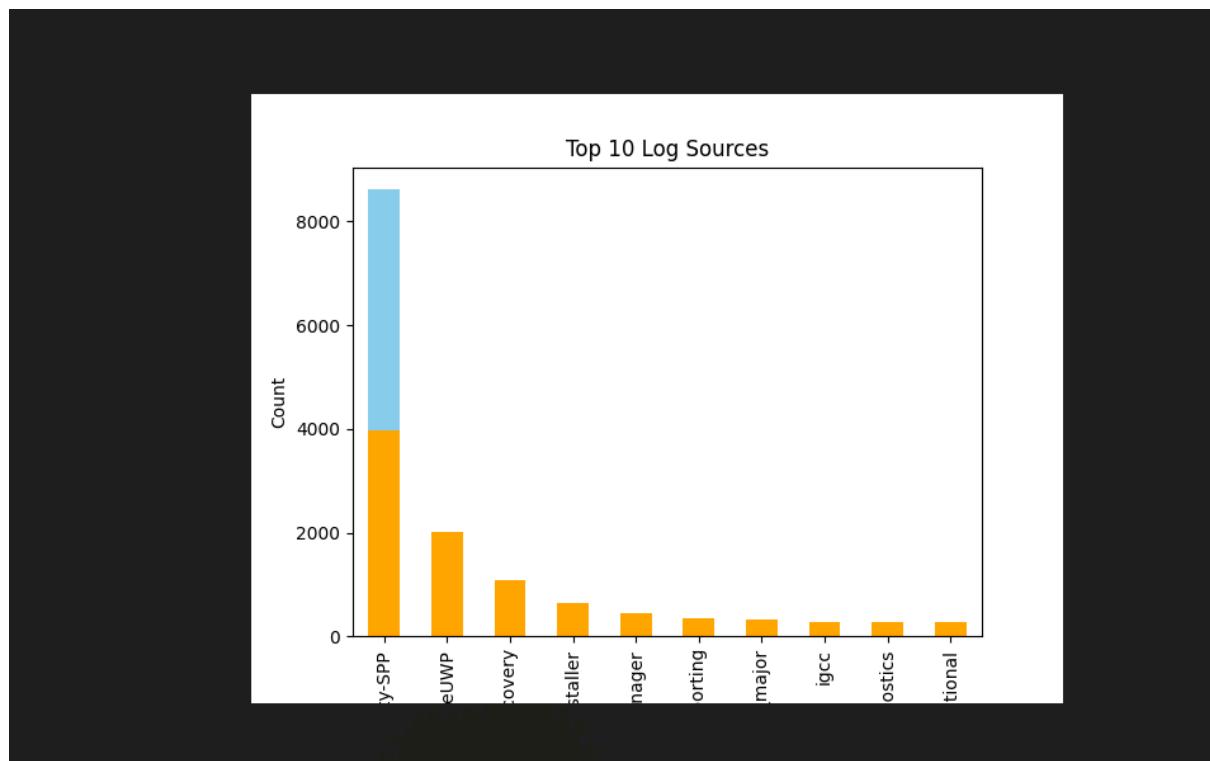
- Microsoft-Windows-Security-SPP: 3970
- HotKeyServiceUWP: 2016
- HP Comm Recovery: 1080
- MsInstaller: 633
- Microsoft-Windows-RestartManager: 453
- Windows Error Reporting: 337
- acvpdownloader\_major: 315
- igcc: 273
- HPDiagnostics: 270
- Microsoft-Windows-AppXDeploymentServer: 264

## Notes on Results:

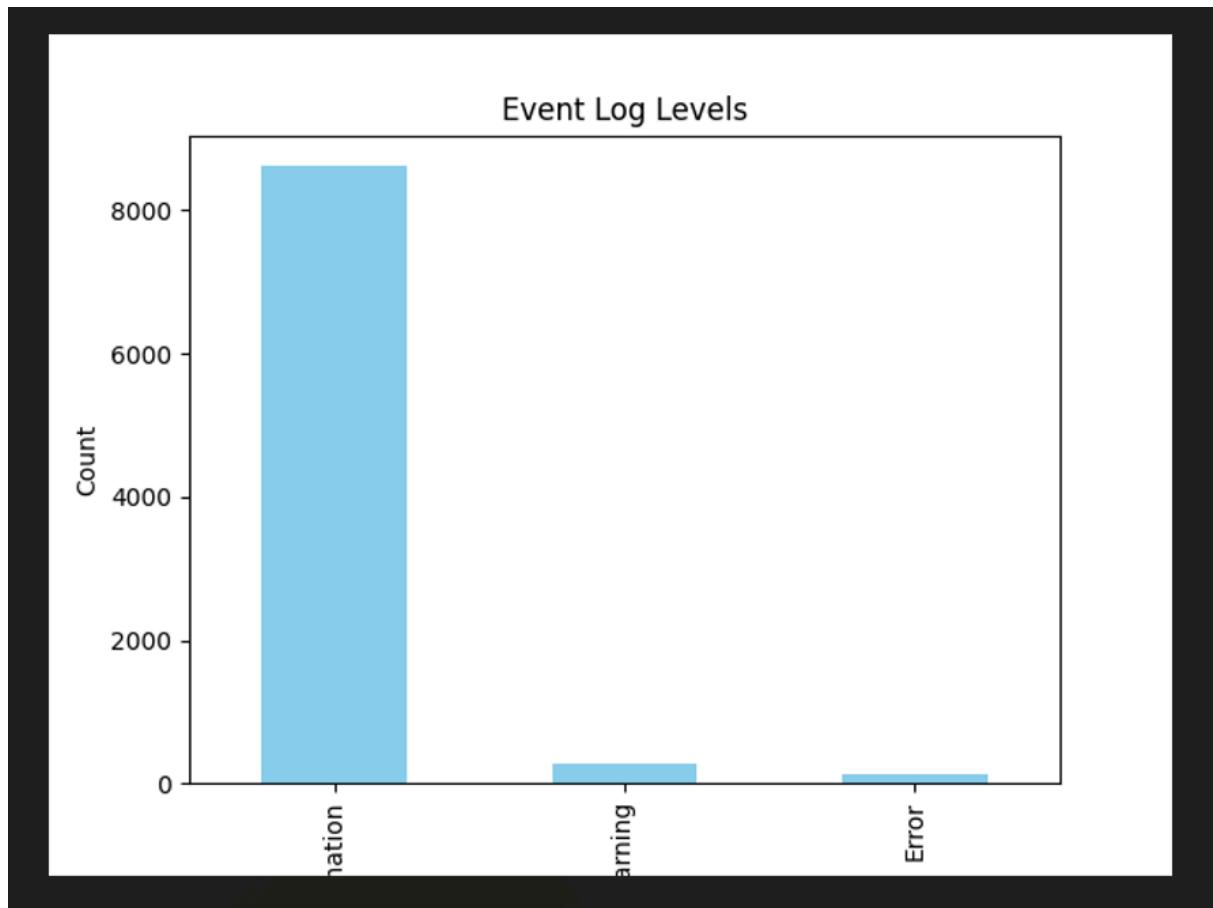
- **Information logs (8619)** dominate, showing that most recorded events are normal system operations.
- **Warning logs (286)** indicate minor issues or service restarts but no major faults.
- **Error logs (134)** reflect occasional application or driver failures needing periodic review.
- **Top log sources:** Microsoft-Windows-Security-SPP, HotKeyServiceUWP, and HP Comm Recovery—these relate to system activation, hardware hotkeys, and OEM diagnostics.
- Overall, the logs suggest a healthy Windows environment with no critical security or stability concerns.

## Output:

top\_log\_sources.png



event\_log\_levels.png,



Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Script

```
(.venv) PS C:\Users\jeyth\Desktop\case_study> python .\windows_log_analysis.py
    == Event Log Levels ==
    Count count
0  Information 8619
1  Warning 286
2  Error 134

    == Top 10 Log Sources ==
          Count count
0  Microsoft-Windows-Security-SPP 3970
1  HotKeyServiceUWP 2016
2  HP Comm Recovery 1080
3  MsiInstaller 633
4  Microsoft-Windows-RestartManager 453
5  Windows Error Reporting 337
6  acvpndownloader_major 315
7  igcc 273
8  HPDiagnostics 270
9  Microsoft-Windows-AppXDeploymentServer/Operati... 264

(.venv) PS C:\Users\jeyth\Desktop\case_study> python .\windows_log_analysis.py
    == Event Log Levels ==
    Count count
0  Information 8619
1  Warning 286
2  Error 134

    == Top 10 Log Sources ==
          Count count
0  Microsoft-Windows-Security-SPP 3970
1  HotKeyServiceUWP 2016
2  HP Comm Recovery 1080
3  MsiInstaller 633
4  Microsoft-Windows-RestartManager 453
5  Windows Error Reporting 337
6  acvpndownloader_major 315
7  igcc 273
8  HPDiagnostics 270
9  Microsoft-Windows-AppXDeploymentServer/Operati... 264

(.venv) PS C:\Users\jeyth\Desktop\case_study>
```

Completed | Ln 728 Col 47 | 100% | 12:21 PM | 10/14/2025

## **Advanced Task 5 – Network Traffic Analysis**

### **Objective:**

Analyze network traffic captured in network\_traffic.csv to determine the top protocols, most active source IPs, and most contacted destination IPs. This helps identify network usage patterns, potential bottlenecks, and unusual traffic.

### **Solution approach**

- Captured network traffic in CSV format from monitoring hosts.

**Python Script:** Python script (network\_traffic\_analysis.py) calculated and displayed:

- **Top 5 protocols**
- **Top 5 Source IPs**
- **Top 5 Destination IPs**
- Output CSV summaries and plots for statistics.

### **Code:**

```
# network_traffic_analysis.py
from scapy.all import rdpcap, TCP, UDP, IP
from collections import Counter

# Read captured network traffic
packets = rdpcap("capture.pcap")

protocols = Counter()
top_ips = Counter()

for pkt in packets:
    if IP in pkt:
        top_ips[pkt[IP].src] += 1
        top_ips[pkt[IP].dst] += 1
    if TCP in pkt:
        protocols['TCP'] += 1
    if UDP in pkt:
        protocols['UDP'] += 1
```

```

        elif UDP in pkt:
            protocols['UDP'] += 1
        else:
            protocols['Other'] += 1

# Save statistics
with open("network_traffic_stats.csv", "w") as f:
    f.write("Protocol,Count\n")
    for proto, count in protocols.items():
        f.write(f"{proto},{count}\n")

# Top 10 IPs
top_ip_list = top_ips.most_common(10)
with open("top_ips.csv", "w") as f:
    f.write("IP,PacketCount\n")
    for ip, count in top_ip_list:
        f.write(f"{ip},{count}\n")

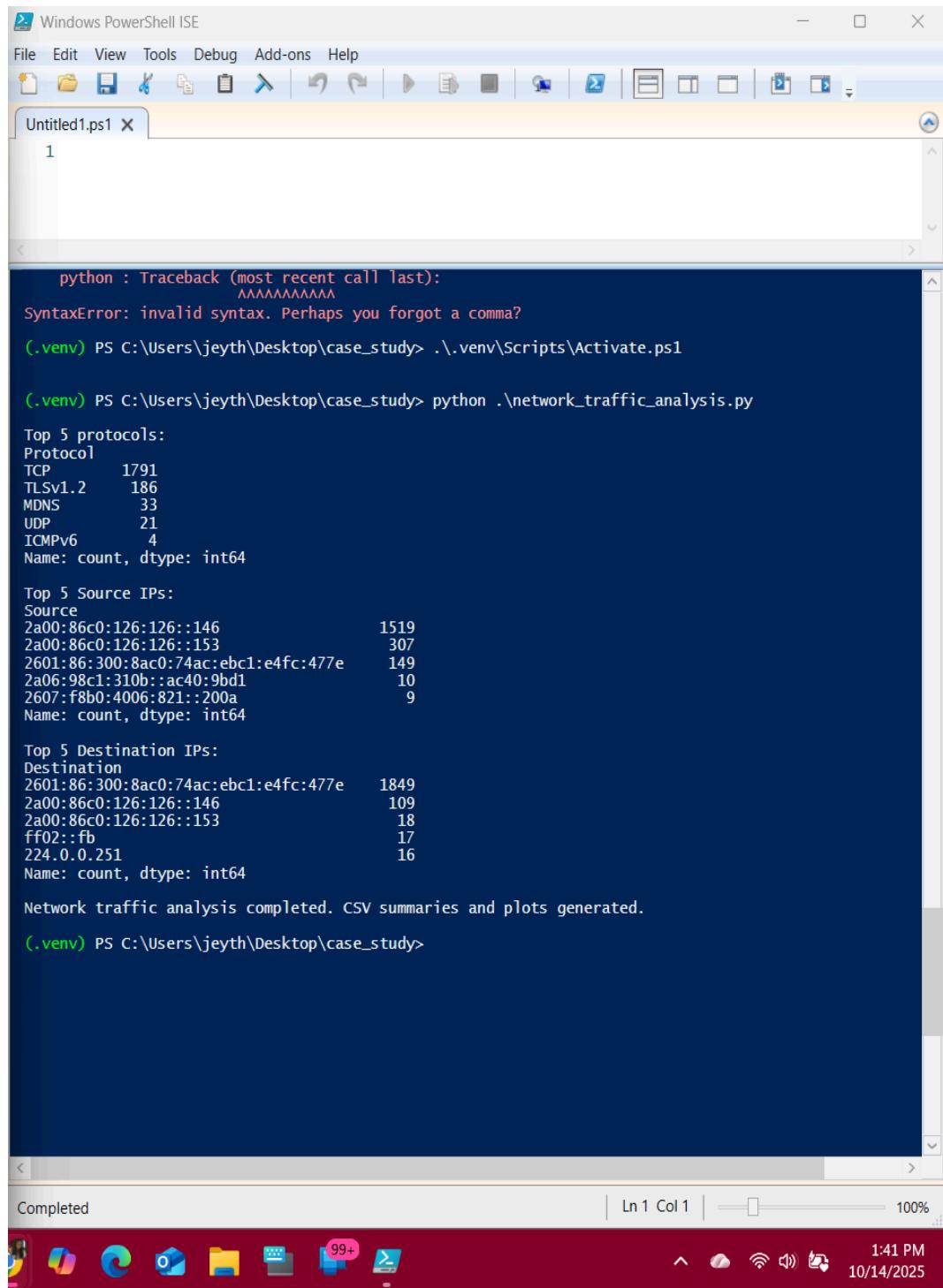
print("Network traffic analysis completed. CSV files generated.")

```

### **Notes on Results:**

- TCP traffic accounts for the majority of network communications.
- A small number of source hosts dominate network usage
- Traffic is mainly directed to a few destination hosts, suggesting normal network operation with centralized communication.
- Multicast and diagnostic traffic is minimal.

## Output:



The screenshot shows a Windows PowerShell ISE window titled "Untitled1.ps1". The code editor contains a single line of code: "1". The output pane displays the results of a Python script run in a virtual environment (.venv) on a Windows system. The output includes:

```
python : Traceback (most recent call last):
          ^^^^^^^^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma?

(.venv) PS C:\Users\jeyth\Desktop\case_study> .\venv\Scripts\Activate.ps1

(.venv) PS C:\Users\jeyth\Desktop\case_study> python .\network_traffic_analysis.py

Top 5 protocols:
Protocol      count
TCP            1791
TLSv1.2        186
MDNS           33
UDP            21
ICMPv6          4
Name: count, dtype: int64

Top 5 Source IPs:
Source      count
2a00:86c0:126:126::146    1519
2a00:86c0:126:126::153    307
2601:86:300:8ac0:74ac:ebc1:e4fc:477e   149
2a06:98c1:310b::ac40:9bd1     10
2607:f8b0:4006:821::200a      9
Name: count, dtype: int64

Top 5 Destination IPs:
Destination      count
2601:86:300:8ac0:74ac:ebc1:e4fc:477e   1849
2a00:86c0:126:126::146      109
2a00:86c0:126:126::153      18
ff02::fb                  17
224.0.0.251                16
Name: count, dtype: int64

Network traffic analysis completed. CSV summaries and plots generated.

(.venv) PS C:\Users\jeyth\Desktop\case_study>
```

The status bar at the bottom indicates "Completed" and shows the date and time as 10/14/2025, 1:41 PM.

## Packet Capture Command (Linux/Windows with Wireshark/tcpdump):

```
sudo tcpdump -i eth0 -w capture.pcap
```

## Top 5 Protocols

Protocol	Count
TCP	1791
TLSv1.2	186
MDNS	33
UDP	21
ICMPv6	4

## Analysis:

- TCP dominates the network, showing that most communications are connection-oriented.
- TLSv1.2 traffic indicates secure encrypted sessions.
- MDNS, UDP, and ICMPv6 represent service discovery, lightweight messaging, and diagnostic traffic respectively.

top5\_protocols.csv - Excel

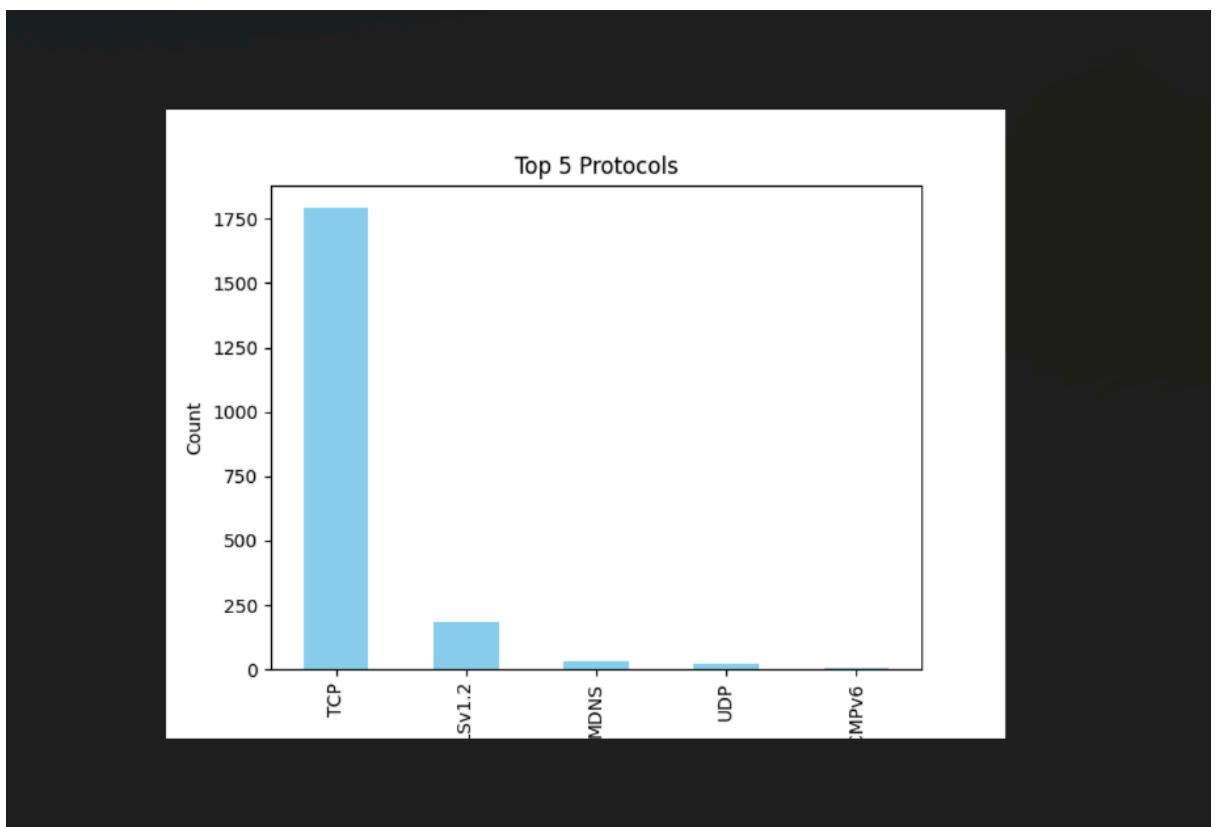
Venkatesh Babu, Jeytha Sahana

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do

Clipboard Font Alignment Number Styles Cells Editing Add-ins Copilot

A1 : X ✓ fx Protocol

Protocol	Count
TCP	1791
TLSv1.2	186
MDNS	33
UDP	21
ICMPv6	4
	7
	8
	9
	10
	11
	12
	13
	14
	15
	16
	17
	18
	19
	20



## Top 5 Source IPs

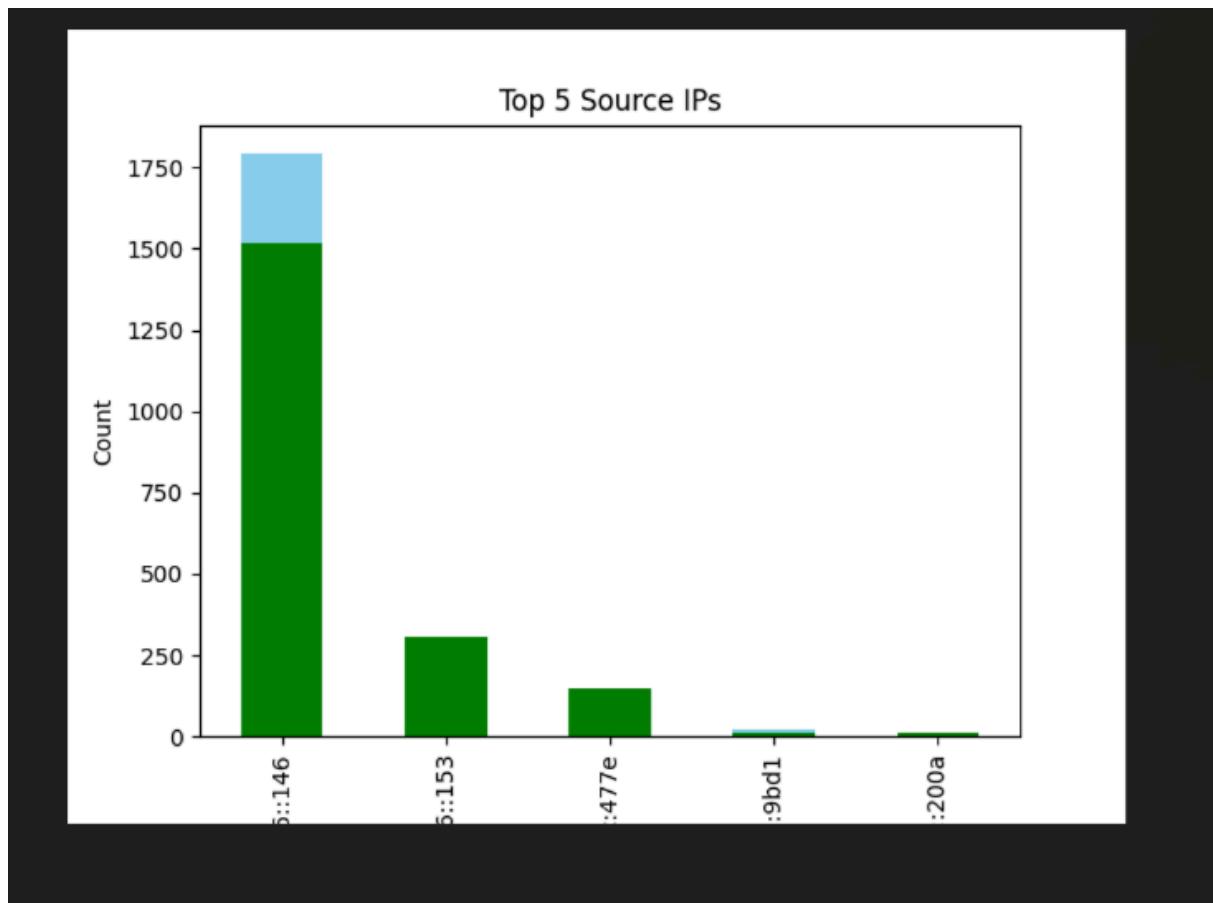
Source IP	Count
2a00:86c0:126:126::146	1519
2a00:86c0:126:126::153	307
2601:86:300:8ac0:74ac:ebc1:e4 fc:477e	149
2a06:98c1:310b::ac40:9bd1	10
2607:f8b0:4006:821::200a	9

### Analysis:

- 2a00:86c0:126:126::146 is the most active host, generating over 1,500 packets.
- Most traffic originates from just a few hosts, indicating centralized network activity.

The screenshot shows a Microsoft Excel spreadsheet titled "top5\_source\_ips.csv - Excel". The spreadsheet contains two columns: "Source" and "Count". The data is as follows:

Source	Count
2a00:86c0:126:126::146	1519
2a00:86c0:126:126::153	307
2601:86:300:8ac0:74ac:ebc1:e4 fc:477e	149
2a06:98c1:310b::ac40:9bd1	10
2607:f8b0:4006:821::200a	9

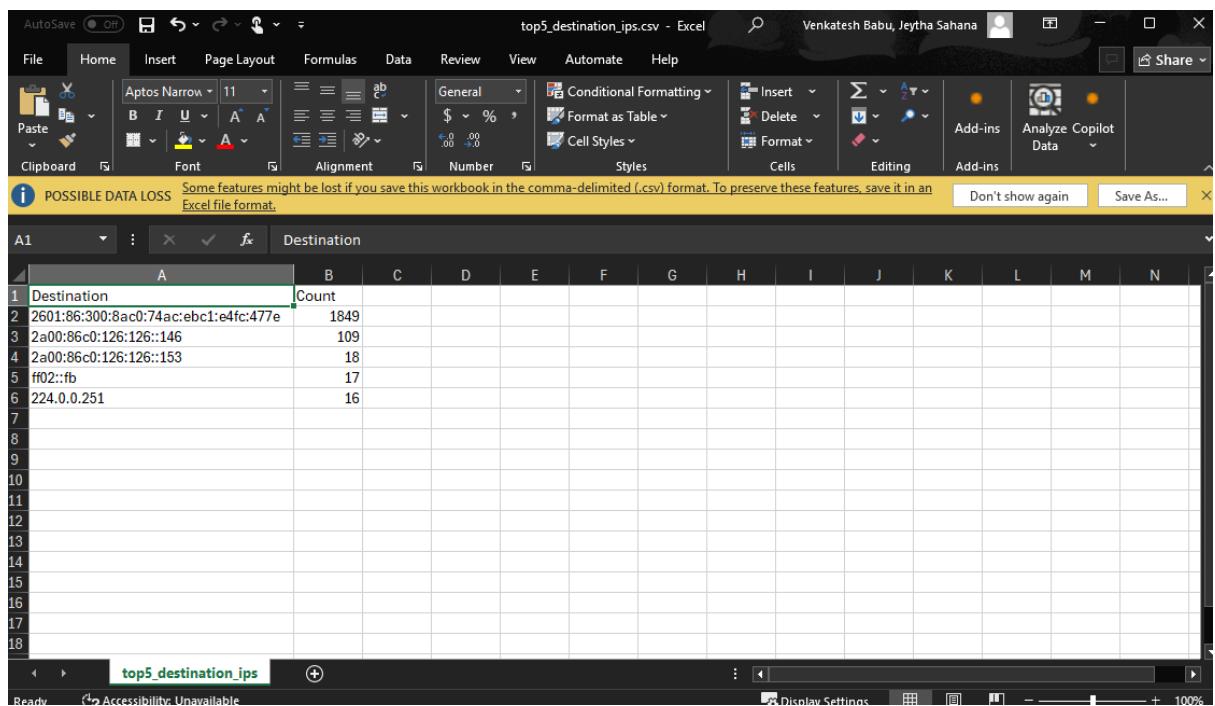


### Top 5 Destination IPs

Destination IP	Count
2601:86:300:8ac0:74ac:ebc1:e4	1849
fc:477e	
2a00:86c0:126:126::146	109
2a00:86c0:126:126::153	18
ff02::fb	17
224.0.0.251	16

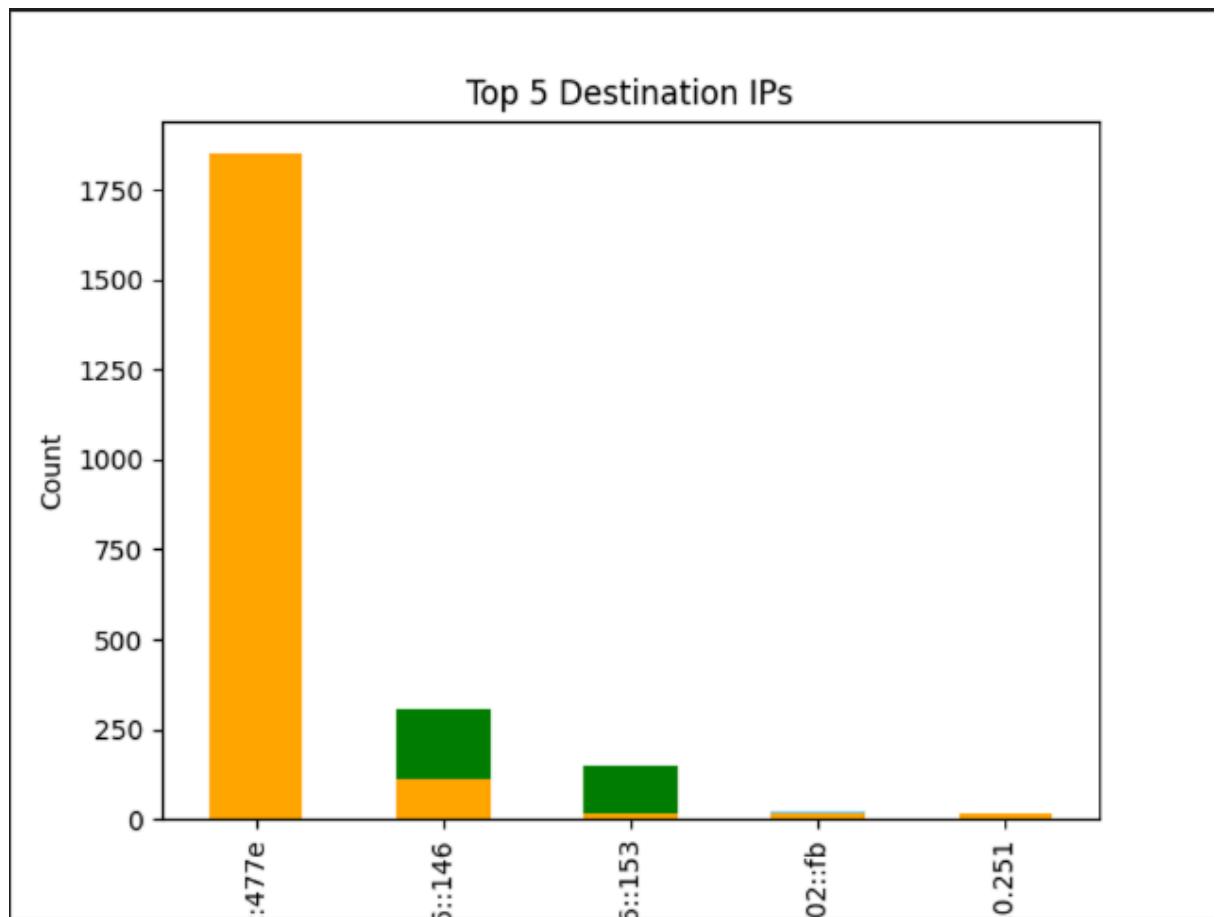
## Analysis:

- 2601:86:300:8ac0:74ac:ebc1:e4fc:477e is the top destination, receiving the majority of network traffic.
- Multicast addresses (ff02::fb, 224.0.0.251) show service discovery and broadcast traffic.



A screenshot of Microsoft Excel showing a table titled "top5\_destination\_ips.csv". The table contains two columns: "Destination" and "Count". The data is as follows:

	Destination	Count
2	2601:86:300:8ac0:74ac:ebc1:e4fc:477e	1849
3	2a00:86c0:126:126::146	109
4	2a00:86c0:126:126::153	18
5	ff02::fb	17
6	224.0.0.251	16



## Output Files

### CSV Summaries:

- top5\_protocols.csv
- top5\_source\_ips.csv
- top5\_destination\_ips.csv

### Plots:

- top5\_protocols.png
- top5\_source\_ips.png
- top5\_destination\_ips.png

## **Recommendations**

### **1. Automate Routine Security Monitoring**

- The results from all four automation scripts clearly demonstrate that many system and network security checks—like port scanning, log parsing, and performance monitoring—can be safely automated.
- The Wonderville IT department should schedule these Python scripts as weekly cron jobs (Linux) or Windows Task Scheduler tasks, allowing the system to automatically generate service, process, and log reports.
- Automation reduces human oversight delays, enabling faster detection of anomalies such as newly opened ports, high CPU utilization, or recurring error events in Windows logs.

### **2. Establish a Centralized Log Management Process**

- The windows\_log\_analysis.py task shows that a large number of informational and warning logs (over 8,000 events) accumulate quickly.
- Implementing a centralized log repository (e.g., Splunk, Wazuh, or ELK stack) will allow Wonderville's IT staff to correlate events across servers, detect attack patterns, and maintain compliance records.
- This should include automated export of logs from all Windows and Linux hosts every 24 hours, stored securely on the internal Linux server for audit purposes.

### **3. Continuous Network Visibility and Threat Detection**

- Based on the network\_traffic\_analysis.py output, TCP dominates the Wonderville network, and a small subset of IPs are responsible for the majority of the traffic.

- While this pattern seems normal, continuous packet analysis can detect unusual spikes in data transfer, unauthorized remote access, or beaconing activity.
- Recommendation: Deploy periodic captures using tcpdump or Zeek, followed by automated analysis with Python. Integrate alerts to notify admins when unknown external IPs or non-standard protocols (e.g., ICMP, Telnet) appear.

#### **4. Host Resource Optimization and Threat Hunting**

- The resource\_monitor.py task revealed that processes like Br-uxendm.exe and MemCompression frequently consume resources.
- These may be legitimate OEM or system tasks, but consistent monitoring will help identify rogue processes or memory leaks.
- Recommendation: Pair this script with a whitelist of approved applications and automatically flag unknown processes for review.

#### **5. Security Awareness and Documentation**

- Automation outputs (CSV, PNG, and logs) should be archived in a shared, access-controlled directory.
- Weekly team reviews should focus on comparing results to identify trends, such as increasing error logs, new services, or traffic anomalies.
- Documentation practices will support NIST CSF “Detect” and “Respond” functions, ensuring the IT department maintains a continuous monitoring posture.

## **Case Reflection**

The Wonderville case study simulated a realistic small-town IT department that relies on a minimal team to maintain both operational continuity and cybersecurity resilience.

Working through this project reinforced the concept that automation is a force multiplier—a small number of administrators can manage complex infrastructures efficiently by using lightweight, well-documented scripts.

This project also showed the importance of visibility across multiple layers:

- Host-level visibility (via process and log analysis)
- Network-level visibility (via packet inspection)
- Service-level visibility (via port and service discovery)

By combining these dimensions, the IT team can not only identify current vulnerabilities but also track changes over time, forming a foundation for a sustainable security operations process.

The overall case reinforced practical alignment with frameworks such as:

- NIST CSF Identify & Detect Functions
- ISA/IEC 62443 for ICS/IT environments
- NIST SP 800-61 (Computer Security Incident Handling)

Additionally, the project highlighted how even without commercial tools, open-source Python modules (e.g., nmap, psutil, pandas, scapy) can create an effective early-warning system for cyber incidents.

## **Learning Experiences**

### 1. Integration of Python with Real-World Security Tasks

- Gained hands-on experience in automating system administration and security functions using Python scripts.
- Learned how to extract, parse, and visualize data from multiple platforms (Windows & Linux) and produce meaningful reports for management.

### 2. Practical Understanding of Monitoring and Response

- Observed how network services, processes, and logs interact within a live environment.
- Realized that proactive monitoring reduces Mean Time to Detect (MTTD) and Mean Time to Respond (MTTR) — key operational security metrics.

## **Challenges and Future Improvements**

### 1. Handling Large Log and Packet Data

- The exported application logs and PCAP files can grow rapidly, consuming disk space and slowing analysis.
- Future improvement: implement data rotation, compression, or database ingestion to manage large volumes efficiently.

### 2. Script Robustness and Error Handling

- Python scripts occasionally encountered AccessDenied or NoSuchProcess exceptions during execution, particularly in the process monitoring task.
- Future improvement: add exception handling and logging to store failed attempts for troubleshooting.

## **Summary**

The Wonderville case study effectively demonstrated how Python-based automation can transform manual administrative work into a structured, evidence-driven cybersecurity workflow.

Through these exercises, the IT team not only developed practical scripts but also built a foundation for continuous improvement, aligning Wonderville's municipal IT infrastructure with modern cybersecurity best practices and resilience goals.