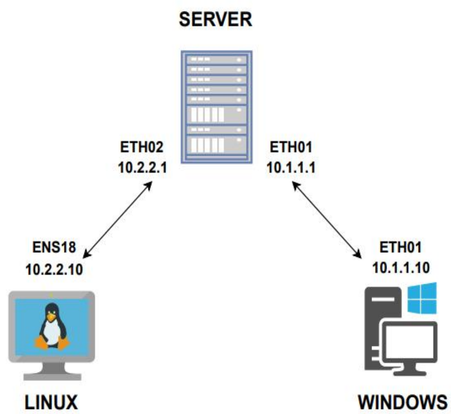
Title : Project Security Automation

NAME OF TASK DONE :

- Identified changes in active services/processes
- Identified active services/processes
- Developed a Python program for host log analysis
- Implemented Windows notifications

DATE : 12/07/2023

Basic task 1:



Basic task 2:

```
Starting Nmap 7.94 ( https://nmap.org ) at 2023-12-07 19:48 Pacific Standard Time
NSE: Loaded 156 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 19:48
Completed NSE at 19:48, 0.00s elapsed
Initiating NSE at 19:48
Completed NSE at 19:48, 0.00s elapsed
Initiating NSE at 19:48
Completed NSE at 19:48, 0.00s elapsed
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid servers with --dns-servers
Initiating SYN Stealth Scan at 19:48
Scanning 10.1.1.1 [1000 ports]
Discovered open port 139/tcp on 10.1.1.1
Discovered open port 135/tcp on 10.1.1.1
Discovered open port 80/tcp on 10.1.1.1
Discovered open port 445/tcp on 10.1.1.1
Discovered open port 8000/tcp on 10.1.1.1
Discovered open port 8089/tcp on 10.1.1.1
Completed SYN Stealth Scan at 19:48, 0.08s elapsed (1000 total ports)
Initiating Service scan at 19:48
Scanning 6 services on 10.1.1.1
Completed Service scan at 19:49, 29.59s elapsed (6 services on 1 host)
Initiating OS detection (try #1) against 10.1.1.1
Retrying OS detection (try #2) against 10.1.1.1
Retrying OS detection (try #3) against 10.1.1.1
Retrying OS detection (try #4) against 10.1.1.1
Retrying OS detection (try #5) against 10.1.1.1
NSE: Script scanning 10.1.1.1.
Initiating NSE at 19:49
Completed NSE at 19:49, 14.13s elapsed
Initiating NSE at 19:49
Completed NSE at 19:49, 0.10s elapsed
Initiating NSE at 19:49
Completed NSE at 19:49, 0.00s elapsed
Nmap scan report for 10.1.1.1
Host is up (0.00030s latency).
Not shown: 994 closed tcp ports (reset)
```

```
PORT     STATE SERVICE      VERSION
80/tcp   open  http         Microsoft IIS httpd 10.0
|_http-title: IIS Windows Server
| http-methods:
|   Supported Methods: OPTIONS TRACE GET HEAD POST
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
135/tcp  open  msrpc        Microsoft Windows RPC
139/tcp  open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds?
8000/tcp open  http         Splunkd httpd
|_http-server-header: Splunkd
|_http-favicon: Unknown favicon MD5: E60C968E8FF3CC2F4FB869588E83AFC6
| http-robots.txt: 1 disallowed entry
|_/
| http-title: Site doesn't have a title (text/html; charset=UTF-8).
|_Requested resource was http://10.1.1.1:8000/en-US/account/login?return_to=%2Fen-US%2F
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
8089/tcp open  ssl/http     Splunkd httpd
| ssl-cert: Subject: commonName=SplunkServerDefaultCert/organizationName=SplunkUser
| Issuer: commonName=SplunkCommonCA/organizationName=Splunk/stateOrProvinceName=CA/countryName=US
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-11-02T20:26:24
| Not valid after:  2026-11-01T20:26:24
| MD5:   8672:ce2b:75dd:b7fc:2937:6055:03ca:c77a
|_SHA-1: 2774:cf3d:cd45:9ee3:34d5:9d1f:828d:86fb:3095:da16
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D
|_  Server returned status 401 but no WWW-Authenticate header.
|_http-title: Site doesn't have a title (text/xml; charset=UTF-8).
|_http-server-header: Splunkd
```

```
Starting Nmap 7.94 ( https://nmap.org ) at 2023-12-07 19:54 Pacific Standard Time
NSE: Loaded 156 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 19:54
Completed NSE at 19:54, 0.00s elapsed
Initiating NSE at 19:54
Completed NSE at 19:54, 0.00s elapsed
Initiating NSE at 19:54
Completed NSE at 19:54, 0.00s elapsed
Initiating ARP Ping Scan at 19:54
Scanning 10.2.2.10 [1 port]
Completed ARP Ping Scan at 19:54, 1.46s elapsed (1 total hosts)
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 10.2.2.10 [host down]
NSE: Script Post-scanning.
Initiating NSE at 19:54
Completed NSE at 19:54, 0.00s elapsed
Initiating NSE at 19:54
Completed NSE at 19:54, 0.00s elapsed
Initiating NSE at 19:54
Completed NSE at 19:54, 0.00s elapsed
Read data files from: C:\Program Files (x86)\Nmap
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 2.37 seconds
           Raw packets sent: 2 (56B) | Rcvd: 0 (0B)
```

```
Starting Nmap 7.94 ( https://nmap.org ) at 2023-12-07 19:55 Pacific Standard Time
NSE: Loaded 156 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 19:55
Completed NSE at 19:55, 0.00s elapsed
Initiating NSE at 19:55
Completed NSE at 19:55, 0.00s elapsed
Initiating NSE at 19:55
Completed NSE at 19:55, 0.00s elapsed
Initiating ARP Ping Scan at 19:55
Scanning 10.1.1.10 [1 port]
Completed ARP Ping Scan at 19:55, 1.43s elapsed (1 total hosts)
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 10.1.1.10 [host down]
NSE: Script Post-scanning.
Initiating NSE at 19:55
Completed NSE at 19:55, 0.00s elapsed
Initiating NSE at 19:55
Completed NSE at 19:55, 0.00s elapsed
Initiating NSE at 19:55
Completed NSE at 19:55, 0.00s elapsed
Read data files from: C:\Program Files (x86)\Nmap
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 1.97 seconds
           Raw packets sent: 2 (56B) | Rcvd: 0 (0B)
```

Techincal solution :

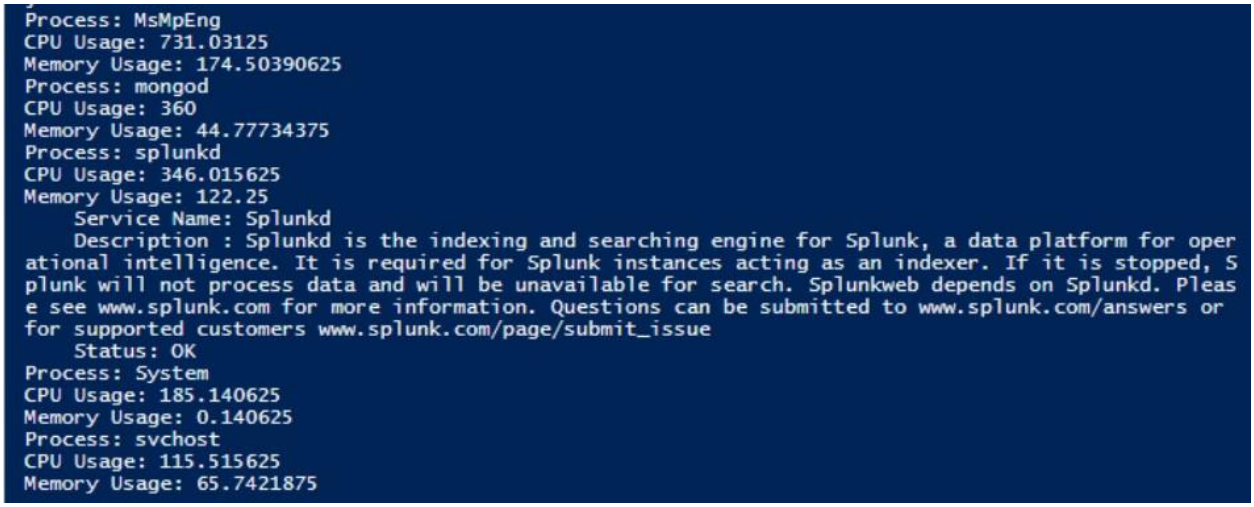
The script appears to be a PowerShell script used to get service information from a server. It uses the Get-Process cmdlet to retrieve a list of all running processes on the server. Then, it sorts the list by CPU usage in descending order and selects the first 5 processes. For each process, the script uses the Get-WmiObject cmdlet to retrieve the corresponding service object. If the service object exists, the script writes out the following information:

- Process name
- Memory usage
- CPU usage
- Service name
- Service description
- Service status

This script is helpful for troubleshooting performance problems. By identifying the processes that are using the most CPU, you can determine which services are responsible for the high CPU usage.

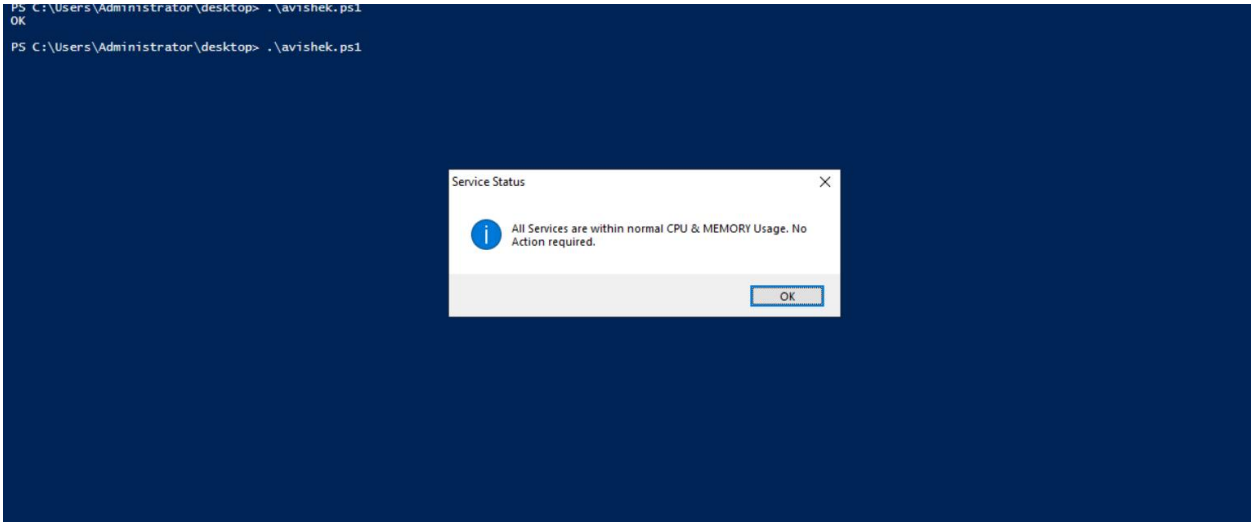
1] **A script for identifying active services/processes**

Report on server



Report on windows

2]



3]

The Python script provided utilizes the Scapy library to perform network packet sniffing on a chosen network interface. Here's what the script does:

Interface Selection: The script selects a network interface for packet sniffing. There are two lines specifying the interface, but only the second one takes effect. Please choose the appropriate line according to the network interface you wish to use.

Packet Filtering:The script captures only TCP packets. The sniff function is set up with a filter to capture packets with the TCP protocol. You can modify or remove the filter based on your specific needs.

Packet Processing: For each captured packet, the script checks if it has a TCP layer. If it does, the script extracts and prints information about the packet, including source and destination IP addresses, source and destination port numbers, and the protocol (TCP).

Printing Information: The print_packet function formats and prints the relevant details of each TCP packet.

Execution: The script initiates the packet sniffing process using the sniff function, capturing a specific number of packets (in this case, 10). The captured packets trigger the print_packet function to display information about each TCP packet.

Permissions and Considerations: Packet sniffing usually requires elevated privileges, so please ensure that the script is executed with the necessary permissions. Additionally, be conscious of legal and ethical considerations when sniffing network traffic, as unauthorized interception may violate privacy and regulations.

In conclusion, the script is a fundamental packet sniffer that concentrates on TCP traffic. It provides insights into the specifics of the captured packets on the chosen network interface.

```
from scapy.all import*

interface = "Realtek RTL8139C+ Fast Ethernet NIC"
interface = "Realtek RTL8139C+ Fast Ethernet NIC #2"

def print_packet(packet):
    if packet.haslayer(TCP):
        ip_layer = packet[IP]
        tcp_layer = packet[TCP]
        print("[!] New Packet: {src}:{sport}->{dst}:{dport} using {proto}".format(
            src=ip_layer.src , sport=tcp_layer.sport, dst=ip_layer.dst, dport=tcp_layer.dport, proto=ip_layer.proto))

print("[*] Start sniffing...")
sniff(count=10, iface=interface, filter="tcp", prn=print_packet)
print("[*] Stop sniffing")
```

```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

========== RESTART: C:\Users\Administrator\Desktop\test5.py ==========
[*] Start sniffing...
[!] New Packet: 10.1.1.10:54679->10.1.1.1:9997 using 6
[!] New Packet: 10.1.1.1:9997->10.1.1.10:54679 using 6
[!] New Packet: 10.1.1.10:54679->10.1.1.1:9997 using 6
[!] New Packet: 10.1.1.10:54679->10.1.1.1:9997 using 6
[!] New Packet: 10.1.1.1:9997->10.1.1.10:54679 using 6
[!] New Packet: 10.1.1.10:54679->10.1.1.1:9997 using 6
[!] New Packet: 10.1.1.1:9997->10.1.1.10:54679 using 6
[!] New Packet: 10.1.1.10:54679->10.1.1.1:9997 using 6
[!] New Packet: 10.1.1.1:9997->10.1.1.10:54679 using 6
[!] New Packet: 10.1.1.10:59057->10.1.1.1:9997 using 6
[*] Stop sniffing

========== RESTART: C:\Users\Administrator\Desktop\test5.py ==========
[*] Start sniffing...
[!] New Packet: 10.2.2.10:56070->192.168.90.10:53 using 6
[!] New Packet: 10.2.2.10:59610->192.168.90.10:53 using 6
[!] New Packet: 10.2.2.10:59610->192.168.90.10:53 using 6
[!] New Packet: 10.2.2.10:59610->192.168.90.10:53 using 6
[!] New Packet: 10.2.2.10:59610->192.168.90.10:53 using 6
[!] New Packet: 10.2.2.10:55550->192.168.90.10:53 using 6
[!] New Packet: 10.2.2.10:55550->192.168.90.10:53 using 6
[!] New Packet: 10.2.2.10:55550->192.168.90.10:53 using 6
[!] New Packet: 10.2.2.10:55550->192.168.90.10:53 using 6
[!] New Packet: 10.2.2.10:56830->192.168.90.10:53 using 6
[*] Stop sniffing
```

The purpose of this script is to analyze network traffic by capturing a specified number of packets on a specific interface using Scapy. The captured packets are then processed using a callback function which extracts IP layer information and timestamps each packet. The script maintains two dictionaries to count the occurrences of source and destination IP addresses, and records the timestamps of each packet in a list. Matplotlib is used to create a histogram representing the frequency of packets over time based on their timestamps. The script then prints out the captured source and destination IP address frequencies and displays the histogram. Please note that the network adapter selected in the experiment was Ethernet0, but this can be changed for desired results on a Windows server. Additionally, due to import issues with matplotlib.pyplot, alternative modules such as plotly, seaborn, bokeh or altair can be used. Overall, this script provides valuable insights into network traffic analysis.

CYB631-3

Stats.py - C:\Users\Public\CYB631\Lab 5\Stats.py (3.11.3)

File Edit Format Run Options Window Help

```
from scapy.all import *
import time
import matplotlib.pyplot as plt

interface = "Ethernet0"
packet_count = 5000

source_ip_counts = {}
destination_ip_counts = {}
packet_times = []

def analyze_packet(packet):
    global packet_count

    if packet_count > 0:
        ip_layer = packet.getlayer(IP)
        if ip_layer:
            timestamp = int(time.time())
            packet_times.append(timestamp)

            # Count source IP addresses
            if ip_layer.src in source_ip_counts:
                source_ip_counts[ip_layer.src] += 1
            else:
                source_ip_counts[ip_layer.src] = 1

            # Count destination IP addresses
            if ip_layer.dst in destination_ip_counts:
                destination_ip_counts[ip_layer.dst] += 1
            else:
                destination_ip_counts[ip_layer.dst] = 1

            packet_count -= 1

print("[*] Start sniffing...")
sniff(count=packet_count, iface=interface, prn=analyze_packet)

# Plot packet frequencies over time
if packet_times:
    time_range = max(packet_times) - min(packet_times)
```

CYB631-3

Stats.py - C:\Users\Public\CYB631\Lab 5\Stats.py (3.11.3)

File Edit Format Run Options Window Help

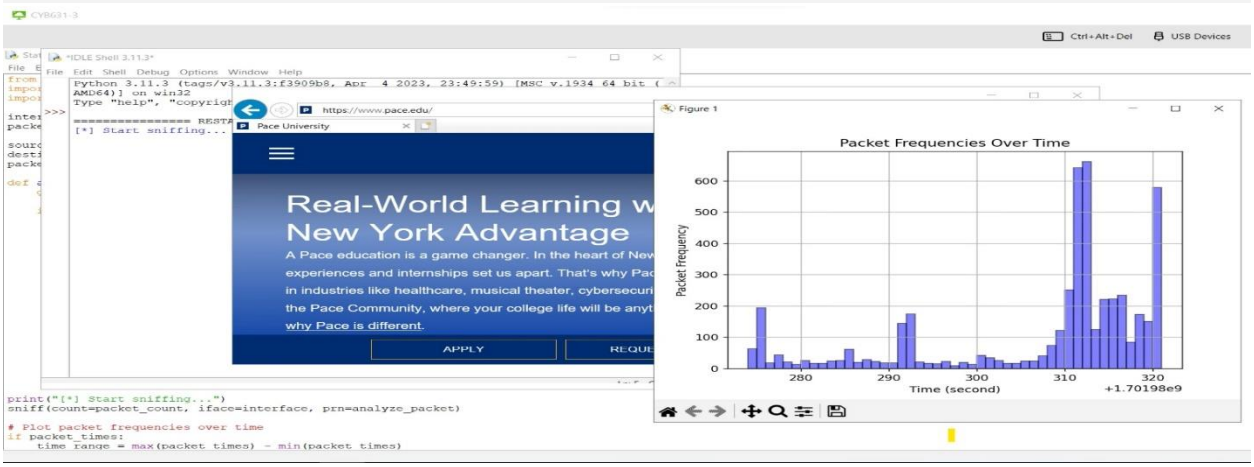
```
            if ip_layer.dst in destination_ip_counts:
                destination_ip_counts[ip_layer.dst] += 1
            else:
                destination_ip_counts[ip_layer.dst] = 1

            packet_count -= 1

print("[*] Start sniffing...")
sniff(count=packet_count, iface=interface, prn=analyze_packet)

# Plot packet frequencies over time
if packet_times:
    time_range = max(packet_times) - min(packet_times)
    if time_range <= 2:
        time_unit = "millisecond"
        bin_size = 0.001
    else:
        time_unit = "second"
        bin_size = 1

    plt.hist(packet_times, bins=int(time_range / bin_size), alpha=0.5, color='b', edgecolor='black')
    plt.xlabel(f"Time ({time_unit})")
    plt.ylabel("Packet Frequency")
    plt.title("Packet Frequencies Over Time")
    plt.grid(True)
    plt.show()

print("[*] Stop sniffing")

# Display source IP address frequencies
print("Source IP Address Frequencies:")
for source_ip, count in source_ip_counts.items():
    print(f"{source_ip}: {count} packets")

# Display destination IP address frequencies
print("Destination IP Address Frequencies:")
for destination_ip, count in destination_ip_counts.items():
    print(f"{destination_ip}: {count} packets")
```

CYB631-3

IDLE Shell 3.11.3

File Edit Shell Debug Options Window Help

```
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\Public\CYB631\Lab 5\Stats.py ================
[*] Start sniffing...
[*] Stop sniffing
Source IP Address Frequencies:
192.168.80.86: 1877 packets
10.1.29.45: 1775 packets
172.26.28.28: 737 packets
192.168.80.74: 35 packets
172.26.29.11: 9 packets
18.164.124.34: 10 packets
172.26.28.10: 2 packets
0.0.0.0: 1 packets
10.1.29.44: 16 packets
18.164.124.71: 30 packets
192.168.81.55: 33 packets
18.164.124.4: 30 packets
192.168.80.156: 8 packets
192.168.80.68: 8 packets
18.164.124.81: 20 packets
192.168.80.124: 8 packets
172.26.28.48: 97 packets
192.168.80.154: 8 packets
151.101.209.193: 104 packets
18.238.4.101: 13 packets
Destination IP Address Frequencies:
10.1.29.45: 1549 packets
192.168.80.86: 2768 packets
172.26.28.28: 181 packets
172.26.29.11: 9 packets
192.168.80.74: 39 packets
18.164.124.34: 9 packets
172.26.28.10: 3 packets
224.0.0.1: 1 packets
10.1.29.44: 15 packets
18.164.124.71: 27 packets
192.168.81.55: 36 packets
18.164.124.4: 27 packets
```



4]

```powershell
while($true){
 $test = (Test-NetConnection -ComputerName 10.1.1.1 -Port 80).TcpTestSucceeded
if($test -eq  $false){
        Add-Type -AssemblyName System.Windows.Forms
        $global:balloon = New-Object System.Windows.Forms.NotifyIcon
        $path = (Get-Process -Id $pid).Path
        $balloon.Icon = [System.Drawing.Icon]::ExtractAssociatedIcon($path)
        $balloon.BalloonTipIcon = [System.Windows.Forms.ToolTifipIcon]::Warning
        $balloon.BalloonTipText = 'Server is not reachable'
        $balloon.BalloonTipTitle = "Attention $Env:USERNAME"
        $balloon.Visible = $true
        $balloon.ShowBalloonTip(9000)
} else {
        Add-Type -AssemblyName System.Windows.Forms
        $global:balloon = New-Object System.Windows.Forms.NotifyIcon
        $path = (Get-Process -Id $pid).Path
        $balloon.Icon = [System.Drawing.Icon]::ExtractAssociatedIcon($path)
        $balloon.BalloonTipIcon = [System.Windows.Forms.ToolTipIcon]::Warning
        $balloon.BalloonTipText = 'Server is running fine!'
        $balloon.BalloonTipTitle = "$Env:USERNAME status update"
        $balloon.Visible = $true
        $balloon.ShowBalloonTip(9000)
 }
 }
```