LABORATORY MANUAL

SOC V – INTRUSION DETECTION SYSTEMS

(20CSC610)

For

IV Year I Sem B. Tech

Academic Year 2023-24

Prepared By

Kuppam Johari

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CYBER SECURITY

MADANAPALLE INSTITUTE OF TECHNOLGY & SCIENCE

(UGC – AUTONOMOUS)

(Affiliated to JNTUA, Ananthapuramu)

Accredited by NBA, Approved by AICTE, New Delhi)

AN ISO 9001:2008 Certified Institution

P. B. No: 14, Angallu, Madanapalle – 517325

2023-2024

UNIT - 1

Experiment No: 1

Configure a virtual network using tools like VirtualBox or VMware.

Aim: Configure a virtual network using tools like VirtualBox or VMware.

Description:

Configuring a virtual network provides flexibility and control over how virtual machines connect to each other and the external world. This is particularly useful for development, testing, and learning environments, as it allows you to simulate a variety of networking scenarios without needing a physical network infrastructure.

There are 4 types of networking scenarios.

Network Address Translation (NAT):

NAT is often used when you want the virtual machine to have internet access but don't necessarily need direct visibility of the virtual machine from other devices on your local network. It's suitable for scenarios where the VM needs outbound connectivity.

Bridged Networking:

Bridged networking is useful when you want your virtual machine to have its own distinct IP address on the local network, allowing other devices on the network to directly communicate with it. This is commonly used for scenarios where the virtual machine should be treated like a separate machine within your network.

Host-Only Networking:

Host-Only networking is employed when you want the virtual machine to be isolated from external networks while still allowing communication with the host machine. This can be useful for development and testing environments where you need to keep the virtual machine and host machine isolated from other network resources.

NOTE: You can customize this network configuration according to the use case.

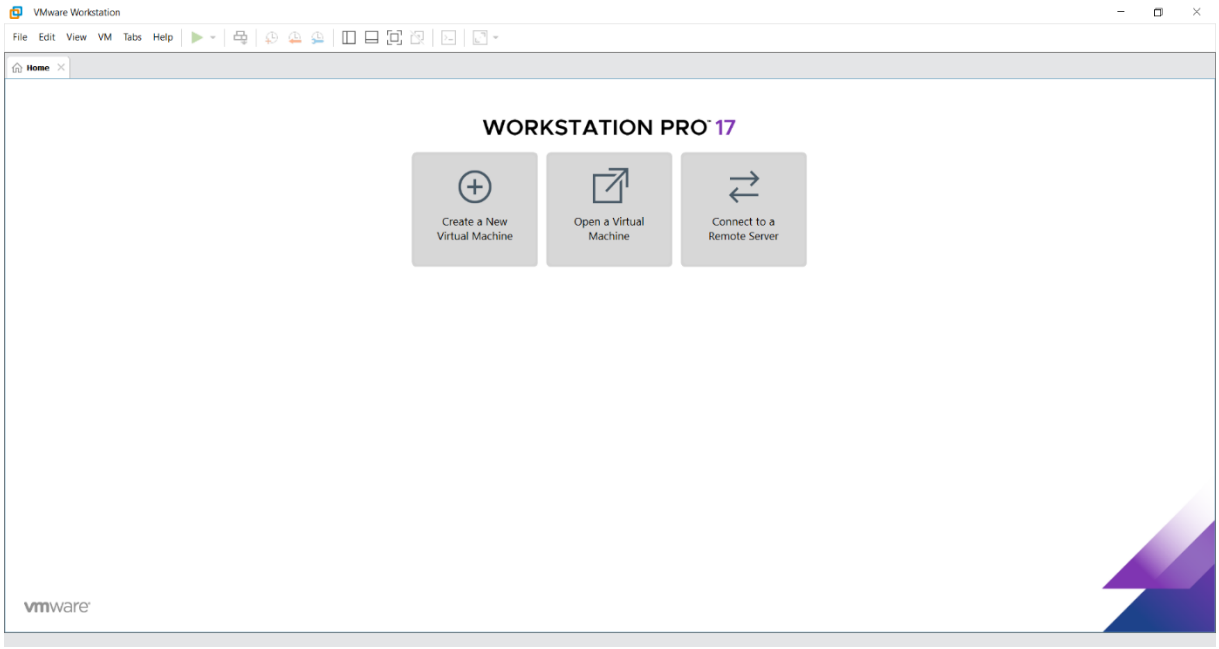
Required tools.

- VMware
- Operating System (any Linux Distribution)

Configure a virtual network using VMware.

Algorithm:

1. **Open VMware:** Launch the VMware application.



2. **Create a Virtual Machine:** If you haven't already created a virtual machine, you can create one by following the below link.

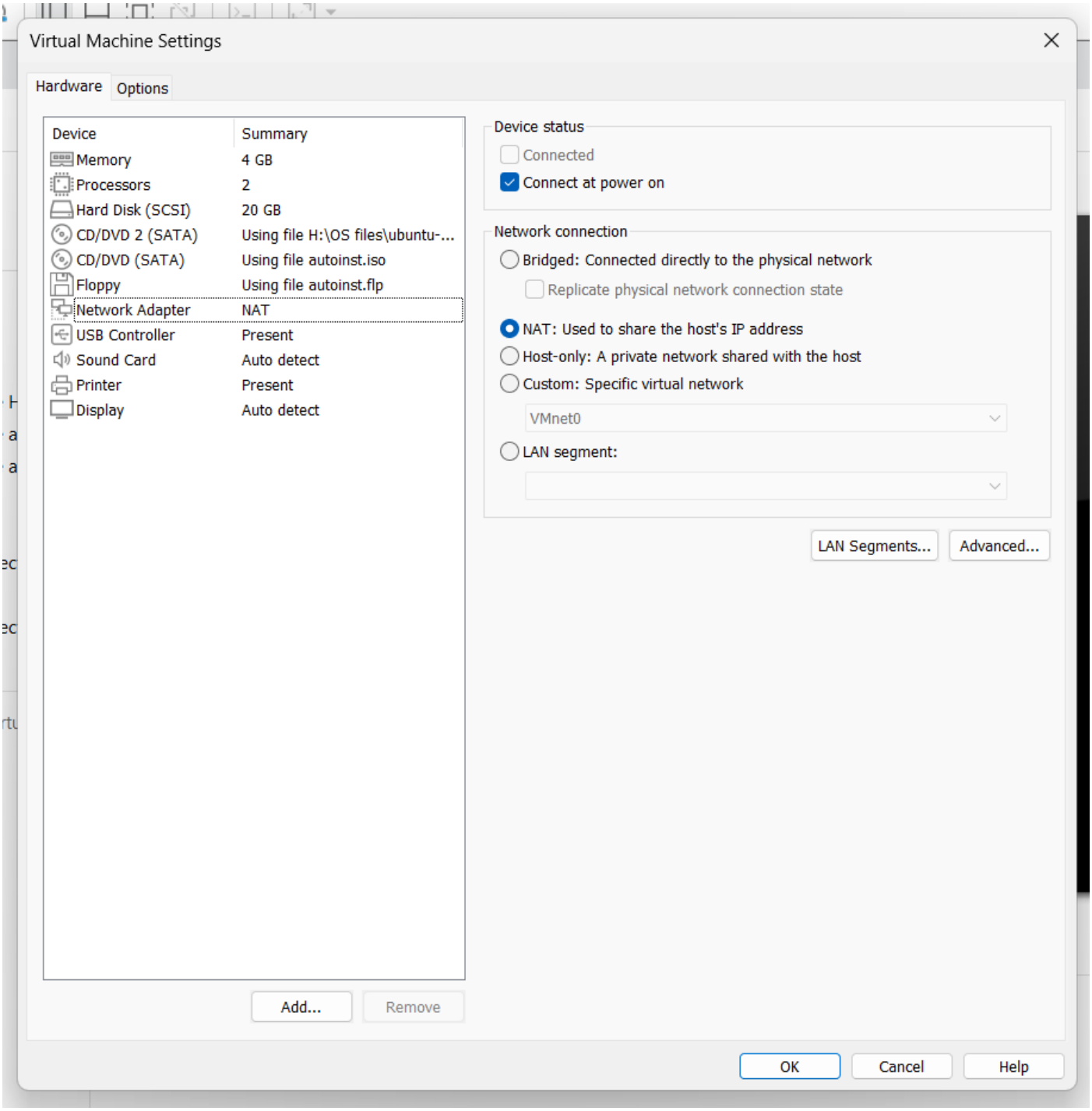
" https://drive.google.com/file/d/1Z1cmi27wmgVZTa0PCnGwEHCHANdAU3US/view?usp=sharing "

3. **Configure Network Settings:** After creating the virtual machine, select it and click on the "Settings" button.
4. **Network Adapter Settings:** In the Settings window, go to the "Network" section. Here, you'll see one or more network adapters. You can choose from several adapter types, such as NAT, Bridged, Host-Only, etc.
5. **Adjust Adapter Settings:** Depending on the adapter type you choose; you may need to adjust additional settings. For example, in Bridged mode, you might need to select the network adapter that your host machine uses.
6. **Save Settings:** Once you've configured the network settings as desired, click "OK" to save the changes.
7. **Start the Virtual Machine:** Start the virtual machine. It should now be able to connect to the network according to the settings you've configured.

Click the below link to watch the process as followed above steps.

" https://drive.google.com/file/d/19RaI2vDkgRwusmFQ0m8y9u1SpJaz1w9d/view?usp=sharing "

Output:



Video source:

" https://drive.google.com/drive/folders/1rt-qFrAc0SevlKEYlldwlEglV8NRI70f?usp=sharing "

Result: Thus, to Configure a virtual network using tools like VirtualBox or VMware is successfully completed.

Experiment No: 2

Deploy an IDS System such as Snort or Suricata, within the virtual network.

Aim: Deploy an IDS System such as Snort or Suricata, within the virtual network.

Description:

Deploying an IDS within a virtual network helps you proactively monitor and defend your network against potential threats, providing an additional layer of security to your virtual environment. It's important to keep the IDS system up to date and continuously adjust its rules and configurations based on the evolving threat landscape.

Tools Required:

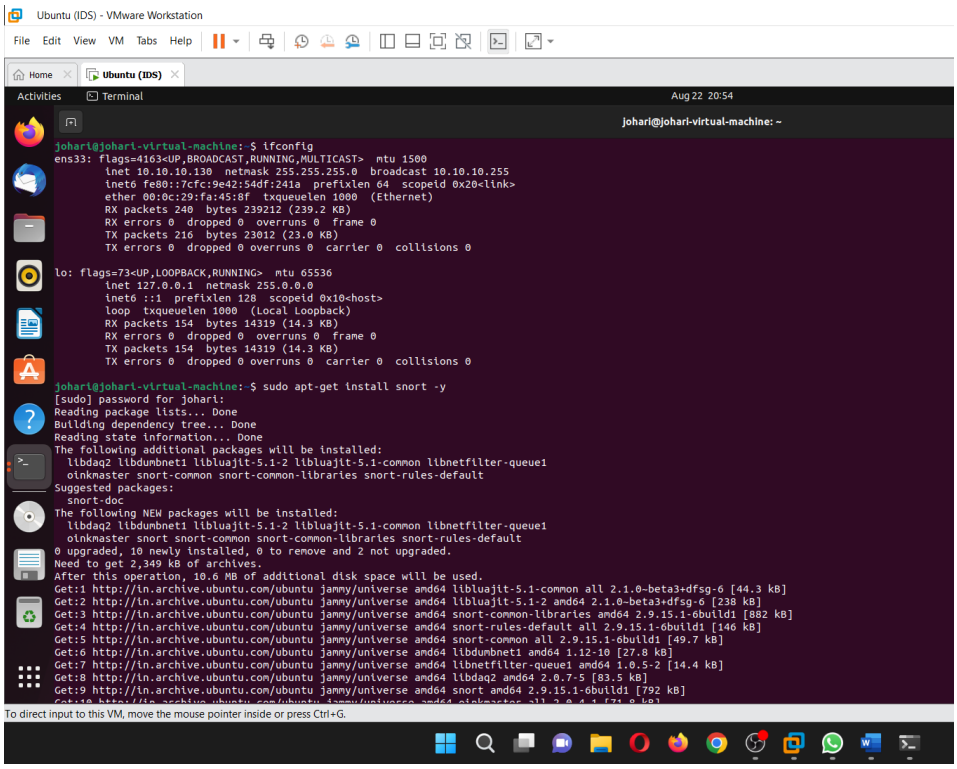
- Virtual Machine's (Which we are created in exp)
- Snort Tool
- Ping, Zenmap (nmap) Tool & Hping3
- Vim editor

Algorithm:

Installation steps for Snort:

Video: "https://drive.google.com/file/d/1oe9qyBnnum3yCiT8w71Kuj4_0jAy-ysC/view?usp=sharing"

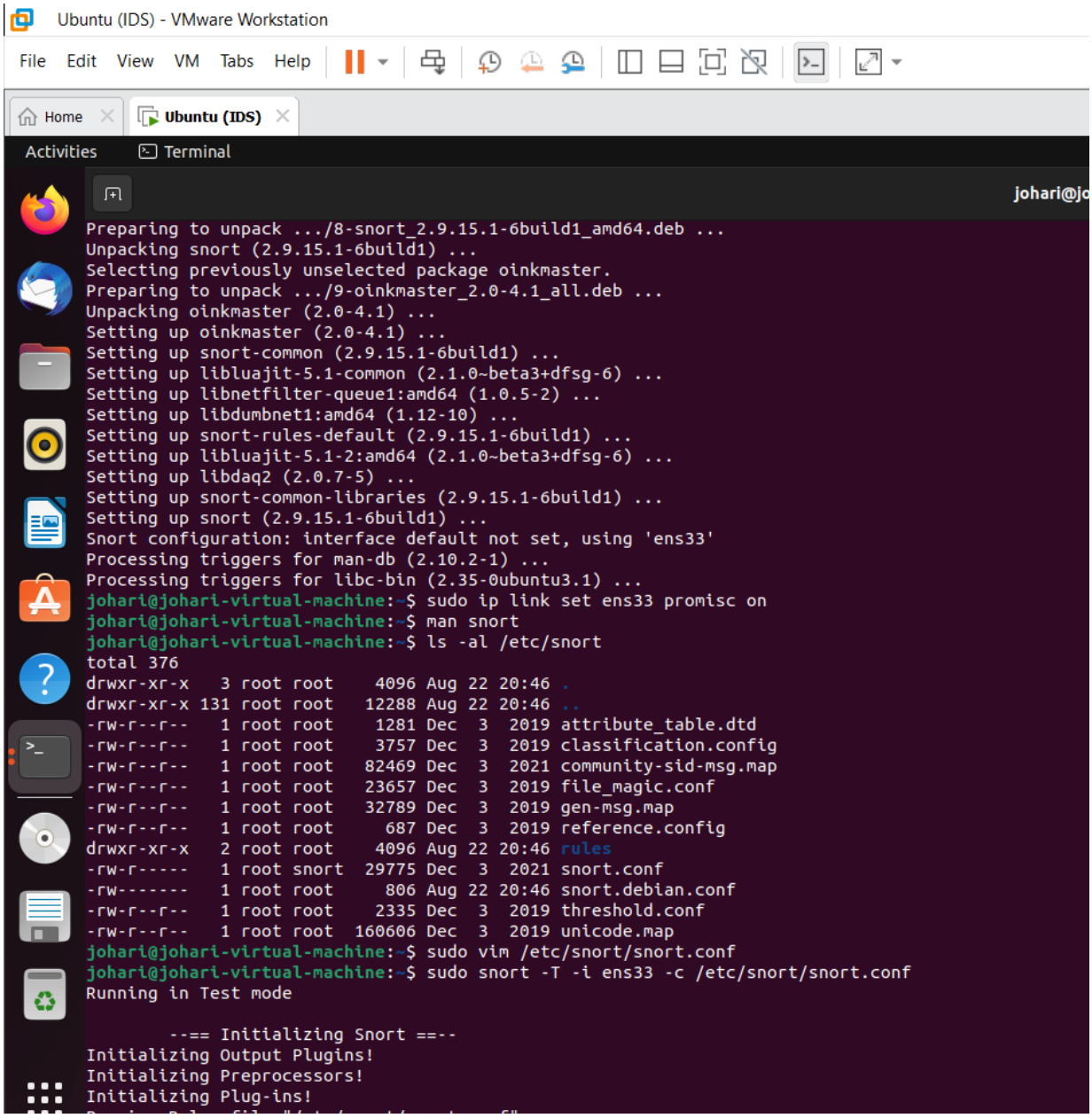
- Sudo apt-get install snort -y



- Enter ip address 10.10.10.12/24
- To check enter ip a s
- And hit enter.
- Snort --version

Configuration of Snort:

- ls -al /etc/snort

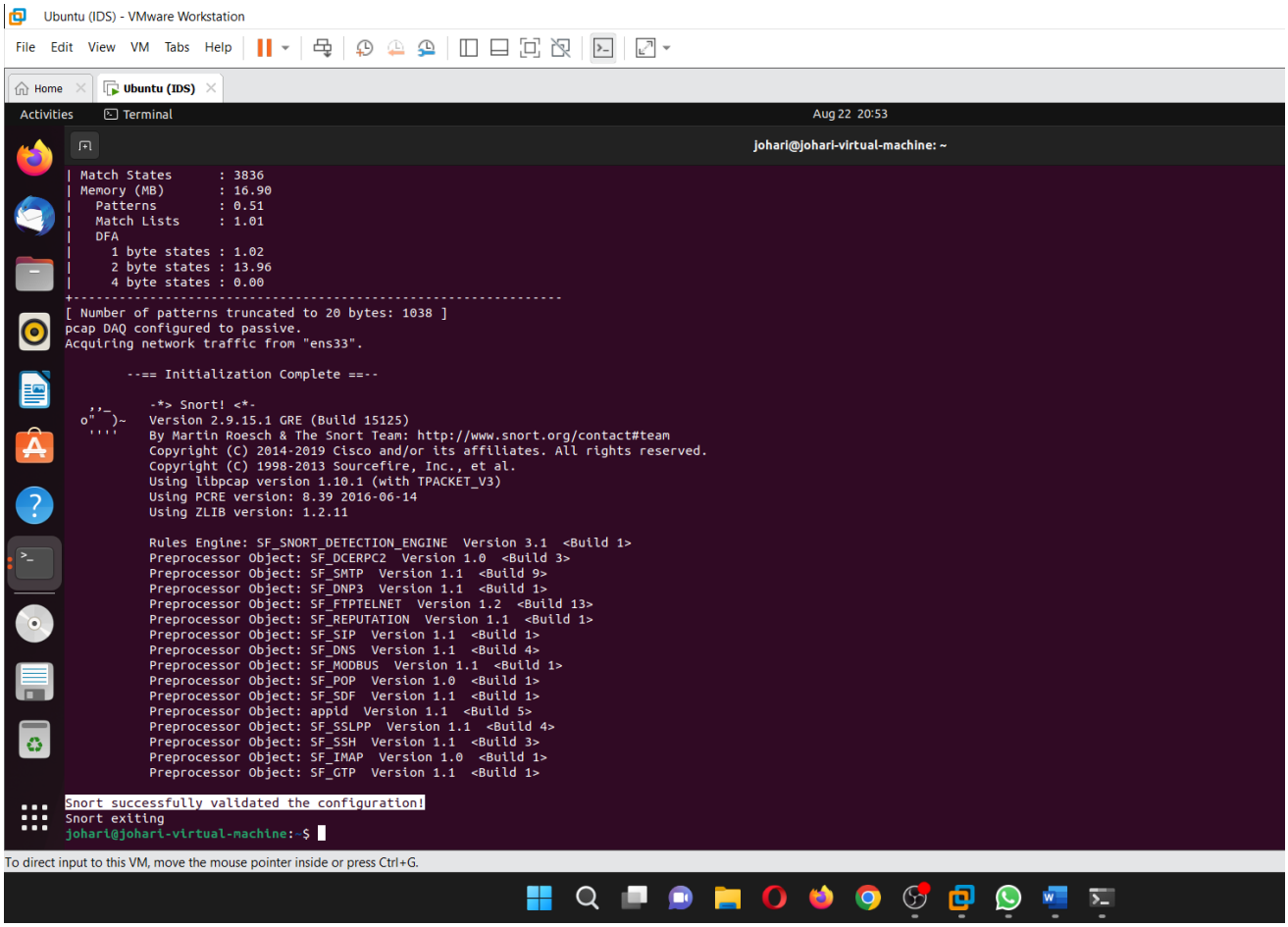


- sudo vim /etc/snort/snort.conf
- change under step 1 ipvar HOME_NET any to ipvar HOME_NET 10.10.10.12/24

Note: refer the video to do.

Checking Configuration to confirm no errors:

- sudo snort -T -i ensp -c /etc/snort/snort.conf



Rules:

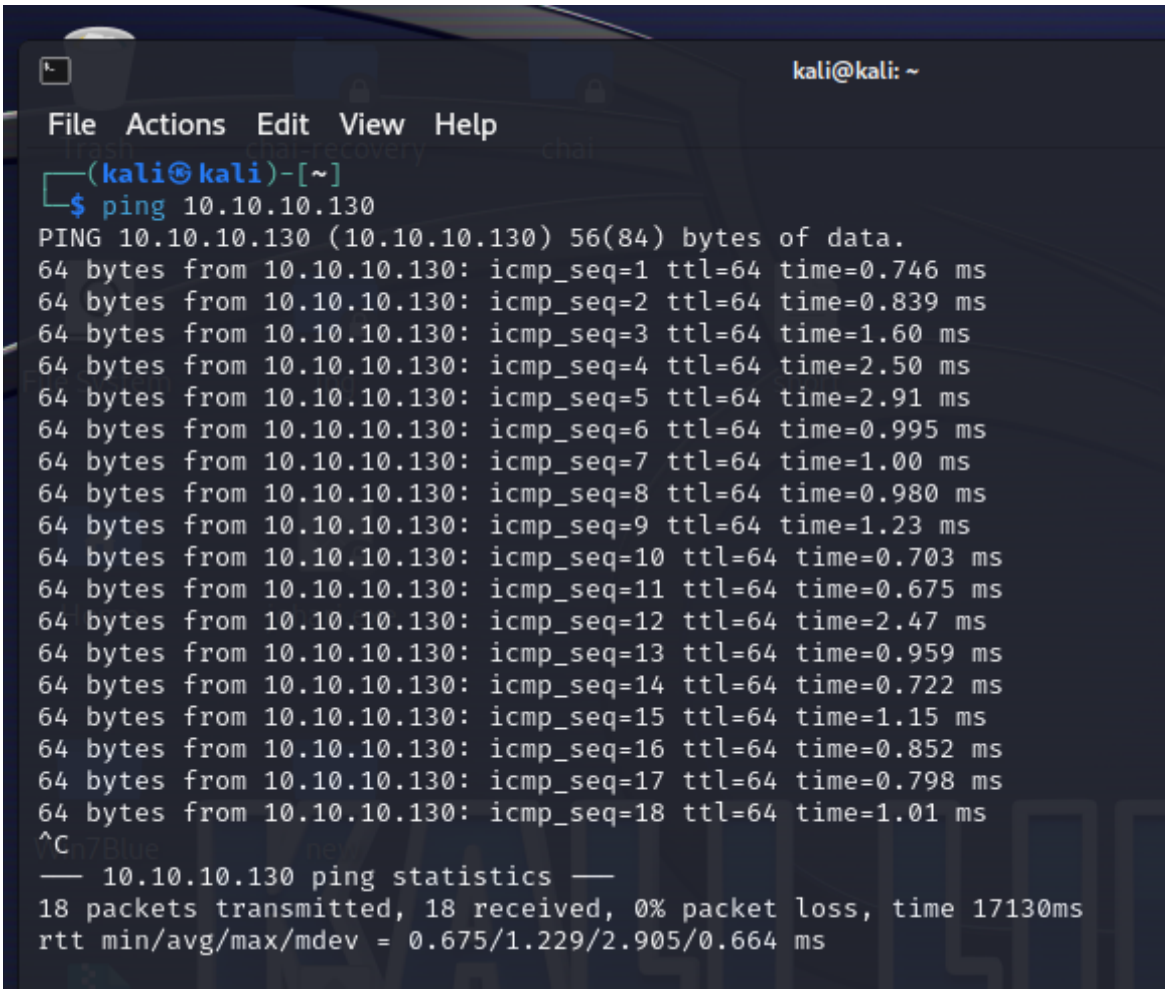
- alert icmp any any -> $HOME_NET any (msg:"Ping Detected"; sid:100001; rev:1;)
- alert icmp any any -> $HOME_NET 22 (msg:"SSH Detected"; sid:100002; rev:1;)

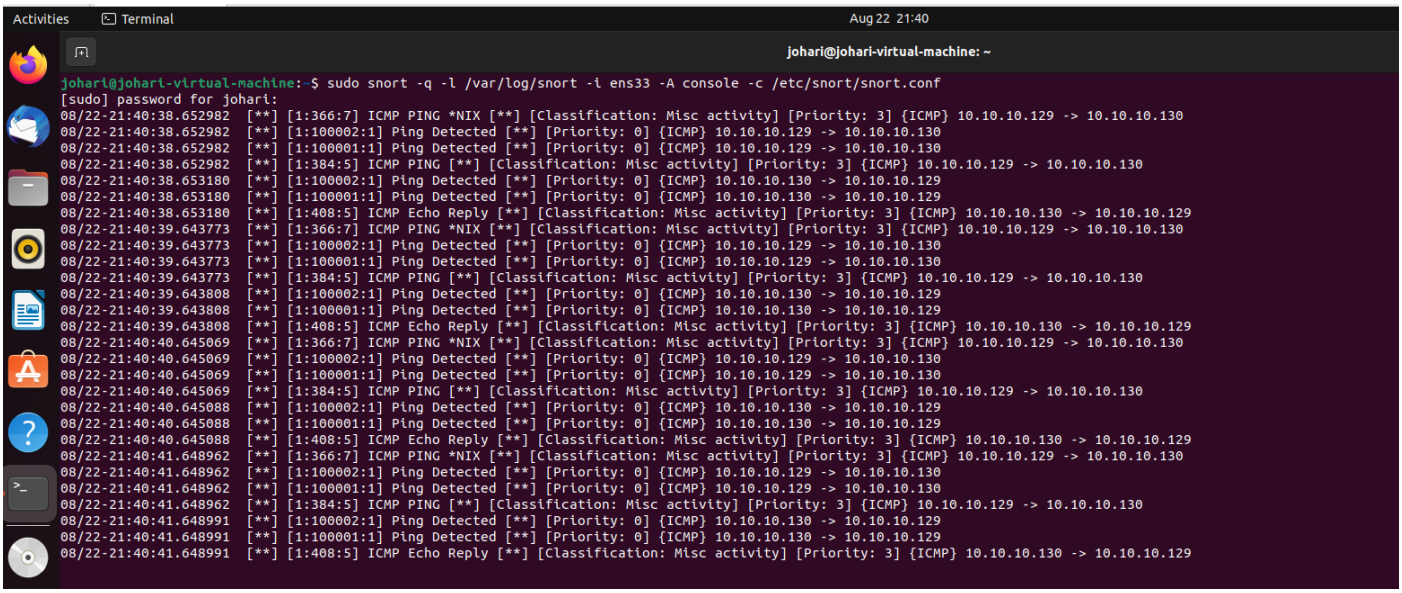
Testing Snort using ping, Zenmap & Hping3:

- ping 10.10.10.130
- ssh username@ipaddress

Video 2: "https://drive.google.com/file/d/1xsvV-VeIjQdvJFFRkErD_ZikrEi9_7fT/view?usp=sharing"

Output:





Video source:

“ https://drive.google.com/drive/folders/1rt-qFrAc0SevlKEYlldwlEglV8NRI70f?usp=sharing ”

Result: Thus, to deploy an IDS System such as Snort or Suricata, within the virtual network is successfully Executed.

UNIT - 2

Experiment No: 1

Setup a test network using virtual machines or physical devices.

Aim: Setup a test network using virtual machines or physical devices.

Description:

Setting a test network provides flexibility and control over how virtual machines connect to each other and the external world. This is particularly useful for development, testing, and learning environments, as it allows you to simulate a variety of networking scenarios without needing a physical network infrastructure.

Network Address Translation (NAT):

NAT is often used when you want the virtual machine to have internet access but don't necessarily need direct visibility of the virtual machine from other devices on your local network. It's suitable for scenarios where the VM needs outbound connectivity.

Bridged Networking:

Bridged networking is useful when you want your virtual machine to have its own distinct IP address on the local network, allowing other devices on the network to directly communicate with it. This is commonly used for scenarios where the virtual machine should be treated like a separate machine within your network.

Host-Only Networking:

Host-Only networking is employed when you want the virtual machine to be isolated from external networks while still allowing communication with the host machine. This can be useful for development and testing environments where you need to keep the virtual machine and host machine isolated from other network resources.

NOTE: You can customize this network configuration according to the use case.

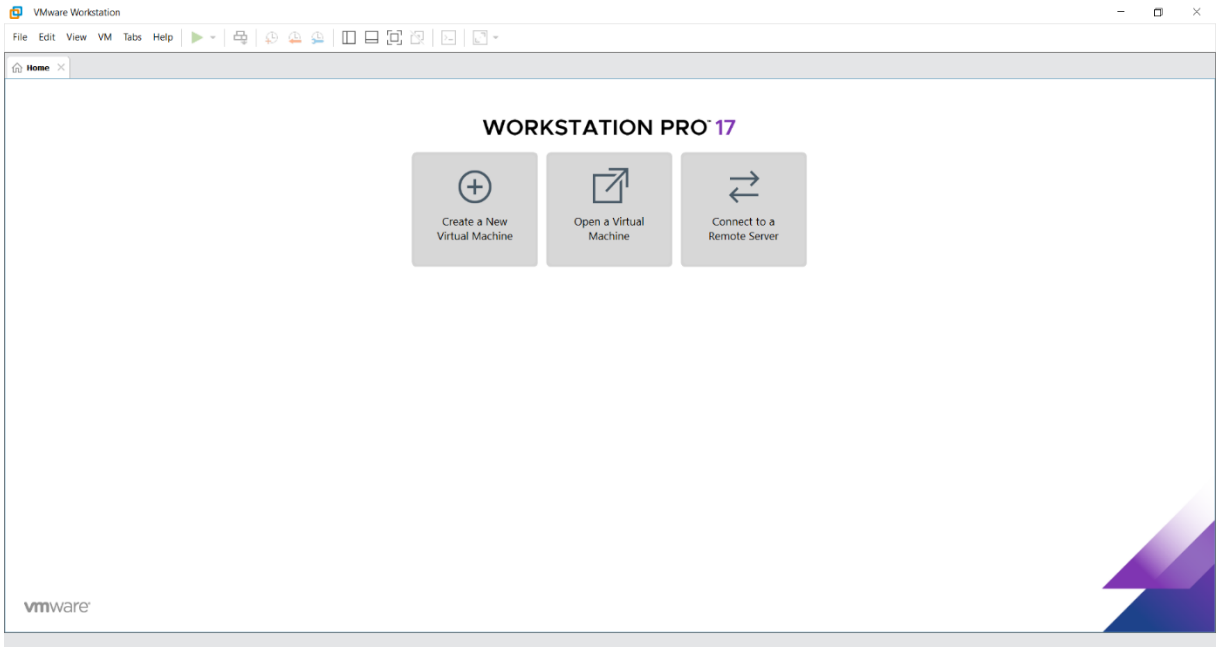
Required tools.

- VMware
- Operating System (any Linux Distribution)

Setting a virtual network using VMware.

Algorithm:

8. **Open VMware:** Launch the VMware application.



9. **Create a Virtual Machine:** If you haven't already created a virtual machine, you can create one by following the below link.

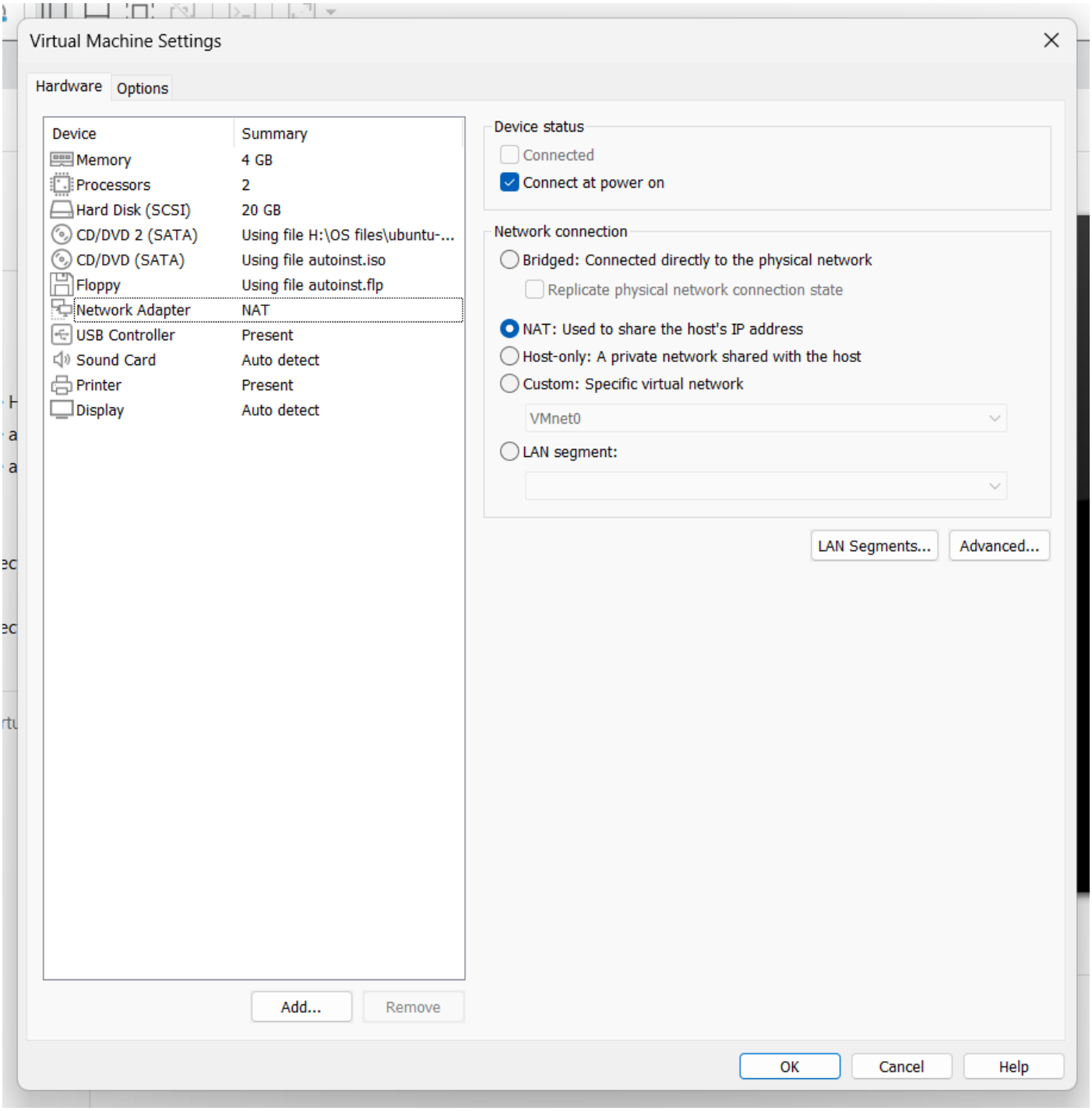
" https://drive.google.com/file/d/1Z1cmi27wmgVZTa0PCnGwEHCHANdAU3US/view?usp=sharing "

10. **Configure Network Settings:** After creating the virtual machine, select it and click on the "Settings" button.
11. **Network Adapter Settings:** In the Settings window, go to the "Network" section. Here, you'll see one or more network adapters. You can choose from several adapter types, such as NAT, Bridged, Host-Only, etc.
12. **Adjust Adapter Settings:** Depending on the adapter type you choose; you may need to adjust additional settings. For example, in Bridged mode, you might need to select the network adapter that your host machine uses.
13. **Save Settings:** Once you've configured the network settings as desired, click "OK" to save the changes.
14. **Start the Virtual Machine:** Start the virtual machine. It should now be able to connect to the network according to the settings you've configured.

Click the below link to watch the process as followed above steps.

" https://drive.google.com/file/d/19RaI2vDkgRwusmFQ0m8y9u1SpJaz1w9d/view?usp=sharing "

Output:



Video source:

" https://drive.google.com/drive/folders/1rt-qFrAc0SevlKEYlldwlEglV8NRI70f?usp=sharing "

Result: Thus, to Setup a test network using virtual machines or physical devices is successfully completed.

Experiment No: 2

Use Wireshark or tcpdump to capture network traffic on the test network.

Aim: Use Wireshark or tcpdump to capture network traffic on the test network.

Description:

1. **Wireshark:** Wireshark is a graphical network protocol analyzer that allows users to capture and inspect network traffic in real-time.
2. **tcpdump:** tcpdump is a command-line packet capture tool for Unix-like operating systems that captures network packets on a specified network interface.

Tools Required:

- Wireshark
- Tcpdump

Algorithm:

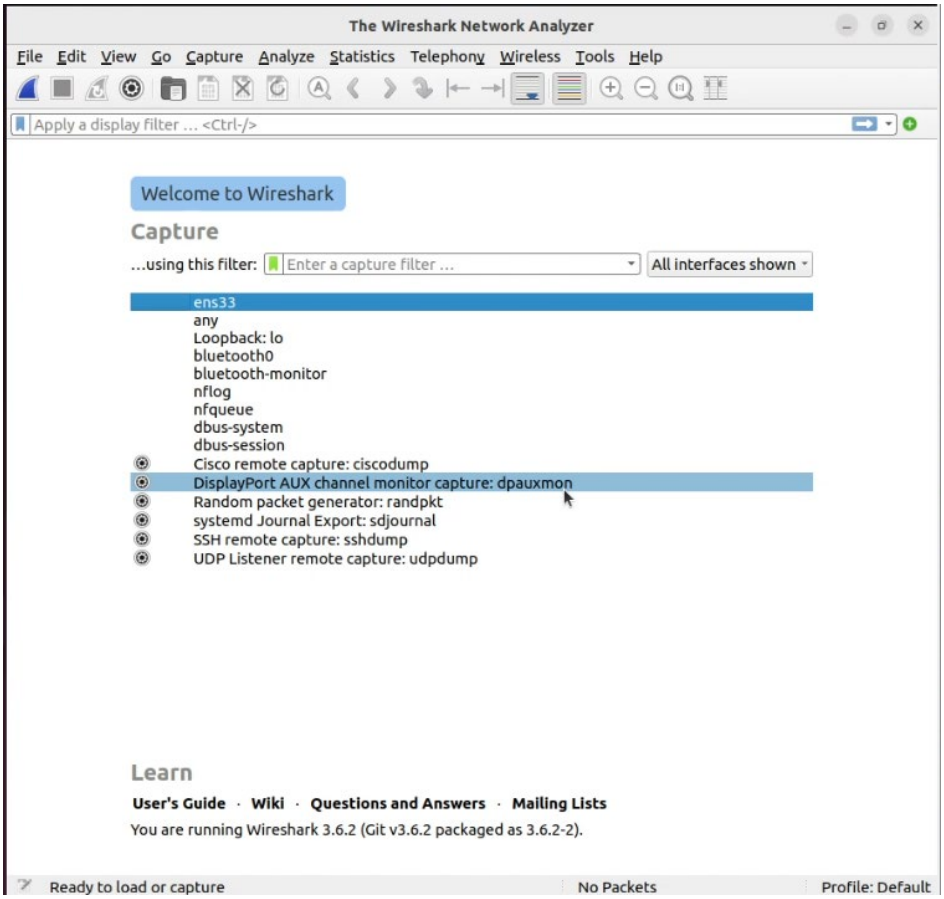
Using Wireshark:

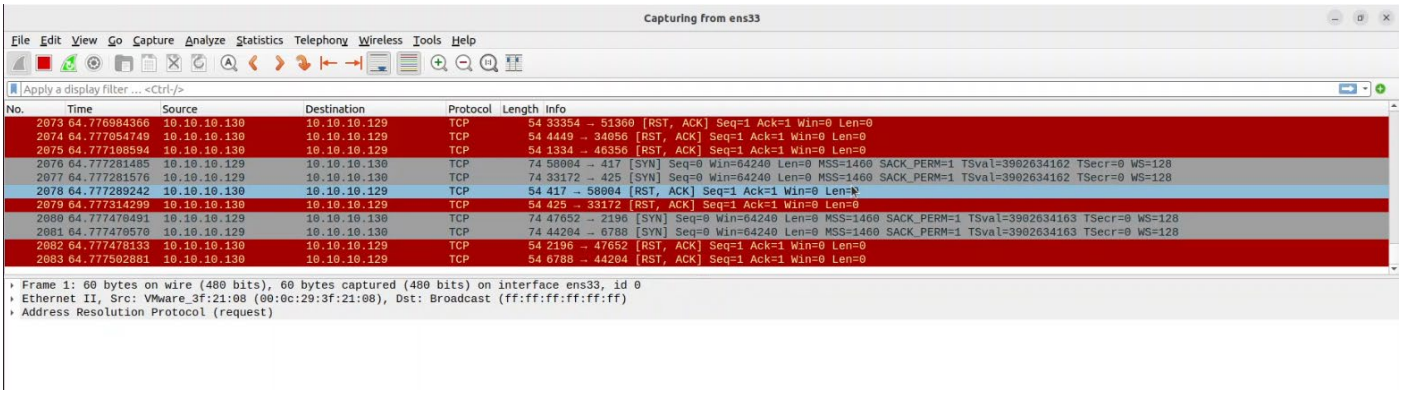
1. **Install Wireshark:** If you don't have Wireshark installed, download and install it from the official website: https://www.wireshark.org/download.html
2. **Launch Wireshark:** Open Wireshark with administrative privileges. You may need to run it as an administrator or use sudo on Linux systems.
3. **Select Capture Interface:**
   - Go to "Capture" in the top menu & choose the network interface you want to capture traffic from.
4. **Start Capturing:**
   - Click the "Start" button to begin capturing network traffic.
   - You can apply filters to capture specific traffic (e.g., filter by IP address, port, protocol).
5. **Stop Capturing:**
   - Click the "Stop" button when you want to stop capturing.
6. **Analyze Traffic:**
   - After stopping the capture, you can analyze the captured packets in the Wireshark interface.
7. **Save the Capture:**
   - If needed, you can save the capture as a .pcap or .pcapng file for further analysis.

Video source:

https://drive.google.com/file/d/1b5kOTBSYLwg-tFdEUdtDX_DtuQCznHw0/view?usp=drive_link

Output:





Using tcpdump:

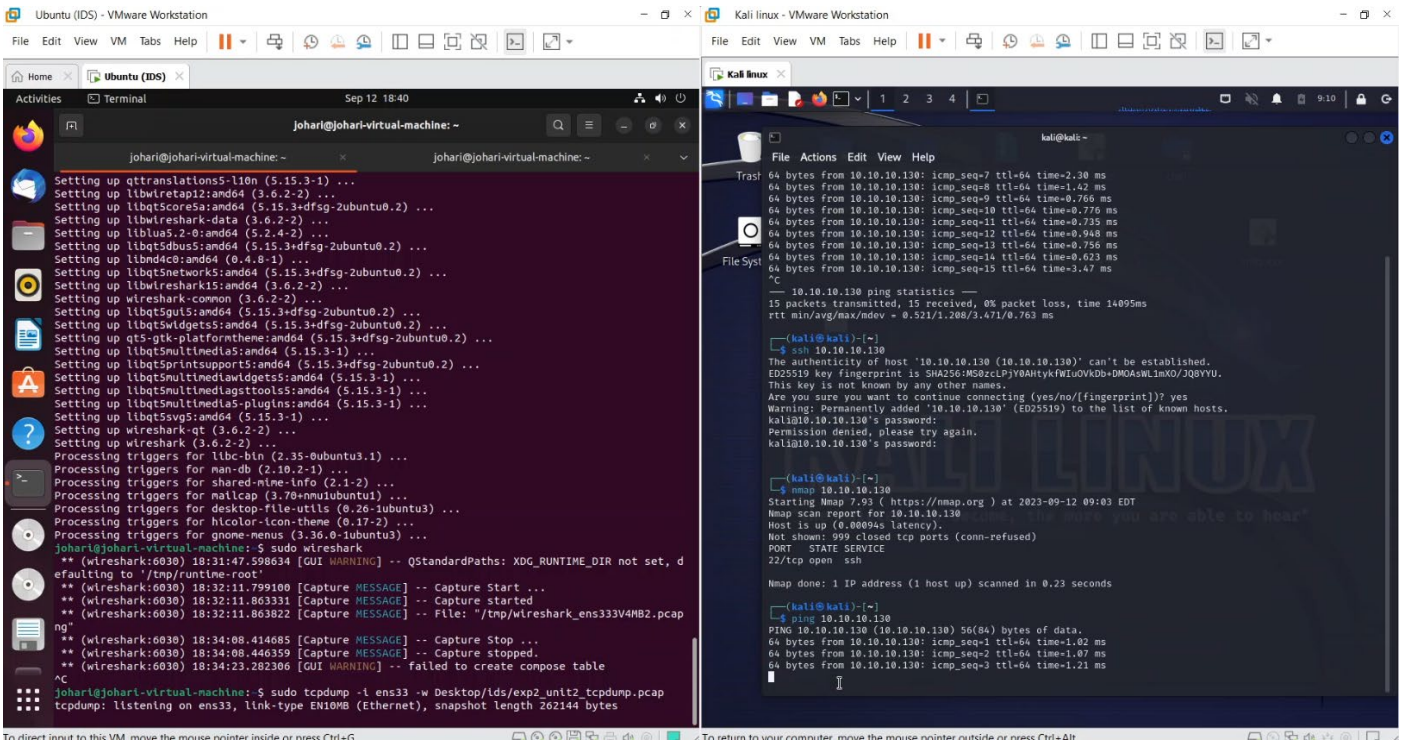
1. **Open Terminal:**
   - Open a terminal window on your computer.
2. **Run tcpdump:**
   - Use the tcpdump command to capture network traffic. For example, to capture all traffic on interface eth0, you can use the following command: **sudo tcpdump -i eth0 -w capture.pcap**

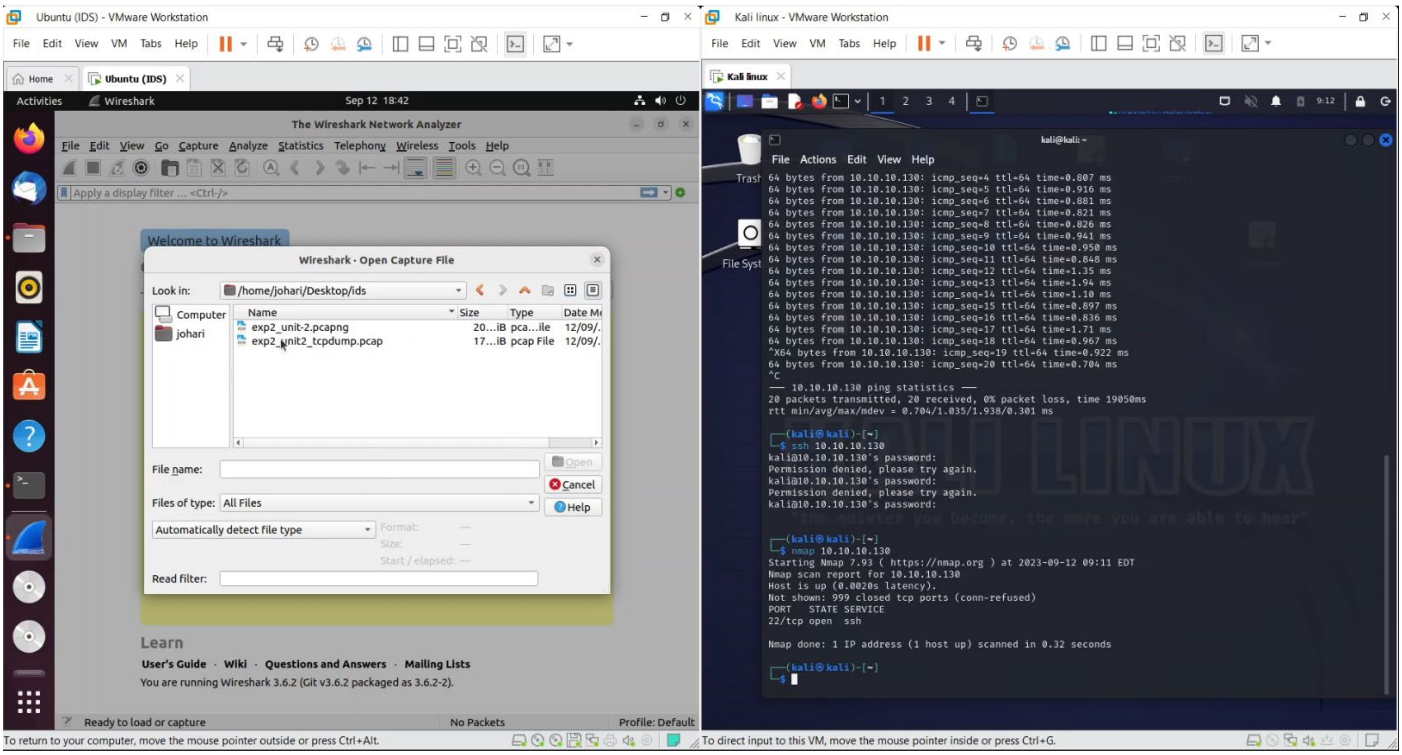
Note: -i (interface name), -w (save the captured details in a file specified).

- This command captures traffic and saves it to a file named "capture.pcap."

3. **Stop Capturing:**
    - To stop capturing, press Ctrl + C in the terminal.
4. **Analyze Traffic:**
    - You can use Wireshark to analyze the saved capture file (e.g., capture.pcap) by opening it in Wireshark.

Output:





Video source:

https://drive.google.com/file/d/1-q5jojtBTtLN5ucsBzt_c8BEuu0NI1DT/view?usp=drive_link

Result: Thus, to Use Wireshark or tcpdump to capture network traffic on the test network Successfully Completed.

Experiment No: 3

Analyse captured packets to identify protocols, extract information from headers and identify any anomalies or suspicious activity.

Aim: Analyse captured packets to identify protocols, extract information from headers and identify any anomalies or suspicious activity.

Algorithm:

1. **Open Wireshark:**
   - Launch Wireshark on your computer.
2. **Open the Packet Capture File:**
   - Go to "File" > "Open" and browse to the location of your captured packet file.
   - Select the file and click "Open."
3. **View Packet List:**
   - The top pane of Wireshark displays a list of captured packets. This is where you'll start your analysis.
   - Each row represents a single packet, and columns provide summary information about each packet (e.g., source and destination addresses, protocol, length).
4. **Identify Protocols:**
   - Wireshark automatically categorizes packets by protocol. You can expand protocol categories in the packet list pane to see specific protocols used in the capture.
5. **Select a Packet for Analysis:**
   - Click on a packet in the list to select it. This will populate the packet details pane below with information about that specific packet.
6. **Extract Information from Headers:**
   - In the packet details pane, expand the various protocol layers to view specific header information.
   - Common information to extract includes source and destination IP addresses, port numbers, protocol versions, and sequence numbers, depending on the protocol.
7. **Apply Filters:**
   - To focus on specific types of traffic or protocols, use Wireshark's display filters.
   - In the display filter field at the top of the screen, enter a filter expression (e.g., "ping" to show only PING traffic).
   - Press "Enter" to apply the filter, and the packet list will update accordingly.
8. **Identify Anomalies:**
   - Look for irregular patterns or anomalies in the captured traffic.

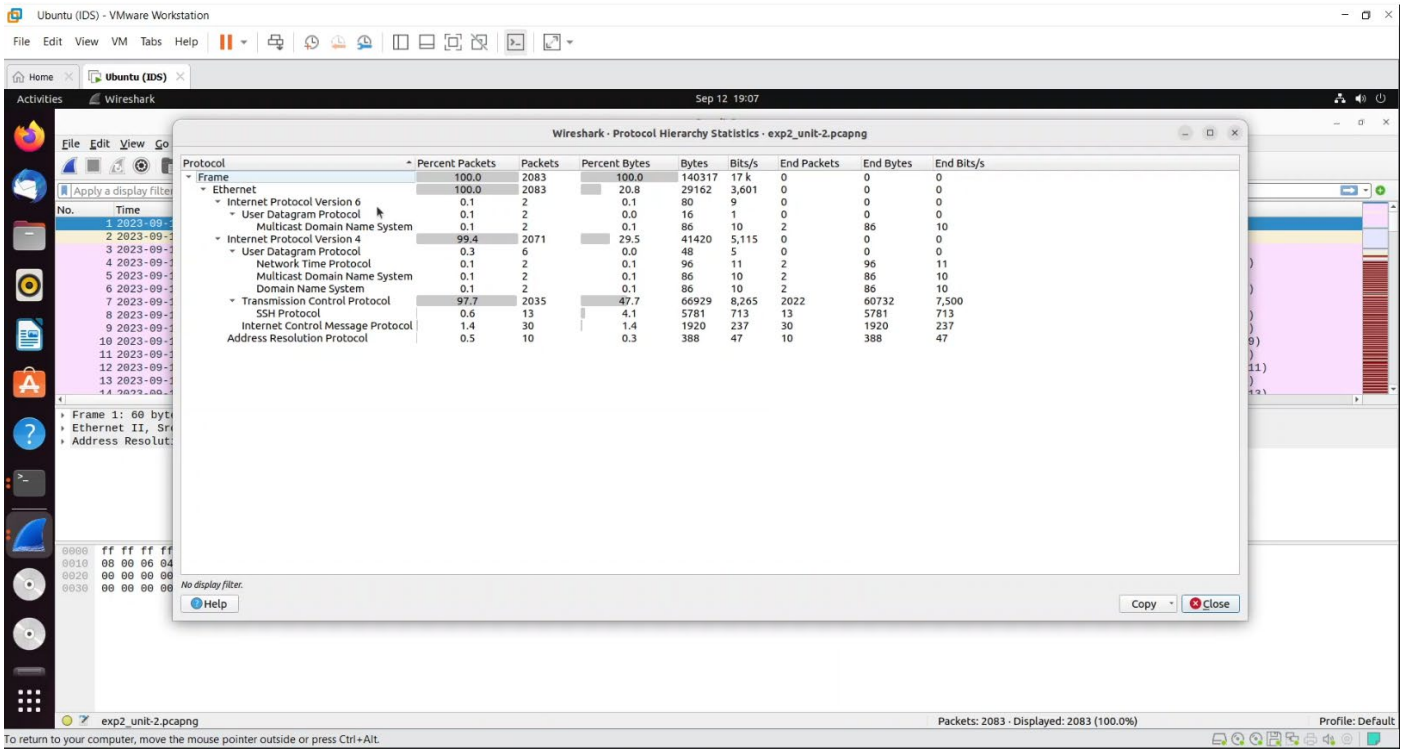
- Pay attention to unexpected or unknown protocols, unusual traffic patterns, repeated connection attempts, incorrect checksums, and unusually large or small packet sizes.

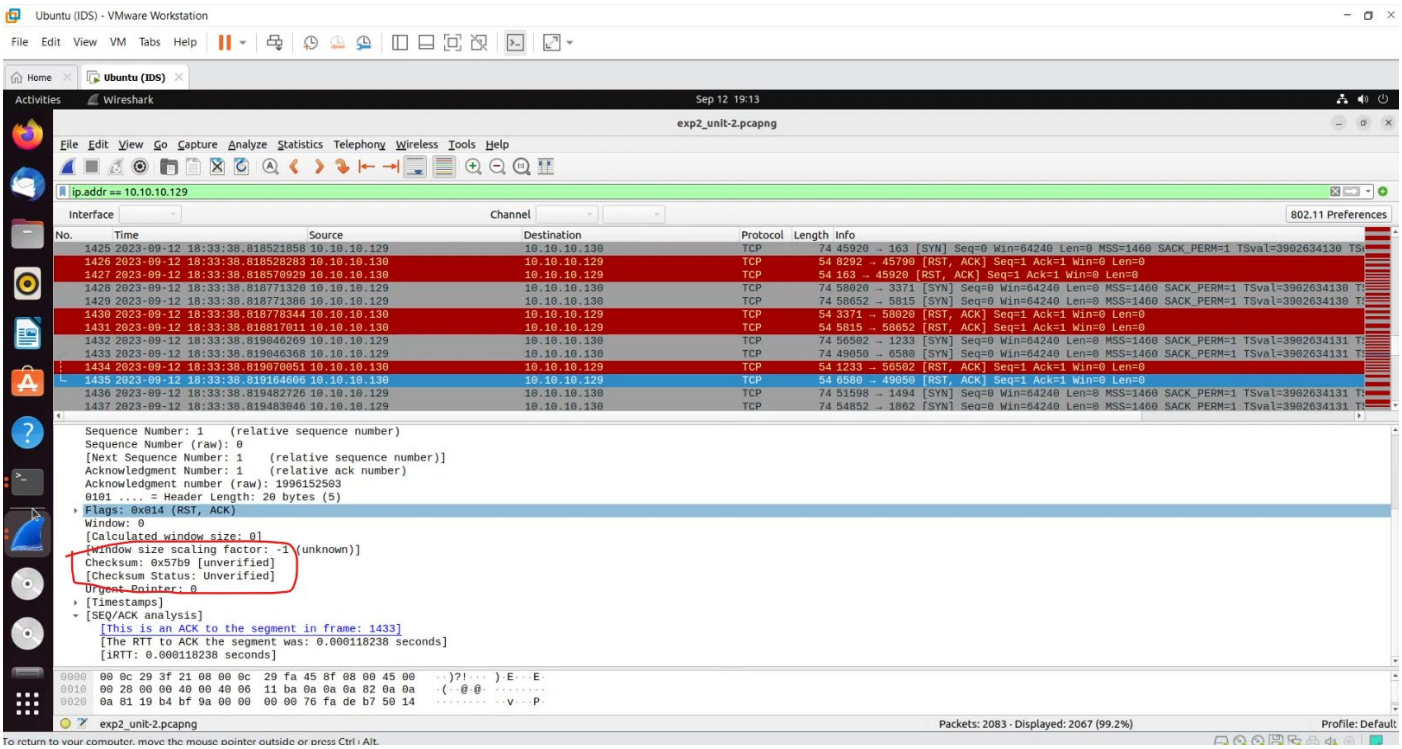
9. **Use Colorization:**
    - Wireshark provides colorization to highlight packets that match specific criteria. For example, suspicious packets can be color-coded to stand out.
10. **Follow TCP Streams (if applicable):**
    - If you're analyzing TCP traffic, you can right-click on a TCP packet and select "Follow" > "TCP Stream" to view the entire conversation between two hosts.
11. **Refer to Documentation:**
    - If you encounter unfamiliar protocols or behavior, refer to documentation or online resources to better understand the expected behavior.
12. **Compare with Baseline:**
    - If available, compare the captured traffic with a baseline of expected network behavior to identify deviations.
13. **Report and Investigate:**
    - If you identify anomalies or suspicious activity that could indicate a security threat, report it to your network security team or follow your organization's incident response procedures.

Video source:

https://drive.google.com/file/d/1KvmWU0fwb5vRMd7iI0Gt3tRj35wM1_CR/view?usp=drive_link

Output:





Result:

Thus, to Analyse captured packets to identify protocols, extract information from headers and identify any anomalies or suspicious activity successfully Executed.

UNIT – 3

Experiment No: 1

Write a program that reads network traffic data or log files.

Aim: Write a program that reads network traffic data or log files.

Tools Required:

- Python Editor
- pyshark (pip install pyshark)
- scapy (pip install scapy)

Reads Network Traffic

Algorithm:

1. Import the necessary libraries, such as pyshark.
2. Display a prompt to the user to include the ".cap" extension in the output file name.
3. Prompt the user to enter the desired output file name for the PCAP capture.
4. Create a LiveCapture object using the specified output file name.
5. Start capturing network traffic using cap.sniff_continuously() with a specified duration or packet count (e.g., 20 seconds or a certain number of packets).
6. Set up a loop that continues until either the specified duration or packet count is reached or the user interrupts the program by pressing 'Ctrl+C'.
7. Inside the loop, check if the user wants to stop capturing. If the user enters "yes," break out of the loop to stop capturing.
8. If the user does not want to stop capturing, continue capturing packets.
9. Handle any exceptions, such as KeyboardInterrupt (triggered by 'Ctrl+C'), and display a message indicating that the capture was stopped by the user.
10. Once the capture is stopped, close the capture object and save the captured packets to the specified PCAP file.

```
# Here's a Python code that follows this algorithm:

import pyshark

print("Note: .pcap extension in the output file name")
file_name = input("Enter the output file name: ")
cap = pyshark.LiveCapture(output_file=file_name)
try:
    print("Capturing traffic, Press 'Ctrl+C' to stop...")
    for packet in cap.sniff_continuously(packet_count=20):
        a = input("Do you want to stop capturing (yes/no): ").strip().lower()
        if a == "yes":
            print("Capture stopped by user.")
            break
except KeyboardInterrupt:
```

```
print("Capture stopped by user.")
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS H:\MITS-Work\karthik_ram_sir-work\IDS_lab\Unit_3\exp1>  & 'C:\Users\Johari\AppData\Local\Programs\Python\Python311\python.
les\lib\python\debugpy\adapter/../..\debugpy\launcher' '20017' '--' 'H:\MITS-Work\karthik_ram_sir-work\IDS_lab\Unit_3\exp1\pa
Note: .pcap extension in the output file name
Enter the output file name: johari.pcap
Capturing traffic, Press 'Ctrl+C' to stop...
Do you want to stop capturing (yes/no): yes
Capture stopped by user.
PS H:\MITS-Work\karthik_ram_sir-work\IDS_lab\Unit_3\exp1>
```

Name	Date modified	Type	Size
johari	13-09-2023 07:50 AM	Wireshark capture ...	18 KB

Network Traffic Viewer

Algorithm:

1. You import the pyshark library.
2. You ask the user to enter the name of the PCAP file they want to read.
3. Inside the try block, you attempt to open and read the specified PCAP file using pyshark.FileCapture.
4. You then iterate through the packets in the file using a for loop and print each packet.
5. If an exception occurs during this process, you catch it and print an error message.

```
# Here's a Python code that follows this algorithm:

import pyshark

print("Note: file name has an extention of .pcap")
file_name = input("Enter file name: ")
try:
    capture = pyshark.FileCapture(file_name)
    for packet in capture:
        print(packet)
except Exception as e:
    print(f"Error: {e}")
```

Output:

```
PS H:\MITS-Work\karthik_ram_sir-work\IDS_lab\Unit_3\exp1>  h:; cd 'h:\
rs\Johari\.vscode\extensions\ms-python.python-2023.16.0\pythonFiles\li
py'
Note: file name has an extention of .pcap
Enter file name: johari.pcap
```

```
TCP payload (1 byte)
DATA
Packet (Length: 56)
Layer NULL
:       Family: IP (2)
Layer IP
:       0100 .... = Version: 4
        .... 0101 = Header Length: 20 bytes (5)
        Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
        Total Length: 52
        Identification: 0x9287 (37511)
        010. .... = Flags: 0x2, Don't fragment
        0... .... = Reserved bit: Not set
        .1.. .... = Don't fragment: Set
        ..0. .... = More fragments: Not set
        ...0 0000 0000 0000 = Fragment Offset: 0
        Time to Live: 128
        Protocol: TCP (6)
        Header Checksum: 0x0000 [validation disabled]
        Header checksum status: Unverified
        Source Address: 127.0.0.1
        Destination Address: 127.0.0.1
```

Network log Capturing

Algorithm:

1. **Import Required Modules**:

    - Import necessary Python modules such as os, datetime, and Scapy's sniff and wrpcap for packet capture and file operations.

2. **Define Packet Handler Function**:

- Create a function to handle captured packets (packet_handler in this example).
- Inside the function:
    - Extract relevant information from each packet, such as the timestamp, source IP, destination IP, protocol, and packet length.
    - Format this information into a log entry.
    - Append the log entry to a log file (e.g., "network_logs.txt").

3. **Main Program**:

- Check if the log file ("network_logs.txt") exists. If not, create it and add an initial header ("Network Logs:") to the file.

4. **Packet Capture**:

- Start capturing network packets using Scapy's sniff function.
- Specify the packet handler function (packet_handler) to process each captured packet.
- Use store=False to prevent Scapy from storing packets in memory.
- Monitor for a KeyboardInterrupt (Ctrl+C) to gracefully stop the packet capture.

5. **End of Program**:

- Print a termination message to indicate that the program has finished running.

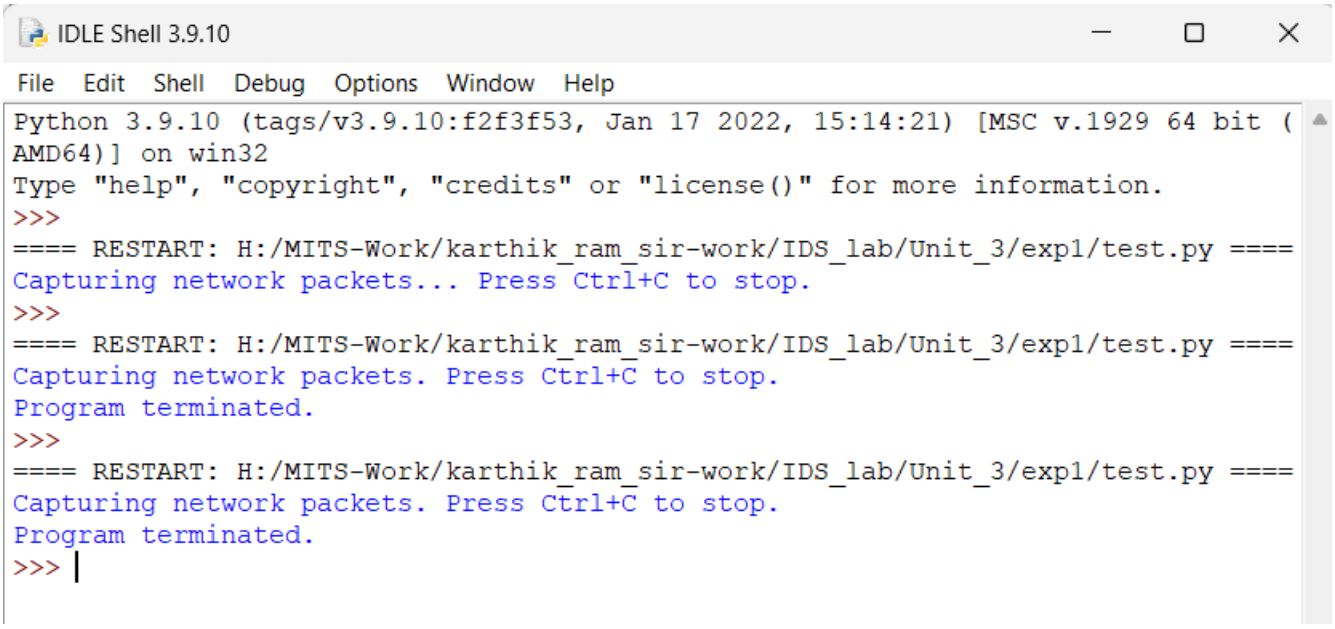
6. **Running the Script**:

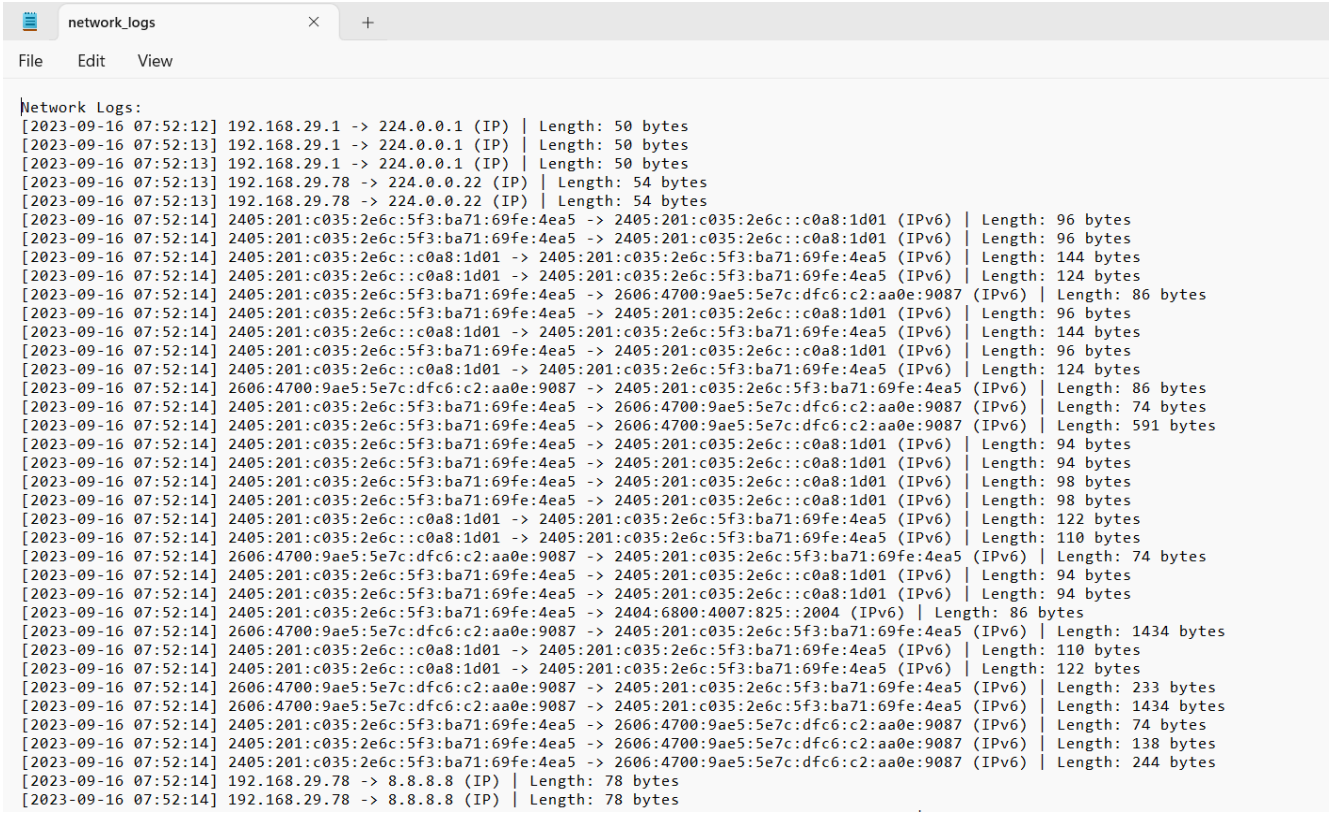
- Ensure Scapy is installed (pip install scapy).
- Save the script to a Python file (e.g., "network_logger.py").
- Execute the script using python network_logger.py.

```python
# Here's a Python code that follows this algorithm:

import os
from scapy.all import sniff, wrpcap
from datetime import datetime

def packet_handler(packet):
    timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    src_ip = packet[0][1].src
    dst_ip = packet[0][1].dst
    protocol = packet[0][1].name
    length = len(packet)

    log_entry = f"[{timestamp}] {src_ip} -> {dst_ip} ({protocol}) | Length: {length} bytes"

    with open("network_logs.txt", "a") as log_file:
        log_file.write(log_entry + "\n")

if __name__ == "__main__":
    log_file_path = "network_logs.txt"
    if not os.path.exists(log_file_path):
        with open(log_file_path, "w") as log_file:
            log_file.write("Network Logs:\n")

    print("Capturing network packets. Press Ctrl+C to stop.")
    try:
        sniff(prn=packet_handler, store=False)
    except KeyboardInterrupt:
        print("Packet capture stopped. Saving logs to 'network_logs.txt'.")

    print("Program terminated.")
```

Output:

- The program will capture network packets in real-time.
- For each packet captured, it will extract relevant information and log it to "network_logs.txt."
- Press Ctrl+C to stop the packet capture.
- The log entries in "network_logs.txt" will contain detailed information about each captured packet.





Resource source:

https://drive.google.com/drive/folders/1RZtMP50ZZBOALFhL3_dsn4Ft2_kPYvv4?usp=drive_link

Result: Thus, to Write a program that reads network traffic data or log files. Successfully.

Experiment No: 2

IMPLEMENT A PROGRAM TO MANAGE A SIGNATURE DATABASE

Aim: Implement a program to manage a signature database.

Tools Required:

- Jupyter Notebook
- pandas (pip install pandas)
- scapy (pip install scapy)

Resource:

Jupyter Notebook File

(https://drive.google.com/file/d/1-cGXOeED4Jcl7U6I7lJU9dK6csdqualr/view?usp=drive_link)

Sample .pcap

(https://drive.google.com/drive/folders/1Mg-XPWOiN2itMbGmv7RF6wopMKsVC697?usp=drive_link)

Pdf & html File

(https://drive.google.com/drive/folders/1zuHHbuO4E6fcOvtSR9iCO0QU37s9vPf7?usp=drive_link)

Algorithm:

1. Import Libraries:

- Import the necessary libraries: `pandas` for data manipulation and `scapy` for reading .pcap files.

2. Process .pcap File (Function: `process_pcap(file_path)`):

- Read the .pcap file using `rdpcap()` function from the `scapy` library.
- Iterate through each packet in the .pcap file.
- For packets with an "IP" layer:
- Extract source IP address, destination IP address, and protocol number.
- Add the extracted information to a list.
- Create a DataFrame using `pandas` containing columns: "Source IP", "Destination IP", and "Protocol".
- Return the DataFrame for further processing.

3. Define Signature-Based Detection Rules (Function: `detect_attacks(packet_df)`):

- Implement detection rules:

- Identify SSH attacks by filtering packets with Protocol 6 (TCP) and print the detected SSH attacks.
- Identify HTTP attacks by filtering packets with Protocol 17 (UDP) and print the detected HTTP attacks.
- Additional rules can be added based on specific use cases.

4. Main Function (Function: `main()`):

- Specify the path to the UNB ISCX IDS 2012 .pcap file.
- Call `process_pcap()` function to extract packet information and create a DataFrame.
- Call `detect_attacks()` function to apply detection rules on the DataFrame and print detected attacks.

5. Execution (Condition: `if __name__ == "__main__":`):

- Execute the `main()` function when the script is run.

Note:

- The program currently focuses on SSH and HTTP traffic detection as per the specified rules.
- Additional rules and more complex logic can be implemented for a more comprehensive intrusion detection system.

Program:

```
# Here's a Python code that follows this algorithm:

import pandas as pd
from scapy.all import rdpcap

# Step 1: Read .pcap file and extract relevant information
def process_pcap(file_path):
    packets = rdpcap(file_path)
    packet_list = []

    for packet in packets:
        if packet.haslayer("IP"):
            src_ip = packet["IP"].src
            dst  ip = packet["IP"].dst
            protocol = packet["IP"].proto
            packet_list.append((src_ip, dst_ip, protocol))

    # Create a DataFrame for easier manipulation
    df = pd.DataFrame(packet_list, columns=["Source IP", "Destination IP", "Protocol"])
    return df

# Step 2: Define signature-based detection rules (example rules)
def detect_attacks(packet_df):
    # Example rules: detecting SSH and HTTP traffic
```

```
    ssh_attacks = packet_df[packet_df["Protocol"] == 6]  # Protocol 6 is TCP (SSH uses TCP)
    http_attacks = packet_df[packet_df["Protocol"] == 17]  # Protocol 17 is UDP (HTTP uses UDP)

    # You can add more rules based on your specific use case

    # Print detected attacks (for demonstration purposes)
    print("Detected SSH Attacks:")
    print(ssh_attacks)
    print("\nDetected HTTP Attacks:")
    print(http_attacks)

# Step 3: Main function
def main():
    # Replace 'your_file_path.pcap' with the actual path to your UNB ISCX IDS 2012 .pcap file
    pcap_file_path = 'testbed-12jun.pcap'
    packet_df = process_pcap(pcap_file_path)
    detect_attacks(packet_df)

if __name__ == "__main__":
    main()
```

Output:

```
Detected SSH Attacks:
            Source IP Destination IP  Protocol
0       192.168.1.101  192.168.5.122         6
1       192.168.5.122  192.168.1.101         6
2       192.168.5.122  192.168.1.101         6
3       192.168.5.122  192.168.1.101         6
4       192.168.5.122  192.168.1.101         6
...               ...            ...       ...
245937  97.74.104.201  192.168.4.121         6
245938  97.74.104.201  192.168.4.121         6
245939  192.168.4.121  97.74.104.201         6
245940  97.74.104.201  192.168.4.121         6
245941  97.74.104.201  192.168.4.121         6

[239632 rows x 3 columns]

Detected HTTP Attacks:
            Source IP Destination IP  Protocol
54      192.168.4.119  192.168.4.255        17
88      192.168.4.119  192.168.5.122        17
89      192.168.5.122   198.164.30.2        17
98       198.164.30.2  192.168.5.122        17
99      192.168.5.122  192.168.4.119        17
...               ...            ...       ...
245726  192.168.2.108  192.168.5.122        17
245727  192.168.5.122   198.164.30.2        17
245761   198.164.30.2  192.168.5.122        17
245762  192.168.5.122  192.168.2.108        17
245763  192.168.5.122  192.168.2.108        17

[6289 rows x 3 columns]
```

Result: Thus, to Implement a program to manage a signature database is Successfully Executed.

UNIT – 4

Experiment No: 1

Aim: Remove outliers from a dataset using z-score or modified z-score and perform feature scaling and normalization on a dataset.

Tools Required:

- **Jupyter notebook**
- **Python packages:**
  - **Pandas (**pip install pandas**)**
  - **Numpy (**pip install numpy**)**
- **Resource:**
  - Jupyter Notebook File

(https://drive.google.com/file/d/1vACZqj82lFqZ3H3qgpXbglL7RnCNBkNN/view?usp=drive_link)

  - Sample dataset

(https://drive.google.com/drive/folders/1oCE1Yd9Ww4daprRjPUn1TW_4Abbb20HB?usp=drive_link)

  - Pdf & html File

(https://drive.google.com/drive/folders/1aBh5s45VvWz4eqnDZ9_yAKveP3pq6gAQ?usp=drive_link)

Algorithm:

Step 1: Import the necessary libraries.

Step 2: Load the dataset.

Step 3: Remove outliers using Z-score.

Step 4: Encode categorical variables using one-hot encoding.

Step 5: Separate features and target variable after one-hot encoding.

Step 6: Perform feature scaling using StandardScaler.

Step 7: Perform feature normalization using MinMaxScaler.

Step 8: Display the processed data (optional, for visualization purposes)

Program:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Load the dataset
# Replace 'kddcup.data_10_percent_corrected' with the path to your downloaded dataset file.
data = pd.read_csv('kddcup.data_10_percent_corrected', header=None)

# Assign meaningful column names to the dataset (you can find these in the dataset description)
columns = [
    "duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes", "land",
    "wrong_fragment", "urgent", "hot", "num_failed_logins", "logged_in", "num_compromised",
    "root_shell", "su_attempted", "num_root", "num_file_creations", "num_shells",
    "num_access_files", "num_outbound_cmds", "is_host_login", "is_guest_login",
    "count", "srv_count", "serror_rate", "srv_serror_rate", "rerror_rate", "srv_rerror_rate",
    "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate", "dst_host_count",
    "dst_host_srv_count", "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
    "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate",
    "dst_host_srv_serror_rate", "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "attack_type"
]
data.columns = columns

# Display the first few rows of the dataset to understand its structure
data.head()
```

```
# Calculate Z-scores for numerical columns
numerical_cols = data.select_dtypes(include=[np.number]).columns
z_scores = np.abs((data[numerical_cols] - data[numerical_cols].mean()) / data[numerical_cols].std())

# Define a threshold for Z-score (e.g., 3) to identify outliers
threshold = 3
outliers = (z_scores > threshold).any(axis=1)

# Remove outliers from the dataset
data = data[~outliers]
```

```
# Encode categorical variables using one-hot encoding
data_encoded = pd.get_dummies(data, columns=['protocol_type', 'service', 'flag'])


# Separate features and target variable after one-hot encoding
X_encoded = data_encoded.drop("attack_type", axis=1)
y_encoded = data_encoded["attack_type"]

# Perform feature scaling using StandardScaler
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

# Perform feature normalization using MinMaxScaler
min_max_scaler = MinMaxScaler()
X_normalized = min_max_scaler.fit_transform(X_scaled)
```

```
# Display the processed data
processed_data = pd.DataFrame(X_normalized, columns=X_encoded.columns)
processed_data.head()
```

Output:

- Display Original dataset.

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromised	...	service_vmnet	service_whois	flag_REJ	flag_RSTO	flag_RSTR	flag_S0	flag_S1	flag_S2	flag_S3	flag_SF
0	0.0	0.000100	0.013819	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.000099	0.021002	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	0.0	0.000099	0.021002	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.000117	0.008455	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.000106	0.002636	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 114 columns

- Removed outliers from a dataset using z-score

Out[1]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate	dst_host_
0	0	tcp	http	SF	181	5450	0	0	0	0	...	9	1.0	0.0	0.11	0.0	
1	0	tcp	http	SF	239	486	0	0	0	0	...	19	1.0	0.0	0.05	0.0	
2	0	tcp	http	SF	235	1337	0	0	0	0	...	29	1.0	0.0	0.03	0.0	
3	0	tcp	http	SF	219	1337	0	0	0	0	...	39	1.0	0.0	0.03	0.0	
4	0	tcp	http	SF	217	2032	0	0	0	0	...	49	1.0	0.0	0.02	0.0	

5 rows × 42 columns

Result: Thus, to remove outliers from a dataset using z-score or modified z-score and perform feature scaling and normalization on a dataset is Successfully Executed.

UNIT – 4

Experiment No: 2

Aim: Apply moving average or exponential smoothing techniques to detect anomalies in a time series dataset.

Tools Required:

- **Jupyter notebook**
- **Python packages:**
  - **Pandas (**pip install pandas**)**
  - **Numpy (**pip install numpy**)**
- **Resource:**
  - Jupyter Notebook File

(https://drive.google.com/file/d/1u-vsa9DXoV2CJj1Efs_kEUQFPS0M2JBT/view?usp=drive_link)

  - Sample dataset

(https://drive.google.com/drive/folders/1Uno8hTLxhcZPVsnkEkr_QiXfIik-XMbz?usp=drive_link)

  - Pdf & html File

(https://drive.google.com/drive/folders/1ojoSzhsBhAaHjyTaosUqHMXsQjiRap6q?usp=drive_link)

Algorithm:

Step 1: Import the necessary libraries.

Step 2: Load the Network Traffic Dataset.

Step 3: Calculate Moving Average

Calculate moving average of the network traffic data using a specified window size (e.g., 10 minutes).

Step 4: Calculate Exponential Smoothing

Calculate exponential smoothing of the network traffic data using a specified smoothing factor (e.g., 0.2).

Step 5: Detect Anomalies using Moving Average

Detect anomalies based on the difference between the actual traffic volume and the moving average. Define a threshold (e.g., two times the standard deviation) to identify anomalies.

Step 6: Detect Anomalies using Exponential Smoothing

Detect anomalies based on the difference between the actual traffic volume and the exponential smoothing. Define a threshold (e.g., two times the standard deviation) to identify anomalies.

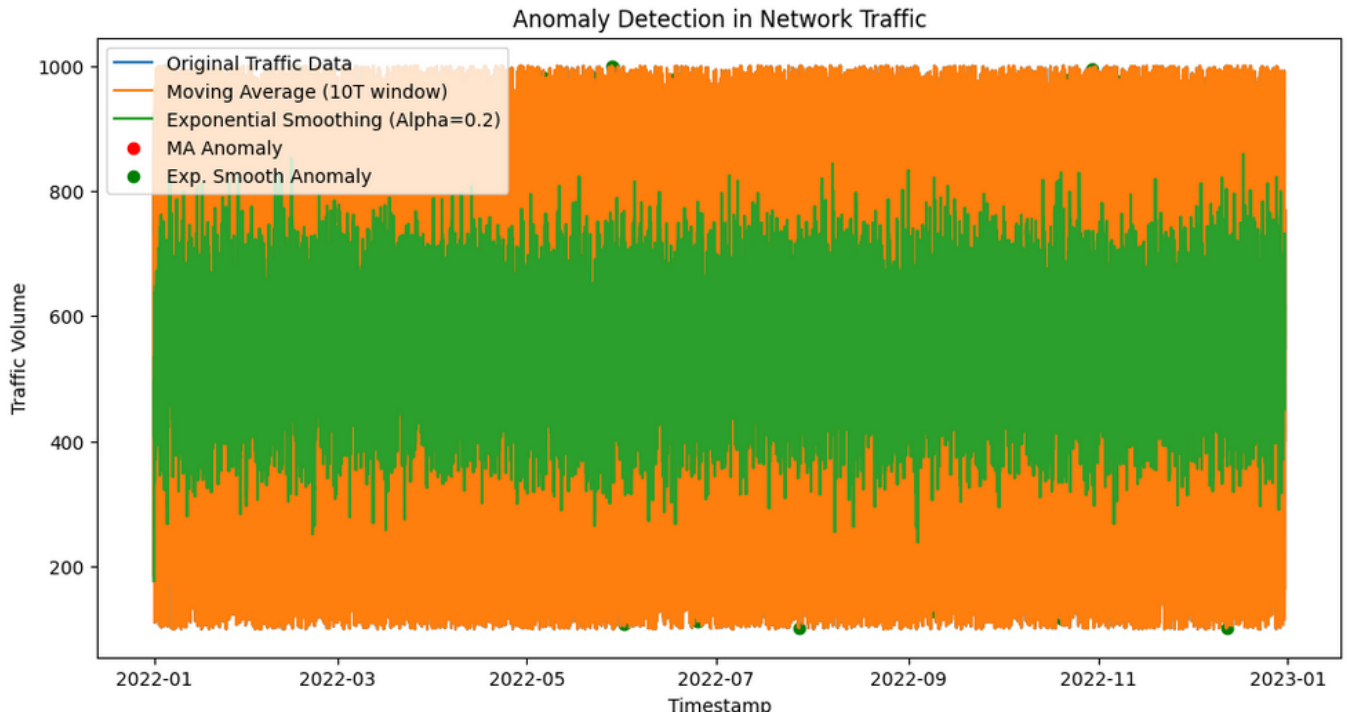
Step 7: Visualize Results

Plot the original network traffic data, moving average, and exponential smoothing. Highlight detected anomalies using different colors.

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the network traffic dataset (replace 'network_traffic.csv' with your dataset file)
df = pd.read_csv('network_traffic.csv', parse_dates=True, index_col=0)

# Calculate moving average with window size 10 minutes (adjust window size based on your data)
window_size = '10T'  # 10 minutes
df['MA'] = df['Traffic'].rolling(window=window_size).mean()

# Calculate exponential smoothing with smoothing factor (adjust alpha based on your data)
alpha = 0.2
df['Exp_Smooth'] = df['Traffic'].ewm(alpha=alpha, adjust=False).mean()

# Detect anomalies using moving average
ma_std = df['Traffic'].std() * 2  # Adjust the multiplier according to the dataset
df['MA_Anomaly'] = np.abs(df['Traffic'] - df['MA']) > ma_std

# Detect anomalies using exponential smoothing
exp_smooth_std = df['Traffic'].std() * 2  # Adjust the multiplier according to the dataset
df['Exp_Smooth_Anomaly'] = np.abs(df['Traffic'] - df['Exp_Smooth']) > exp_smooth_std

# Plot the original traffic data, moving average, and exponential smoothing
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Traffic'], label='Original Traffic Data')
plt.plot(df.index, df['MA'], label=f'Moving Average ({window_size} window)')
plt.plot(df.index, df['Exp_Smooth'], label=f'Exponential Smoothing (Alpha={alpha})')
plt.scatter(df.index[df['MA_Anomaly']], df['Traffic'][df['MA_Anomaly']], color='red', label='MA Anomaly')
plt.scatter(df.index[df['Exp_Smooth_Anomaly']], df['Traffic'][df['Exp_Smooth_Anomaly']], color='green', label='Exp.
Smooth Anomaly')
plt.xlabel('Timestamp')
plt.ylabel('Traffic Volume')
plt.title('Anomaly Detection in Network Traffic')
plt.legend()
plt.show()
```

Output:



Result: Thus, to apply moving average or exponential smoothing techniques to detect anomalies in a time series dataset is Successfully Executed.

UNIT – 5

Experiment No: 1

Aim: Measure the IDS response time under different traffic loads and analyze the performance metrics.

Tools Required:

- **Jupyter notebook**
- **Python packages:**
    - **Pandas (**pip install pandas**)**
- **Resource:**
    - Jupyter Notebook File

(https://drive.google.com/file/d/1tt9nVwg1O1orZW2HH9H27zOkZcnLDyGa/view?usp=drive_link)

- Sample dataset

(https://drive.google.com/drive/folders/1GNhgY8FYeMMN-8_8_UilZI-hxvn_f3Sb?usp=drive_link)

- Pdf & html File

(https://drive.google.com/drive/folders/1wfEYcuPEPZoulzl7AYA2WL1RmpUNWmqJ?usp=drive_link)

Algorithm:

Step 1: Import Necessary Libraries.

Step 2: Load the Sample Dataset.

Step 3: Train the Random Forest Classifier.

Step 4: Make Predictions on the Test Set.

Step 5: Evaluate the Performance.

Step 6: Interpret the Results.

- The accuracy score represents the proportion of correctly classified instances in the test set.
- The classification report provides detailed metrics such as precision, recall, and F1-score for both classes (normal and intrusion). It helps you understand the model's performance for each class.

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report


# Load the dataset
data = pd.read_csv("ids_dataset.csv")

# Split features and labels
X = data.drop('label', axis=1)
y = data['label']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the classifier
clf = RandomForestClassifier(random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test set
predictions = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy:.2f}")

# Generate and print a classification report
print("Classification Report:")
print(classification_report(y_test, predictions))
```

Output:

```
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1006
         1.0       1.00      1.00      1.00       194

    accuracy                           1.00      1200
   macro avg       1.00      1.00      1.00      1200
weighted avg       1.00      1.00      1.00      1200
```

Result: Thus, to measure the IDS response time under different traffic loads and analyze the performance metrics is Successfully Executed.

UNIT – 5

Experiment No: 2

Aim: Analyze the IDS alerts generated during detection testing to identify false positives.

Tools Required:

- **Jupyter notebook**
- **Python packages:**
    - **Pandas (**pip install pandas**)**
- **Resource:**
    - Jupyter Notebook File

(https://drive.google.com/file/d/1HPX_kvbfj8OI23hU1_ISP_gqWNH-D6Zn/view?usp=drive_link)

    - Sample dataset

(https://drive.google.com/drive/folders/1dkVDDgK8I6cBDN5Ayt03XUAcT46Q5OqZ?usp=drive_link)

    - Pdf & html File

(https://drive.google.com/drive/folders/1P9Y3Vx3iDFilTMmNDBSmTMBkcJNA5fOI?usp=drive_link)

Algorithm:

Step 1: Get the dataset.

Step 2: Import Required Libraries.

Step 3: Load the Dataset.

- Load the IDS alerts dataset from the CSV file into a Pandas DataFrame.

Step 4: Filter False Positives.

- Filter the DataFrame to identify false positives (where is_intrusion is True).

Step 5: Print False Positives.

- Print the false positives to the Jupyter Notebook output.

Step 6: Visualize Alert Types of False Positives.

- Create a bar chart to visualize the distribution of alert types for false positives.

Step 7: Run the Notebook.

- Execute the Jupyter Notebook cells one by one. Make sure to have the sample dataset (ids_alerts.csv) in the same directory as your Jupyter Notebook file.

Program:

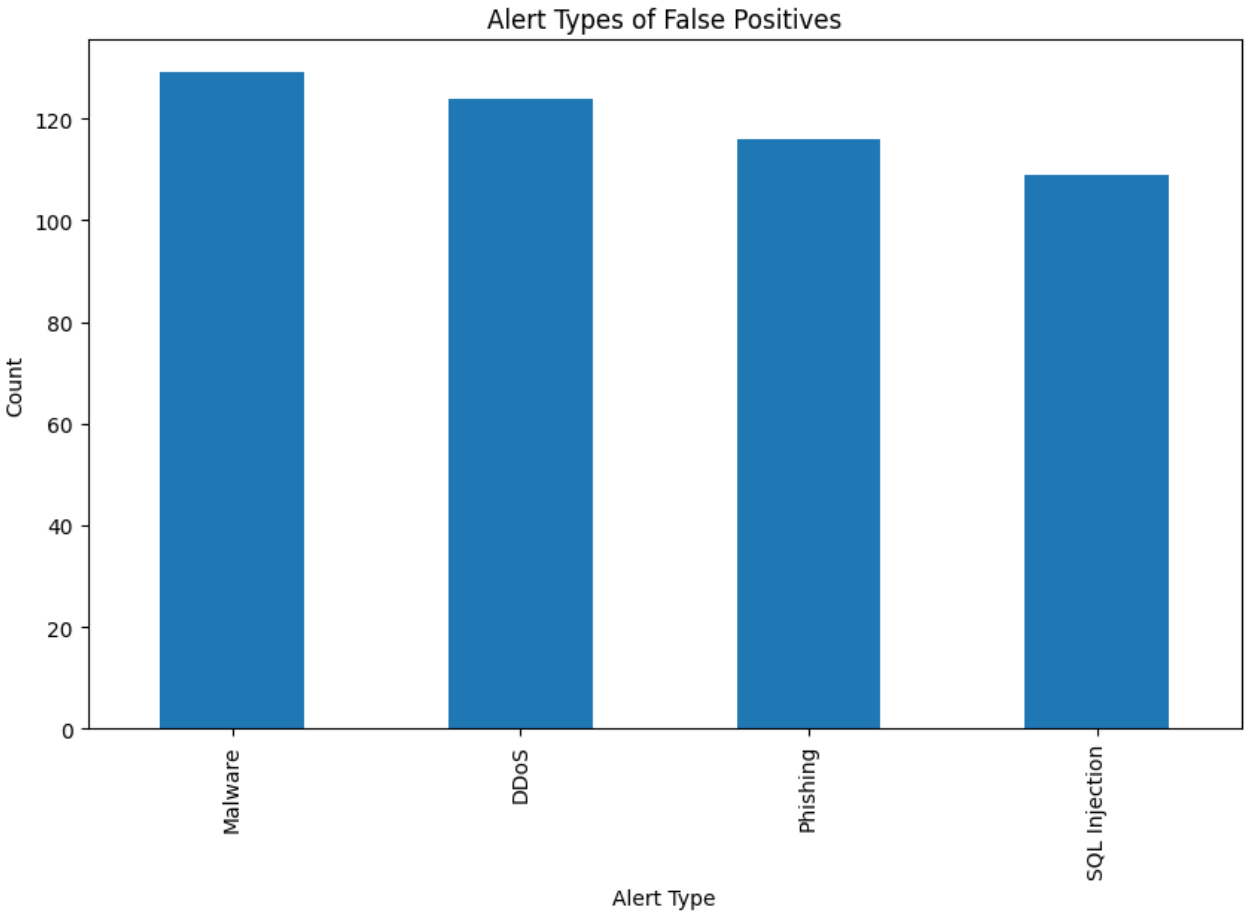
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('ids_alerts.csv')

# Analyze false positives
false_positives = df[df['is_intrusion'] == True]

# Print false positives
print("False Positives:")
print(false_positives)

# Visualize alert types of false positives
alert_type_counts = false_positives['alert_type'].value_counts()
alert_type_counts.plot(kind='bar', figsize=(10, 6))
plt.title('Alert Types of False Positives')
plt.xlabel('Alert Type')
plt.ylabel('Count')
plt.show()
```

Output:

- **false positives**

```
False Positives:
                    timestamp       source_ip   destination_ip  \
1    2023-10-25 13:35:15.810533    68.93.22.196    172.28.235.49
5    2023-10-10 13:35:15.810533   1.157.232.205   238.137.21.104
6    2023-10-17 13:35:15.810533   128.79.175.25    178.199.45.19
7    2023-11-02 13:35:15.810533    16.128.31.92     82.34.142.62
8    2023-10-21 13:35:15.810533  53.144.110.118     31.21.116.12
..                          ...             ...              ...
987  2023-10-28 13:35:15.810533  118.128.237.19  254.237.153.134
988  2023-10-30 13:35:15.810533  117.221.211.14      53.25.56.25
990  2023-10-25 13:35:15.810533   179.228.17.57    34.194.80.181
992  2023-10-19 13:35:15.810533  218.110.92.252    224.33.227.82
997  2023-10-23 13:35:15.810533   249.56.33.239     69.39.205.50

        alert_type  is_intrusion
1             DDoS          True
5          Malware          True
6             DDoS          True
7          Malware          True
8             DDoS          True
..              ...           ...
987        Malware          True
988  SQL Injection          True
990  SQL Injection          True
992       Phishing          True
997  SQL Injection          True

[478 rows x 5 columns]
```

- **Visualize alert types of false positives.**



Result: Thus, to analyze the IDS alerts generated during detection testing to identify false positives is Successfully Executed.

Note all Resource of IDS

(https://drive.google.com/drive/folders/1rt-qFrAc0SevlKEYlldwlEglV8NRI70f?usp=drive_link)