FCS Question 4

1)Internet Protocol Version 6 is a network layer protocol which was developed as a successor to the IPv4 though it has not been fully adopted yet.It uses a 128 bit address space and was conceived to counter the exhaustion of 32 bit IPv4 address space.
Key Difference with IPv4:
a)**Address Space**:IPv6 has a 128 bit address space against IPv4 with a 32 bit address space.
b)**Address Notation:**IPv6 is written in hexadecimal notation as compared to IPv4 in decimal notation eg. 2001::aff2:e580:0370:8344(IPv6) and 192.168.1.130(IPv4)
c)**NATting:** Network Address Translation is a key feature of IPv4 due to the presence of both private and public IP addresses. IPv6 has no concept of public and private address so NAT adapters are not required.
d)**Header Format:**IPv4 headers are variable in length leading to overhead and making header processing complex while IPv6 headers have a fixed size which reduces overhead.
e)**IPSec:**IPSec provides for IPSec protocol by default providing greater security than IPv4.
2)
Benefits of IPv6 over IPv4:
a)**IP Spoofing:** Since IPv6 does not require public and private IP address, the router/gateway does not change the IP address of packets from private to public. This eliminates a core vulnerability as it makes IP spoofing harder. Now source IP could be verified by a checksum.
b)**IPSec**: IPv6 makes IPSec mandatory which comes encryption of message and an authentication header thus improving security as compared to IPv4 where IPSec is only optional.
c) **Elimination of IP fragmentation:**By avoiding fragmentation of Ip packets IPv6 mitigates fragmentation based attacks.

Challenges of IPv6 vs IPv4:
a)**Absence of NAT:**While the absence of NAT may be a security benefit in some cases it makes our IP address visible to the entire Internet.
b)**Extension Header Attack:** Extension Headers in IPv6 are chained just like linked lists.An attacker can easily attack the chain of extension layers and get through the header for deep packet inspection
c**)Reconnaissance Attack:** The attacker may do some host probing and port scanning to find the weakest link of the host. IPv6 allows the attacker to find the router and DHCP server.Since IPv6 header is so large it also becomes tough to identify the attacker.
d)**Transition Phase:** Since most devices in the network core are legacy IPv4 devices, the dual IPv4/IPv6 stack would create many security risks and vulnerabilities during transition from IPv4 to IPv6.
3)
We use nmap to find open ports on the given IPv6 host.I have chosen host with IPv6 address-
fe80::f55e:ddc6:6352:aff2

```
target_ipv6 = "fe80::f55e:ddc6:6352:aff2"
# target_ipv6_range="fe80::"
scan_results=scan_network(target_ipv6)
```

This is the function to scan the network for the given IPv6 host. Here -6 is to specify that the given address is IPv6 and -sV indicates a verbose output.

```
def scan_network(ip):
    nm = nmap.PortScanner()
    nm.scan(ip, arguments='-6 -sV',timeout=600)
    return nm
```

We analyze the nmap.PortScanner object obtained to check for open ports. If we find that the protocol['state'] is set to open in the dictionary then the port is printed.

```
def analyze_scan_results(ip,scan_results):
    print("Open Ports:")
    # print(scan_results[host])
    for port, protocol in scan_results[ip]['tcp'].items():
        if protocol['state'] == 'open':
            print(f"  Port {port}  {protocol['name']} is open")
```

Output:
We find that ports 22(ssh) and 80(http) are open in the host.

```
root@Ubuntu2:/home/arnav/Documents# python3 Vulnerability.py
<nmap.nmap.PortScanner object at 0x7f6d31f67f40>
Open Ports:
  Port 22  tcpwrapped is open
  Port 80  tcpwrapped is open
```

4)

We create a client socket using the socket module in python and connect to the server using the IP address and port number of the server. We sent the message Hello, Server! To the server. The Exception handling done includes Connection error,Socket error and general exception. Finally the socket is closed.

Note:We send message.encode() instead of the message itself since we need to send a byte-like object instead of a string.

```
import socket


try:

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


    client_socket.connect(('192.168.56.121', 80))


    message = "Hello, Server!"

    client_socket.send(message.encode())


except ConnectionError as e:
    print(f"Error Connecting: {e}")
except socket.error as e:
    print(f"Some error in socket {e}")
except Exception as e:
    print(e)
finally:
    try:
        client_socket.close()
    except Exception as e:
```

5)Server program to send response to client and a client program to handle this response:

Client:

```python
import socket
import hashlib

try:
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('192.168.56.121', 80))
    message = "Hello, Server!"
    client_socket.send(message.encode())
    response = client_socket.recv(512).decode()
    if(response=="Hello, Client!"):
        print("Server says:", response)
        print("Server received message successfully")
    else:
        print(f"Server ran into an error: {response}")
except ConnectionError as e:
    print(f"Error Connecting: {e}")
except socket.error as e:
    print(f"We encountered a socket error:{e}")
except Exception as e:
    print(e)
finally:
    try:
        client_socket.close()
    except socket.error as e:
        print(f"We encountered a socket error trying to close the socket:{e}")
```

We receive the server response and check if the response was Hello, Client!. If yes the response returned to stdout and the function exits. Else we raise an error message.

Server:

```python
import socket

try:
    server_address = ('192.168.56.121', 80)
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(server_address)
    server_socket.listen(1)
    print("Server is listening for incoming connections...")

    while True:
        client_socket, client_address = server_socket.accept()
        print(f"Accepted connection from {client_address}")
        data = client_socket.recv(1024).decode()
        response="Hello, Client!"
        client_socket.send(response.encode())
        print("Response sent to client.")
except ConnectionError as e:
    print(f"We ran into a connection error{e}")
except socket.error as e:
    print(f"We ran into a socket error:{e}")
except Exception as e:
    print(e)
finally:
    try:
        server_socket.close()
    except socket.error as e:
```

We create a server socket and bind it onto the server IP address. Now we listen for TCP connection on this socket. We accept the connection from the given client address and decode the message received. The server responds with Hello, Client! And sends response to client.

Output:
Server:

```
root@ComputerNetworks:/home/arnav/Documents/Python#
Server is listening for incoming connections...
Accepted connection from ('192.168.56.1', 18508)
Response sent to client.
```

Client:

```
PS C:\Users\ARNAV\Documents\FCS> python -u "c:\Users\ARNAV\Documents\FCS\TCPConn2.py"
Server says: Hello, Client!
```

6)
Client:

```
try:
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('192.168.56.121', 80))
    message = "Hello, Server!"
    message_hash = hashlib.sha256(message.encode("utf-8")).hexdigest()
    data_to_send = f"{message}|{message_hash}"
    client_socket.send(data_to_send.encode())
    response = client_socket.recv(512).decode()
    if(response=="Hello, Client!"):

        print("Server says:", response)
        print("Server received message successfully")
    else:
        print(f"Server ran into an error: {response}")
```

Here we also compute the hash of the message we are sending using the hashlib library and sha256. The message and hash are combined into a single message using a delimiter(|). This data is encoded and sent.

Server:

```
try:
    server_address = ('192.168.56.121', 80)
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(server_address)
    server_socket.listen(1)
    print("Server is listening for incoming connections...")
    while True:
        client_socket, client_address = server_socket.accept()
        print(f"Accepted connection from {client_address}")
        data = client_socket.recv(1024).decode()
        message, received_hash = data.split('|')
        computed_hash = hashlib.sha256(message.encode("utf-8")).hexdigest()

        if received_hash == computed_hash and message=="Hello, Server!":
            response = "Hello, Client!"
        else:
            if(received_hash != computed_hash):
                response = "Hash values do not match. Message may be corrupted."
            else:
                response=f"Message sent may be wrong: {message}"
        client_socket.send(response.encode())
        print("Response sent to client.")
```

On the server side we split the data received from client using the delimiter(|). Now we compute the hash of the message received using the sha256 from hashlib library. If the received and computed hash are equal and the message is “Hello, Server!” a success message is sent to the client. Else we send back an error message.

Output:

Client:

```
PS C:\Users\ARNAV\Documents\FCS> python -u c:\Users\ARNAV\Documents\FCS\TCPconn.py
THe data sent is: Hello, Server!|5edeb8781824c4abfcc07748f1142ffdf25a0417cd44e7aa3caca68139a341d1
Server says: Hello, Client!
Server received message successfully
```

Server:

```
Password:
root@ComputerNetworks:/home/arnav/Documents/Python# python3 TCpconn.py
Server is listening for incoming connections...
Accepted connection from ('192.168.56.1', 18755)
Response sent to client.
```