Indian Institute of Technology Jodhpur

CSL6010 – Cyber Security
LAB Report

Name- Aman Srivastava
Roll No.- B20CS100

Date-12th Mar 2023

1. Packet capturing:

After starting a packet capture in Wireshark wireless interface, I observed a continuous stream of data packets being captured and displayed on the screen in real time. These packets could be related to various types of network activity, such as web browsing, file transfers, email. I can also see the source and destination IP addresses. There are also see some packets that look unusual or corrupted, which could indicate network or device issues that need to be investigated further.

2. Packet capturing on visiting IITJ website:

Following is the DNS request.

1520	14.555520	172.31.24.13	172.16.100.205	DNS	74 Standard query 0x81b8 A rnd.iitj.ac.in
1521	14.562057	172.16.100.205	172.31.24.13	DNS	124 Standard query response 0x81b8 A rnd.iitj.ac.in A 172.16.100.121 NS dns.iitj.ac.in A 172.16.100.205

IP address of IITJ server is 172.31.24.13
Below is the TCP and HTTP request and acknowledge.

1526	14.607347	172.31.24.13	172.16.100.121	TCP	66 59406 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1527	14.611093	172.16.100.121	172.31.24.13	TCP	66 80 → 59406 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM WS=128
1528	14.611245	172.31.24.13	172.16.100.121	TCP	54 59406 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
1529	14.611605	172.31.24.13	172.16.100.121	HTTP	402 GET / HTTP/1.1
1530	14.615873	172.16.100.121	172.31.24.13	TCP	54 80 → 59406 [ACK] Seq=1 Ack=349 Win=30336 Len=0
1531	14.617247	172.16.100.121	172.31.24.13	HTTP	542 HTTP/1.1 302 Found (text/html)
1532	14.669139	172.31.24.13	172.16.100.121	TCP	54 59406 → 80 [ACK] Seq=349 Ack=489 Win=130816 Len=0

Once the TCP connection is established, The device send an HTTP request to the server requesting the webpage. The server will respond with an HTTP response containing the HTML code for the webpage.

3. Packets highlighted in black colour:

Packets in black color signify Bad TCP, HSRP state change, spanning tree topology change, IMCP errors.

<table>
<tr><td>Bad TCP</td><td>tcp.analysis.flags && !tcp.analysis.window_update && !tcp.analysis.keep_alive && !tcp.analysis.keep_alive_ack</td></tr>
<tr><td>HSRP State Change</td><td>hsrp.state != 8 && hsrp.state != 16</td></tr>
<tr><td>Spanning Tree Topology Change</td><td>stp.type == 0x80</td></tr>
<tr><td>OSPF State Change</td><td>ospf.msg != 1</td></tr>
<tr><td>ICMP errors</td><td>icmp.type in { 3..5, 11 } || icmpv6.type in { 1..4 }</td></tr>
</table>

4. Filters in wireshark:

1) **http:** This filter displays only the HTTP traffic in the capture. This can be useful for analyzing web traffic, such as requests and responses.

http

No.	Time	Source	Destination	Protocol	Length	Info
1621	13.749969	10.23.0.112	23.46.187.137	HTTP	431	GET /roots/dstrootcax3.p7c HTTP/1.1
1634	13.795346	23.46.187.137	10.23.0.112	HTTP	324	HTTP/1.1 304 Not Modified
18884	60.734141	10.23.0.139	10.23.0.112	HTTP/X…	904	POST /a6b1811f-d512-426d-bde6-ec9da5acc44f/ HTTP/1.1
18886	60.737332	10.23.0.112	10.23.0.139	HTTP/X…	937	HTTP/1.1 200
23253	73.986333	10.23.0.112	172.16.100.160	HTTP	568	GET /Aryabhatta_New/ HTTP/1.1
23270	74.086242	10.23.0.112	172.16.100.160	HTTP	572	GET /Aryabhatta_New/style.css HTTP/1.1
23278	74.113726	172.16.100.160	10.23.0.112	HTTP	1013	HTTP/1.1 200 (text/html)
23285	74.144159	10.23.0.112	172.16.100.160	HTTP	625	GET /Aryabhatta_New/DSC_0362.JPG HTTP/1.1

2) **ip.addr:** This filter displays only the packets that originate from or are destined for a specific IP address. This can be useful for identifying traffic from a particular host or network.

ip.addr

No.	Time	Source	Destination	Protocol	Length	Info
37678	75.319693	172.16.100.160	10.23.0.112	TCP	1514	8080 → 65226 [ACK] Seq=13897968 Ack=572 Win=64128 Len=1460 [TCP
37679	75.319708	10.23.0.112	172.16.100.160	TCP	54	65226 → 8080 [ACK] Seq=572 Ack=13899428 Win=131328 Len=0
37680	75.319933	172.16.100.160	10.23.0.112	TCP	1514	8080 → 65226 [ACK] Seq=13899428 Ack=572 Win=64128 Len=1460 [TCP
37681	75.319933	172.16.100.160	10.23.0.112	TCP	1514	8080 → 65226 [ACK] Seq=13900888 Ack=572 Win=64128 Len=1460 [TCP
37682	75.319956	10.23.0.112	172.16.100.160	TCP	54	65226 → 8080 [ACK] Seq=572 Ack=13902348 Win=131328 Len=0
37683	75.320192	172.16.100.160	10.23.0.112	TCP	1514	8080 → 65226 [PSH, ACK] Seq=13902348 Ack=572 Win=64128 Len=1460

3) **tcp.port:** This filter displays only the packets that use a specific TCP port. This can be useful for identifying traffic related to a particular service or application.

tcp.port==80

No.	Time	Source	Destination	Protocol	Length	Info
1638	13.841419	10.23.0.112	23.46.187.137	TCP	54	65156 → 80 [ACK] Seq=378 Ack=271 Win=131072 Len=0
16617	23.520051	10.23.0.112	23.46.187.137	TCP	54	65156 → 80 [FIN, ACK] Seq=378 Ack=271 Win=131072 Len=0
16642	23.573347	23.46.187.137	10.23.0.112	TCP	60	80 → 65156 [FIN, ACK] Seq=271 Ack=379 Win=64128 Len=0
16643	23.573378	10.23.0.112	23.46.187.137	TCP	54	65156 → 80 [ACK] Seq=379 Ack=272 Win=131072 Len=0

4) **dns:** This filter displays only the DNS traffic in the capture. This can be useful for analyzing domain name resolution and identifying potential DNS issues.

dns

No.	Time	Source	Destination	Protocol	Length	Info
78	2.369108	10.23.0.112	172.16.100.206	DNS	75	Standard query 0x4a0b A wpad.iitj.ac.in
79	2.369266	10.23.0.112	172.16.100.206	DNS	75	Standard query 0x67a7 A wpad.iitj.ac.in
80	2.370256	172.16.100.206	10.23.0.112	DNS	120	Standard query response 0x4a0b No such name A wpad.iitj.ac.in SO
81	2.370256	172.16.100.206	10.23.0.112	DNS	120	Standard query response 0x67a7 No such name A wpad.iitj.ac.in SO

5) **ssl:** This filter displays only the SSL/TLS traffic in the capture. This can be useful for analyzing encrypted traffic, such as HTTPS traffic.

ssl

No.	Time	Source	Destination	Protocol	Length	Info
57	1.100454	52.22.241.30	10.23.0.112	TLSv1.2	174	New Session Ticket
58	1.100454	52.22.241.30	10.23.0.112	TLSv1.2	105	Change Cipher Spec, Encrypted Handshake Message
59	1.100454	52.22.241.30	10.23.0.112	TLSv1.2	132	Application Data
62	1.101052	10.23.0.112	52.22.241.30	TLSv1.2	92	Application Data
66	1.346837	52.22.241.30	10.23.0.112	TLSv1.2	174	New Session Ticket

5. **Filter command for all outgoing traffic:**
   ip.src == 10.23.0.112
   This command filters all traffic with the specified source IP address, which is the host's IP address sending the traffic. This will show all outgoing traffic.

ip.src == 10.23.0.112

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.23.0.112	142.250.194.174	UDP	75	61883 → 443 Len=33
3	0.281475	10.23.0.112	172.217.166.206	UDP	75	62537 → 443 Len=33
5	1.085876	10.23.0.112	13.224.22.71	TCP	55	49288 → 443 [ACK] Seq=1 Ack=1 Win=
7	1.481971	10.23.0.112	142.250.192.195	UDP	127	52710 → 443 Len=85
9	1.531813	10.23.0.112	142.250.192.195	UDP	75	52710 → 443 Len=33
12	1.588026	10.23.0.112	142.250.192.195	UDP	77	52710 → 443 Len=35
13	1.614937	10.23.0.112	142.250.192.195	UDP	75	52710 → 443 Len=33

6. **3-way handshake:**
   3-way handshake on visiting cricinfo.com
   Below is the IPv4 and IPv4 DNS server address of my PC as there is IITJ server in between.

IPv4 address:	10.23.0.112
IPv4 DNS servers:	172.16.100.206 (Unencrypted)

284	5.678047	10.23.0.112	172.16.100.206	DNS	70	Standard query 0x5708 A dns.google
285	5.678434	10.23.0.112	172.16.100.206	DNS	70	Standard query 0x0918 HTTPS dns.google
286	5.679313	172.16.100.206	10.23.0.112	DNS	355	Standard query response 0x5708 A dns.google A 8.8.4.4 A 8.8.8.8 NS r
287	5.679313	172.16.100.206	10.23.0.112	DNS	146	Standard query response 0x0918 HTTPS dns.google SOA ns1.zdns.google

Step 1 (SYN): In the first step, the client wants to establish a connection with a server, so it sends a segment with SYN(Synchronize Sequence Number) which informs the server that the client is likely to start communication and with what sequence number it starts segments with.

383	6.141157	10.23.0.112	13.251.68.88	TCP	66	49295 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM

Step 2 (SYN + ACK): Server responds to the client request with SYN-ACK signal bits set.

387	6.261887	13.251.68.88	10.23.0.112	TCP	66	443 → 49295 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS

Step 3 (ACK): In the final part client acknowledges the response of the server and they both establish a reliable connection with which they will start the actual data transfer

388	6.261950	10.23.0.112	13.251.68.88	TCP	54	49295 → 443 [ACK] Seq=1 Ack=1 Win=131328 Len=0
389	6.262384	10.23.0.112	13.251.68.88	TLSv1.2	571	Client Hello
390	6.263443	13.251.68.88	10.23.0.112	TCP	60	443 → 49295 [ACK] Seq=1 Ack=518 Win=1214720 Len=0

7. **Why DNS uses UDP and HTTP uses TCP?**
   DNS and HTTP are two protocols used on the internet. DNS uses a fast and efficient protocol called UDP, while HTTP uses a reliable data transmission protocol called TCP. UDP is quick because it doesn't need a special connection, while TCP makes sure the data is transmitted correctly. This choice of protocols depends on the kind of data being sent. DNS sends small queries that need fast responses, while HTTP sends larger amounts of data that need to be sent correctly and in the right order.

8. TCP communication in socket programming:

client.py

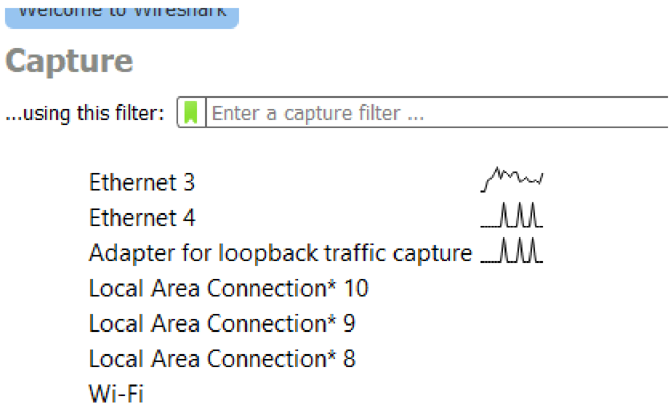
```
import socket

def main():
    host = 'localhost'
    port = 5000

    client_socket = socket.socket()
    try:
        client_socket.connect((host, port))
        while True:
            expr = input("Enter expression: ")
            client_socket.send(expr.encode())
            data = client_socket.recv(1024).decode()
            print("Result: " + data)
    except ConnectionRefusedError:
        print("Another client is already connected to the server. Please try again later.")
    client_socket.close()

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

server.py

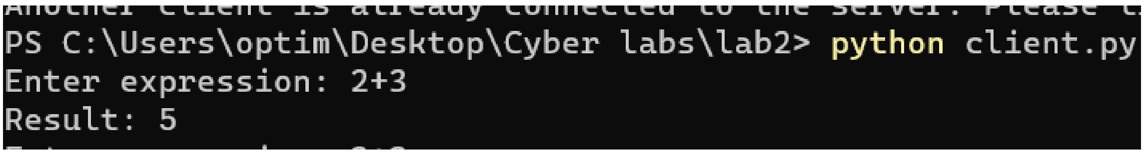
```
import socket
def handle_client(conn, addr):
    print("Connection from: " + str(addr))
    while True:
        data = conn.recv(1024).decode()
        if not data:
            break
        try:
            result = eval(data)
            print("Evaluated Result:",result)
            conn.send(str(result).encode())
        except:
            conn.send("Invalid Expression".encode())
    conn.close()
def main():
    host = 'localhost'
    port = 5000
    server_socket = socket.socket()
    server_socket.bind((host, port))
    server_socket.listen(1)
```

```
    conn, addr = server_socket.accept()
    handle_client(conn, addr)
    conn.close()
    print("Connection closed")
    try:
        conn, addr = server_socket.accept()
    except:
        print("Error: Another client is already connected")
if __name__ == '__main__':
    main()
```

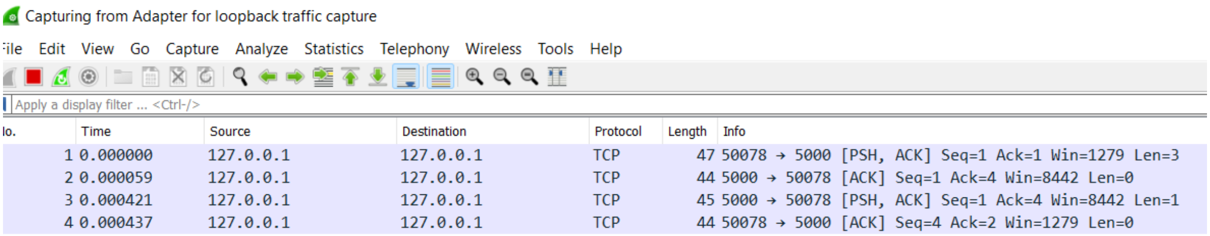
After running the above client and server program a new packet capture is created in the wireshark application with name “Adapter for loopback traffic capture” and ip address 127.0.0.1



After performing some task in the client server at the terminal.



The following is observed in the loopback traffic capture.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	47	50078 → 5000 [PSH, ACK] Seq=1 Ack=1 Win=1279 Len=3
2	0.000059	127.0.0.1	127.0.0.1	TCP	44	5000 → 50078 [ACK] Seq=1 Ack=4 Win=8442 Len=0
3	0.000421	127.0.0.1	127.0.0.1	TCP	45	5000 → 50078 [PSH, ACK] Seq=1 Ack=4 Win=8442 Len=1
4	0.000437	127.0.0.1	127.0.0.1	TCP	44	50078 → 5000 [ACK] Seq=4 Ack=2 Win=1279 Len=0

During the communication between the client and server, TCP communication is easily noticeable. Since both the client and server are on the same machine, the source and destination IP addresses are identical, i.e., 127.0.0.1. We can see from the screenshot that the TCP communication is happening at two different ports, Port number 50078 and Port number 5000.

9. Observing Packet capturing in SSH connection:

Established SSH with IITJ home folder as below:

```
aman@linux:~$ ssh aman@192.168.137.64
```

After running the above a new packet capture with name “adapter for loopback traffic capture” is created.

Welcome to Wireshark

Capture

...using this filter: Enter a capture filter ...

Ethernet 3
Local Area Connection* 10
Local Area Connection* 9
Local Area Connection* 8
Wi-Fi
Local Area Connection* 12
Local Area Connection* 1
Ethernet 4
Adapter for loopback traffic capture
Ethernet
Local Area Connection

The content of the packet capture is as below:

0.517395	172.31.10.74	172.31.10.74	SSH	92	Client: Encrypted packet (len=36)
0.517458	172.31.10.74	172.31.10.74	TCP	56	22 → 61837 [ACK] Seq=37 Ack=73 Win=6320 Len=0 TSval=2073772202 TSecr=2221900372
0.517790	172.31.10.74	172.31.10.74	SSH	100	Server: Encrypted packet (len=44)
0.517817	172.31.10.74	172.31.10.74	TCP	56	61837 → 22 [ACK] Seq=73 Ack=81 Win=6322 Len=0 TSval=2221900372 TSecr=2073772202
1.287484	172.31.10.74	172.31.10.74	SSH	92	Client: Encrypted packet (len=36)
1.287548	172.31.10.74	172.31.10.74	TCP	56	22 → 61837 [ACK] Seq=81 Ack=109 Win=6320 Len=0 TSval=2073772972 TSecr=2221901142
1.287849	172.31.10.74	172.31.10.74	SSH	92	Server: Encrypted packet (len=36)
1.287875	172.31.10.74	172.31.10.74	TCP	56	61837 → 22 [ACK] Seq=109 Ack=117 Win=6321 Len=0 TSval=2221901142 TSecr=2073772972
1.355222	172.31.10.74	172.31.10.74	SSH	92	Client: Encrypted packet (len=36)
1.355274	172.31.10.74	172.31.10.74	TCP	56	22 → 61837 [ACK] Seq=117 Ack=145 Win=6319 Len=0 TSval=2073773040 TSecr=2221901210

Conclusions:

- The IP addresses are the same because the SSH connection is between the same machine.
- SSH uses TCP for transferring data.
- The two ports used for the SSH connection are port 22 and 61837. Usually, SSH works on port 22.
- Since the SSH connection is secure, the transmitted data is not visible. Instead, the info shows as "Encrypted packet (len=36)".
- TCP handshaking is happening, but some steps like SYN and SYN-ACK are hidden. Only the server's ACK step can be seen in Wireshark.