

NETWORK PACKET CAPTURE & TCP ANALYSIS REPORT

Analyst: Precious Anyanwu

Date: 24/01/2026

Case ID: NET-ANALYSIS-001

1. Case Summary

This lab exercise demonstrates how data packets are transmitted and captured within a local test environment. The objective was to observe real network communication, understand the TCP three-way handshake process, and analyze how a client and server establish and maintain a connection.

A local HTTP server was configured, and traffic between the client and server was monitored using packet capture tools.

2. Lab Environment

Component	Description
OS	Kali Linux
Capture Tool	tcpdump
Server Tool	Python HTTP Server
Client Tool	Netcat (nc)
Network Interface	Loopback Interface (lo)

3. Tools Used

- **tcpdump** – Network packet capture and analysis

- **Python HTTP Server** – To simulate a local web server
 - **Netcat (nc)** – To simulate client connection requests
-

4. Packet Capture Setup

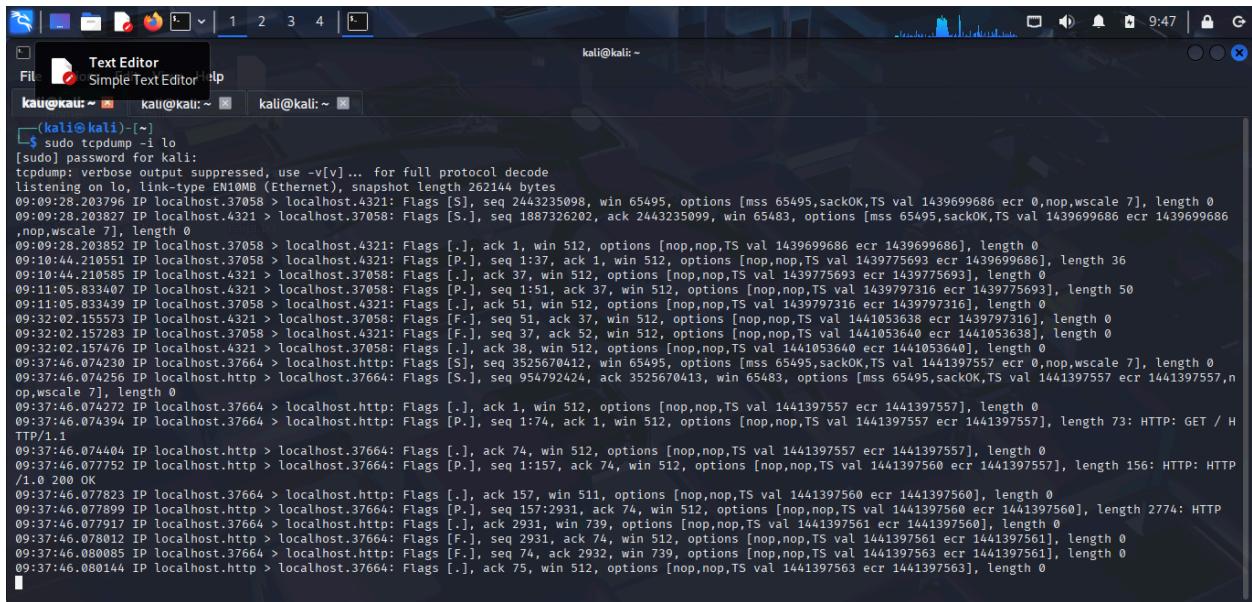
To begin monitoring traffic, the available network interfaces were listed:

```
tcpdump -D
```

The loopback interface was selected for capture:

```
sudo tcpdump -i lo
```

This command captures packets flowing through the local system (127.0.0.1), allowing observation of client-server communication within the machine.



A screenshot of a Kali Linux desktop environment showing a terminal window. The terminal window title is 'kali@kali: ~'. Inside the terminal, the command 'sudo tcpdump -i lo' is run, followed by a list of captured network packets. The output shows various TCP connections between the localhost IP (127.0.0.1) and port 37664, illustrating the capture of client-server communication.

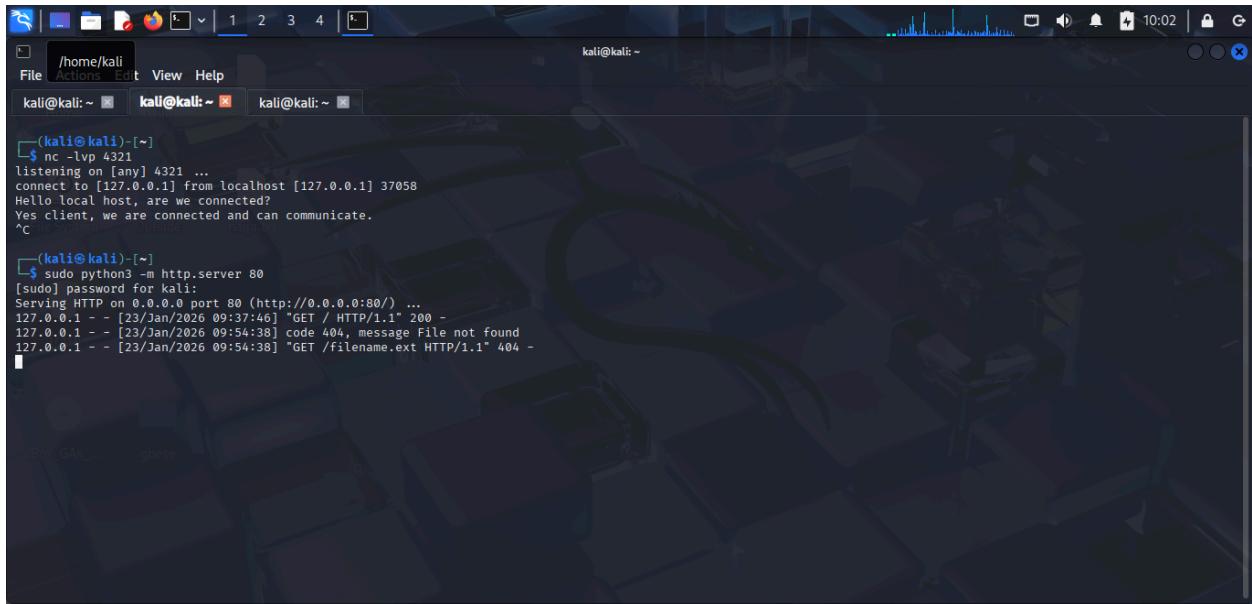
```
[-] (Kali㉿kali)-[~]
$ sudo tcpdump -i lo
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
09:09:28.203794 IP localhost.4321 > localhost.37058: Flags [S], seq 2443235098, win 65495, options [mss 65495,sackOK,TS val 1439699686 ecr 0,nop,wscale 7], length 0
09:09:28.203827 IP localhost.4321 > localhost.37058: Flags [S.], seq 1887326202, ack 2443235099, win 65483, options [mss 65495,sackOK,TS val 1439699686 ecr 1439699686 ,nop,wscale 7], length 0
09:09:28.203852 IP localhost.37058 > localhost.4321: Flags [.], ack 1, win 512, options [nop,nop,TS val 1439699686 ecr 1439699686], length 0
09:10:44.210551 IP localhost.37058 > localhost.4321: Flags [P.], seq 1:37, ack 1, win 512, options [nop,nop,TS val 1439775693 ecr 1439699686], length 36
09:10:44.210585 IP localhost.4321 > localhost.37058: Flags [.], ack 37, win 512, options [nop,nop,TS val 1439775693 ecr 1439775693], length 0
09:11:05.833407 IP localhost.4321 > localhost.37058: Flags [P.], seq 1:51, ack 37, win 512, options [nop,nop,TS val 1439797316 ecr 1439775693], length 50
09:11:05.833430 IP localhost.37058 > localhost.4321: Flags [.], ack 51, win 512, options [nop,nop,TS val 1439797316 ecr 1439797316], length 0
09:32:02.155573 IP localhost.4321 > localhost.37058: Flags [F.], seq 51, ack 37, win 512, options [nop,nop,TS val 1441053638 ecr 1439797316], length 0
09:32:02.157283 IP localhost.37058 > localhost.4321: Flags [F.], seq 37, ack 52, win 512, options [nop,nop,TS val 1441053640 ecr 1441053638], length 0
09:32:02.157476 IP localhost.4321 > localhost.37058: Flags [.], ack 38, win 512, options [nop,nop,TS val 1441053640 ecr 1441053640], length 0
09:37:46.074230 IP localhost.37664 > localhost.http: Flags [S], seq 3525670412, win 65495, options [mss 65495,sackOK,TS val 1441397557 ecr 0,nop,wscale 7], length 0
09:37:46.074256 IP localhost.http > localhost.37664: Flags [S.], seq 954792424, ack 3525670413, win 65483, options [mss 65495,sackOK,TS val 1441397557 ecr 1441397557,nop,wscale 7], length 0
09:37:46.074272 IP localhost.37664 > localhost.http: Flags [.], ack 1, win 512, options [nop,nop,TS val 1441397557 ecr 1441397557], length 0
09:37:46.074394 IP localhost.37664 > localhost.http: Flags [P.], seq 1:74, ack 1, win 512, options [nop,nop,TS val 1441397557 ecr 1441397557], length 73: HTTP: GET / HTTP/1.1
09:37:46.074404 IP localhost.http > localhost.37664: Flags [.], ack 74, win 512, options [nop,nop,TS val 1441397557 ecr 1441397557], length 0
09:37:46.074472 IP localhost.http > localhost.37664: Flags [P.], seq 1:157, ack 74, win 512, options [nop,nop,TS val 1441397560 ecr 1441397557], length 156: HTTP: HTTP/1.0 200 OK
09:37:46.077823 IP localhost.37664 > localhost.http: Flags [.], ack 157, win 511, options [nop,nop,TS val 1441397560 ecr 1441397560], length 0
09:37:46.077899 IP localhost.http > localhost.37664: Flags [P.], seq 157:2931, ack 74, win 512, options [nop,nop,TS val 1441397560 ecr 1441397560], length 2774: HTTP: 09:37:46.077917 IP localhost.37664 > localhost.http: Flags [.], ack 2931, win 739, options [nop,nop,TS val 1441397561 ecr 1441397560], length 0
09:37:46.078012 IP localhost.http > localhost.37664: Flags [F.], seq 2931, ack 74, win 512, options [nop,nop,TS val 1441397561 ecr 1441397561], length 0
09:37:46.080085 IP localhost.37664 > localhost.http: Flags [F.], seq 74, ack 2932, win 739, options [nop,nop,TS val 1441397563 ecr 1441397561], length 0
09:37:46.080144 IP localhost.http > localhost.37664: Flags [.], ack 75, win 512, options [nop,nop,TS val 1441397563 ecr 1441397563], length 0
```

5. Server Configuration

A local HTTP server was created on port 80:

```
sudo python3 -m http.server 80
```

This allowed the system to act as a web server, waiting for incoming client requests.



The screenshot shows a terminal window on a Kali Linux desktop environment. The terminal is running a netcat listener on port 4321 and a simple HTTP server on port 80 using Python's http.server module. The netcat listener logs a connection from the local host. The HTTP server responds to a 'GET /' request with a 200 status code and a 'File not found' message. The terminal session is as follows:

```
(kali㉿kali)-[~]
$ nc -lvp 4321 ...
listening on [any] 4321 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 37058
Hello local host, are we connected?
Yes client, we are connected and can communicate.
^C

(kali㉿kali)-[~]
$ sudo python3 -m http.server 80
[sudo] password for kali:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
127.0.0.1 - - [23/Jan/2026 09:37:46] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Jan/2026 09:54:38] "GET /filename.ext HTTP/1.1" 404 -
```

6. Client Connection

A client was created using Netcat to initiate communication with the server:

```
nc 127.0.0.1 80 -v
```

This command established a connection between the client and the local HTTP server.

The screenshot shows a terminal window on a Kali Linux system. The user has run a netcat listener on port 80 with the command: \$ nc -l 0.0.0.1 80 -v. A connection from localhost on port 127.0.0.1:80 is established. The user types 'hello' into the terminal, and the server responds with an error message:

```
nc 127.0.0.1 80 -v
localhost [127.0.0.1] 80 (http) open
hello
<!DOCTYPE HTML>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Error response</title>
</head>
<body>
    <h1>Error response</h1>
    <p>Error code: 400</p>
    <p>Message: Bad request syntax ('hello').</p>
    <p>Error code explanation: 400 - Bad request syntax or unsupported method.</p>
</body>
</html>
```

Below the terminal, there's a GitHub repository page for 'DidierStevens/TCM-Course-Resources'. The repository contains a single file, README.md, which includes the above HTML code.

7. TCP Three-Way Handshake Observation

The packet capture displayed the TCP three-way handshake, which establishes reliable communication between systems:

1. **SYN** – The client sends a synchronization packet with an initial sequence number.
2. **SYN-ACK** – The server responds, acknowledging the client's sequence number and sending its own.
3. **ACK** – The client acknowledges the server's sequence number.

After this exchange, the TCP session is established, and data transfer begins.

This process confirms TCP's connection-oriented nature and reliability.

8. Traffic Analysis Test

To generate additional traffic, a request was made for a non-existent file:

```
curl http://127.0.0.1/filename.ext -O filename.ext
```

This action produced observable HTTP request and response packets in tcpdump, demonstrating how application-layer requests appear in packet captures.

(Insert Screenshot: tcpdump showing request/response)

9. Key Observations

- Successful capture of TCP handshake packets
 - Sequence and acknowledgment numbers incremented correctly
 - HTTP request traffic was visible in packet capture
 - Demonstrated how local traffic flows through the loopback interface
-

10. Conclusion

This lab provided practical exposure to real network packet flow and TCP communication behavior. Observing the three-way handshake reinforced understanding of TCP as a reliable, connection-oriented protocol that ensures ordered and controlled data delivery.

Hands-on packet analysis is a foundational skill in network security and SOC operations, where analysts monitor traffic to detect anomalies, malicious communication, or policy violations.