Task 5 — Packet Capture & Analysis (Wireshark)

Executive summary

Performed a short, controlled packet capture to demonstrate capture technique, protocol identification, and basic traffic analysis. The capture (~60–90 seconds) intentionally generated normal user activity (web browsing + ping) to show DNS, HTTPS (TLS), TCP behavior, ARP, and ICMP. No malicious traffic was observed in this sample. Recommendations cover analysis best practices and quick mitigations for suspicious behavior.

Objectives

- Capture live network traffic using Wireshark (or tshark) on an active interface.
- Identify DNS, HTTP/HTTPS, TCP, ARP, and ICMP traffic and interpret key fields.
- Use display filters and follow TCP streams for deeper inspection.
- Export pcap and supporting artifacts to a GitHub-ready submission.

Environment / tools

- Wireshark (GUI) — tested latest stable release.
- tshark (CLI) — for scripted captures and quick stats.
- OS: Linux (commands present) — Windows/macOS equivalents noted.
- Target: local workstation (capture performed on local NIC).

How I captured (step-by-step)

GUI (Wireshark)

1. Open Wireshark.
2. Select the active network interface (Ethernet or Wi-Fi).
3. Ensure “Promiscuous mode” is enabled (optional).
4. Click **Start** to begin capture.
5. Perform basic traffic generation:
    - Open a browser and visit https://example.com
    - Run ping -c 5 google.com

6. Stop capture after 60–90 seconds.
7. File → Export Packet Dissections → as analysis_capture.pcapng (or pcap).

CLI (tshark)

Capture 90 seconds, save to pcap:

run with privileges if required

timeout 90 tshark -i eth0 -w analysis_capture.pcap

Capture specific number of packets (example 200):

tshark -i eth0 -c 200 -w analysis_capture.pcap

Export readable CSV for quick inspection:

tshark -r analysis_capture.pcap -T fields -e frame.number -e frame.time -e ip.src -e ip.dst -e _ws.col.Protocol -e _ws.col.Info > capture_summary.csv

Useful display filters (Wireshark)

Save these to a sample_filters.txt file:

dns

http

tls

tcp

tcp.port == 80

tcp.port == 443

arp

icmp

ip.addr == 8.8.8.8

tcp contains "GET"

Quick CLI stats (tshark / capinfos)

General file info:

capinfos analysis_capture.pcap

Top talkers (by bytes):

tshark -r analysis_capture.pcap -q -z conv,ip

Protocol distribution:

tshark -r analysis_capture.pcap -q -z io,phs

What I looked for (analysis checklist)

- DNS queries: name, type, response IPs, NXDOMAIN or SERVFAIL.
- TLS: ClientHello (SNI), TLS version, cipher negotiation.
- TCP: 3-way handshakes, retransmissions, excessive RSTs, unusual ports.
- ARP: repeated who-has requests (possible ARP poisoning).
- ICMP: ping, unreachable messages.
- Suspicious patterns: repeated failed connections, port scans (many SYNs), traffic to known malicious IPs/domains.

Sample findings (professional narrative you can paste into the report)

Capture summary

- Capture duration: 75 seconds
- Packet count: 1,850 (example; report your real number)
- Top protocols (by packet count): TLS/HTTPS, DNS, TCP, ARP, ICMP

Notable observations

1. **DNS**
    - Several A/AAAA lookups for www.example.com and google.com. All responses returned valid A records.
    - No DNS NXDOMAIN or suspicious redirect responses observed.
2. **HTTPS / TLS**
    - TLS ClientHello packets observed with SNI fields revealing hosts such as www.example.com and accounts.google.com.
    - TLS negotiation completed normally (ClientHello → ServerHello → Encrypted Handshake).

3. **TCP**
    - Normal 3-way handshakes (SYN → SYN/ACK → ACK) for client connections.
    - Small number of retransmissions during page load (expected under normal network conditions).
4. **ARP**
    - Periodic ARP who-has requests for gateway and local hosts — normal for local network discovery.
5. **ICMP**
    - Manual ping to google.com produced expected echo requests and replies, round-trip average ~20 ms.

Conclusion
No indicators of compromise or suspicious scanning activity in this short, controlled capture. The traffic profile matches normal browsing + OS background services.

Example packet analyses (copyable)

DNS query example (fields to show)

- Frame: #123
- Time: 2025-11-XX 12:34:56
- Source IP: 192.168.1.10
- Destination IP: <local resolver>
- Query: www.example.com Type A
- Response: A 93.184.216.34

TCP handshake example

Show the three frame IDs and brief description:

Frame 201: 192.168.1.10 → 93.184.216.34: TCP SYN

Frame 202: 93.184.216.34 → 192.168.1.10: TCP SYN, ACK

Frame 203: 192.168.1.10 → 93.184.216.34: TCP ACK

TLS ClientHello example

- SNI: www.example.com

- Supported TLS versions/ciphers: TLS1.2 / ECDHE-RSA-AES256-GCM-SHA384 (example)
- Certificate chain: standard CA issuance (verify in follow-up if required)

Suspicious patterns to flag (what to look for)

- Many source IPs sending SYNs to many destination ports (classic port scan).
- DNS responses pointing to unexpected IPs (possible DNS poisoning).
- Outbound connections to IPs on known threat lists — block and investigate.
- Repeated ARP announcements — possible ARP spoofing.
- Large number of RSTs or retransmissions from many endpoints — possible DDoS symptomatic noise.

Remediation & recommendations

- Enforce network segmentation and limit unnecessary inbound services.
- Use HTTPS and HSTS to reduce plaintext credential leaks.
- Configure DNSSEC if possible; prefer secure resolvers.
- Monitor for ARP anomalies; consider static ARP entries for critical hosts.
- Archive pcaps for incident response; redact PII before sharing.
- For suspicious IPs, block at gateway and run further investigation (Whois, passive DNS).

README.md (copy this into the repo root)

Task 5 — Packet Capture & Analysis Using Wireshark

Objective

Capture and analyze local network traffic to identify common protocols and document findings.

How to reproduce

GUI (Wireshark)

1. Select interface and click Start.

2. Perform browsing and a ping test.

3. Stop capture after 60–90 seconds.

4. File → Export Packet Dissections → Save as ` analysis_capture.pcapng` .

CLI (tshark)

```bash

timeout 90 tshark -i eth0 -w analysis_capture.pcap

tshark -r analysis_capture.pcap -T fields -e frame.number -e frame.time -e ip.src -e ip.dst -e _ws.col.Protocol -e _ws.col.Info > capture_summary.csv
```

Files included

- analysis_capture.pcapng — packet capture (redact before sharing if needed)
- capture_summary.csv — summary extracted via tshark
- report.md — this report (findings and recommendations)
- sample_filters.txt — Wireshark display filters
- protocol_distribution.png and packet_timeline.png — illustrative visuals

Findings (short)

- Normal DNS, HTTPS, TCP, ARP, and ICMP observed.
- No suspicious scanning or malicious traffic in this sample.
- Recommended actions: review longer captures for anomalies, enforce HTTPS, monitor ARP.

Notes

Only capture on networks you own or have permission to test.

Example commands you can run to produce artifacts

1. Export pcap from Wireshark GUI: File → Export Packet Dissections → as pcapng.

2. Generate CSV summary with `tshark`:

```bash
tshark -r analysis_capture.pcap -T fields -e frame.number -e frame.time -e ip.src -e ip.dst -e _ws.col.Protocol -e _ws.col.Info > capture_summary.csv
```

3. Produce a protocol distribution chart (quick Python snippet)

```
# requires matplotlib, pandas
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('capture_summary.csv', names=['frame','time','src','dst','proto','info'])
counts = df['proto'].value_counts().head(10)
counts.plot.pie(autopct='%1.0f%%', figsize=(6,6))
plt.savefig('protocol_distribution.png', bbox_inches='tight')
```