

Network Traffic Analysis with AI

Capturing, dissecting, and leveraging AI to find needles in network haystacks.

WARNING — Authorized Use Only: Capturing network traffic on networks you do not own or do not have explicit written authorization to test is illegal in most jurisdictions. The techniques in this lesson are intended solely for CTF competitions, lab environments, and authorized penetration tests. Always confirm scope and obtain written permission before capturing any traffic.

Why Network Traffic Analysis Matters for Red Teams

Network traffic is the circulatory system of every organization. Every authentication attempt, database query, file transfer, and command-and-control heartbeat eventually becomes a sequence of packets on a wire. For red teamers, traffic analysis serves two complementary purposes: understanding what a target network looks like from the inside, and detecting (or avoiding detection of) malicious activity.

During a penetration test, capturing traffic can reveal:

- **Clear-text credentials** transmitted over legacy protocols (FTP, Telnet, HTTP Basic Auth, LDAP without TLS)
- **Internal network topology** — broadcast traffic, ARP tables, and DNS queries expose hosts, services, and naming conventions
- **Lateral movement opportunities** — SMB shares, RDP sessions, and internal API calls show trust relationships between machines
- **Data exfiltration channels** — DNS tunneling, ICMP covert channels, and oversized HTTP POST requests can indicate ongoing compromise

Manually reviewing hundreds of thousands of packets is impractical. This is where AI-assisted analysis becomes a force multiplier: an LLM can summarize traffic patterns, flag anomalies, and correlate indicators across sessions far faster than a human analyst scrolling through Wireshark.

Packet Capture Fundamentals

Three tools form the backbone of traffic capture and inspection:

tcpdump

The command-line workhorse. Available on virtually every Unix system, tcpdump captures packets directly from a network interface and can write them to a pcap file for later analysis.

```
# Capture all traffic on eth0, write to file
sudo tcpdump -i eth0 -w capture.pcap

# Capture only DNS traffic (port 53)
sudo tcpdump -i eth0 port 53 -w dns_traffic.pcap

# Capture HTTP traffic and print ASCII payload
sudo tcpdump -i eth0 port 80 -A

# Capture first 500 packets on any interface
sudo tcpdump -i any -c 500 -w sample.pcap

# Read back a capture with verbose output
tcpdump -r capture.pcap -vvv
```

Wireshark

The graphical packet analyzer. Wireshark provides deep protocol dissection, conversation tracking, and powerful display filters. It is the go-to tool for interactive analysis of pcap files. Use it when you need to follow TCP streams, reconstruct file transfers, or visually trace a multi-step attack chain.

tshark

The command-line interface to Wireshark's dissection engine. tshark combines Wireshark's protocol awareness with the scriptability of a CLI tool, making it ideal for automated analysis pipelines.

```
# Extract all HTTP requests from a capture
tshark -r capture.pcap -Y "http.request" -T fields \
  -e frame.time -e ip.src -e http.host -e http.request.uri
```

```
# List all DNS queries
tshark -r capture.pcap -Y "dns.qr == 0" -T fields \
  -e frame.time -e ip.src -e dns.qry.name

# Show all unique source/destination IP pairs
tshark -r capture.pcap -T fields -e ip.src -e ip.dst | sort -u

# Export HTTP objects (files transferred over HTTP)
tshark -r capture.pcap --export-objects http,exported_files/
```

Protocol Analysis Essentials

Effective traffic analysis requires understanding the protocols you are looking at:

Protocol	Layer	Red Team Relevance
ARP	2 (Data Link)	ARP spoofing enables man-in-the-middle attacks; gratuitous ARP reveals host discovery
TCP/IP	3–4	SYN scans, connection tracking, port enumeration; RST patterns reveal firewalls
DNS	7 (Application)	Domain reconnaissance, DNS tunneling for exfil, zone transfer attempts (AXFR)
HTTP/HTTPS	7	Credential harvesting, API enumeration, web shell callbacks; TLS hides content but metadata (SNI, cert subject) remains visible
SMB	7	Lateral movement, file share enumeration, pass-the-hash relay
ICMP	3	Host discovery (ping), covert channels in payload bytes, traceroute topology mapping

The pcap File Format and Analysis Workflow

The **pcap** (Packet Capture) format is the universal interchange format for captured network traffic. Its successor, **pcapng**, adds metadata like interface descriptions and

capture comments. Both are supported by tcpdump, Wireshark, tshark, and Scapy.

A typical analysis workflow looks like this:

1. **Capture** — Record raw traffic with tcpdump or Wireshark to a pcap file
2. **Filter** — Use BPF filters (capture-time) or display filters (analysis-time) to isolate relevant traffic
3. **Summarize** — Generate statistics: protocol distribution, top talkers, connection durations
4. **Deep dive** — Follow individual TCP streams, reconstruct transferred files, decode payloads
5. **Correlate** — Cross-reference findings with logs, threat intelligence, or other captures

BPF Filter Syntax

Berkeley Packet Filters (BPF) are used by tcpdump, Wireshark capture filters, and many other tools. They operate at capture time and are highly efficient.

```
# Filter by host
sudo tcpdump -i eth0 host 192.168.1.100

# Filter by network (CIDR)
sudo tcpdump -i eth0 net 10.0.0.0/8

# Filter by port
sudo tcpdump -i eth0 port 443

# Filter by protocol
sudo tcpdump -i eth0 icmp

# Combine with logical operators
sudo tcpdump -i eth0 'host 192.168.1.100 and (port 80 or port 443)'

# Capture only TCP SYN packets (flag value 0x02)
sudo tcpdump -i eth0 'tcp[tcpflags] & tcp-syn != 0'

# Capture DNS queries only (QR bit = 0)
sudo tcpdump -i eth0 'udp port 53 and udp[10] & 0x80 = 0'
```

Using AI to Analyze Packet Captures

Large language models like Claude excel at pattern recognition across structured text. The key insight is that network traffic, when exported as human-readable text, becomes a natural language analysis problem. Here is a practical workflow:

Step 1: Export traffic as text

```
# Export packet summaries for AI analysis
tshark -r capture.pcap -T fields \
  -e frame.number -e frame.time_relative -e ip.src -e ip.dst \
  -e _ws.col.Protocol -e _ws.col.Info \
  > traffic_summary.txt

# Export a specific TCP stream as ASCII
tshark -r capture.pcap -z "follow,tcp,ascii,0" > stream0.txt
```

Step 2: Feed to Claude with targeted prompts

```
import anthropic

client = anthropic.Anthropic()

with open("traffic_summary.txt") as f:
    traffic_data = f.read()

message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=4096,
    messages=[{
        "role": "user",
        "content": f"""Analyze this network traffic capture summary. Identify:
1. Any clear-text credentials or authentication attempts
2. Suspicious DNS queries (tunneling, DGA domains, unusual TLDs)
3. Potential C2 beacon patterns (regular intervals, consistent payload sizes)
4. Signs of lateral movement (SMB, RDP, WinRM connections)
5. Data exfiltration indicators (large outbound transfers, DNS TXT queries)

Traffic data:
{traffic_data}"""
    }]
}
```

```
)  
print(message.content[0].text)
```

What AI catches well

- **Beacon detection** — LLMs quickly spot periodic connection patterns (e.g., a host connecting to the same external IP every 60 seconds with similarly sized payloads)
- **DNS anomalies** — Unusually long subdomain labels, high entropy domain names, or excessive TXT record queries that suggest tunneling
- **Protocol misuse** — HTTP traffic on non-standard ports, DNS responses with unexpectedly large payloads, ICMP packets with non-zero data sections
- **Credential exposure** — Base64-encoded Authorization headers, FTP USER/PASS sequences, LDAP simple binds

Identifying Interesting Traffic Patterns

Clear-text Credentials

Legacy protocols transmit credentials in plain text. Look for FTP (`USER` / `PASS` commands), HTTP Basic Auth (`Authorization: Basic` header with base64), Telnet login sequences, and SNMP community strings.

DNS Exfiltration

Attackers encode stolen data in DNS subdomain labels. The query

`dGh1IHNLy3JldCBkYXRh.evil.com` embeds base64-encoded data as a subdomain.

Indicators: long queries, high query volume to a single domain, TXT record responses with large payloads.

C2 Beacons

Command-and-control implants call home at regular intervals. Look for connections to the same destination at consistent time deltas, similar request sizes, and responses that vary only slightly. Jittered beacons add randomness but still cluster around a mean interval.

Lateral Movement

After initial compromise, attackers pivot through the internal network. Watch for a single source IP initiating connections to many internal hosts on SMB (445), RDP (3389), WinRM (5985/5986), or SSH (22) in rapid succession.

Scapy: Python-Based Packet Crafting and Analysis

Scapy is a Python library that lets you build, send, capture, and dissect packets programmatically. It is invaluable for CTF challenges where you need to parse custom protocols or craft specific packets.

```
from scapy.all import rdpcap, IP, TCP, DNS, DNSQR, Raw

# Load a pcap file
packets = rdpcap("capture.pcap")

# Print summary of first 10 packets
for pkt in packets[:10]:
    print(pkt.summary())

# Extract all DNS queries
dns_queries = []
for pkt in packets:
    if pkt.haslayer(DNS) and pkt[DNS].qr == 0: # qr=0 means query
        query_name = pkt[DNSQR].qname.decode()
        dns_queries.append(query_name)

print(f"Unique DNS queries: {len(set(dns_queries))}")
for q in sorted(set(dns_queries))[:20]:
    print(f"    {q}")
```

```
from scapy.all import rdpcap, IP, TCP, Raw

# Find packets containing potential credentials
packets = rdpcap("capture.pcap")
keywords = [b"password", b"passwd", b"user", b"login", b"Authorization"]

for pkt in packets:
    if pkt.haslayer(Raw):
        payload = pkt[Raw].load
        for kw in keywords:
            if kw.lower() in payload.lower():
```

```

src = pkt[IP].src if pkt.haslayer(IP) else "?"
dst = pkt[IP].dst if pkt.haslayer(IP) else "?"
print(f"[CRED?] {src} -> {dst}: {payload[:200]}")
break

```

```

from scapy.all import rdpcap, IP
from collections import Counter

# Analyze connection patterns for C2 beacon detection
packets = rdpcap("capture.pcap")

# Count destination IPs from a specific source
src_filter = "192.168.1.50"
dest_counts = Counter()
timestamps = {}

for pkt in packets:
    if pkt.haslayer(IP) and pkt[IP].src == src_filter:
        dst = pkt[IP].dst
        dest_counts[dst] += 1
        timestamps.setdefault(dst, []).append(float(pkt.time))

# Look for periodic connections (potential beacons)
for dst, times in timestamps.items():
    if len(times) >= 5:
        deltas = [times[i+1] - times[i] for i in range(len(times)-1)]
        avg_delta = sum(deltas) / len(deltas)
        variance = sum((d - avg_delta)**2 for d in deltas) / len(deltas)
        if variance < (avg_delta * 0.3) ** 2: # low variance = regular interval
            print(f"[BEACON?] {src_filter} -> {dst}: "
                  f"{len(times)} connections, avg interval {avg_delta:.1f}s, ")

```

Putting It All Together: An Analysis Playbook

When you encounter a pcap file in a CTF challenge or during a red team engagement, follow this systematic approach:

1. **Get the big picture** — Run `tshark -r file.pcap -z conv,ip` to see all IP conversations and their byte volumes
2. **Check protocol distribution** — Run `tshark -r file.pcap -z io,phs` for a protocol hierarchy breakdown

3. **Extract DNS queries** — DNS is almost always interesting; exfiltration, C2 domains, and internal naming all live here
4. **Search for credentials** — Filter for FTP, HTTP Auth headers, LDAP binds, and Telnet sessions
5. **Export text summaries and feed to AI** — Let Claude identify patterns you might miss in manual review
6. **Follow suspicious streams** — Use `tshark -z follow,tcp,ascii,N` or Wireshark to reconstruct full conversations
7. **Script with Scapy** — When the challenge requires custom parsing or statistical analysis, write Python

Further Reading

- [tcpdump man page](#) — complete reference for capture filters and output options
- [Wireshark Display Filters](#) — syntax reference for analysis-time filtering
- [Scapy documentation](#) — official guide for Python packet manipulation
- [Public pcap repository \(Netresec\)](#) — practice captures for analysis
- [CyberDefenders](#) — blue team CTF challenges with real-world pcap analysis

Challenge Resources

Download Files

Text-based capture files used in the lab exercises. Each contains simulated network traffic with embedded indicators of compromise to discover using AI-assisted analysis.

↓ [tcpdump-capture.txt](#) ~9 KB

↓ [dns-queries.txt](#) ~6 KB

↓ [http-sessions.txt](#) ~5 KB

↓ [netflow-data.txt](#) ~7 KB