

Network Traffic Analysis with AI

Using AI-assisted techniques to capture, dissect, and interpret network traffic for red team operations and CTF challenges.

What You'll Learn

Packet capture, protocol analysis, traffic pattern recognition, and AI-powered anomaly detection.

Tools We'll Use

tcpdump, Wireshark, tshark, Claude API, Python/Scapy

PCAP

tcpdump

Wireshark

C2 Detection

DNS Tunneling

AI Analysis

Before We Begin

⚠ Educational Use Only

Network traffic analysis techniques are for authorized security testing and CTF competitions only. Unauthorized network sniffing is illegal in most jurisdictions.

Always ensure you have written authorization before capturing traffic on any network. In CTF contexts, you are working with provided PCAP files or explicitly authorized lab environments.

Notes: Emphasize legal boundaries. Wiretap laws (18 U.S.C. 2511), CFAA, and equivalents in other jurisdictions apply to unauthorized packet capture.

Why Network Traffic Matters for Red Teams

- **Attack surface mapping** — discover services, protocols, and communication paths
- **Credential harvesting** — unencrypted protocols leak usernames, passwords, tokens
- **C2 identification** — detect and understand command-and-control channels
- **Data exfiltration detection** — spot data leaving the network through covert channels
- **Lateral movement tracing** — follow an attacker's path through internal hosts
- **CTF flag extraction** — flags are often embedded in packet payloads, headers, or DNS queries

The AI Advantage

Modern captures can contain millions of packets. AI helps you find needles in haystacks — summarizing conversations, flagging anomalies, and correlating events across protocols.

Packet Capture Fundamentals

The Capture Stack

Tool	Use Case	Format
<code>tcpdump</code>	CLI capture & quick filtering	PCAP
<code>Wireshark</code>	GUI deep-dive analysis	PCAPNG / PCAP
<code>tshark</code>	Scriptable Wireshark CLI	PCAP / JSON / text
<code>dumpcap</code>	Long-running capture (low overhead)	PCAPNG
<code>Scapy</code>	Python packet crafting & parsing	PCAP / raw

All tools read/write the standard PCAP format, making them interchangeable for analysis workflows.

tcpdump — The Swiss Army Knife

Essential Capture Commands

```
# Capture all traffic on eth0, write to file
```

```
sudo tcpdump -i eth0 -w capture.pcap
```

```
# Capture only TCP port 80 (HTTP)
```

```
sudo tcpdump -i eth0 tcp port 80 -w http_traffic.pcap
```

```
# Capture DNS traffic with verbose output
```

```
sudo tcpdump -i any udp port 53 -vv
```

```
# Capture traffic to/from a specific host
```

```
sudo tcpdump -i eth0 host 192.168.1.100 -w target.pcap
```

```
# Capture first 500 bytes of each packet (headers + partial payload)
```

```
sudo tcpdump -i eth0 -s 500 -w partial.pcap
```

Notes: The `-w` flag writes raw packets. Without it, tcpdump prints a text summary which loses data. Always capture to file first, analyze second.

tcpdump — BPF Filter Expressions

Berkeley Packet Filters let you surgically select traffic before it hits disk.

```
# Capture SYN packets only (port scans, new connections)
tcpdump -r capture.pcap 'tcp[tcpflags] & tcp-syn != 0'

# Capture HTTP GET requests
tcpdump -r capture.pcap 'tcp port 80 and tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420'

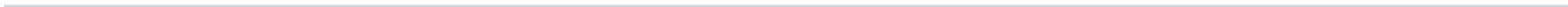
# Capture traffic between two subnets
tcpdump -r capture.pcap 'net 10.0.0.0/8 and net 192.168.0.0/16'

# Exclude SSH (reduce noise during red team ops)
tcpdump -i eth0 'not port 22' -w clean.pcap

# Capture ICMP (ping sweeps, tunneling)
tcpdump -r capture.pcap icmp
```

Red Team Tip

During CTFs, start with a broad capture, then use BPF filters to isolate interesting conversations. Saves time over GUI-only analysis.



tshark — Scriptable Deep Analysis

`tshark` is Wireshark's command-line sibling. It understands all Wireshark dissectors and display filters.

```
# List all HTTP requests with URLs
tshark -r capture.pcap -Y "http.request" \
  -T fields -e ip.src -e http.host -e http.request.uri

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

# Show all TCP conversations sorted by bytes
tshark -r capture.pcap -q -z conv,tcp

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

# Output as JSON for AI processing
```

Notes: The `-T fields -e` syntax is the key to extracting structured data from PCAPs for feeding into AI models.

Wireshark — Visual Deep Dive

Essential Display Filters for CTFs

Filter	Purpose
<code>http.request.method == "POST"</code>	Find form submissions / data uploads
<code>dns.qry.name contains "flag"</code>	DNS queries mentioning flags
<code>tcp.flags.syn == 1 && tcp.flags.ack == 0</code>	New connection attempts (SYN only)
<code>frame contains "password"</code>	Packets containing "password" string
<code>ip.addr == 10.0.0.50 && tcp.port == 4444</code>	Suspicious reverse shell traffic
<code>tcp.stream eq 5</code>	Follow a specific TCP conversation
<code>data.len > 0</code>	Only packets with payload data

Right-click a packet → **Follow** → **TCP Stream** to reconstruct full conversations. This is how you recover transferred files, shell sessions, and exfiltrated data.

Understanding the TCP Handshake

Every TCP connection begins with a three-way handshake. Recognizing this pattern is fundamental to traffic analysis.

Client (10.0.0.5)	Server (10.0.0.10:443)
--- SYN (seq=1000) ----->	Step 1: Client initiates
<-- SYN-ACK (seq=5000, ack=1001) -----	Step 2: Server responds
--- ACK (seq=1001, ack=5001) ----->	Step 3: Client confirms
===== DATA EXCHANGE =====	Connection established
--- FIN ----->	Teardown begins
<-- FIN-ACK -----	
--- ACK ----->	

What to Watch For

SYN scans show SYN packets with no completing handshake. **Half-open connections** (SYN, SYN-ACK, RST) indicate port scanning. **Rapid sequential SYN**s to different ports on one host = port scan.

Reading HTTP Requests in PCAPs

HTTP is plaintext — every header and body is visible in the capture.

```
GET /admin/config.php?debug=true HTTP/1.1
Host: 192.168.1.50
User-Agent: Mozilla/5.0 (X11; Linux x86_64) rv:102.0
Cookie: session=YWRtaW46cGFzc3dvcmQxMjM=
Authorization: Basic YWRtaW46czNjcjN0
Accept: text/html,application/xhtml+xml
X-Forwarded-For: 10.0.0.99
Connection: keep-alive
```

Red Flags in This Request

- **debug=true parameter** — admin debug endpoint being accessed
- **Base64 cookie** — `YWRtaW46cGFzc3dvcmQxMjM=` decodes to `admin:password123`
- **Basic Auth header** — credentials sent in cleartext (Base64-encoded)

- **X-Forwarded-For** — attacker may be spoofing source IP
-

HTTP Response Analysis

```
HTTP/1.1 200 OK
Server: Apache/2.4.49
Content-Type: application/json
X-Powered-By: PHP/7.4.3
Set-Cookie: admin_token=eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoIYWRTaW4i.Sf1K;
  path=/; HttpOnly

{
  "status": "success",
  "db_host": "internal-db.corp.local",
  "db_user": "root",
  "db_pass": "Pr0d_DB_2024!",
  "flag": "CTF{pcap_analysis_reveals_secrets}"
}
```

Intelligence Extracted

- **Apache/2.4.49** — known path traversal vulnerability (CVE-2021-41773)
- **PHP/7.4.3** — version fingerprinting for exploit research
- **JWT token** in Set-Cookie — can be decoded and potentially forged
- **Database credentials** leaked in response body

- **Internal hostname** reveals naming convention
-

DNS Query Analysis

DNS is often overlooked but reveals critical intelligence about network activity.

Timestamp	Source	Query Type	Query Name
14:23:01.001	10.0.0.5	A	c2-server.evil.com
14:23:01.200	10.0.0.5	A	update.legitimate-corp.com
14:23:05.440	10.0.0.5	TXT	aGVsbG8gd29ybGQ=.data.evil.com
14:23:05.880	10.0.0.5	TXT	dGhpcyBpcyBleGZpbA==.data.evil.com
14:23:06.100	10.0.0.5	TXT	dHJhdGVkIGRhdGE=.data.evil.com
14:23:10.500	10.0.0.5	AAAA	staging.internal.corp.local
14:23:15.003	10.0.0.5	MX	corp.local
14:23:20.770	10.0.0.5	A	pastebin.com

Suspicious Indicators

- **c2-server.evil.com** — direct C2 domain resolution
- **Base64 subdomains** on `data.evil.com` — classic DNS exfiltration pattern
- **TXT record queries** to suspicious domains — often used for tunneling

- **Internal domain queries** — attacker performing reconnaissance
-

DNS Tunneling Deep Dive

DNS tunneling encodes data in DNS queries and responses, bypassing firewalls that allow DNS traffic.

How It Works

Data is encoded (Base32/Base64/hex) as subdomain labels. Each query sends ~63 bytes per label, ~253 bytes per domain name.

Responses carry data in TXT, CNAME, or NULL records.

Detection Signals

Unusually long subdomain labels, high query frequency to one domain, TXT record queries, high entropy in domain names, consistent query intervals.

```
# Normal DNS query
wiki.corp.local                (15 chars)

# DNS tunnel query (Base32-encoded payload)
JBSWY3DPEHPK3PXP.MFZWIZLTOQ.t.evil.com  (42 chars, high entropy)

# Detecting with tshark
```

```
tshark -r capture.pcap -Y "dns.qry.name.len > 50" \  
-T fields -e ip.src -e dns.qry.name
```

Notes: Tools like `iodine`, `dnscat2`, and `dns2tcp` automate DNS tunneling. Knowing how they work helps you detect them in captures.

Identifying C2 Beacons

Command-and-control traffic has distinctive patterns even when encrypted.

Beacon Characteristics

Indicator	What to Look For	Example
Regular intervals	Connections at fixed periods (jitter $\pm 10\%$)	Every 60s $\pm$ 6s
Consistent size	Check-in packets have similar byte counts	Small GET, variable response
Unusual User-Agent	Default/generic strings or known C2 signatures	Mozilla/4.0 (compatible)
Long-lived sessions	Connection persists for hours/days	Single TCP stream, many hours
Uncommon ports	HTTPS on non-443 ports, HTTP on high ports	TCP 8443, 1337, 9999
JA3/JA3S fingerprint	TLS client hello fingerprint matches known C2	Cobalt Strike JA3 hash

C2 Beacon — Timing Analysis

```
# Extract connection timestamps to a suspicious host
tshark -r capture.pcap -Y "ip.dst == 203.0.113.50 and tcp.flags.syn == 1" \
  -T fields -e frame.time_relative -e ip.src -e tcp.dstport
```

Output:

```
# 0.000000      10.0.0.5      443
# 60.234100     10.0.0.5      443
# 120.118900    10.0.0.5      443
# 179.987600    10.0.0.5      443
# 240.312400    10.0.0.5      443
# 300.089100    10.0.0.5      443
```

Analysis

Connections every ~60 seconds with slight jitter. This is a textbook C2 beacon. Human browsing is irregular; automated beacons are periodic. Calculate the **inter-arrival time standard deviation** — low values indicate automation.

```
# Python: Calculate beacon interval statistics
import statistics
times = [0.0, 60.23, 120.12, 179.99, 240.31, 300.09]
intervals = [times[i+1] - times[i] for i in range(len(times)-1)]
print(f"Mean interval: {statistics.mean(intervals):.2f}s")
print(f"Std deviation: {statistics.stdev(intervals):.2f}s")
# Mean interval: 60.02s | Std deviation: 0.15s → BEACON
```

Data Exfiltration Patterns

Common Exfiltration Channels

Network-Based

- **DNS tunneling** — data in subdomain labels
- **ICMP tunneling** — data in ping payloads
- **HTTP/S POST** — to cloud storage / paste sites
- **Encrypted channels** — TLS to unknown IPs
- **SMTP** — data as email attachments

Detection Indicators

- **Volume asymmetry** — more data OUT than IN
- **Off-hours traffic** — large transfers at 3 AM
- **New destinations** — IPs never seen before
- **Protocol anomalies** — DNS with large payloads
- **Encoding artifacts** — Base64 in unexpected places

```
# Find hosts sending the most data outbound
tshark -r capture.pcap -q -z endpoints,ip | sort -t'|' -k4 -nr | head -20

# Look for large DNS responses (possible tunneling)
```

```
tshark -r capture.pcap -Y "dns.resp.len > 512" \  
-T fields -e ip.src -e ip.dst -e dns.qry.name -e dns.resp.len
```

Lateral Movement in Traffic

After initial compromise, attackers move through the internal network. These patterns are visible in PCAPs.

Protocols That Reveal Lateral Movement

Protocol / Port	Technique	What to Look For
SMB (445)	PsExec, file copy	Admin\$ / C\$ share access, service creation
WinRM (5985/5986)	PowerShell remoting	HTTP SOAP requests to /wsman
RDP (3389)	Remote desktop	New RDP sessions from compromised hosts
SSH (22)	Remote shell / tunneling	SSH from servers that normally don't initiate SSH
WMI (135 + dynamic)	Remote execution	DCOM/RPC traffic followed by process creation
DCE/RPC (various)	Pass-the-hash, Kerberoasting	Unusual Kerberos TGS requests, NTLM auth

```
# Find internal SMB connections (lateral movement indicator)
tshark -r capture.pcap -Y "smb2 and ip.src == 10.0.0.0/8 and ip.dst == 10.0.0.0/8" \
-T fields -e ip.src -e ip.dst -e smb2.cmd
```

Unusual Port Usage

Attackers often disguise traffic by running services on unexpected ports.

```
# Find HTTP traffic on non-standard ports
tshark -r capture.pcap -Y "http and tcp.port != 80 and tcp.port != 443 and tcp.port != 8080" \
  -T fields -e ip.src -e ip.dst -e tcp.dstport -e http.host

# Example output:
# 10.0.0.5 203.0.113.50 9090 c2.evil.com
# 10.0.0.5 203.0.113.50 1337 update.evil.com

# Find services running on ports that don't match their expected protocol
tshark -r capture.pcap -Y "tcp.port == 53 and !dns"
tshark -r capture.pcap -Y "tcp.port == 443 and !tls"
```

Common Evasion Techniques

- **HTTP over 443** — plaintext HTTP disguised as HTTPS
- **C2 over 53** — non-DNS traffic on the DNS port
- **Reverse shells on 80/443** — blending with web traffic

- **Domain fronting** — TLS SNI shows legitimate domain, Host header points to C2
-

Encrypted Traffic Analysis

TLS encryption hides payload content, but metadata still reveals patterns.

What's Visible in Encrypted Traffic

- **TLS handshake details** — SNI (Server Name Indication), certificate info, cipher suites
- **JA3 / JA3S fingerprints** — unique hash of TLS client/server hello parameters
- **Packet sizes and timing** — reveal communication patterns even when encrypted
- **Certificate properties** — self-signed, short-lived, unusual issuers
- **Connection duration** — long-lived TLS = possible C2 channel

```
# Extract TLS SNI (Server Name Indication) values
tshark -r capture.pcap -Y "tls.handshake.type == 1" \
  -T fields -e ip.src -e ip.dst -e tls.handshake.extensions_server_name

# Extract certificate details
tshark -r capture.pcap -Y "tls.handshake.type == 11" \
  -T fields -e ip.src -e x509ce.dNSName -e x509af.utcTime

# JA3 fingerprinting (requires Wireshark 3.x+ or zeek)
```

```
tshark -r capture.pcap -Y "tls.handshake.type == 1" \  
-T fields -e ip.src -e ja3.hash
```

Netflow Data Analysis

When full PCAPs aren't available, netflow records summarize connection metadata.

Start Time	Duration	Proto	Src IP:Port	Dst IP:Port	Packets	Bytes	Flags
2024-03-15 02:15	3600s	TCP	10.0.0.5:49221	203.0.113.50:443	14400	864000	.AP.SF
2024-03-15 02:15	0.1s	TCP	10.0.0.5:49222	10.0.0.20:445	6	420	.AP.SF
2024-03-15 02:15	0.1s	TCP	10.0.0.5:49223	10.0.0.21:445	6	420	.AP.SF
2024-03-15 02:15	0.1s	TCP	10.0.0.5:49224	10.0.0.22:445	4	280	.AP.R.
2024-03-15 02:16	1800s	TCP	10.0.0.5:49225	10.0.0.20:445	8200	5242880	.AP.SF
2024-03-15 02:48	120s	TCP	10.0.0.5:49226	203.0.113.50:443	240	2097152	.AP.SF

What This Tells Us

- **Line 1:** 1-hour persistent connection to external host (C2 beacon — 864KB over 1 hour)
 - **Lines 2-4:** Rapid SMB scans across internal hosts (lateral movement recon)
 - **Line 5:** 5MB transferred over SMB to 10.0.0.20 (file access / staging)
 - **Line 6:** 2MB sent to external host in 2 minutes (data exfiltration)
-

Preparing PCAP Data for AI Analysis

Raw PCAPs are too large and binary for direct AI input. You need to extract structured summaries.

Extraction Pipeline

```
# Step 1: Export conversation list
tshark -r capture.pcap -q -z conv,tcp > conversations.txt

# Step 2: Extract HTTP request/response pairs
tshark -r capture.pcap -Y "http" -T fields \
  -e frame.number -e frame.time_relative -e ip.src -e ip.dst \
  -e http.request.method -e http.request.uri -e http.response.code \
  -e http.content_type -E header=y -E separator=, > http_summary.csv

# Step 3: Extract DNS queries with timestamps
tshark -r capture.pcap -Y "dns" -T fields \
  -e frame.time_relative -e ip.src -e dns.qry.name \
  -e dns.qry.type -e dns.resp.name -e dns.a \
  -E header=y -E separator=, > dns_summary.csv
```

Feeding Traffic Data to Claude

Structure your AI prompts with clear context, data format, and specific questions.

```
import anthropic

client = anthropic.Anthropic()

# Load extracted data
with open("http_summary.csv") as f:
    http_data = f.read()
with open("dns_summary.csv") as f:
    dns_data = f.read()

prompt = f"""You are a network forensics analyst investigating a potential
security incident. Analyze the following network traffic data extracted
from a PCAP file.

## HTTP Traffic Summary
```

AI Prompt Patterns for Traffic Analysis

Pattern 1: Anomaly Detection

```
Analyze these DNS queries from host 10.0.0.5.  
Identify anomalies compared to normal corporate DNS usage:  
- Unusually long domain names  
- High-entropy subdomain labels  
- Queries to recently registered domains  
- Excessive query frequency to a single domain  
- TXT/NULL record queries to external domains  
  
DNS Data:  
[paste tshark DNS export here]
```

Pattern 2: Timeline Reconstruction

```
Given these TCP conversations sorted by start time,  
reconstruct the attack timeline:
```

1. When did the initial compromise occur?
2. What reconnaissance was performed?
3. When did lateral movement begin?
4. When was data exfiltrated?

Conversation data:

[paste tshark conversation list here]

AI Prompt Patterns (continued)

Pattern 3: C2 Protocol Identification

```
I have HTTP traffic between an internal host (10.0.0.5)
and an external server (203.0.113.50). Analyze the
request/response patterns to determine:
```

1. Is this likely C2 traffic? Why?
2. What C2 framework might this be? (Cobalt Strike, Metasploit, custom, etc.)
3. What is the beacon interval?
4. Are there any tasking/response patterns visible?

```
HTTP headers and partial payloads:
```

```
[paste extracted HTTP data here]
```

Pattern 4: Exfiltration Quantification

Analyze these outbound data transfers and estimate:

1. Total data volume exfiltrated
2. Exfiltration method(s) used
3. Encoding/compression detected
4. Duration and rate of exfiltration

Netflow records for outbound traffic:

[paste netflow data here]

Python + Scapy for Automated Analysis

```
from scapy.all import rdpcap, DNS, DNSQR, IP, TCP
from collections import Counter
import math

packets = rdpcap("capture.pcap")

# --- DNS Analysis ---
dns_queries = []
for pkt in packets:
    if pkt.haslayer(DNS) and pkt[DNS].qr == 0: # queries only
        qname = pkt[DNSQR].qname.decode()
        dns_queries.append({
            "src": pkt[IP].src,
            "query": qname,
            "type": pkt[DNSQR].qtype
```

Scapy — Beacon Detection

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

packets = rdpcap("capture.pcap")

# Find all SYN packets to a suspect IP
target_ip = "203.0.113.50"
syn_times = []

for pkt in packets:
    if (pkt.haslayer(TCP) and pkt.haslayer(IP)
        and pkt[IP].dst == target_ip
        and pkt[TCP].flags & 0x02): # SYN flag
        syn_times.append(float(pkt.time))
```

Scapy — Exfiltration Detection

```
from scapy.all import rdpcap, IP, TCP, DNS, DNSQR
from collections import defaultdict
import base64

packets = rdpcap("capture.pcap")

# Track bytes per destination
outbound = defaultdict(int)
internal_net = "10.0.0."

for pkt in packets:
    if pkt.haslayer(IP):
        src, dst = pkt[IP].src, pkt[IP].dst
        if src.startswith(internal_net) and not dst.startswith(internal_net):
            outbound[dst] += len(pkt)
```

End-to-End AI Analysis Pipeline

```
import anthropic, subprocess, json

def analyze_pcap_with_ai(pcap_path):
    """Complete pipeline: PCAP → tshark extraction → AI analysis"""

    # Extract HTTP summary
    http = subprocess.run([
        "tshark", "-r", pcap_path, "-Y", "http",
        "-T", "fields", "-e", "frame.time_relative",
        "-e", "ip.src", "-e", "ip.dst",
        "-e", "http.request.method", "-e", "http.request.uri",
        "-e", "http.response.code", "-E", "header=y"
    ], capture_output=True, text=True).stdout

    # Extract DNS summary
```

ICMP Tunneling Detection

ICMP (ping) packets can carry arbitrary data in their payload field. Normal pings have small, predictable payloads.

```
# Normal ping payload (Windows default: alphabet pattern)
# 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70  abcdefghijklmnop

# Suspicious ICMP payload (encoded data)
# 50 4b 03 04 14 00 00 00 08 00 ... (ZIP file header!)

# Detect ICMP packets with unusually large payloads
tshark -r capture.pcap -Y "icmp and data.len > 64" \
  -T fields -e ip.src -e ip.dst -e data.len -e data.data

# Count ICMP packets per source (high volume = tunneling)
tshark -r capture.pcap -Y "icmp.type == 8" \
  -T fields -e ip.src | sort | uniq -c | sort -rn
```

Key Indicators

- **Payload size > 64 bytes** — standard ping uses 32-64 bytes

- **Non-ASCII payloads** — binary data instead of alphabet pattern
 - **High frequency** — hundreds of pings per minute to one host
 - **Asymmetric payload** — request and reply have different data sizes
-

TLS Certificate Analysis

Self-signed and suspicious certificates are strong indicators of malicious infrastructure.

```
# Extract certificate details from TLS handshakes
tshark -r capture.pcap -Y "tls.handshake.type == 11" \
  -T fields \
  -e ip.src -e ip.dst \
  -e x509sat.utf8String \
  -e x509ce.dNSName \
  -e x509af.utctime

# Example output:
# 203.0.113.50 10.0.0.5 "Internet Widgits Pty Ltd" evil.com
# 230101000000Z-240101000000Z
```

Certificate Red Flags

Indicator	Why It's Suspicious
Default organization ("Internet Widgits Pty Ltd")	OpenSSL default — not changed by attacker
Self-signed certificate	No CA validation, easy to generate

Indicator	Why It's Suspicious
Very short validity period	Temporary infrastructure
Mismatched CN / SAN	Certificate doesn't match the domain
Let's Encrypt on suspicious domain	Free certs, commonly used by attackers

Putting It All Together — Analysis Workflow

1. **Orientation** — protocol hierarchy, conversation list, capture duration
2. **Identify endpoints** — map internal vs external hosts, find top talkers
3. **DNS analysis** — resolve suspicious domains, check for tunneling patterns
4. **HTTP inspection** — examine requests/responses for credentials, C2, data leaks
5. **Encrypted traffic** — TLS SNI, certificate analysis, JA3 fingerprints
6. **Timing analysis** — identify beacons, correlate events across protocols
7. **Data flow** — quantify bytes per destination, identify exfiltration
8. **AI-assisted review** — feed structured extracts to Claude for pattern analysis
9. **Timeline reconstruction** — build the attack narrative from evidence

Pro Tip

Start with `tshark -r file.pcap -q -z io,phs` (protocol hierarchy) to see what protocols are present before diving into specific filters. This saves enormous time on large captures.

Quick Reference — tshark One-Liners

```
# Protocol hierarchy (what's in this capture?)
tshark -r cap.pcap -q -z io,phs

# All unique destination IPs
tshark -r cap.pcap -T fields -e ip.dst | sort -u

# HTTP requests with full URLs
tshark -r cap.pcap -Y "http.request" -T fields \
  -e http.request.method -e http.host -e http.request.full_uri

# All user-agents (fingerprint tools/malware)
tshark -r cap.pcap -Y "http.user_agent" -T fields \
  -e ip.src -e http.user_agent | sort -u

# TLS SNI values (what domains are being accessed via HTTPS)
```

CTF Challenge: Operation Shadow Beacon

Scenario

Your SOC detected anomalous traffic from a developer workstation (10.0.0.5). A PCAP was captured over 2 hours. Your mission: analyze the capture to identify the attacker's C2 server and determine what data was exfiltrated.

Challenge Objectives

1. **Identify the C2 server IP and port** — find the beacon in the traffic
2. **Determine the beacon interval** — calculate timing with statistical analysis
3. **Find the exfiltration channel** — how is data leaving the network?
4. **Reconstruct the exfiltrated data** — decode and recover the stolen content
5. **Build the attack timeline** — from initial compromise to exfiltration

6. **Extract the flag** — hidden in the traffic

CTF Challenge — Hints & Approach

Step 1: Orientation

```
tshark -r shadow.pcap -q -z io,phs
tshark -r shadow.pcap -q -z conv,tcp
tshark -r shadow.pcap -q -z endpoints,ip
```

Step 2: DNS Recon

```
tshark -r shadow.pcap -Y "dns" \
-T fields -e ip.src \
-e dns.qry.name | sort -u
```

Step 3: Beacon Hunt

```
# Check timing to suspect IPs
tshark -r shadow.pcap \
-Y "ip.dst==SUSPECT_IP" \
-T fields -e frame.time_relative
```

Step 4: Exfil Recovery

```
# Look for Base64 in DNS
tshark -r shadow.pcap \
-Y "dns.qry.name.len > 50" \
-T fields -e dns.qry.name
```

Use Claude to analyze extracted conversation data and help identify patterns you might miss manually.

CTF Challenge — AI-Assisted Solution

```
import anthropic, subprocess

# Extract all the data we need
def extract(pcap, display_filter, fields):
    cmd = ["tshark", "-r", pcap, "-Y", display_filter,
           "-T", "fields"] + [f for field in fields
                              for f in ("-e", field)]
    return subprocess.run(cmd, capture_output=True, text=True).stdout

pcap = "shadow.pcap"
dns_data = extract(pcap, "dns.qr==0",
                   ["frame.time_relative", "ip.src", "dns.qry.name", "dns.qry.type"])
http_data = extract(pcap, "http",
                   ["frame.time_relative", "ip.src", "ip.dst",
                    "http.request.method", "http.request.uri", "http.response.code"])
```

Common CTF PCAP Challenges

Challenge Type	What to Look For	Tools
Hidden flag in HTTP	Response bodies, headers, cookies	<code>tshark --export-objects</code>
DNS exfiltration	Base64/hex in subdomains, TXT records	Scapy, tshark + decode script
Credential theft	FTP, Telnet, HTTP Basic/Form auth	<code>tshark -z credentials</code>
File transfer recovery	FTP-DATA, SMB, HTTP objects	Wireshark "Export Objects"
Steganography in packets	Unusual ICMP payloads, TCP ISN encoding	Custom scripts, Scapy
Wireless (802.11)	WPA handshake, deauth frames, probe requests	aircrack-ng, Wireshark
VoIP / RTP	Audio streams, DTMF tones	Wireshark "Telephony" menu

Notes: Always check the protocol hierarchy first. If you see unusual protocols (ICMP with large payloads, DNS with many TXT queries), those are likely where the challenge flag is hidden.

Key Takeaways

Technical Skills

- Capture with `tcpdump`, analyze with `tshark`
- BPF and display filters for surgical extraction
- TCP handshake, HTTP, and DNS protocol analysis
- Beacon detection through timing analysis
- DNS tunneling and ICMP covert channel detection
- TLS metadata and certificate analysis

AI Integration

- Extract structured data before feeding to AI
- Use specific prompt patterns for each analysis type
- AI excels at pattern recognition in large datasets
- Combine automated extraction with AI interpretation
- Build reusable analysis pipelines
- Always verify AI findings against raw data

Next Lesson

Lesson 04: Code Auditing — Using AI to find vulnerabilities in source code for CTF challenges and security assessments.
