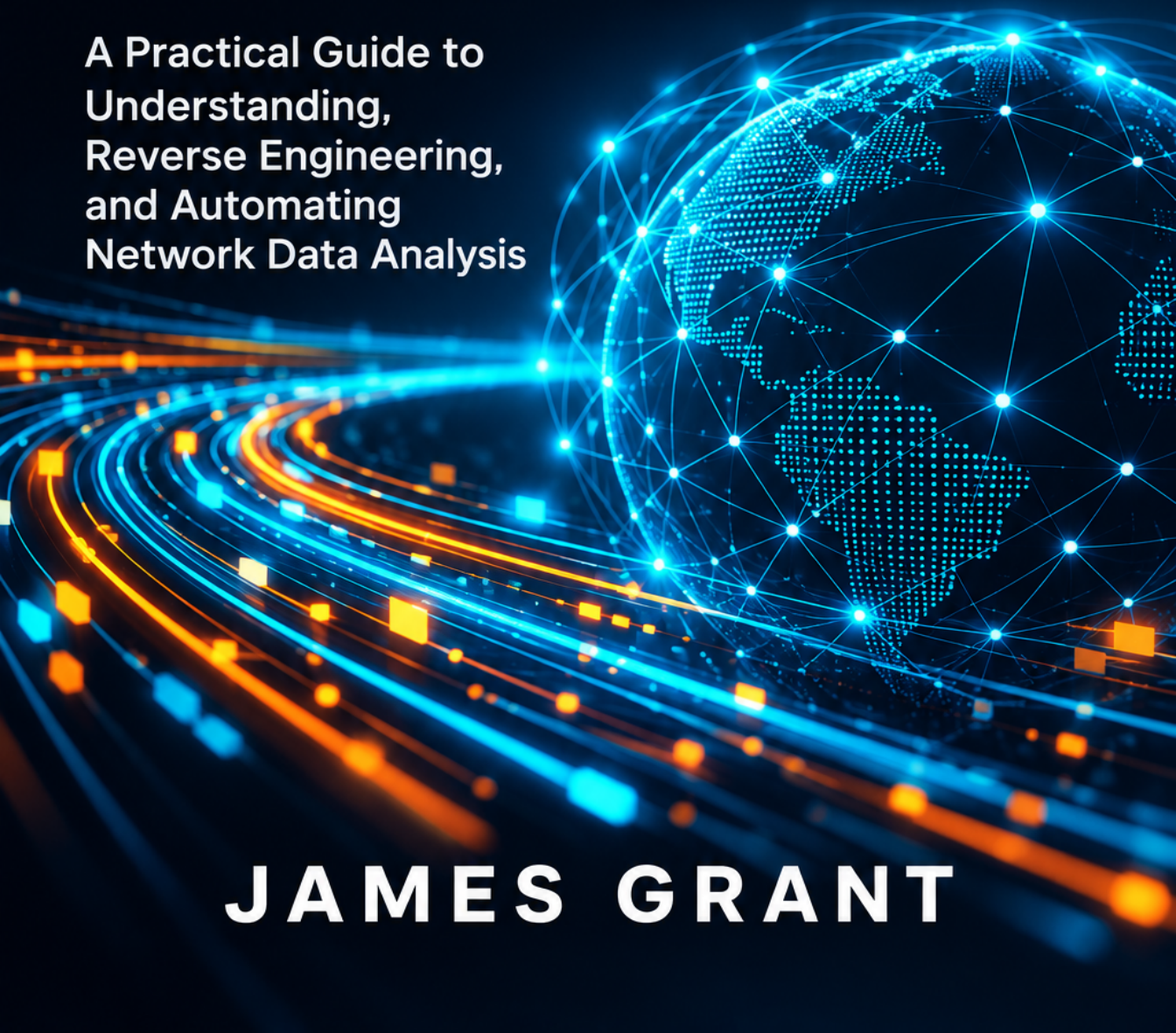
DECODING THE NETWORK WITH CLAUDE CODE

A Practical Guide to Understanding, Reverse Engineering, and Automating Network Data Analysis



JAMES GRANT

Decoding the Network with Claude Code

A Practical Guide to Understanding, Reverse Engineering, and Automating Network Data Analysis

Steve Publications

This book is available at
https://leanpub.com/decodingthenetworkwithclaudecode

This version was published on 2026-09-14

Contents

A Practical Guide to Understanding, Reverse Engineering, and Automating Network Data Analysis

This book teaches you how to investigate network protocol traffic with discipline, rigour and the right tools. You will learn a systematic methodology for examining packet captures and live traffic, from the basic structure of network protocols through advanced reverse engineering and statistical analysis. Claude Code serves as the analysis partner throughout: we will show you how to use it as a programmer, investigator and engineering agent while maintaining independent verification of every conclusion. The goal is practical competence: by the end, you will be able to approach unfamiliar network data with confidence, build working analysis pipelines and document your findings in a way that others can reproduce and trust.

Introduction

On a Thursday morning at 07:32, an operations engineer receives an alert: a new internal service is generating 15 gigabytes of traffic per hour to an unfamiliar port on a database server. The service was deployed yesterday. There is no documentation for the protocol it uses. The on-call engineer captures traffic with tcpdump, opens the capture in Wireshark and sees only binary blobs on port 19841. Within an hour, someone needs to know whether this traffic is normal service behaviour, a misconfiguration or something worse.

This is the moment where protocol analysis matters. Not the comfortable version from a training exercise with labelled examples, but the real thing: unfamiliar data, incomplete context and a decision that has to be made. This book exists to prepare you for that moment and to make you better at every other protocol analysis task that comes before and after it.

What Protocol Analysis Is Not

Before defining what protocol analysis is, it helps to state what it is not.

It is not pattern matching. Opening Wireshark and searching for known signatures works for common protocols and well-known attacks, but it leaves you helpless when facing something new. Real protocol analysis requires understanding structure and behaviour from first principles, not just recognising what you have seen before.

It is not tool use. Knowing which Wireshark filters to type or how to configure Zeek scripts is valuable, but tools without methodology produce confident wrong answers faster than careful human inspection alone. The tools we cover in this book are powerful, and we will teach you how to use them effectively. But the tool is not the craft.

It is not AI prompt engineering. Claude Code and other AI systems can accelerate analysis enormously when used correctly. They can parse binary data, generate Python scripts, spot patterns and document findings. But they also hallucinate structures, invent plausible field names and confidently explain nonsense. Treating AI output as authority rather than hypothesis is how analysis fails.

What Protocol Analysis Is

Protocol analysis is the disciplined investigation of network communication through packet-level evidence. It is a structured form of detective work with a specific methodology:

1. You start with a question. Why is this service sending so much traffic? Is this protocol behaving correctly? What is communicating on this port?
2. You gather initial evidence. You capture traffic, you inspect headers, you follow streams, you note what you can directly observe without interpretation.
3. You form hypotheses. This traffic might be a custom RPC protocol. The volume might indicate excessive retries. The binary structure might follow a Type-Length-Value pattern.
4. You test those hypotheses. You look for confirming evidence and, equally importantly, for evidence that would disprove them. You write parsers to extract fields. You compare message patterns across sessions. You check whether observations are consistent.
5. You document your conclusions with appropriate confidence. Some facts are directly observed: a packet contained these bytes at this timestamp. Others are reasonable inferences: these bytes appear to encode a request-response pair. Still others are working hypotheses that require further testing.

The distinction between fact, inference and hypothesis is not academic. It is the difference between an investigation that produces reliable results and one that builds a house of cards that collapses when someone asks how you know.

Why Claude Code

Claude Code is Anthropic's agentic coding tool that lives in your terminal. It reads your codebase, edits files, runs commands and manages multi-step tasks through natural language interaction. For protocol analysis, it is valuable for several reasons.

First, it can program. Protocol analysis increasingly requires custom scripts to process packet data, extract fields and perform statistical analysis. Claude

Code can write, debug and run Python code that integrates with Scapy, tshark and other tools. It can generate parsers for binary protocols, build analysis pipelines and automate repetitive inspection tasks.

Second, it can reason about structure. Given a hex dump or a set of packet captures, Claude Code can identify patterns, suggest message boundaries and propose field layouts. It recognises common protocol idioms like length-prefixed framing, Type-Length-Value structures and magic number headers.

Third, it can manage context. Protocol analysis involves many moving parts: multiple streams, many observations, evolving hypotheses. Claude Code can maintain project structure, keep analysis notes organised and track the state of an investigation across sessions.

But Claude Code is not a protocol analyst. It does not understand networking fundamentals unless you teach them in the context window. It does not know what constitutes normal traffic on your network unless you tell it. And it has no internal model of truth about the traffic you are analysing. Its output is probabilistic text generation, not evidence. Using it effectively means treating it as a highly capable junior analyst: fast, creative and dangerous to trust without verification.

Throughout this book, we will show you concrete patterns for using Claude Code as an analysis partner while maintaining rigorous standards for evidence and validation. You will see how to structure projects, write prompts that produce useful results and systematically verify everything Claude Code claims.

Who This Book Is For

This book assumes you are comfortable with command-line tools and basic networking concepts. You should know what an IP address is, understand the difference between TCP and UDP and be able to write a simple Python script. If you are a network engineer who has used Wireshark but never analysed unfamiliar protocols, this book will give you a systematic methodology. If you are a security analyst who wants to move beyond signature-based detection, this book will teach you how to investigate new threats. If you are a developer approaching protocol analysis for the first time, this book will give you the fundamentals plus the practical skills to do real work.

The book is also written for experienced practitioners who want to incorporate Claude Code and modern tooling into their workflow. You will find specific

patterns for using AI as an analysis partner, building scalable analysis pipelines and maintaining reproducibility in complex investigations.

How to Read This Book

The book is structured as a progression from fundamentals to advanced practice.

Part 1 establishes the foundations: how network protocols actually work, what evidence is available in packet captures and how to use Claude Code effectively as an analysis partner.

Part 2 introduces the investigation methodology: a systematic approach to protocol analysis that threads through the entire book and underlies every technique we cover.

Part 3 covers capture and processing: the tools and techniques for getting packet data from the network into a form you can analyse.

Part 4 develops core analysis skills: stream reconstruction, protocol detection, message parsing and understanding protocol structure and state.

Part 5 moves into advanced analysis: statistical methods, anomaly detection, cross-flow correlation and automated pipelines.

Part 6 presents complete case studies: end-to-end investigations that demonstrate the methodology in action with realistic scenarios.

The final section addresses production concerns: building reliable analysis systems, maintaining reproducibility and making operational decisions from analysis results.

Every chapter contains working code examples and command-line demonstrations. You should follow along where possible. Running a tshark command yourself, writing a Scapy script, or inspecting a packet capture produces understanding that reading alone cannot.

A Note on Ethics and Safety

Protocol analysis often involves accessing network traffic that contains sensitive data: credentials, personal information, proprietary business data and

potentially malicious content. This book addresses these concerns throughout, but some principles are worth stating upfront.

Capture traffic only on networks where you have explicit permission. Analyse PCAP files only from sources you are authorised to investigate. Handle captured data with the same care you would apply to production databases. When working with potentially malicious traffic, use isolated environments and follow malware analysis safety practices.

The examples in this book use legally appropriate sample captures and generated test data. We provide instructions for creating controlled test captures so you can reproduce the examples without accessing production network traffic.

The Central Thesis

This book is built around one central idea: protocol analysis is a disciplined investigative craft that combines systematic methodology, appropriate tooling and careful reasoning about evidence and uncertainty. Claude Code and modern analysis tools amplify your capabilities enormously when used within that discipline. They reduce the time spent on repetitive tasks, surface patterns you might miss and accelerate the iteration cycle between hypothesis and test. But they do not replace the fundamental skills: understanding network protocols, reasoning about evidence, maintaining appropriate scepticism and validating conclusions independently.

If you walk away with only one insight from this book, let it be this: every observation must be traceable to evidence, every hypothesis must be tested and every conclusion must be open to revision. Whether you are using Claude Code to write a parser or tshark to extract a stream, the standard for what counts as a justified conclusion remains the same.

The rest of this book is about how to meet that standard in practice.

Chapter 1: How Networks Talk

When you open a packet capture and see hundreds of packets with fields you have never noticed before, the question is not what protocol this is. The question is how does this protocol work and what evidence is available to understand it? This chapter establishes the mental models you need to answer those questions. We cover what protocols actually are, how layered communication works in practice and the key concepts that govern how data moves across networks. This is not a networking tutorial in the abstract sense. Every concept we cover maps directly to something observable in packet data.

What a protocol really is

A protocol is a set of rules that governs communication between entities. That definition is correct but unhelpful. A better description is: a protocol is a shared agreement about structure, behaviour and meaning.

Structure refers to the format of messages. How are bytes arranged into fields? What does each field contain? How do you know where one message ends and another begins? Structure is the part of a protocol that you can see directly in packet captures. When you look at a hex dump and see repeating patterns, fixed headers, or length fields, you are looking at protocol structure.

Behaviour refers to how entities interact over time. What message does each side send first? How does one entity respond to another? What happens when a message is lost or corrupted? Behaviour is visible in captures as sequences of messages exchanged between hosts. A TCP three-way handshake is a behavioural pattern. A request-response cycle is a behavioural pattern. Stateful interaction where each message depends on what came before is a behavioural pattern.

Meaning refers to what messages represent. A field containing the value 404 does not mean anything until you understand it within HTTP status codes. A byte sequence that encodes “CONNECT” means nothing without knowing it is part of a connection request protocol. Meaning is the hardest part to recover

from packet data alone because it often depends on context that is not in the packets.

These three elements define what you can learn from a protocol by observation:

- Structure is directly observable from packet bytes
- Behaviour is directly observable from message sequences
- Meaning must be inferred from structure, behaviour and external knowledge

Protocol analysis is largely the process of recovering structure and behaviour from observations and then using those to infer meaning. Claude Code excels at pattern recognition that helps with structure discovery and can suggest behaviours and meanings based on training data. But it can also invent plausible-sounding meanings that are wrong, so verification against actual observations is essential.

The layers that matter (and which ones do not)

The OSI model with its seven layers is a pedagogical tool that rarely maps cleanly to real protocols. Protocol analysts benefit from a simpler mental model based on actual observable layers in packet data.

At the physical bottom sits the link layer. This is Ethernet, Wi-Fi, or some other mechanism for putting bits on a wire. The link layer header contains MAC addresses and a type field that indicates what protocol is encapsulated. In a capture, you can see link-layer information only if you captured at a point that had access to it. If your capture is from a spanned port, you see Ethernet frames. If your capture is from a host tap after the NIC, you might see Ethernet or you might see only IP and above. If your capture is from a virtual interface or tunnel, you might see only the inner protocol. The link layer also tells you about the local network topology: which devices are directly connected, which MAC addresses are present, whether ARP requests indicate address changes or connectivity issues.

Above the link layer is the network layer. IPv4 and IPv6 are the protocols you will encounter. The IP header contains source and destination addresses, protocol field, total length and for IPv4, fragmentation fields. IP is connectionless: each packet is routed independently. This means a capture of IP traffic

shows you individual packets, not sessions and you must reconstruct higher-level communication yourself. IP also tells you about the addressing structure of your network: which subnets exist, which hosts communicate, whether NAT is in play.

The transport layer is where things become interesting for analysis. TCP provides ordered, reliable, connection-oriented delivery. UDP provides best-effort, unordered, connectionless delivery. SCTP exists but is rare. These protocols fundamentally change how you analyse traffic.

With TCP, sequence numbers give you a mechanism for reassembling streams. You can reconstruct what the application sent and received, even if packets arrived out of order or were retransmitted. TCP connections have a lifecycle: They are established with a handshake, data is exchanged and they are terminated with a close sequence. Each of these phases is visible in captures and each tells you something about communication.

With UDP, there are no sequence numbers, no reassembly and no connection state visible in the protocol. A UDP "session" is an inference based on port pairs and timing. Multiple applications can share UDP ports and packet loss is silent. UDP analysis requires different assumptions and different techniques.

Above the transport layer sit application protocols. HTTP, DNS, TLS, SSH, SMTP and thousands of custom protocols all use TCP or UDP as transport but define their own structures and behaviours. This is where protocol analysis becomes most important because this is where the actual meaning of communication resides. A protocol analyst's job is to understand what application protocols are doing, how they work and whether they are behaving as expected.

The key insight is: each layer provides specific evidence. Link layer gives you local topology. Network layer gives you addressing and routing information. Transport layer gives you connection state and delivery guarantees. Application layer gives you the actual data and meaning. Protocol analysis requires understanding what each layer can and cannot tell you.

For example, if you see two hosts communicating on port 443, the transport layer tells you it is a TCP connection. The port number suggests TLS. But the port is not definitive evidence: anything can run on port 443. You need to inspect the application-layer payload to confirm TLS. And if TLS is present, you can see metadata about the connection but not the actual data, because TLS encrypts the payload. Knowing what each layer reveals and what it hides

is fundamental to protocol analysis.

Connection-oriented and connectionless communication

Understanding whether a protocol uses connection-oriented or connectionless communication changes everything about how you analyse it.

TCP is connection-oriented. Before any application data flows, the two endpoints perform a three-way handshake: The client sends a SYN segment, the server responds with SYN-ACK and the client acknowledges with ACK. This handshake establishes sequence numbers, negotiates options and confirms that both sides are ready to communicate. The connection persists until explicitly closed. This structure gives you several advantages for analysis.

First, connections have clear boundaries. You can identify where a conversation starts and ends. This makes it easy to isolate individual communication sessions for analysis. In tshark, you can follow TCP streams: tshark -r capture.pcap -z “follow,tcp,hex,N” where N is the stream number. This extracts all data from one connection in order.

Second, TCP provides delivery guarantees. Retransmissions handle packet loss. Sequence numbers allow reassembly of out-of-order data. If you capture all packets for a TCP connection, you can reconstruct the complete data stream that each side sent.

Third, TCP state is visible. The flags in TCP headers tell you what is happening: SYN for connection initiation, ACK for acknowledgments, FIN for graceful closure, RST for abrupt termination. Wireshark’s TCP dissector tracks state and flags anomalies like retransmissions, out-of-order segments and duplicate acknowledgments. These signals tell you about network quality, application behaviour and potential problems.

UDP is connectionless. Each datagram is independent. There is no handshake, no sequence numbers, no delivery guarantee. This has implications for analysis.

First, you must define sessions yourself. Two hosts exchanging UDP packets on particular ports might constitute a session, or they might be unrelated transmissions. You look for patterns: repeated port pairs, request-response timing, or protocol-specific session identifiers in the payload.

Second, you cannot guarantee completeness. If a UDP packet is lost, there is no retransmission. Your analysis might be based on incomplete data and you may not know it.

Third, many application protocols run over UDP while maintaining their own connection semantics. DNS uses UDP for individual queries but each query-response pair is a logical transaction. QUIC, used by modern HTTP/3, multiplexes multiple streams over UDP while implementing its own reliability and ordering. Protocol analysis of UDP traffic requires understanding the application protocol's own state management.

The distinction matters when you approach unknown traffic. If you see TCP traffic to an unfamiliar port, you can follow the stream and see the complete data. If you see UDP traffic, you must work harder to establish what is happening and accept that your picture might be incomplete.

Addressing, routing and the path a packet takes

When you analyse a capture, the addresses in IP headers are the endpoints visible at your capture point. They are not necessarily the original sender or final recipient. Understanding this distinction prevents a common category of mistakes in protocol analysis.

Routing moves packets across networks. Each router makes forwarding decisions based on the destination IP address. The packet may traverse dozens of networks before reaching its destination. If you capture traffic on one link in that path, you see the packet as it appeared on that link. You do not see what happened before or after.

Network Address Translation (NAT) complicates this further. NAT devices modify source or destination addresses as packets pass through. A private address like 192.168.1.100 might appear as 203.0.113.50 to the rest of the internet. If you capture traffic after NAT, you see the translated address, not the original. You cannot directly determine which internal host originated traffic to an external server without additional information from the NAT device.

Protocol analysts must track this carefully. In a capture, note which addresses you see and where the capture point is relative to network boundaries. If you see many different private addresses mapped to a single public address, NAT is present. If you see traffic between addresses in different subnets but

the capture is on a single subnet, routing is involved. If you see unexpected address changes, something is translating addresses.

For protocol analysis, the addresses tell you about communication patterns. Host A talks to host B on port 80: that is likely HTTP. Host A talks to host B on port 53: that is likely DNS. But addresses alone do not tell you what protocol is in use or whether the communication is legitimate. The application-layer payload provides that evidence.

Routing also affects timing. Packets that travel longer paths take longer to arrive. Retransmissions and out-of-order delivery often result from routing changes or congestion along the path. TCP analysis tools can detect these conditions and flag them for review.

Ports, sockets and multiplexing

Ports are the mechanism that allows multiple applications to share a single network interface. When a process wants to communicate over a network, it binds to a port. TCP and UDP both use 16-bit port numbers, giving 65535 possible ports per protocol per IP address.

Well-known ports (0-1023) are conventionally assigned to standard services: port 80 for HTTP, 443 for HTTPS, 22 for SSH, 53 for DNS. These assignments are documented in IANA registries but are conventions, not rules. Any application can listen on any port, and protocol detection based solely on port number is unreliable. Port 80 might carry HTTP, or it might carry an encrypted custom protocol that happens to use port 80.

The practical implication is: port number is a hint, not evidence. When you see traffic on port 80, assume HTTP until you verify it by examining the payload. When you see traffic on an unusual port, investigate it rather than dismissing it as noise or assuming it is something specific.

Sockets are the endpoint of a connection, identified by a combination of IP address, port and protocol. A TCP connection is uniquely identified by the four-tuple: source IP, source port, destination IP, destination port. This four-tuple is called a flow or a stream in analysis tools. When you ask tshark to list conversations, it groups packets by four-tuple.

Multiplexing means that a single IP address can host many simultaneous connections, distinguished by port numbers. A web server on 192.168.1.10:80

might have hundreds of concurrent TCP connections from different client addresses and ports. Analysis tools must be able to separate these connections to reconstruct individual streams.

Ephemeral ports (typically 49152-65535) are assigned dynamically to client-side connections. When your browser connects to a web server, it uses a random high-numbered port for the client side. This allows many connections from the same client to the same server. In captures, you will see ephemeral ports for client-side endpoints and well-known ports for server-side endpoints, but this pattern is not guaranteed.

Port scanning appears in captures as many connection attempts from one host to many ports on another. This is detectable through the pattern of SYN packets without subsequent data transfer. Port scanning might indicate reconnaissance, service discovery, or misbehaving software.

State and conversations

Protocols often maintain state: information about what has happened so far in a conversation that affects how future messages are interpreted. State can be explicit, stored in protocol fields, or implicit, maintained by the communicating parties.

Explicit state is visible in packet data. HTTP cookies carry state between requests. TLS session IDs allow resumption of encrypted sessions. Custom protocols might include sequence numbers, session tokens, or connection identifiers in their headers. When you see these fields, you can trace state across messages and understand how the protocol manages its conversation.

Implicit state is harder to observe. A TCP connection has implicit state: both sides know the sequence numbers, window sizes and connection status. This state is maintained by the TCP implementations, not stored in the packet payload. But the TCP protocol itself is designed around state, and TCP state machine transitions are documented in RFC 793. When you see SYN, SYN-ACK, ACK, data packets, FIN or RST, you are observing the effects of implicit state.

Protocol analysis often involves reconstructing the state machine that governs a conversation. This is especially important for unknown protocols. By observing which messages appear in which order, which messages trigger

responses and how the conversation ends, you can infer the states that the protocol uses and the transitions between them.

Consider a simple request-response protocol. The client sends a request, the server sends a response. You might observe that certain request types always produce responses while others do not. You might observe that some requests are only valid after an initial handshake message. These patterns reveal the protocol's state machine.

Claude Code can help with state machine reconstruction. Given a set of message sequences, it can identify patterns and suggest state diagrams. But you must verify the suggested states and transitions against actual observations and you must be careful to distinguish correlation from causation. Just because message B often follows message A does not mean A causes B. They might both be effects of some external event.

Conversations can be bidirectional or unidirectional. In a web browsing session, the client sends requests and the server sends responses, but both might initiate communication (for example, with WebSocket or Server-Sent Events). In a logging protocol, clients might send data to a collector without receiving responses. Understanding the directionality of a protocol's conversations is part of understanding the protocol itself.

Conversations also have lifecycles. They start, they exchange data and they end. The patterns of start and end tell you about protocol design. TCP connections have explicit start (handshake) and end (close sequence) procedures. UDP "sessions" might start when the first packet is sent and end when packets stop flowing for a timeout period. Custom protocols might use explicit login and logout messages or they might assume continuous connection. Observing how conversations begin and end is a fundamental step in protocol analysis.

Understanding state and conversations gives you the framework for understanding what protocols do, not just what bytes they carry. A protocol that requires authentication before data transfer has different security properties than one that does not. A protocol that maintains long-lived connections has different resource implications than one that uses short transactions. State and conversation patterns are evidence about protocol design and behaviour.

This foundation of networking concepts maps directly to what you will see in packet captures. Every header field, every sequence number, every port assignment carries evidence about the communication you are analysing. The rest of this book builds on these fundamentals to show you how to extract that

evidence systematically, how to use Claude Code and other tools to accelerate the process and how to maintain rigorous standards for what counts as a justified conclusion.

Chapter 2: The Evidence in Packets

A packet capture is the primary evidence in protocol analysis. It is a record of what appeared on a network link at a particular point in time, preserved in a standard format that analysis tools can read. Understanding what a capture contains, what it can tell you and what it can never tell you is fundamental to doing protocol analysis correctly. This chapter covers the PCAP format, the information available at each layer, the boundaries of what captures can reveal and the limitations you must account for when drawing conclusions.

What PCAP is and is not

PCAP is the file format used by libpcap, the packet capture library that tcpdump, Wireshark, tshark and many other tools rely on. The format is documented in the libpcap savefile specification. A PCAP file contains a 24-octet file header followed by a sequence of packet records, each consisting of a 16-octet packet header and the captured packet data.

The file header identifies the format with a magic number. The value 0xa1b2c3d4 indicates standard microsecond timestamps and big-endian byte order. The value 0xa1b23c4d indicates nanosecond resolution timestamps. Reading the magic number tells you the file format and byte order. The header also contains the snapshot length, which specifies the maximum number of bytes captured per packet, and the link-layer type, which identifies what kind of frame headers are present.

Each packet record has a packet header with four fields: timestamp seconds, timestamp microseconds or nanoseconds, captured length and original length. The captured length might be less than the original length if the snapshot length was set too small, in which case part of the packet was truncated during capture. This is a critical detail: truncated packets contain incomplete evidence and you cannot reconstruct what was cut off.

What PCAP is:

- A faithful record of what appeared on a link at a capture point

- A standardised, portable format that many tools can read
- A source of precise timing information for each packet
- A complete record of all bytes captured up to the snapshot length

What PCAP is not:

- A complete record of all traffic (captures can lose packets)
- A record of traffic at every point in the network (captures are point-in-time and point-in-space)
- A record of decrypted data (encrypted payloads remain encrypted)
- A record of intent or meaning (only bytes are preserved)

The newer PCAPNG format addresses several limitations of classic PCAP. It supports multiple interfaces with different link-layer types in one file, per-packet comments and metadata, nanosecond timestamps and embedded decryption keys. PCAPNG uses a block-based structure with section header blocks, interface description blocks and enhanced packet blocks. The section header magic number is 0x0a0d0d0a. Wireshark uses PCAPNG as its default capture format since version 1.8 in 2012. Modern tools handle both formats.

Understanding the format matters for protocol analysis because it tells you what you can trust. The bytes in a packet are what appeared on the wire (up to the snapshot length). The timestamps are when the capture system saw them. The header fields are what the packet contained. These are directly observable facts. Everything else is interpretation.

Link-layer visibility and blind spots

The link layer tells you about local network topology but only at your capture point. If you capture on an Ethernet interface, you see Ethernet frames with source and destination MAC addresses and an EtherType field that identifies the encapsulated protocol. 0x0800 means IPv4. 0x86DD means IPv6. 0x0806 means ARP. Other values indicate different protocols like VLAN tags or proprietary encapsulations.

MAC addresses identify devices on a local link. They tell you which devices are directly connected to the segment where you capture. ARP traffic reveals the MAC-IP mappings on the local network. Changes in ARP tables can indicate device replacement, MAC spoofing, or connectivity issues.

But link-layer evidence has blind spots. If your capture point is behind a router, you do not see the link-layer headers from beyond that router. The MAC addresses in your capture are only meaningful for the local segment. A packet from another network has the MAC address of the router interface, not the original sender.

If you capture on a virtual interface, tunnel interface, or after network address translation, the link-layer information might not represent physical topology at all. You might see only IP-layer data with no link-layer headers. In some capture modes, you see only traffic destined for your host, missing broadcast and multicast traffic that could provide context about the network.

Link-layer types vary. Ethernet is common but not universal. Wi-Fi captures include 802.11 headers with additional information about radio signals, authentication and association. Token Ring, FDDI and proprietary link types exist but are rare. The PCAP file header tells you the link type. If you see an unexpected link type, it might indicate unusual capture conditions or encapsulation.

For protocol analysis, link-layer evidence is useful for:

- Identifying directly connected devices on a segment
- Understanding local network topology
- Detecting ARP anomalies that might indicate spoofing or misconfiguration
- Confirming whether capture includes all traffic on the segment or only host-directed traffic

It is not useful for determining end-to-end communication paths beyond the local segment or for identifying devices beyond the first router hop.

IP, TCP, UDP: what each header tells you

The IP header is a source of factual evidence about each packet's addressing and routing properties.

An IPv4 header contains: version 4, header length in 32-bit words, type of service, total length, identification, flags including Don't Fragment and More Fragments, fragment offset, time to live, protocol, header checksum, source address and destination address. Each field provides evidence.

The protocol field tells you what transport layer protocol is encapsulated: 6 for TCP, 17 for UDP, 1 for ICMP. This is reliable evidence because it is defined by the IP header itself, not by convention or port number.

The identification field and fragmentation flags tell you about IP fragmentation. If More Fragments is set or fragment offset is non-zero, the packet is part of a fragmented datagram. The identification field groups fragments from the same original datagram. Fragmented packets complicate analysis because the payload is split across multiple packets and you must reassemble them to see the complete message.

The time to live field provides evidence about how far the packet has travelled. Each router decrements TTL by at least one. If you see packets with very low TTL values arriving at a destination, they have traversed many hops. If you see TTL-based anomalies like packets with TTL 1 reaching unexpected destinations, it might indicate traceroute probes or misconfiguration.

An IPv6 header simplifies some of this. It removes the header checksum (upper layers handle integrity checking) and moves fragmentation to an extension header. The Next Header field replaces IPv4's protocol field. The structure is cleaner but conveys similar information.

The TCP header is rich with evidence about connection state and data delivery. It contains source port, destination port, sequence number, acknowledgment number, header length, flags FIN SYN RST PSH ACK URG, window size, checksum, urgent pointer and optional fields.

The sequence and acknowledgment numbers are the most important fields for analysis. The sequence number identifies the first byte of data in this segment. The acknowledgment number identifies the next byte the receiver expects. These numbers allow you to reconstruct the byte stream that each side sent, detect retransmissions, identify out-of-order delivery and measure data transfer volumes.

The flags tell you about connection lifecycle events. SYN indicates connection initiation. SYN-ACK is the server's response in the three-way handshake. ACK acknowledges received data. FIN initiates graceful closure. RST indicates abnormal termination. PSH encourages immediate delivery of buffered data. By tracking flags across packets, you can see how connections start, exchange data and end.

The window size tells you about flow control. A window of zero means the receiver is not ready for more data. Patterns of zero windows might indicate

slow processing on the receiving end or resource exhaustion.

TCP options appear after the main header fields. They negotiate features like maximum segment size, window scaling, timestamps and selective acknowledgments. Options can be useful for protocol fingerprinting because different operating systems and TCP stacks advertise different options.

The UDP header is simple: source port, destination port, length and checksum. There are no sequence numbers, no flags, no connection state. UDP provides best-effort delivery with no guarantees. This simplicity has implications for analysis.

Without sequence numbers, you cannot reassemble streams or detect retransmissions. Without connection state, you cannot identify where a session starts or ends. Each UDP datagram is independent evidence, and you must infer relationships between datagrams from port pairs, timing and payload patterns.

But UDP's simplicity is also a feature. There is no hidden state to reconstruct, no sequence number logic to understand. If you capture all UDP packets for a flow, you have all the evidence. Missing packets are simply gaps in the record, not mysteries to solve.

Payloads and encryption boundaries

The payload is the application data carried by the transport protocol. This is where protocol analysis becomes most interesting and most challenging.

Unencrypted payloads are directly readable. HTTP requests and responses contain human-readable method names, URLs, headers and bodies. DNS queries contain domain names. FTP commands are text-based. These protocols are transparent: you can see exactly what is being communicated.

Encrypted payloads are opaque. TLS, SSH, IPsec and custom encryption all produce byte sequences that look like random noise. You can see that encrypted data is being transferred. You can measure volumes, timing and direction. But you cannot see the actual content.

This creates an encryption boundary in protocol analysis. On one side of the boundary, you can see everything. On the other side, you can see only metadata. Understanding where the boundary is matters because it defines what evidence is available.

TLS is the most common encryption boundary. A TLS connection starts with a handshake that is not encrypted. During the handshake, the client and server negotiate protocol version, cipher suite and session keys. The client sends a ClientHello with supported ciphers, extensions and elliptic curves. The server responds with ServerHello selecting the parameters. This handshake is visible in packet captures and provides evidence about the TLS implementation, the certificate being used and the negotiated security parameters.

After the handshake, all application data is encrypted. You see TLS Application Data records but cannot read their contents. You can observe record sizes, which sometimes reveal information about the underlying application protocol. A large burst of data might be a file download. Small periodic messages might be keep-alives or heartbeats. But the actual content is hidden.

There are techniques for analysing encrypted traffic without decrypting it. Traffic analysis examines patterns in volumes, timing and direction.

For protocol analysis of encrypted traffic, focus on what is observable: handshake details, certificate information, traffic patterns and metadata. Do not speculate about encrypted content. If decryption keys are available, you can decrypt the traffic, but this requires specific configuration during capture or access to the endpoints' keys.

Mixed traffic captures are common. Some flows are encrypted, some are not. Some protocols use encryption selectively. Understanding the encryption boundaries in your capture helps you know where you can analyse content directly and where you must rely on metadata.

Timestamps, sequence numbers and reassembly signals

Timestamps in packet captures are the primary mechanism for understanding when things happened. The PCAP packet header contains a timestamp for each packet, typically with microsecond or nanosecond resolution. These timestamps tell you the order in which packets arrived at the capture point.

But timestamps have limitations. They depend on the clock of the machine doing the capture. If that clock is not synchronised using NTP or similar mechanisms, timestamps may drift or jump, making correlation with events on other systems unreliable. If you are correlating multiple captures from different points, clock synchronisation between capture systems is essential.

Capture overhead can introduce timing errors. At high packet rates, the capture system might drop packets or introduce delays between when packets arrive and when they are recorded. These delays are usually small but can be significant for timing-sensitive analysis.

TCP sequence numbers provide a mechanism for reassembling streams. Each byte in a TCP stream has a sequence number. By tracking sequence numbers across segments, you can reconstruct the complete byte stream that each side sent, even if packets arrived out of order.

The initial sequence number (ISN) is chosen randomly during connection establishment. Analysis tools subtract the ISN from subsequent sequence numbers to produce relative offsets that are easier to work with. Wireshark and tshark do this automatically when following streams.

Retransmissions appear when the same sequence number range is sent more than once. Wireshark flags these with the "TCP Retransmission" tag. Retransmissions indicate packet loss or timeout conditions in the network. They are important for analysis because they tell you about delivery quality and potential issues.

Out-of-order segments arrive with sequence numbers that do not match the expected next byte. Wireshark flags these when it detects gaps in the sequence. Out-of-order delivery can indicate routing changes, network congestion, or implementation issues.

The tcp.completeness filter in Wireshark is useful for verifying that you have complete evidence for a connection. It uses bitmask values: 1 for SYN, 2 for SYN-ACK, 4 for ACK, 8 for DATA, 16 for FIN, 32 for RST. A value of 24 (SYN + SYN-ACK + ACK + DATA) indicates a complete connection with data transfer. Values less than this might indicate incomplete captures or connection issues.

For protocol analysis, understanding reassembly signals matters because incomplete or out-of-order data can lead to incorrect conclusions about protocol behaviour. If you are parsing application-layer messages from a TCP stream and packets were lost or out of order, your parser might see garbled data and infer incorrect structure. Always check stream completeness before trusting reconstructed data.

What you can never see in a capture

A packet capture is limited by what appears on the wire at the capture point. Many important things never appear on the wire or are lost before they reach the capture point.

You cannot see dropped packets. If a network device drops a packet before it reaches your capture point, the packet does not exist in your capture. You might infer that it was dropped from TCP retransmissions or other signals, but you never see the original packet.

You cannot see endpoint processing. The capture shows what was sent and received on the wire, not what happened on the endpoints. You cannot see application-level state, processing delays, memory contents, or decisions made by software. Two applications might send identical packets but have completely different internal states and behaviours.

You cannot see traffic you did not capture. If your capture filter excluded certain traffic, that traffic is gone. If your capture point does not see certain traffic due to routing or network topology, that traffic is also gone. Be explicit about what your capture includes and excludes.

You cannot see decrypted data without keys. Encrypted payloads remain encrypted regardless of your analysis tools. Do not speculate about encrypted content.

You cannot see the intent of communication. The capture shows what was sent, not why. A large data transfer might be a legitimate file backup, a malware exfiltration, or a misconfigured service. The bytes do not tell you which.

You cannot see the complete path. Routing information is not preserved in packets (except in unusual cases like IP options). You cannot determine which routers a packet traversed or where delays occurred along the path.

You cannot see lost context. If a capture starts mid-conversation or stops before a conversation ends, you have partial evidence. Protocol state established before the capture started is invisible. Events that occurred after the capture ended are also invisible.

These limitations are not weaknesses of the PCAP format. They are fundamental properties of passive observation. A capture is a partial record of network communication, and responsible analysis accounts for what is missing as carefully as what is present.

The practice implications are clear: always note your capture's limitations. Document where the capture point is, what filters were applied, whether timestamps are synchronised and whether you have complete stream data. When drawing conclusions, distinguish between what the capture directly shows, what you can reasonably infer and what remains unknown. This discipline separates rigorous protocol analysis from speculation.

Chapter 3: Claude Code as Analysis Partner

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

What Claude Code is and how it differs from chat interfaces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Setting up your analysis environment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Context management: what to share, when and how

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Effective prompts for protocol analysis tasks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

When Claude Code excels and when it fails

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Trust calibration: validating AI-generated analysis

Chapter 4: The Investigation Method

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Starting from a question, not a tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Forming and documenting hypotheses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Gathering initial evidence: the first thirty minutes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Generating competing hypotheses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Testing and falsifying hypotheses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Documenting confidence levels and assumptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

When to stop and what “solved” actually means

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Chapter 5: Project Structure and Context Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Directory layout for analysis projects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Context files: what Claude Code needs to know

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Note-taking patterns for investigations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Code organisation: modular, testable analysis scripts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Version control and collaboration considerations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Reproducibility: capturing analysis decisions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Chapter 6: Capture and Ingestion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

tcpdump: the foundation, filters, options, pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

tshark: powerful CLI analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Wireshark: when the GUI matters and when it does not

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Capture placement and scope decisions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Capture loss: recognizing and mitigating it

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

File formats and interoperability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Chapter 7: Stream and Session Reconstruction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

TCP streams: sequence numbers and reassembly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

UDP and pseudo-sessions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Handling retransmissions and out-of-order delivery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Fragmentation and its complications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

NAT: recognizing and working through it

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Implementation: stream extraction with tshark and Python

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Chapter 8: Protocol Detection and Classification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Port-based identification and why it lies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

TLS handshake fingerprints and version detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

ASCII signatures and protocol banners

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Statistical protocol detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Unknown protocols: when everything fails

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Automated detection pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Chapter 9: Text and Binary Decoding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Reading text-based protocols: HTTP, SMTP, etc.

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Binary representation: hex, endian and structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Base64, URL encoding and other transformations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Finding structure in binary blobs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Mixed-content protocols

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Claude Code as a decoder: patterns and pitfalls

Chapter 10: Message Boundaries and Framing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Length-prefixed framing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Delimiter-based framing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Fixed-length messages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Session-based and timing-based boundaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Protocol multiplexing and sub-streams

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Practical techniques for boundary detection

Chapter 11: Field Discovery and Schema Inference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Constant vs variable fields

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Numeric fields: integers, floats, bitfields

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

String fields and character encodings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Repeated fields and arrays

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Nested structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Using Claude Code for schema hypothesis generation

Chapter 12: State Machines and Protocol Logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Observable state transitions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Request-response patterns and session flows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Timeout and retransmission behaviour

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Error handling and recovery paths

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Inferring state from message sequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Chapter 13: Fingerprinting and Identification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

TLS fingerprinting (JA3, JA4, etc.)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

HTTP client and server fingerprints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

TCP stack fingerprints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Timing and behavioural fingerprints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Application-level fingerprints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Privacy implications and evasion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Chapter 14: Statistical Analysis and Pattern Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Basic statistics: distributions, outliers, correlations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Traffic volume and timing analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Protocol-level metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Using Python for statistical analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Chapter 15: Anomaly Detection and Suspicious Behaviour

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Defining normal and abnormal

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Signature-based detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Statistical anomaly detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Protocol violations and edge cases

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Malformed and crafted packets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

False positives and validation

Chapter 16: Correlation and Cross-Flow Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Host-centric correlation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Temporal correlation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Payload correlation across streams

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Session linkage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Attack-reconnaissance correlation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Tools for correlation analysis

Chapter 17: Case Study - Reverse Engineering a Custom Protocol

The scenario: new internal service with unknown protocol

Initial reconnaissance and tool selection

Stream extraction and first observations

Hypothesis generation with Claude Code

Iterative decoding and validation

Final protocol documentation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Chapter 18: Case Study - Encrypted and Obscured Traffic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

The scenario: unexpected encrypted service

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

TLS analysis and certificate inspection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

SNI and JA3 analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Traffic pattern analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Metadata and timing analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Limits and what remains unknown

Chapter 19: Case Study - Large-Scale Analysis Pipeline

The scenario: weekly analysis of gigabytes of traffic

Architecture: ingestion, processing, storage

Flow extraction and indexing

Automated protocol classification

Anomaly detection integration

Report generation and validation

Chapter 20: Automated Analysis Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Pipeline architecture patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Zeek for scalable protocol analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Scapy for custom processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Database and columnar storage options

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Orchestration and scheduling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Monitoring and reliability

Chapter 21: Reproducibility, Validation and Trust

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Reproducible analysis environments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Independent verification of Claude Code output

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Validation techniques: cross-checking, manual review

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Documenting analysis methodology

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Peer review and collaboration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

Operational decision-making from analysis

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/decodingthenetworkwithclaudecode.