



Wireshark and tshark for Troubleshooting and Analysis

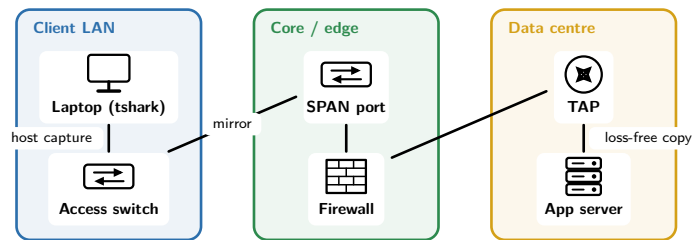
Tools track | Capture vs display filters, per-protocol filters, follow stream, statistics and IO graphs, decrypting your own TLS, tshark field extraction, colouring rules and profiles

#Wireshark #PacketAnalysis #Troubleshooting #TLS #NetworkForensics

👤 Capture: where, with what, and the two filter languages

📌 Scope and vocabulary

Wireshark is the graphical protocol analyser, **tshark** its command-line twin (same dissectors, same display filters), **dumpcap** the small privileged capture engine both call. A capture file is **pcapng** (PCAP Next Generation, default since Wireshark 1.8: several interfaces, comments, embedded TLS secrets) or legacy **pcap**. **BPF** = Berkeley Packet Filter, the kernel filter language used *while capturing*. Capture only on networks and hosts you own or are authorised to monitor; packets contain passwords, cookies and personal data.



Where to put the capture point. A SPAN (Switched Port Analyzer, port mirroring) copy is free but drops frames under load and hides physical-layer errors; a TAP (Test Access Point, passive splitter) is loss-free; capturing on the host itself sees loopback and pre-encryption VPN traffic but can be disturbed by the problem you are chasing. Capture on BOTH ends when you must prove where packets are lost

Capture filter (BPF)	Display filter (Wireshark syntax)
Applied by the kernel while capturing; dropped packets are gone for ever	Applied after capture; hides packets, never deletes them
host 10.0.0.5 and tcp port 443	ip.addr == 10.0.0.5 && tcp.port == 443
Few primitives: host, net, port, portrange, proto, byte offsets	Every dissected field (about 300 000 fields in 4.x)
tshark -f, dumpcap -f, Capture Options dialog	tshark -Y, filter toolbar, colouring rules
Use to keep files small on busy links	Use for everything analytical

Capture filter (BPF)	Keeps
host 10.0.0.5	both directions of one host
net 10.20.0.0/16 and not port 22	a subnet without your own SSH session
tcp port 443 or udp port 443	HTTPS over TCP and QUIC (HTTP/3)
udp port 53 or tcp port 53	all DNS, including zone transfers
tcp[tcpflags] & (tcp-syn tcp-rst) != 0	only SYN and RST: connection attempts

port 67 or port 68	DHCP (Dynamic Host Configuration Protocol)
ether host 00:50:56:aa:bb:cc	one MAC address (device before it has an IP)
vlan and host 10.0.0.5	802.1Q-tagged traffic (tags visible on trunks)

Four display filters that do not do what the analyst thinks

Each struck token is the analyst's version; the correction is written above it.

- Show traffic of one host: ~~host~~ ip.addr == 10.0.0.5
- Port 443 only: tcp.port ~~==~~ 443
- GET requests: http.request.method == ~~GET~~ "GET"
- DNS errors (not NOERROR): dns.flags.rcode ~~>=~~ 0

BPF words (host, port) are not display-filter fields; comparison is ==; strings are quoted; an rcode of 0 means NOERROR, so '>= 0' matches every DNS packet. Wireshark colours the filter bar green (valid), yellow (valid but suspicious, for example a deprecated field) or red (syntax error)

🔍 Essential display filters by protocol

Display filter	Shows	Reading it
tcp.analysis.flags && !tcp.analysis.window_update	every TCP problem Wireshark detected	default 'Bad TCP' colour rule
tcp.analysis.retransmission	segments sent again	loss or a too-short RTO (retransmission timeout)
tcp.analysis.zero_window	receiver buffer full	slow application, not the network
tcp.flags.syn==1 && tcp.flags.ack==0	connection attempts	many with no SYN/ACK = filtered or down
tcp.flags.reset==1	resets	RST right after SYN = port closed
tcp.time_delta > 1	gaps > 1 s inside a stream	needs TCP pref 'Calculate conversation timestamps'
dns.flags.rcode == 3	NXDOMAIN (name does not exist)	typos, DGA malware, broken search suffix
dns.time > 0.5	DNS answers slower than 500 ms	slow resolver: every page load waits
dns.flags.response==0 && !dns.response_in	queries never answered	resolver unreachable or dropped
http.response.code >= 400	HTTP client and server errors	4xx = request side, 5xx = server side
http.time > 2	HTTP responses slower than 2 s	time from request to response
tls.handshake.type == 1	TLS ClientHello	add SNI column: tls.handshake.extensions_server_name
tls.alert_message	TLS alerts	handshake_failure, unknown_ca, bad_certificate

quic	QUIC / HTTP/3 on UDP 443	encrypted from the first flight except the Initial
dhcp.option.dhcp == 1	DHCP Discover	1 Discover, 2 Offer, 3 Request, 5 ACK, 6 NAK
arp.duplicate-address-detected	two MACs claim one IP	IP conflict or ARP spoofing
icmp.type == 3 && icmp.code == 4	fragmentation needed	path MTU (maximum transmission unit) problems
smb2.nt_status != 0	failed SMB2 operations	ACCESS_DENIED, OBJECT_NAME_NOT_FOUND
kerberos.msg_type == 30	Kerberos KRB-ERROR	read kerberos.error_code: 24 pre-auth failed, 37 clock skew
ldap.resultCode != 0	failed LDAP operations	49 = invalidCredentials on a bind

Reading a capture: the TCP handshake behind a slow HTTPS page

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000	10.0.0.23	93.184.215.14	TCP	74	51544 -> 443 [SYN] Seq=0 Win=64240 MSS=1460
2	0.021	93.184.215.14	10.0.0.23	TCP	74	443 -> 51544 [SYN, ACK] Seq=0 Ack=1
3	0.021	10.0.0.23	93.184.215.14	TCP	66	51544 -> 443 [ACK] Seq=1 Ack=1
4	0.022	10.0.0.23	93.184.215.14	TLSv1.3	583	Client Hello (SNI=www.example.org)
5	1.874	93.184.215.14	10.0.0.23	TLSv1.3	1514	Server Hello, Change Cipher Spec

▼ Transmission Control Protocol, Src Port: 51544, Dst Port: 443

Flags: 0x002 (SYN)

Window: 64240

Options: MSS 1460, SACK permitted, window scale 8

Display filter `tcp.flags.syn == 1 && tcp.flags.ack == 0` shows only connection attempts; the highlighted field is the SYN flag in the selected packet. A long gap between SYN and SYN/ACK points at the network or server, a long gap after the ClientHello at TLS or the application

- tcp.stream** — index Wireshark gives each TCP connection (0, 1, 2...); `tcp.stream == 7` isolates one conversation exactly, unlike an address and port filter.
- Expert Info** — Wireshark's own diagnoses (chat, note, warning, error) such as retransmission, zero window or malformed packet; Analyze > Expert Information or `tshark -z expert`.
- Dissector** — decoder for one protocol; Decode As forces a dissector on a non-standard port (HTTP on 8081, TLS on 8443).
- Snaplen** — maximum bytes saved per packet; a short snaplen (`-s 128`) keeps headers only and shrinks files, but payload-based fields become empty.
- Profile** — named set of preferences, columns, colouring rules and filter buttons; switch per task (Edit > Configuration Profiles or `-C name` in tshark).

Operators worth knowing	Right-click shortcuts
<code>ip.addr == 10.0.0.0/8</code> subnet match	Apply as Filter > Selected: builds the exact field filter
<code>tcp.port in {80 443 8080}</code> set membership	Prepare as Filter: edit before applying
<code>http.host contains "api"</code> substring	Apply as Column: any field becomes a sortable column
<code>dns.qry.name matches "[a-z0-9]{20,}"</code> regex	Conversation Filter > TCP: isolates the stream
<code>eth.src[0:3] == 00:50:56</code> byte slice (VMware OUI)	Colorize Conversation: highlight without hiding
<code>frame.time >= "2026-09-28 10:00:00"</code> time window	Copy > as Filter: paste into tshark -Y
<code>len(dns.qry.name) > 50</code> long names	

 Save filters as buttons

Filter toolbar > + creates a one-click button (stored in the profile file `dfilter_buttons`). A troubleshooting profile with buttons for `tcp.analysis.flags`, `dns.time > 0.5`, `http.time > 2` and `tls.alert_message` turns a first look at a capture into four clicks.

 Follow stream, statistics and IO graphs

- Follow TCP / TLS / HTTP stream** reassembles one conversation as text (client red, server blue); `tshark -q -z follow,tcp,ascii,7` for stream 7
- Statistics > Conversations** who talks to whom, bytes and duration per pair: find the top talker or a long-lived beacon
- Statistics > Endpoints** per host totals, with GeoIP columns when MaxMind databases are configured
- Statistics > Protocol Hierarchy** share of each protocol: unexpected SMB or IRC on a web segment stands out
- Statistics > IO Graphs** packets or bytes per interval, one line per display filter (e.g. all traffic vs `tcp.analysis.retransmission`)
- Statistics > TCP Stream Graphs** Stevens (sequence vs time) and throughput graphs: stalls show as flat steps
- Analyze > Expert Information** all warnings and errors grouped, double-click jumps to the packet

The same statistics from tshark (illustrative)

```
analyst@ws

$ tshark -r case.pcapng -q -z conv,tcp | head -6
TCP Conversations
      | <-  | | ->  | | Total | Dur
10.0.0.5:51514 <-> 203.0.113.7:443 1210 kB 88 kB 1298 kB 612.4
10.0.0.5:51530 <-> 198.51.100.9:443 2 kB 3 kB 5 kB 3601.0

$ tshark -r case.pcapng -q -z io,stat,300,"tcp.analysis.retransmission"
| Interval | Frames | Bytes | retrans |
| 0 <- 300 | 18420 | 14 MB | 12 |
| 300 <- 600 | 19002 | 15 MB | 961 |

$ tshark -r case.pcapng -q -z expert,warn | head -4
Warns (1873)
Frequency Group Protocol Summary
961 Sequence TCP This frame is a (suspected) retransmission
412 Sequence TCP Previous segment(s) not captured

$
```

Read it: the second conversation is small but lasts an hour, the shape of a periodic check-in worth a look; retransmissions jump in the second 5-minute interval, so compare with change logs for that window. 'Previous segment not captured' often means capture loss (SPAN overload), not network loss

 Decrypting your own TLS traffic

TLS 1.3 and TLS 1.2 with ECDHE use ephemeral keys, so a server private key no longer decrypts a capture. Instead, have the **client** write its session secrets to a key log file (NSS key log format):

Chrome, Firefox, curl and many libraries honour the SSLKEYLOGFILE environment variable. Do this only on your own systems and test accounts: the file decrypts everything recorded with it.

Key log, capture, decrypt, share (illustrative)

```
analyst@ws
$ export SSLKEYLOGFILE=$HOME/case/keys.log
$ curl -s https://app.example.com/health -o /dev/null
$ head -c 70 $HOME/case/keys.log
CLIENT_HANDSHAKE_TRAFFIC_SECRET 3f2a9c... 7b41e0...
$ tshark -r cap.pcapng -o tls.keylog_file:$HOME/case/keys.log -Y http2 -T fields -e http2.headers.path | head -2
/health
/api/v1/status
$ editcap --inject-secrets tls,$HOME/case/keys.log cap.pcapng cap-dsb.pcapng
$
```

editcap --inject-secrets embeds the keys into the pcapng (Decryption Secrets Block) so a colleague opens one file; treat that file like the key log. In the GUI: Preferences > Protocols > TLS > (Pre)-Master-Secret log filename. QUIC (HTTP/3) is decrypted from the same key log

⚠ Key logs are secrets

Never leave SSLKEYLOGFILE set globally on a workstation or in a container image: every TLS session of every program using a compatible library is written in clear. Unset it after the test and delete or protect the file.

tshark in pipelines and long captures

tshark -D	list capture interfaces with their numbers
dumpcap -i eth0 -f 'not port 22' -b filesize:100000 -b files:20 -w /var/tmp/cap.pcapng	ring buffer: 20 files of about 100 MB, oldest overwritten
tshark -r cap.pcapng -Y 'dns.flags.rcode==3' -T fields -e frame.time -e ip.src -e dns.qry.name	NXDOMAIN answers as tab-separated fields
tshark -r cap.pcapng -Y tls.handshake.type==1 -T fields -e tls.handshake.extensions_server_name sort uniq -c sort -rn	most requested TLS server names (SNI)
tshark -r cap.pcapng -T ek > cap.ndjson	one JSON document per packet, ready for Elasticsearch or jq
capinfos cap.pcapng	duration, packet count, data rate, dropped packets
editcap -A '2026-09-28 10:00:00' -B '2026-09-28 10:05:00' in.pcapng slice.pcapng	cut a time window before sharing
mergcap -w all.pcapng a.pcapng b.pcapng	merge captures chronologically

TIP

Capture with dumpcap (small, privileged) and analyse with tshark or Wireshark as a normal user: add yourself to the wireshark group on Linux instead of running the GUI as root, since dissectors parse hostile input.

LAB Capture your own DNS lookups and measure how long each answer took

BEGINNER10 MIN

PREREQUISITES

- tshark and dumpcap installed, with capture rights (wireshark group or sudo)
- dig (package dnsutils or bind-utils)

SETUP

- Find the interface that carries your default route

\$ tshark -D

STEPS

1

Start a 30-second capture of DNS only

\$ tshark -i eth0 -f 'udp port 53' -a duration:30 -w dns.pcapng

replace eth0 with the name from tshark -D

2

In a second terminal, make one good and one bad lookup

\$ dig @9.9.9.9 example.org A

\$ dig @9.9.9.9 nothing.invalid A

Expect: status: NOERROR, then status: NXDOMAIN in the dig headers

3

When the capture stops, print each answer's name, return code and delay

\$ tshark -r dns.pcapng -Y 'dns.flags.rcode == 1' -T fields -e dns.qry.name -e dns.flags.rcode -e dns.time

CHECKPOINT

Two lines: rcode 0 (NOERROR) and rcode 3 (NXDOMAIN); dns.time is the query-to-answer delay in seconds (typically 0.005-0.05 s)

example.org00.018342

nothing.invalid30.021907

CLEANUP

- Delete the capture: it lists every name looked up during the 30 s

\$ rm dns.pcapng

SAFETY

Capture only on a machine and network you own or are authorised to monitor

Hands-on lab on your own machine. -f is the BPF capture filter, -a duration:30 stops the capture after 30 s, -Y is the display filter applied when reading. dig @9.9.9.9 asks the Quad9 public resolver directly, so the queries leave through the real interface (a system resolver such as systemd-resolved on 127.0.0.53 would keep them on the loopback interface). .invalid is a reserved top-level domain (RFC 6761) that never exists, so it always gets NXDOMAIN

Worked case: 'the application is slow'

- 1 Capture at the client ring buffer during the slow period
- 2 Split the time dns.time, tcp handshake RTT, tls handshake, http.time
- 3 Look at TCP health tcp.analysis.flags: retransmissions, zero windows
- 4 Conclude network (loss, RTT) or server (think time) or client

Measure	Display filter / column	Healthy (LAN)	In this case
DNS answer time	dns.time	< 50 ms	20 ms
TCP handshake	tcp.analysis.initial_rtt	< 5 ms	3 ms
TLS handshake	ClientHello to Finished	< 50 ms	40 ms
Server think time	http.time	< 300 ms	4.8 s
Receiver window	tcp.analysis.zero_window	none	none

Illustrative numbers: the network parts are fast, the wait is between the request and the first byte of the response, so the problem is on the server (application or database), not the network

✓ Self-test

Statement	True	False
1. A capture filter can be changed afterwards to recover dropped packets → what BPF drops is never written	☐	☒
2. ip.addr == 10.0.0.5 matches source or destination → ip.src / ip.dst for one side	☒	☐
3. A server private key decrypts TLS 1.3 captures → ephemeral keys: use a client key log	☐	☒
4. tcp.stream == 7 isolates exactly one TCP connection → stream index assigned by Wireshark	☒	☐
5. Many 'previous segment not captured' warnings always mean network loss → often capture loss on a SPAN port	☐	☒

Wireshark and tshark check

Q1

Display filter for DNS names that do not exist?

A

dns.flags.rcode == 3 (NXDOMAIN)

Q2

How do you share a decryptable capture as one file?

A

editcap --inject-secrets tls,keys.log in.pcapng out.pcapng

Q3

tshark option to print chosen fields?

A

-T fields -e field1 -e field2 (with -Y to filter)