Everything is encrypted

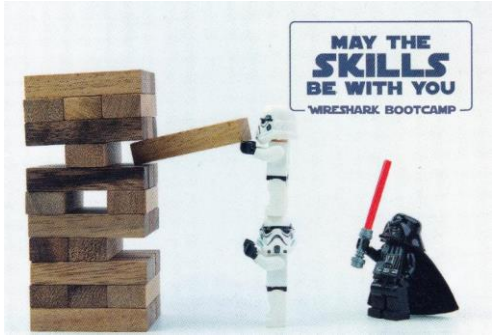
André Luyer
@AndreLuyer

Let me introduce myself

- I am André Luyer
- I use Wireshark and network captures to analyze applications
- And these are my socials:
  @AndreLuyer

Rabobank

Wireshark Users NL Meetup

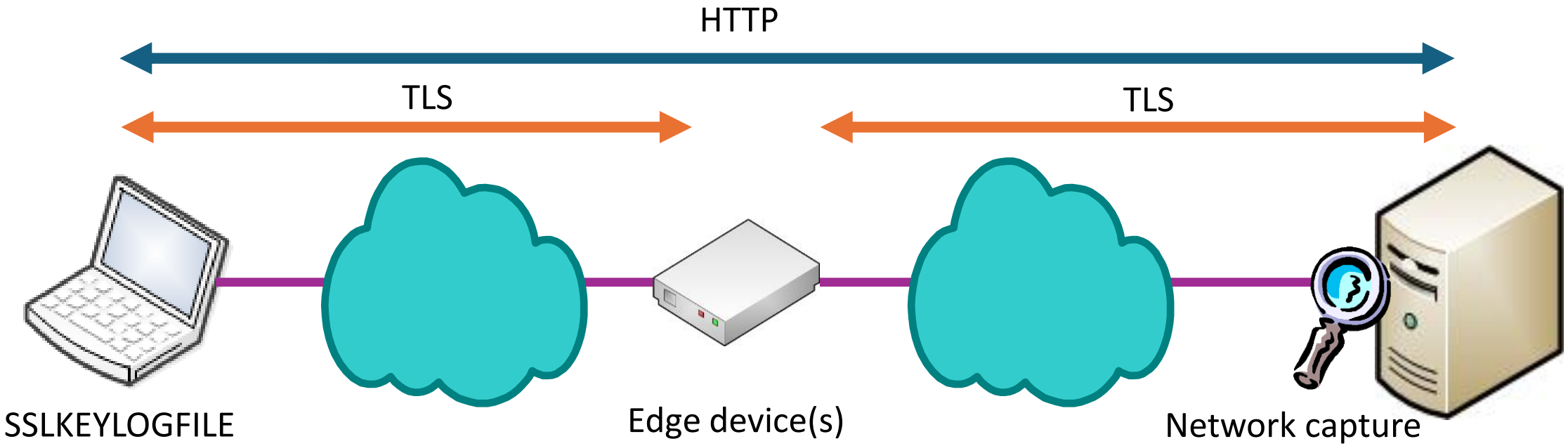


Everything is encrypted

Why would you want to analyze encrypted traffic?

- Production issue: find root cause fast
  - And setting up capture session keys or decrypted traffic takes time
- Snapped packets
- Start not captured
- Decryption not allowed
- Decryption fails

Decryption fails

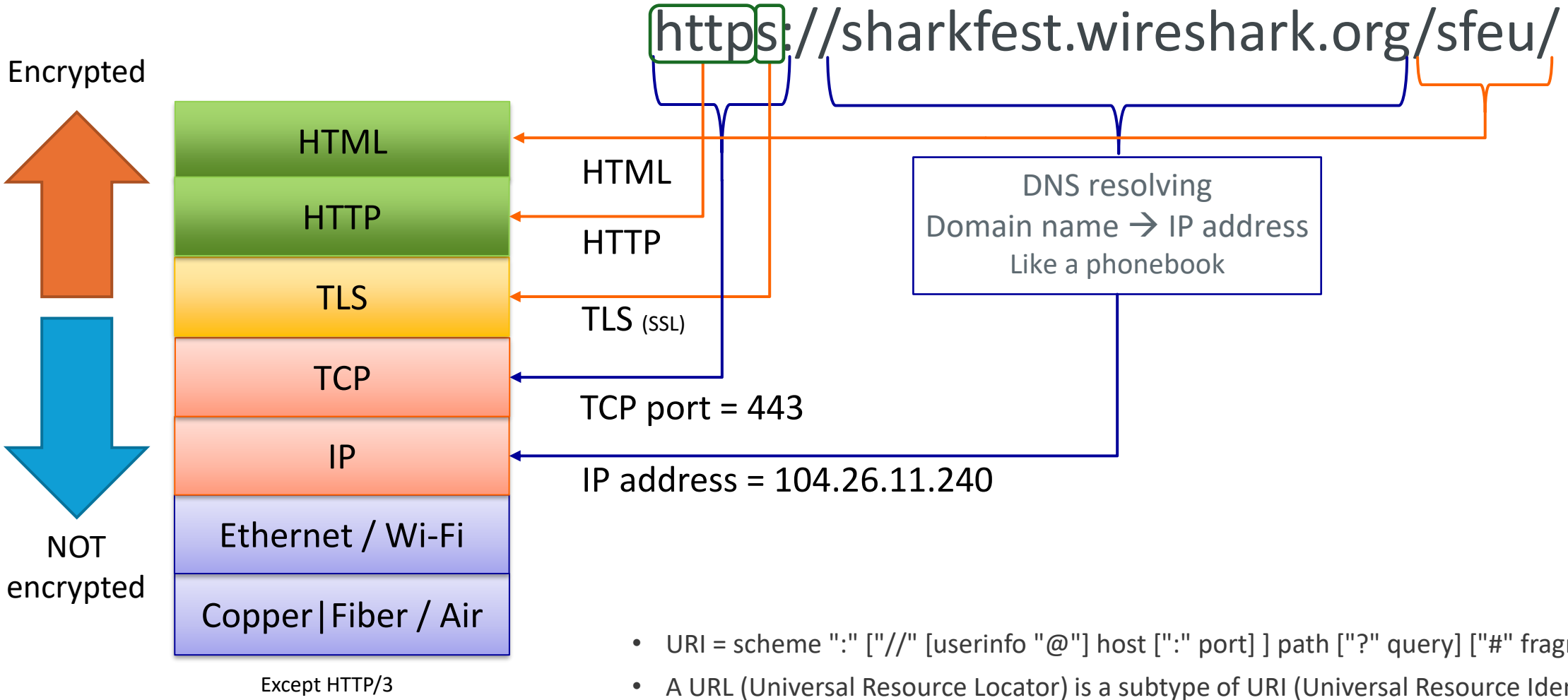


API Application Programming Interface

Hackers: Always Pwning Infrastructure

Edwin van Andel

URL to protocol stack

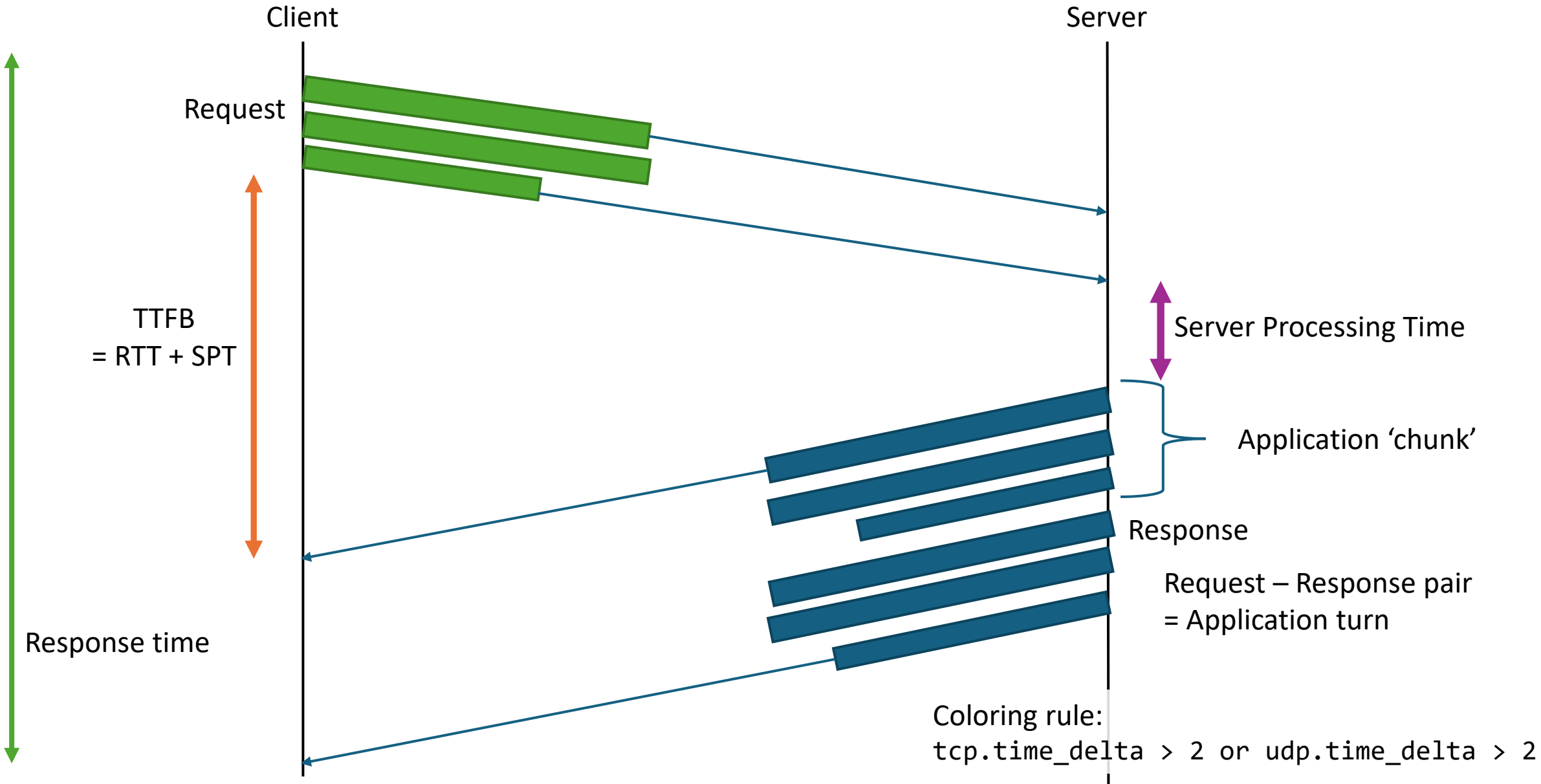


- URI = scheme ":" ["//" [userinfo "@"] host [":" port]] path ["?" query] ["#" fragment]
- A URL (Universal Resource Locator) is a subtype of URI (Universal Resource Identifier) containing both a resource and a protocol
- all URLs are URIs, but not all URIs are URLs

Unencrypted meta data

- IP protocol
  - Source and destination address
    - Capture DNS as well
- TCP
  - Port numbers – can be a clue to top level protocol used
  - Payload size (`tcp.len`)
  - RTT (`tcp.analysis.initial_rtt`)
  - Timing!
  - Maximum Segment Size (MSS) for "full packets"
  - Push flag (`tcp.flags.push`)
  - Who closes / aborts the connection
- TLS
  - Handshake not encrypted (≤ v1.2)
  - Handshake partially not encrypted (v1.3)

Typical application flow – half duplex – TCP



Example of how to calculate application turns:

```
tshark -r $file -Tfields -e tcp.stream -e frame.number -e ip.src
-Y 'tls.record.content_type==23 || tls.record.opaque_type' |
sort -k1,1n -k2,2n |
awk -F '\t' 'function mprint() { print nr, turns/2 }
BEGIN { nr=-1 }
$1 != nr { if (nr > 0) mprint(); turns=0; nr = $1; dir="" }
{ if ($3 != dir) { turns++; dir=$3 } }
END { mprint() }'
```

Filter on Application Data and count direction switches

Pattern interpretation

Tip: Use your own traffic as reference

- Wireshark Tools menu / TLS Keylog Launcher
- SSLKEYLOGFILE
  - Handled by encryption library (OpenSSL, GnuTLS, BoringSSL, …)
  - Also server side (jsslkeylog for Java (e.g. Tomcat), Nginx, …)
  - Not Windows Schannel
- Developer tool in browser
- chrome://net-export
- Application debug logging (e.g. Java `-Djavax.net.debug=all`)

TLS version ≤ 1.2

TLS version ≤ 1.2 Record types

- Different record types clearly visible:

Type	tls.record.content_type (dec)
Handshake	22
Change Cipher Spec (start encryption)	20
Application Data	23
Alert (Close Notify)	21

- Simply filter on `tls.record.content_type == 23` for records that contain payload
  - But `tls.record.length` is not payload size

TLS version ≤ 1.2 Client Hello

Information from Client Hello:

- Supported versions
  - Min version = `tls.record.version`
  - Max version = `tls.handshake.version`
  - If Client supports v1.3 then use Supported Versions extension instead
- Supported Cipher Suites (`tls.handshake.ciphersuite`)
- Extensions – thus optional!!
  - Server Name Indication (SNI) (`tls.handshake.extensions_server_name`)
  - Next protocol: Application Layer Protocol Negotiation (ALPN) (`tls.handshake.extensions_alpn_str`)
- FA3/FA4 Fingerprints (`tls.handshake.ja3 …4`)
  - Cipher Suites list and order to identify browsers

TLS version ≤ 1.2 Server Hello

Information from Server Hello:

- Selected version (`tls.handshake.version`)
  - TLS 1.1 was **deprecated** in 2021
- Selected Cipher Suite (`tls.handshake.ciphersuite`)
- Extension – thus optional
  - Selected next protocol: Application Layer Protocol Negotiation (ALPN) (`tls.handshake.extensions_alpn_str`)
- FA3/FA4 Fingerprints (`tls.handshake.ja3s …4s`)

TLS version ≤ 1.2 Certificate

- Certificate(s) are also send in clear text.
  - Thus also easy to export
- Useful information from certificates:
  - Host name(s) (`x509ce.dNSName`)
  - Common name (`x509if.RelativeDistinguishedName_item_element`)
  - Validity; not before & not after (`x509af.notAfter`)
- Mutual TLS? Then also client certificate
- Not available when resumed

TLS version ≤ 1.2 Application Data

- Application Data, how to determine real payload size

Application Data record	Remark
header | **payload** | MAC | padding	
header | IV | **payload** | MAC | padding	Since 1.1 and block cipher in CBC mode
header | nonce | **payload**	Since 1.2 and AEAD cipher

- Header: type (1 byte) + version (2 bytes) + length (2 bytes)
- Total Appl Data record size is `tls.record.length + 5`
- Shortcut: lookup length of TLS Alert
  It has a fixed 'payload' length of 2 bytes
- Actual size of overhead may vary due to padding (rounding to block size)
  - Small error in size acceptable when looking at patterns or doing statistics

TLS version ≤ 1.2 Alert

- A TLS Alert (Close Notify) is always last record per direction and is small in size
- Followed by TCP-FIN or TCP-Reset
- Alert during handshake is not encrypted

- Calculations:
  `overhead = alert(tls.record.length) - 2`
- Payload size in Application Data:
  `Payload size ≈ tls.record.length – overhead`
- Question is this ok too?
  `Payload size ≈ tcp.len – 5 – overhead`

TLS version ≤ 1.2 Overhead ratio

Example of how to calculate overhead ratio:

```
tshark -r $myfile -Y tcp.port==443 -T fields -e tcp.len
-e tls.record.content_type -e tls.record.length |
awk '{ total += $1; split($2, arr, ","); split($3, lens, ",");
for (tp in arr) if (arr[tp] == "23") payload += lens[tp] - 24 }
END { overhead = total - payload; printf "%6d payload\n%6d
%6.3g%% overhead\n", payload, overhead, overhead/total * 100 }'
```

Total = sum(tcp.len)

Payload = sum(tls.record.length – 24) for each Application Data

TLS version 1.3

TLS version 1.3 Handshake

- Some record type are not encrypted
  Handshake Types:
  - Client Hello
  - Hello Retry Request
  - Server Hello
  - Change Cipher Spec (start encryption)

- Finished record is encrypted, send in both directions, small in size
- Display filter `tls.record.opaque_type` (== 23) present

TLS version 1.3 Client Hello

Information from Client Hello:

- Supported versions
  - Min version = `min(tls.handshake.extensions.supported_version)`
  - Max version = `max(tls.handshake.extensions.supported_version)`
- For backwards compatibility:
  - tls.record.version = TLS v1.0
  - tls.handshake.version= TLS v1.2
- Supported Cipher Suites (`tls.handshake.ciphersuite`)
- Extensions – thus optional!!
  - Server Name Indication (SNI) (`tls.handshake.extensions_server_name`)
  - Next protocol: Application Layer Protocol Negotiation (ALPN) (`tls.handshake.extensions_alpn_str`)
  - Early Data (`tls.handshake.extension.type == 42`)

TLS version 1.3 Server Hello and Hello Retry Request

Information from Server Hello and Hello Retry Request:

- Selected version (`tls.handshake.extensions.supported_version`)
  - Thus TLS 1.3…
  - tls.record.version and tls.handshake.version no longer used
- Selected Cipher Suite (`tls.handshake.ciphersuite`)

TLS version 1.3 Application Data

When handshake complete and start first real Application Data

- Last handshake message Finished is encrypted (and small in size)
- Optionally followed by Session ticket(s)
  - Always send by server
  - When captured at client side, it may arrive after sending first request
- First real Application Data from client (most cases)

TLS version 1.3 Alert

- A TLS Alert (Close Notify) is always last record per direction and is small in size
- Followed by TCP-FIN or TCP-Reset

- Calculations:
  `overhead = tls.record.length – 2`
- Payload size in Application Data:
  `Payload size ≈ tls.record.length – overhead`

TLS version 1.3 Early Data (0-RTT)

- Client Hello contains Early Data extension
- Flow:
  - Client Hello
  - Change Cipher Spec, Application Data
  - Server Hello, etc…
  - End of Early Data
- Session ticket needed
- No forward secrecy (don't send sensitive data)
- HTTP 425 Too Early

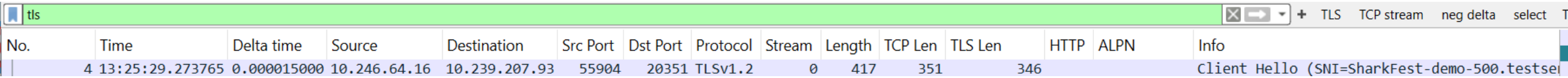
- Never used in back-end calls (APIs)

Summary

- Even when encrypted some possible causes can be ruled out
  - Helps to narrow search for root cause
- Or point to the possible cause
- Most timing issues can also be seen at TCP level
- If TLS Handshake successful, then not "network problem"
- Use 'own decrypted' traffic as reference

- At least: figure out where in the network path the traffic needs to be decrypted for further analysis

Profile I used



- Delta time: tcp.time_delta or udp.time_delta
- Stream: tcp.stream
- TCP Len: tcp.len
- TLS Len: tls.record.length
- HTTP len: len(http)
- ALPN: tls.handshake.extensions_alpn_str

Buttons:

- TLS: tls
- TCP Stream: tcp.stream == ${tcp.stream}

RFCs

- Overview: https://developer.mozilla.org/en-US/docs/Web/HTTP/Resources_and_specifications
- HTTP: https://datatracker.ietf.org/doc/html/rfc9110
- HTTP/2: https://datatracker.ietf.org/doc/html/rfc9113
- HTTP/3: https://datatracker.ietf.org/doc/html/rfc9114
- TLS 1.2: https://datatracker.ietf.org/doc/html/rfc5246
- TLS 1.3: https://datatracker.ietf.org/doc/html/rfc8446
- TCP: https://datatracker.ietf.org/doc/html/rfc9293 (First RFC 793 September 1981)
- IP: https://datatracker.ietf.org/doc/html/rfc791 (First RFC 760 January 1980)
- DNS: https://datatracker.ietf.org/doc/html/rfc1035 (First RFC 882 November 1983)

- First RFC: https://datatracker.ietf.org/doc/html/rfc1 7 April 1969
- History: https://www.computerhistory.org/internethistory/