Network Secure Protocols

learned:

- Secure network protocols protect data by encrypting it, helping prevent credentials and sensitive information from being stolen.

- When using Wireshark, you can identify whether a packet is using TLS or HTTP.

- Access to the encryption keys used in a protocol with TLS allows you to decrypt and view the contents of HTTPS traffic.

- Protocols like TLS can be added on top of unencrypted protocols like HTTP to make them secure and encrypt traffic.

- IMAP, POP3, and SMTP are used to send and receive emails. Secure versions, like IMAPS and POP3S, encrypt the email data to keep it safe during transfer.

TLS

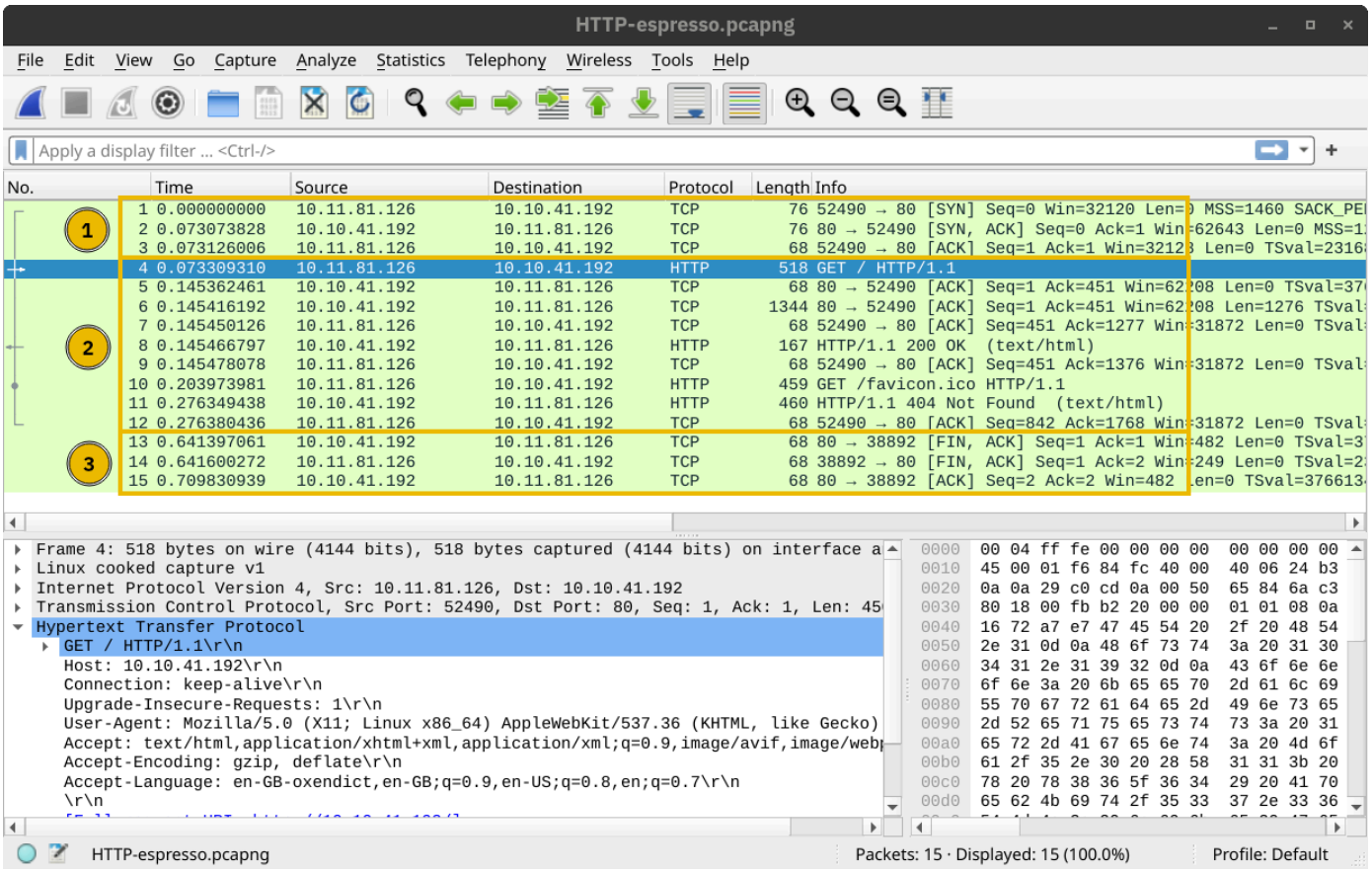
TLS is a cryptographic protocol operating at the OSI model's transport layer. It allows secure communication between a client and a server over an insecure network. By secure, we refer to confidentiality and integrity; TLS ensures that no one can read or modify the exchanged data.

HTTPS

After resolving the domain name to an IP address, the client will carry out the following two steps:

1. Establish a TCP three-way handshake with the target server
2. Communicate using the HTTP protocol; for example, issue HTTP requests, such as `GET / HTTP/1.1`

The two steps described above are shown in the window below. The three packets for the TCP handshake (marked with 1) precede the first HTTP packet with `GET` in it. The HTTP communication is marked with 2. The last three displayed packets are for TCP connection termination and are marked with 3.



HTTP Over TLS

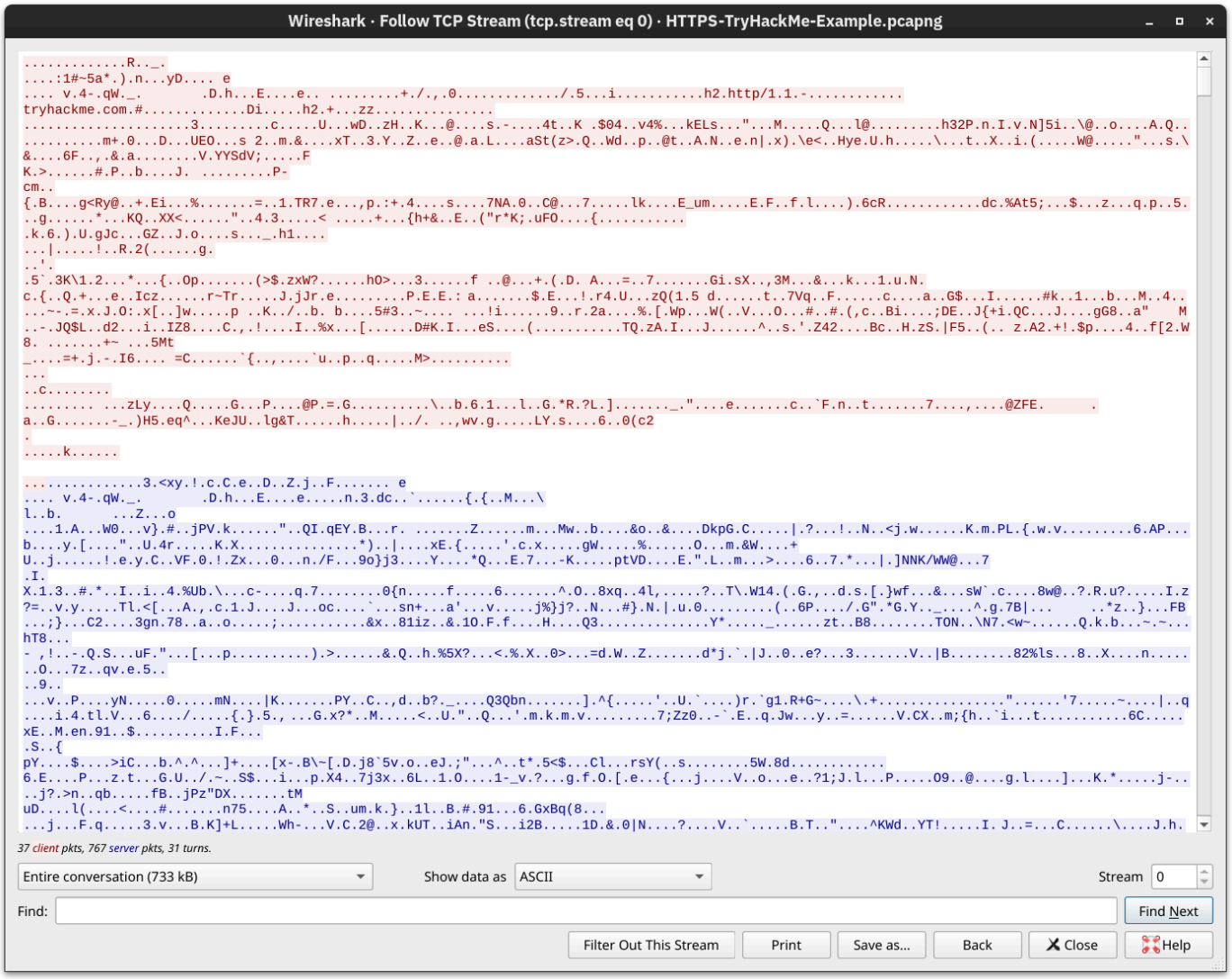
HTTPS stands for Hypertext Transfer Protocol Secure. It is basically HTTP over TLS. Consequently, requesting a page over HTTPS will require the following three steps (after resolving the domain name):

1. Establish a TCP three-way handshake with the target server
2. Establish a TLS session
3. Communicate using the HTTP protocol; for example, issue HTTP requests, such as `GET / HTTP/1.1`

The screenshot below shows that a TCP session is established in the first three packets, marked with `1`. Then, several packets are exchanged to negotiate the TLS protocol, marked with `2`. `1` and `2` are where the **TLS negotiation and establishment** take place.

Finally, HTTP application data is exchanged, marked with `3`. Looking at the Wireshark screenshot, we see that it says “Application Data” because there is no way to know if it is indeed HTTP or some other protocol sent over port 443.

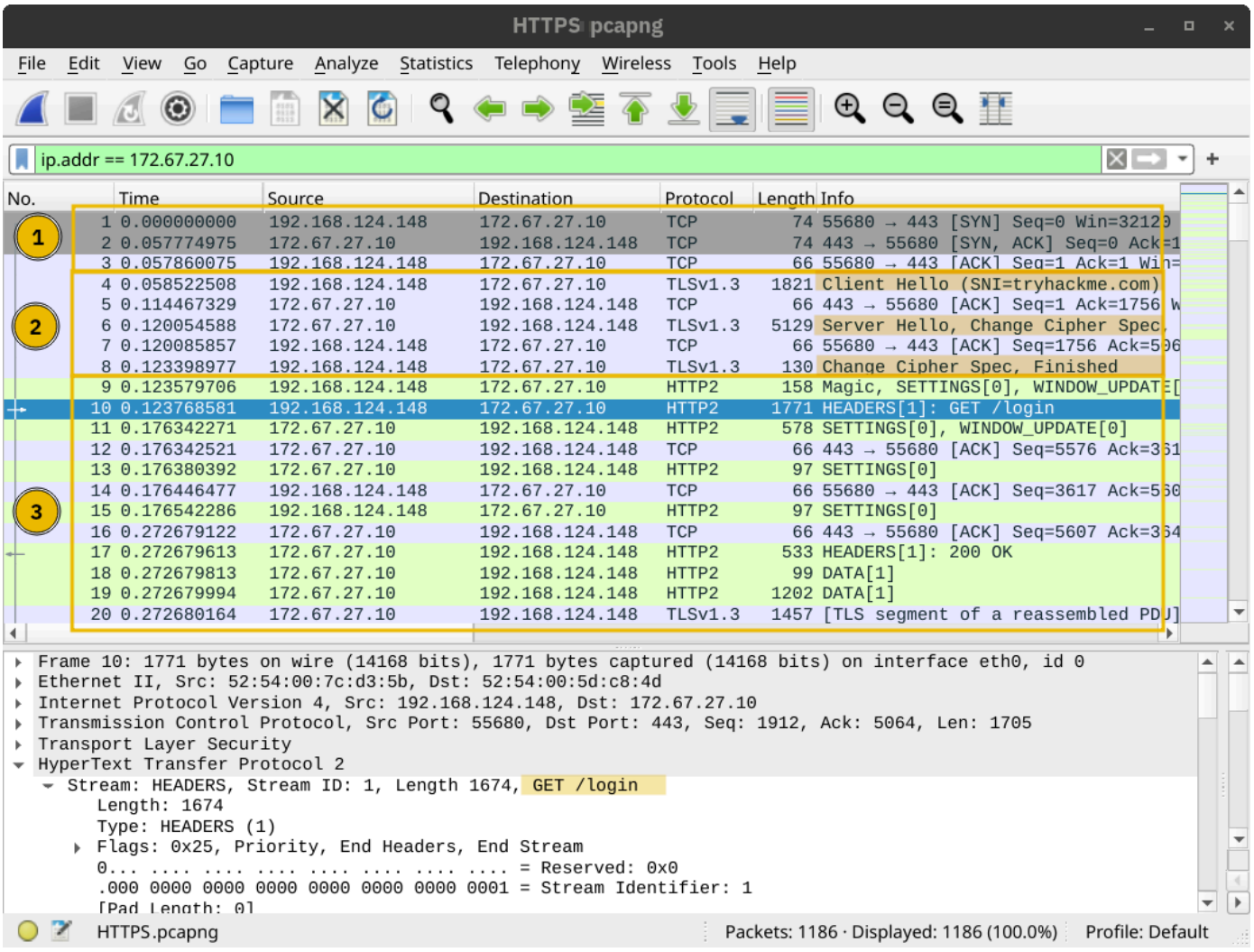
The exchanged traffic is encrypted; the red is sent by the client, and the blue is sent by the server. There is no way to know the contents without acquiring the encryption key.

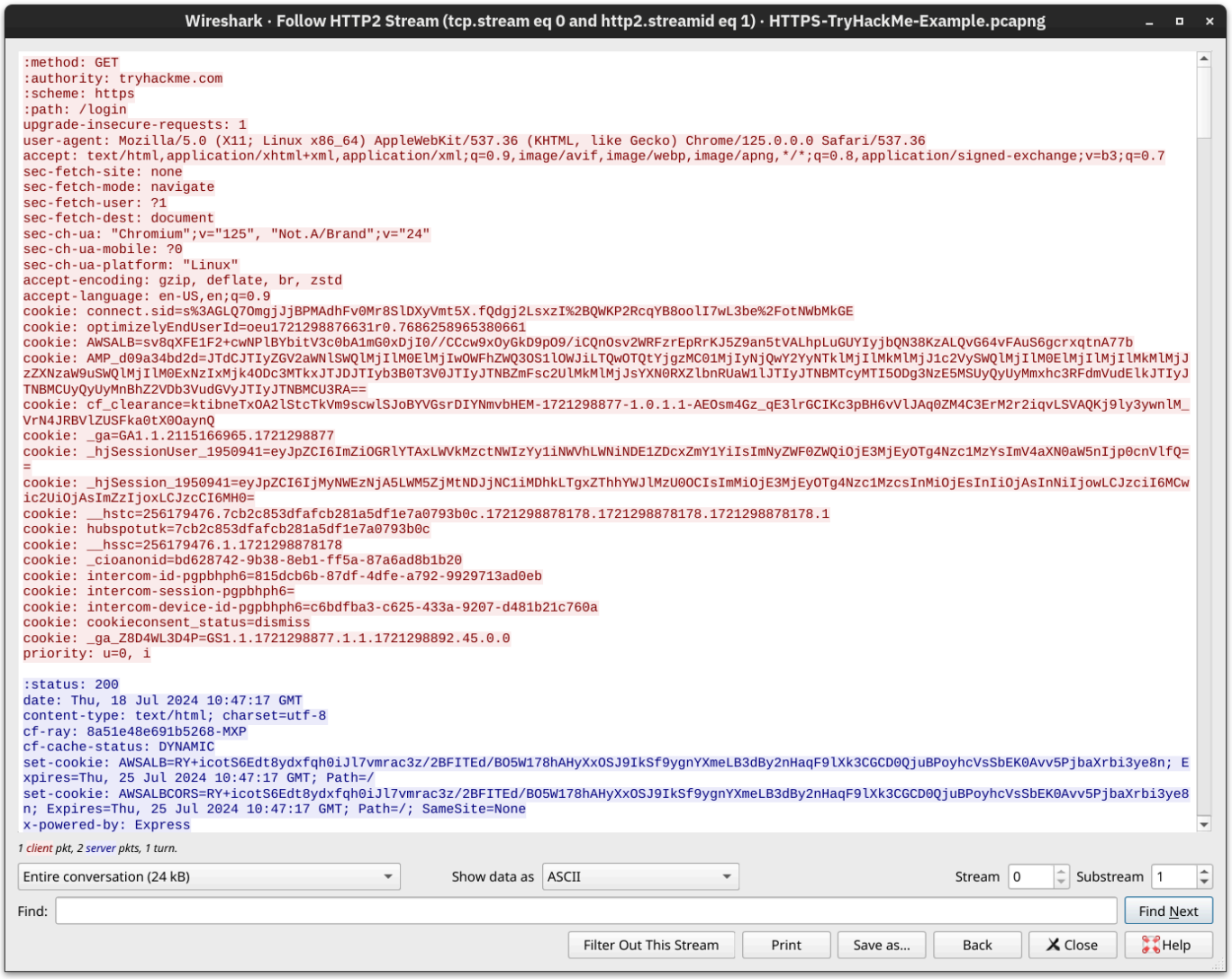


Getting the Encryption Key

Adding TLS to HTTP leads to all the packets being encrypted. We can no longer see the contents of the exchanged packets unless we get access to the private key.

We repeated the above screenshots after providing the decryption key to Wireshark. The TCP and TLS handshakes don't change; the main difference starts with the HTTP protocol marked 3. For instance, we can see when the client issues a `GET`.


HTTPS.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ip.addr == 172.67.27.10
No. Time Source Destination Protocol Length Info
1 0.000000000 192.168.124.148 172.67.27.10 TCP 74 55680 → 443 [SYN] Seq=0 Win=32120
2 0.057774975 172.67.27.10 192.168.124.148 TCP 74 443 → 55680 [SYN, ACK] Seq=0 Ack=1
3 0.057860075 192.168.124.148 172.67.27.10 TCP 66 55680 → 443 [ACK] Seq=1 Ack=1 Win=
4 0.058522508 192.168.124.148 172.67.27.10 TLSv1.3 1821 Client Hello (SNI=tryhackme.com)
5 0.114467329 172.67.27.10 192.168.124.148 TCP 66 443 → 55680 [ACK] Seq=1 Ack=1756 W
6 0.120054588 172.67.27.10 192.168.124.148 TLSv1.3 5129 Server Hello, Change Cipher Spec,
7 0.120085857 192.168.124.148 172.67.27.10 TCP 66 55680 → 443 [ACK] Seq=1756 Ack=506
8 0.123398977 192.168.124.148 172.67.27.10 TLSv1.3 130 Change Cipher Spec, Finished
9 0.123579706 192.168.124.148 172.67.27.10 HTTP2 158 Magic, SETTINGS[0], WINDOW_UPDATE[
10 0.123768581 192.168.124.148 172.67.27.10 HTTP2 1771 HEADERS[1]: GET /login
11 0.176342271 172.67.27.10 192.168.124.148 HTTP2 578 SETTINGS[0], WINDOW_UPDATE[0]
12 0.176342521 172.67.27.10 192.168.124.148 TCP 66 443 → 55680 [ACK] Seq=5576 Ack=361
13 0.176380392 172.67.27.10 192.168.124.148 HTTP2 97 SETTINGS[0]
14 0.176446477 192.168.124.148 172.67.27.10 TCP 66 55680 → 443 [ACK] Seq=3617 Ack=560
15 0.176542286 192.168.124.148 172.67.27.10 HTTP2 97 SETTINGS[0]
16 0.272679122 172.67.27.10 192.168.124.148 TCP 66 443 → 55680 [ACK] Seq=5607 Ack=364
17 0.272679613 172.67.27.10 192.168.124.148 HTTP2 533 HEADERS[1]: 200 OK
18 0.272679813 172.67.27.10 192.168.124.148 HTTP2 99 DATA[1]
19 0.272679994 172.67.27.10 192.168.124.148 HTTP2 1202 DATA[1]
20 0.272680164 172.67.27.10 192.168.124.148 TLSv1.3 1457 [TLS segment of a reassembled PDU]
1
2
3
Frame 10: 1771 bytes on wire (14168 bits), 1771 bytes captured (14168 bits) on interface eth0, id 0
Ethernet II, Src: 52:54:00:7c:d3:5b, Dst: 52:54:00:5d:c8:4d
Internet Protocol Version 4, Src: 192.168.124.148, Dst: 172.67.27.10
Transmission Control Protocol, Src Port: 55680, Dst Port: 443, Seq: 1912, Ack: 5064, Len: 1705
Transport Layer Security
HyperText Transfer Protocol 2
Stream: HEADERS, Stream ID: 1, Length 1674, GET /login
Length: 1674
Type: HEADERS (1)
Flags: 0x25, Priority, End Headers, End Stream
0... .... = Reserved: 0x0
.000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
[Pad Length: 0]
HTTPS.pcapng
Packets: 1186 · Displayed: 1186 (100.0%)
Profile: Default



SMTPS,POP3S, and IMAPS

Adding TLS to SMTP, POP3, and IMAP is no different than adding TLS to HTTP. Similar to how HTTP gets an appended S for *Secure* and becomes HTTPS, SMTP, POP3, and IMAP become SMTPS, POP3S, and IMAPS.

The insecure versions use the default TCP port numbers shown in the table below:

Protocol	Default Port Number
HTTP	80
SMTP	25
POP3	110
IMAP	143

The secure versions, i.e., over TLS, use the following TCP port numbers by default:

Protocol	Default Port Number
HTTPS	443
SMTPS	465 and 587
POP3S	995
IMAPS	993

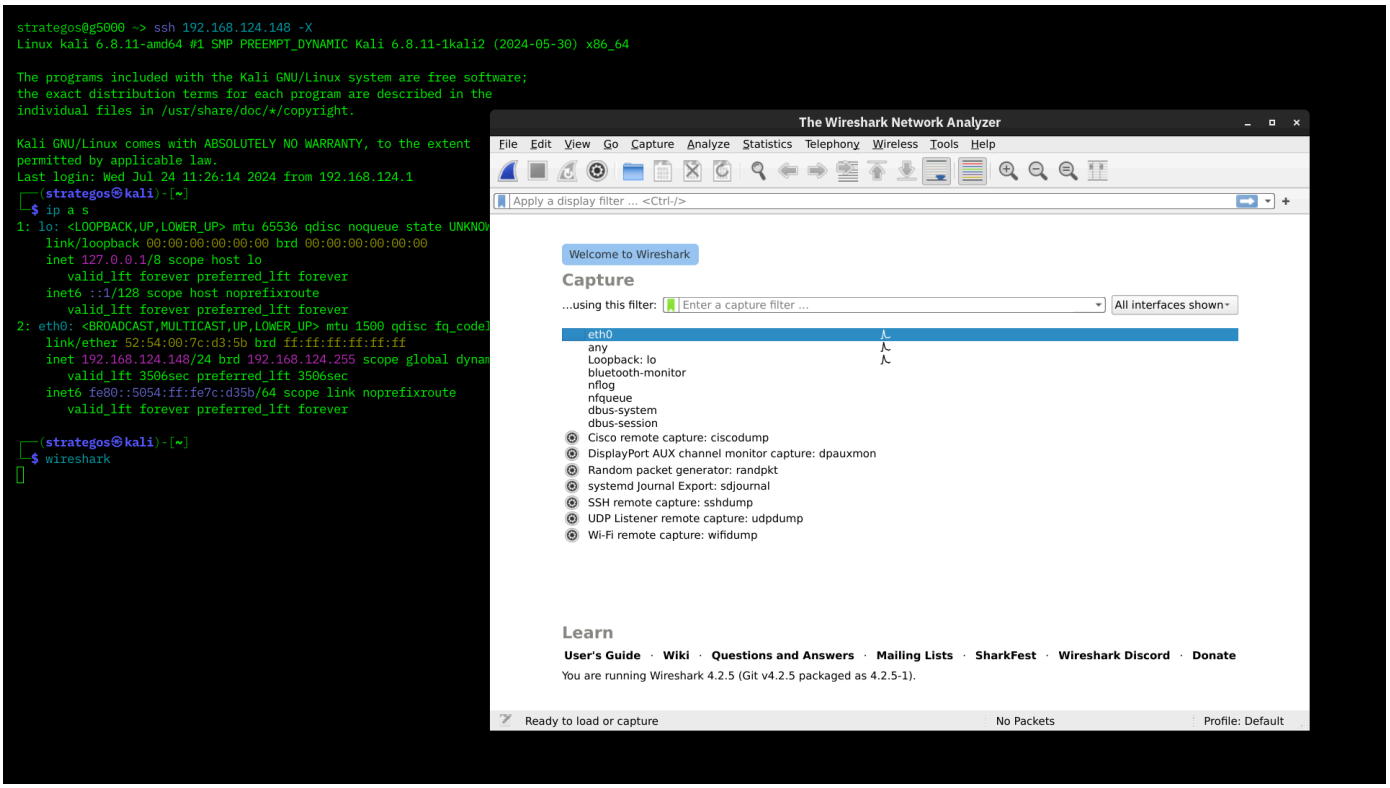
SSH

OpenSSH offers several benefits. We will list a few key points:

- **Secure authentication**: Besides password-based authentication, SSH supports public key and two-factor authentication.
- **Confidentiality**: OpenSSH provides end-to-end encryption, protecting against eavesdropping. Furthermore, it notifies you of new server keys to protect against man-in-the-middle attacks.
- **Integrity**: In addition to protecting the confidentiality of the exchanged data, cryptography also protects the integrity of the traffic.
- **Tunneling**: SSH can create a secure “tunnel” to route other protocols through SSH. This setup leads to a VPN-like connection.
- **X11 Forwarding**: If you connect to a Unix-like system with a graphical user interface, SSH allows you to use the graphical application over the network.

You would issue the command `ssh username@hostname` to connect to an SSH server. If the username is the same as your logged-in username, you only need `ssh hostname`. Then, you will be asked for a password; however, if public-key authentication is used, you will be logged in immediately.

The screenshot below shows an example of running Wireshark on a remote Kali Linux system. The argument `-X` is required to support running graphical interfaces, for example, `ssh 192.168.124.148 -X`. (The local system needs to have a suitable graphical system installed.)



While the TELNET server listens on port 23, the SSH server listens on port 22.

SFTP and FTPS

SFTP stands for SSH File Transfer Protocol and allows secure file transfer. It is part of the SSH protocol suite and shares the same port number, 22. If enabled in the OpenSSH server configuration, you can connect using a command such as `sftp username@hostname`. Once logged in, you can issue commands such as `get filename` and `put filename` to download and upload files.

Like HTTPS, SMTPS, POP3S, IMAPS, and other protocols that rely on TLS for security, FTPS requires a proper TLS certificate to run securely.