جامعة عفت

EFFAT UNIVERSITY

Computer science, College of Engineering

Effat University

Saudi Arabia, Jeddah

Traffic Capture and Security Analysis Using Wireshark

CS 3093 - CRYPTO. &
NETWORK SECURITY

Final Project - Fall 2025

Instructor: **Dr. Sohail Khan**

Abeer Hussain	S22107714
Sarah Eid	S22107757
Nancy Elhaddad	S20206871

Contents

Introduction

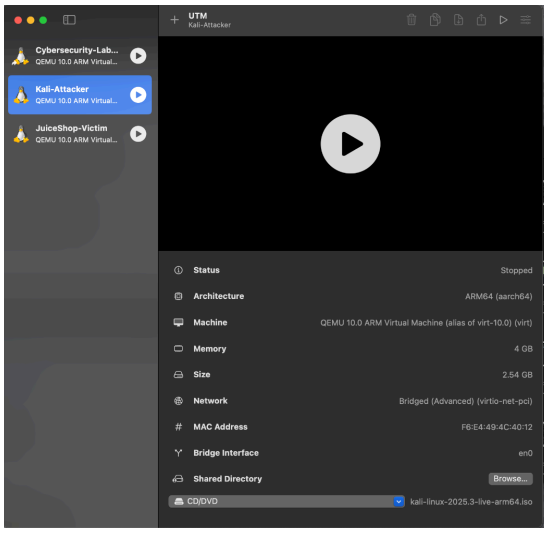
This project focuses on setting up a safe virtual lab environment, capturing network traffic using Wireshark, and analyzing TCP/IP, HTTP, and web application behavior in a vulnerable web application (DVWA or OWASP Juice Shop). The goal is to identify common security issues such as plaintext credentials, exposed session tokens, and lack of encryption.

1 Environment Setup

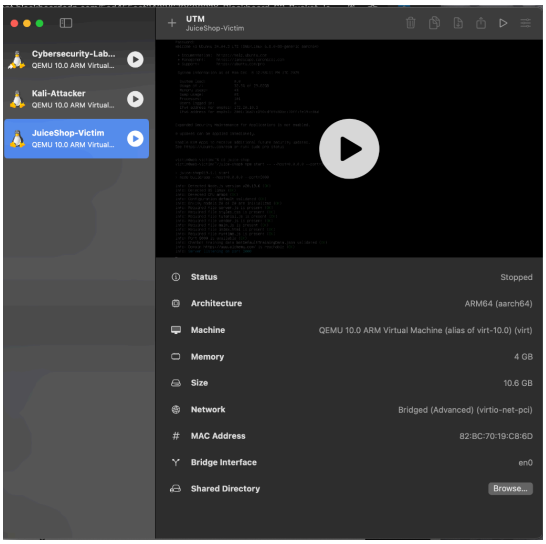
- Virtual Machines Used

VM Name	OS	Purpose	IP Address
Kali-Attacker	Kali Linux	Packet capture, analysis, performing traffic interaction	172.20.10.3
JuiceShop-Victim	Ubuntu 24.04 running OWASP Juice Shop	Target vulnerable web app	172.20.10.4

Kali-Attacker



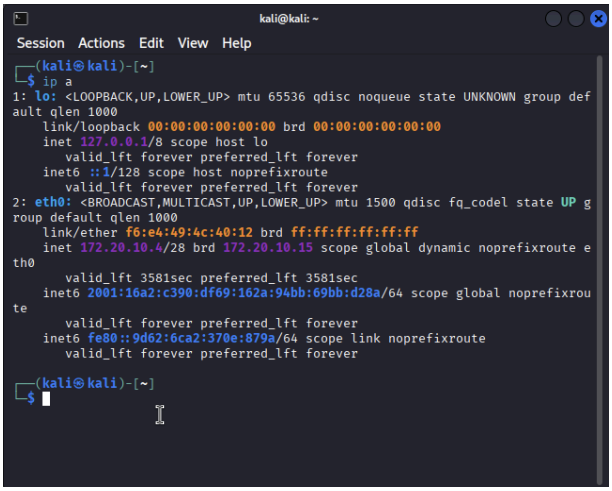
JuiceShop-Victim



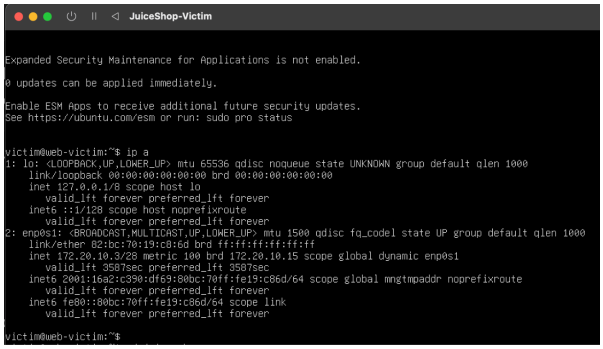
- IP Addresses

Machine / VM Name	Role	IP Address	MAC Address	Network Mode
Kali-Attacker	Attacker	172.20.10.X	f6:e4:49:4c:40:12	Bridged (Advanced)
JuiceShop-Victim	Target	172.20.10.Y	82:bc:70:19:c8:60	Bridged (Advanced)

Kali-Attacker IP Address



JuiceShop-Victim IP Address



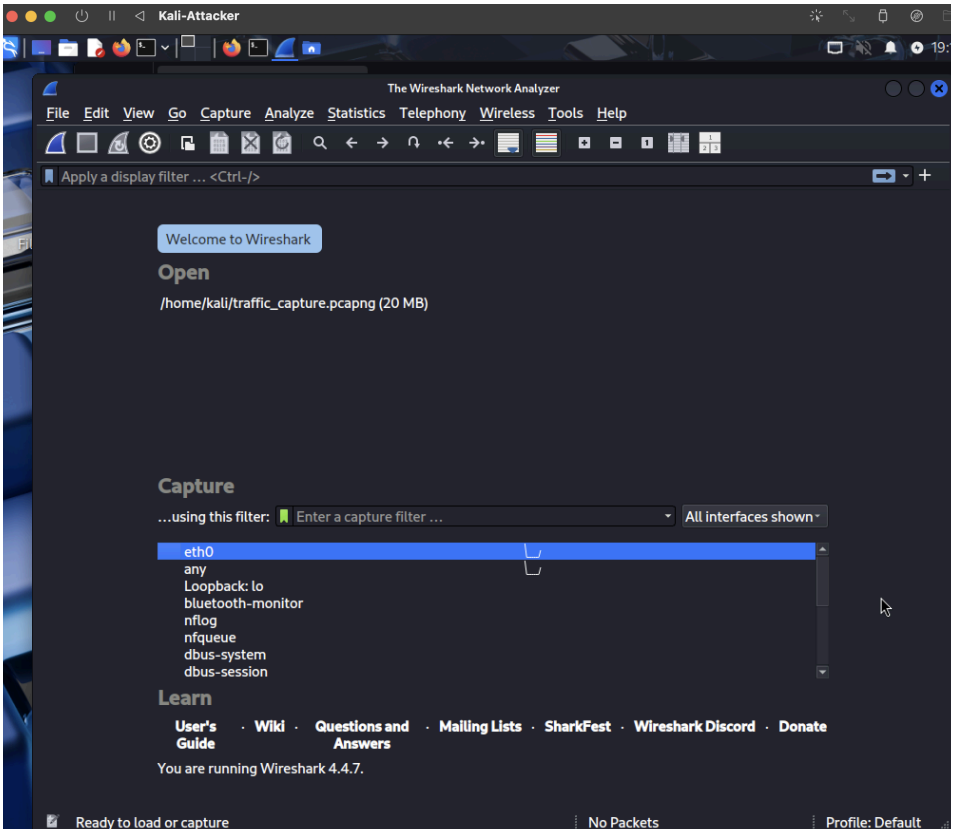
- Reason for selected network configuration

In this project, Bridged networking is utilized for both the Kali attacker VM and the Juice Shop victim VM, allowing each virtual machine to obtain its own IP address on the same LAN as the host. This setup simulates a realistic attack environment, facilitating direct interaction between Kali and the Juice Shop server. It also enhances data capture within Wireshark, as no NAT translation is needed, enabling effective monitoring of TCP Connection Handshakes, HTTP Requests, Cookies, Session Tokens, and Login Information. Overall, this configuration provides a stable and realistic platform for network security analysis.

2 Capture Summary

- Interface Captured:

  The traffic was captured on the interface eth0, which is the active network interface used by the Kali attacker VM.



- Duration of Capture

  The capture was running for approximately 3-4 minutes, covering the entire interaction with OWASP Juice Shop, including login, browsing, form submissions, and API requests.

- Filters Used During Analysis:

The following Wireshark display filters were used to analyze specific packet types:

Purpose	Filter
All HTTP traffic	`http`
GET requests	`http.request.method == "GET"`

POST requests	`http.request.method == "POST"`
Cookie/session token exposure	`http.cookie`
TCP handshake packets	tcp.flags.syn == 1 || tcp.flags.ack == 1

- pcap File:
  The pcap file containing the full capture (traffic_capture.pcapng) is attached with the report.

3 Packet Analysis Results

- TCP Handshake Analysis

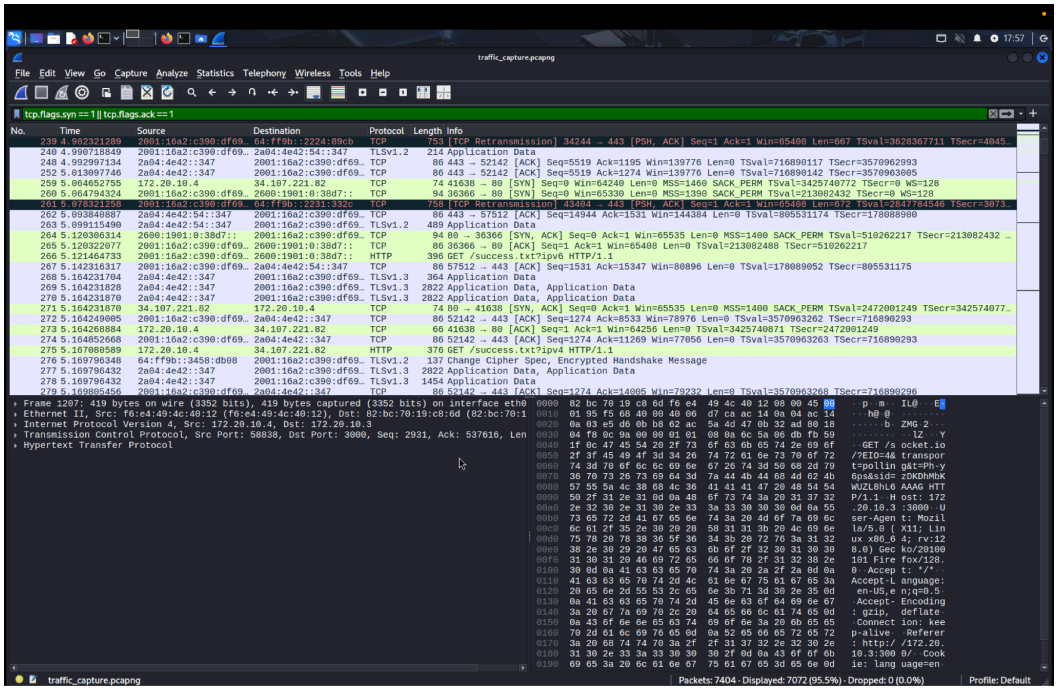
The TCP 3-way handshake was captured in packets 259 (SYN), 260 (SYN-ACK), and 266 (ACK).
It establishes a reliable connection before any HTTP traffic can be sent.

1. SYN (Packet 259):
   The client (172.20.10.4) sends a SYN to start the connection.

2. SYN-ACK (Packet 260):
   The server responds with a SYN-ACK.
   (The ACK flag is present but truncated in the display.)

3. ACK (Packet 266):
   The client sends the final ACK, completing the handshake.

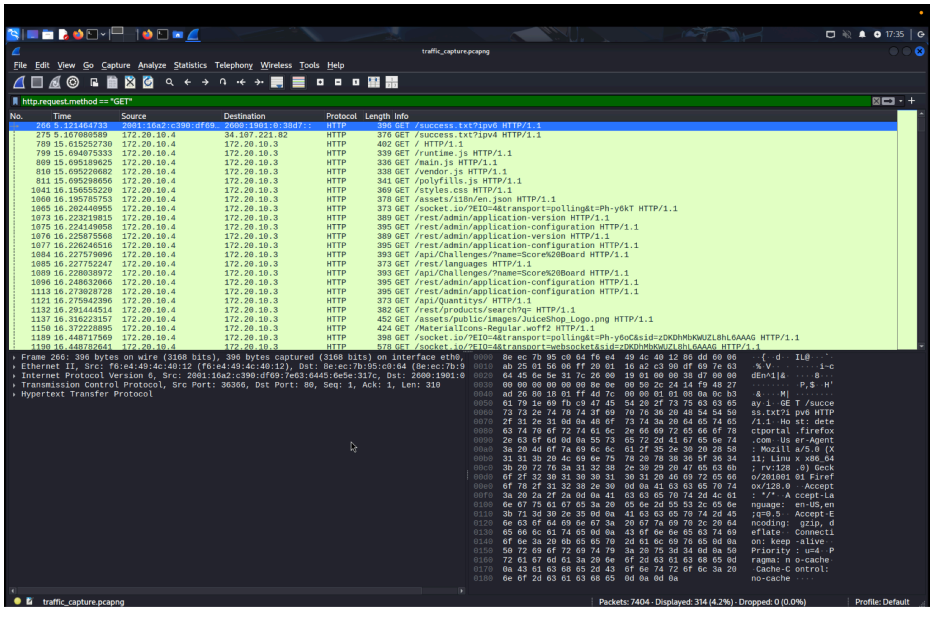
This process ensures both sides are synchronized and ready for data transfer.



- HTTP request and response findings

During the capture, several HTTP requests and responses were recorded. Using the filters http, http.request.method=="GET", and http.request.method=="POST", the following patterns were observed:

1. GET Requests



The GET traffic includes:

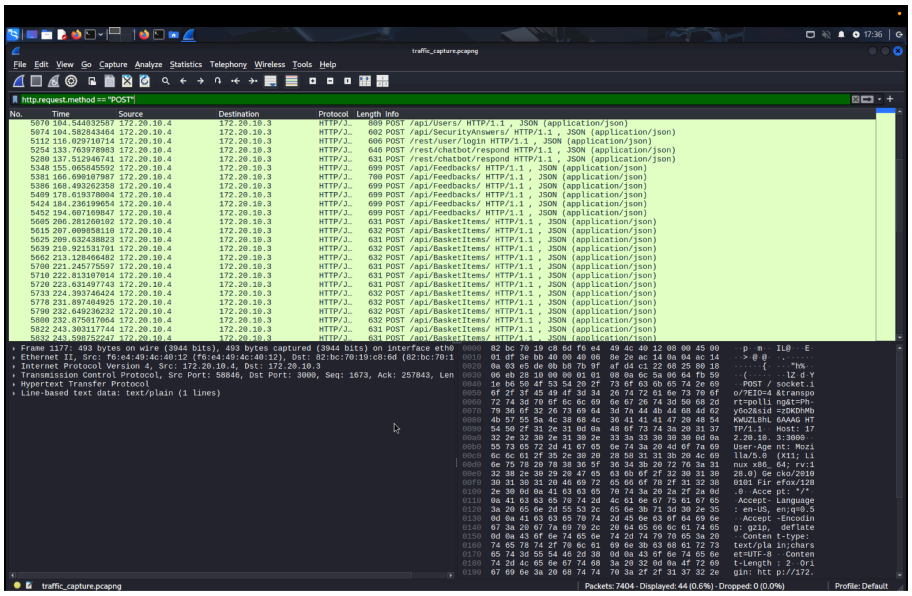
- /success.txt?P1vP6
- /runtime.js
- /main.js
- /vendor.js
- /styles.css
- /assets/...
- /rest/products/search
- /socket.io/?EIO=4&transport=polling

These GET requests show the browser loading page content, scripts, stylesheets, images, and product data from the Juice Shop API. They all return HTTP/1.1 200 OK, confirming successful responses from the server.

2. POST Requests (Shown in Your Screenshots)

Your POST traffic includes:

- POST /api/Users
- POST /api/SecurityAnswers
- POST /rest/user/login
- POST /api/Feedbacks
- POST /api/BasketItems
- POST /socket.io/?EIO=4...

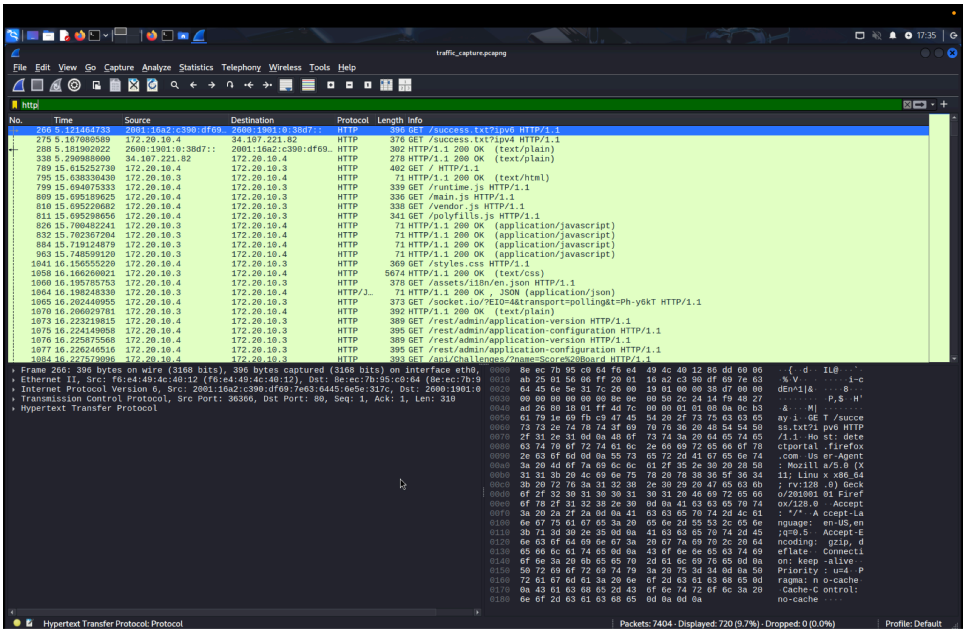


These contain JSON payloads used for login attempts, creating accounts, sending feedback, adding basket items, and interacting with the chat engine. This confirms that form submissions and login actions were captured successfully.

3. HTTP Responses

Each request has a visible response in your screenshots, including:

- 200 OK
- application/json payloads
- text/html document responses



These responses confirm that the application sent back full page data, JSON responses, and API

results in plaintext HTTP (no HTTPS).

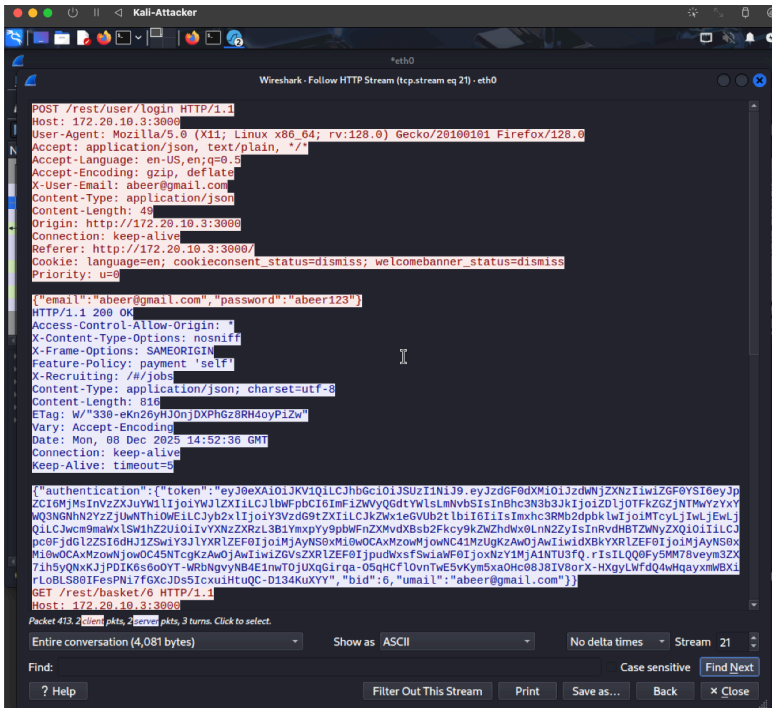
- Credentials exposure (Scenario A)

We captured the login request while Wireshark was running. We located the POST request: POST /rest/user/login, which carries the username and password.

The HTTP Stream shows the credentials in plaintext:

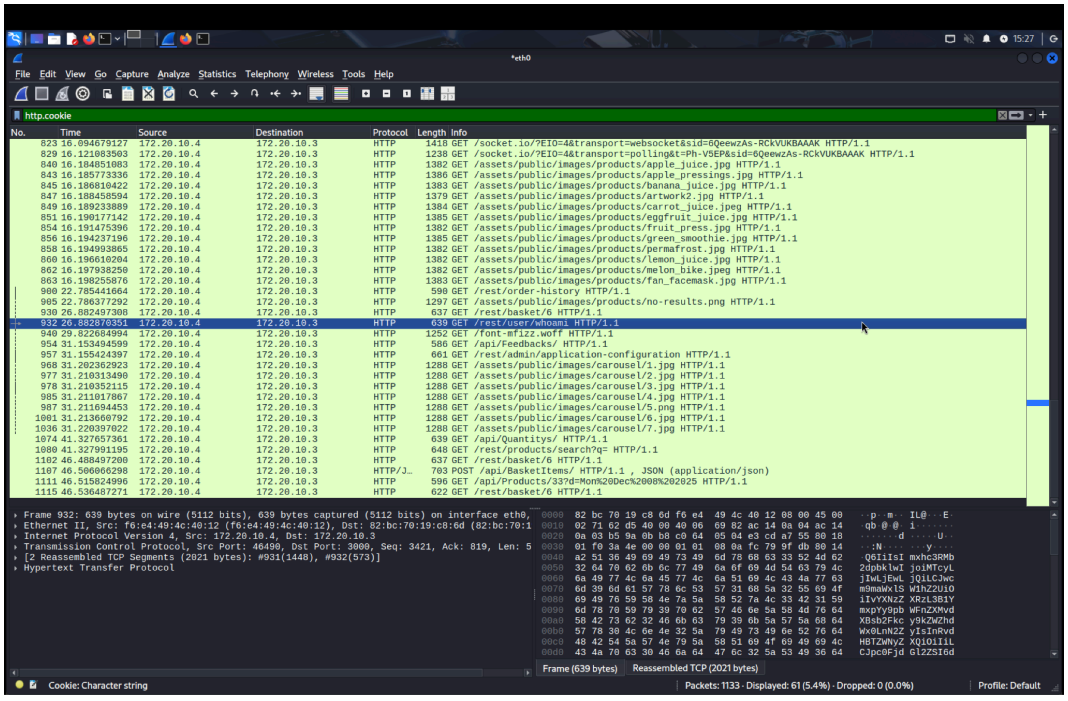
"email":"abeer@gmail.com",
"password":"abeer123"

The login form sends the username and password over HTTP without any encryption. Because the traffic is not protected by TLS/HTTPS, anyone on the same network can capture and read the credentials. This makes the login extremely insecure and vulnerable to packet sniffing or man-in-the-middle attacks. Secure websites should always use HTTPS to protect sensitive information.



- Session token exposure (Scenario B)

After logging in, we filtered the traffic using http.cookie to see all packets that carried session cookies.



The capture reveals that the client at 172.20.10.3 sends a session cookie back to the server at 172.20.10.4 with every authenticated request. It appears in plaintext within HTTP GET and POST packets.

That's dangerous, because if an attacker intercepts that cookie, they could reuse the token to impersonate the user (session hijacking) without having to know the password. Because HTTP isn't encrypted, an attacker can sniff traffic on the same network and steal the token.

4 Identified Weaknesses

Major Security Weaknesses

- Credentials in plaintext during login
- Session tokens exposed in every request
- No TLS encryption (HTTP used)
- Cookies missing security flags:
    - Secure
    - HttpOnly
    - SameSite
- API endpoints return excessive data, increasing exposure risk
- Predictable session patterns

5 Recommendations

To secure the application:

A. Encrypt Communications

- Enforce HTTPS/TLS on all pages
- Redirect all HTTP → HTTPS

B. Secure Cookies

- Add:

    - Secure
    - HttpOnly
    - SameSite=Strict

C. Improve Session Management

- Regenerate session tokens after login
- Shorten session lifetime
- Invalidate token on logout

D. Harden the Web Application

- Disable debug modes
- Reduce overly verbose API responses
- Sanitize all form submissions

6 Reflection

This project has taught us how a network traffic analysis using Wireshark can be done. We have learned how key protocols such as ARP, DNS, TCP, and HTTP work. We were also introduced to how a network traffic inspection for security problems such as cleartext credentials and missing encryption might look in a practical example.

Traffic analysis forms a critical part of cybersecurity because it exposes loopholes that cannot be observed when using the application in a conventional manner. By looking at the packets themselves, we are able to monitor vulnerable usage and determine whether our critical data is properly safeguarded. Analysis of data flow within a network forms a vital skill for increasing security.