Task 5: Capture and Analyze Network Traffic Using Wireshark

Introduction

As part of my cybersecurity internship, I was assigned a practical task to capture and analyze network traffic using Wireshark. Although I had read about packet sniffing and protocols before, this task gave me the chance to actually see the network traffic in action — and that changed my understanding completely. I could finally connect theory to practice, which made learning both exciting and insightful.

Step-by-Step Process

1. Installing Wireshark

I began by downloading and installing Wireshark, a free and widely used network protocol analyzer. The installation process was smooth, and I also made sure to install WinPcap/Npcap, which is required for capturing packets.

2. Starting the Capture

After launching Wireshark, I selected my active network interface (Wi-Fi in my case) and clicked on the Start Capture button.

To generate traffic:

I opened a few websites in my browser like google.com, openai.com, and

wikipedia.org.

I also ran ping commands in my terminal to observe ICMP traffic.

This helped me simulate a real-life scenario where multiple protocols are actively being used.

3. Stopping and Filtering the Capture
After letting the capture run for about 1 minute, I stopped the session to begin analyzing.

To filter the data, I used Wireshark's powerful Display Filters:

http to isolate web traffic,

dns to view domain name lookups,

tcp to focus on reliable communication protocols,

and I even tried icmp to view ping-related packets.

Protocols Identified and Observations
1. DNS (Domain Name System)
I observed DNS queries my system made to resolve website addresses.

Each DNS request had a corresponding response with the resolved IP.

Example: A record for www.google.com pointing to an IP address.

2. TCP (Transmission Control Protocol)
I saw the three-way handshake (SYN, SYN-ACK, ACK) clearly in the packet list.

This helped me understand how reliable connections are established.

Also, I noticed packet retransmissions which might indicate latency or packet loss.

3. HTTP (HyperText Transfer Protocol)
GET requests for HTML pages, CSS files, and images were visible.

I could see headers like User-Agent, Host, and Accept-Language, which was fascinating.

Some of the requests had HTTP response codes like 200 OK or 301 Redirect.

ICMP (Internet Control Message Protocol)
From the ping command I ran, I could see Echo Request and Echo Reply packets.

It was interesting to see this layer of communication that helps diagnose network availability.

What I Learned
This task helped me get a real sense of how protocols work together behind the scenes:

Every time I open a website, a lot more is happening than I realized — DNS lookups, TCP handshakes, HTTP requests, etc.

I learned to filter large volumes of packet data effectively using Wireshark's syntax.

I also began to appreciate how important tools like Wireshark are for network troubleshooting, security analysis, and performance monitoring.

Reflections

Before this task, networking protocols were mostly just theoretical concepts to me. But after capturing live packets, I now see networks as living systems constantly talking, requesting, and responding — all in real time.

I also realized how exposed some information can be if not encrypted, which underlined the importance of HTTPS and encryption in general.