Wireshark Network Packet Capture Demonstration

Owen Lindsey

Professor Sluiter, Shad

CST-407

Grand Canyon University

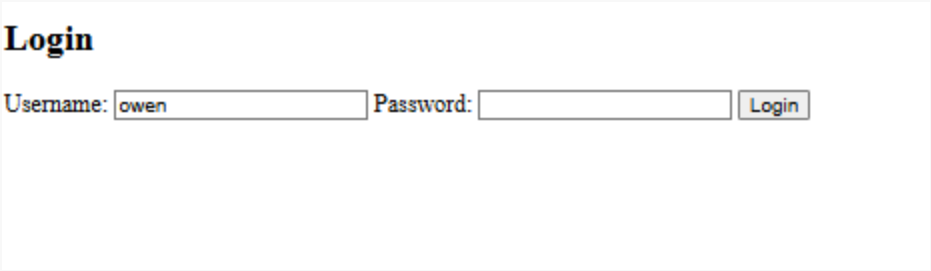
Overview

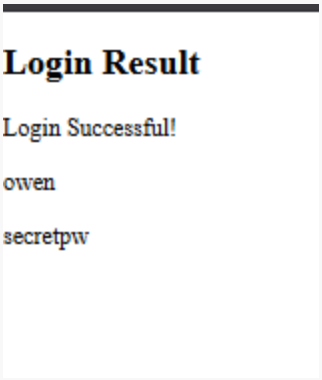
Application security is both an internal and external challenge. The way in which an application sends data across a network will determine how secure the data remains. This lesson will demonstrate that unencrypted network requests are vulnerable to packet sniffers, as seen in Figure 1.

Wireshark is a popular hacking tool as well as network engineer's diagnostic tool for viewing communication over a computer network. For engineers, seeing error messages, response times and traffic routes can be valuable to improving the performance of network devices. For hackers, network traffic may reveal sensitive information.

1. Tools Needed: (a) Java IDE (b) Wireshark application.
2. Using this tutorial, create a very simple application that has a login form and form processor. The Java application will run a web server that operates a login screen and results screen.
3. We will capture the network traffic using Wireshark and steal credentials from an unencrypted login session.
4. Finally, we will convert the unencrypted http traffic to an encrypted https protocol and compare the traffic captured by Wireshark.

Output of Spring Boot Project





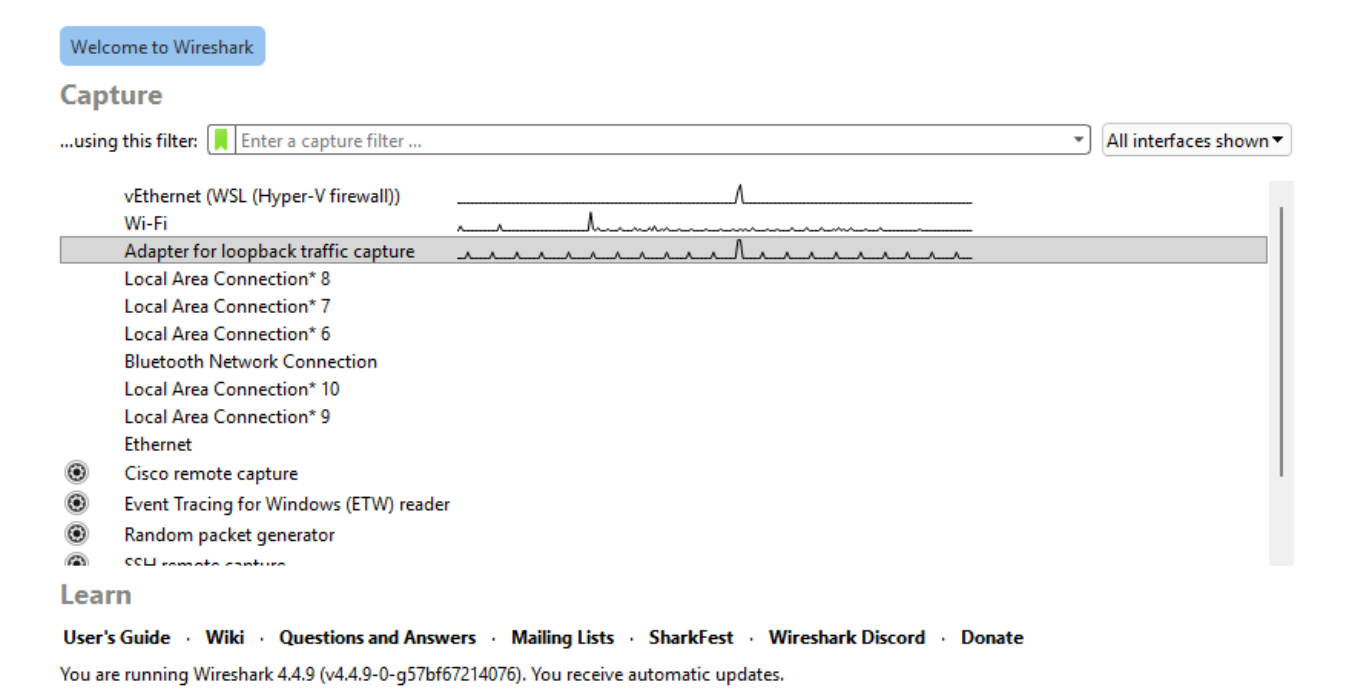
Wireshark Instructions

Download and install Wireshark.

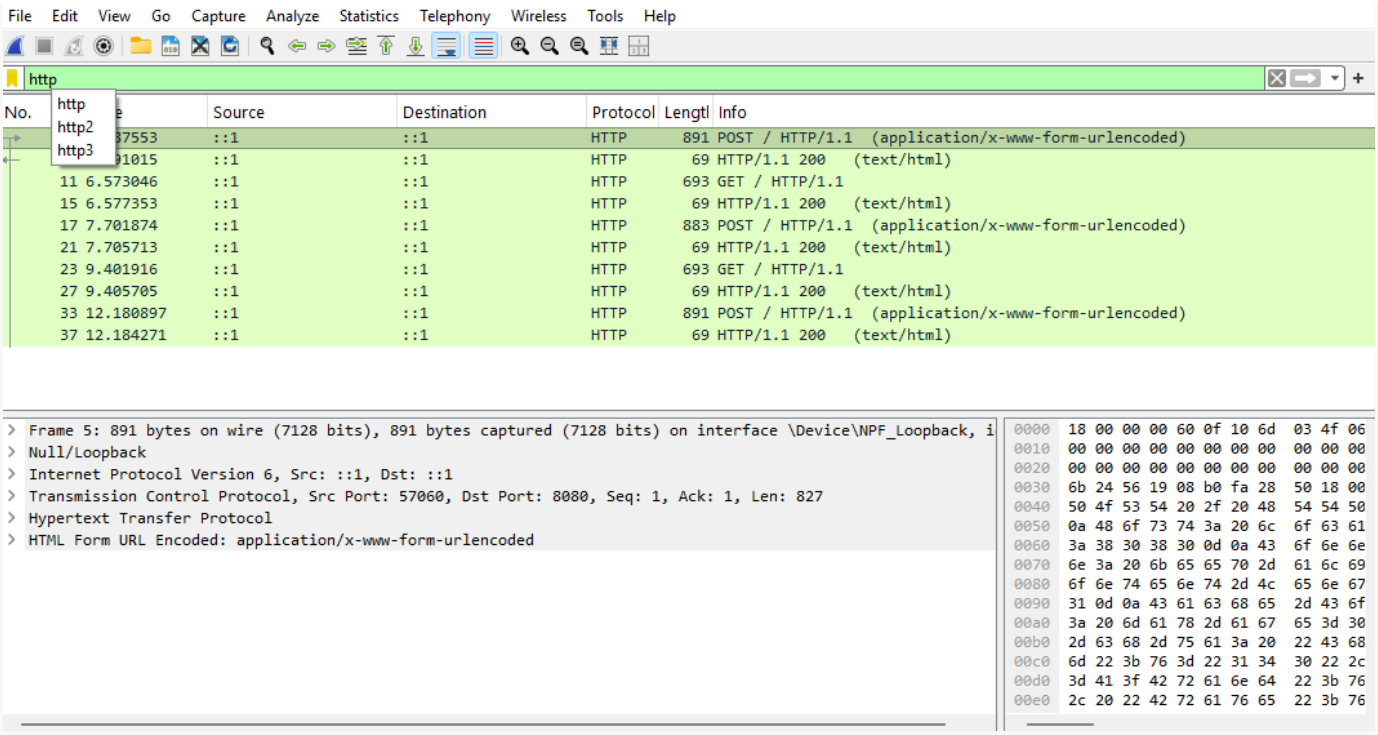
The application might ask you to install ChmodBPF and relaunch the app.

Windows will need a module called npcap. Npcap is a packet capture library for Windows

that includes a "loopback adapter" that allows you to capture loopback traffic with



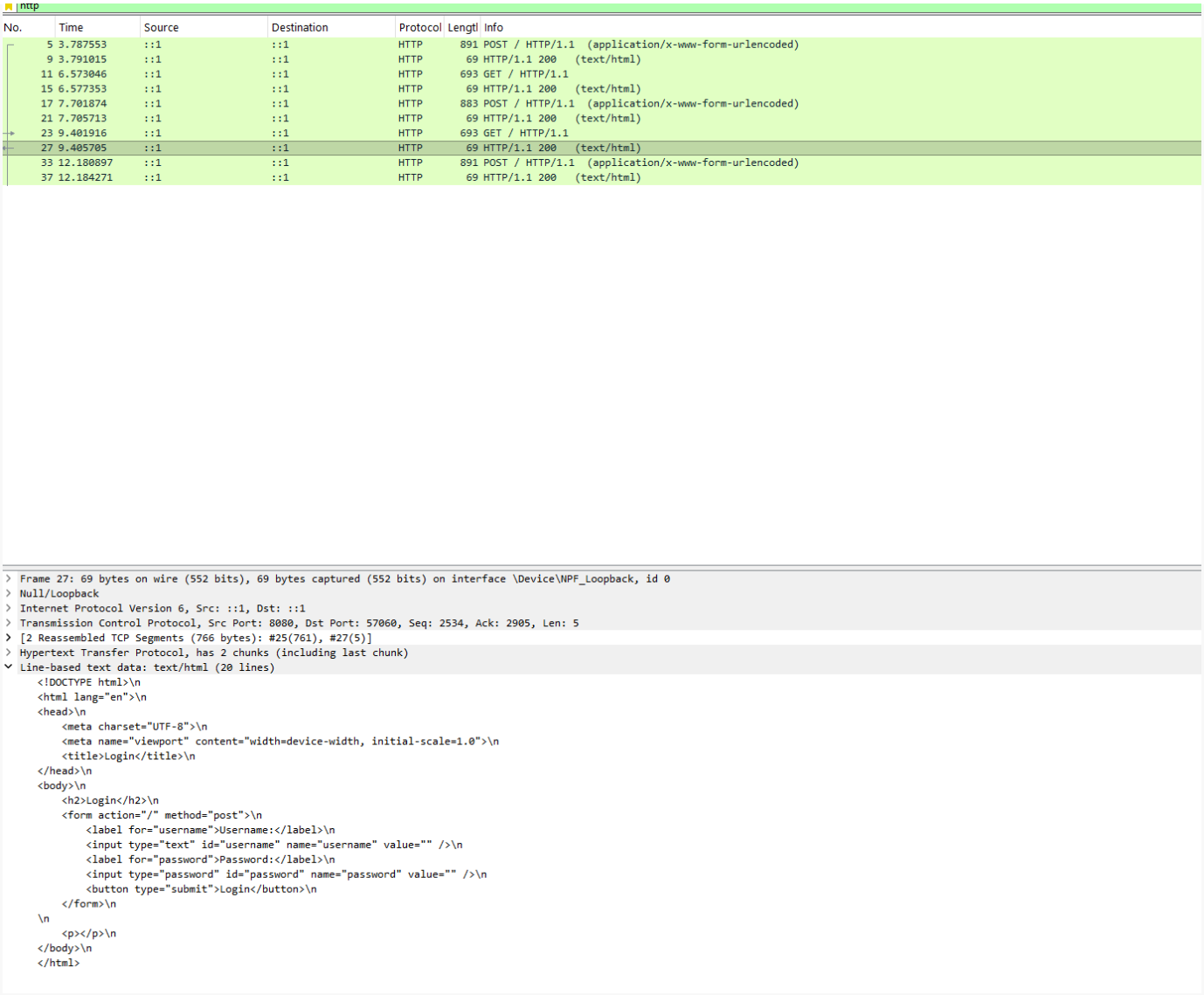
You should see a long list of text. Each line in the report represents a packet of communications between your computer and the localhost web server.



Apply a filter to show only "http" packets.

Select one of the http packets with a 200 result and text/html content.

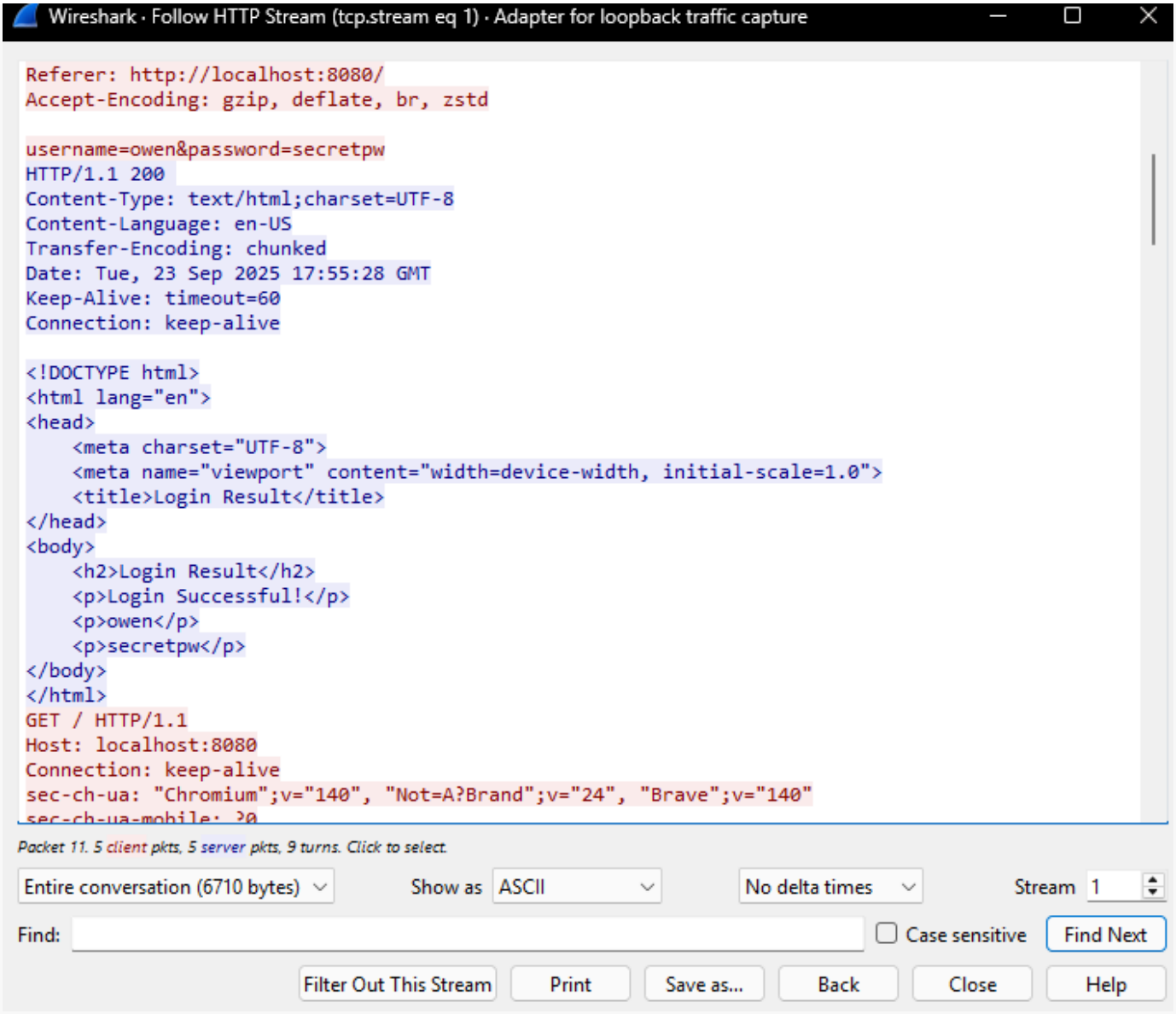
Expand the "Line-based text data" to see the contents of this request



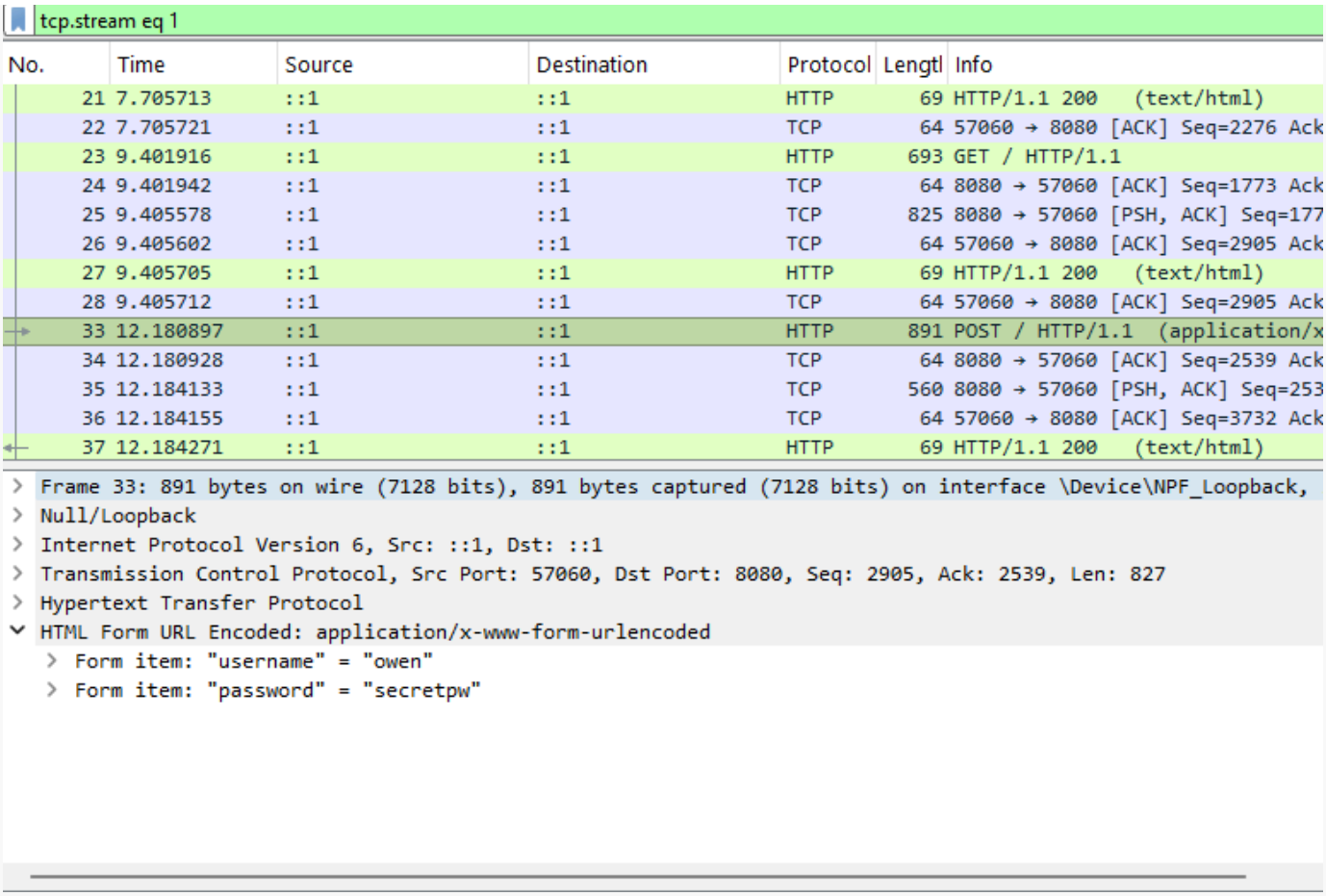
No.	Time	Source	Destination	Protocol	Length	Info
5	3.787553	::1	::1	HTTP	891	POST / HTTP/1.1 (application/x-www-form-urlencoded)
9	3.791015	::1	::1	HTTP	69	HTTP/1.1 200 (text/html)
11	6.573046	::1	::1	HTTP	693	GET / HTTP/1.1
15	6.577353	::1	::1	HTTP	69	HTTP/1.1 200 (text/html)
17	7.701874	::1	::1	HTTP	883	POST / HTTP/1.1 (application/x-www-form-urlencoded)
21	7.705713	::1	::1	HTTP	69	HTTP/1.1 200 (text/html)
23	9.401916	::1	::1	HTTP	693	GET / HTTP/1.1
27	9.405705	::1	::1	HTTP	69	HTTP/1.1 200 (text/html)
33	12.180897	::1	::1	HTTP	891	POST / HTTP/1.1 (application/x-www-form-urlencoded)
37	12.184271	::1	::1	HTTP	69	HTTP/1.1 200 (text/html)

```
> Frame 27: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 8080, Dst Port: 57060, Seq: 2534, Ack: 2905, Len: 5
> [2 Reassembled TCP Segments (766 bytes): #25(761), #27(5)]
> Hypertext Transfer Protocol, has 2 chunks (including last chunk)
v Line-based text data: text/html (20 lines)
    <!DOCTYPE html>\n
    <html lang="en">\n
    <head>\n
        <meta charset="UTF-8">\n
        <meta name="viewport" content="width=device-width, initial-scale=1.0">\n
        <title>Login</title>\n
    </head>\n
    <body>\n
        <h2>Login</h2>\n
        <form action="/" method="post">\n
            <label for="username">Username:</label>\n
            <input type="text" id="username" name="username" value="" />\n
            <label for="password">Password:</label>\n
            <input type="password" id="password" name="password" value="" />\n
            <button type="submit">Login</button>\n
        </form>\n
    \n
        <p></p>\n
    </body>\n
    </html>
```

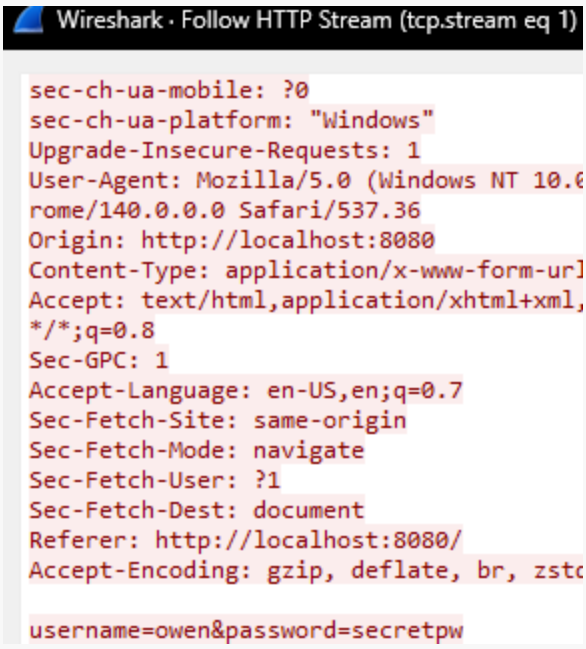
You can retrieve a more readable version of the packet by right-clicking the line in the log >Follow > HTTP Stream



Select the POST request and expand the contents. You should be able to see the form items for username and password by opening the items in the bottom window



You can also display the entire packet in a more readable format by right-clicking > Follow > HTTP stream



About Internet Protocol Exchanges

Every time a service needs to communicate with a client over digital networks, a protocol must be designed to make the transaction work smoothly. In human life, we have developed protocols of communication that help make the process standardized.

“Hello?”
“Hi, this is ___ from the ___. How are you today?”
“I’m fine, thank you. What can I do for you?”

(Payload of the conversation)

“OK. Thank you calling. That helps a lot.”
“Glad to help. Good bye”

In digital communications the clients and servers work in a similar manner. Each transaction usually begins with a "hello" packet. There are many "Acknowledge" packets, which are essentially "OK" messages. Finally, a "Goodbye" packet ends the transaction. Wireshark enables you to see many conversations that are taking place simultaneously on a network in a variety of protocols.

Other Common Protocols

Here is a list of common protocols. Complete the table by researching the name and a one sentence description of each protocol.

Initials	Name	What it is used for
HTTP	Hypertext Transfer Protocol	Send hypertext
TCP	Transmission Control Protocol	Reliable data transmission over networks
SNMP	Simple Network Management Protocol	Monitor and manage network devices
FTP	File Transfer Protocol	Transfer files between computers
SMTP	Simple Mail Transfer Protocol	Send email messages
IMAP	Internet Message Access Protocol	Access email messages stored on server
POP3	Post Office Protocol 3	Download email messages to local device
DNS	Domain Name System	Translate domain names to IP addresses
SSH	Secure Shell	Secure remote access to computers
UDP	User Datagram Protocol	Fast but unreliable data transmission

Initials	Name	What it is used for
RDP	Remote Desktop Protocol	Remote desktop access to Windows systems
VoIP	Voice over Internet Protocol	Make voice calls over internet networks
DHCP	Dynamic Host Configuration Protocol	Automatically assign IP addresses
LDAP	Lightweight Directory Access Protocol	Access and manage directory services
TLS	Transport Layer Security	Encrypt communications for security

How to Fix the Application's Vulnerability

To protect data in transit, we need to add encryption to the login application. The **http** protocol is unencrypted and therefore vulnerable to network sniffers like Wireshark. The **https** protocol was developed to provide secure transport. At first, https was used only for sensitive information such as logins or financial transactions. Unencrypted data is less computationally expensive. Later, https became the default mode to transfer all data.

Here are the tasks we must do to transmit data in encrypted format:

1. Create an SSL / TLS certificate.
2. Configure the application to use the https protocol.
3. Direct old http requests to https.

About Certificates

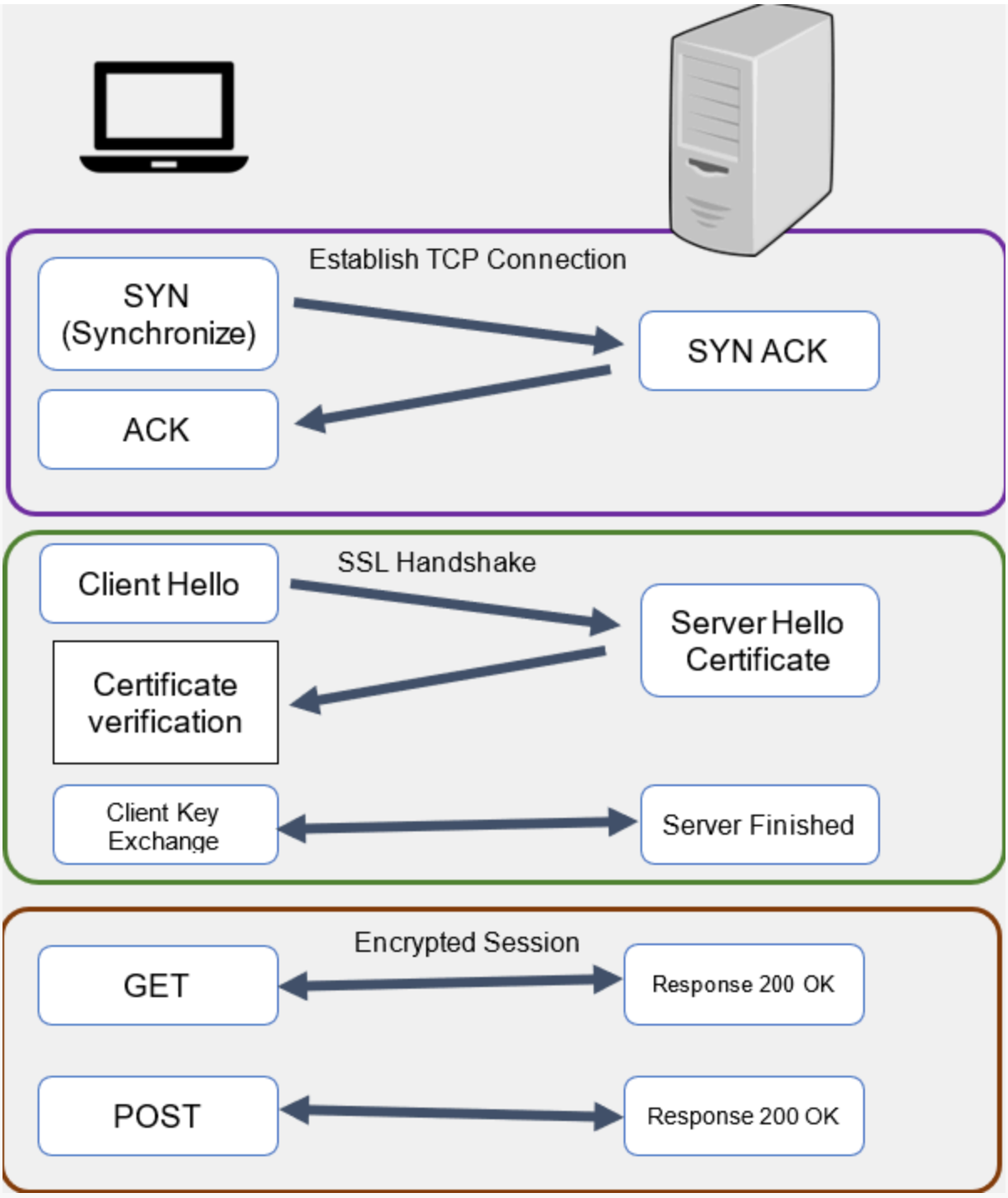
What Are SSL Certificates?

SSL (secure sockets layer) certificates are digital certificates that provide a way to encrypt communication between a user's browser and a web server. SSL was replaced by TLS in 2015, but the protocol remains to be called SSL. SSL certificates are like the public and private keys used in the GPG exercise done earlier in this course.

How SSL Certificates Help Encrypt Data

When a client connects to a server via HTTPS, an SSL handshake occurs. During this handshake, the client and server use the public and private keys to create a shared encryption key that is used for the duration of the session, as seen in Figure 43.

- The server presents its SSL certificate to the client. The server's public key is part of the certificate.
- The client and server agree on an encryption method.
- A unique session key is created for this specific connection.
- This session key is used to encrypt and decrypt the data exchanged during the session.


Establish TCP Connection
SYN (Synchronize)
SYN ACK
ACK
SSL Handshake
Client Hello
Server Hello Certificate
Certificate verification
Client Key Exchange
Server Finished
Encrypted Session
GET
Response 200 OK
POST
Response 200 OK

**How Do We Know if an SSL Certificate is Valid?

Any computer can generate an SSL. In fact, in the next steps we will create our own certificate. However, on a real website you need to choose from a list of "white listed" certificate providers. A **certificate Aathority (CA)** is an organization or entity responsible for issuing digital certificates. Some well-known CA organizations include **Symantec, Comodo**, and **GoDaddy**, as seen in Figures 44 and 45. These companies are recognized and trusted by Apple, Google, Microsoft, and other leading tech companies. As of August 2024, 133 CA organizations are trusted by Microsoft Windows. The US Federal Government manages the **Federal PKI** for use with all government applications.

- **Validity Period**: SSL certificates have a validity period, often one or two years. They must be renewed before they expire to continue ensuring secure connections.

Error Messages from Self-Signed Certificates

Self-signed certificates are SSL/TLS certificates that are generated and signed by the organization itself, rather than a trusted CA. These certificates can be used for testing and internal purposes but are not trusted by browsers by default because they do not come from a recognized CA.

Common Errors with Self-Signed Certificates:

- **"Your connection is not private"**: Seen in Figure 46, this error occurs because the browser cannot verify the certificate's authenticity.
- **"Invalid certificate" or "Untrusted certificate"**: This indicates that the certificate is not from a trusted CA.
- **"Self-signed certificate in the certificate chain"**: This error specifically indicates that the certificate is self-signed and not trusted.

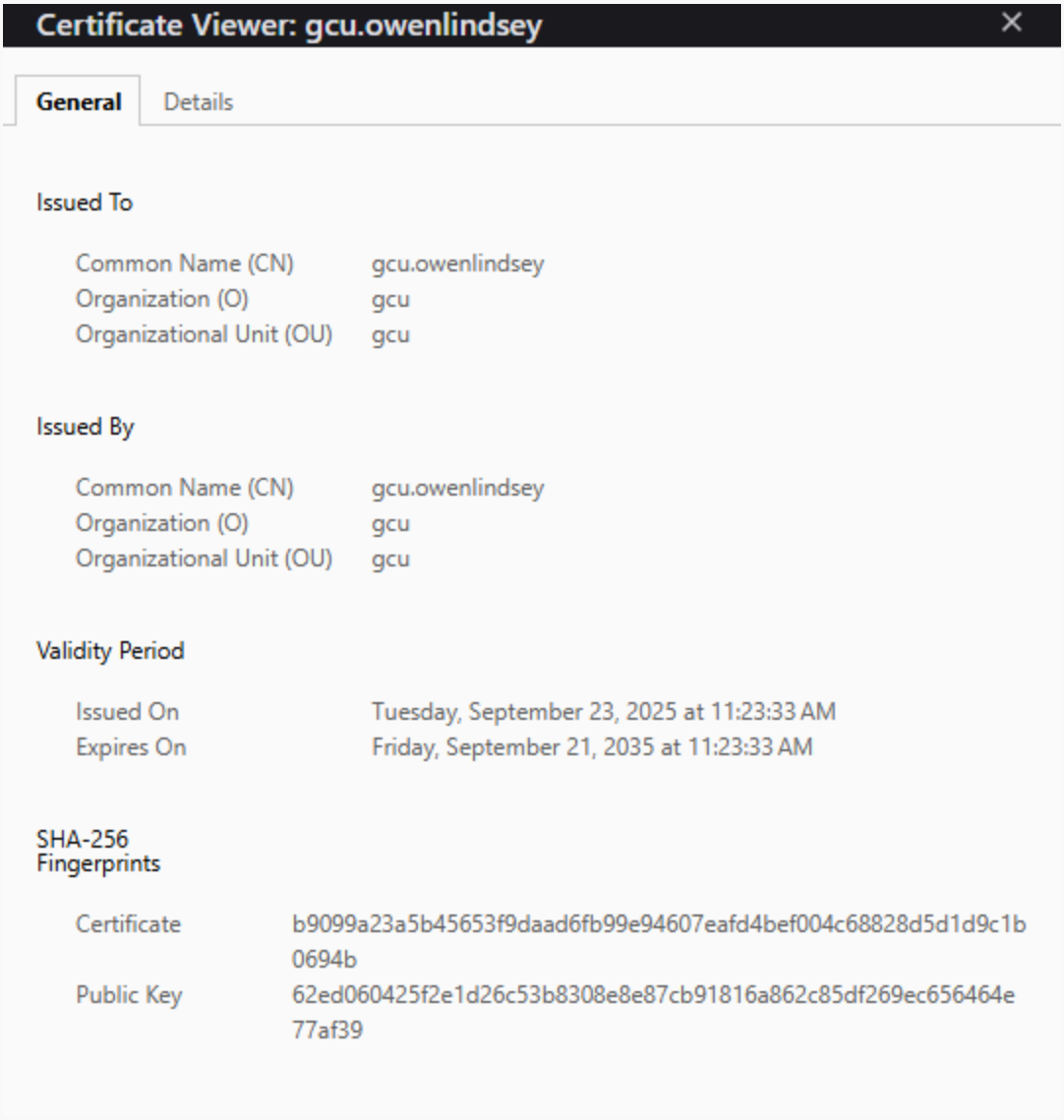
About the SSL Create Statement

keytool -genkeypair -alias springboot -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore keystore.p12 -validity 3650

Here's a breakdown of each part of the SSL generator.

- **-genkeypair**: Generates a public-private key pair.
- **-alias springboot**: Specifies the alias for the key pair in the keystore.
- **-keyalg RSA**: Specifies the algorithm to be used for generating the key pair, in this case, RSA.
- **-keysize 2048**: Specifies the size of the key, which is 2048 bits.

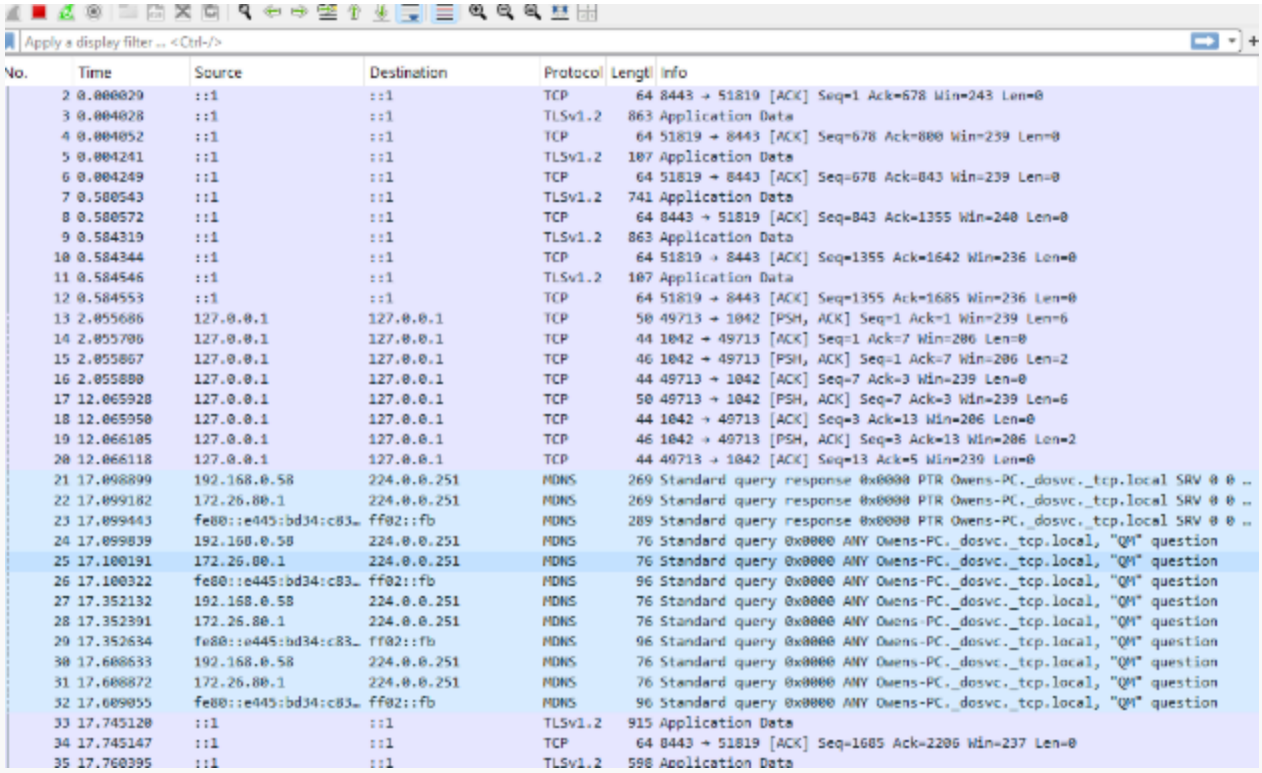
- **-storetype PKCS12**: Specifies the type of keystore to be created, PKCS12 is a standard for storing cryptographic information.
- **-keystore keystore.p12**: Specifies the name of the keystore file to be created.
- **-validity 3650**: Specifies the validity period of the certificate in days, which in this case is 10 years.



Repeat the Wireshark Process with the SSL-Enabled Application

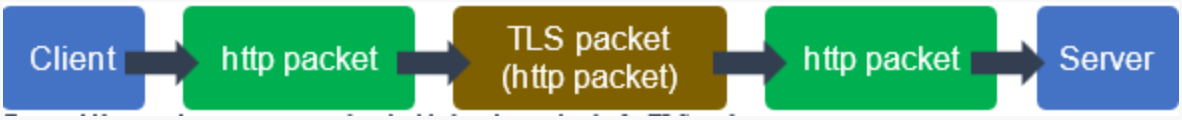
Complete the following steps to demonstrate that even though the login credentials can still be captured by Wireshark, the packets are now unreadable due to SSL encryption.

- Start Wireshark or restart the capture



No.	Time	Source	Destination	Protocol	Length	Info
2	0.000029	::1	::1	TCP	64	8443 → 51819 [ACK] Seq=1 Ack=678 Win=243 Len=0
3	0.004028	::1	::1	TLSv1.2	863	Application Data
4	0.004052	::1	::1	TCP	64	51819 → 8443 [ACK] Seq=678 Ack=800 Win=239 Len=0
5	0.004241	::1	::1	TLSv1.2	107	Application Data
6	0.004249	::1	::1	TCP	64	51819 → 8443 [ACK] Seq=678 Ack=843 Win=239 Len=0
7	0.580543	::1	::1	TLSv1.2	741	Application Data
8	0.580572	::1	::1	TCP	64	8443 → 51819 [ACK] Seq=843 Ack=1355 Win=240 Len=0
9	0.584319	::1	::1	TLSv1.2	863	Application Data
10	0.584344	::1	::1	TCP	64	51819 → 8443 [ACK] Seq=1355 Ack=1642 Win=236 Len=0
11	0.584546	::1	::1	TLSv1.2	107	Application Data
12	0.584553	::1	::1	TCP	64	51819 → 8443 [ACK] Seq=1355 Ack=1685 Win=236 Len=0
13	2.055686	127.0.0.1	127.0.0.1	TCP	50	49713 → 1042 [PSH, ACK] Seq=1 Ack=1 Win=239 Len=6
14	2.055706	127.0.0.1	127.0.0.1	TCP	44	1042 → 49713 [ACK] Seq=1 Ack=7 Win=206 Len=0
15	2.055867	127.0.0.1	127.0.0.1	TCP	46	1042 → 49713 [PSH, ACK] Seq=1 Ack=7 Win=206 Len=2
16	2.055880	127.0.0.1	127.0.0.1	TCP	44	49713 → 1042 [ACK] Seq=7 Ack=3 Win=239 Len=0
17	12.065928	127.0.0.1	127.0.0.1	TCP	50	49713 → 1042 [PSH, ACK] Seq=7 Ack=3 Win=239 Len=6
18	12.065950	127.0.0.1	127.0.0.1	TCP	44	1042 → 49713 [ACK] Seq=3 Ack=13 Win=206 Len=0
19	12.066105	127.0.0.1	127.0.0.1	TCP	46	1042 → 49713 [PSH, ACK] Seq=3 Ack=13 Win=206 Len=2
20	12.066118	127.0.0.1	127.0.0.1	TCP	44	49713 → 1042 [ACK] Seq=13 Ack=5 Win=239 Len=0
21	17.098899	192.168.0.58	224.0.0.251	MDNS	269	Standard query response 0x0000 PTR Owens-PC._dosvc._tcp.local SRV 0 0 …
22	17.099182	172.26.80.1	224.0.0.251	MDNS	269	Standard query response 0x0000 PTR Owens-PC._dosvc._tcp.local SRV 0 0 …
23	17.099443	fe80::e445:bd34:c83…	ff02::fb	MDNS	289	Standard query response 0x0000 PTR Owens-PC._dosvc._tcp.local SRV 0 0 …
24	17.099839	192.168.0.58	224.0.0.251	MDNS	76	Standard query 0x0000 ANY Owens-PC._dosvc._tcp.local, "QM" question
25	17.100191	172.26.80.1	224.0.0.251	MDNS	76	Standard query 0x0000 ANY Owens-PC._dosvc._tcp.local, "QM" question
26	17.100322	fe80::e445:bd34:c83…	ff02::fb	MDNS	96	Standard query 0x0000 ANY Owens-PC._dosvc._tcp.local, "QM" question
27	17.352132	192.168.0.58	224.0.0.251	MDNS	76	Standard query 0x0000 ANY Owens-PC._dosvc._tcp.local, "QM" question
28	17.352391	172.26.80.1	224.0.0.251	MDNS	76	Standard query 0x0000 ANY Owens-PC._dosvc._tcp.local, "QM" question
29	17.352634	fe80::e445:bd34:c83…	ff02::fb	MDNS	96	Standard query 0x0000 ANY Owens-PC._dosvc._tcp.local, "QM" question
30	17.608633	192.168.0.58	224.0.0.251	MDNS	76	Standard query 0x0000 ANY Owens-PC._dosvc._tcp.local, "QM" question
31	17.608872	172.26.80.1	224.0.0.251	MDNS	76	Standard query 0x0000 ANY Owens-PC._dosvc._tcp.local, "QM" question
32	17.609055	fe80::e445:bd34:c83…	ff02::fb	MDNS	96	Standard query 0x0000 ANY Owens-PC._dosvc._tcp.local, "QM" question
33	17.745128	::1	::1	TLSv1.2	915	Application Data
34	17.745147	::1	::1	TCP	64	8443 → 51819 [ACK] Seq=1685 Ack=2206 Win=237 Len=0
35	17.760395	::1	::1	TLSv1.2	598	Application Data

The traffic protocol used by the application is no longer listed as **http** (clear text). Nor is there a protocol listed as **https**. Instead, you will see several TLSv1.2 events in the log. Wireshark labels the packets as "TLS" (or "SSL" for older versions) instead of "HTTPS" because it is indicating that the traffic is encrypted with the TLS protocol. Wireshark does not show "HTTPS" as the protocol because HTTPS is not a protocol layer on its own. Https it is just HTTP wrapped inside a secure TLS/SSL "tunnel,"



Explanation of the Wireshark data

- The captured packets show a sequence of secure communications.
- It is impossible to determine if these occurred during a login event or some other transaction.
- The encryption provided by TLS ensures that sensitive information like usernames and passwords cannot be easily intercepted or viewed by unauthorized parties.
- Any attempt to view the packets is futile. For example, the description of packet #1 shown below is "application data," a generic term.
- Viewing the contents results in a message that describes the data as "encrypted data."
- Each TLS packets is followed by a TCP Acknowledge packet, as seen in Figures 67 and 68. The Acknowledge packet is simply a confirmation that the previous transaction occurred without communication errors.

```
TCP        44 49713 → 1042 [ACK] Seq=67 Ack=23 Win=239 Len=0
TCP        50 49713 → 1042 [PSH, ACK] Seq=67 Ack=23 Win=239 Len=6
TCP        44 1042 → 49713 [ACK] Seq=23 Ack=73 Win=206 Len=0
TCP        46 1042 → 49713 [PSH, ACK] Seq=23 Ack=73 Win=206 Len=2
TCP        44 49713 → 1042 [ACK] Seq=73 Ack=25 Win=239 Len=0
TCP        50 49713 → 1042 [PSH, ACK] Seq=73 Ack=25 Win=239 Len=6
TCP        44 1042 → 49713 [ACK] Seq=25 Ack=79 Win=206 Len=0
TCP        46 1042 → 49713 [PSH, ACK] Seq=25 Ack=79 Win=206 Len=2
TCP        44 49713 → 1042 [ACK] Seq=79 Ack=27 Win=239 Len=0
TCP        65 [TCP Keep-Alive] 51819 → 8443 [ACK] Seq=2205 Ack=2303 Win=233 Len=1
TCP        76 [TCP Keep-Alive ACK] 8443 → 51819 [ACK] Seq=2303 Ack=2206 Win=237 Len=…
TCP        50 49713 → 1042 [PSH, ACK] Seq=79 Ack=27 Win=239 Len=6
TCP        44 1042 → 49713 [ACK] Seq=27 Ack=85 Win=206 Len=0
TCP        46 1042 → 49713 [PSH, ACK] Seq=27 Ack=85 Win=206 Len=2
TCP        44 49713 → 1042 [ACK] Seq=85 Ack=29 Win=239 Len=0
```

Summary of Key Concepts

This lesson demonstrated the critical importance of data encryption in web applications through hands-on network traffic analysis. Using Wireshark, we observed how unencrypted HTTP communications expose sensitive information such as usernames and passwords in plain text, making them vulnerable to network sniffing attacks. The exercise showed that any data transmitted over HTTP can be easily intercepted and read by unauthorized parties monitoring network traffic. To address this vulnerability, we implemented HTTPS by generating an SSL/TLS certificate and configuring our Spring Boot application for secure communications. The comparison between HTTP and HTTPS traffic in Wireshark clearly illustrated how TLS encryption transforms readable login credentials into unintelligible encrypted data packets labeled as "Application Data." This demonstration reinforced that HTTPS is essentially HTTP wrapped within a secure TLS tunnel, and that proper certificate implementation is essential for protecting data in transit. The lesson emphasized that while network monitoring tools like Wireshark serve legitimate purposes for network diagnostics and security analysis, they also highlight why encryption protocols are fundamental to modern web security and user privacy protection.