Getnet Dejene

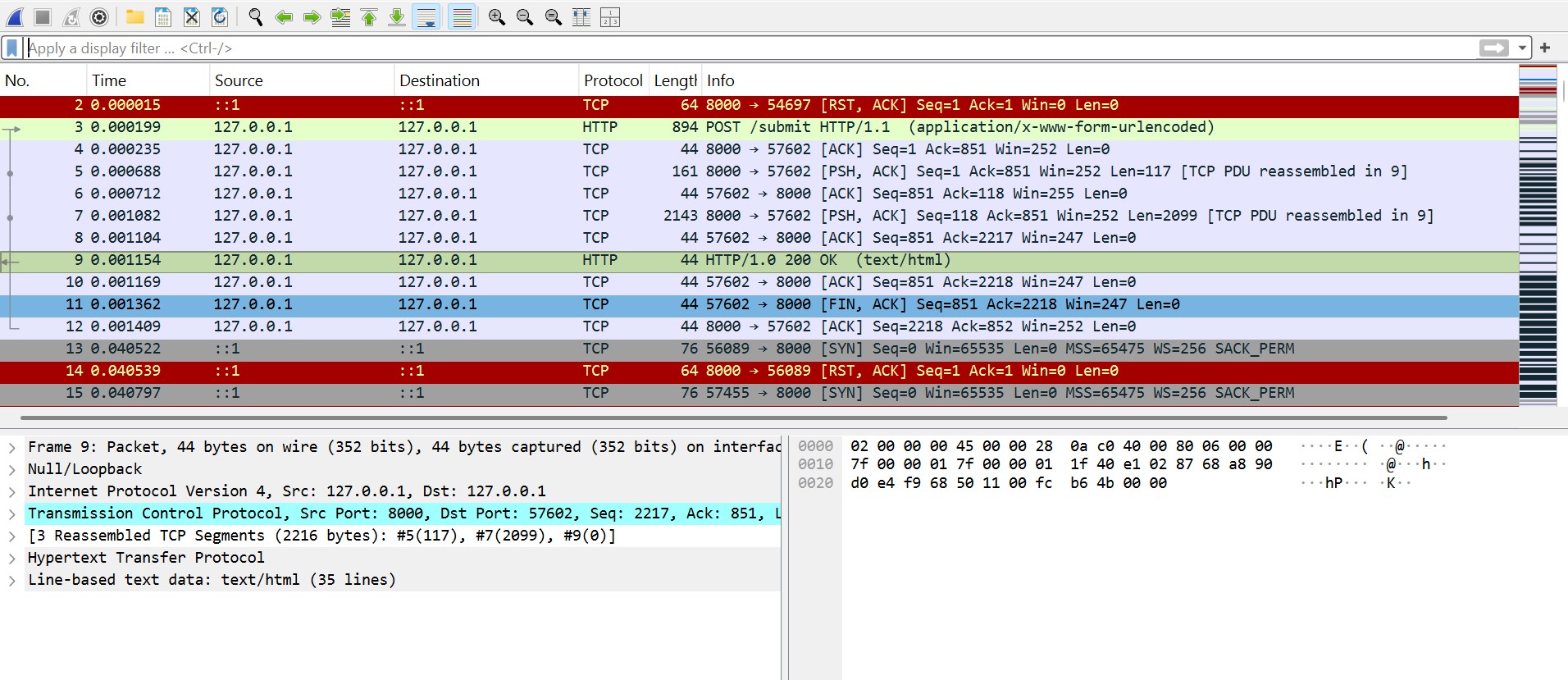
Assignment: PKI

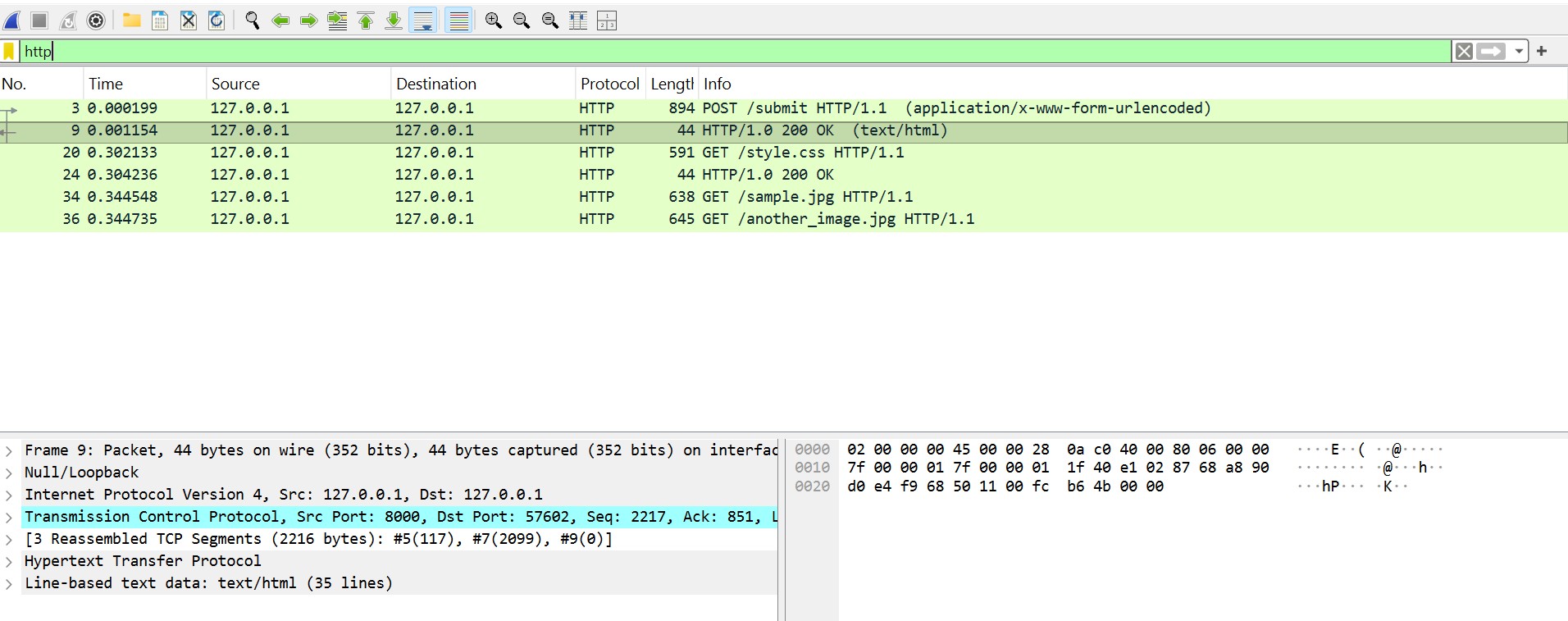
Part 1:

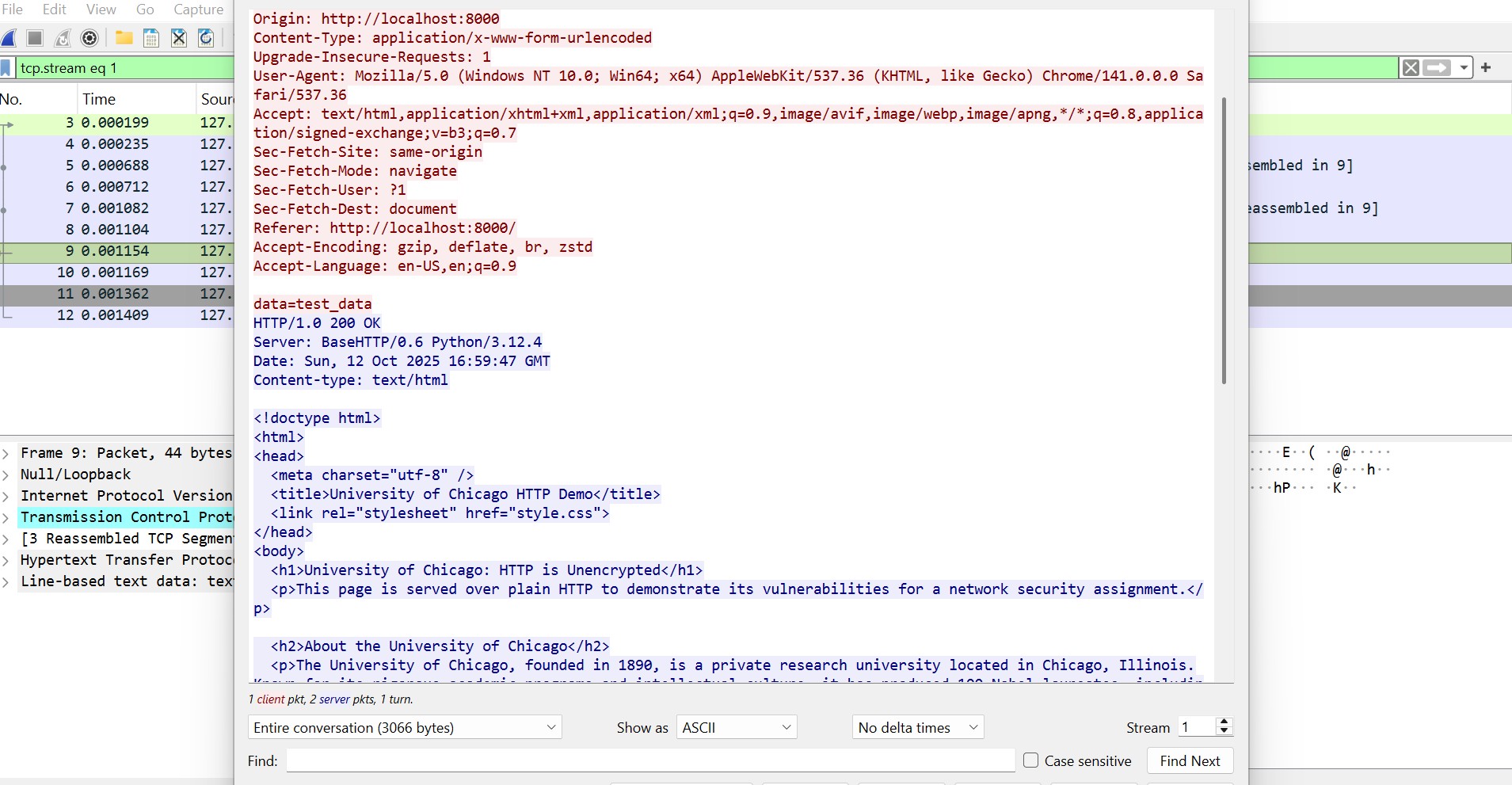
HTTP is insecure because it transmits data between the client (browser) and server in plain text, without encryption. This allows eavesdroppers to intercept and read all requests, responses, and their contents, including HTML, images, and form data. Unlike HTTPS, which uses TLS for encryption, HTTP lacks confidentiality, authentication, and integrity, making it vulnerable to man-in-the-middle (MITM) attacks.

I set up a Python HTTP server (server.py) to serve index.html at http://localhost:8000. The page includes detailed information about the University of Chicago, two images (sample.jpg, another\_image.jpg), and a form submitting to /submit via POST.

Below are screenshots that are captured using Wireshark.







The above screenshots demonstrate that HTTP is insecure and vulnerable for the following reasons.

* No Confidentiality: All data, including form submissions requested resources (e.g., index.html, sample.jpg, another\_image.jpg), and their contents, are readable by anyone who intercepts the network traffic.
* Vulnerability to Eavesdropping: An attacker on the same network can use tools like Wireshark to "sniff" packets and see:
  + Requested Resources: URLs like /index.html, /sample.jpg,another\_image.jpg
  + Resource Contents: The HTML code, image data
  + Form Data: POST data like data=test\_data sent to /submit.
* No Authentication: HTTP doesn’t verify the server’s identity, so a client could communicate with a malicious server.
* No Integrity: Data can be modified in transit without detection.

The primary issue is that HTTP is unencrypted, making all communication visible to eavesdroppers.

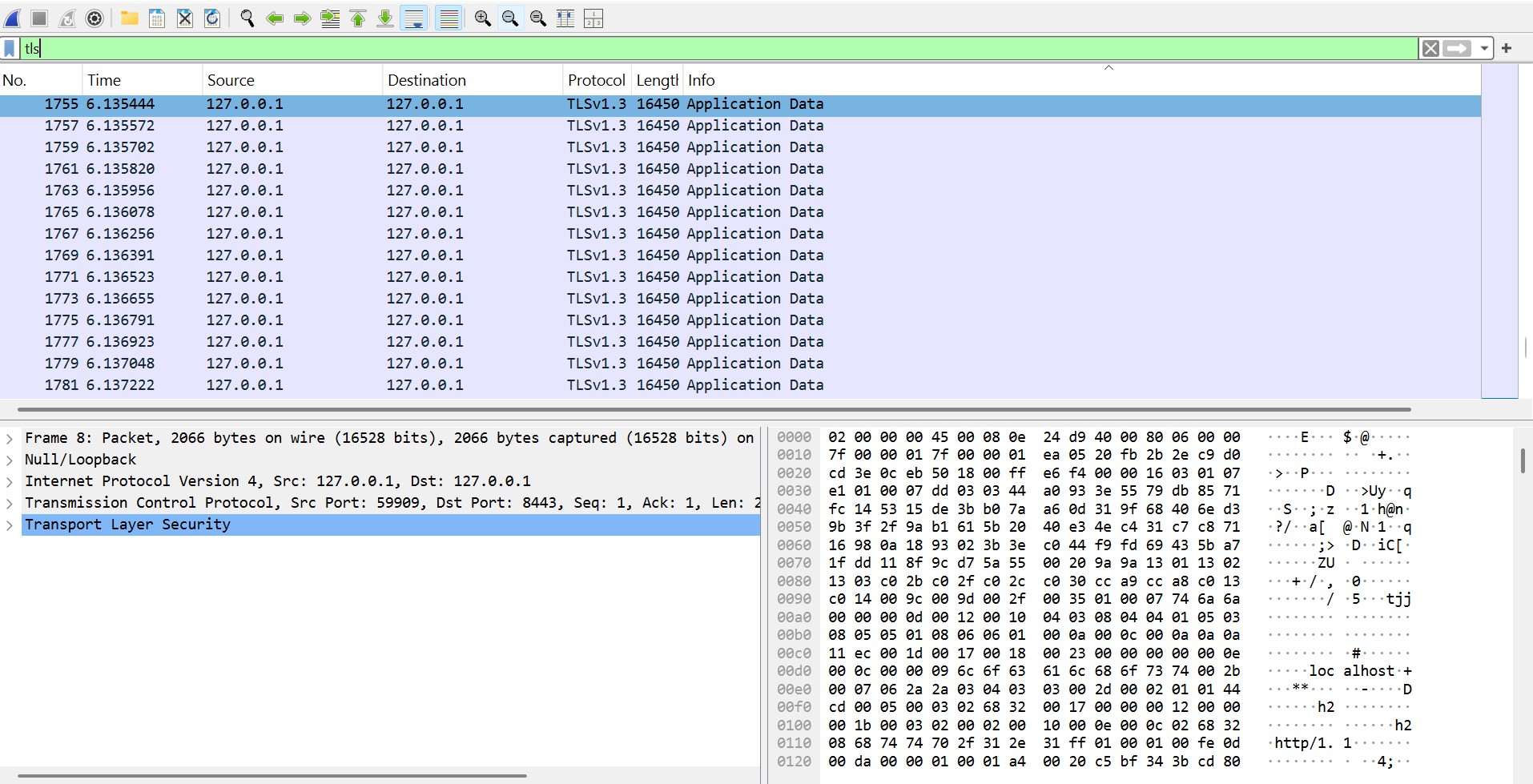
Part 3

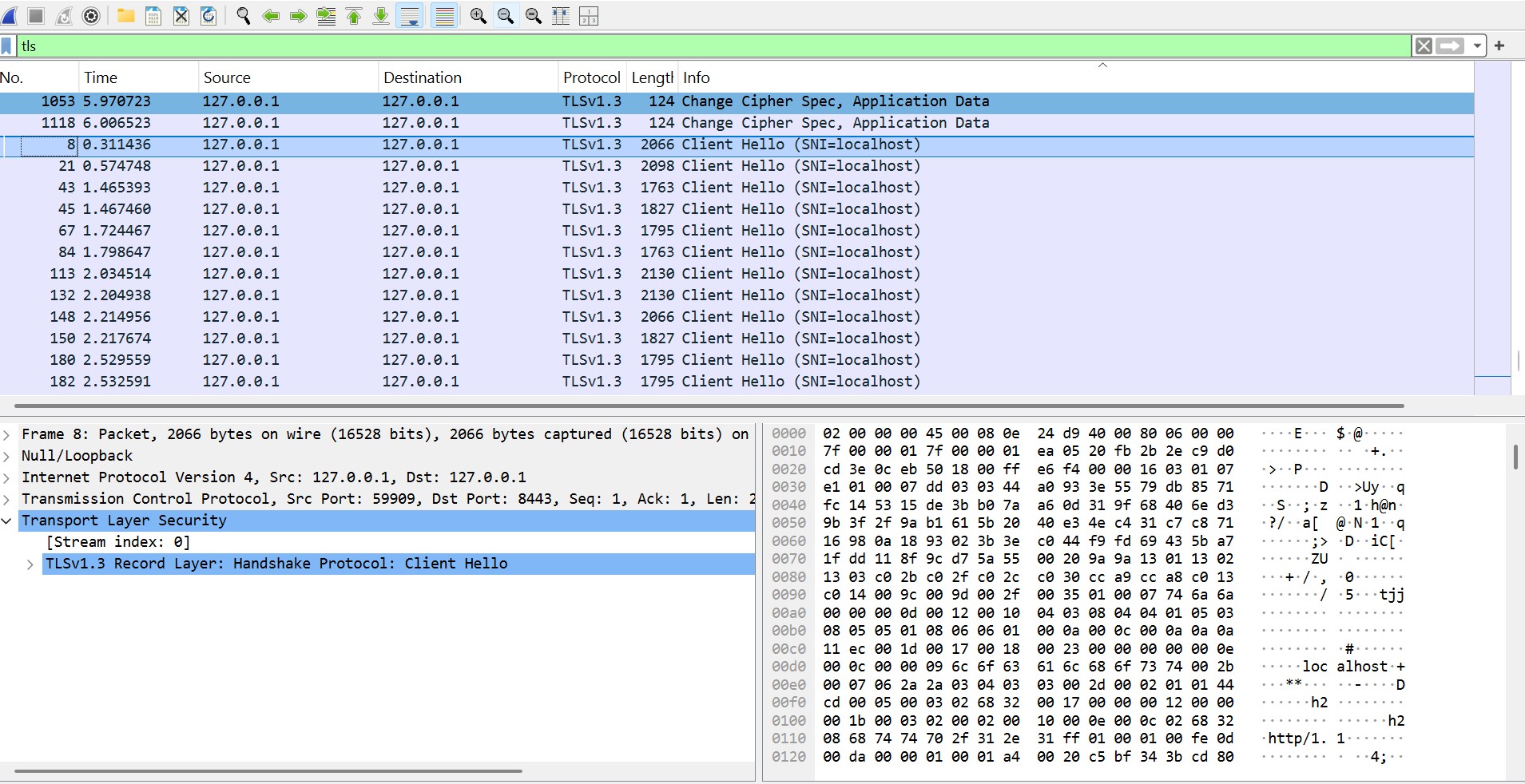
A Certificate Authority (CA) won’t issue an SSL certificate for localhost because it requires a publicly resolvable domain to verify ownership. Localhost and private IPs (e.g., 127.0.0.1) aren’t registered in public DNS, making verification impossible. A self-signed certificate is used for local testing, added to trusted roots manually.

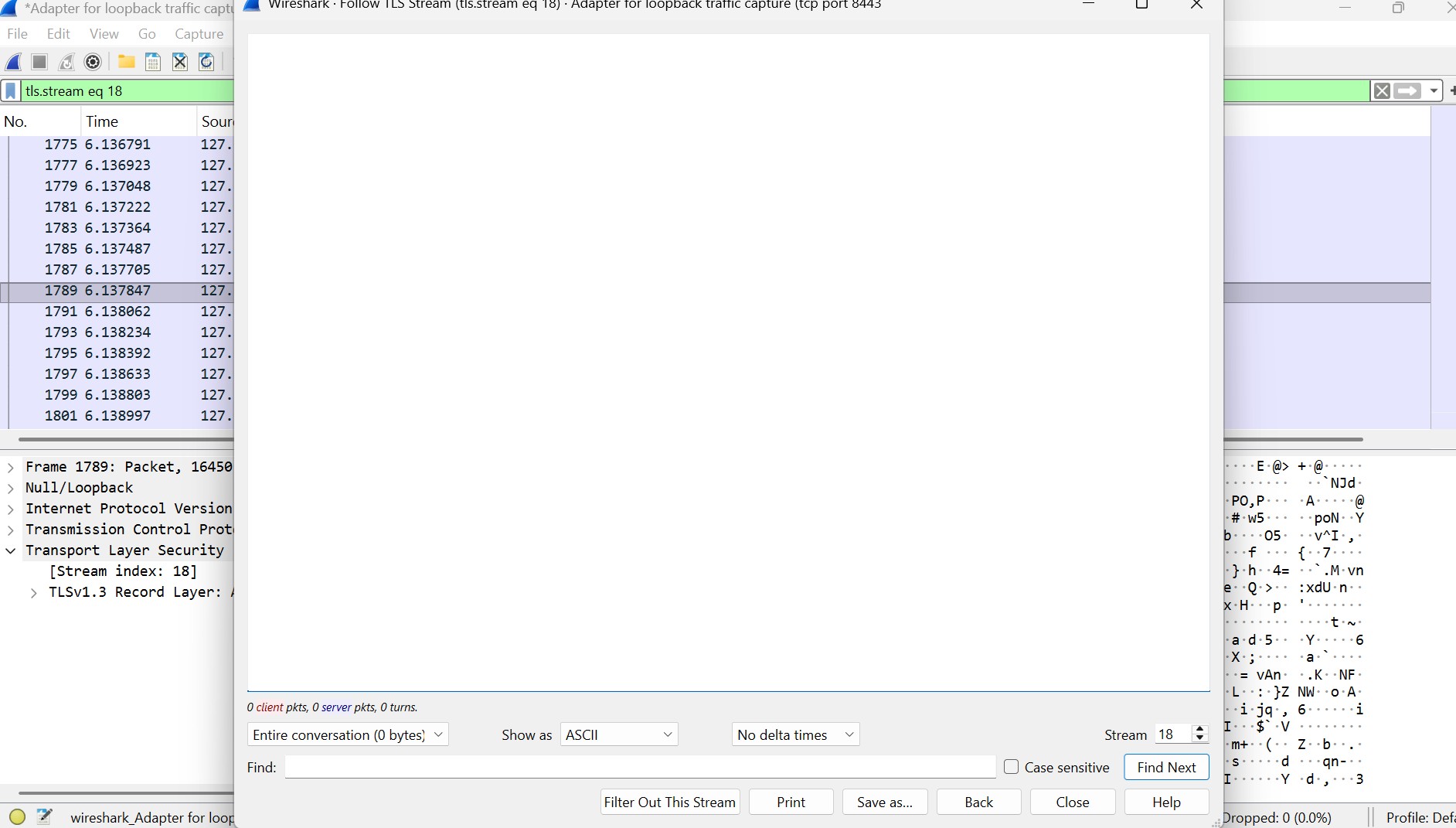
Why a CA Won’t Issue a Certificate

A Certificate Authority (CA) won’t issue an SSL certificate for localhost because it requires a publicly resolvable domain to verify ownership via DNS or HTTP challenges. Localhost and private IPs (e.g., 127.0.0.1) aren’t in public DNS, so verification is impossible. For local testing, I used a self-signed certificate, trusted manually.HTTPS Implementation

I generated a self-signed certificate in Git Bash with Common Name localhost. I added server.crt to Windows’ Trusted Root Certification Authorities to avoid browser warnings. I updated server.py to use HTTPS on port 8443, loading server.crt and server.key with Python’s ssl module. The server runs at https://localhost:8443, serving index.html with University of Chicago info, two images (sample.jpg, another\_image.jpg), and a form. Submitting the form displays Received: data=test\_data. Below are screenshots after upgrading to HTTPS.







Wireshark Comparison

In part two, Wireshark captured HTTP traffic at http://localhost:8000 (filter: tcp port 8000). GET / showed the HTML with University of Chicago details (e.g., “founded in 1890”) in plain text, GET /sample.jpg and /another\_image.jpg showed resource URLs, and POST /submit showed data=test\_data in plain text [Screenshot: post\_request.png]. In part three, capturing HTTPS traffic at https://localhost:8443 (filter: tcp port 8443) showed TLS packets (e.g., “Client Hello”, “Application Data”). The HTML, image requests, and POST data were encrypted, appearing as unreadable “Application Data” [Screenshot: tls\_packet.png]. HTTPS’s TLS encryption prevents eavesdroppers from reading data, unlike HTTP’s plain-text exposure.

Conclusion

Upgrading to HTTPS secured communication. HTTP exposed university info and form data in plain text, while HTTPS’s TLS encryption ensured confidentiality, protecting sensitive data.