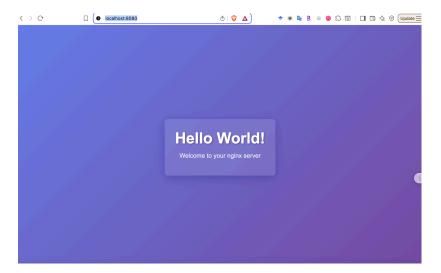
Lab 1 PKI

Dixi Yao, dixi@uchicago.edu

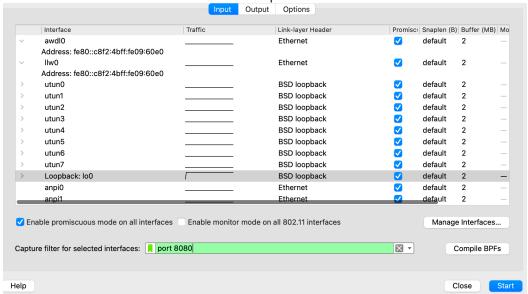
Task 1: Host a local web server

For web-server, I used nginx. I asked cursor with prompt "write me a simple webpage, with hello world, using nginx on localhost" and it generates a webpage with nginx for me. I checked the code and revised a little bit to make it running on my local host. After installing nginx and run command `sudo nginx -c "\$(pwd)/nginx.conf" `, I got my localhost working on http://localhost:8080/.



2. Identify why HTTP is not secure

To answer the question, I first tried to use Wireshark to hi-jack the traffic. Because my web server is pretty simple. So the only basic command my client do is to send the simplest httrequest. I used `response = requests.get(server_url, timeout=10)` and let it repeat for 5 times to see the transmission. Here is the set up of wireshark:



So the basic idea is that as we can see in the Wireshark, an eavesdropper can capture the traffic and packets. For packets, we can see that there are a lot of TCP flows. HTTP is an application built upon the TCP flows. We put our http information into the TCP packet. As a result, if an eavesdropper gets these TCP packets. It can try to recover the HTTP content. We can see after the TCP protocol handshaking and http get send, we will have a response with HTTP/1.1 200 ok

No.		Time	Source	Destination	Protocol	Length	Info	
_	1	0.000000	127.0.0.1	127.0.0.1	TCP	68	62660 → 8080 [SYN] Seg=0 Win=65535 Len=0 MSS=1	634
	2	0.000114	127.0.0.1	127.0.0.1	TCP	68	8080 → 62660 [SYN, ACK] Seg=0 Ack=1 Win=65535	Lei
	3	0.000148	127.0.0.1	127.0.0.1	TCP	56	62660 → 8080 [ACK] Seg=1 Ack=1 Win=408256 Len=	0 .
>	4	0.000164	127.0.0.1	127.0.0.1	HTTP	205	GET / HTTP/1.1	
	5	0.000169	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8080 → 62660 [ACK] Seg=1 A	ck:
	6	0.000190	127.0.0.1	127.0.0.1	TCP	56	8080 → 62660 [ACK] Seq=1 Ack=150 Win=408128 Le	n=(
-	7	0.000747	127.0.0.1	127.0.0.1	HTTP	1145	HTTP/1.1 200 OK (text/html)	
	8	0.000770	127.0.0.1	127.0.0.1	TCP	56	62660 → 8080 [ACK] Seq=150 Ack=1090 Win=407168	L
	9	0.001255	127.0.0.1	127.0.0.1	TCP	56	62660 → 8080 [FIN, ACK] Seq=150 Ack=1090 Win=4	07:
	10	0.001284	127.0.0.1	127.0.0.1	TCP	56	8080 → 62660 [ACK] Seq=1090 Ack=151 Win=408128	Le
	11	0.001292	127.0.0.1	127.0.0.1	TCP	56	8080 → 62660 [FIN, ACK] Seq=1090 Ack=151 Win=4	08:
L	12	0.001321	127.0.0.1	127.0.0.1	TCP	56	62660 → 8080 [ACK] Seq=151 Ack=1091 Win=407168	Le
	13	1.004933	127.0.0.1	127.0.0.1	TCP	68	62665 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=1	634
	14	1.005111	127.0.0.1	127.0.0.1	TCP	68	8080 → 62665 [SYN, ACK] Seq=0 Ack=1 Win=65535	Lei
	15	1.005136	127.0.0.1	127.0.0.1	TCP	56	62665 → 8080 [ACK] Seq=1 Ack=1 Win=408256 Len=	0 -
	16	1.005154	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8080 → 62665 [ACK] Seq=1 A	ck:
	17	1.005208	127.0.0.1	127.0.0.1	HTTP	205	GET / HTTP/1.1	
	18	1.005242	127.0.0.1	127.0.0.1	TCP	56	8080 → 62665 [ACK] Seq=1 Ack=150 Win=408128 Le	n=(
	19	1.005851	127.0.0.1	127.0.0.1	HTTP	1145	HTTP/1.1 200 OK (text/html)	
	20	1.005878	127.0.0.1	127.0.0.1	TCP	56	62665 → 8080 [ACK] Seq=150 Ack=1090 Win=407168	L
	21	1 006286	127 0 0 1	127 0 0 1	TCP	56	62665 → 8080 [FTN ACK] Sen=150 Ack=1000 Win=4	97·
	[Full request URI: http://localhost:8080/]						00 00 45 00 04 75 00 00 40 00 40 06 00 00	· · E ·
	> HTTP chunked response						00 01 71 00 00 01 11 70 14 04 10 40 04 00	
	Content-encoded entity body (gzip): 579 bytes -> 1176 bytes						,, 0, 00 10 10 0, 01 00 00 01 01 00 0	
		e Data: 1176 k	,					00 O
∨ Li	Line-based text data: text/html (44 lines)							ginx
		OCTYPE html>\r			0060	61 74 6	65 3a 20 53 75 6e 2c 20 30 35 20 4f 63 74 at	e: S
	<htr< td=""><td>nl lang="en">\</td><td>\n</td><td></td><td></td><td></td><td></td><td>025</td></htr<>	nl lang="en">\	\n					025
	<he< td=""><td>ad>\n</td><td></td><td></td><td>0080</td><td></td><td></td><td>··Co</td></he<>	ad>\n			0080			··Co
		<meta charse<="" td=""/> <td>t="UTF-8">\n</td> <td></td> <td></td> <td></td> <td></td> <td>text Modi</td>	t="UTF-8">\n					text Modi
				.dth=device-width, init				5 Oc
	<title>Hello World</title> \n						3a 30 39 20 47 4d 54 0d 0a 54 72 61 6e 73 11	:09
		<style>\n</td><td></td><td></td><td>00d0</td><td>66 65 7</td><td>72 2d 45 6e 63 6f 64 69 6e 67 3a 20 63 68 fe</td><td>r-En</td></tr><tr><td></td><td colspan=6>body {\n</td><td>De-chunked entity body (579 byt Uncompressed entity body (1176</td><td>byt</td></tr><tr><td>0 3</td><td>1</td><td>oopback: lo0: <live</td><td>capture in progress></td><td></td><td></td><td></td><td>Packets: 60 Profile: Def</td><td>ault</td></tr></tbody></table></style>						

And we can see that the original webpage of them is also encoded in the protocol (because server needs to send it to the client for get so that client and show in the browser). Then, all information about http is exposed. For eavesdropper, it first checks all TCP protocol and sort them with frame number. Then on sequence with sync, sync ACK, and ACK of TCP, HTTP protocol will be identified. And similar know which on contains the content of HTTP.

3. Create a self-signed certificate and upgrade your web server to HTTPS So the first step is to upgrade my web server to HTTPS server. I directly put the question into cursor and let it upgrade my web server. First, we can use openssl to generate an SSL with `openssl req -x509 -newkey rsa:4096 -keyout ssl/server.key -out ssl/server.crt -days 365 -nodes -subj "/C=US/ST=Illinois/L=Chicago/O=CMSC30350/OU=Lab1/CN=localhost".Next, because my local root is MacOS, I directly added my SSL into the keychain so that when I connect my client with the server, it will be easier to verify. Here are some changes to the HTTPS server. So first, I need to add the configuration of using SSL in the configuration file:

The next thing is add the SSL certificate into my server

```
# HTTPS Server
server {
    listen 8443 ssl;
    server_name localhost;

# SSL Certificate Configuration
    ssl_certificate "/Users/dixiyao/Desktop/Courses/CMSC 30350 1 Security, Privacy, and Consumer
    Protection/lab1/ssl/server.crt";
    ssl_certificate_key "/Users/dixiyao/Desktop/Courses/CMSC 30350 1 Security, Privacy, and Consumer
    Protection/lab1/ssl/server.key";
```

Again, we can use the wireshark to get the traffic and let's see the traffic this time. Because we have added certificates to our local host, we can also get the traffic. However, if I use the client to do it, which does not contain the certificate. I will get HTTP 1.1 /400 Bad request.

Apply a display filter <\%\/>							
No.	Time	Source	Destination	Protocol	ol Length Info		
	43 0.304735	127.0.0.1	127.0.0.1	TCP	56 8080 → 56437 [FIN, ACK] Seq=3801 Ack=3239 Win=40		
4	44 0.304777	127.0.0.1	127.0.0.1	TCP	56 56437 → 8080 [ACK] Seq=3239 Ack=3802 Win=404480		
4	45 156.620775	127.0.0.1	127.0.0.1	TCP	68 57154 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=163		
4	46 156.620879	127.0.0.1	127.0.0.1	TCP	68 8080 → 57154 [SYN, ACK] Seq=0 Ack=1 Win=65535 Le		
4	47 156.620921	127.0.0.1	127.0.0.1	TCP	56 57154 → 8080 [ACK] Seq=1 Ack=1 Win=408256 Len=0		
4	48 156.620934	127.0.0.1	127.0.0.1	HTTP	205 GET / HTTP/1.1		
4	49 156.620941	127.0.0.1	127.0.0.1	TCP	56 [TCP Window Update] 8080 → 57154 [ACK] Seq=1 Ack		
į	50 156.620954	127.0.0.1	127.0.0.1	TCP	56 8080 → 57154 [ACK] Seq=1 Ack=150 Win=408128 Len=		
į	51 156.621272	127.0.0.1	127.0.0.1	HTTP	753 HTTP/1.1 400 Bad Request (text/html)		
į	52 156.621295	127.0.0.1	127.0.0.1	TCP	56 57154 → 8080 [ACK] Seq=150 Ack=698 Win=407552 Le		
į	53 156.621516	127.0.0.1	127.0.0.1	TCP	56 57154 → 8080 [FIN, ACK] Seq=150 Ack=698 Win=4075		
í	54 156.621536	127.0.0.1	127.0.0.1	TCP	56 8080 → 57154 [ACK] Seq=698 Ack=151 Win=408128 Le		
į	55 156.621690	127.0.0.1	127.0.0.1	TCP	56 8080 → 57154 [FIN, ACK] Seq=698 Ack=151 Win=4081		
į	56 156.621718	127.0.0.1	127.0.0.1	TCP	56 57154 → 8080 [ACK] Seq=151 Ack=699 Win=407552 Le		
į	57 157.630333	127.0.0.1	127.0.0.1	TCP	68 57159 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=163		
į	58 157.630681	127.0.0.1	127.0.0.1	TCP	68 8080 → 57159 [SYN, ACK] Seq=0 Ack=1 Win=65535 Le		
į	59 157.630738	127.0.0.1	127.0.0.1	TCP	56 57159 → 8080 [ACK] Seq=1 Ack=1 Win=408256 Len=0		
	60 157.630774	127.0.0.1	127.0.0.1	TCP	56 [TCP Window Update] 8080 → 57159 [ACK] Seq=1 Ack		
(61 157.630782	127.0.0.1	127.0.0.1	HTTP	205 GET / HTTP/1.1		
(62 157.630824	127.0.0.1	127.0.0.1	TCP	56 8080 → 57159 [ACK] Seq=1 Ack=150 Win=408128 Len=		

So back to our question, Why can't you obtain an SSL certificate for your local web server from a certificate authority? Because for a certificate authority, it needs a public verifiable DNS or global address so that they can give a certificate so that we can also send requests to a trustworthy identity with certificate. However, localhost is not a trustworthy certificate because everyone computer has a localhost. This is like my name is Dixi Yao and this is a identifiable information and I can be given the certificate. But there are a lot of husband and it is impossible to verify which husband is which and represents which house so it won't be able to get SSL certificate.

Another important reason is that my localhost is not logged into the certificate authority. As professor has demoed in the lecture, if I add my localhost into the authority in my computer keychain. If I save localhost into certificate authorities trusted my browser into my operating system, it will also work.