

Q1. Internet working

For this assignment I decided to use a website that doesn't serve HTTPS by default. The reason being I wanted to see how HTTP data can be intercepted and modified. I used the website

`neverssl.com`.

1.1 DNS lookup

The first step in the process is to resolve the domain name to an IP address. This is done by querying a DNS server. The DNS server is specified in the network settings of the computer. In my case, the DNS server is `comcast.net`. The DNS server is queried using the `dig` command. The command is shown below:

```
dig neverssl.com
```

The output of the command is shown below:

```
; <<>> DiG 9.10.6 <<>> neverssl.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11543
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags::; udp: 512
;; QUESTION SECTION:
;neverssl.com.                IN      A

;; ANSWER SECTION:
neverssl.com.                60      IN      A      34.223.124.45

;; Query time: 153 msec
;; SERVER: 2001:558:feed::1#53(2001:558:feed::1)
;; WHEN: Wed Jan 17 18:48:20 EST 2024
;; MSG SIZE rcvd: 57
```

Then I did a reverse DNS lookup to find the domain name of the IP address.

```
dig -x 2001:558:feed::1
```

Therefore, the DNS server responds with the IP address of the website. In this case the IP address is for neverSSL.com is 34.223.124.45

1.2 TCP connection

Once the IP address is known, the client initiates a TCP connection with the server. I used a browser to connect to the IP address and wire shark to capture the packets. The packets are shown below:

ip.addr == 34.223.124.45						
No.	Time	Source	Destination	Protocol	Length	Info
684	10.130877	10.0.0.55	34.223.124.45	TCP	78	49680 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=14
688	10.192893	10.0.0.55	34.223.124.45	TCP	78	49681 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=14
702	10.235936	34.223.124.45	10.0.0.55	TCP	74	80 → 49680 [SYN, ACK] Seq=0 Ack=1 Win=26847 L
703	10.236010	10.0.0.55	34.223.124.45	TCP	66	49680 → 80 [ACK] Seq=1 Ack=1 Win=131712 Len=0
708	10.288276	34.223.124.45	10.0.0.55	TCP	74	80 → 49681 [SYN, ACK] Seq=0 Ack=1 Win=26847 L
709	10.288444	10.0.0.55	34.223.124.45	TCP	66	49681 → 80 [ACK] Seq=1 Ack=1 Win=131712 Len=0
710	10.288688	10.0.0.55	34.223.124.45	HTTP	430	GET / HTTP/1.1
722	10.386905	34.223.124.45	10.0.0.55	TCP	66	80 → 49681 [ACK] Seq=1 Ack=365 Win=28032 Len=
723	10.388928	34.223.124.45	10.0.0.55	TCP	1514	80 → 49681 [ACK] Seq=1 Ack=365 Win=28032 Len=
724	10.388990	10.0.0.55	34.223.124.45	TCP	66	49681 → 80 [ACK] Seq=365 Ack=1449 Win=130304
725	10.390367	34.223.124.45	10.0.0.55	HTTP	891	HTTP/1.1 200 OK (text/html)
726	10.390419	10.0.0.55	34.223.124.45	TCP	66	49681 → 80 [ACK] Seq=365 Ack=2274 Win=130240
861	13.175868	10.0.0.55	34.223.124.45	TCP	66	49681 → 80 [FIN, ACK] Seq=365 Ack=2274 Win=13
862	13.175871	10.0.0.55	34.223.124.45	TCP	66	49680 → 80 [FIN, ACK] Seq=1 Ack=1 Win=131712
865	13.271449	34.223.124.45	10.0.0.55	TCP	66	80 → 49680 [FIN, ACK] Seq=1 Ack=2 Win=26880 L
866	13.271449	34.223.124.45	10.0.0.55	TCP	66	80 → 49681 [FIN, ACK] Seq=2274 Ack=366 Win=28
867	13.271521	10.0.0.55	34.223.124.45	TCP	66	49680 → 80 [ACK] Seq=2 Ack=2 Win=131712 Len=0
868	13.271552	10.0.0.55	34.223.124.45	TCP	66	49681 → 80 [ACK] Seq=366 Ack=2275 Win=131072

1.2.1 TCP handshake

1. Source sends a SYN packet to the destination's IP address and port.
2. The source continues sending SYN packets until the destination replies with a SYN-ACK packet with the acknowledgment number set to 1.
 - It keeps sending SYN packets until it receives a SYN-ACK packet from the destination.
 - We can see that the source sends 2 SYN packets before receiving a SYN-ACK packet.
3. Source responds with an ACK packet, establishing the TCP connection.
4. Destination sends another SYN-ACK packet.
5. Source sends an ACK packet, finalizing the TCP handshake.

1.3 HTTP request

Once the TCP connection is established, the client sends an HTTP request to the server. The HTTP request is shown below:

1.2.2 HTTP request

1. Source sends an HTTP GET request to the destination, specifying the desired resource (e.g., /index.html).
2. Destination acknowledges the request with an ACK packet.

```
> Frame 710: 430 bytes on wire (3440 bits), 430 bytes captured (3440 bit
✓ Ethernet II, Src: Apple_05:9e:c3 (18:4a:53:05:9e:c3), Dst: VantivaUSA_
  > Destination: VantivaUSA_55:be:04 (5c:7d:7d:55:be:04)
  > Source: Apple_05:9e:c3 (18:4a:53:05:9e:c3)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 10.0.0.55, Dst: 34.223.124.45
> Transmission Control Protocol, Src Port: 49681, Dst Port: 80, Seq: 1,
✓ Hypertext Transfer Protocol
  ✓ GET / HTTP/1.1\r\n
    ✓ [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
      [GET / HTTP/1.1\r\n]
      [Severity level: Chat]
      [Group: Sequence]
      Request Method: GET
      Request URI: /
      Request Version: HTTP/1.1
      Host: 34.223.124.45\r\n
      Upgrade-Insecure-Requests: 1\r\n
      Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
      User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit
      Accept-Language: en-US,en;q=0.9\r\n
      Accept-Encoding: gzip, deflate\r\n
      Connection: keep-alive\r\n
      \r\n
      [Full request URI: http://34.223.124.45/]
      [HTTP request 1/1]
      [Response in frame: 725]
```

1.2.3 HTTP response

1. Destination sends an HTTP response with the requested resource.
2. Source acknowledges the response with an ACK packet

```

> Frame 725: 891 bytes on wire (7128 bits), 891 bytes captured (7128 bit
✓ Ethernet II, Src: VantivaUSA_55:be:04 (5c:7d:7d:55:be:04), Dst: Apple_
  > Destination: Apple_05:9e:c3 (18:4a:53:05:9e:c3)
  > Source: VantivaUSA_55:be:04 (5c:7d:7d:55:be:04)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 34.223.124.45, Dst: 10.0.0.55
> Transmission Control Protocol, Src Port: 80, Dst Port: 49681, Seq: 144
> [2 Reassembled TCP Segments (2273 bytes): #723(1448), #725(825)]
✓ Hypertext Transfer Protocol
  ✓ HTTP/1.1 200 OK\r\n
    ✓ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      [HTTP/1.1 200 OK\r\n]
      [Severity level: Chat]
      [Group: Sequence]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
      Date: Wed, 17 Jan 2024 23:20:50 GMT\r\n
      Server: Apache/2.4.58 ( )\r\n
      Upgrade: h2,h2c\r\n
      Connection: Upgrade, Keep-Alive\r\n
      Last-Modified: Wed, 29 Jun 2022 00:23:33 GMT\r\n
      ETag: "f79-5e28b29d38e93-gzip"\r\n
      Accept-Ranges: bytes\r\n
      Vary: Accept-Encoding\r\n
      Content-Encoding: gzip\r\n
    > Content-Length: 1900\r\n
      Keep-Alive: timeout=5, max=100\r\n
      Content-Type: text/html; charset=UTF-8\r\n
      \r\n
      [HTTP response 1/1]
      [Time since request: 0.101679000 seconds]
      \[Request in frame: 710\]
      [Request URI: http://34.223.124.45/]
      Content-encoded entity body (gzip): 1900 bytes -> 3961 bytes

```

3. A snapshot of the HTTP sent by the destination is shown below:

```
\n
<h1 id="status"></h1>\n
<script>document.querySelector("#status").textContent = "Connecting ...";</script>\n
<noscript>\n
\n
<t<h2>What?</h2>\n
<t<p>This website is for when you try to open Facebook, Google, Amazon, etc\n
on a wifi network, and nothing happens. Type "http://neverssl.com"\n
into your browser's url bar, and you'll be able to log on.</p>\n
\n
<t<h2>How?</h2>\n
<t<p>neverssl.com will never use SSL (also known as TLS). No\n
encryption, no strong authentication, no <a\n
href="https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security">HSTS</a>,\n
no HTTP/2.0, just plain old unencrypted HTTP and forever stuck in the dark\n
tages of internet security.</p>\n
\n
<t<h2>Why?</h2>\n
<t<p>Normally, that's a bad idea. You should always use SSL and secure\n
encryption when possible. In fact, it's such a bad idea that most websites\n
tare now using https by default.</p>\n
\n
<t<p>And that's great, but it also means that if you're relying on\n
tpoorly-behaved wifi networks, it can be hard to get online. Secure\n
trowsers and websites using https make it impossible for those wifi\n
tnetworks to send you to a login or payment page. Basically, those networks\n
tcan't tap into your connection just like attackers can't. Modern browsers\n
tare so good that they can remember when a website supports encryption and\n
tven if you type in the website name, they'll use https.</p>\n
\n
<t<p>And if the network never redirects you to this page, well as you can\n
tsee, you're not missing much.</p>\n
\n
<a href="https://twitter.com/neverssl">Follow @neverssl</a>\n
\n
```

1.4 TCP connection termination

Once the HTTP response is received, the client terminates the TCP connection. The packets are shown below:

861	13.175868	10.0.0.55	34.223.124.45	TCP	66	49681 → 80	[FIN, ACK] Seq=365 Ack=2274 Win=131072 Len=0 TSval=159802
862	13.175871	10.0.0.55	34.223.124.45	TCP	66	49680 → 80	[FIN, ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=1345685995
865	13.271449	34.223.124.45	10.0.0.55	TCP	66	80 → 49680	[FIN, ACK] Seq=1 Ack=2 Win=26880 Len=0 TSval=199406680 TS
866	13.271449	34.223.124.45	10.0.0.55	TCP	66	80 → 49681	[FIN, ACK] Seq=2274 Ack=366 Win=28032 Len=0 TSval=1994066
867	13.271521	10.0.0.55	34.223.124.45	TCP	66	49680 → 80	[ACK] Seq=2 Ack=2 Win=131712 Len=0 TSval=1345686091 TS
868	13.271552	10.0.0.55	34.223.124.45	TCP	66	49681 → 80	[ACK] Seq=366 Ack=2275 Win=131072 Len=0 TSval=1598023305

1.3.1 TCP connection termination

1. Source sends a FIN, ACK packet to the destination.
2. Destination acknowledges the FIN, ACK packet with an ACK packet.