CS378 Lab-6 Report

Saksham Rathi (22B1003)
Department of Computer Science, IIT Bombay

Part 1: Investigating HTTP Traffic with Wireshark

Part (a) Both client and server are running version 1.1 of HTTP.
Part (b) The IP Address of the client machine is 192.168.1.102 and the IP Address of the server machine is 128.119.245.12
Part (c) The status code returned by the server to the client browser is 200 (OK).
Part (d) The HTML file which the client is retrieving last modified at the server is "/ethereal-labs/lab2-1.html". The last modified time is "Tue, 23 Sep 2003 05:29:00 GMT". (From follow → HTTP)
Part (e) The number of bytes in the server's response is 439 bytes.

Part 2: Basic Authentication over HTTP

Part (a) The response of the server to the first get request is "Authorization Required". The status code is 401. The server's response of "Authorization Required" with a status code of 401 indicates that the client (your application or browser) is trying to access a resource that requires authentication, but no valid credentials were provided.
Part (b) This request contains the authorisation field too:
"Authorization: Basic ZXRoLXN0dWRlbnRzOm5ldHdvcmtz".
$< credentials >$ (the content after Basic) is the Base64 encoding of ID and password joined by a single colon :.
Part (c) The username is "eth-students" and the password is "network". The username and password are separated by a colon and then encoded in Base64. (Decoded using a python script)

Part 3: Analysing and Decrypting TLS Traffic

Public-key cryptography provides a method for secure communication between two parties over the Internet, even if they have never interacted before. It works on the principle of using a pair of keys: a public key, which can be shared openly, and a private key, which is kept secret by its owner. When one party wants to send a secure message to the other, they encrypt the message using the recipient's public key. Since the public key is publicly available, anyone can use it to encrypt messages, but only the recipient can decrypt them with their private key.
Part (a) The cipher common between both SSL End points is "ECDHE-RSA-AES256-GCM-SHA384"
Part (b) The protocol which was used for communication is "TLSv1.2".

Part (c) The master key is
"9F9A0F19A02BDDBE1A05926597D622CCA06D2AF416A28AD9C03163B87
FF1B0C67824BBDB595B32D8027DB566EC04FB25".

Part 4: Capturing Your Own Traffic

The last modified time of the website is: "Wed, 29 Jun 2022 00:23:22 GMT". Here is the screenshot displaying the same:

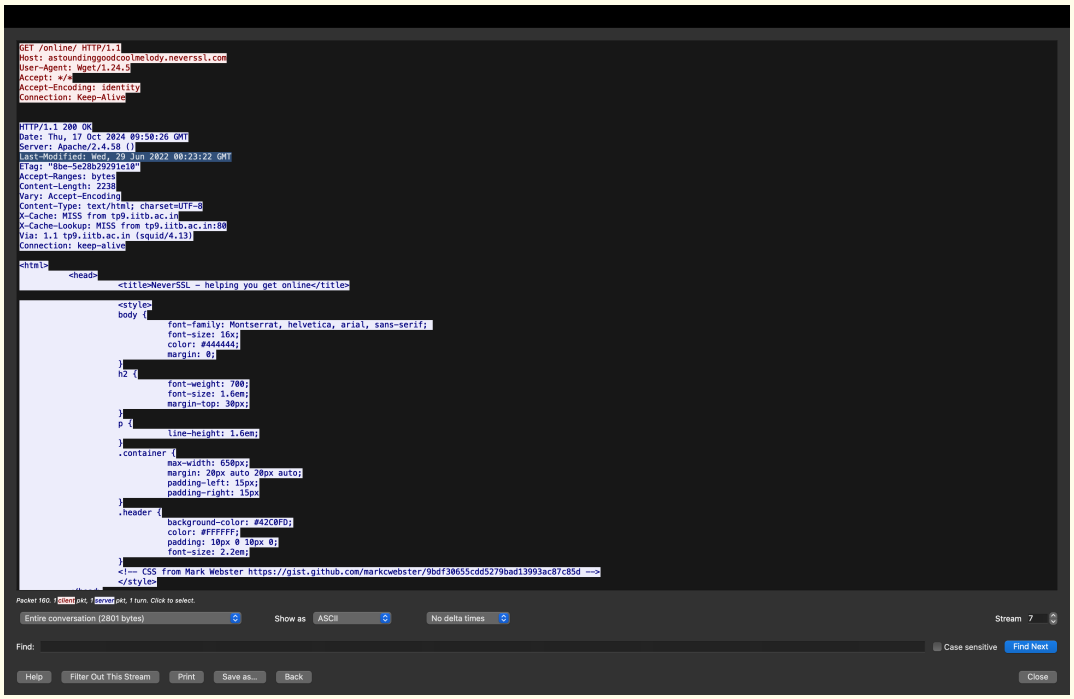

Figure 1: Last Modified Time of the Website

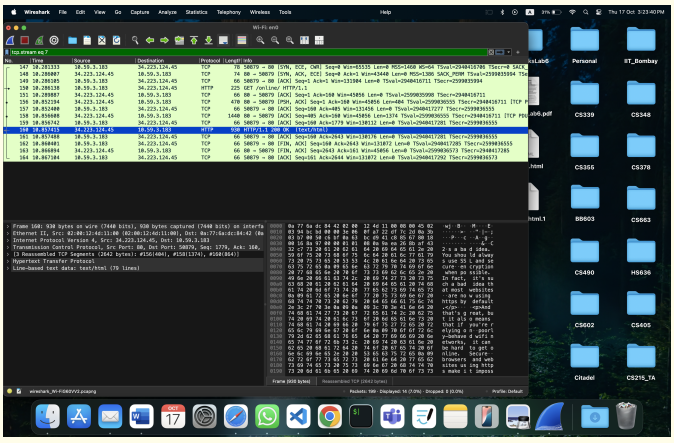

Figure 2: The packet which was followed

Then I used Wireshark for loopback interface too. Here is the screenshot of data which was sent from sender.c:

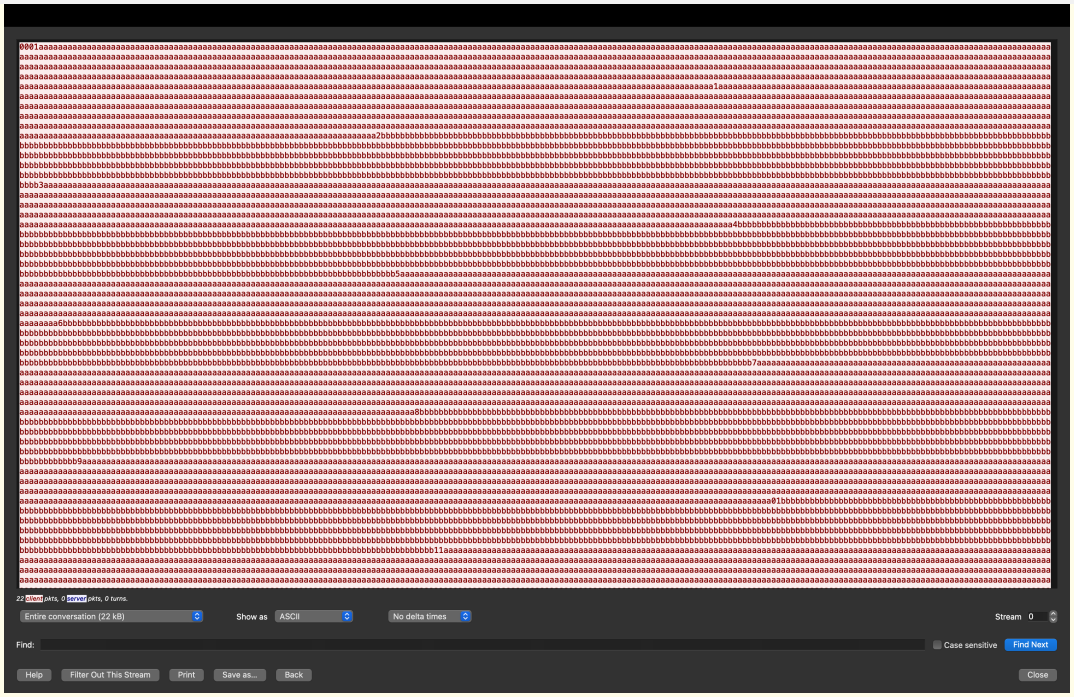

Figure 3: Data sent from sender.c

The characters ‘a’ represent the first packet and characters ‘b’ represent the second packet of the pair. The first numbers represent the packet size (the first packet). Then the numbers in between are the packet numbers. All of this was visible from the follow UDP Stream on Wireshark.

Part 5: Using Scapy for Packet Manipulation

In the first part, I have sent a simple ICMP packet containing the message ”Hello Saksham” to my loopback IP Address. The packet was sent successfully and the response was received too. Here is the screenshot of the same:

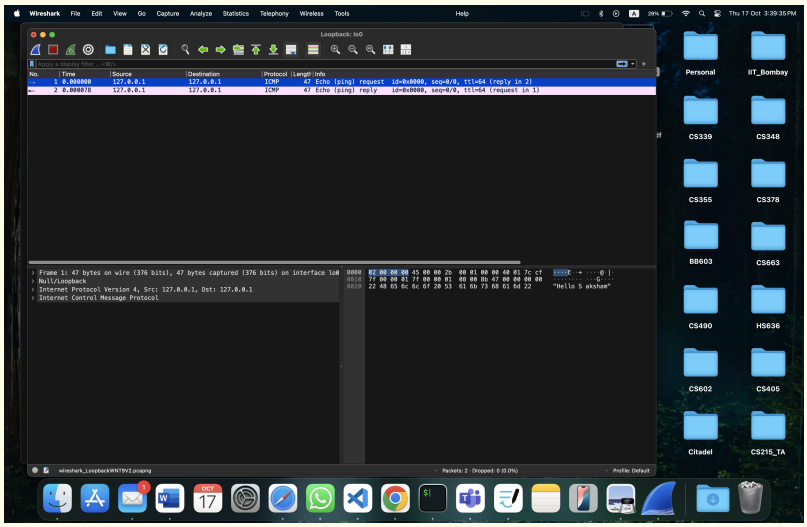

Figure 4: Sending ICMP Packet using Scapy

The lower right corner also shows the message which was sent.

In the second part, we need to request a non-standard DNS record type and send to a DNS Server. This python script expects the server IP Address (I did to google.com whose IP Address

is 8.8.8.8) and the domain name. Here is the output from the script:

```
Enter the DNS server IP address (e.g., 8.8.8.8): 8.8.8.8
Enter the domain name to query (e.g., example.com): google.com
Begin emission
...
Finished sending 1 packets
*
Received 4 packets, got 1 answers, remaining 0 packets
Received DNS response from 8.8.8.8:
IP / UDP / DNS Ans b'ns3.google.com.'
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 128
  id        = 41225
  flags     = DF
  frag      = 0
  ttl       = 61
  proto     = udp
  chksum    = 0x7e62
  src       = 8.8.8.8
  dst       = 10.59.3.183
  \options   \
###[ UDP ]###
     sport     = domain
     dport     = domain
     len       = 108
     chksum    = 0x5878
###[ DNS ]###
        id        = 0
        qr        = 1
        opcode    = QUERY
        aa        = 0
        tc        = 0
        rd        = 1
        ra        = 1
        z         = 0
        ad        = 0
        cd        = 0
        rcode     = ok
        qdcount   = 1
        ancount   = 4
        nscount   = 0
        arcount   = 0
        \qd        \
         |###[ DNS Question Record ]###
         |  qname     = b'google.com.'
         |  qtype     = ALL
         |  unicastresponse= 0
         |  qclass    = IN
```

```
\an         \
 |###[ DNS Resource Record ]###
 |  rrname    = b'google.com.'
 |  type      = NS
 |  cacheflush= 0
 |  rclass    = IN
 |  ttl       = 345589
 |  rdlen     = None
 |  rdata     = b'ns3.google.com.'
 |###[ DNS Resource Record ]###
 |  rrname    = b'google.com.'
 |  type      = NS
 |  cacheflush= 0
 |  rclass    = IN
 |  ttl       = 345589
 |  rdlen     = None
 |  rdata     = b'ns2.google.com.'
 |###[ DNS Resource Record ]###
 |  rrname    = b'google.com.'
 |  type      = NS
 |  cacheflush= 0
 |  rclass    = IN
 |  ttl       = 345589
 |  rdlen     = None
 |  rdata     = b'ns1.google.com.'
 |###[ DNS Resource Record ]###
 |  rrname    = b'google.com.'
 |  type      = NS
 |  cacheflush= 0
 |  rclass    = IN
 |  ttl       = 345589
 |  rdlen     = None
 |  rdata     = b'ns4.google.com.'
\ns         \
\ar         \
```

Here is the screenshot from Wireshark:

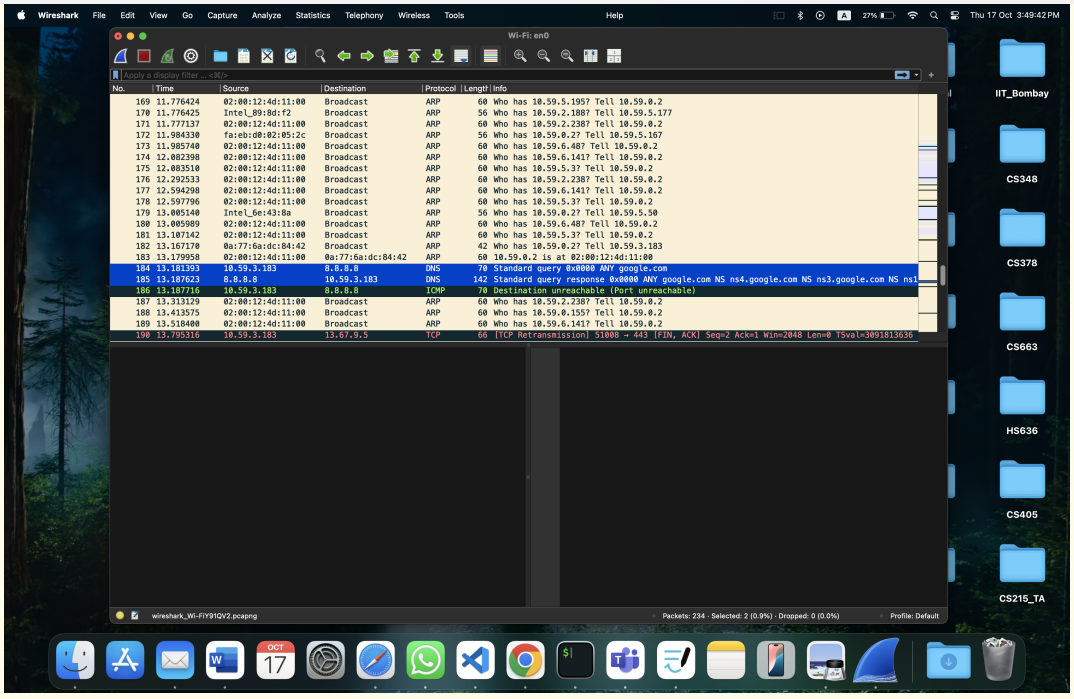

Figure 5: Wireshark Output for DNS Query

The selected packets are the ones which were sent.

In the third part, we were supposed to create a TCP SYN packet with a spoofed source IP Address.

Here is the screenshot of the same:

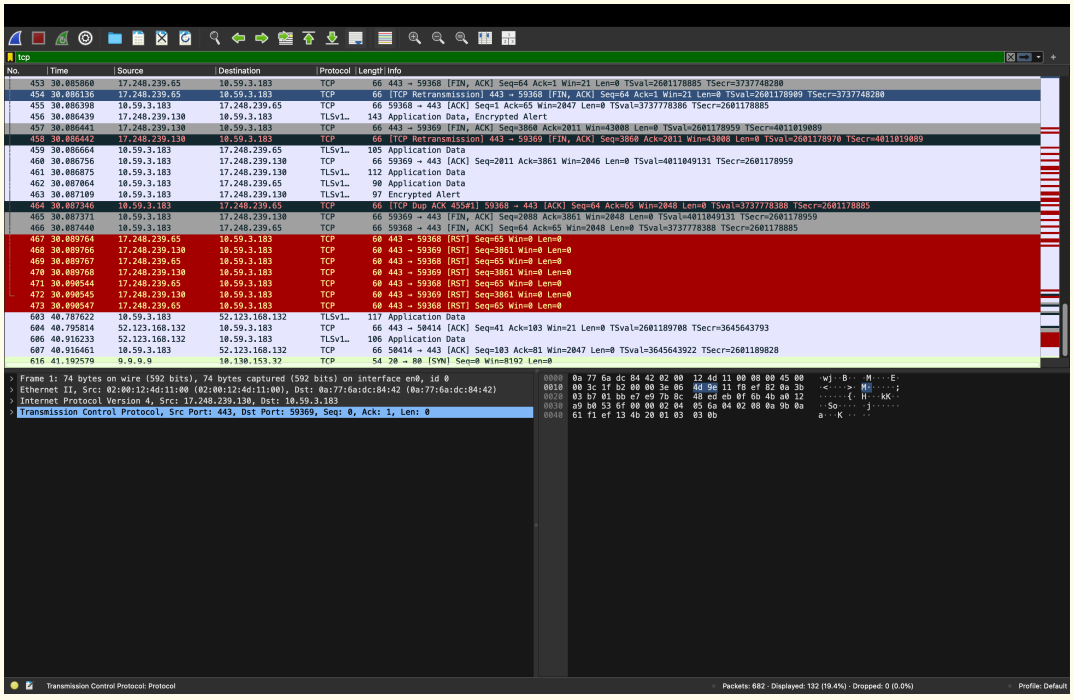

Figure 6: Sending TCP SYN Packet with Spoofed Source IP Address

I spoofed my IP to 9.9.9.9 (line number 616 in the image shown).