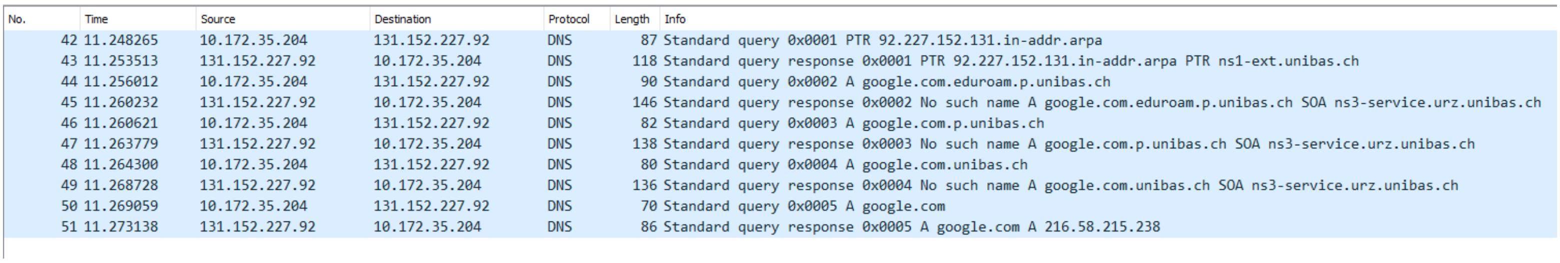
FS 23
Jannick Heisch

Benedikt Heuser
Maria Desteffani

Sheet 2

1) DNS



```
No.   Time       Source          Destination     Protocol Length Info
   42 11.248265  10.172.35.204   131.152.227.92  DNS        87 Standard query 0x0001 PTR 92.227.152.131.in-addr.arpa
   43 11.253513  131.152.227.92  10.172.35.204   DNS       118 Standard query response 0x0001 PTR 92.227.152.131.in-addr.arpa PTR ns1-ext.unibas.ch
   44 11.256012  10.172.35.204   131.152.227.92  DNS        90 Standard query 0x0002 A google.com.eduroam.p.unibas.ch
   45 11.260232  131.152.227.92  10.172.35.204   DNS       146 Standard query response 0x0002 No such name A google.com.eduroam.p.unibas.ch SOA ns3-service.urz.unibas.ch
   46 11.260621  10.172.35.204   131.152.227.92  DNS        82 Standard query 0x0003 A google.com.p.unibas.ch
   47 11.263779  131.152.227.92  10.172.35.204   DNS       138 Standard query response 0x0003 No such name A google.com.p.unibas.ch SOA ns3-service.urz.unibas.ch
   48 11.264300  10.172.35.204   131.152.227.92  DNS        80 Standard query 0x0004 A google.com.unibas.ch
   49 11.268728  131.152.227.92  10.172.35.204   DNS       136 Standard query response 0x0004 No such name A google.com.unibas.ch SOA ns3-service.urz.unibas.ch
   50 11.269059  10.172.35.204   131.152.227.92  DNS        70 Standard query 0x0005 A google.com
   51 11.273138  131.152.227.92  10.172.35.204   DNS        86 Standard query response 0x0005 A google.com A 216.58.215.238
```

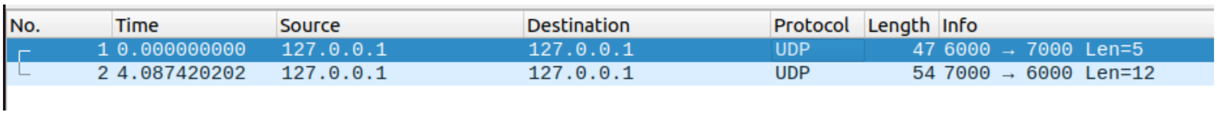
First, we do a reverse DNS lookup, which means that we want to get a domain name from an IP-address. Here we're asking for the name of the DNS server we're communicating with.

Next a query goes out to look up Google's IP. Since the server at google.com.eduoram.p.unibas.ch doesn't know the IP, we go one level up to google.com.p.unibas.ch. When this one doesn't know either we go further up and so forth. Finally, we reach the DNS server of google.com which can give us the IP we need.

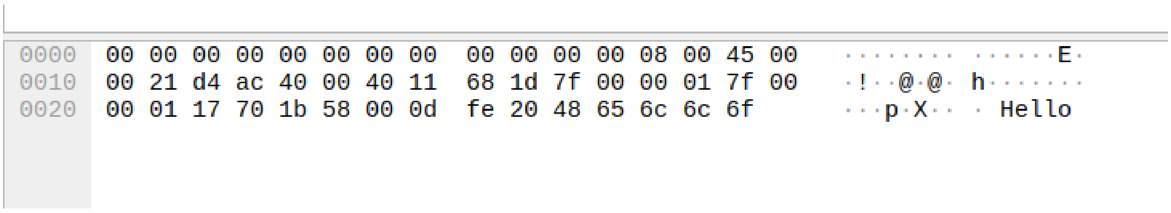
2) Comparison TCP, UDP and UDP Multicast

2a) UDP & TCP

UDP:

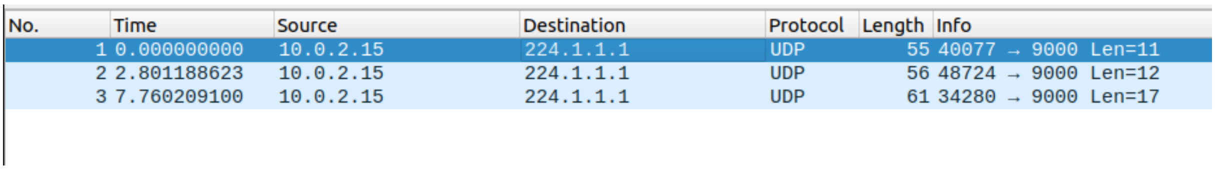


```
No.  Time         Source      Destination  Protocol  Length Info
   1 0.000000000  127.0.0.1   127.0.0.1    UDP          47 6000 → 7000 Len=5
   2 4.087420202  127.0.0.1   127.0.0.1    UDP          54 7000 → 6000 Len=12
```

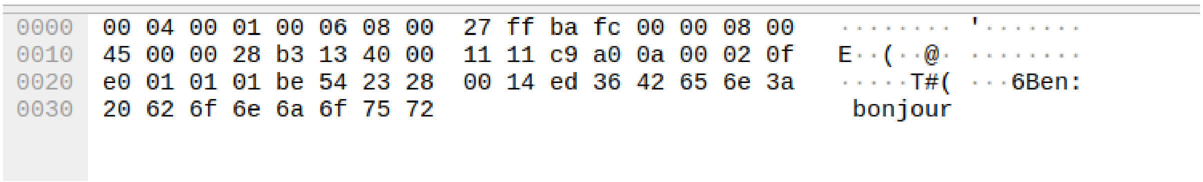


```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 21 d4 ac 40 00 40 11  68 1d 7f 00 00 01 7f 00   ·!··@·@· h·······
0020  00 01 17 70 1b 58 00 0d  fe 20 48 65 6c 6c 6f      ···p·X·· · Hello
```

UDP Multicast:

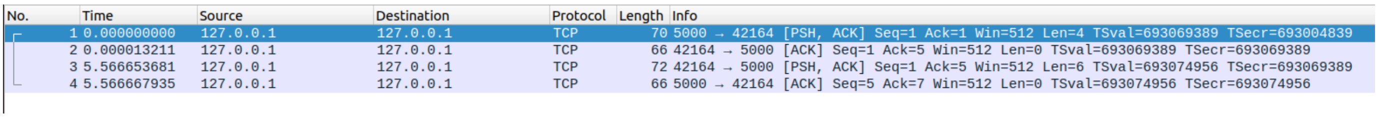


```
No.  Time         Source     Destination  Protocol  Length Info
   1 0.000000000  10.0.2.15  224.1.1.1    UDP          55 40077 → 9000 Len=11
   2 2.801188623  10.0.2.15  224.1.1.1    UDP          56 48724 → 9000 Len=12
   3 7.760209100  10.0.2.15  224.1.1.1    UDP          61 34280 → 9000 Len=17
```

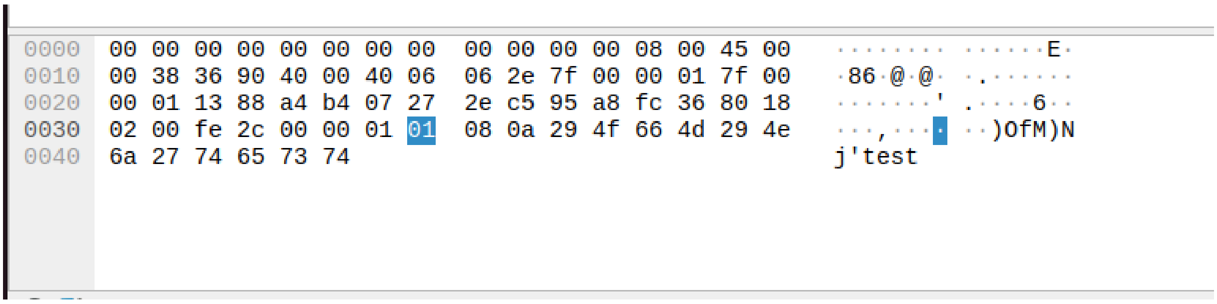


```
0000  00 04 00 01 00 06 08 00  27 ff ba fc 00 00 08 00   ········ '·······
0010  45 00 00 28 b3 13 40 00  11 11 c9 a0 0a 00 02 0f   E··(··@· ········
0020  e0 01 01 01 be 54 23 28  00 14 ed 36 42 65 6e 3a   ·····T#( ···6Ben:
0030  20 62 6f 6e 6a 6f 75 72                             bonjour
```

TCP:



```
No.  Time         Source     Destination  Protocol  Length Info
   1 0.000000000  127.0.0.1  127.0.0.1    TCP          70 5000 → 42164 [PSH, ACK] Seq=1 Ack=1 Win=512 Len=4 TSval=693069389 TSecr=693004839
   2 0.000013211  127.0.0.1  127.0.0.1    TCP          66 42164 → 5000 [ACK] Seq=1 Ack=5 Win=512 Len=0 TSval=693069389 TSecr=693069389
   3 5.566653681  127.0.0.1  127.0.0.1    TCP          72 42164 → 5000 [PSH, ACK] Seq=1 Ack=5 Win=512 Len=6 TSval=693074956 TSecr=693069389
   4 5.566667935  127.0.0.1  127.0.0.1    TCP          66 5000 → 42164 [ACK] Seq=5 Ack=7 Win=512 Len=0 TSval=693074956 TSecr=693074956
```



```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 38 36 90 40 00 40 06  06 2e 7f 00 00 01 7f 00   ·86·@·@· ·.······
0020  00 01 13 88 a4 b4 07 27  2e c5 95 a8 fc 36 80 18   ·······' .····6··
0030  02 00 fe 2c 00 00 01 01  08 0a 29 4f 66 4d 29 4e   ···,···· ··)OfM)N
0040  6a 27 74 65 73 74                                  j'test
```

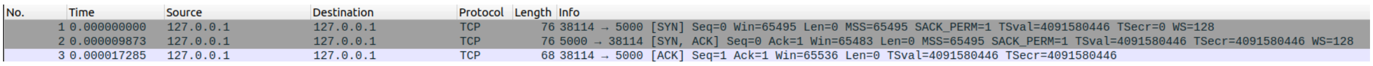
When two clients are talking the source and destination are both set to localhost, regardless of whether we use TCP or UDP. If we use UDP multicast our source is 10.0.2.15 which is a private IP used for local networks and our destination is the IP of the multicast room the client has entered (here: 224.1.1.1). The UDP messages don't get an answer to sent packages while the TCP communication always sends and answer with an ACK in it back. All three versions let us see the sent message inside the packets, nothing is encrypted.

UDP and TCP packages are on the network layer of the of the internet protocol stack. The data itself is on the application layer.

2b) Advantages & Disadvantages

UDP has the advantage that there are less packets being sent, it "looks cleaner" and causes half the amount of traffic TCP causes. TCP meanwhile has the clear advantage in that there are less packets lost and we know when packets don't reach their destination due to the response the receiver sends back.

2c) TCP Handshake

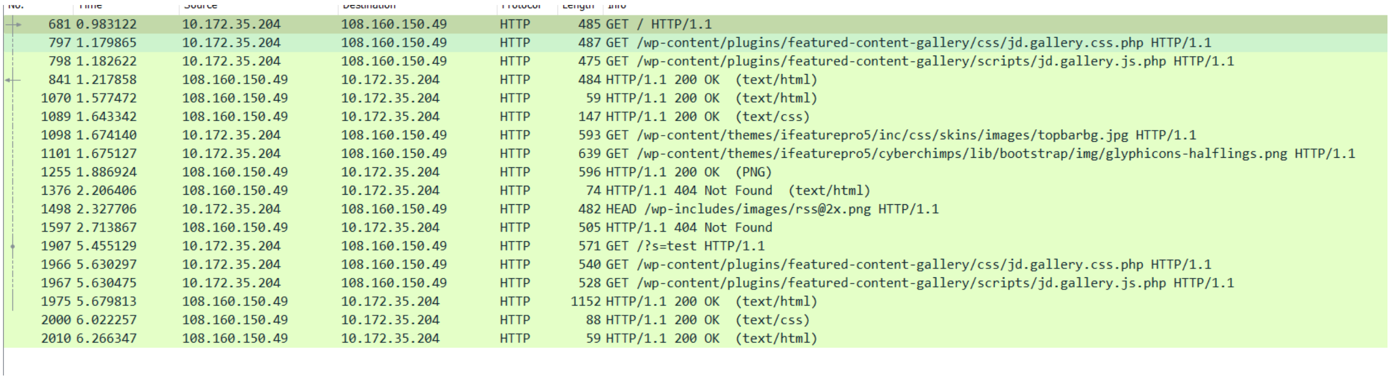


```
No.   Time          Source      Destination  Protocol Length Info
    1 0.000000000   127.0.0.1   127.0.0.1    TCP          76 38114 → 5000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=4091580446 TSecr=0 WS=128
    2 0.000009873   127.0.0.1   127.0.0.1    TCP          76 5000 → 38114 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=4091580446 TSecr=4091580446 WS=128
    3 0.000017285   127.0.0.1   127.0.0.1    TCP          68 38114 → 5000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=4091580446 TSecr=4091580446
```

Here we have a client connecting to a server, both have the same IP here since it's run on localhost. First, the client sends the server a message having set the SYN flag on which means it wants to connect to the server. Next, the server answers and sets the SYN and ACK flag on, he agrees to connect. Lastly, the client sends a message including a set ACK flag as well to acknowledge that he understood. This three-way handshake causes way less problems than a two-way handshake, for this reason the last message with the ACK flag is sent by the client after the server has already agreed to connect.

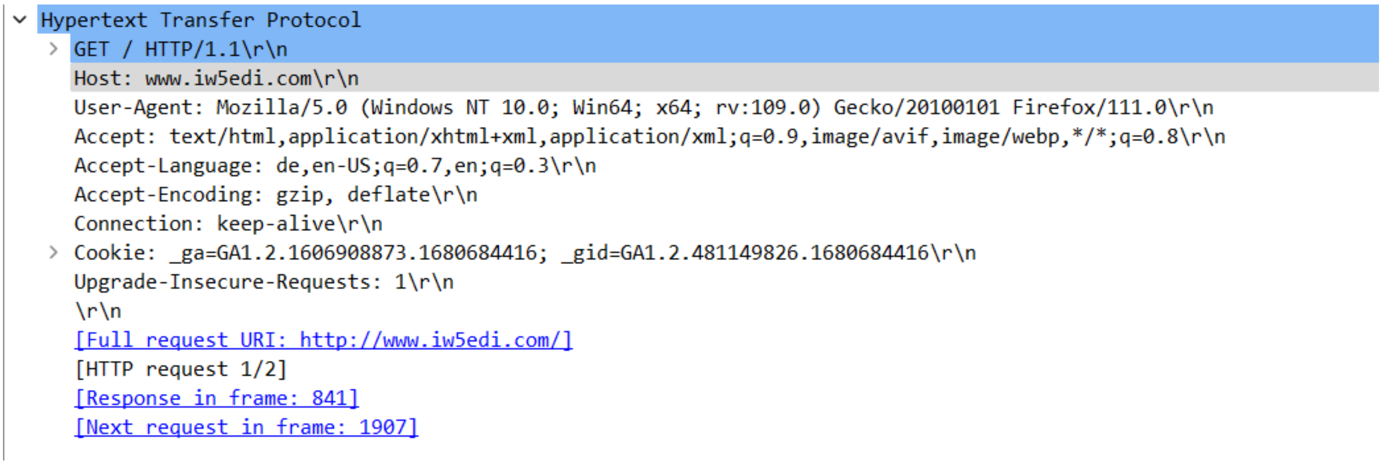
3) HTTP vs. HTTPS

3a) HTTP



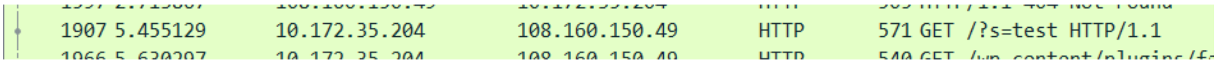
```
No.   Time      Source          Destination     Protocol Length Info
  681 0.983122  10.172.35.204   108.160.150.49  HTTP      485 GET / HTTP/1.1
  797 1.179865  10.172.35.204   108.160.150.49  HTTP      487 GET /wp-content/plugins/featured-content-gallery/css/jd.gallery.css.php HTTP/1.1
  798 1.182622  10.172.35.204   108.160.150.49  HTTP      475 GET /wp-content/plugins/featured-content-gallery/scripts/jd.gallery.js.php HTTP/1.1
  841 1.217858  108.160.150.49  10.172.35.204   HTTP      484 HTTP/1.1 200 OK  (text/html)
 1070 1.577472  108.160.150.49  10.172.35.204   HTTP       59 HTTP/1.1 200 OK  (text/html)
 1089 1.643342  108.160.150.49  10.172.35.204   HTTP      147 HTTP/1.1 200 OK  (text/css)
 1098 1.674140  10.172.35.204   108.160.150.49  HTTP      593 GET /wp-content/themes/ifeaturepro5/inc/css/skins/images/topbarbg.jpg HTTP/1.1
 1101 1.675127  10.172.35.204   108.160.150.49  HTTP      639 GET /wp-content/themes/ifeaturepro5/cyberchimps/lib/bootstrap/img/glyphicons-halflings.png HTTP/1.1
 1255 1.886924  108.160.150.49  10.172.35.204   HTTP      596 HTTP/1.1 200 OK  (PNG)
 1376 2.206406  108.160.150.49  10.172.35.204   HTTP       74 HTTP/1.1 404 Not Found  (text/html)
 1498 2.327706  10.172.35.204   108.160.150.49  HTTP      482 HEAD /wp-includes/images/rss@2x.png HTTP/1.1
 1597 2.713867  108.160.150.49  10.172.35.204   HTTP      505 HTTP/1.1 404 Not Found
 1907 5.455129  10.172.35.204   108.160.150.49  HTTP      571 GET /?s=test HTTP/1.1
 1966 5.630297  10.172.35.204   108.160.150.49  HTTP      540 GET /wp-content/plugins/featured-content-gallery/css/jd.gallery.css.php HTTP/1.1
 1967 5.630475  10.172.35.204   108.160.150.49  HTTP      528 GET /wp-content/plugins/featured-content-gallery/scripts/jd.gallery.js.php HTTP/1.1
 1975 5.679813  108.160.150.49  10.172.35.204   HTTP     1152 HTTP/1.1 200 OK  (text/html)
 2000 6.022257  108.160.150.49  10.172.35.204   HTTP       88 HTTP/1.1 200 OK  (text/css)
 2010 6.266347  108.160.150.49  10.172.35.204   HTTP       59 HTTP/1.1 200 OK  (text/html)
```

The http request for the website is easily trackable with wireshark. We just have to filter for the IP of the web-server and that we want HTTP packets. We can see the /GET requests for the html and the answers from the server.



```
v Hypertext Transfer Protocol
  > GET / HTTP/1.1\r\n
    Host: www.iw5edi.com\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/111.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\n
    Accept-Language: de,en-US;q=0.7,en;q=0.3\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
  > Cookie: _ga=GA1.2.1606908873.1680684416; _gid=GA1.2.481149826.1680684416\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: http://www.iw5edi.com/]
    [HTTP request 1/2]
    [Response in frame: 841]
    [Next request in frame: 1907]
```

If we dive further and open the first sent package we can look under "Hypertext transfer Protocol" and can see things like the used browser (firefox), that we use windows, which languages we accept (german and english) and so forth.

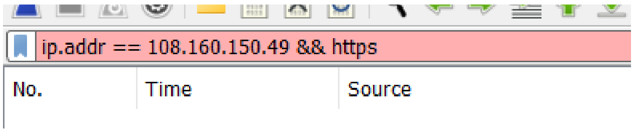


```
1907 5.455129   10.172.35.204   108.160.150.49   HTTP   571 GET /?s=test HTTP/1.1
```

We searched "test" on the website and as we can see our search request shows up in the package info for everyone to see.

3b) HTTPS

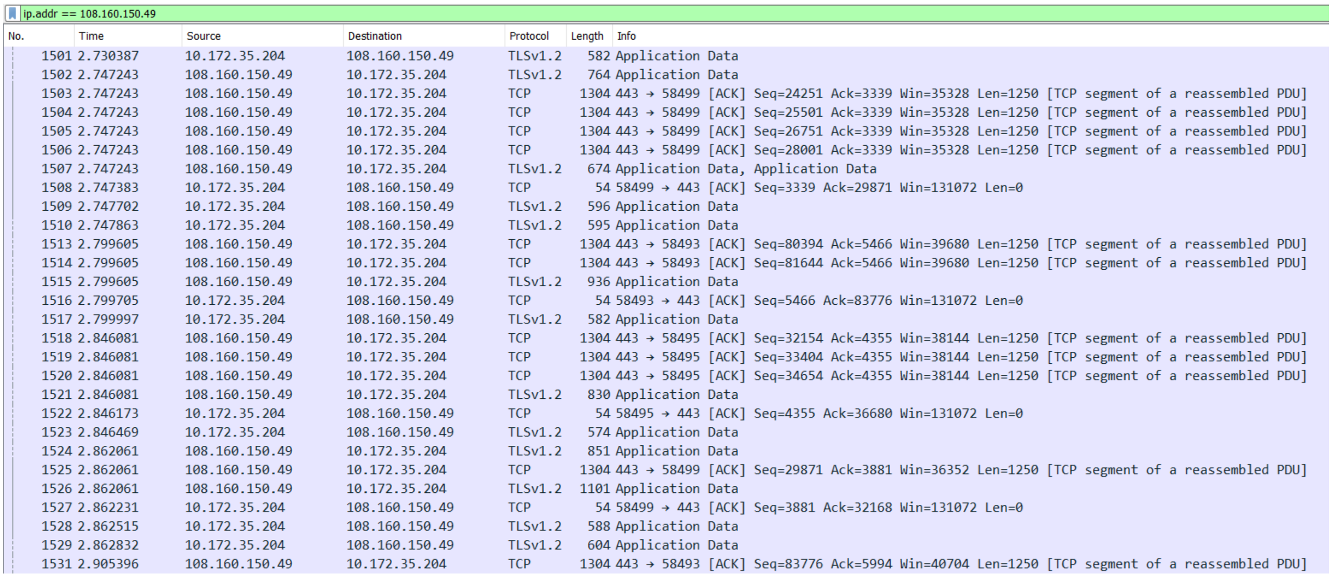
We can no longer just filter wireshark for the web-server IP and HTTPS since nothing will show up.



```
ip.addr == 108.160.150.49 && https

No.      Time       Source
```

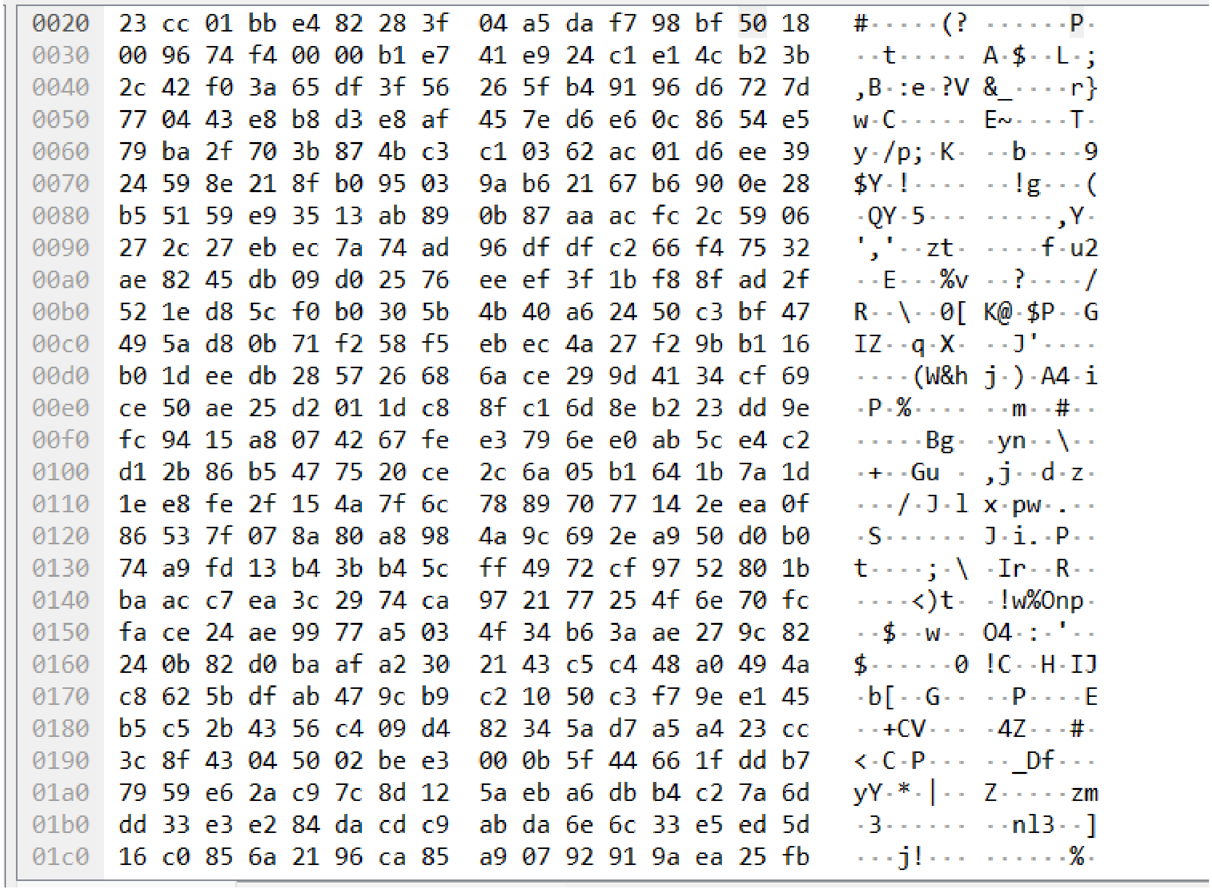
If we filter for the IP address we get something like this:



```
ip.addr == 108.160.150.49

No.     Time      Source           Destination      Protocol  Length Info
   1501 2.730387  10.172.35.204    108.160.150.49   TLSv1.2    582 Application Data
   1502 2.747243  108.160.150.49   10.172.35.204    TLSv1.2    764 Application Data
   1503 2.747243  108.160.150.49   10.172.35.204    TCP       1304 443 → 58499 [ACK] Seq=24251 Ack=3339 Win=35328 Len=1250 [TCP segment of a reassembled PDU]
   1504 2.747243  108.160.150.49   10.172.35.204    TCP       1304 443 → 58499 [ACK] Seq=25501 Ack=3339 Win=35328 Len=1250 [TCP segment of a reassembled PDU]
   1505 2.747243  108.160.150.49   10.172.35.204    TCP       1304 443 → 58499 [ACK] Seq=26751 Ack=3339 Win=35328 Len=1250 [TCP segment of a reassembled PDU]
   1506 2.747243  108.160.150.49   10.172.35.204    TCP       1304 443 → 58499 [ACK] Seq=28001 Ack=3339 Win=35328 Len=1250 [TCP segment of a reassembled PDU]
   1507 2.747243  108.160.150.49   10.172.35.204    TLSv1.2    674 Application Data, Application Data
   1508 2.747383  10.172.35.204    108.160.150.49   TCP         54 58499 → 443 [ACK] Seq=3339 Ack=29871 Win=131072 Len=0
   1509 2.747702  10.172.35.204    108.160.150.49   TLSv1.2    596 Application Data
   1510 2.747863  10.172.35.204    108.160.150.49   TLSv1.2    595 Application Data
   1513 2.799605  108.160.150.49   10.172.35.204    TCP       1304 443 → 58493 [ACK] Seq=80394 Ack=5466 Win=39680 Len=1250 [TCP segment of a reassembled PDU]
   1514 2.799605  108.160.150.49   10.172.35.204    TCP       1304 443 → 58493 [ACK] Seq=81644 Ack=5466 Win=39680 Len=1250 [TCP segment of a reassembled PDU]
   1515 2.799605  108.160.150.49   10.172.35.204    TLSv1.2    936 Application Data
   1516 2.799705  10.172.35.204    108.160.150.49   TCP         54 58493 → 443 [ACK] Seq=5466 Ack=83776 Win=131072 Len=0
   1517 2.799997  10.172.35.204    108.160.150.49   TLSv1.2    582 Application Data
   1518 2.846081  108.160.150.49   10.172.35.204    TCP       1304 443 → 58495 [ACK] Seq=32154 Ack=4355 Win=38144 Len=1250 [TCP segment of a reassembled PDU]
   1519 2.846081  108.160.150.49   10.172.35.204    TCP       1304 443 → 58495 [ACK] Seq=33404 Ack=4355 Win=38144 Len=1250 [TCP segment of a reassembled PDU]
   1520 2.846081  108.160.150.49   10.172.35.204    TCP       1304 443 → 58495 [ACK] Seq=34654 Ack=4355 Win=38144 Len=1250 [TCP segment of a reassembled PDU]
   1521 2.846081  108.160.150.49   10.172.35.204    TLSv1.2    830 Application Data
   1522 2.846173  10.172.35.204    108.160.150.49   TCP         54 58495 → 443 [ACK] Seq=4355 Ack=36680 Win=131072 Len=0
   1523 2.846469  10.172.35.204    108.160.150.49   TLSv1.2    574 Application Data
   1524 2.862061  108.160.150.49   10.172.35.204    TLSv1.2    851 Application Data
   1525 2.862061  108.160.150.49   10.172.35.204    TCP       1304 443 → 58499 [ACK] Seq=29871 Ack=3881 Win=36352 Len=1250 [TCP segment of a reassembled PDU]
   1526 2.862061  108.160.150.49   10.172.35.204    TLSv1.2   1101 Application Data
   1527 2.862231  10.172.35.204    108.160.150.49   TCP         54 58499 → 443 [ACK] Seq=3881 Ack=32168 Win=131072 Len=0
   1528 2.862515  10.172.35.204    108.160.150.49   TLSv1.2    588 Application Data
   1529 2.862832  10.172.35.204    108.160.150.49   TLSv1.2    604 Application Data
   1531 2.905396  108.160.150.49   10.172.35.204    TCP       1304 443 → 58493 [ACK] Seq=83776 Ack=5994 Win=40704 Len=1250 [TCP segment of a reassembled PDU]
```

Our messages now get encrypted with TLS. While we can look at TLS packets we will only see encrypted data



```
0020  23 cc 01 bb e4 82 28 3f  04 a5 da f7 98 bf 50 18   #·····(? ······P·
0030  00 96 74 f4 00 00 b1 e7  41 e9 24 c1 e1 4c b2 3b   ··t····· A·$··L·;
0040  2c 42 f0 3a 65 df 3f 56  26 5f b4 91 96 d6 72 7d   ,B·:e·?V &_····r}
0050  77 04 43 e8 b8 d3 e8 af  45 7e d6 e6 0c 86 54 e5   w·C····· E~····T·
0060  79 ba 2f 70 3b 87 4b c3  c1 03 62 ac 01 d6 ee 39   y·/p;·K· ··b····9
0070  24 59 8e 21 8f b0 95 03  9a b6 21 67 b6 90 0e 28   $Y·!···· ··!g···(
0080  b5 51 59 e9 35 13 ab 89  0b 87 aa ac fc 2c 59 06   ·QY·5··· ·····,Y·
0090  27 2c 27 eb ec 7a 74 ad  96 df df c2 66 f4 75 32   ','··zt· ····f·u2
00a0  ae 82 45 db 09 d0 25 76  ee ef 3f 1b f8 8f ad 2f   ··E···%v ··?····/
00b0  52 1e d8 5c f0 b0 30 5b  4b 40 a6 24 50 c3 bf 47   R··\··0[ K@·$P··G
00c0  49 5a d8 0b 71 f2 58 f5  eb ec 4a 27 f2 9b b1 16   IZ··q·X· ··J'····
00d0  b0 1d ee db 28 57 26 68  6a ce 29 9d 41 34 cf 69   ····(W&h j·)·A4·i
00e0  ce 50 ae 25 d2 01 1d c8  8f c1 6d 8e b2 23 dd 9e   ·P·%···· ··m··#··
00f0  fc 94 15 a8 07 42 67 fe  e3 79 6e e0 ab 5c e4 c2   ·····Bg· ·yn··\··
0100  d1 2b 86 b5 47 75 20 ce  2c 6a 05 b1 64 1b 7a 1d   ·+··Gu · ,j··d·z·
0110  1e e8 fe 2f 15 4a 7f 6c  78 89 70 77 14 2e ea 0f   ···/·J·l x·pw·.··
0120  86 53 7f 07 8a 80 a8 98  4a 9c 69 2e a9 50 d0 b0   ·S······ J·i.·P··
0130  74 a9 fd 13 b4 3b b4 5c  ff 49 72 cf 97 52 80 1b   t····;·\ ·Ir··R··
0140  ba ac c7 ea 3c 29 74 ca  97 21 77 25 4f 6e 70 fc   ····<)t· ·!w%Onp·
0150  fa ce 24 ae 99 77 a5 03  4f 34 b6 3a ae 27 9c 82   ··$··w·· O4·:·'··
0160  24 0b 82 d0 ba af a2 30  21 43 c5 c4 48 a0 49 4a   $······0 !C··H·IJ
0170  c8 62 5b df ab 47 9c b9  c2 10 50 c3 f7 9e e1 45   ·b[··G·· ··P····E
0180  b5 c5 2b 43 56 c4 09 d4  82 34 5a d7 a5 a4 23 cc   ··+CV··· ·4Z···#·
0190  3c 8f 43 04 50 02 be e3  00 0b 5f 44 66 1f dd b7   <·C·P··· ··_Df···
01a0  79 59 e6 2a c9 7c 8d 12  5a eb a6 db b4 c2 7a 6d   yY·*·|·· Z·····zm
01b0  dd 33 e3 e2 84 da cd c9  ab da 6e 6c 33 e5 ed 5d   ·3······ ··nl3··]
01c0  16 c0 85 6a 21 96 ca 85  a9 07 92 91 9a ea 25 fb   ···j!··· ······%·
```

The reasons to use HTTPS are pretty obvious from these findings. Not only can you see who is communicating with who but also things like what browser was used for requests and even what exactly was looked up on certain websites. Using HTTPS will solve those things by encrypting sent messages.