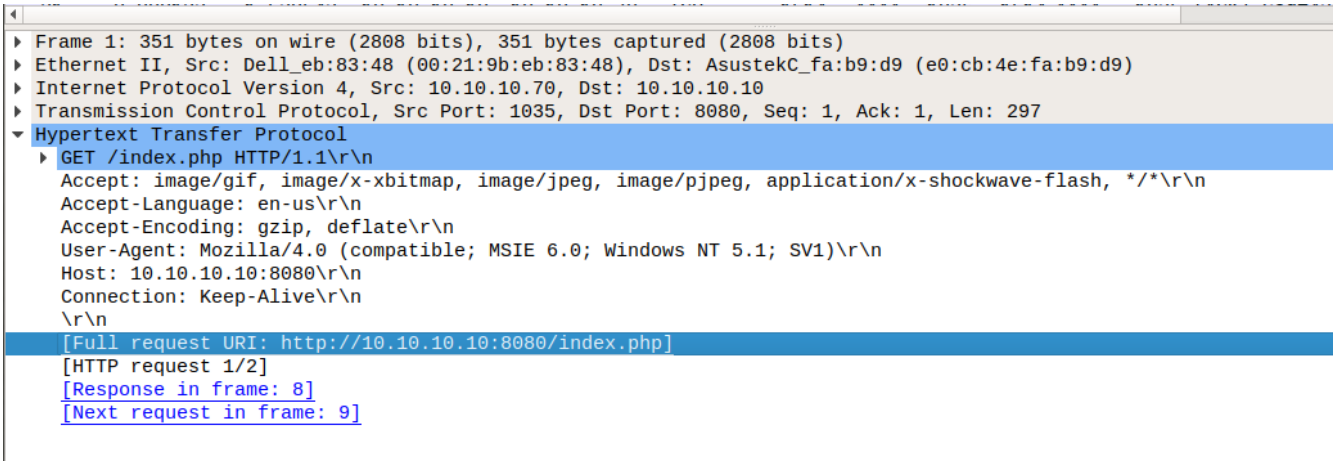
Ann's Aurora – An Advanced Persistent Threat based challenge by SANS.

Walkthrough by Andrew Mayer

This is a classic to me, one of the first network forensics CTFs I did and a very good one in my opinion. Coming back to it for a writeup proved it to be just as fun as it was back then, albeit a bit easier!

1.What was the full URI of Vick Timmes' original web request? (Please include the port in your URI.)

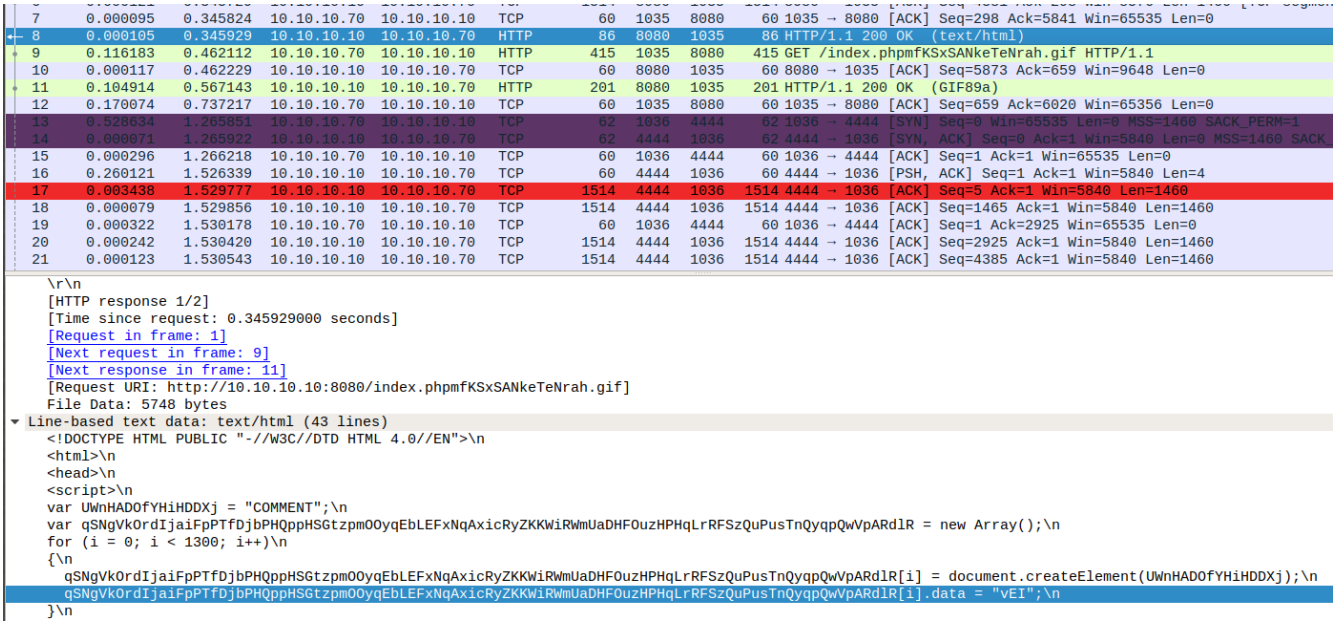
For this scenario, I had to look for http traffic at the beginning of the capture. I payed attention to the info field for a GET request, signifying that the client computer, which I quickly found to be 10.10.10.70, was requesting a page from the server.. The initial request was actually the first packet sent, requesting 10.10.10.10:8080/index.php.



2.In response, the malicious web server sent back obfuscated JavaScript. Near the beginning of this code, the attacker created an array with 1300 elements labeled

"COMMENT", then filled their data element with a string. What was the value of this string?

Here, I just needed to find the response to the GET request, an HTTP OK, and check for the code In question. As such, it was found by finding the <script> tag. In here, it just required some basic javascript reading and the field of the data element revealed itself as vEI.



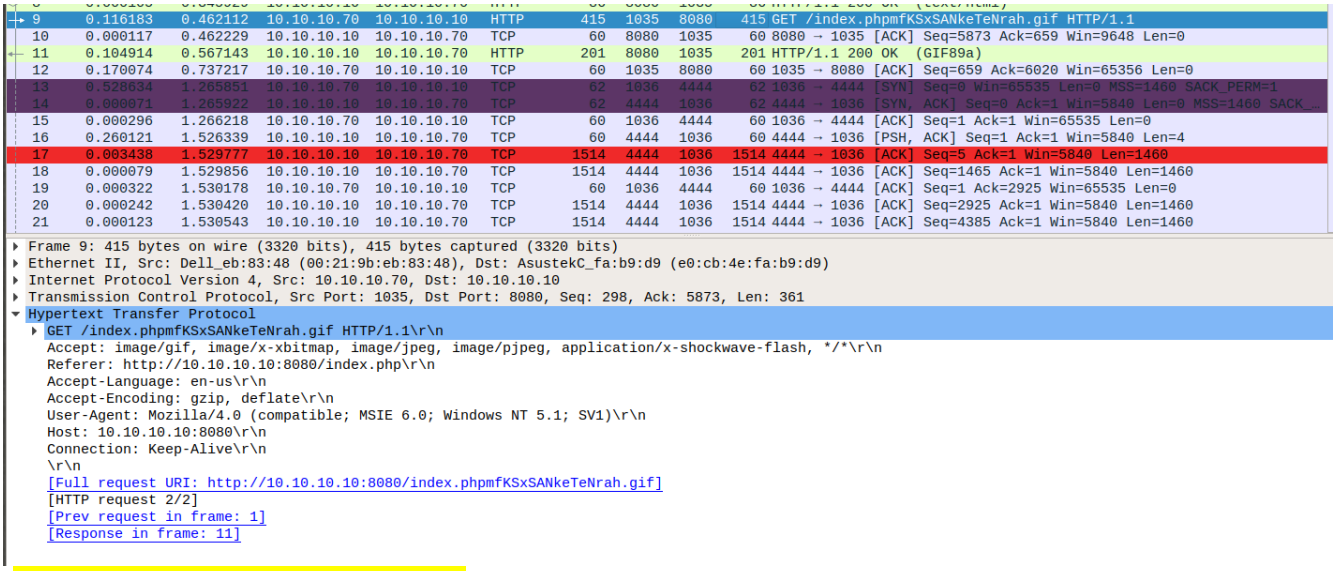
7	0.000095	0.345824	10.10.10.70	10.10.10.10	TCP	60	1035	8080	60	1035 → 8080 [ACK] Seq=298 Ack=5841 Win=65535 Len=0
8	0.000105	0.345929	10.10.10.10	10.10.10.70	HTTP	86	8080	1035	86	HTTP/1.1 200 OK (text/html)
9	0.116183	0.462112	10.10.10.70	10.10.10.10	HTTP	415	1035	8080	415	GET /index.phpmfKSxSANkeTeNrah.gif HTTP/1.1
10	0.000117	0.462229	10.10.10.10	10.10.10.70	TCP	60	8080	1035	60	8080 → 1035 [ACK] Seq=5873 Ack=659 Win=9648 Len=0
11	0.104914	0.567143	10.10.10.10	10.10.10.70	HTTP	201	8080	1035	201	HTTP/1.1 200 OK (GIF89a)
12	0.170074	0.737217	10.10.10.70	10.10.10.10	TCP	60	1035	8080	60	1035 → 8080 [ACK] Seq=659 Ack=6020 Win=65356 Len=0
13	0.528634	1.265851	10.10.10.70	10.10.10.10	TCP	62	1036	4444	62	1036 → 4444 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
14	0.000071	1.265922	10.10.10.10	10.10.10.70	TCP	62	4444	1036	62	4444 → 1036 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK
15	0.000296	1.266218	10.10.10.70	10.10.10.10	TCP	60	1036	4444	60	1036 → 4444 [ACK] Seq=1 Ack=1 Win=65535 Len=0
16	0.260121	1.526339	10.10.10.10	10.10.10.70	TCP	60	4444	1036	60	4444 → 1036 [PSH, ACK] Seq=1 Ack=1 Win=5840 Len=4
17	0.003438	1.529777	10.10.10.10	10.10.10.70	TCP	1514	4444	1036	1514	4444 → 1036 [ACK] Seq=5 Ack=1 Win=5840 Len=1460
18	0.000079	1.529856	10.10.10.10	10.10.10.70	TCP	1514	4444	1036	1514	4444 → 1036 [ACK] Seq=1465 Ack=1 Win=5840 Len=1460
19	0.000322	1.530178	10.10.10.70	10.10.10.10	TCP	60	1036	4444	60	1036 → 4444 [ACK] Seq=1 Ack=2925 Win=65535 Len=0
20	0.000242	1.530420	10.10.10.10	10.10.10.70	TCP	1514	4444	1036	1514	4444 → 1036 [ACK] Seq=2925 Ack=1 Win=5840 Len=1460
21	0.000123	1.530543	10.10.10.10	10.10.10.70	TCP	1514	4444	1036	1514	4444 → 1036 [ACK] Seq=4385 Ack=1 Win=5840 Len=1460

```
\r\n
[HTTP response 1/2]
[Time since request: 0.345929000 seconds]
[Request in frame: 1]
[Next request in frame: 9]
[Next response in frame: 11]
[Request URI: http://10.10.10.10:8080/index.phpmfKSxSANkeTeNrah.gif]
File Data: 5748 bytes
Line-based text data: text/html (43 lines)
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">\n
<html>\n
<head>\n
<script>\n
var UWnHADOfYHiHDDXj = "COMMENT";\n
var qSNgVkOrdIjaiFpPTfDjbPHQppHSGtzpmOOyqEbLEFxNqAxicRyZKKWiRWmUaDHFOuzHPHqLrRFSzQuPusTnQyqpQwVpARdlR = new Array();\n
for (i = 0; i < 1300; i++)\n
{\n
  qSNgVkOrdIjaiFpPTfDjbPHQppHSGtzpmOOyqEbLEFxNqAxicRyZKKWiRWmUaDHFOuzHPHqLrRFSzQuPusTnQyqpQwVpARdlR[i] = document.createElement(UWnHADOfYHiHDDXj);\n
  qSNgVkOrdIjaiFpPTfDjbPHQppHSGtzpmOOyqEbLEFxNqAxicRyZKKWiRWmUaDHFOuzHPHqLrRFSzQuPusTnQyqpQwVpARdlR[i].data = "vEI";\n
}\n
```

3.Vick's computer made a second HTTP request for an object.

a.What was the filename of the object that was requested?

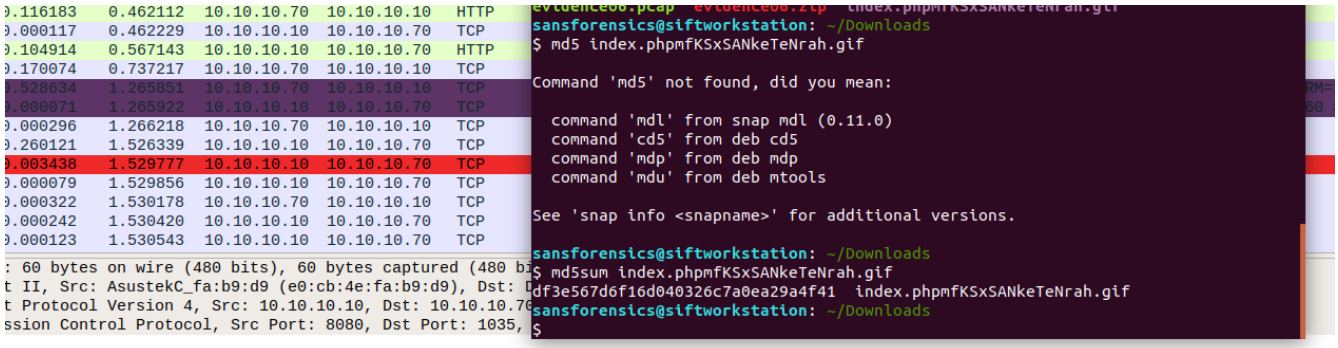
So here, I just needed to find the second http request going from .70 to .10 server, as was said in the question. A quick look at the http section of the packet revealed the answer to be a gif,



index.phpmfKSxSANkeTeNrah.gif

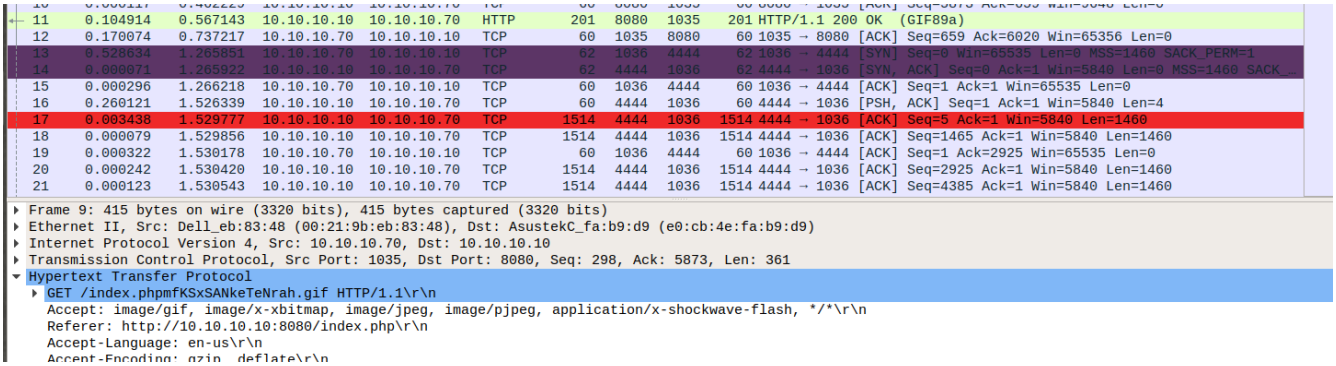
b.What is the MD5sum of the object that was returned?

Here, since I knew it was an HTML object and the name of it, I just had to go over to wiresharks built in feature to export objects, hit html, and hit the specific file. Then it's as easy as terminal and md5sum command, showing df3e567d6f16d040326c7a0ea29a4f41.



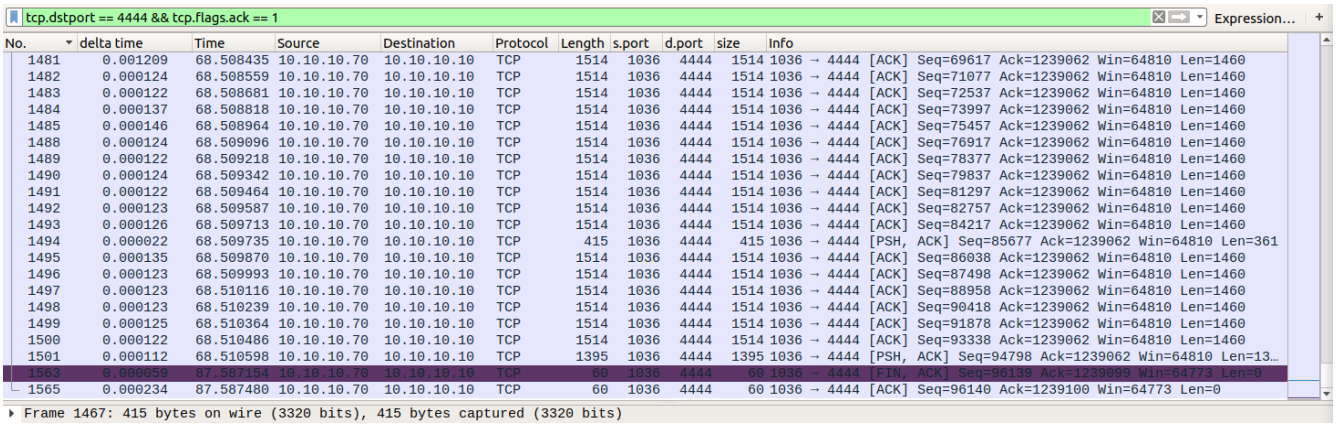
4.When was the TCP session on port 4444 opened? (Provide the number of seconds since the beginning of the packet capture, rounded to tenths of a second. ie, 49.5 seconds)

Here, I actually have a coloring rule that helps out quite a bit. I have a specific color set up for when a tcp handshake happens, so I am able to easily see where any and all sessions like this are open. Again, within the first couple packets, the answer is found. You can see in the dark purple background rows,the time is 1.3 seconds.



5.When was the TCP session on port 4444 closed? (Provide the number of seconds since the beginning of the packet capture, rounded to tenths of a second. ie, 49.5 seconds)

As I did not want to scroll until finding the correct bit of dark purple(as I used in the last question), I simply put in a filter for “tcp.dstport == 4444 && tcp.flags.ack =- 1” to filter out the ack that would be sent back to the server upon the fin/ack sequence finishing up. As such, the answer came out to be 87.6 seconds.



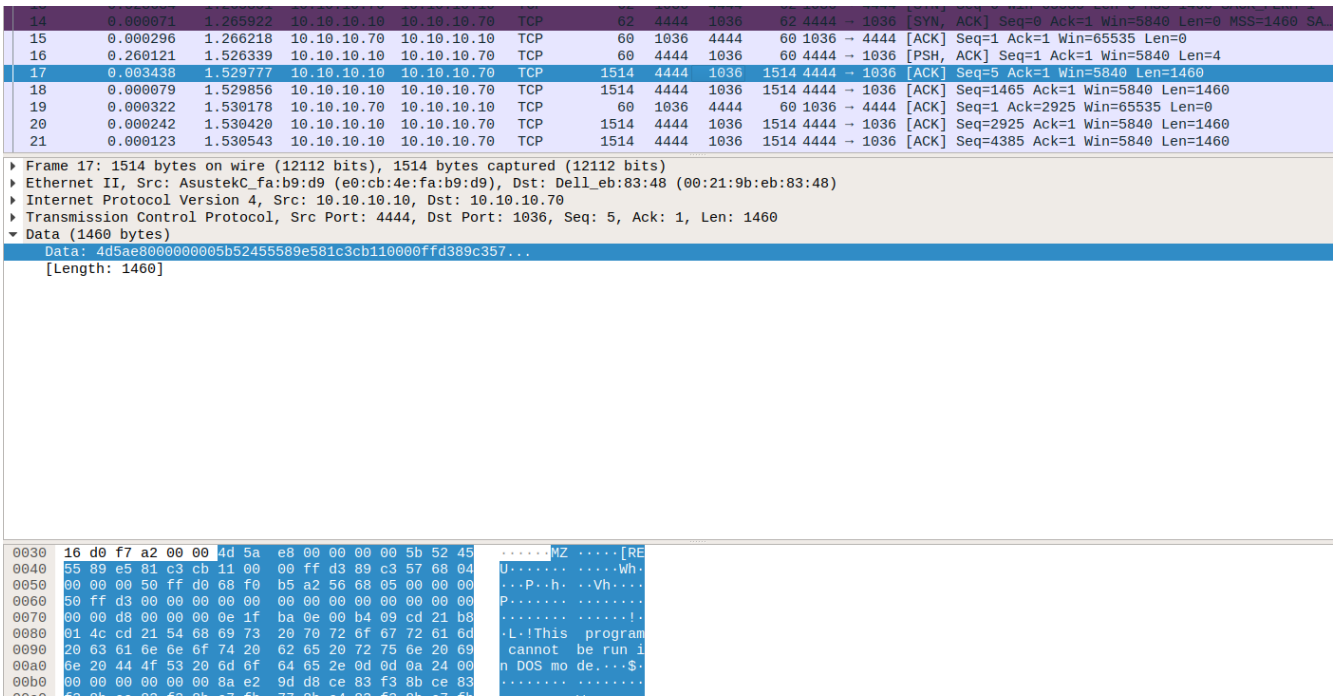
6.In packet 17, the malicious server sent a file to the client.

c.What type of file was it? Choose one:

- **Windows executable**
- **GIF image**
- **PHP script**
- **Zip file**
- **Encrypted data**

d.What was the MD5sum of the file?

This was a windows exeuctable file, as shown by the 4d5a header in the data field of packet 17. Again, my coloring rules came in very helpful, as I have one set to bright red when it sees the words “DOS mode” in the data field



7.Vick's computer repeatedly tried to connect back to the malicious server on port 4445, even after the original connection on port 4444 was closed. With respect to these repeated failed connection attempts:

a.How often does the TCP initial sequence number (ISN) change? (Choose one.)

- **Every packet**
- **Every third packet**
- **Every 10-15 seconds**
- **Every 30-35 seconds**
- **Every 60 seconds**

b.How often does the IP ID change? (Choose one.)

- **Every packet**
- **Every third packet**
- **Every 10-15 seconds**

•Every 30-35 seconds

•Every 60 seconds

c.How often does the source port change? (Choose one.)

•Every packet

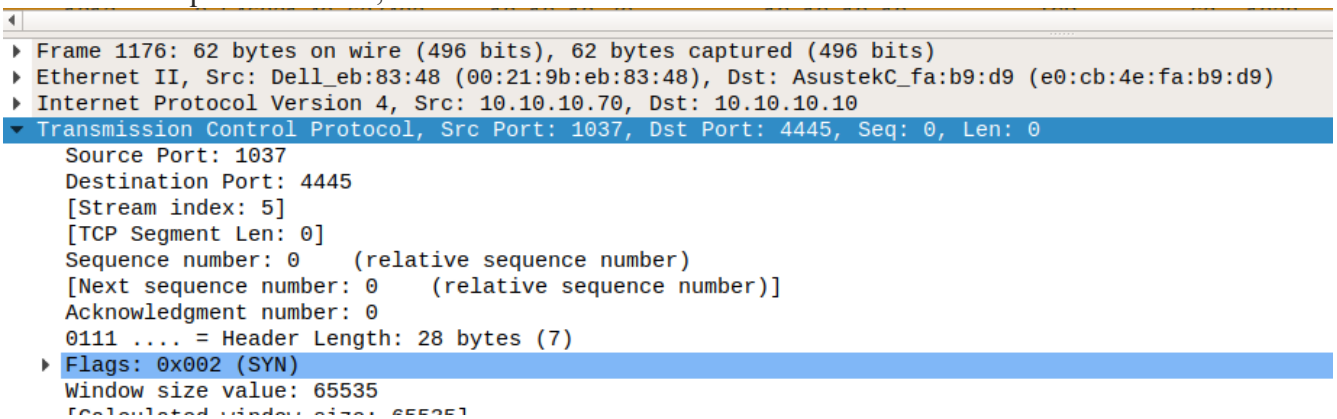
•Every third packet

•Every 10-15 seconds

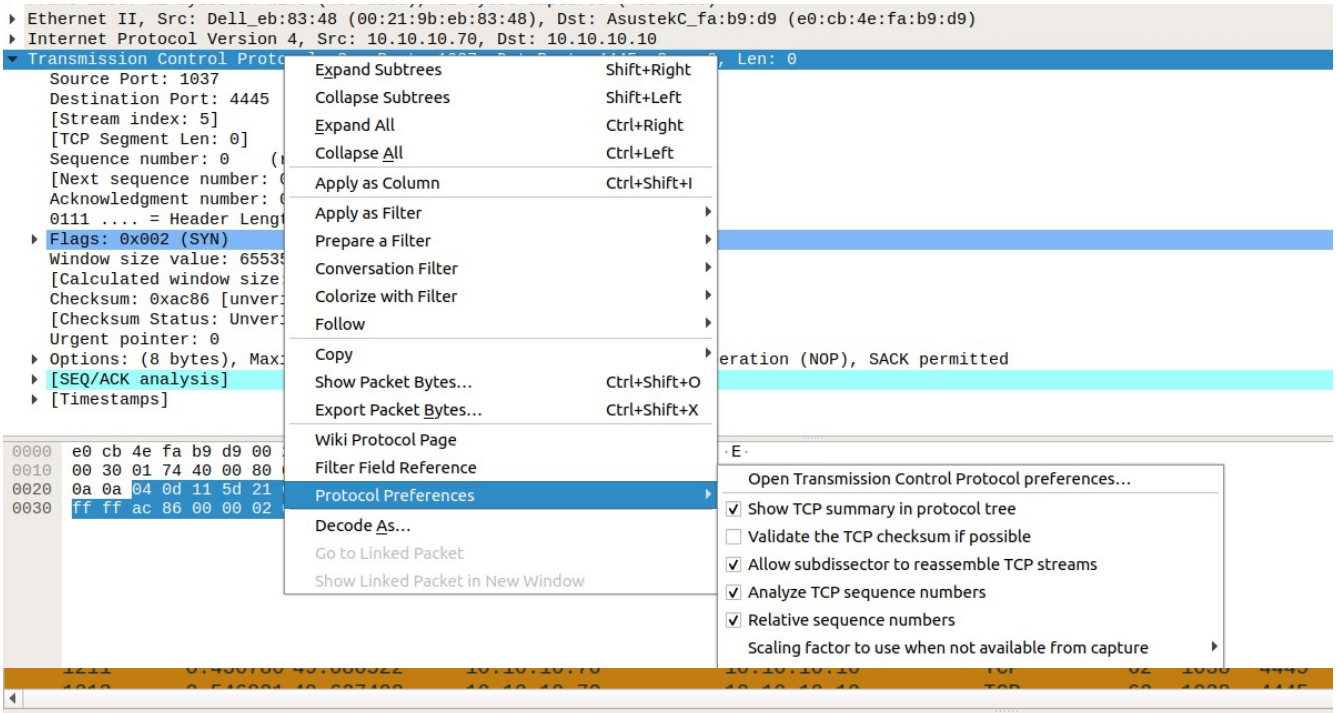
•Every 30-35 seconds

•Every 60 seconds

There was a few different areas in this question, ill go through them one by one as I worked them. The ISN number is the initial number for a new tcp connection when it is formed. Sequence numbers are used to make sure packets arrive in order, or are able to be reordered by the receiver if it is not *too many* that are out of order. It is able to keep track of this order with the sequence number! Following the ISN, the sequence number of each subsequent packet will go up by the amount of data contained in the packet. Wireshark has a neat trick by default that will basically zero out ISN number, and name it the relative sequence number, as seen below:

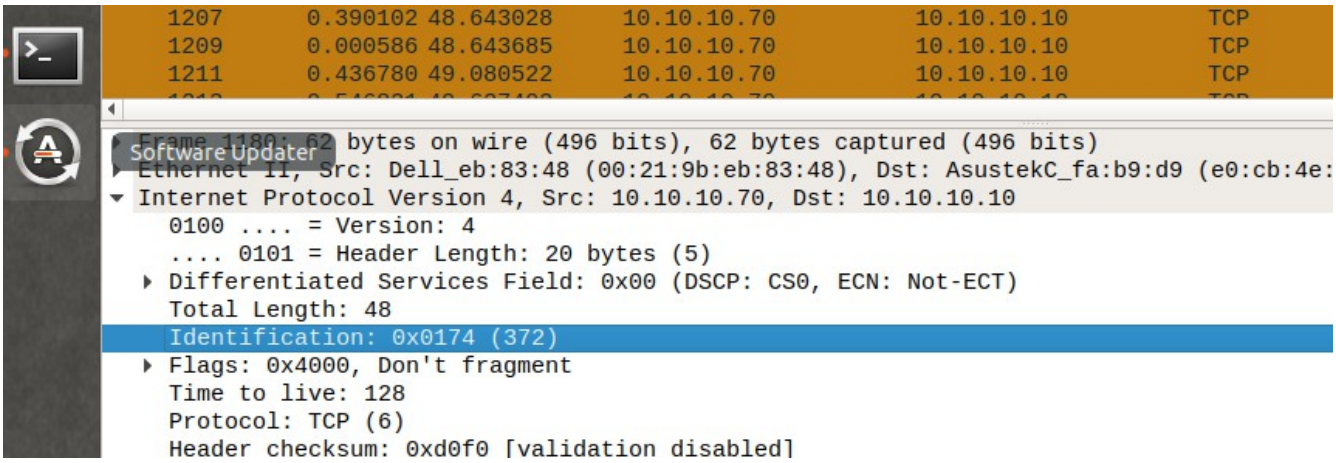


so the two ways to see it changing are to watch the bytes themselves in the lower part of the wireshar pane, or to turn off the feature that zeros out the ISN. As seen below, by right clicking the tcp header and hitting protocol preferences, it gives the option, and then, as you see below that, the ISNs are shown in their true numbers, and show to change every 3rd packet:




```
Frame 1163: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
Ethernet II, Src: Dell_eb:83:48 (00:21:9b:eb:83:48), Dst: AsustekC_fa:b9:d9 (e0:cb:4e:fa:b9:d9)
Internet Protocol Version 4, Src: 10.10.10.70, Dst: 10.10.10.10
Transmission Control Protocol, Src Port: 1037, Dst Port: 4445, Seq: 553800369, Len: 0
    Source Port: 1037
    Destination Port: 4445
    [Stream index: 3]
    [TCP Segment Len: 0]
    Sequence number: 553800369
    [Next sequence number: 553800369]
    Acknowledgment number: 0
    0111 .... = Header Length: 28 bytes (7)
  ▸ Flags: 0x002 (SYN)
    Window size value: 65535
    [Calculated window size: 65535]
    Checksum: 0xd19e [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (8 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
```

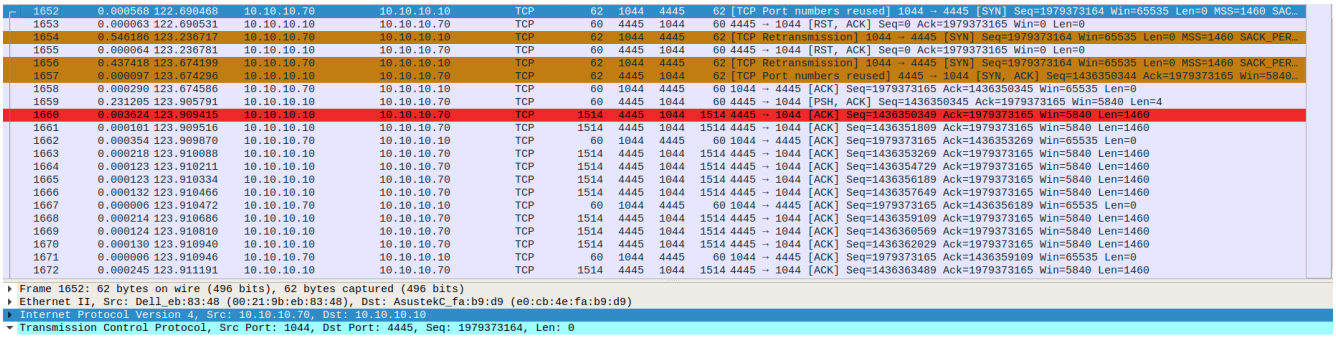
The next question, the IP ID change frequency, was just a matter of opening and watching the id field in the ip header, which shows to change every packet:



For the source port changing, I simply used a “tcp.dstport == 4445” filter, looked at the s. port column I have set, and timed it to 12 seconds between initial SYN packets, each having a separate src port

8.Eventually, the malicious server responded and opened a new connection. When was the TCP connection on port 4445 first successfully completed? (Provide the number of seconds since the beginning of the packet capture, rounded to tenths of a second. ie, 49.5 seconds)

Here, it is just asking when the server responded with a SYN/ACK, and the client with an ACK, signifying that the connection has been established and is ready for data to be sent, the basic and well known three way TCP handshake. I simply scrolled down to when the retransmissions stopped and ACKs from the client started, and right clicking the first ACK to follow the stream. I immediately had a packet catch my attention, but that was not part of this question. A quick look at the time field reveals the answer to be 123.7 seconds:



9.Subsequently, the malicious server sent an executable file to the client on port 4445. What was the MD5 sum of this executable file?

This was exactly the file that caught my attention, as I have that filter mentioned before that is very useful for me. It popped out as having the qualities of an executable, but was not able to be exported using my earlier method. So for this, I followed the stream and saved the content to my vm. From here, it was a simple foremost command and then md5sum to get the sum, which came to b062cb8344cd3e296d8868fbef289c7c:

10.When was the TCP connection on port 4445 closed? (Provide the number of seconds since the beginning of the packet capture, rounded to tenths of a second. ie, 49.5 seconds)

Here, this was as easy as following the TCP filtered stream down to the end of the capture, where we see the FIN/ACK sequence ends at 198.4 seconds: