

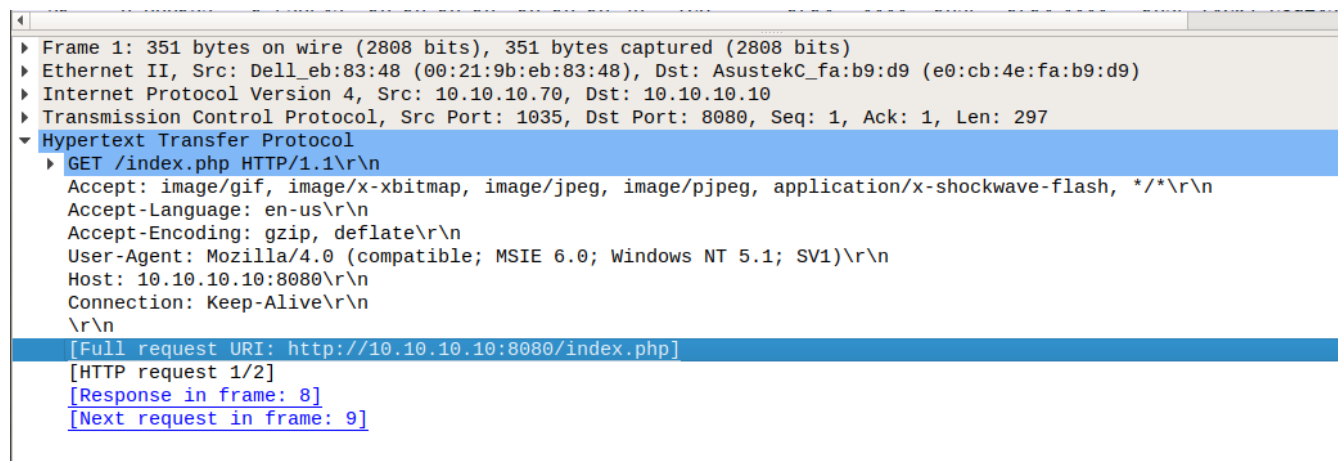
Ann's Aurora – An Advanced Persistent Threat based challenge by SANS.

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 paid 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**.



```

▶ Frame 1: 351 bytes on wire (2808 bits), 351 bytes captured (2808 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: 1035, Dst Port: 8080, Seq: 1, Ack: 1, Len: 297
▼ Hypertext Transfer Protocol
  ▶ GET /index.php HTTP/1.1\r\n
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*\r\n
    Accept-Language: en-us\r\n
    Accept-Encoding: gzip, deflate\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)\r\n
    Host: 10.10.10.10:8080\r\n
    Connection: Keep-Alive\r\n
    \r\n
  [Full request URI: http://10.10.10.10:8080/index.php]
  [HTTP request 1/2]
  [Response in frame: 8]
  [Next request in frame: 9]
```

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 UwnHADOfYH1HDDXj = "COMMENT";\n
var qSngVkOrdIjaiFpPTFdjbPHQppHSGTzpm00yqEbLEfxNqAxicRyZKKWiRwmUaDHF0uzHPHqLrRFSzQuPusTnQyqpQwVpARd1R = new Array();\n
for (i = 0; i < 1300; i++)\n
{\n
qSngVkOrdIjaiFpPTFdjbPHQppHSGTzpm00yqEbLEfxNqAxicRyZKKWiRwmUaDHF0uzHPHqLrRFSzQuPusTnQyqpQwVpARd1R[i] = document.createElement(UwnHADOfYH1HDDXj);\n
qSngVkOrdIjaiFpPTFdjbPHQppHSGTzpm00yqEbLEfxNqAxicRyZKKWiRwmUaDHF0uzHPHqLrRFSzQuPusTnQyqpQwVpARd1R[i].data = "vEI";\n
}\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,

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

▶ Frame 9: 415 bytes on wire (3320 bits), 415 bytes captured (3320 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: 1035, Dst Port: 8080, Seq: 298, Ack: 5873, Len: 361
 ▶ Hypertext Transfer Protocol
 ▶ GET /index.phpmfKSxSANkeTeNrah.gif HTTP/1.1\r\n
 Accept: image/gif, image/x-bitmap, image/jpeg, application/x-shockwave-flash, */*\r\n
 Referer: http://10.10.10.10:8080/index.php\r\n
 Accept-Language: en-us\r\n
 Accept-Encoding: gzip, deflate\r\n
 User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)\r\n
 Host: 10.10.10.10:8080\r\n
 Connection: Keep-Alive\r\n
 \r\n
[\[Full request URI: http://10.10.10.10:8080/index.phpmfKSxSANkeTeNrah.gif\]](#)
[\[HTTP request 2/2\]](#)
[\[Prev request in frame: 1\]](#)
[\[Response in frame: 11\]](#)

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.

9.116183	0.462112	10.10.10.70	10.10.10.10	HTTP	415	1035	8080	415 GET /index.phpmfKSxSANkeTeNrah.gif
9.000117	0.462229	10.10.10.10	10.10.10.70	TCP	60	8080	1035	sansforensics@siftworkstation: ~/Downloads
9.104914	0.567143	10.10.10.10	10.10.10.70	HTTP	201	8080	1035	\$ md5 index.phpmfKSxSANkeTeNrah.gif
9.170074	0.737217	10.10.10.70	10.10.10.10	TCP	60	1035	8080	Command 'md5' not found, did you mean:
9.528634	1.265851	10.10.10.70	10.10.10.10	TCP	62	1036	4444	command 'mdl' from snap mdl (0.11.0)
9.000071	1.265922	10.10.10.10	10.10.10.70	TCP	62	4444	1036	command 'cd5' from deb cd5
9.000296	1.266218	10.10.10.70	10.10.10.10	TCP	60	1036	4444	command 'mdp' from deb mdp
9.260121	1.526339	10.10.10.10	10.10.10.70	TCP	60	4444	1036	command 'mdu' from deb mtools
9.003438	1.529777	10.10.10.10	10.10.10.70	TCP	1514	4444	1036	See 'snap info <snapname>' for additional versions.
9.000079	1.529856	10.10.10.10	10.10.10.70	TCP	1514	4444	1036	sansforensics@siftworkstation: ~/Downloads
9.000322	1.530178	10.10.10.70	10.10.10.10	TCP	60	1036	4444	\$ md5sum index.phpmfKSxSANkeTeNrah.gif
9.000242	1.530420	10.10.10.10	10.10.10.70	TCP	1514	4444	1036	df3e567d6f16d040326c7a0ea29a4f41 index.phpmfKSxSANkeTeNrah.gif
9.000123	1.530543	10.10.10.10	10.10.10.70	TCP	1514	4444	1036	sansforensics@siftworkstation: ~/Downloads

: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
 t II, Src: AsustekC_fa:b9:d9 (e0:cb:4e:fa:b9:d9), Dst: 10.10.10.70
 t Protocol Version 4, Src: 10.10.10.10, Dst: 10.10.10.70
 ssion Control Protocol, Src Port: 8080, Dst Port: 1035,

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**.

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

▶ Frame 9: 415 bytes on wire (3320 bits), 415 bytes captured (3320 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: 1035, Dst Port: 8080, Seq: 298, Ack: 5873, Len: 361
 ▶ Hypertext Transfer Protocol
 ▶ GET /index.phpmfKSxSANKteNrah.gif HTTP/1.1\r\n
 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*\r\n
 Referer: http://10.10.10.10:8080/index.php\r\n
 Accept-Language: en-us\r\n
 Accept-Encoding: gzip, deflate\r\n

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**.

tcp.dstport == 4444 && tcp.flags.ack == 1											Expression...
No.	delta time	Time	Source	Destination	Protocol	Length	s.port	d.port	size	Info	
1481	0.001209	68.508435	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=69617 Ack=1239062 Win=64810 Len=1460	
1482	0.000124	68.508559	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=71077 Ack=1239062 Win=64810 Len=1460	
1483	0.000122	68.508681	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=72537 Ack=1239062 Win=64810 Len=1460	
1484	0.000137	68.508818	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=73997 Ack=1239062 Win=64810 Len=1460	
1485	0.000146	68.508964	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=75457 Ack=1239062 Win=64810 Len=1460	
1488	0.000124	68.509096	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=76917 Ack=1239062 Win=64810 Len=1460	
1489	0.000122	68.509218	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=78377 Ack=1239062 Win=64810 Len=1460	
1490	0.000124	68.509342	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=79837 Ack=1239062 Win=64810 Len=1460	
1491	0.000122	68.509464	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=81297 Ack=1239062 Win=64810 Len=1460	
1492	0.000123	68.509587	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=82757 Ack=1239062 Win=64810 Len=1460	
1493	0.000126	68.509713	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=84217 Ack=1239062 Win=64810 Len=1460	
1494	0.000022	68.509735	10.10.10.70	10.10.10.10	TCP	415	1036	4444	415 1036 → 4444	[PSH, ACK] Seq=85677 Ack=1239062 Win=64810 Len=361	
1495	0.000135	68.509870	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=86938 Ack=1239062 Win=64810 Len=1460	
1496	0.000123	68.509993	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=87498 Ack=1239062 Win=64810 Len=1460	
1497	0.000123	68.510116	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=88958 Ack=1239062 Win=64810 Len=1460	
1498	0.000123	68.510239	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=90418 Ack=1239062 Win=64810 Len=1460	
1499	0.000125	68.510364	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=91878 Ack=1239062 Win=64810 Len=1460	
1500	0.000122	68.510486	10.10.10.70	10.10.10.10	TCP	1514	1036	4444	1514 1036 → 4444	[ACK] Seq=93338 Ack=1239062 Win=64810 Len=1460	
1501	0.000112	68.510598	10.10.10.70	10.10.10.10	TCP	1395	1036	4444	1395 1036 → 4444	[PSH, ACK] Seq=94798 Ack=1239062 Win=64810 Len=13...	
1563	0.000059	87.587154	10.10.10.70	10.10.10.10	TCP	60	1036	4444	60 1036 → 4444	[FIN, ACK] Seq=96139 Ack=1239099 Win=64773 Len=0	
1565	0.000234	87.587480	10.10.10.70	10.10.10.10	TCP	60	1036	4444	60 1036 → 4444	[ACK] Seq=96140 Ack=1239100 Win=64773 Len=0	

▶ Frame 1467: 415 bytes on wire (3320 bits), 415 bytes captured (3320 bits)

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

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 SA
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

▶ Frame 17: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)

▶ Ethernet II, Src: AsustekC_fa:b9:d9 (e0:cb:4e:fa:b9:d9), Dst: Dell_eb:83:48 (00:21:9b:eb:83:48)

▶ Internet Protocol Version 4, Src: 10.10.10.10, Dst: 10.10.10.70

▶ Transmission Control Protocol, Src Port: 4444, Dst Port: 1036, Seq: 5, Ack: 1, Len: 1460

▼ Data (1460 bytes)

Data: 4d5ae000000005b52455589e581c3cb110000fffd389c357...

[Length: 1460]

0030	16 d0 f7 a2 00 00	4d 5a e8 00 00 00 00 5b 52 45MZ[RE
0040	55 89 e5 81 c3 cb 11 00 00 ff d3 89 c3 57 68 04	U..... ..wh	
0050	00 00 00 50 ff d0 68 f0 b5 a2 56 68 05 00 00 00	...P..h. ..Vh...	
0060	50 ff d3 00 00 00 00 00 00 00 00 00 00 00 00	P..... ..	
0070	00 00 d8 00 00 00 0e 1f ba 0e 00 b4 09 cd 21 b8!..	
0080	01 4c cd 21 54 68 09 73 20 70 72 6f 67 72 61 6d	..L!This program	
0090	20 63 61 6e 6e 6f 74 20 62 65 20 72 75 6e 20 09	cannot be run i	
00a0	6e 20 44 4f 53 20 6d 6f 64 65 2e 0d 0d 0a 24 00	n DOS mo de....\$.	
00b0	00 00 00 00 00 00 8a e2 9d d8 ce 83 f3 8b ce 83	

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 wireshark 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:

The screenshot displays the Wireshark network protocol analyzer interface. The packet list on the left shows a captured packet with the following details in the packet details pane:

- Source Port: 1037
- Destination Port: 4445
- [Stream index: 5]
- [TCP Segment Len: 0]
- Sequence number: 0
- [Next sequence number: 0]
- Acknowledgment number: 0
- 0111 ... = Header Length
- Flags: 0x002 (SYN)
- Window size value: 65535
- [Calculated window size: 65535]
- Checksum: 0xac86 [unverified]
- [Checksum Status: Unverified]
- Urgent pointer: 0
- Options: (8 bytes), Maximum Segment Size, Window Scale, Sack Permitted, Timestamp
- [SEQ/ACK analysis]
- [Timestamps]

The packet bytes pane shows the raw data of the packet. A context menu is open over the packet list, showing various actions that can be performed on the selected packet. The 'Protocol Preferences' option is highlighted in the menu.

```

▶ 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**:

The image shows a Wireshark packet capture interface. The top pane displays a list of captured packets. The bottom pane shows the detailed view of the selected packet (Frame 1180).

No.	Time	Source	Destination	Protocol	
1207	0.390102	48.643028	10.10.10.70	10.10.10.10	TCP
1209	0.000586	48.643685	10.10.10.70	10.10.10.10	TCP
1211	0.436780	49.080522	10.10.10.70	10.10.10.10	TCP

Frame 1180: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
Ethernet II, Src: Dell_EB:83:48 (00:21:9b:eb:83:48), Dst: AsustekC_fa:b9:d9 (e0:cb:4e:14:b9:d9)
Internet Protocol Version 4, Src: 10.10.10.70, Dst: 10.10.10.10
0100 = Version: 4
.... 0101 = Header Length: 20 bytes (5)
▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 48
Identification: 0x0174 (372)
▸ Flags: 0x4000, Don't fragment
Time to live: 128
Protocol: TCP (6)
Header checksum: 0xd0f0 [validation disabled]

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:

1652	0.000568	122.690468	10.10.10.70	10.10.10.10	TCP	62	1044	4445	62 [TCP Port numbers reused] 1044 → 4445 [SYN] Seq=1979373164 Win=65535 Len=0 MSS=1460 SAC
1653	0.000063	122.690531	10.10.10.10	10.10.10.70	TCP	60	4445	1044	60 4445 → 1044 [RST, ACK] Seq=0 Ack=1979373165 Win=0 Len=0
1654	0.546186	123.236717	10.10.10.70	10.10.10.10	TCP	62	1044	4445	62 [TCP Retransmission] 1044 → 4445 [SYN] Seq=1979373164 Win=65535 Len=0 MSS=1460 SACK_PER
1655	0.000064	123.236781	10.10.10.10	10.10.10.70	TCP	60	4445	1044	60 4445 → 1044 [RST, ACK] Seq=0 Ack=1979373165 Win=0 Len=0
1656	0.437418	123.674199	10.10.10.70	10.10.10.10	TCP	62	1044	4445	62 [TCP Retransmission] 1044 → 4445 [SYN] Seq=1979373164 Win=65535 Len=0 MSS=1460 SACK_PER
1657	0.000097	123.674296	10.10.10.10	10.10.10.70	TCP	62	4445	1044	62 [TCP Port numbers reused] 4445 → 1044 [SYN, ACK] Seq=1436350344 Ack=1979373165 Win=5840
1658	0.000290	123.674586	10.10.10.70	10.10.10.10	TCP	60	1044	4445	60 1044 → 4445 [ACK] Seq=1979373165 Ack=1436350345 Win=65535 Len=0
1659	0.231205	123.905791	10.10.10.10	10.10.10.70	TCP	60	4445	1044	60 4445 → 1044 [PSH, ACK] Seq=1436350345 Ack=1979373165 Win=5840 Len=4
1660	0.003624	123.909415	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436350349 Ack=1979373165 Win=5840 Len=1460
1661	0.000181	123.909516	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436351809 Ack=1979373165 Win=5840 Len=1460
1662	0.000354	123.909870	10.10.10.70	10.10.10.10	TCP	60	1044	4445	60 1044 → 4445 [ACK] Seq=1979373165 Ack=1436353269 Win=65535 Len=0
1663	0.000218	123.910088	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436353269 Ack=1979373165 Win=5840 Len=1460
1664	0.000123	123.910211	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436354729 Ack=1979373165 Win=5840 Len=1460
1665	0.000123	123.910334	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436356189 Ack=1979373165 Win=5840 Len=1460
1666	0.000132	123.910466	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436357649 Ack=1979373165 Win=5840 Len=1460
1667	0.000006	123.910472	10.10.10.70	10.10.10.10	TCP	60	1044	4445	60 1044 → 4445 [ACK] Seq=1979373165 Ack=1436356189 Win=65535 Len=0
1668	0.000214	123.910686	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436358109 Ack=1979373165 Win=5840 Len=1460
1669	0.000124	123.910810	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436360569 Ack=1979373165 Win=5840 Len=1460
1670	0.000130	123.910940	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436362029 Ack=1979373165 Win=5840 Len=1460
1671	0.000006	123.910946	10.10.10.70	10.10.10.10	TCP	60	1044	4445	60 1044 → 4445 [ACK] Seq=1979373165 Ack=1436359189 Win=65535 Len=0
1672	0.000245	123.911191	10.10.10.10	10.10.10.70	TCP	1514	4445	1044	1514 4445 → 1044 [ACK] Seq=1436363489 Ack=1979373165 Win=5840 Len=1460

▶ Frame 1652: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
 ▶ Ethernet II, Src: Dell eb:83:48 (00:00:21:0b: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: 1044, Dst Port: 4445, Seq: 1979373164, Len: 0

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:

```

Hp.....u
...p.....3..E..t2..t..}.h.r..St..D.....p.....
..M.....t.Q..Y..[.]..t.Vh.....u.....j..u..u.....^..V3.95..    .t ...    .P.g...F..i.....    ..8.Yu.^..V3.95..    .t ...
.P.....F..i.....    ..8.Yu.^..t$.?...?..3.Y@..$.=..*
t"=..Vt.=..2..u..j...p..3..@..j.....p.....p.....U...M.3.@It!t..It.Iu/.M...t{...
.....u.....u.....Y..M..
..
.]...3..@..D$....D$....
..
..4..t..a..Y..m..Y..%.p.....D$....t$.t...=...Y..I..Y..t$....Y.U.....B...@.
3..E..V..W..v.....V.
{..Y3.Y..G.....P..j..j.....P..F..P.....Dr...t..j..h.....PV..Hr...u.....p.....Y.M..3.^.....U.....W.
V..r..V..U..YY3..U..U..RQ.Q.M.....Q@P.....Dr...v...C...Y..._.W.V...!.....$. ".h. .h:.....
..b.....P.....3....
..u
.v.....Y...9.....~.....
..
..G..!...;|.hB...3...$. ...$.H.....;...$.M.....$.f.....Y3.G..W3.;.u.
3...v..d...v.....v.....W.....W.....W.....W.R..W.....~.....
4.....YG..|...;|.v.
...v.....YY3..@..W.v.3.G.....V.g....Q..P..F.....j..j..j!P..F.....v..w..j..j..P..F.....v..v..0...4..u.3...v.....Y;..u..j..h
s..v.....v.....Y...U..Q.e..V..WV.....Y..uc.@jd.....Y...~0..E.PV.....YY..u5..u..Vh.....t..P.....Y..|....
j..p.....YY..t.....^..U..QV3..!u.95..
.u+hTs...Hp.....t.h<s..V...p.....
..=..
..t..E..P..xp..P....
..u..M...t..V...|p...E..^..h....h0. ....].....V3.W.....P.....VW.....P.....9=..
..u..W...p.....
..W...P.....YY..}.....
..S.k...Y..;..u

```

52 client pkts, 596 server pkts, 48 turns.

Entire conversation (844 kB)
Show and save data as ASCII
Stream 41

Find:
Find Next

Help
Filter Out This Stream
Print
Save as...
Back
Close

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**:

2540	192.946230	10.10.10.70	10.10.10.10	TCP	224 1044 → 4445 [PSH, ACK] Seq=10827 Ack=1437183283 Win=65386 Len=170
2542	192.955445	10.10.10.70	10.10.10.10	TCP	128 1044 → 4445 [PSH, ACK] Seq=10827 Ack=1437183283 Win=65386 Len=74
2544	192.955828	10.10.10.70	10.10.10.10	TCP	208 1044 → 4445 [PSH, ACK] Seq=10901 Ack=1437183283 Win=65386 Len=154
2547	194.393200	10.10.10.70	10.10.10.10	TCP	128 1044 → 4445 [PSH, ACK] Seq=11055 Ack=1437183400 Win=65269 Len=74
2549	194.393349	10.10.10.70	10.10.10.10	TCP	304 1044 → 4445 [PSH, ACK] Seq=11129 Ack=1437183400 Win=65269 Len=250
2552	198.440669	10.10.10.70	10.10.10.10	TCP	60 1044 → 4445 [FIN, ACK] Seq=11379 Ack=1437183437 Win=65232 Len=0
2554	198.441738	10.10.10.70	10.10.10.10	TCP	60 1044 → 4445 [ACK] Seq=11380 Ack=1437183438 Win=65232 Len=0

▶ Frame 1153: 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: 0, Len: 0