CND Assignment 5 Rajat

Ans1.

sudo tcpdump -i en0 -w netanalysis2.pcap | for i in {161.253.119.125,161.253.113.105}; do for j in {1337,12345,1,23,8,5900,25,1433,135,22}; do nc -v -n -z -w 1 $i $j; done; for k in {1337,53,8,5060,137,8000}; do nc –u -v -n -z -w 1 $i $k; done; done



Please find the netanalysis2.pcap file for reference in zip folder.

Ans2.

sudo nmap -d -d -r 161.253.113.105 -oX portslist.xml

Please fing the portslist.xml file for reference

nmap –d -d -vv --reason -r 161.253.113.105 -oN portscanlist

Please find the portscanlist file for reference

tcpdump -s 0 -w rajat\_homework5.pcap | nmap -d -d -r 161.253.113.105

Please find the pcap file in zip folder

NOTE: I was unable to find where I stored my pcap file first time so I reran the script with different ip to get pcap file later which is attached in zip folder.

Ans 3.

mkfifo pip

Server

sudo tcpdump -s 0 -w rajat\_homework5\_ans3\_server.pcap | nc -l 5555 0<pip|nc google.com 80 2>pip

Client

Terminal 1

tcpdump –i eth2 -s 0 -w rajat\_homework5\_ans3\_client.pcap

Ternimal 2

nc 161.253.120.102 5555

GET / HTTP/1.1

Please find the pcap files and the stdout file in Ans 3 folder

Ans4.

Please find the python program and its output file in Ans 4 folder along with PCAP file(Although I feel pcap file is not capturing anything related).

**from** dnslib **import** DNSRecord,DNSHeader,DNSQuestion,RR,A  
  
c=list()  
**for** i **in** range(0,100):  
 d = DNSRecord(DNSHeader(qr=1,aa=1,ra=1,auth=1,bitmap=0 ),q=DNSQuestion(qname='z.tiwaz.net',qtype=1,qclass=1),a=RR(rname="www.google"+str(i)+".com",rdata=A("127.0.0.1"),rtype=2))  
 c.append(d)  
  
**print** c

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2nd Attempt for this Question:

Since previously I just generated dns response using dnslib, this time I tried to get the response using scapy. I am attaching the scapy python file along with pcap for the same.

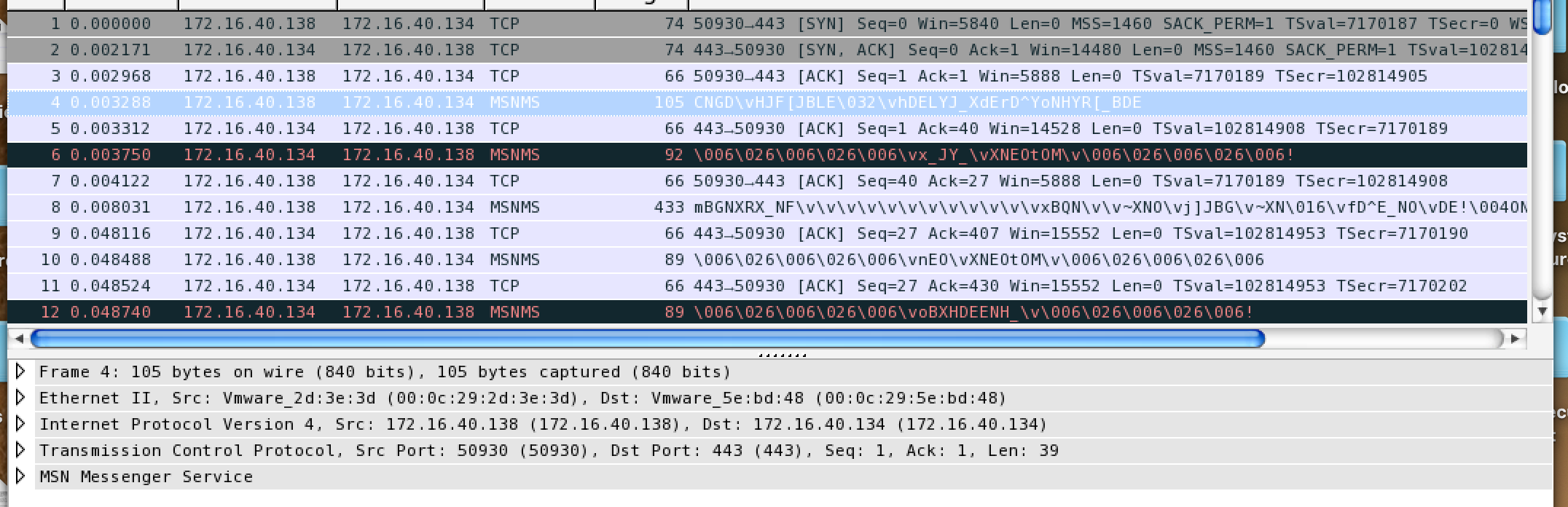
To receive the response from 127.0.0.1 I used dnsmasq to use my localhost as dnsresolver and entered z.tiwaz.net in my hosts file(I used dnsmasq.hosts to provide additional hosts and tied z.tiwaz.net to 127.0.0.1). But I wasn’t able to get any response when I ran scapy code with my dns server address as 127.0.0.1. So I used google’s dns to get the dns resposne.

To get the response from 127.0.0.1 which I was able to get using nslookup and dig. I used a python file to loop dig command 100 times to get dns response from 127.0.0.1. Please refer the attached python file along with their pcap capture for reference.

Ans 5.

All the TCP handshake requests are initiated by 172.163.40.138. It chooses a random port and increments for subsequent requests. The request is always made to 172.16.40.134 on port 443.

After analyzing the file using wire shark; tried to decode the TCP to other formats like IMAP, SSH, DNS etc. I saw some changes; however I could not retrieve any message from it. IMAP made most sense when the SSH Continuous data got transformed to Requests and responses



After analyzing the PCAP file i can understand that there is a set pattern here, multiple messages are being repeated and we can look for that raw data that is being transferred over the network and write snort rules for that.

Patterns in PCAP

First message from the client in a message block between and client and server is always :

1. CNGD\vHJF[JBLE\032\vhDELYJ\_XdErD^YoNHYR[\_BDE

Second Message is always the response from server as :

1. \006\026\006\026\006\vx\_JY\_\vXNEOtOM\v\006\026\006\026\006!

I think that third message and the message onwards are the payload sent by client which are in continuation of the message 1 as . Some Examples are :

1. [truncated]mBGNXRX\_NF\v\v\v\v\v\v\v\v\v\v\v\vxBQN\v\v~XNO\vj]JBG\v~XN\016\vfD^E\_NO\vDE!\004ON]\004XOJ\032\v\v\v\v\v\v\v\v\v\v\v\v\v\v\032\022l\v\v\023\005\037l\v\v\022\005\035l\v\v\037\034\016\v\004!\_F[MX\v\v\v\v\v\v\v\v\v\v\v\v\v\v\v\v\v

or

[truncated]N\_C\033\v\v\v\v\v\vgBE@\vNEHJ[\021n\_CNYEN\_\v\vc|JOOY\v\033\033\021\033H\021\031\022\021\031O\021\030N\021\030O\v\v!\v\v\v\v\v\v\v\v\v\vBEN\_\vJOOY\021\032\034\031\005\032\035\005\037\033\005\032\030\023\v\viHJX\_\021\032\034\031\

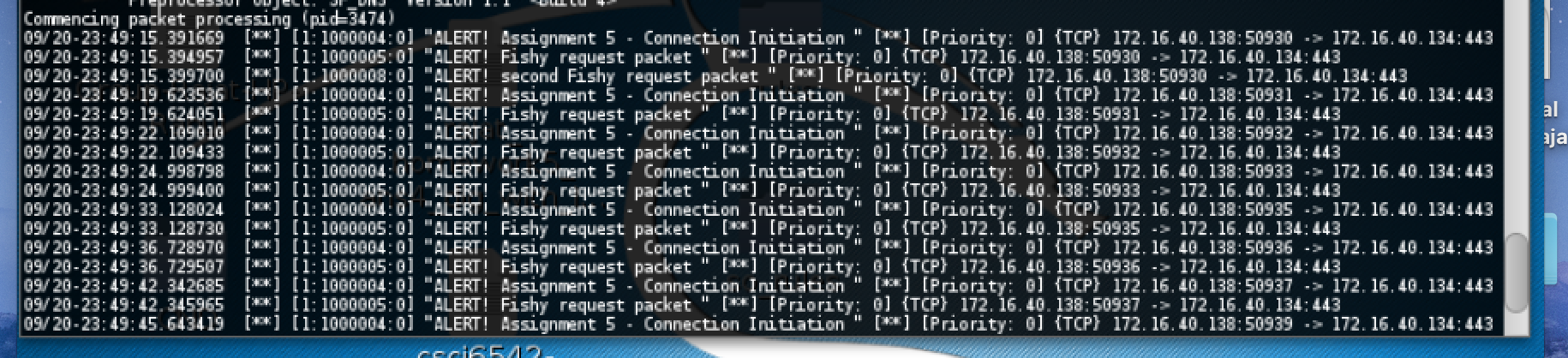
Second last message is always :

1. \006\026\006\026\006\vnEO\vXNEOtOM\v\006\026\006\026\006

Last Message is always :

1. \006\026\006\026\006\voBXHDEENH\_\v\006\026\006\026\006!

I tried to write snort rules based on the patterns I found, I was able to get alerts for client requests but wasn’t able to get anything for server response as shown in the below screenshot.



To begin with snort rules I would expect the header part to e as below:

**Alert tcp 172.163.40.138 any -> 172.16.40.134 443 (msg:”ALERT! Assignment 5 - Connection Initiation! “; flags: s; sid:1000001);**

Fields:

|  |  |  |
| --- | --- | --- |
| Action: | **Alert** | because we need an alert to pop up when a snort rule matches |
| Protocol: | **tcp** | because the protocol under analysis from the given pcap is TCP |
| SRC IP adderess : | **172.163.40.138** | because the TCP handshake is being initiated by 172.163.40.138 |
| SRC port : | **any** | because in the pcap 172.163.40.138 chooses a random port and increases the port number for subsequent requests |
| Direction operator: | **->** | direction of traffic for this rule |
| Dest IP : | **172.16.40.134** | TCP request is made to 172.16.40.134 from the source |
| Dest port: | **443** | port for 172.16.40.134 |
| Msg: | **ALERT! Assignment 5 - Connection Initiation** | Message to be displayed |
| Flags : | **S** | because we want to check for the case when the flag is set to SYN |
| Sid : | **1000001** | Assign a Snort id to the rule |

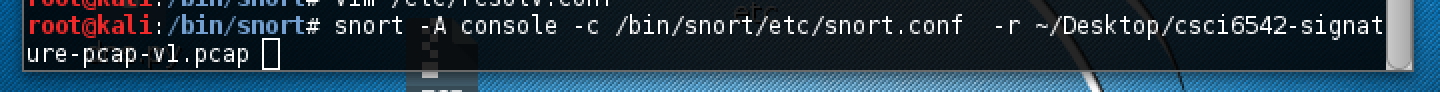
This above rule would give an alert when 172.163.40.138 tries to initiate a connection to 172.16.40.134 on port 443.

**alert tcp 172.16.40.138 any -> 172.16.40.134 443 (msg:”ALERT! Fishy request packet “; sid:1000005; content:"|43 4E 47 44|"; rawbytes;)**

I noticed that all the initial request packets have a common repeating text “CNGD” hence we search the hex equivalent 43 4E 47 44 in raw bytes.

The above rule gives an alert when the first request packet is sent.

Below is the screenshot after running the pcap against the rules

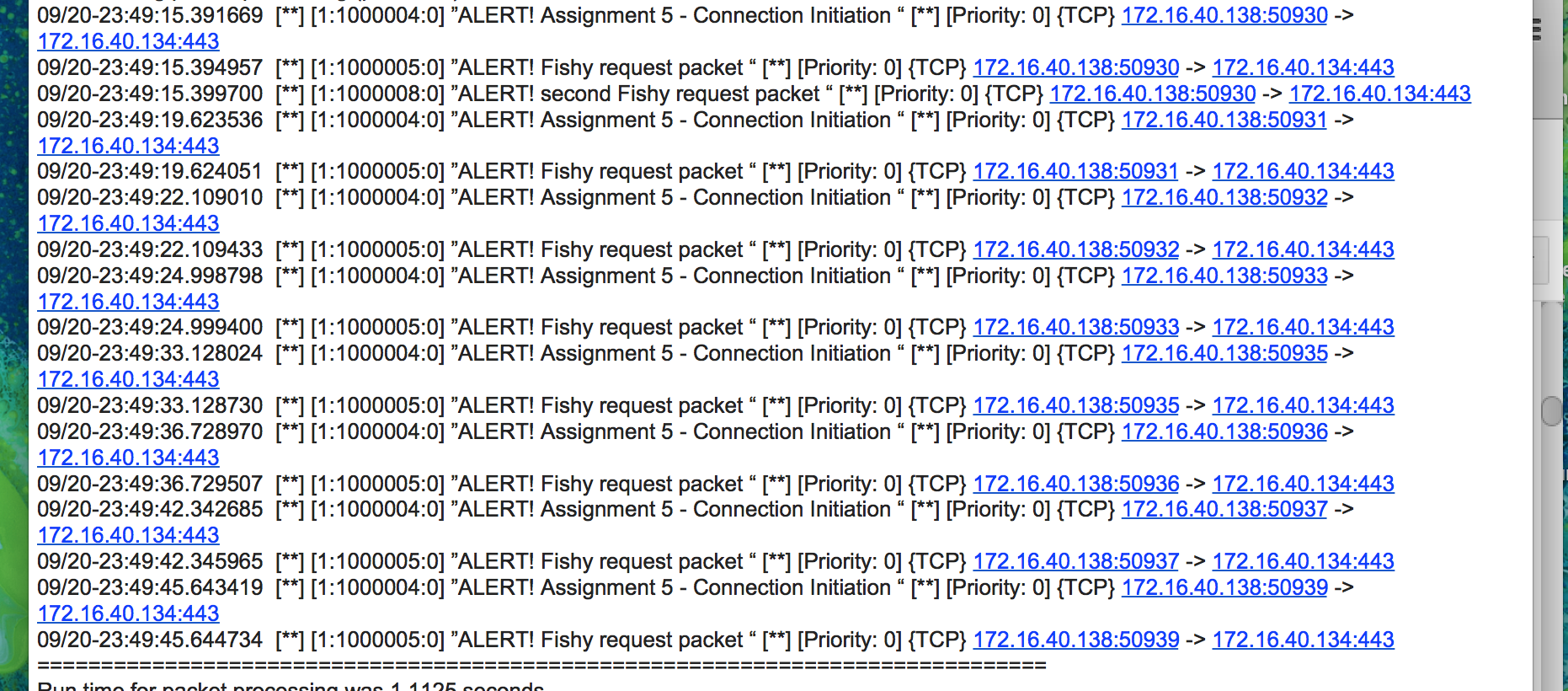


-c is to choose the conf file

-A is to display the output on the console

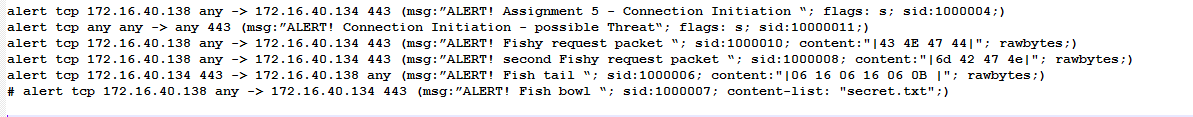
-r is to provide the pcap file as input

Below is the output received and I have attached the same in a text file:

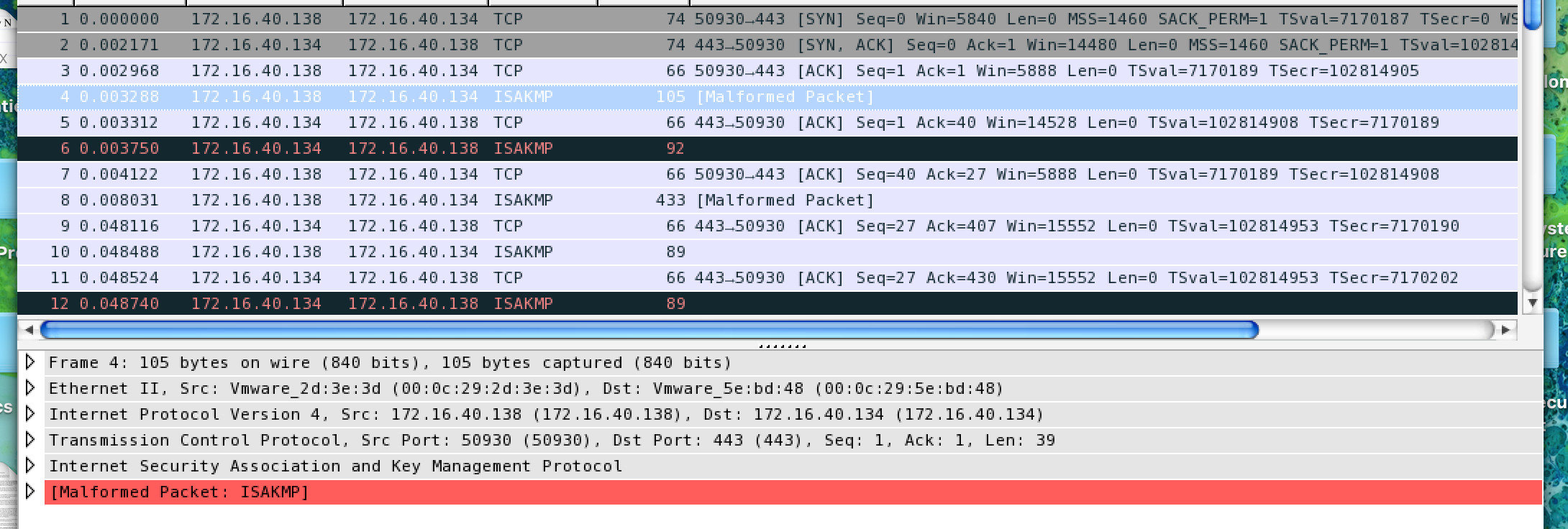


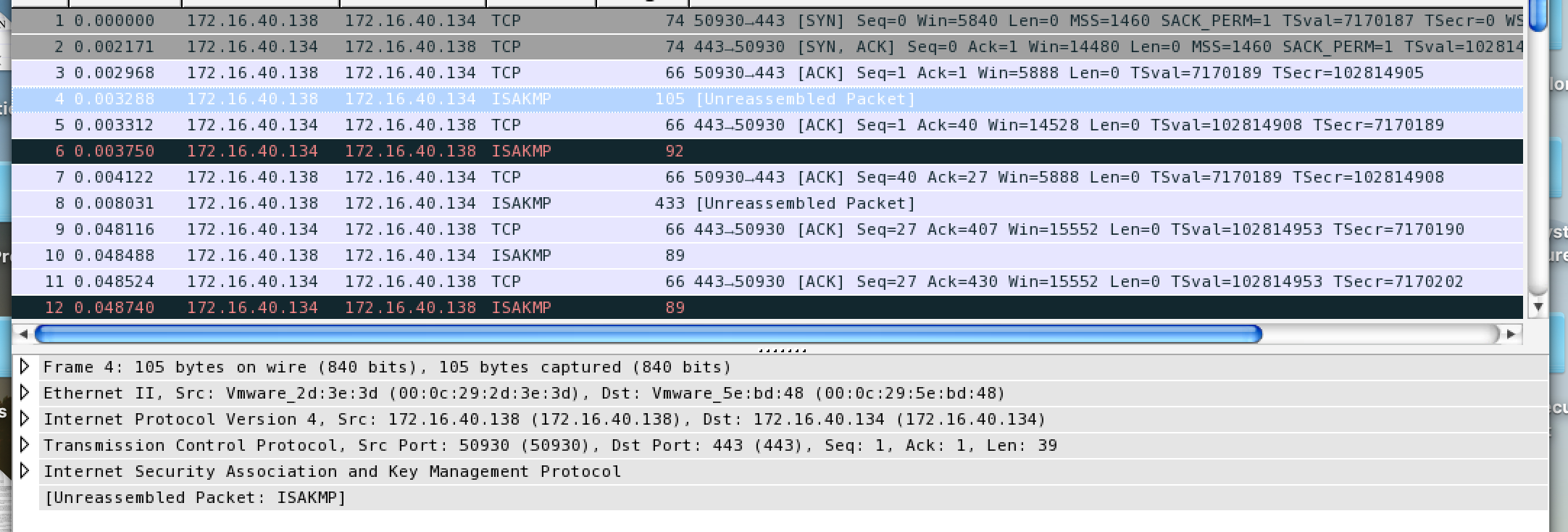
I am able to capture alerts for connection initiation and first request packet.

I wrote some more rules, however I could not capture any response packets. Below are my rules

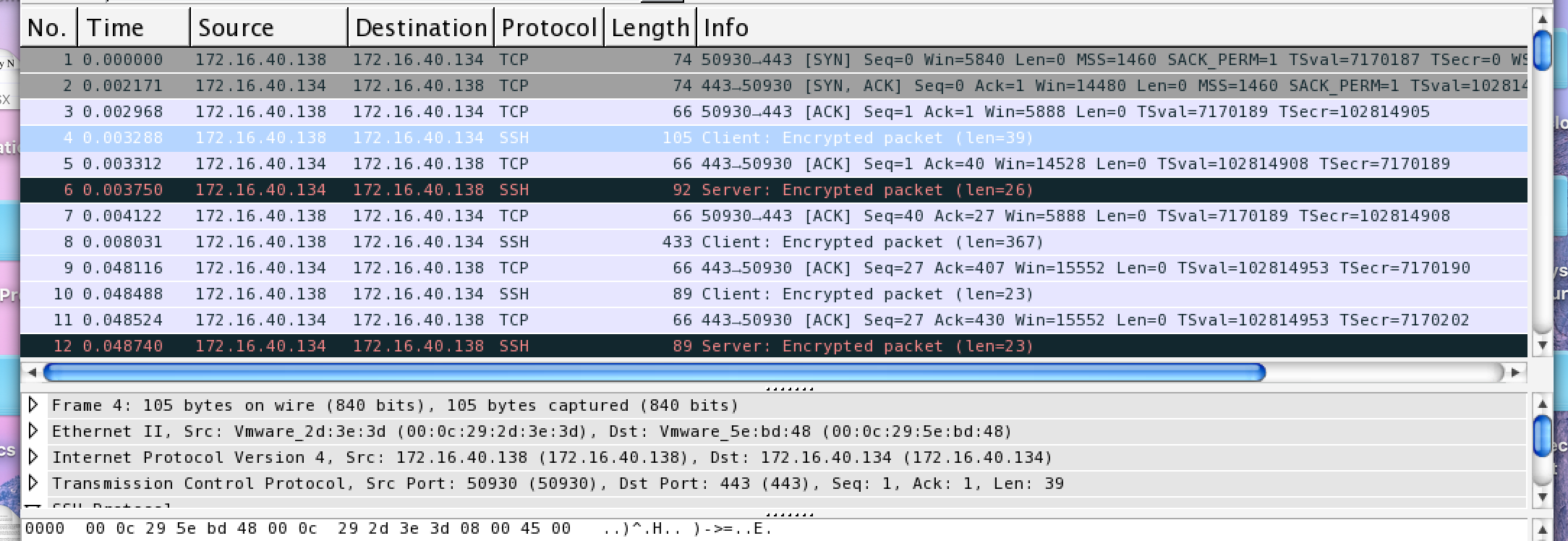


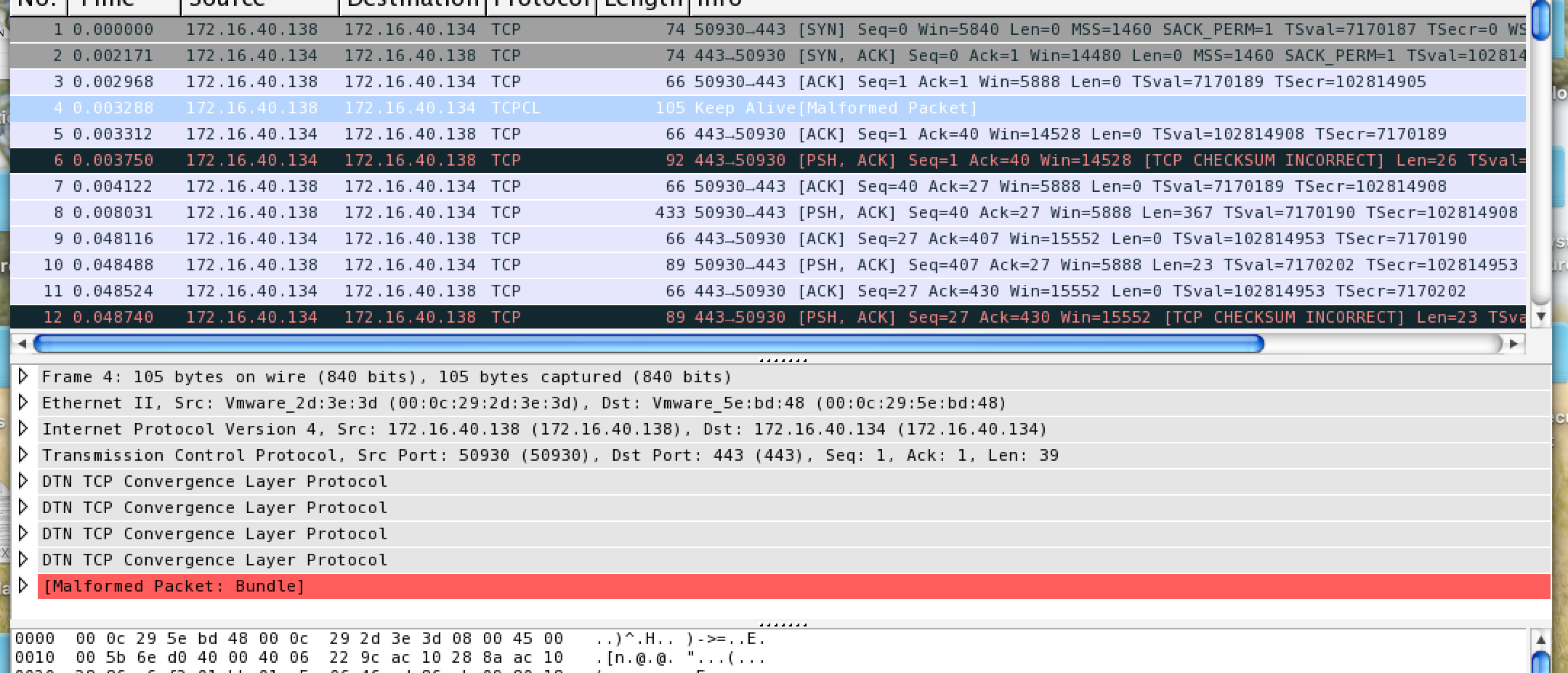
Ans 6. After Decoding the PCAP file in different decoders I found that in some decoder we got few malformed packets like shown in below screenshot. So to view the information I unchecked the PDU reassembly at TCP protocol.



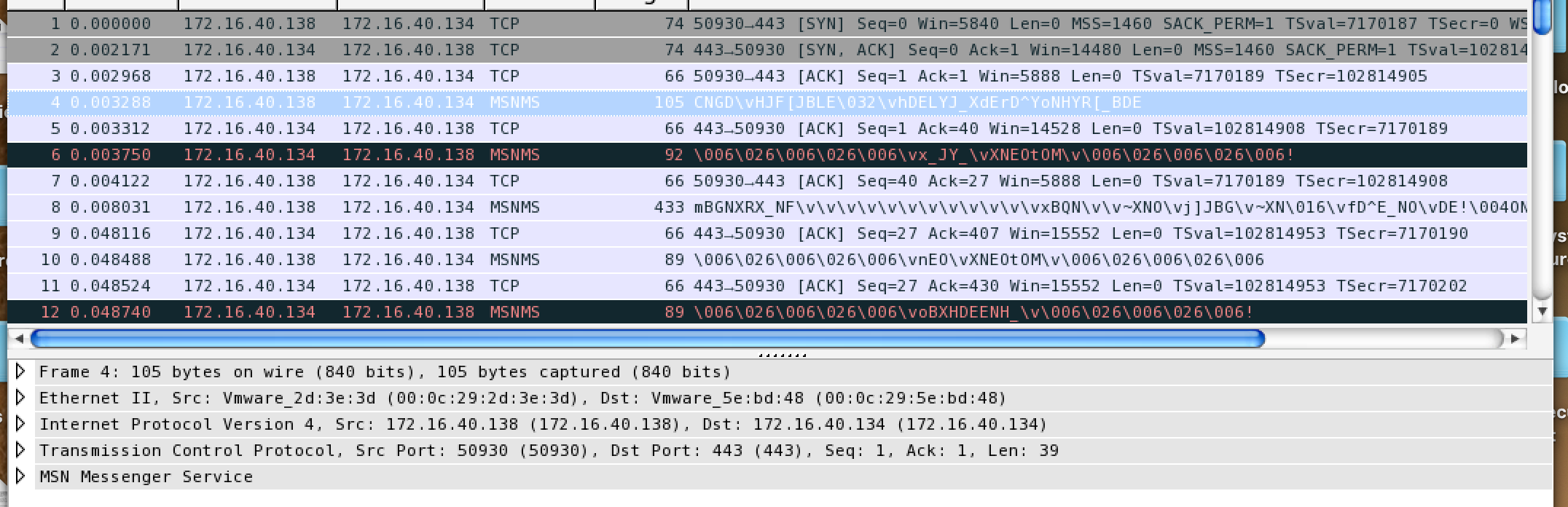


For SSH decoding we got that the data is sent by client and server with encryption. So we know that the data being sent is encrypted and its needs decryptions





In ICMP or MSMNS decoding we we able to the the encrypted data we we can analyze and try to decrypt it to clear text.



After getting the data and analyzing the PCAP file we found that the traffic is having a set pattern and payload is sent in between that. So can try decrypting that pattern data.

Ans 7.

In this we can use tcp reset attack to disrupt the communication. We just need to change the reset flag in TCP during our man in the middle attack.

So I used pcap to set the interface en0 to be promiscuous mode and monitor the TCP traffic. When it detects a packet that has TCP layer, it will strip its fields and search if the flag for the TCP packet is being set to 1 (This is the predefined string value that it will look for). If the flag is being set to 1, it will inject a new packet to disrupt the TCP session. However, the code was not successful.

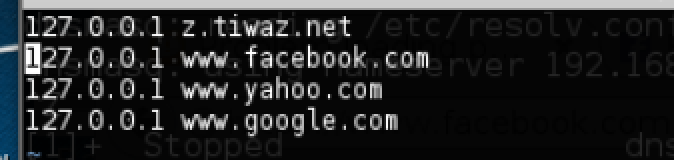
I am attaching the python for the same

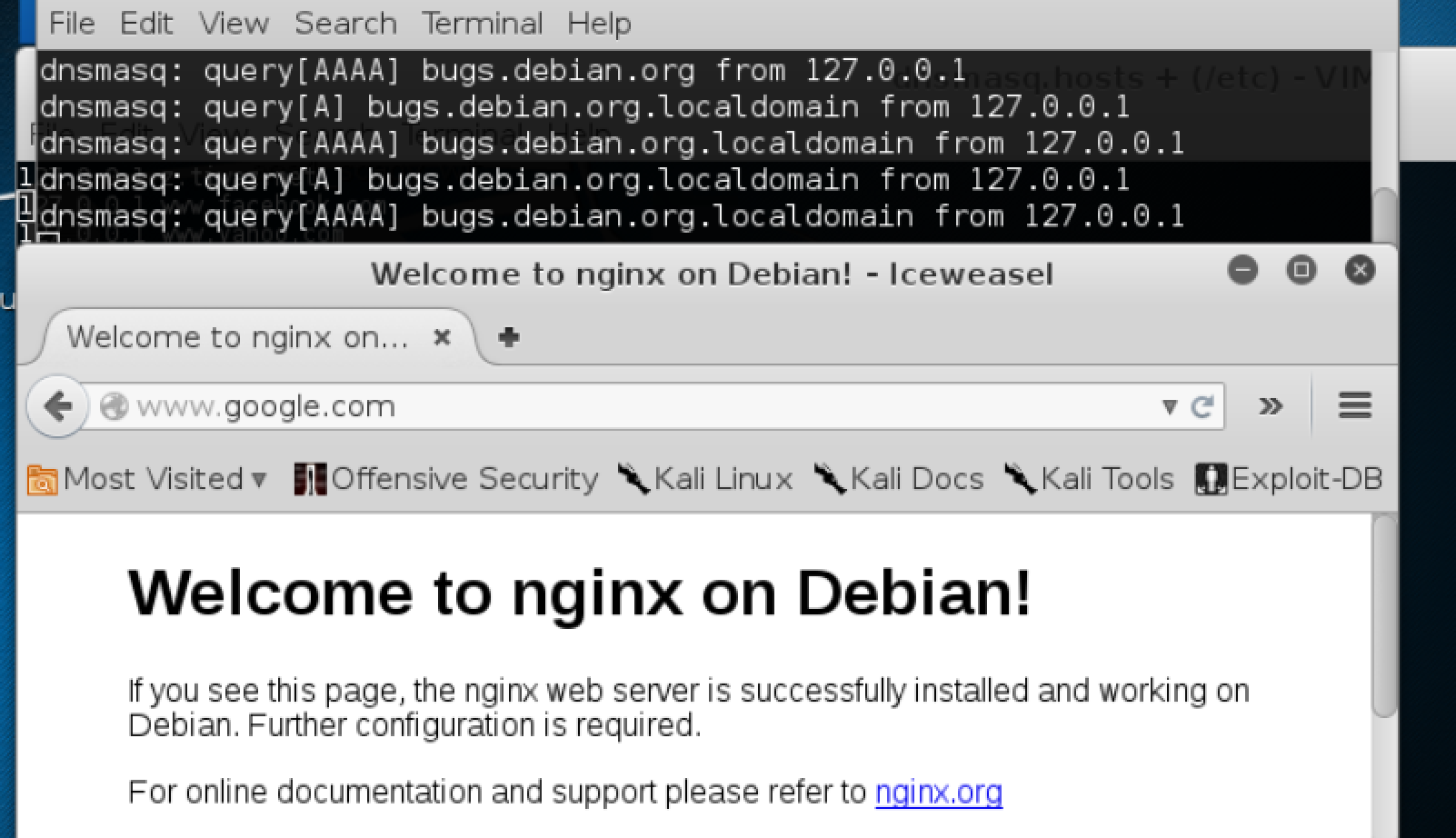
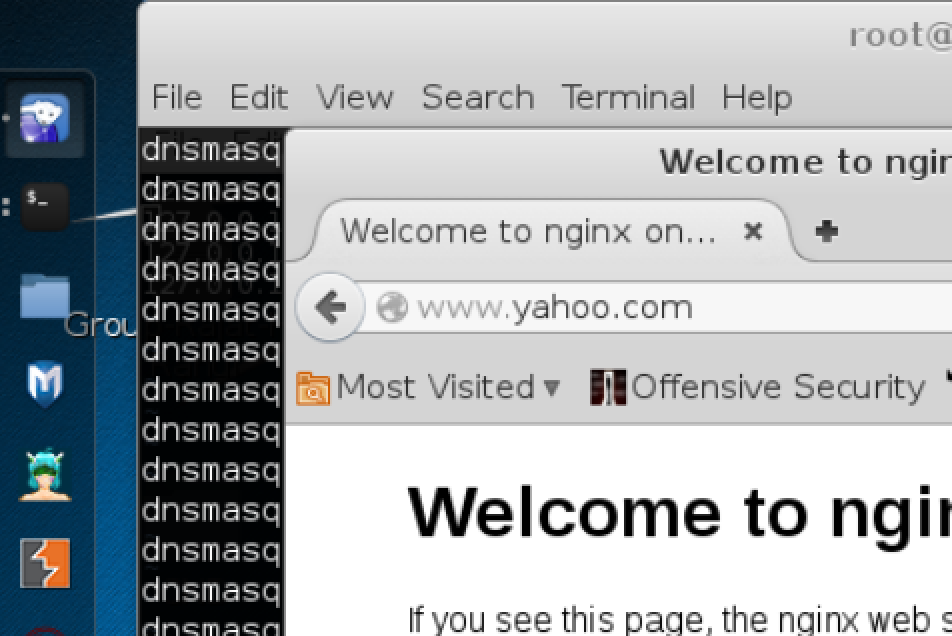
Ans 8. DNS Man in the middle attack:

We can use dnsmasq for this so create a DNS Server that has hosts file pointing of different IP addresses. I used my localhost as dns server to do that by pointing facebook, yahoo and google to my localhost.

So when I tried to access them I got my localhost in the browser.

Dnsmasq.conf file



Tests for google and yahoo gives localhost.

In my PCAP file I have some other packets captured too, but we can see that initially yahoo.com is getting query response of 127.0.0.1.

NOTE: I collaborated with Divya and Yin for the Assignment, so it might be possible to have some similarities in our write ups.