Problem 1.

List of http servers that were involved in a valid http response:

['111.4.186.50', '136.93.4.213', '138.59.102.27', '142.165.192.177', '142.165.192.188', '143.138.4.147', '143.138.66.97', '143.179.11.189', '154.27.68.55', '154.87.109.177', '154.87.109.40', '155.111.186.252', '155.231.237.70', '159.70.229.173', '159.79.22.194', '159.79.22.198', '159.79.22.249', '205.232.201.218', '205.232.203.30', '205.234.49.157', '248.78.109.66', '251.235.172.148', '33.247.152.101', '33.247.152.113', '34.30.235.180', '35.183.215.204', '37.120.175.85', '37.199.226.67', '40.187.57.142', '44.111.85.82', '44.131.48.102', '44.131.51.161', '44.131.51.48', '93.119.134.44', '93.199.112.45', '97.145.19.119']

How do we obtain the list of servers?

For determining all http servers that were involved in a valid http response we can use a filter such as: http and http.response.code != 0

Next, of all the filtered http packets obtained after applying the above filter, we extract the IP addresses of the source (which is the http server that sends the response). We consider all valid responses with response codes such as 200, 404, 302 etc. There are 118 such http servers in total.

Problem 2)

We can easily identify a source attempting a directory traversal using the filter:
http.request.uri contains "../.."

Following are some sample http requests where an attacker is attempting a directory traversal for the /etc/passwd file:

GET icsc/ICSC09_Advance_Program.pdf/index.php?p=../../../../../../../../../../../../../etc/passwd%00 HTTP/1.1

GET /icsc/index.php?p=../../../../../../../../../../../../../etc/passwd%00 HTTP/1.1

GET /index.php?p=../../../../../../../../../../../../../etc/passwd%00 HTTP/1.1

In all these cases, we have a malicious host with IP address **42.9.203.117** is attempting to perform a directory traversal in the response.

How do we obtain this?

We apply the filter http and http.request.uri contains "../..". An attacker attempting a directory traversal at the server side will send in a http request whose URI will have a directory structure such as the ones given above.

So after filtering out the http requests using this filter, we next obtain a list of all source IP

addresses from the packets that have a http request with directory traversal. In our case, there's one such host which attempts a directory traversal.

Problem 3) In this question we need to find out a malicious host that is attempting password guessing on a secure ftp server.
We can check the password guessing attack on the FTP server by using a "ftp" filter and then check for "Login Incorrect" responses as the one shown below:

Following is a snipped from the trace containing failed login attempts:
USER Administrator
331 Password required for Administrator.
PASS Volley
530- *** ERROR ***
USER Administrator
530- Only anonymous FTP is available on ftp.ICSI.Berkeley.EDU.
530- Please use uftp.ICSI.Berkeley.EDU for user-based FTP authentication.
530 Login incorrect.
USER Administrator
331 Password required for Administrator.
PASS ashley
331 Password required for Administrator.
PASS ashley
530- *** ERROR ***
530- Only anonymous FTP is available on ftp.ICSI.Berkeley.EDU.
530- Please use uftp.ICSI.Berkeley.EDU for user-based FTP authentication.
530 Login incorrect.
USER Administrator
503 Login with USER first.
331 Password required for Administrator.

In this example, a malicious user is attempting to login as an "Administrator" and is executing the "password guessing" attack by trying out different possible passwords for the user "Administrator".
In this case, the malicious user with IP address **159.79.22.194** attempted password guessing for the administrator account for a FTP server hosted at **248.35.162.92**

How do we obtain this?
We can use the following filter for ftp packets: ftp and ftp.response.code == 530
The idea is that on a failed login attempt, FTP returns a status response code of 530 and is returned as an error/login failed. Now, a single failed login may not be suspicious. But if we have multiple failed login attempts, then we can say that the given host is malicious.

Problem 4) There are several protocols that can send out unencrypted usernames and passwords. Some of the popular ones include ftp, http, telnet and pop3.

In the trace file, we can identify a real username and password sent out in the clear while making an FTP request with the following details:
From Source: 172.27.37.232 to Destination: 151.37.121.114.
Username: **calrules** and **Password**: **thisissosecure**
After specifying the username calrules to the FTP server, it sends back a response back to the FTP server returns a response "Please specify the password" after which the host (172.27.37.232) sends out a password "thisissosecure" out in the clear.

How do we automate this?
Automating and scaling this problem is especially challenging since we really do not know what is the format in which a username and password is sent and by which protocol. So we can first apply this filter, "ftp or http or telnet or pop". This removes other packets and retains only those packets that belong to the said protocols.

Next, we examine the tcp streams associated with the given protocol packets in the wireshark GUI. This is done by right clicking the packet and then selecting "Follow TCP stream". Then we can check for packets that send out login (username/password) information from the source host to the server and see if we can find a human readable password sent out in cleartext. If yes, then that is our answer.

In the current problem, we obtain a clear text password for ftp with username calrules and password thisissosecure.

Problem 5) We know that an http web server can run Apache. This problem deals with identifying the http server that runs the oldest version of Apache.
In the given problem, the http server running the oldest version of Apache is:
Server IP address: **205.232.201.218**
Apache Version: **Apache/1.3.28**

How do we get this?
We apply a filter such as this one: http and http.server contains "Apache/"
This returns packets corresponding to http response from the server back to the client. A http response contains information about the "http server" (in our case Apache) and its current version. So we collect all such http responses, extracting the IP address of the server and the current Apache version that it is running.
Then we can compare the Apache versions and the server running the oldest version is our answer (there can be more than one such servers but this trace file returns only one).

Problem 6) Most clients now use a random UDP source port when making DNS queries to help prevent “Kaminsky attack”. In this questions, we need to identify hosts that “do not” use a random port i.e they use the same source udp port for all dns requests.
Such hosts can be easily identified by looking at the dns request packets for dns and checking the udp source port in them.

The below are the representative packets for hosts that use a static UDP source port number:
a) Internet Protocol Version 4, Src: **205.232.201.218,** Dst: 205.232.183.94
User Datagram Protocol, Src Port: 33814, Dst Port: 53

b) Internet Protocol Version 4, Src: **205.232.201.195**, Dst: 205.232.183.50
User Datagram Protocol, Src Port: 32927, Dst Port: 53

As we can see, host **205.232.201.218** always makes a dns request with a static port number 33814. Similarly, the host **205.232.201.195** always makes a dns request with a static port number of 32927.

How do we identify such hosts?
We first use a filter: dns && dns.flags.response == 0
This gives us only those packets that use dns as the application level protocol. Further, we are interest in the query request packets (not query responses from the dns server) since we want to check for clients that use static port numbers for dns queries. That is what the above filter returns.

Next, we examine the udp source port number of the returned packets for “Each client” and see if the client is using multiple different port numbers or the same port number at all times. This gives us the 2 clients mentioned above.

Problem 7) TCP connection is expected to use a random Initial Sequence Number (ISN) to protect against TCP hijacking attacks. If the endpoints were to use a deterministic ISN which an adversary can guess, then the adversary can hijack a TCP connection and send malicious packets with a correct sequence number, thus potentially compromising the target.

Almost every host today does not use a deterministic ISN but only some hosts use an ISN that is truly random across multiple connection requests to a given target.

How to compute this?
We first need to obtain the information (source ip address, destination ip address and ISN) for each TCP connection endpoint.

To obtain the TCP endpoint, we can check for TCP packets which have a “SYN” request in them. This is because, TCP is a 3-way handshake protocol and the first step to initiate a new

connection is to send a SYN packet from the source to the destination. This will be followed by a SYN/ACK packet from the destination and then an ACK packet from the source to destination.

We want those packets which have only the “SYN” flag set in order to get the ISN. This is done using the filter: tcp and tcp.flags.syn == 1 and tcp.flags.ack == 0

This gives us SYN-only packets and leaves out SYN/ACK type of packets. Finally, we iterate over all these packets to record the information (source ip, destination ip, ISN).

For each of the tcp endpoints with 5 or more connections between them, we need to identify an endpoint with the broadest coverage which is max(isn_no) - min(isn_no) where isn_no is a list of all ISN’s used by the given endpoint.

This gives us the following TCP endpoint that has the broadest coverage:
('251.215.153.250', '159.79.22.249')

Problem 8)
Traceroute is generally used to find the IP addresses of the routers along the path from the host to the destination. Traceroute utility uses UDP protocol to send a stream of udp packets with increasing ttl values for a given a destination/target. We start with a ttl=1, this packet gets dropped at the first router from the source and returns back a response with destination unreachable message and round trip time. Next the host sends a UDP packet with ttl=2 and this is dropped at the 2nd router along the path. This process continues until we actually reach the desired destination.

In our example, we need to identify a host running “traceroute” by checking for udp packets that do not involve any application layer protocol (like dns) so we accept “pure” udp packets and remove other application layer protocol packets (like dns) that also use udp for transport. This is done using the following filter:

udp and not dns and not browser and not mdns and not cups and not nbns and not auto_rp and not smb_netlogon

Next, we examine the udp packets for a given udp endpoint (src,dest) where the host uses increasing ttl values (increasing by 1 each time). We can find one such host in the trace which sends udp packets to the destination with ttl values ranging from ttl=1 to 64.
The host and the target destination are : Internet Protocol Version 4, Src: **248.86.240.130**, Dst: **159.79.23.208**

Problem 9)
In this question we need to identify a host that is attempting a reflected cross site scripting (XSS) attack.

This can be done using the filter: http.request.uri contains “<script>” and http.request.uri contains "</script>"
In a reflected XSS attack, the attacker sends out Javascript code enclosed in <script> tag to the server via the request uri. The server simply reflects back the script in the returned and this gets rendered on the client’s browser and gets executed.

So we check for http requests whose URI contains a <script> ….</script> tag. The host sending this request is our malicious host attempting the XSS attack.

The host in this case is Src: **251.215.153.250** and the target server is Dst: **159.79.22.249.** Below are snippets from the trace file showing evidence of a XSS attack.

Internet Protocol Version 4, Src: 251.215.153.250, Dst: 159.79.22.249
Evidence of XSS attack:
Some of the traces that show the XSS attack are as shown below:
GET /v9j2h7a7.cgi?<script>document.cookie=%22testhzlg=9267;%22</script> HTTP/1.1\r\n
GET <script>document.cookie=%22testhzlg=9267;%22</script> HTTP/1.1\r\n
GET /pub/bootstrap/?"><script>alert('struts_sa_surl_xss.nasl')</script> HTTP/1.1\r\n
GET /pub/bootstrap?username="<script>foo</script HTTP/1.1\r\n