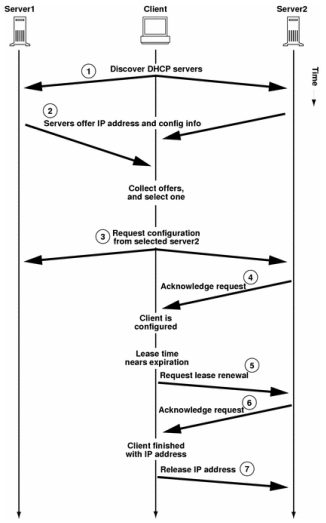
Cybersecurity Fundamentals (CS3308/7308) Assignment 7 Example Answers

Question 1 (4 point) DHCP

1) The 4 messages in a DHCP sequence, Discover, Offer, Request, Acknowledge are each sent as (Layer-2) unicast or broadcast packets?

They are Layer 2 Broadcast, Unicast, Broadcast, Unicast, respectively. Since the requesting host does not have an IP address assigned until the end of Acknowledge, everything happens at Layer 2. First must be broadcast (send to FF:FF:FF:FF:FF:FF) as the client knows no idea where to send packets. The second message is sent just to the requester to its MAC address (say, 11:22:33:44:55:66:77). The third is sent as broadcast, just in case there were multiple DHCP servers making offers.



https://docs.oracle.com/cd/E19455-01/806-5529/6jehkcs2r/index.html

2) Therefore, in a switched network, which of the 4 messages would a malicious sniffer see (without doing active man-in-the-middle)?

In a switched network, in theory, the attacker might just see the broadcasted DISCOVER and REQUEST packets. In reality, many switches have DHCP snooping features in place, and forwards the DISCOVER/REQUEST packets only to the trusted DHCP servers.

3) Which DHCP configuration options (IP, DNS, etc) are attackers interested in forging to be able to sniff and manipulate network traffic? There are mainly 3 that are useful.

The attacker can forge the IP address, DNS, Default Route to do network MITM or DNS forging (in case of just forging DNS). Apologies, the IP address is strictly not an option in the DHCP protocol. Forging the DHCP address is also useful for intercepting the renewal REQUEST.

4) Which DHCP configuration options (IP, DNS, etc) are attackers interested in forging to be able to sniff and manipulate network traffic? There are mainly 3 that are useful.

The DHCP Starvation attack can be used to effectively perform a DoS attack against legitimate DHCP server, to make it easier to send forged DHCP OFFER/ACK to the client.

5) In a successful DHCP spoofing attack, the attacker would end up sending how many messages to the victim host?

It turns out that you only need to forge the final ACK packet (as seen from the Ettercap example in Question 3). Even if the specified parameters (IP, DNS, Router) are different from the one requested in the REQUEST, the client appears to accept it. The attacker could forge both OFFER and ACK as well.

6) True or False? "Rogue DHCP Server" refers to both accidental plugging-in of a DHCP-enabled device (e.g., home routers) and a malicious attack (i.e., DNS Spoofing) with intention to sniff/MITM network traffic

True.

7) Where is "DHCP Snooping" implemented and what does it do? Describe in one sentence

DHCP Snooping is a feature in network switches that prevent rogue DHCP by only forwarding packets from trusted, legitimate DHCP servers. The kind of attack described in this assignment would not work when the feature is turned on.

[Marking guide: Give full 4 marks if all questions are answered correctly. Deduct 0.5 points for any false statements]

Question 2 (2 point) Ettercap 1

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x2a772721
2	0.000265357	10.0.2.3	255.255.255.255	DHCP	590	DHCP Offer - Transaction ID 0x2a772721
3	0.000513493	0.0.0.0	255.255.255.255	DHCP	590	DHCP Discover - Transaction ID 0x2b772721
4	0.000667040	10.0.2.3	255.255.255.255	DHCP	590	DHCP Offer - Transaction ID 0x2b772721
5	0.000915084	0.0.0.0	255.255.255.255	DHCP	590	DHCP Request - Transaction ID 0x2c772721
6	0.011156071	10.0.2.3	255.255.255.255	DHCP	590	DHCP ACK - Transaction ID 0x2c772721

I stole this picture from a student, but this is what you would see in both promiscuous mode and deny mode (switched) mode. This appears to be the way the virtual DHCP server on Virtual Box functions, and would be different on a real switch.

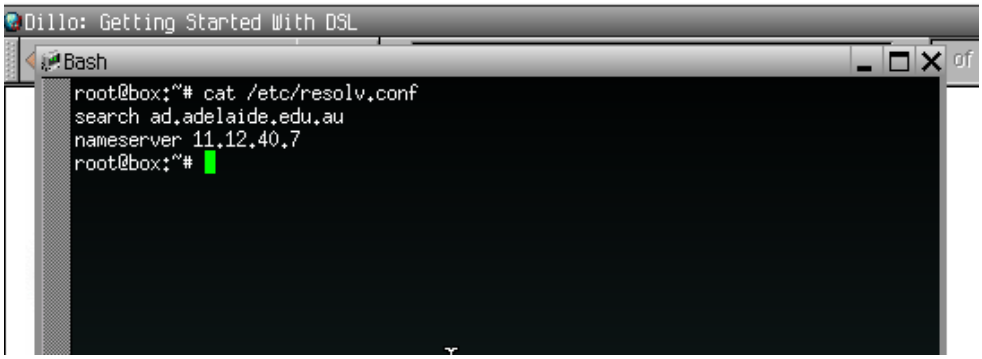
[Marking guide: give full points for similar screenshot(s)]

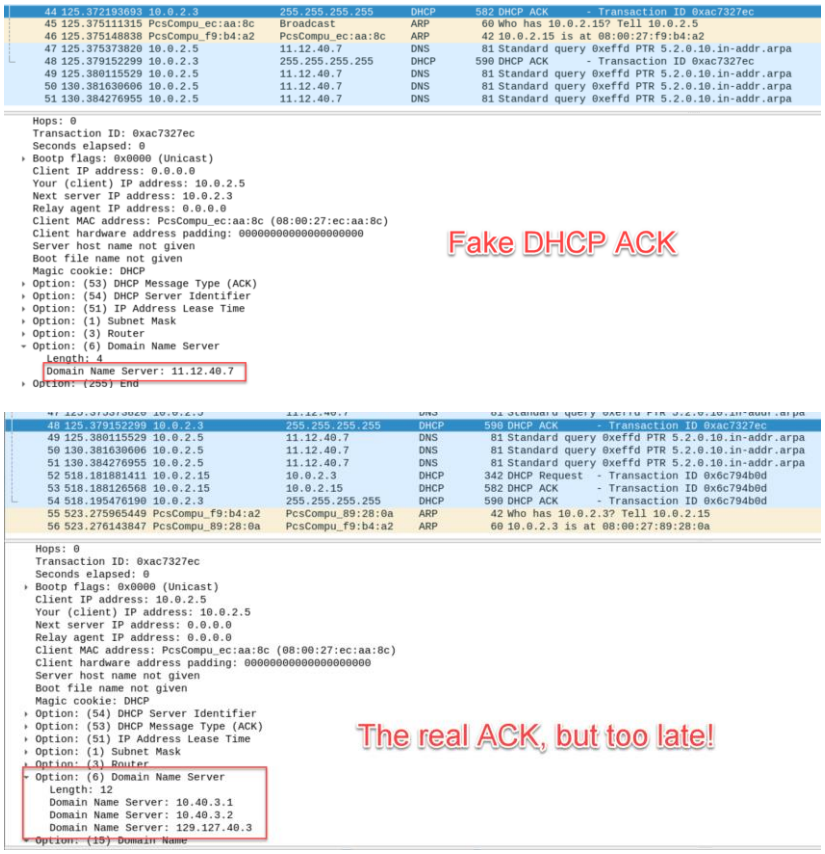
Question 3 (2 point) Ettercap 2

Ettercap seems to listen for the REQUEST message from the client and respond with a forged ACK message faster than the real DHCP. You should see two ACK messages, but the **<u>two are indistinguishable</u>** except for the forged DNS address part.

DSL [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help





Fake DHCP ACK

The real ACK, but too late!

Question 4 (2 point) tcpdump

```
root@kali:~# tcpdump -c 2 -i eth0 -w dns.pcap udp port 53
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 by
es
2 packets captured
2 packets received by filter
0 packets dropped by kernel
[1]+  Done                    wireshark
root@kali:~# hexdump -C dns.pcap
00000000  d4 c3 b2 a1 02 00 04 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 04 00 01 00 00 00  a1 ef dc 5c 21 bc 01 00  |...........\!...|
00000020  45 00 00 00 45 00 00 00  08 00 27 f9 b4 a2 08 00  |E...E.....'.....|
00000030  27 ec aa 8c 08 00 45 00  00 37 f8 ad 40 00 40 11  |'.....E..7..@.@.|
00000040  02 f1 0a 00 02 05 0b 0c  28 07 80 00 00 35 00 23  |........(....5.#|
00000050  b2 b8 ee 04 01 00 00 01  00 00 00 00 00 00 04 62  |...............b|
00000060  6c 61 68 04 62 6c 61 68  00 00 01 00 01 a6 ef dc  |lah.blah........|
00000070  5c a4 c0 01 00 45 00 00  00 45 00 00 00 08 00 27  |\....E...E.....'|
00000080  f9 b4 a2 08 00 27 ec aa  8c 08 00 45 00 00 37 f8  |.....'.....E..7.|
00000090  ae 40 00 40 11 02 f0 0a  00 02 05 0b 0c 28 07 80  |.@.@.........(..|
000000a0  00 00 35 00 23 b2 b8 ee  04 01 00 00 01 00 00 00  |..5.#...........|
000000b0  00 00 00 04 62 6c 61 68  04 62 6c 61 68 00 00 01  |....blah.blah...|
000000c0  00 01                                             |..|
```

Result of doing “ping blah.blah” on DSL.

Question 4 (1 point extra) libpcap

I will have to be honest here, I have not prepared and answer for this question. I will include **Mr Daniel Cotton’s solution** (with his permission) which is very well written and easy to understand.

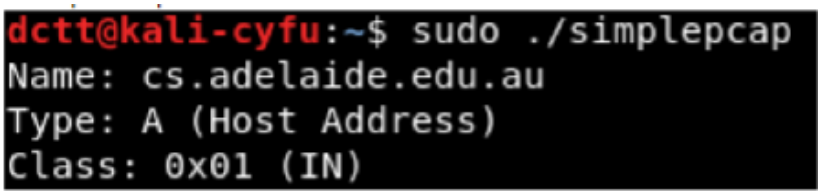
https://gist.github.com/danielcotton/c3fa3876deaee7db08b679fae60c0b34

The key portion is after line 53 as follows:

```
<snip>
if (packet != NULL) {
      // Copy the entire packet into a buffer
      unsigned char* packet_buffer = (char*)malloc((header.len+1) * sizeof(char));
      size_t packet_len = header.len;

      for (size_t i=0; i<header.len; i++) {
        packet_buffer[i] = *(packet + i);
      }
      packet_buffer[header.len] = 0;

      // Ditch the first 14 bytes of the Ethernet frame  (assuming no 802.1Q)
      packet_buffer += 14;
      packet_len -= 14;

      // Read the IPv4 IHL (the second nibble of the first byte)
      // This'll give the number of 32-bit words, so we multiple by 4 (32/8)
      //  to give the header length in bytes;
      int header_len = ((int) *(packet_buffer) & 0xF) * 4;

      // We then ditch the IPv4 header
      packet_buffer += header_len;
      packet_len -= header_len;

      // Ditch the UDP header
      packet_buffer += 8;
      packet_len -= 8;

      // ...and now we get to the DNS info

      // Assuming a DNS query with a single query, we can disregard the first 12 bytes
      packet_buffer += 12;
```

```
      packet_len -= 12;

      // Okay, so we should be up to the query now
      printf("Name: ");
      unsigned int section_len = (unsigned int) *packet_buffer;
      packet_buffer++;

      while (*packet_buffer) { // The name field is null-terminated
        for (int i=0; i<section_len; i++) {
          printf("%c", *packet_buffer);
          packet_buffer++;
        }
        if (*packet_buffer) {
          // new section
          printf(".");
          section_len = (unsigned int) *packet_buffer;
          packet_buffer++;
        }
      }
      packet_buffer++; // Get past the null terminator

      printf("\nType: ");
      unsigned int type = (unsigned int)packet_buffer[1] | (unsigned
int)packet_buffer[0] << 8;

      switch(type) {
        case 1:
          printf("A (Host Address)");
          break;
        case 2:
          printf("NS (Name Server)");
          break;
        case 6:
          printf("SOA (Start of Authority)");
          break;
        case 15:
          printf("MX (Mail Exchange)");
          break;
        default:
          // there's a whole bunch of these, I'm just covering the most popular ones
          printf("%d, see IANA", type);
      }
      printf("\n");
      packet_buffer += 2;

      printf("Class: ");
      unsigned int class = (unsigned int)packet_buffer[1] | (unsigned
int)packet_buffer[0] << 8;

      switch(class) {
        case 1:
          printf("0x01 (IN)");
          break;
        default:
          printf("%02x (Not IN)", class); // most of the alternatives aren't relevant
for a UDPv4 DNS packet
      }
      printf("\n");
    }
```

When run, it produces a result like this:



[Marking guide: any reasonable effort to change the example code to capture DNS requests and dump the content of request is OK].