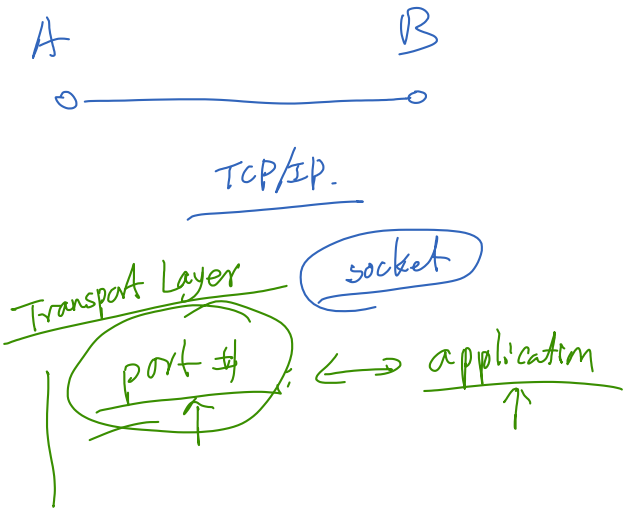
Internet Security

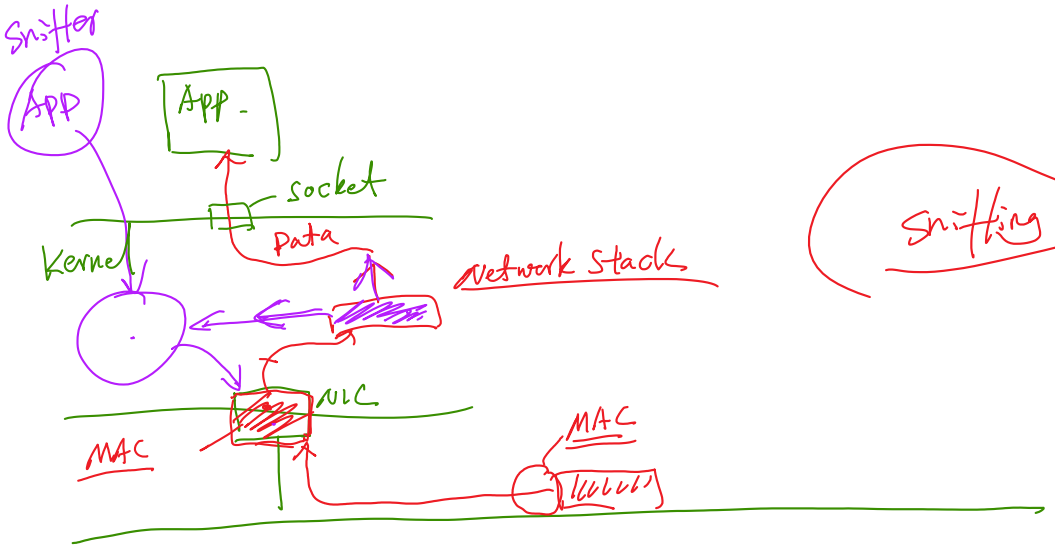
Packet Sniffing and Spoofing

Socket Programming: Receiving Packets

```c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

void main()
{
    struct sockaddr_in server;
    struct sockaddr_in client;
    int clientlen;
    char buf[1500];

    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    memset((char *) &server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = htons(9090);

    if (bind(sock, (struct sockaddr *) &server, sizeof(server)) < 0)
        error("ERROR on binding");

    while (1) {
        bzero(buf, 1500);
        recvfrom(sock, buf, 1500-1, 0, (struct sockaddr *) &client, &clientlen);
        printf("%s\n", buf);
    }
    close(sock);
}
```



How Packets Are Received and Packet Sniffing



promiscuous mode

Packet Capturing Using Raw Socket

```
#include <sys/socket.h>
#include <linux/if_packet.h>
#include <net/ethernet.h>
#include <stdio.h>

int main() {
   int PACKET_LEN = 512;
   char buffer[PACKET_LEN];
   struct sockaddr saddr;
   struct packet_mreq mr;

   // Create the raw socket
   int sock = socket(AF_PACKET , SOCK_RAW , htons(ETH_P_ALL)) ;

   // Turn on the promiscuous mode.
   mr.mr_type = PACKET_MR_PROMISC;
   setsockopt(sock, SOL_PACKET, PACKET_ADD_MEMBERSHIP, &mr, sizeof(mr));

   // Getting captured packets
   while (1) {
      int data_size=recvfrom(sock, buffer, PACKET_LEN, 0, &saddr,
                     (socklen_t*)sizeof(saddr));
      if(data_size)
         printf("Got one packet\n");
   }

   close(sock);
   return 0;
}
```

Raw Socket

→ root or special capability.

Capture Packets Using PCAP API

❖ **Set up the packet-capturing logic.**

```c
int main()
{
  pcap_t *handle;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct bpf_program fp;
  //char filter_exp[] = "port 23";
  char filter_exp[] = "";
  bpf_u_int32 net;

  //Open live pcap session on NIC with name eth0
  handle = pcap_open_live("eth18", BUFSIZ, 1, 1000, errbuf);

  //Compile filter_exp into BPF psuedo-code
  pcap_compile(handle, &fp, filter_exp, 0, net);

  pcap_setfilter(handle, &fp); //Setup BPF code on the socket
  pcap_loop(handle, -1, got_packet, NULL); //Capture packets
  pcap_close(handle);  //Close the handle
  return 0;
}
```

❖ **Get a packet and process it.**

```c
/*************************************************************************
  This function will be invoked by pcap, whenever a packet is captured.
**************************************************************************/
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct ethheader *eth = (struct ethheader *)packet;
    if (eth->ether_type != ntohs(0x0800))  return; // not an IP packet

    struct ipheader* ip = (struct ipheader*)(packet + SIZE_ETHERNET);
    int ip_header_len = ip->iph_ihl * 4;

    printf("-----------------------------------\n");
    /* print source and destination IP addresses */
    printf("       From: %s\n", inet_ntoa(ip->iph_sourceip));
    printf("         To: %s\n", inet_ntoa(ip->iph_destip));

    /* determine protocol */
    if (ip->iph_protocol ==  IPPROTO_ICMP){
         printf("   Protocol: ICMP\n");
         spoof_icmp_reply(ip);
    }
}
```

Case Study: Wireshark

```
$ pgrep wireshark
7598
$ ps  -fp 7598
UID   PID   PPID  C STIME TTY   TIME     CMD
seed  7598     1  0 10:01 ?    00:00:01 /usr/bin/wireshark
```

not Set-UID

```
$ pstree -p 7598
wireshark(7598) ____ dumpcap(7919)
                |__ {wireshark}(7601)
                |__ {wireshark}(7602)
```

wireshark → dumpcap

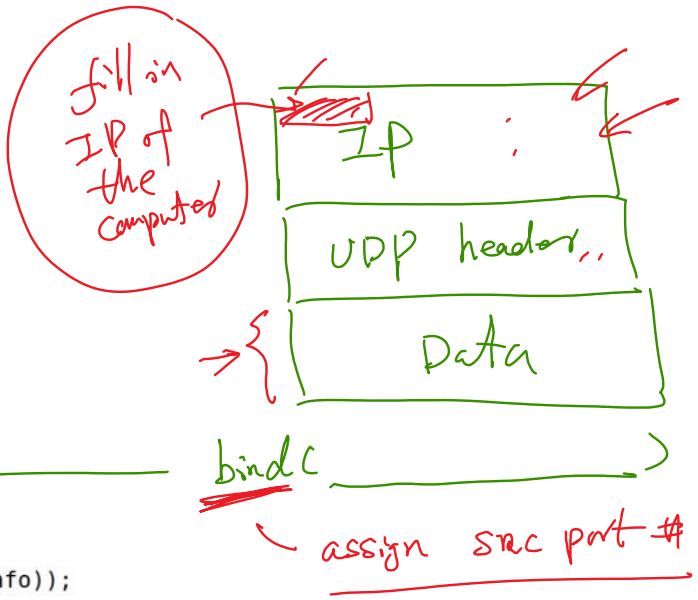
```
$ which dumpcap
/usr/bin/dumpcap
$ ls -l /usr/bin/dumpcap
-rwxr-xr-- 1 root wireshark 66884 Apr 12  2012 /usr/bin/dumpcap
$ getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_admin,cap_net_raw+eip
$ getcap /usr/bin/wireshark
$
```

raw socket

```
$ ls -l /usr/bin/wireshark
-rwxr-xr-x 1 root root 2004552 Apr 12  2012 /usr/bin/wireshark
$ ls -l /usr/bin/dumpcap
-rwxr-xr-- 1 root wireshark 66884 Apr 12  2012 /usr/bin/dumpcap
```

Packet Sending

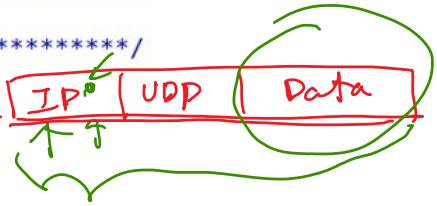
```c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

void main()
{
    struct sockaddr_in dest_info;
    char *data = "Hello Server.\n";

    // Create a network socket.
    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    // Provide needed information about destination.
    memset((char *) &dest_info, 0, sizeof(dest_info));
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr.s_addr = inet_addr("10.0.2.5");
    dest_info.sin_port = htons(9090);

    // Send the packet out.
    sendto(sock, data, strlen(data), 0,
                (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```



Packet Spoofing

Spoofing IP Packet: Code

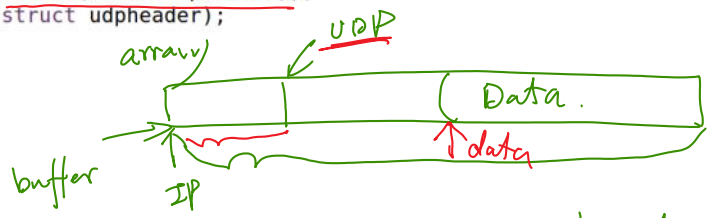
```c
/*****************************************************************************
  Given an IP packet, send it out using raw socket.
*****************************************************************************/
void send_raw_ip_packet(struct ipheader* ip)
{
    struct sockaddr_in dest_info;
    int enable = 1;

    // Create a raw network socket, and set its options.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));

    // Provide needed information about destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    // Send the packet out.
    printf("Sending spoofed IP packet...\n");
    sendto(sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```



Constructing Raw Packets

❖ Type casting

```
char buffer[1500];
memset(buffer, 0, 1500);

struct ipheader *ip = (struct ipheader *) buffer;
struct udpheader *udp = (struct udpheader *) (buffer + sizeof(struct ipheader));
char *data = buffer + sizeof(struct ipheader) + sizeof(struct udpheader);
```



❖ Fill in data

```
/* IP Header */
struct ipheader {
        unsigned char      iph_ihl:4, iph_ver:4; //IP Header length & Version.
        unsigned char      iph_tos; //Type of service
        unsigned short int iph_len; //IP Packet length (Both data and header)
        unsigned short int iph_ident; //Identification
        unsigned short int iph_flag:3, iph_offset:13; //Flags and Fragmentation offset
        unsigned char      iph_ttl; //Time to Live
        unsigned char      iph_protocol; //Type of the upper-level protocol
        unsigned short int iph_chksum; //IP datagram checksum
        struct  in_addr    iph_sourceip; //IP Source address (In network byte order)
        struct  in_addr    iph_destip;//IP Destination address (In network byte order)
};
```

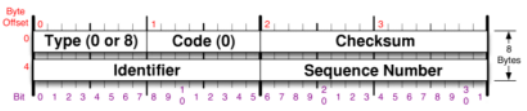
✓ ip → iph_ttl = 50

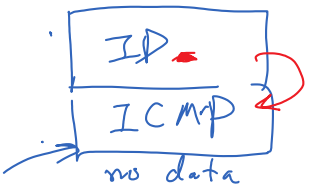
✗ buffer → iph_ttl = 50

Spoofing ICMP Echo Request

❖ Step 1: Fill in the ICMP header.

```
/*************************************************************
  Spoof an ICMP echo request using an arbitrary source IP Address
*************************************************************/
int main() {
   char buffer[PACKET_LEN];

   memset(buffer, 0, PACKET_LEN);

   /*********************************************************
      Step 1: Fill in the ICMP header.
    *********************************************************/
   struct icmpheader *icmp = (struct icmpheader *) (buffer + sizeof(struct ipheader));
   icmp->icmp_type = 8; //ICMP Type: 8 is request, 0 is reply.

   // Calculate the checksum for integrity
   icmp->icmp_chksum = 0;
   icmp->icmp_chksum = in_cksum((unsigned short *)icmp, sizeof(struct icmpheader));
```



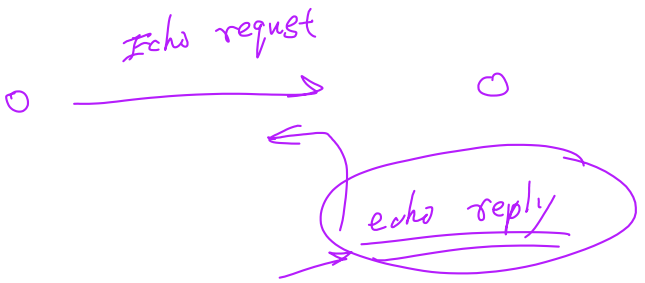


❖ Step 2: Fill in the IP header.

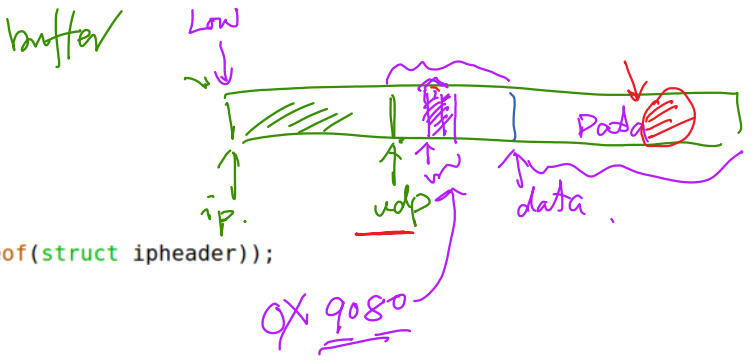
```
   /*********************************************************
      Step 2: Fill in the IP header.
    *********************************************************/
   struct ipheader *ip = (struct ipheader *) buffer;
   ip->iph_ver = 4;
   ip->iph_ihl = 5;
   ip->iph_ttl = 20;
   ip->iph_sourceip.s_addr = inet_addr(SRC_IP);
   ip->iph_destip.s_addr = inet_addr(DEST_IP);
   ip->iph_protocol = IPPROTO_ICMP; // The value is 1, representing ICMP.
   ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));

   // No need to set the following fileds, as they will be set by the system.
   // ip->iph_chksum = ...
```

❖ Step 3: Send the raw IP packet.

```
   /*********************************************************
      Step 3: Finally, send the spoofed packet
    *********************************************************/
   send_raw_ip_packet (ip);
```

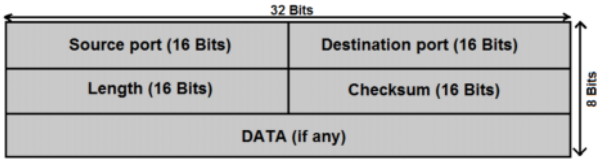


Spoofing UDP Packet



❖ The code

```
char buffer[PACKET_LEN];

memset(buffer, 0, PACKET_LEN);
struct ipheader *ip = (struct ipheader *) buffer;
struct udpheader *udp = (struct udpheader *) (buffer + sizeof(struct ipheader));

/*********************************************************
   Step 1: Fill in the UDP data field.
 *********************************************************/
char *data = buffer + sizeof(struct ipheader) + sizeof(struct udpheader);
const char *msg = "Hello UDP\n";
int data_len = strlen(msg);
strncpy (data, msg, data_len);


/*********************************************************
   Step 2: Fill in the UDP header.
 *********************************************************/
udp->udp_sport = htons(SRC_PORT);
udp->udp_dport = htons(DEST_PORT);
udp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
udp->udp_sum =  0; // Many OSes ignore this field, so we will not caculate it.
```



❖ Test it

- On another machine (e.g., 10.0.2.16)

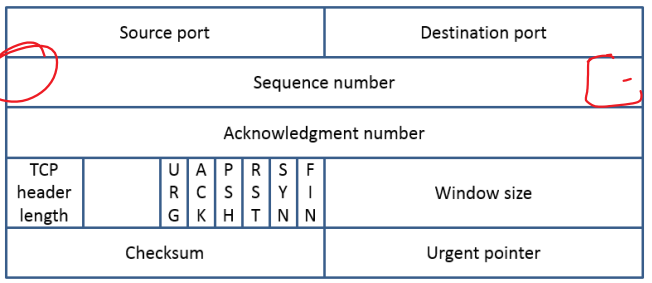
  ```
  $ nc -l -u -v 9090
  ```

- Send a spoofed UDP packet to 10.0.2.16:9090

Spoofing TCP Packet

❖ Construct TCP data and header.

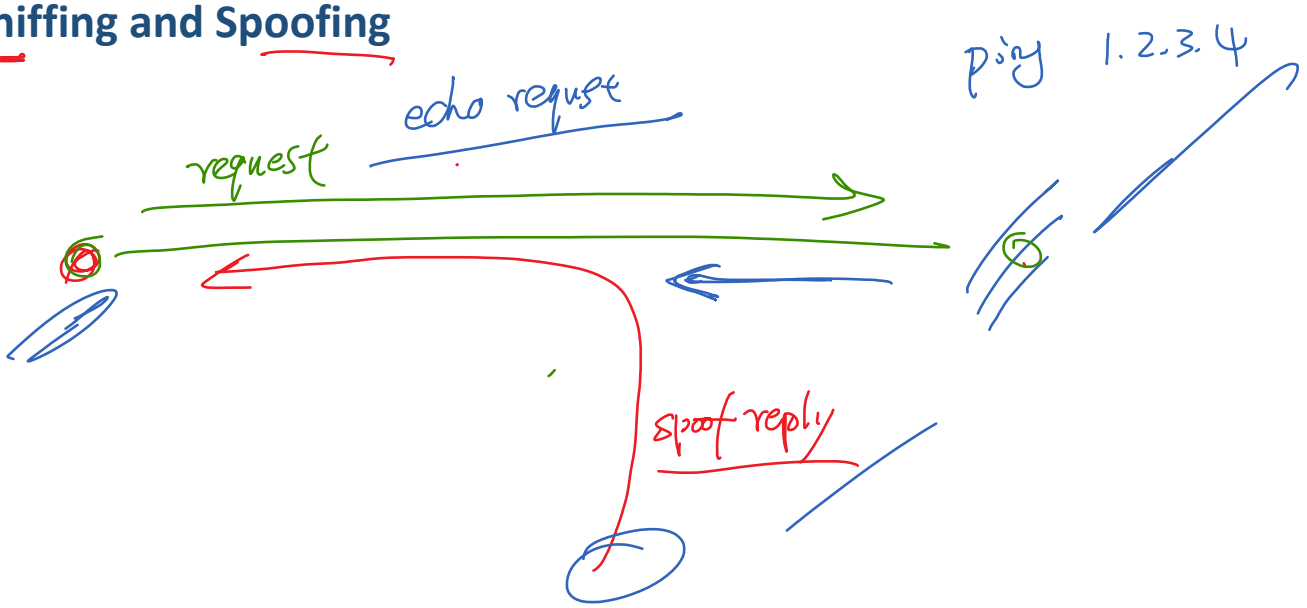
```
int main() {
   char buffer[PACKET_LEN];

   srand(time(0)); // We need to use random numbers for some attacks

   memset(buffer, 0, PACKET_LEN);

   struct ipheader *ip = (struct ipheader *) buffer;
   struct tcpheader *tcp = (struct tcpheader *) (buffer + sizeof(struct ipheader));

   /*********************************************************
      Step 1: Fill in the TCP data field.
    *********************************************************/
   char *data = buffer + sizeof(struct ipheader) + sizeof(struct tcpheader);
   const char *msg = TCP_DATA;
   int data_len = strlen(msg);
   strncpy (data, msg, data_len);


   /*********************************************************
      Step 2: Fill in the TCP header.
    *********************************************************/
   tcp->tcp_sport = htons(SRC_PORT);
   tcp->tcp_dport = htons(DEST_PORT);
   tcp->tcp_seq = htonl(SEQ_NUM);
   tcp->tcp_offx2 = 0x50;
   tcp->tcp_flags = 0x00;
   tcp->tcp_win =  htons(20000);
   tcp->tcp_sum =  0;
```



❖ Construct IP header and compute TCP checksum.

```
   /*********************************************************
      Step 3: Fill in the IP header.
    *********************************************************/
   ip->iph_ver = 4;   // Version (IPV4)
   ip->iph_ihl = 5;   // Header length
   ip->iph_ttl = 20;  // Time to live
   //  ip->iph_sourceip.s_addr = rand(); // Use a random IP address
   ip->iph_sourceip.s_addr = inet_addr(SRC_IP); // Source IP
   ip->iph_destip.s_addr = inet_addr(DEST_IP);  // Dest IP
   ip->iph_protocol = IPPROTO_TCP; // The value is 6.
   ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct tcpheader) + data_len);

   // Calculate tcp checksum here, as the checksum includes some part of the IP header
   tcp->tcp_sum = calculate_tcp_checksum(ip, data_len);

   // No need to fill in the following fileds, as they will be set by the system.
   // ip->iph_chksum = ...
```

Snoofing: Sniffing and Spoofing



Snoofing UDP Communication

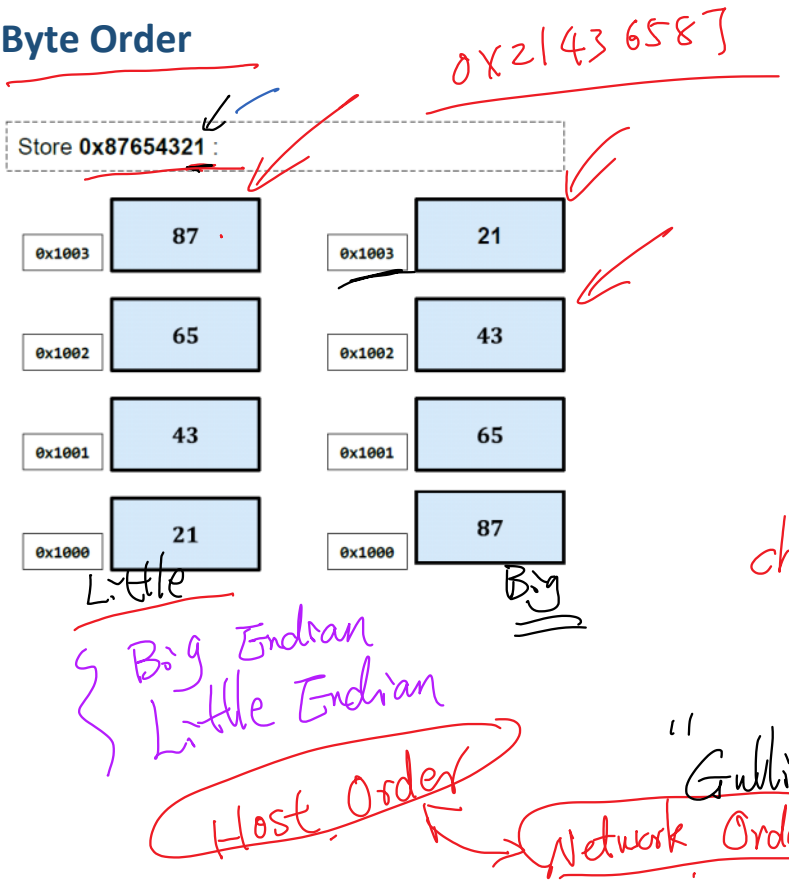
❖ Sniffing UDP packet

```
/********************************************************************
  This function will be invoked by pcap, whenever a packet is captured.
********************************************************************/
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct ethheader *eth = (struct ethheader *)packet;
    if (eth->ether_type != ntohs(0x0800))  return; // not an IP packet

    struct ipheader* ip = (struct ipheader*)(packet + SIZE_ETHERNET);
    int ip_header_len = ip->iph_ihl * 4;

    printf("-----------------------------------\n");
    /* print source and destination IP addresses */
    printf("       From: %s\n", inet_ntoa(ip->iph_sourceip));
    printf("         To: %s\n", inet_ntoa(ip->iph_destip));

    /* determine protocol */
    if (ip->iph_protocol ==  IPPROTO_UDP){
        printf("   Protocol: UDP\n");
        spoof_reply(ip);
    }
}
```

❖ Spoofing UDP reply

```
/********************************************************************
  Given a captured IP packet, construct a spoofed response packet.
********************************************************************/
void spoof_reply(struct ipheader* ip)
{
    int ip_header_len = ip->iph_ihl * 4;
    const char buffer[BUFSIZE];

    struct udpheader* udp = (struct udpheader *) ((u_char *)ip + ip_header_len);
    printf("   SRC Port: %d\n", ntohs(udp->udp_sport));
    printf("   DST Port: %d\n", ntohs(udp->udp_dport));
    if (ntohs(udp->udp_dport) != 9999) {
        return;
    }

    // make a copy from the original packet
    memset((char*)buffer, 0, BUFSIZE);
    memcpy((char*)buffer, ip, ntohs(ip->iph_len));
    struct ipheader * newip = (struct ipheader *) buffer;
    struct udpheader * newudp = (struct udpheader *) (buffer + ip_header_len);
    char *data = (char *)newudp + sizeof(struct udpheader);

    // Construct the UDP payload, keep track of payload size
    const char *msg = "This is a spoofed reply!\n";
    int data_len = strlen(msg);
    strncpy (data, msg, data_len);

    // Construct the UDP Header
    newudp->udp_sport = udp->udp_dport;
    newudp->udp_dport = udp->udp_sport;
    newudp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
    newudp->udp_sum =  0;

    // Construct IP header
    newip->iph_sourceip = ip->iph_destip;
    newip->iph_destip = ip->iph_sourceip;
    newip->iph_ttl = 50;
    newip->iph_len = htons(sizeof(struct ipheader) +
                     sizeof(struct udpheader) + data_len);

    // Send out the spoofed IP packet
    send_raw_ip_packet(newip);
}
```

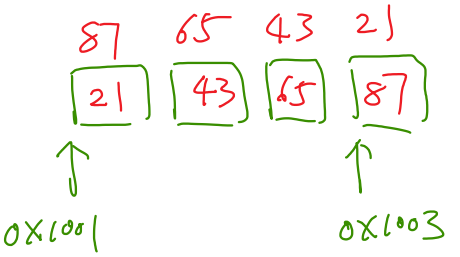
Byte Order



Big Endian
Little Endian

Host Order ↔ Network Order

int 4 bytes



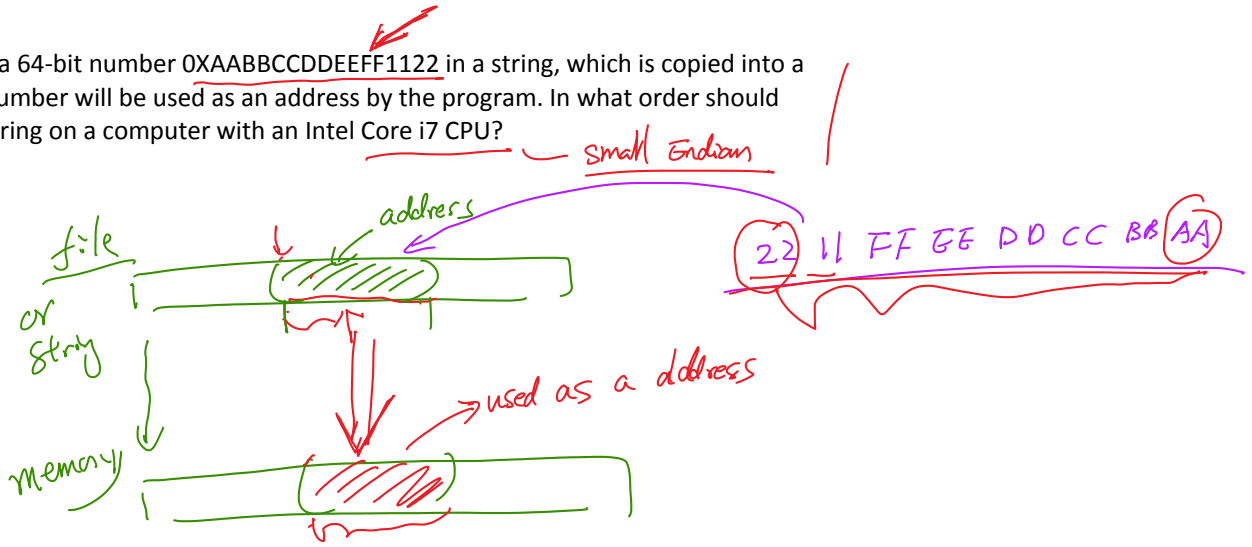
char a[] = 21 43 65 87

⇓ integer, address

"Gulliver's Travel" by Jonathan Swift

Exercise

Question: We need to save a 64-bit number 0XAABBCCDDEEFF1122 in a string, which is copied into a buffer by a program. This number will be used as an address by the program. In what order should we place this number in a string on a computer with an Intel Core i7 CPU?



Byte-Order Conversion

Macro	Description
htons()	Convert unsigned short integer from host order to network order.
htonl()	Convert unsigned integer from host order to network order.
ntohs()	Convert unsigned short integer from network order to host order.
ntohl()	Convert unsigned integer from network order to host order.

h to n

n to h

Checksum

```
unsigned short in_cksum(unsigned short *buf,int length)
{
      unsigned short *w = buf;
      int nleft = length;
      int sum = 0;
      unsigned short temp=0;

      /*
      * The algorithm uses a 32 bit accumulator (sum), adds
      * sequential 16 bit words to it, and at the end, folds back all the
      * carry bits from the top 16 bits into the lower 16 bits.
      */
      while (nleft > 1) {
            sum += *w++;
            nleft -= 2;
      }

      /* treat the odd byte at the end, if any */
      if (nleft == 1) {
            *(u_char *)(&temp) = *(u_char *)w ;
            sum += temp;
      }

      /* add back carry outs from top 16 bits to low 16 bits */
      sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
      sum += (sum >> 16);             // add carry
      return (unsigned short)(~sum);
}
```