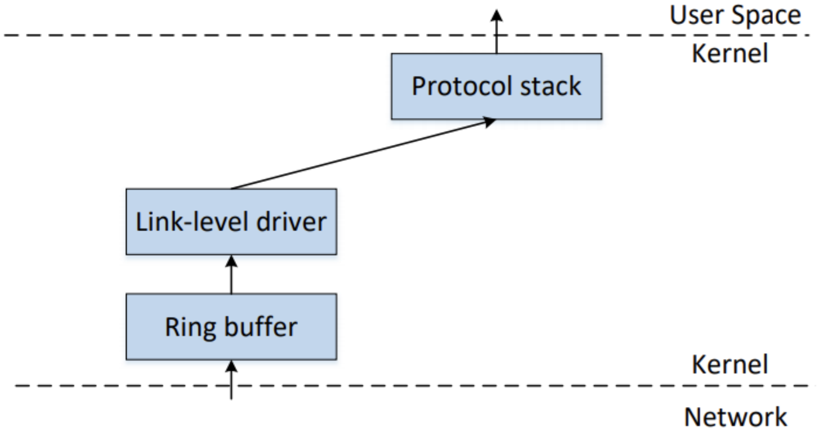
CS915/435 Advanced Computer Security - Network Security (I)

Packet Sniffing and Spoofing

Outline

- Packet Sniffing and Spoofing
    - How packets are received
    - Packet sniffing
    - Packet spoofing
    - Packet sniffing then spoofing
- TCP attacks
- Firewall
- DNS attacks

How Packets Are Received



- Machines are connected to networks through Network Interface Cards
- Each NIC has a MAC address
- Every NIC on the network will hear all the frames on the wire
- If a match is found (destination address in the header), the frame is copied into a buffer and dispatched to user-space programs.

Promiscuous Mode

- The frames that are not destined to a given NIC are discarded
- When operating in promiscuous mode, NIC passes every frame received from the network to the kernel
- If a sniffer program is registered with the kernel, it will be able to see all the packets.
- Normally, enabling Promiscuous Mode requires elevated privilege (root)

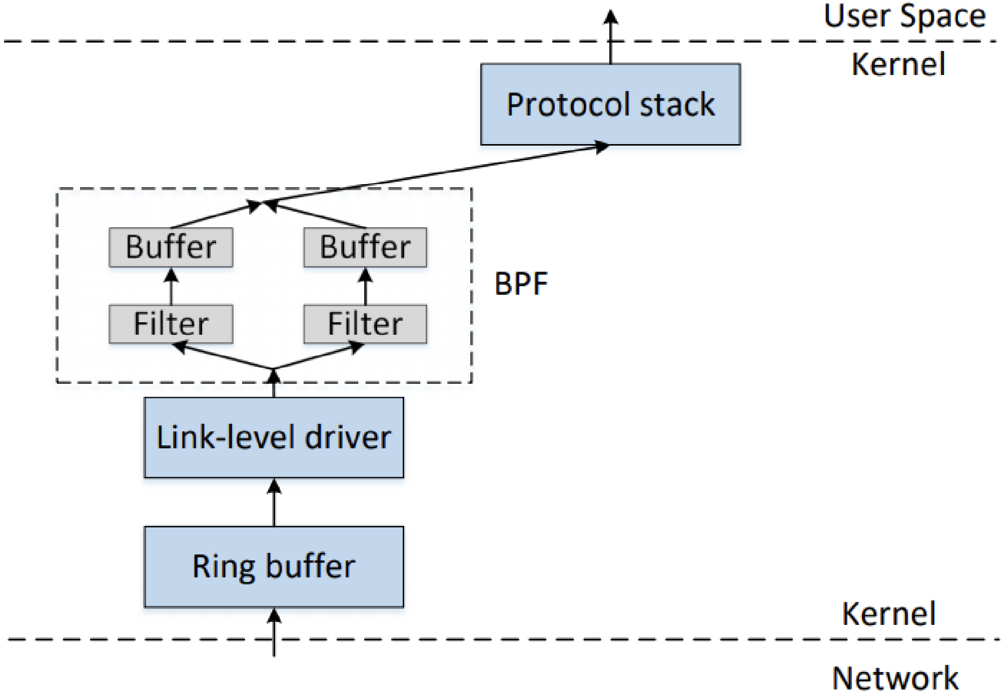
Wireless network

- Promiscuous mode also available in wireless network
- In Wi-Fi, it is called **Monitor Mode**
- However, due to interference, Wi-Fi card can’t copy everything (too much data in the air).
- Wi-Fi cards work with different channels, or slices of the spectrum
- If a Wi-Fi card is put in the monitor mode, they will capture 802.11 frames on the channel that they are listening to.
- Hence, you may miss getting information in Monitor Mode if you’re on a different channel.

How do NIC cards copy sniffed data to buffers?

- Normally sniffers are only interested in certain types of packets, e.g., TCP, DNS queries
- The system can give all the captured packets to the sniffer program, who will discard unwanted packets.
- But this is rather inefficient.
- It is better to filter unwanted packets as early as possible.

BSD Packet Filter (BPF)



- BPF allows a user-program to attach a filter to the socket, which tells the kernel to discard unwanted packets.
- For example, a filter allows only packets on port 22.
- It is possible to have a combination of filters.

Packet Sniffing

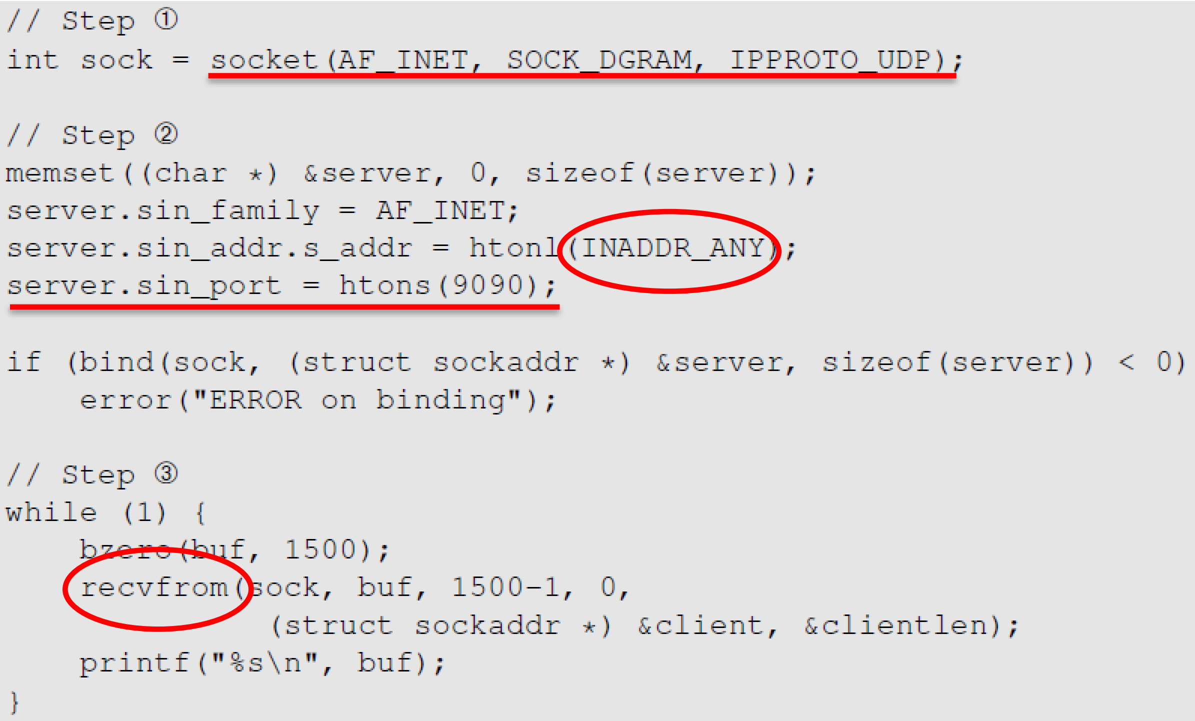
- Packet sniffing describes the process of capturing live data as they flow across a network.
- Packet sniffing tools are also called packet sniffers.
- Let’s first see how computers receive packets.

Receiving Packets Using Socket (UDP server)

Create the socket

Provide information about server

Receive packets

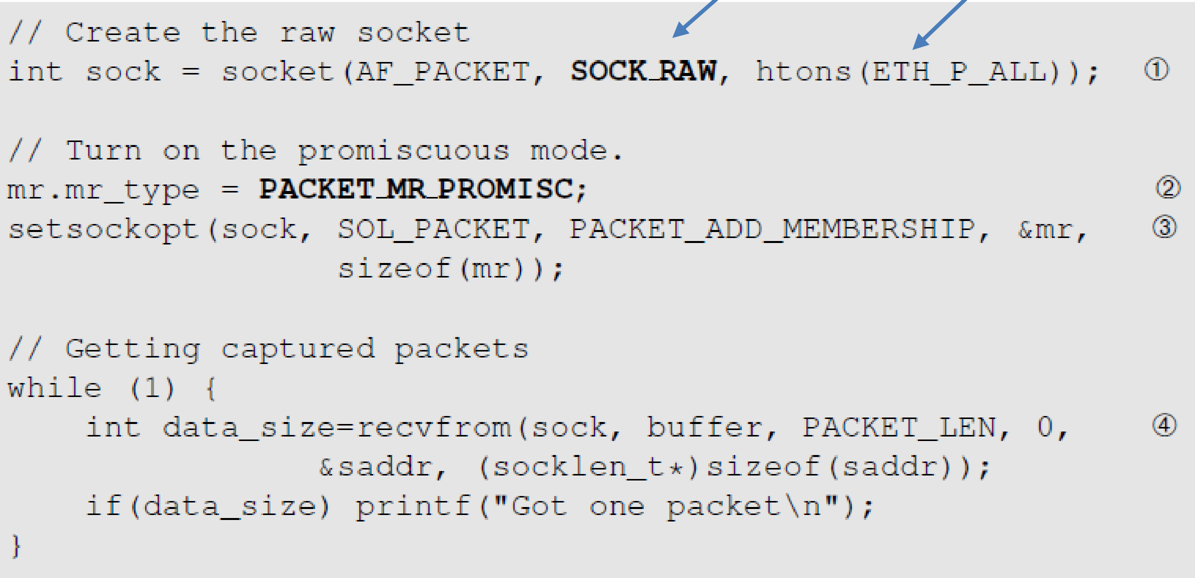


```
// Step ①
int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

// Step ②
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");

// Step ③
while (1) {
    bzero(buf, 1500);
    recvfrom(sock, buf, 1500-1, 0,
                (struct sockaddr *) &client, &clientlen);
    printf("%s\n", buf);
}
```

Receiving Packets Using Raw Socket (sniffer)

Creating a raw socket

Capture all types of packets



```
// Create the raw socket
int sock = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));   ①

// Turn on the promiscuous mode.
mr.mr_type = PACKET_MR_PROMISC;                              ②
setsockopt(sock, SOL_PACKET, PACKET_ADD_MEMBERSHIP, &mr,     ③
                 sizeof(mr));

// Getting captured packets
while (1) {
    int data_size=recvfrom(sock, buffer, PACKET_LEN, 0,      ④
                &saddr, (socklen_t*)sizeof(saddr));
    if(data_size) printf("Got one packet\n");
}
```

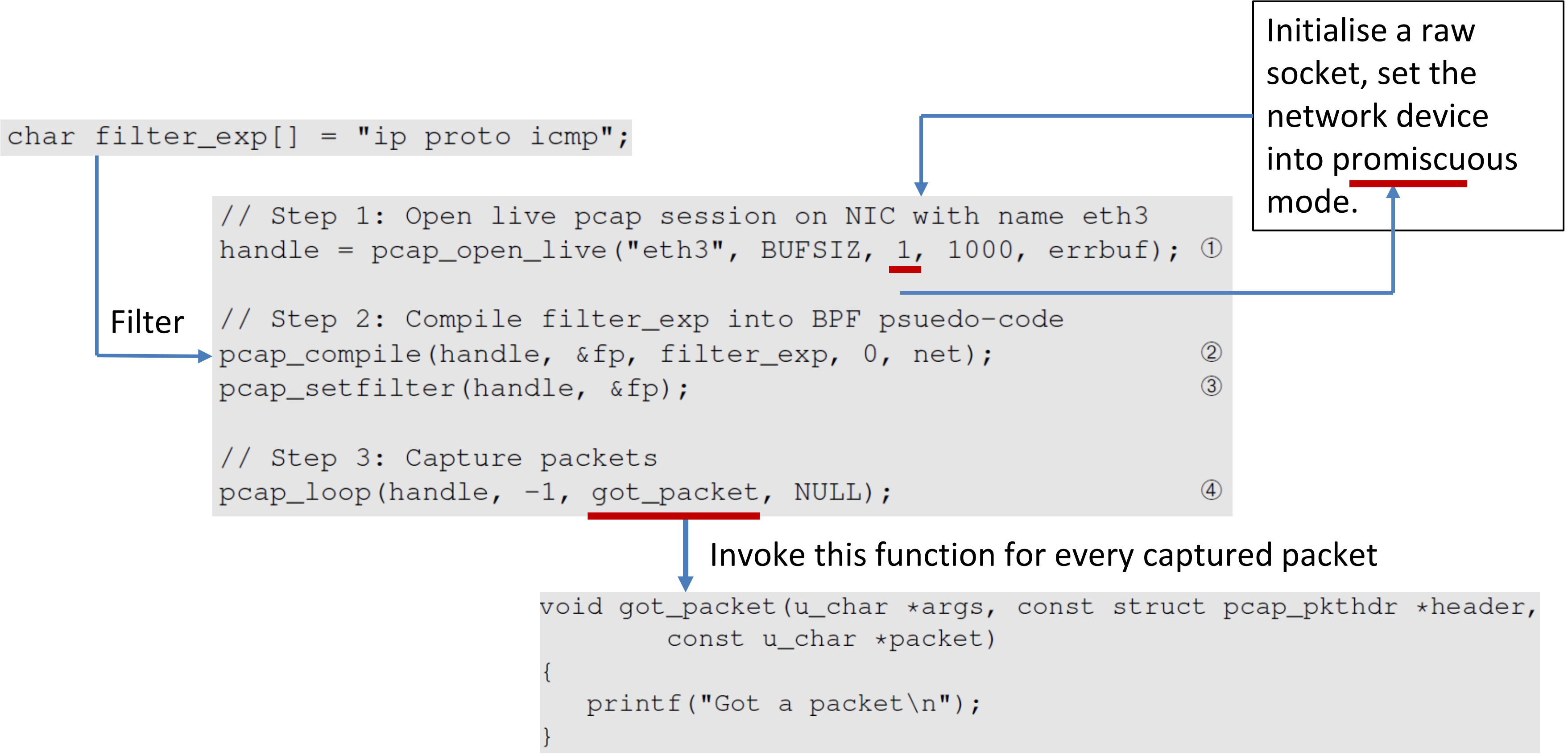
Enable the promiscuous mode

Wait for packets

Limitation of the Approach

- This program is not portable across different operating systems.
- Setting filters is not easy (not included in the code examples)
- The program does not explore any optimisation to improve performance.
- The PCAP (packet capture) library was thus created.
  - It still uses raw sockets internally, but its API is standard across all platforms. OS specifics are hidden by PCAP’s implementation.
  - Allows programmers to specify filtering rules using human readable Boolean expressions.

Packet sniffing using PCAP API



```
char filter_exp[] = "ip proto icmp";
```

Filter

```
// Step 1: Open live pcap session on NIC with name eth3
handle = pcap_open_live("eth3", BUFSIZ, 1, 1000, errbuf); ①

// Step 2: Compile filter_exp into BPF psuedo-code
pcap_compile(handle, &fp, filter_exp, 0, net);             ②
pcap_setfilter(handle, &fp);                               ③

// Step 3: Capture packets
pcap_loop(handle, -1, got_packet, NULL);                   ④
```

Initialise a raw socket, set the network device into promiscuous mode.

Invoke this function for every captured packet

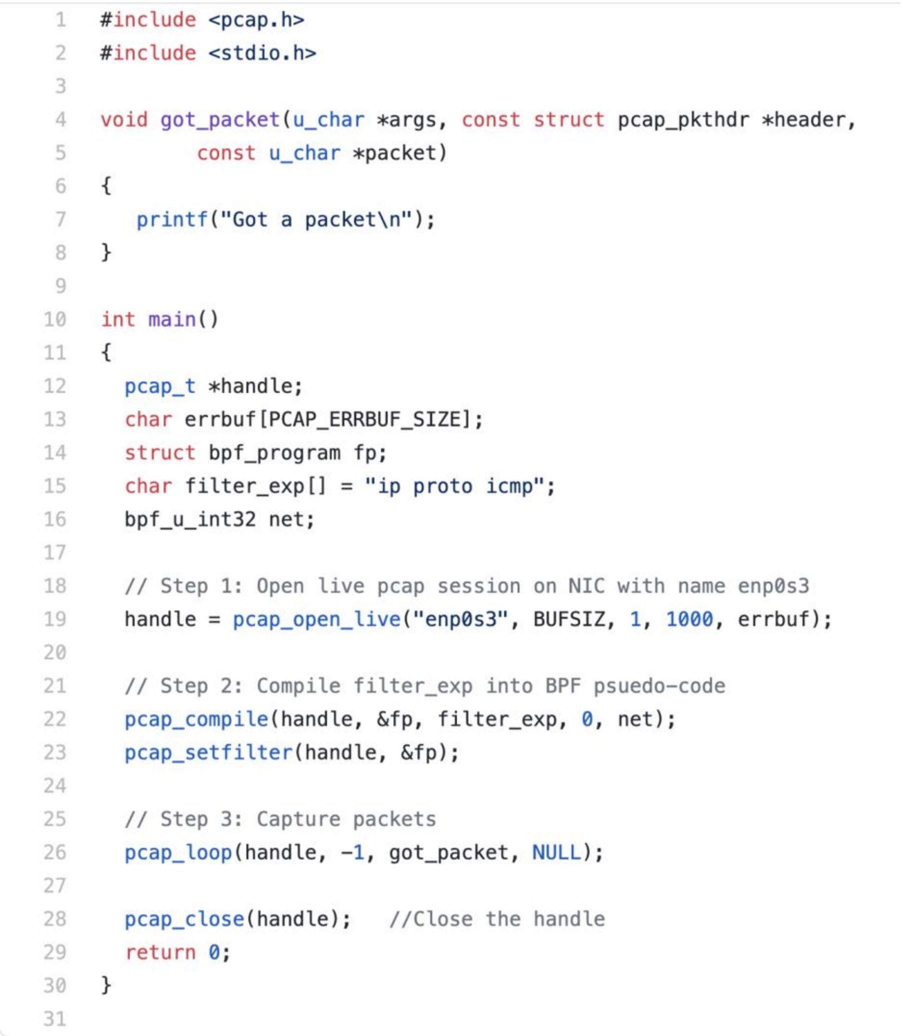
```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
        const u_char *packet)
{
   printf("Got a packet\n");
}
```

A simple sniffer (sniff.c)

https://github.com/kevin-w-du/BookCode/blob/master/Sniffing_Spoofing/C_sniff/sniff.c

```
$ gcc –o sniff sniff.c -lpcap
$ sudo ./sniff
```

Note: root privilege required

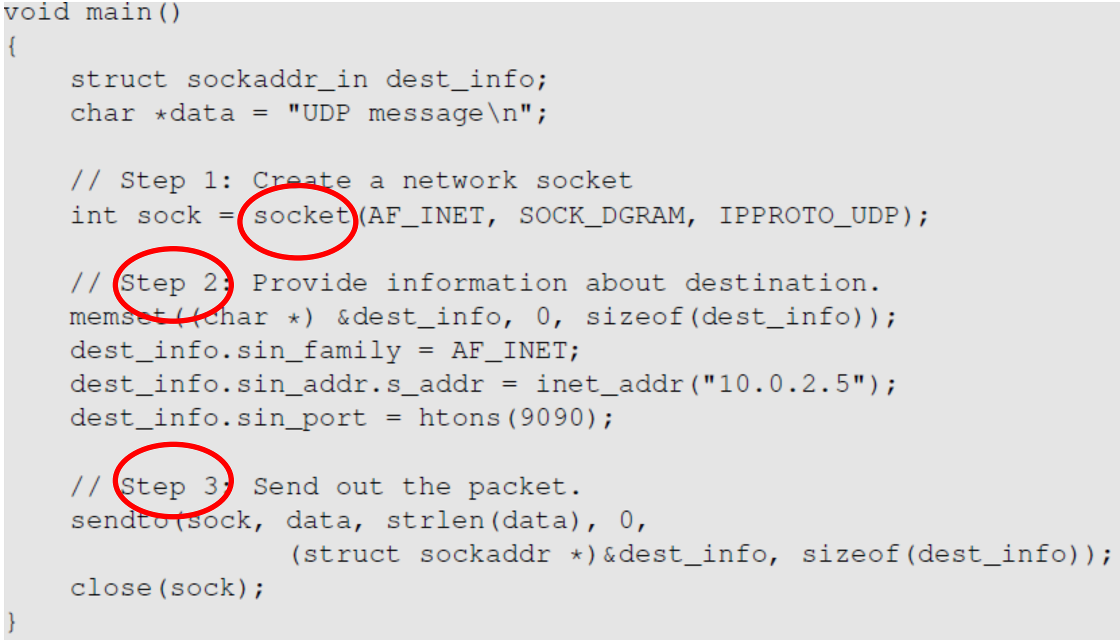


```c
#include <pcap.h>
#include <stdio.h>

void got_packet(u_char *args, const struct pcap_pkthdr *header,
        const u_char *packet)
{
   printf("Got a packet\n");
}

int main()
{
  pcap_t *handle;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct bpf_program fp;
  char filter_exp[] = "ip proto icmp";
  bpf_u_int32 net;

  // Step 1: Open live pcap session on NIC with name enp0s3
  handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);

  // Step 2: Compile filter_exp into BPF psuedo-code
  pcap_compile(handle, &fp, filter_exp, 0, net);
  pcap_setfilter(handle, &fp);

  // Step 3: Capture packets
  pcap_loop(handle, -1, got_packet, NULL);

  pcap_close(handle);   //Close the handle
  return 0;
}

```

Packet Spoofing

- When some critical information in the packet is forged, we refer to it as packet spoofing.
- Many network attacks rely on packet spoofing.
- Let’s see how to send packets without spoofing.

Sending Packets Without Spoofing



```
void main()
{
    struct sockaddr_in dest_info;
    char *data = "UDP message\n";

    // Step 1: Create a network socket
    int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    // Step 2: Provide 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);

    // Step 3: Send out the packet.
    sendto(sock, data, strlen(data), 0,
                 (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```

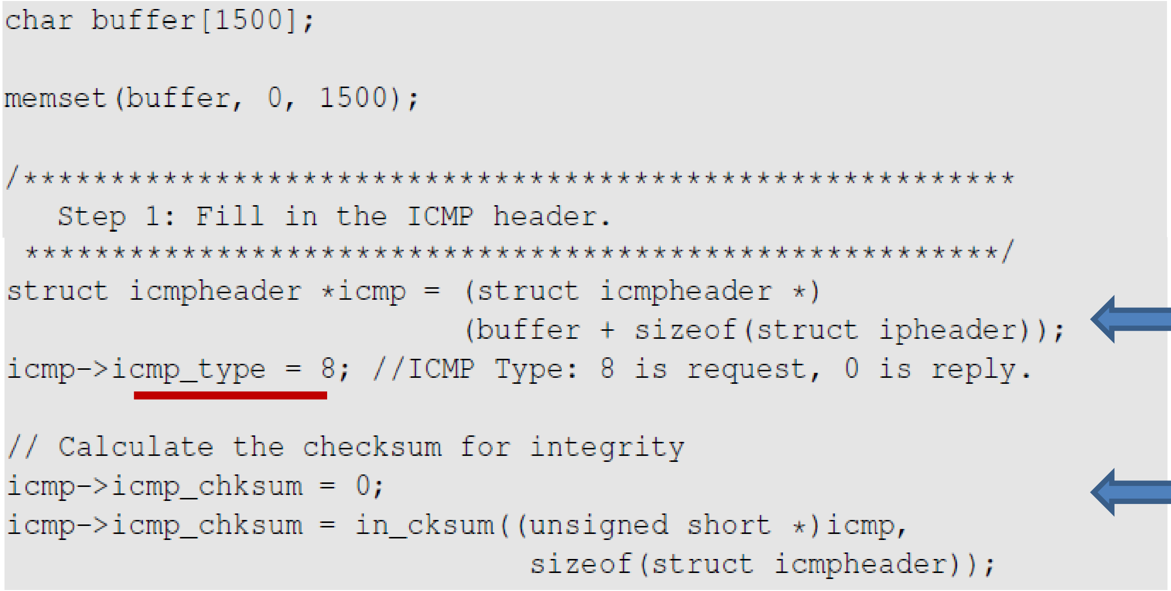
- The first step creates a socket
- The second step provides the destination information
- In the final step, sendto() is used to send out the UDP packet with the provided payload

Spoofing Packets Using Raw Sockets

- However, using the typical socket doesn’t give us much control over the header fields.
- The header fields are set by the OS, e.g., source IP, packet length etc
- We can use a special type of socket called **raw socket**
- With raw socket, we construct the entire packet in a buffer including the IP header before sending it out via the socket

An example: spoofing ICMP packet

Fill in the ICMP Header



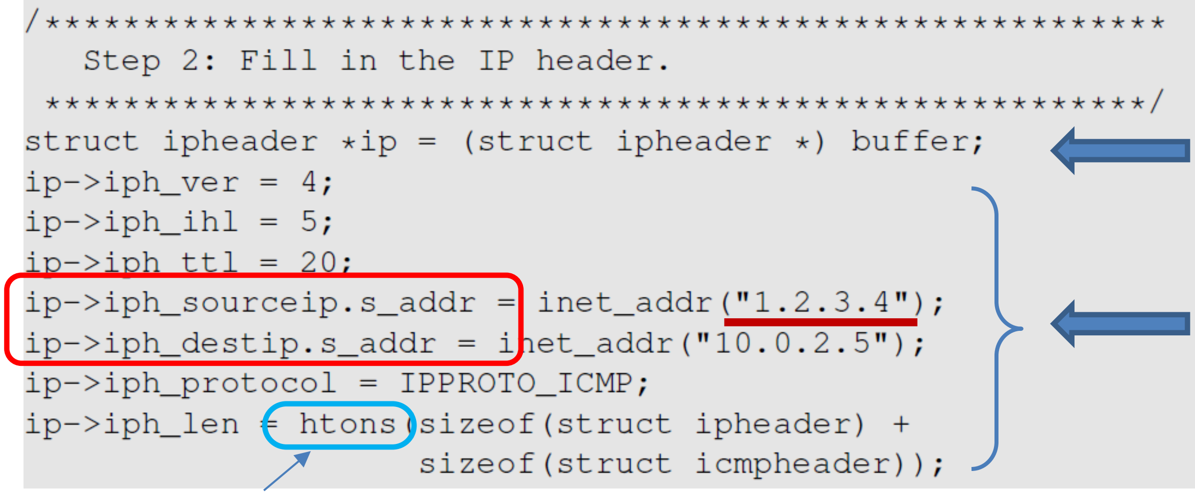
```
char buffer[1500];

memset(buffer, 0, 1500);

/*********************************************************
   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));
```

Find the starting point of the ICMP header, and typecast it to the ICMP structure

Fill in the ICMP header fields

Spoofing Packets: Constructing the Packet

Fill in the IP Header



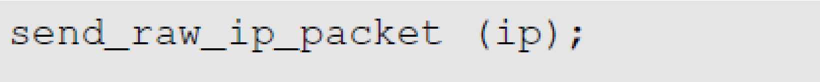
```
/**********************************************************
   Step 2: Fill in the IP header.
 **********************************************************/
struct ipheader *ip = (struct ipheader *) buffer;
ip->iph_ver = 4;
ip->iph_ihl = 5;
ip->iph_ttl = 20;
ip->iph_sourceip.s_addr = inet_addr("1.2.3.4");
ip->iph_destip.s_addr = inet_addr("10.0.2.5");
ip->iph_protocol = IPPROTO_ICMP;
ip->iph_len = htons(sizeof(struct ipheader) +
                    sizeof(struct icmpheader));
```

Typecast the buffer to the IP structure

Fill in the IP header fields

More on this Host to Network function when we discuss Endianness

Finally, send out the packet



```
send_raw_ip_packet (ip);
```

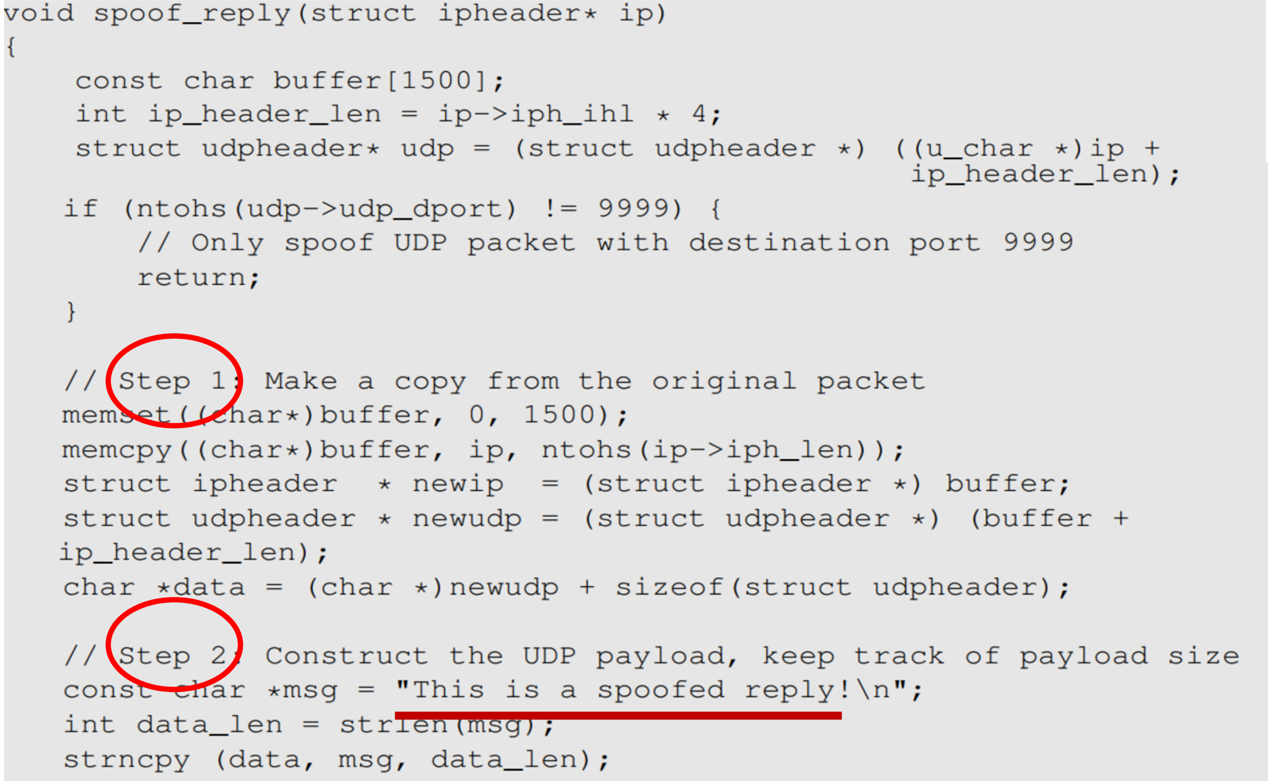
Spoofing UDP Packets

Constructing UDP packets is similar, except that we need to include the payload data.

Sniffing and Then Spoofing

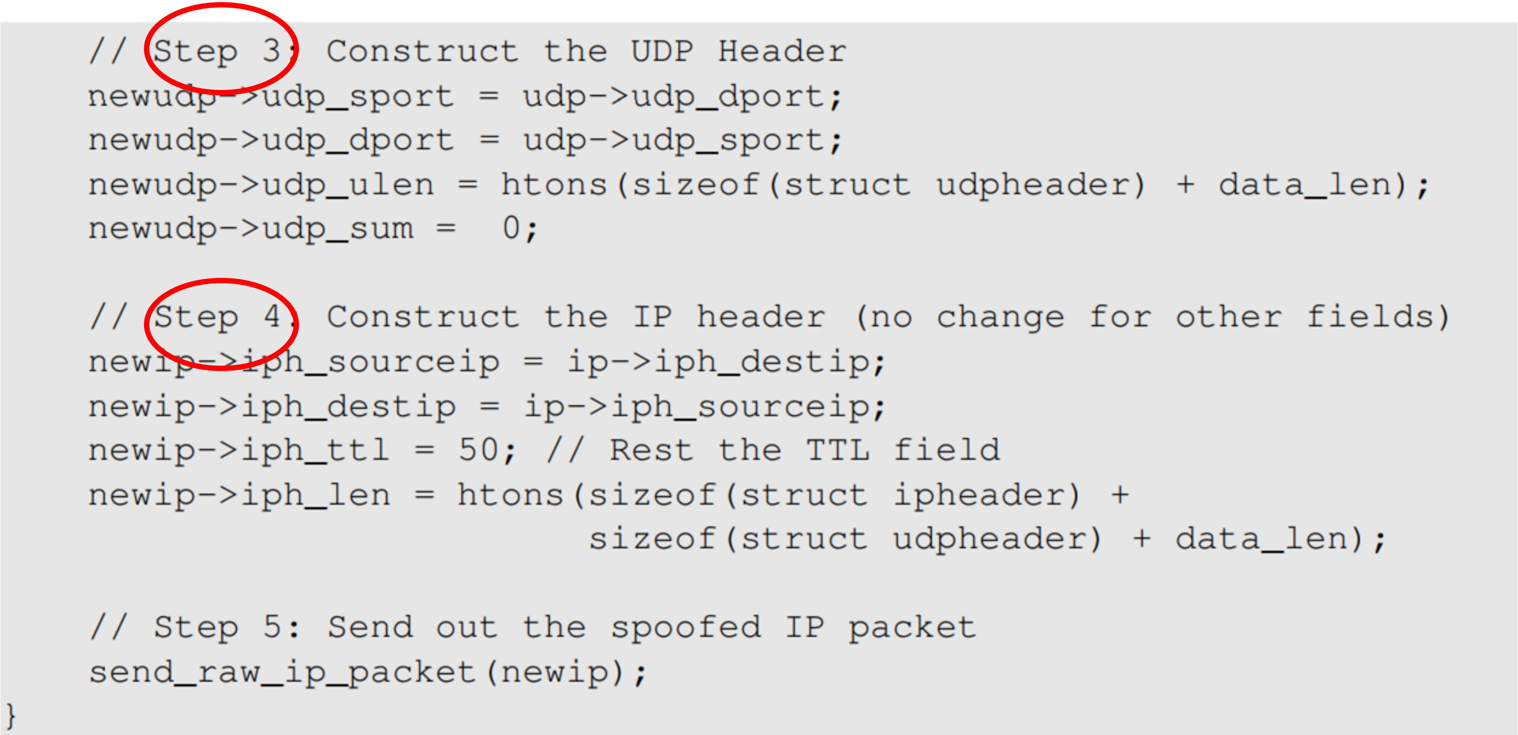
- In many situations, we need to capture packets first, and then spoof a response based on the captured packets.
- Procedure (using UDP as example)
  - Use PCAP API to capture the packets of interests
  - Make a copy from the captured packet
  - Replace the UDP data field with a new message and swap the source and destination fields
  - Send out the spoofed reply
- Instead of C, we can use Scapy in Python to do the same

Sniffing/Spoofing UDP Packet



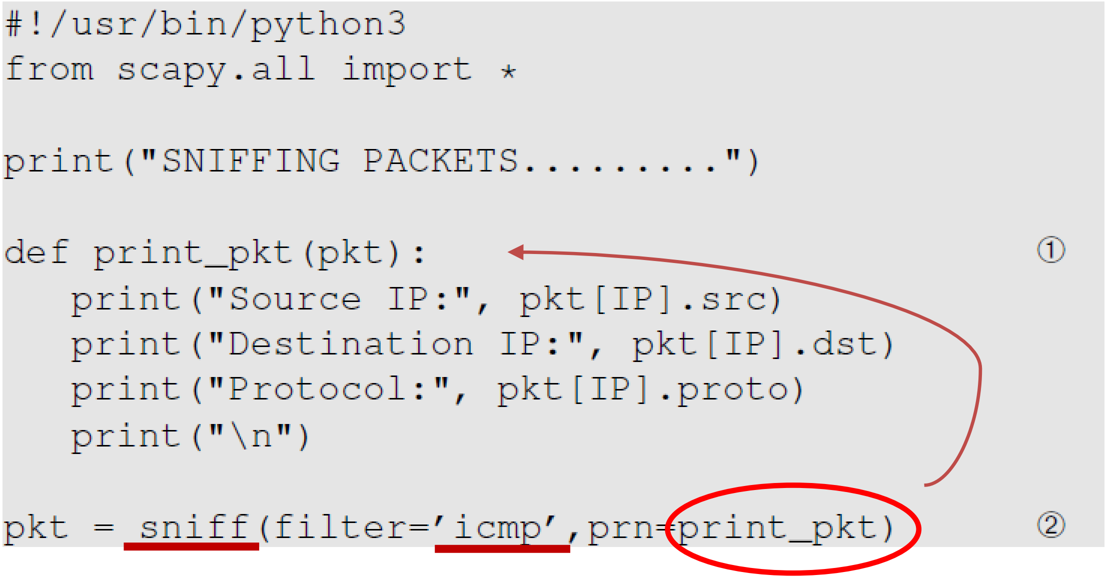
```
void spoof_reply(struct ipheader* ip)
{
    const char buffer[1500];
    int ip_header_len = ip->iph_ihl * 4;
    struct udpheader* udp = (struct udpheader *) ((u_char *)ip +
                                                   ip_header_len);
   if (ntohs(udp->udp_dport) != 9999) {
       // Only spoof UDP packet with destination port 9999
       return;
   }

   // Step 1: Make a copy from the original packet
   memset((char*)buffer, 0, 1500);
   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);

   // Step 2: 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);
```

Sniffing/Spoofing UDP Packet (Continued)



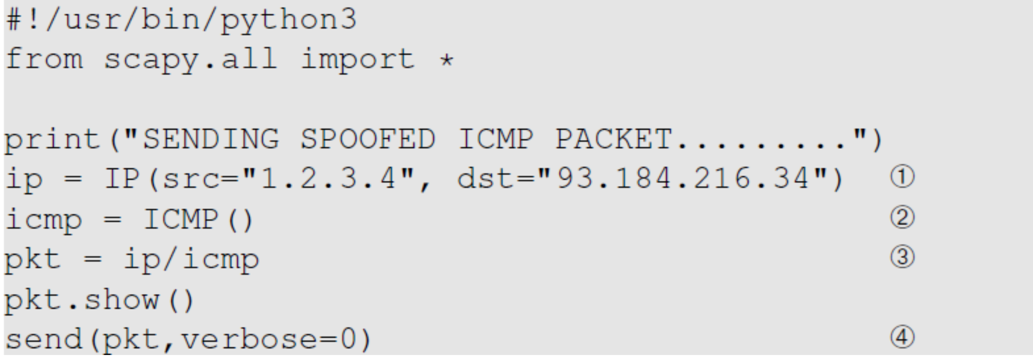
```
    // Step 3: Construct the UDP Header
    newudp->udp_sport = udp->udp_dport;
    newudp->udp_dport = udp->udp_sport;
    newudp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
    newudp->udp_sum =  0;

    // Step 4: Construct the IP header (no change for other fields)
    newip->iph_sourceip = ip->iph_destip;
    newip->iph_destip = ip->iph_sourceip;
    newip->iph_ttl = 50; // Rest the TTL field
    newip->iph_len = htons(sizeof(struct ipheader) +
                           sizeof(struct udpheader) + data_len);

    // Step 5: Send out the spoofed IP packet
    send_raw_ip_packet(newip);
}
```

Packet Sniffing Using Scapy

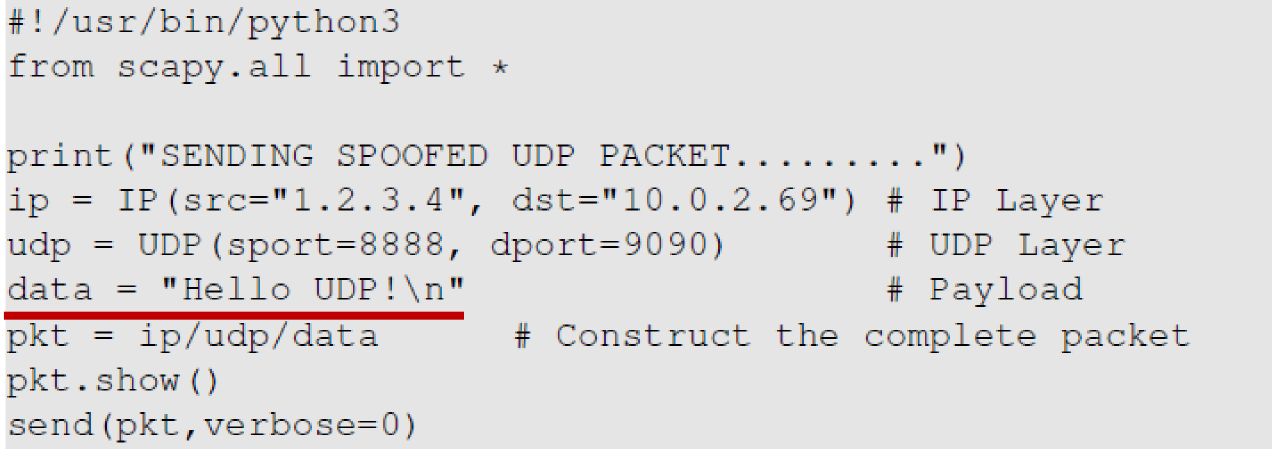


```
#!/usr/bin/python3
from scapy.all import *

print("SNIFFING PACKETS.........")

def print_pkt(pkt):                                          ①
   print("Source IP:", pkt[IP].src)
   print("Destination IP:", pkt[IP].dst)
   print("Protocol:", pkt[IP].proto)
   print("\n")

pkt = sniff(filter='icmp',prn=print_pkt)                     ②
```

Spoofing ICMP & UDP Using Scapy

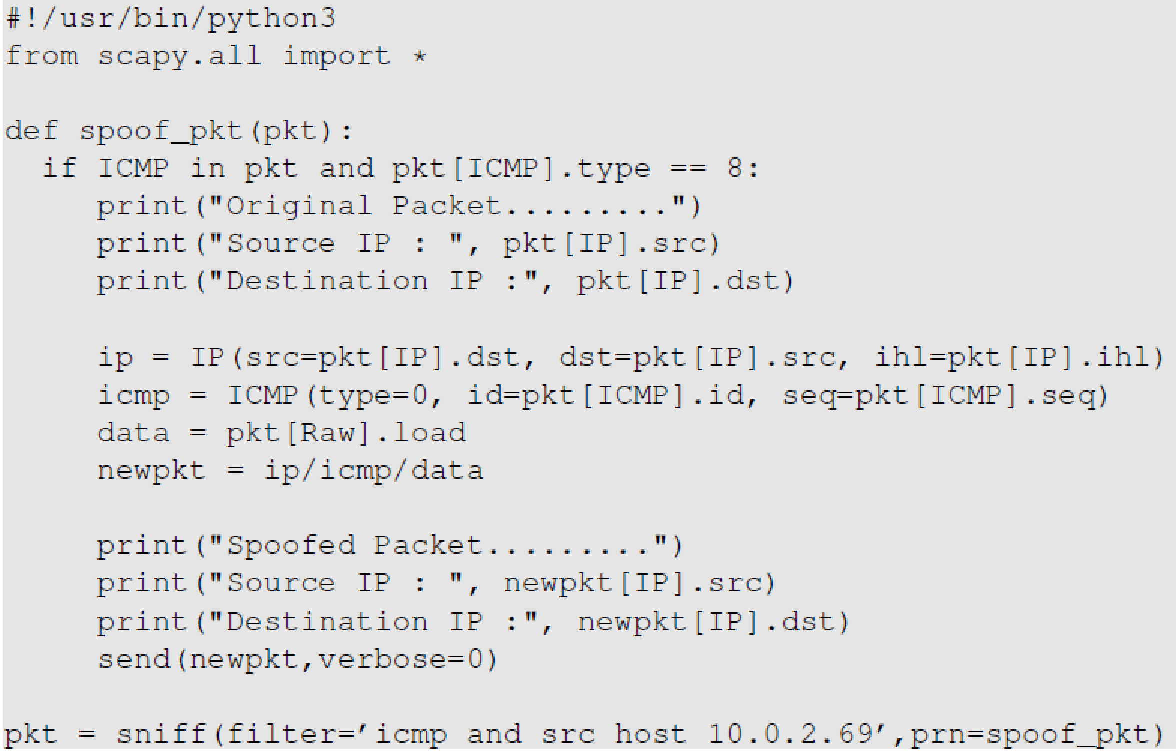


```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SPOOFED ICMP PACKET.........")
ip = IP(src="1.2.3.4", dst="93.184.216.34")  ①
icmp = ICMP()                                ②
pkt = ip/icmp                                ③
pkt.show()
send(pkt,verbose=0)                          ④
```



```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SPOOFED UDP PACKET.........")
ip = IP(src="1.2.3.4", dst="10.0.2.69") # IP Layer
udp = UDP(sport=8888, dport=9090)       # UDP Layer
data = "Hello UDP!\n"                   # Payload
pkt = ip/udp/data      # Construct the complete packet
pkt.show()
send(pkt,verbose=0)
```

Sniffing and Then Spoofing Using Scapy



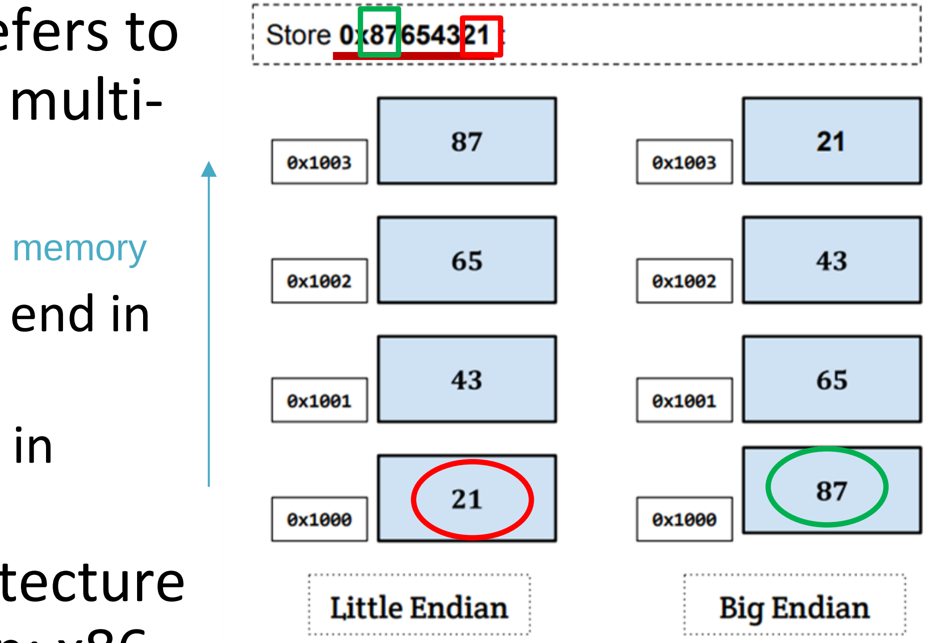
```
#!/usr/bin/python3
from scapy.all import *

def spoof_pkt(pkt):
  if ICMP in pkt and pkt[ICMP].type == 8:
     print("Original Packet.........")
     print("Source IP : ", pkt[IP].src)
     print("Destination IP :", pkt[IP].dst)

     ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
     icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
     data = pkt[Raw].load
     newpkt = ip/icmp/data

     print("Spoofed Packet.........")
     print("Source IP : ", newpkt[IP].src)
     print("Destination IP :", newpkt[IP].dst)
     send(newpkt,verbose=0)

pkt = sniff(filter=’icmp and src host 10.0.2.69’,prn=spoof_pkt)
```

Packet Spoofing: Scapy v.s C

- Python + Scapy
  - Pros: constructing packets is very simple
  - Cons: much slower than C code
- C Program (using raw socket)
  - Pros: much faster
  - Cons: constructing packets is complicated
- Hybrid Approach
  - Using Scapy to construct packets
  - Using C to slightly modify packets and then send packets

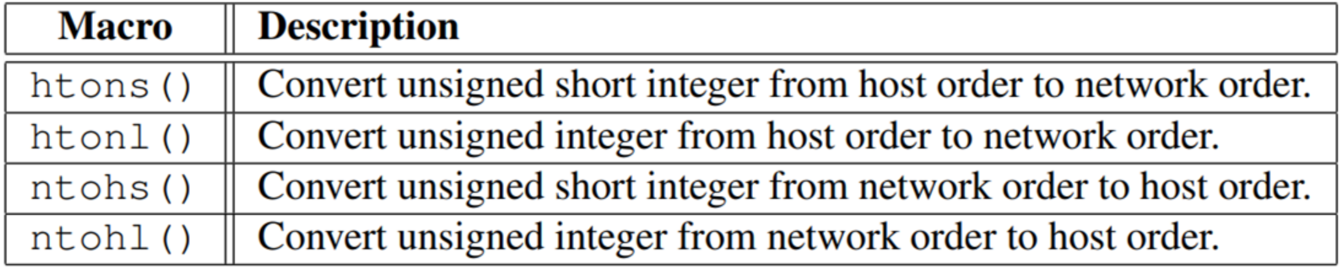
Endianness

- **Endianness**: a term that refers to the order in which a given multi-byte data item is stored in memory.
  - **Little Endian**: put the small end in memory first
  - **Big Endian**: put the big end in memory first
- Atmel AVR32, IBM z/Architecture mainframes use Big-Endian; x86 uses Little-Endian.



Endianness In Network Communication

- Computers with different byte orders will “misunderstand” each other.
  - – Solution: agree upon a common order for communication
  - – This is called “**network order**”, which is the same as **big endian** order
- All computers need to convert data between “host order” and “network order” .



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.

Summary

- Packet sniffing
  - Using raw socket
  - Using PCAP APIs
- Packet spoofing using raw socket
- Sniffing and the spoofing
  - Using C, or Scapy, or a hybrid
- Endianness