Packet Sniffing

Lecture 1

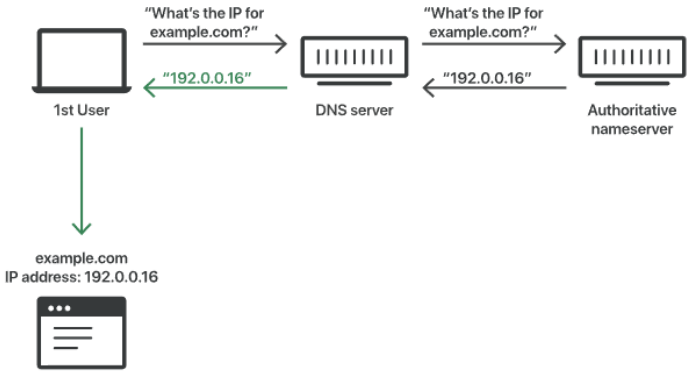
Instructor: C. Pu (Ph.D., Assistant Professor)

puc@marshall.edu

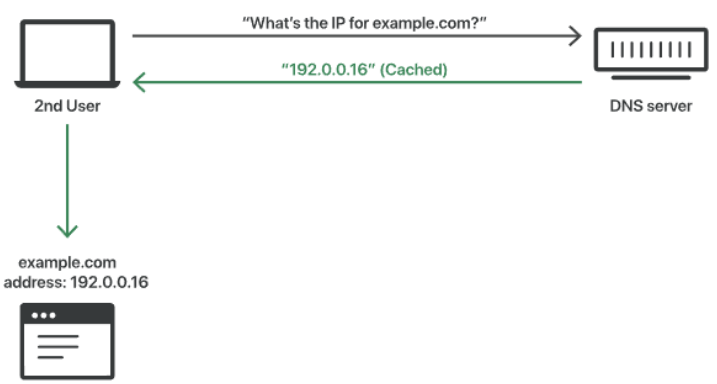
Introduction

- two common attacks on networks:
  - sniffing attack
    - adversary monitors physical network and captures packets
  - spoofing attack
    - adversary issues invalid packets with false identity
- sniffing and spoofing are the basis for other attacks
  - e.g., DNS cache poisoning, TCP session hijacking

DNS uncached response

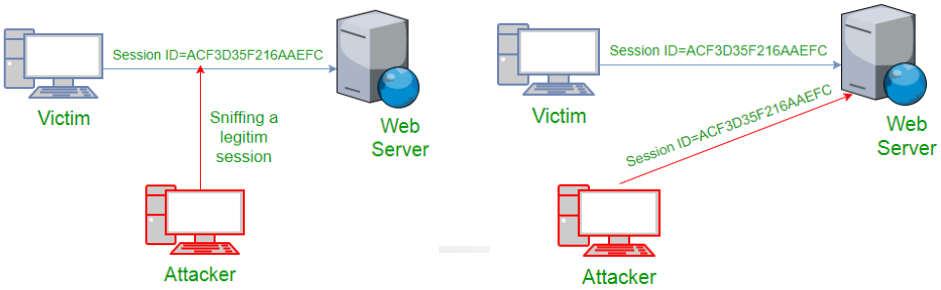


DNS cached response



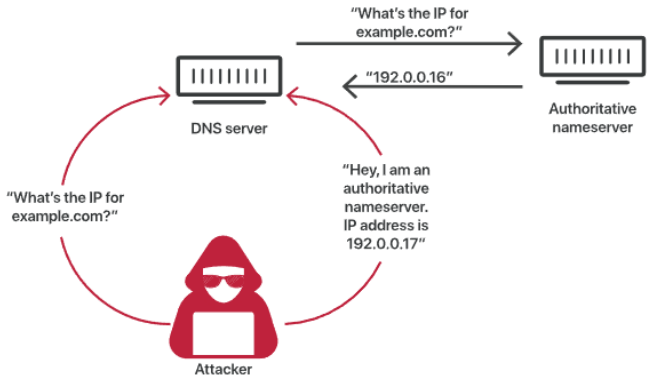
Introduction

TCP session hijacking

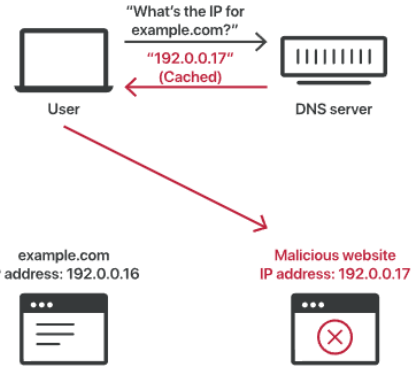


- two common attacks on networks:
  - sniffing attack
    - adversary monitors physical network and captures packets
  - spoofing attack
    - adversary issues invalid packets with false identity
- sniffing and spoofing are the basis for other attacks
  - e.g., DNS cache poisoning, TCP session hijacking

DNS Cache Poisoning Process



Poisoned DNS Cache



Introduction (cont.)

- packet sniffing: common attack on network
  - adversary eavesdrops on a physical network (wires or wireless), and capture the packets transmitted over networks
  - the basis for other Internet attacks, e.g., DNS cache poisoning attack, TCP session hijacking attack
  - available tools: Wireshark, netwox, and Scapy

https://www.wireshark.org/

http://ntwox.sourceforge.net/

https://scapy.net/

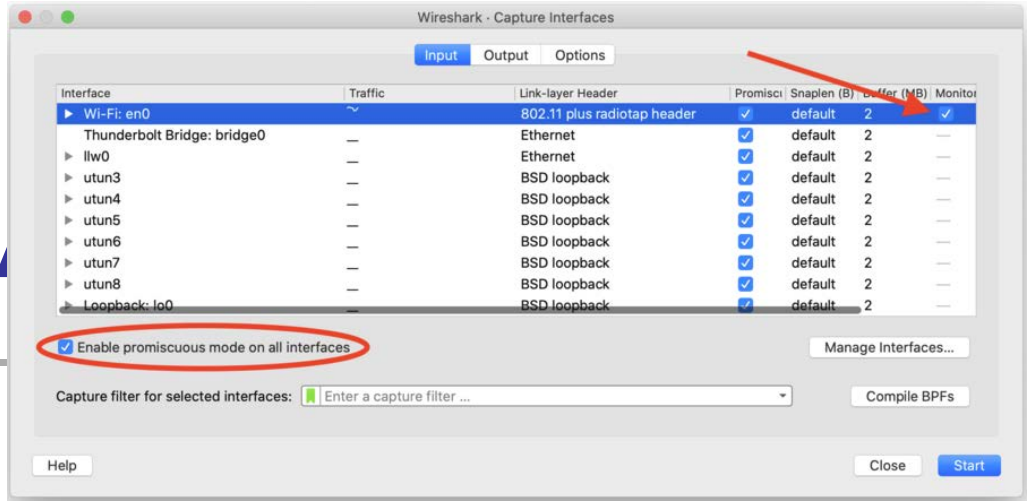
How Packets Are Received?



- network interface card (NIC)
  - a link (physical or logical) between a machine and a network
    - connecting machines to networks
  - has a hardware address: MAC address
- common local comm. techniques: Ethernet and WiFi
  - use broadcast medium (or single shared medium)
  - as data (frame) flow via the medium, every NIC can hear
    - when frame arrives, it is copied into the memory in the NIC
      - checks des. MAC address in the header
        - if match with NIC's MAC add., the frame is copied into kernel buffer
          - interrupts the CPU for new packet
          - CPU copies packet into a queue
        - if not match, the frame is discarded

function calls

How Packets A



- *promiscuous mode*
  - most NIC have this special mode: pass every frame from network to the kernel, regardless of destination MAC add.
  - if registered, the kernel forwards all frames to sniffer program
    - usually require elevated privilege, e.g., root, to use promiscuous mode
- *monitor mode* (wireless)
  - unlike Ethernet, wireless devices suffer interference from other nearby wireless devices
  - to solve this, wireless devices transmit data on different channels
  - when NIC is placed in monitor mode, it captures 802.11 frames transmitting on the channel that it is listening to

BSD (Berkeley Software Distribution) Packet Filter (BPF)

- when sniffing, we're interested in certain types of packet
  - e.g., TCP packets or DNS query packets
- the system can deliver all captured packets to sniffer program, who can discard unwanted packets
  - very inefficient and taking time
    - processing and delivering unwanted packets (if large volume)
- filtering unwanted packets ASAP
  - BSD Packet Filter (BPF): filtering at the lower level
  - user-space program attaches a filer to a socket
    - discarding unwanted packets
  - filter: written in human readable pseudo-code, and interpreted by BSD Pseudo-Machine (packet filteing)

BPF Filter Examples

- capture traffic to and from IP host 192.168.1.1

  ip host 192.168.1.1

- capture traffic from IP host 192.168.1.1

  ip src host 192.168.1.1

- capture Ethernet packets to and from a host with a MAC address of 00:40:D0:13:35:36

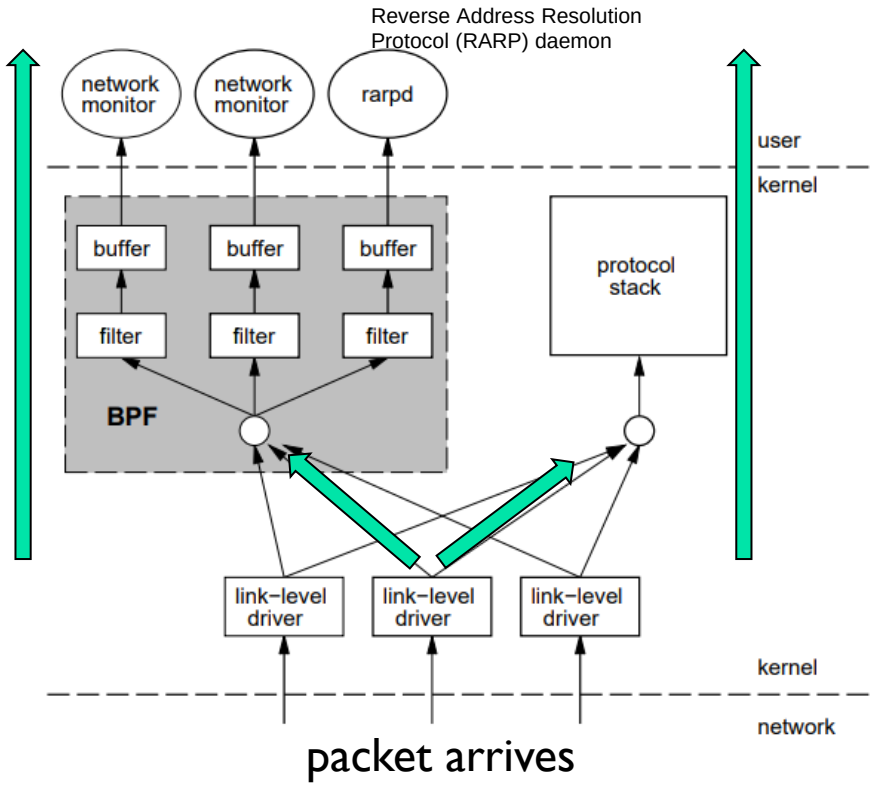
  ether host 00:40:D0:13:35:36

- capture Ethernet packets to host 00:40:D0:13:35:36

  ether dst 00:40:D0:13:35:36

BSD (Berkeley Software Distribution) Packet Filter (BPF) (cont.)

- BPF Overview: interacting with system



BSD Packet Filter (BPF) (cont.)

- an example of a compiled BPF code

```
struct sock_filter code[] = {
  { 0x28,  0,  0, 0x0000000c }, { 0x15,  0,  8, 0x000086dd },
  { 0x30,  0,  0, 0x00000014 }, { 0x15,  2,  0, 0x00000084 },
  { 0x15,  1,  0, 0x00000006 }, { 0x15,  0, 17, 0x00000011 },
  { 0x28,  0,  0, 0x00000036 }, { 0x15, 14,  0, 0x00000016 },
  { 0x28,  0,  0, 0x00000038 }, { 0x15, 12, 13, 0x00000016 },
  { 0x15,  0, 12, 0x00000800 }, { 0x30,  0,  0, 0x00000017 },
  { 0x15,  2,  0, 0x00000084 }, { 0x15,  1,  0, 0x00000006 },
  { 0x15,  0,  8, 0x00000011 }, { 0x28,  0,  0, 0x00000014 },
  { 0x45,  6,  0, 0x00001fff }, { 0xb1,  0,  0, 0x0000000e },
  { 0x48,  0,  0, 0x0000000e }, { 0x15,  2,  0, 0x00000016 },
  { 0x48,  0,  0, 0x00000010 }, { 0x15,  0,  1, 0x00000016 },
  { 0x06,  0,  0, 0x0000ffff }, { 0x06,  0,  0, 0x00000000 },
};

struct sock_fprog bpf = {
   .len = ARRAY_SIZE(code),
   .filter = code,
};
```

these two parameters are used to pass data used by a particular command

a pointer to the buffer in which the value for the requested option is specified

the size, in bytes, of the buffer

- attaching a compiled BPF code to a socket through

```
setsockopt(sock, SOL_SOCKET, SO_ATTACH_FILTER, &bpf, sizeof(bpf))
```

a descriptor that identifies a socket

option level

socket option

  - reference:
    - https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-setsockopt

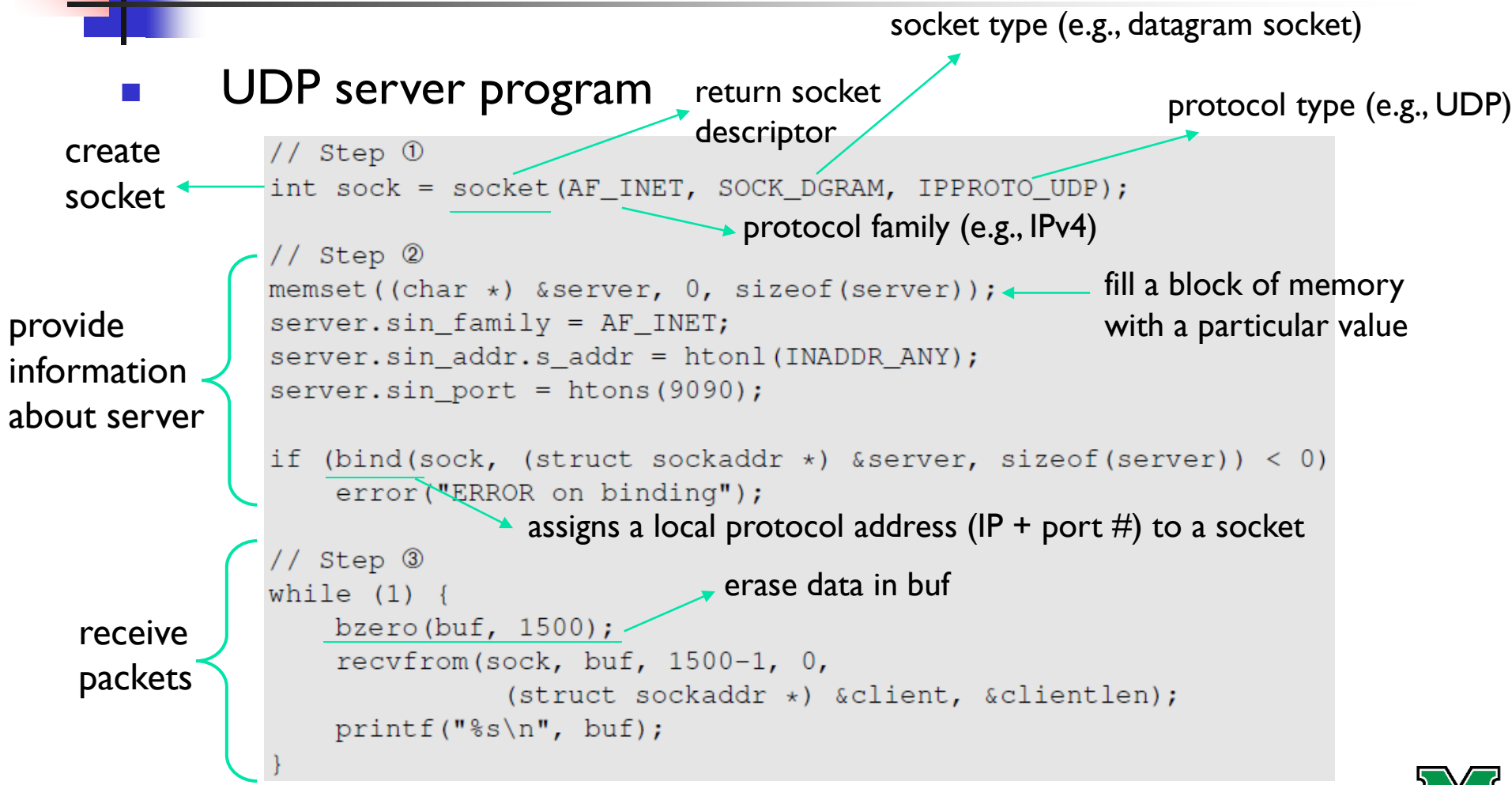
Packet Sniffing

- packet sniffing: capturing live data as they flow across network
  - understand network characteristics
  - diagnose faulty networks and configurations
  - reconnaissance and exploitation

- packet sniffing tools = packet sniffers

Receiving Packets Using Sockets (udp_server.c)

- UDP server program




```
// 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);
}
```

- reference:
  - https://www.tutorialspoint.com/unix_sockets/socket_core_functions.htm

Packet Sniffing Using Raw Sockets

- issue in the previous program: receiving packets that are intended for it
  - if the des. IP add. or the des. port # is not matching, no packets are captured

- what we want: capturing all packeting flowing on the cable, regardless of the des. IP or port #
  - *raw socket*
    - *allows access to the underlying transport provider*
    - *allows user to send and obtain packets of information from the network without interacting with OS*

Packet Sniffing Using Raw Sockets

- packet capture using raw socket

protocol family (e.g., IPv4)

creating a raw socket

```
// 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");
}
```

- normal socket

  kernel receives packet

  ⇩

  pass packet through protocol stack

  ⇩

  pass to applications

- raw socket

  kernel receives packet

  ⇩

  pass a ***copy*** of packet to *socket*

Packet Sniffing Using Raw Sockets

- packet capture using 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");
}
```

- for raw socket, need to specify the type of packets to receive
- protocol is specified the third arg of socket()
- htons(ETH_P_ALL)
  - packets of all protocols should be passed to socket
- htons(ETH_P_IP)
  - only IP packets will be passed to socket

Packet Sniffing Using Raw Sockets

- packet capture using raw socket

```
// 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");
}
```

struct packet_mreq mr;
mr.mr_type: specifies which action to perform

```
struct packet_mreq {
    int            mr_ifindex;    /* interface index */
    unsigned short mr_type;       /* action */
    unsigned short mr_alen;       /* address length */
    unsigned char  mr_address[8]; /* physical layer address */
};
```

- get all packets coming to computer
- but if packets are not destined for us
  - cannot be captured
- turn on promiscuous mode
  - let in all packets on network
  - once they are in, we can get copy

enable promiscuous mode

- PACKET_MR_PROMISC
  - enables receiving all packets on a shared medium (often known as "promiscuous mode")
- PACKET_ADD_MEMBERSHIP
  - to receive all frames, regardless of destination

Packet Sniffing Using Raw Sockets

- packet capture using raw socket

```
// 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");
}
```

wait for packets

print packets

Packet Sniffing Using Raw Sockets (sniff_raw.c)

- summary: four major steps
  1. creating a raw socket
  2. choose the protocol
  3. enable the promiscuous mode
  4. wait for 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");
}
```