网络安全编程

SEED Labs 2.0

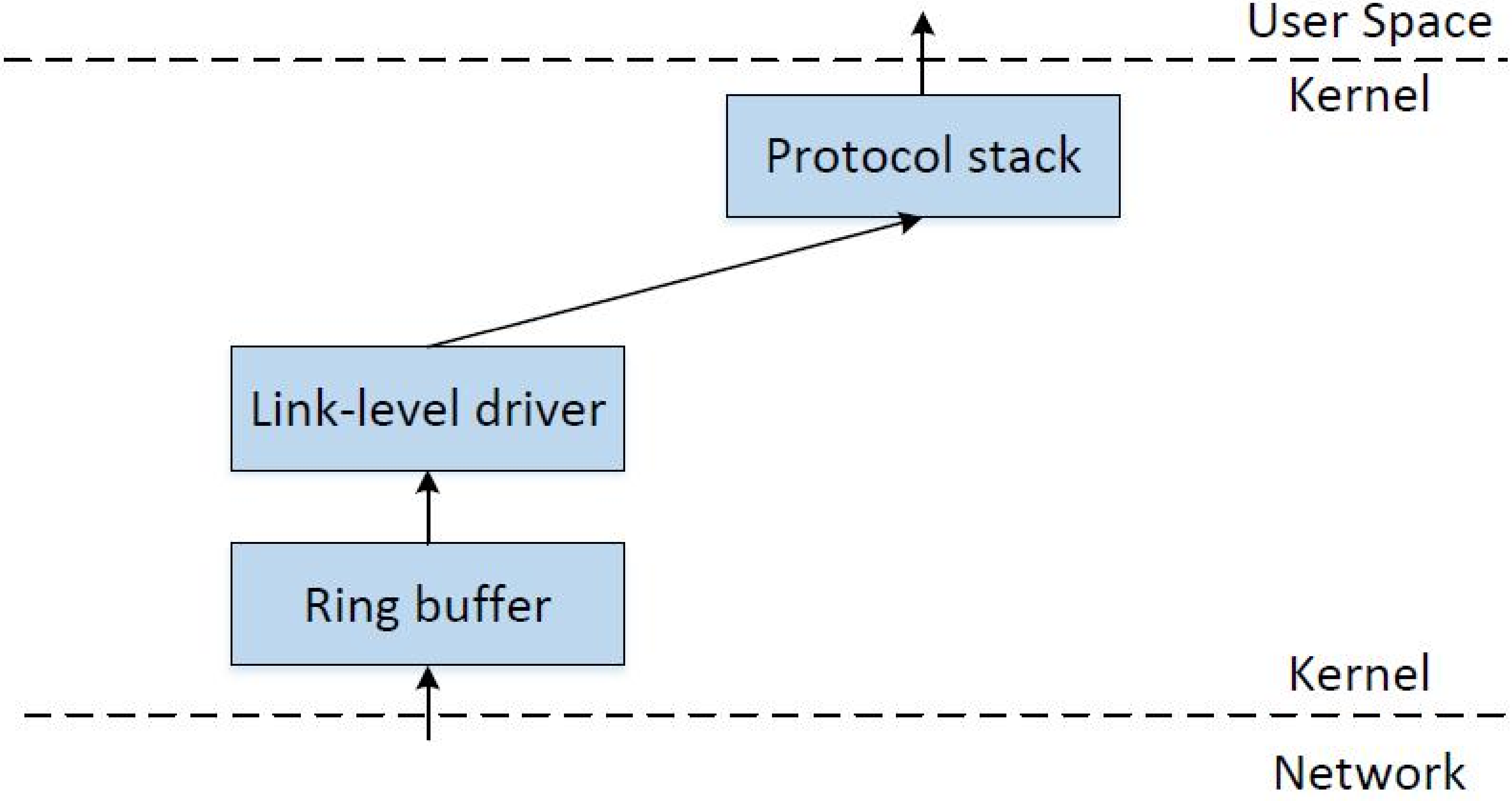
SEED Labs 1.0

https://seedsecuritylabs.org/labs.html

数据包嗅探编程

数据包接收原理

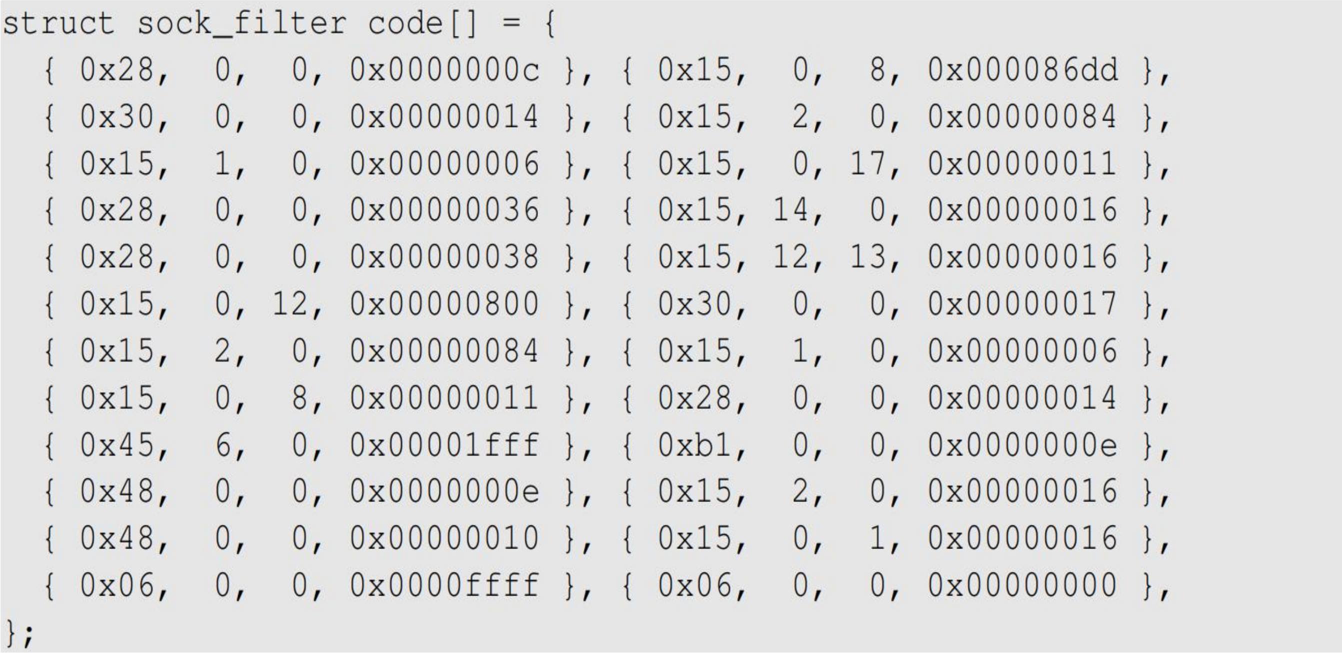
- NIC (Network Interface Card) is a physical or logical link between a machine and a network
- Each NIC has a MAC address
- Every NIC on the network will hear all the frames on the wire
- NIC checks the destination address for every packet, if the address matches the cards MAC address, it is further copied into a buffer in the kernel
  - DMA（Direct Memory Access）
  - 中断
  - 回调函数

User Space
Kernel
Protocol stack
Link-level driver
Ring buffer
Kernel
Network

混杂模式

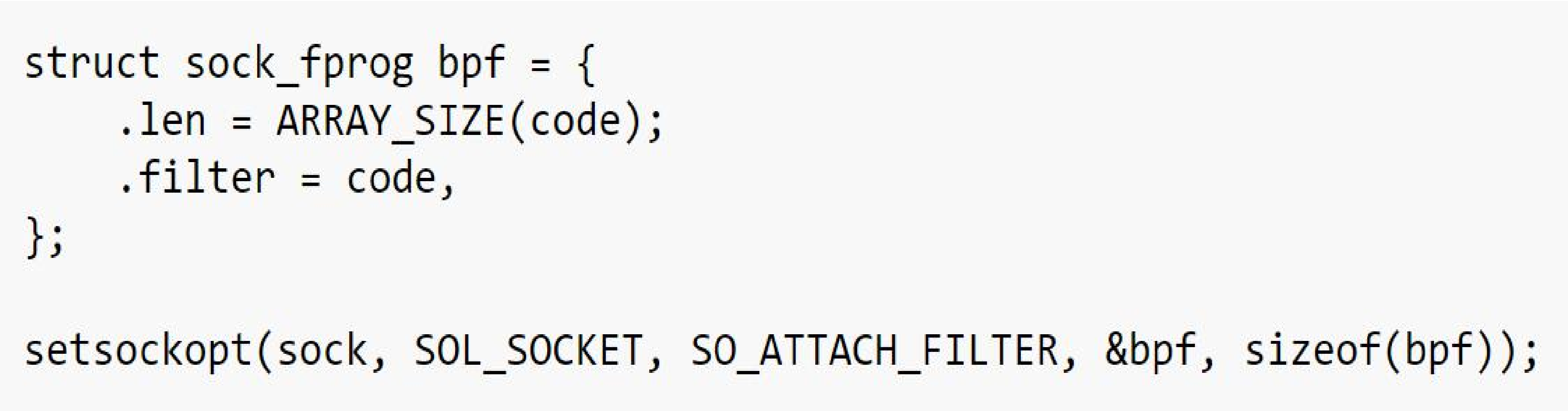
- The frames that are not destined to a given NIC are discarded
- When operating in promiscuous mode, NIC passes every frame received from the network to the kernel
- If a sniffer program is registered with the kernel, it will be able to see all the packets
- In Wi-Fi, it is called Monitor Mode

BSD数据包过滤器



```
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 },
};
```

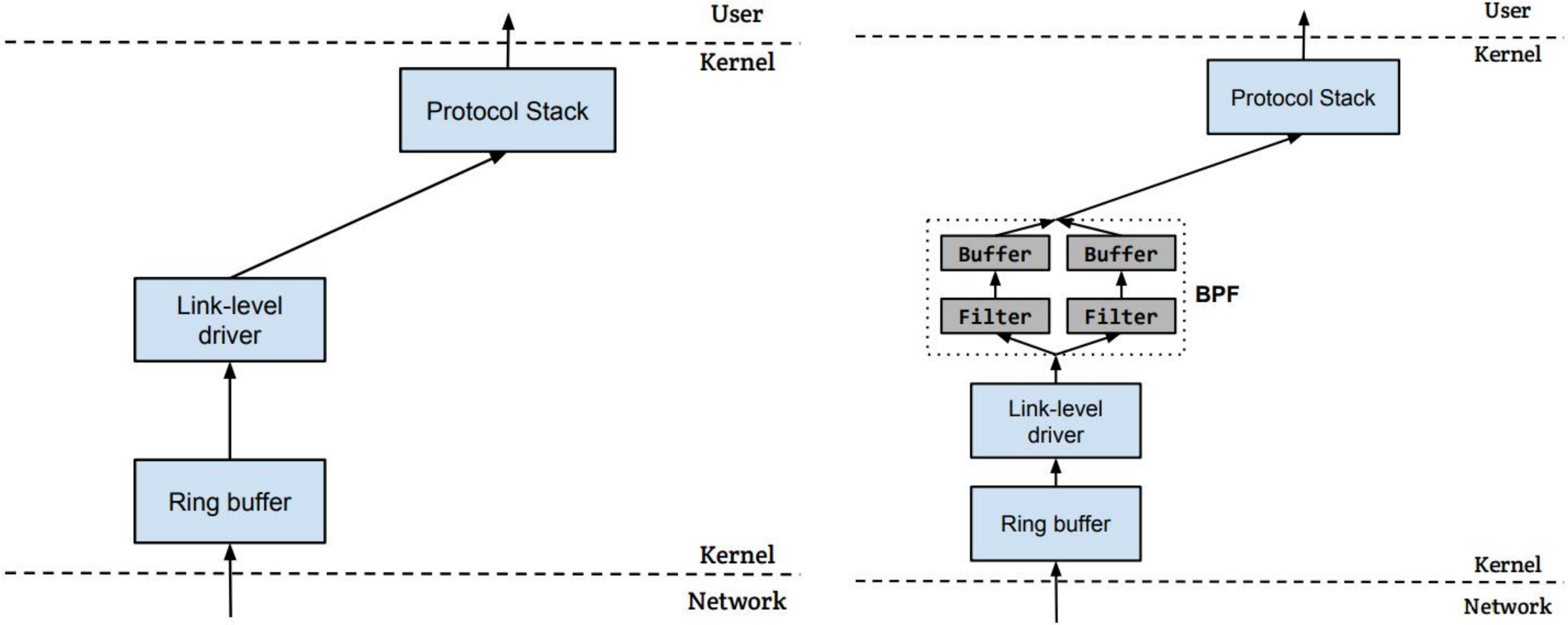
- BPF allows a user-program to attach a filter to the socket, which tells the kernel to discard unwanted packets.
- An example of the compiled BPF code is shown here.



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

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

- A compiled BPF pseudo-code can be attached to a socket through `setsockopt()`
- When a packet is received by kernel, BPF will be invoked
- An accepted packet is pushed up the protocol stack. See the diagram on the following slide.

使用BPF前后对比



(a) Packet flow

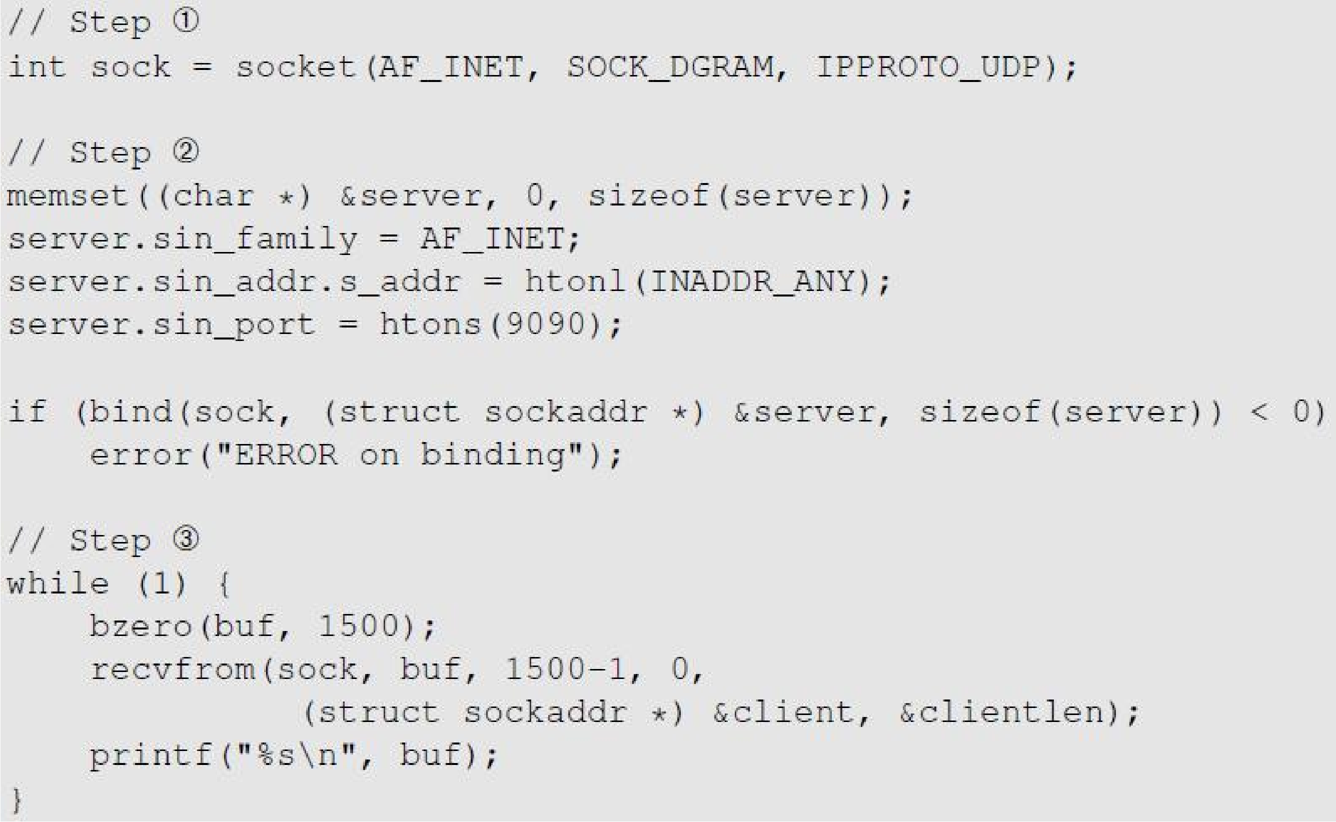
(b) Packet flow with filter

正常的Socket数据包接收

Create the socket

Provide information about server

Receive packets

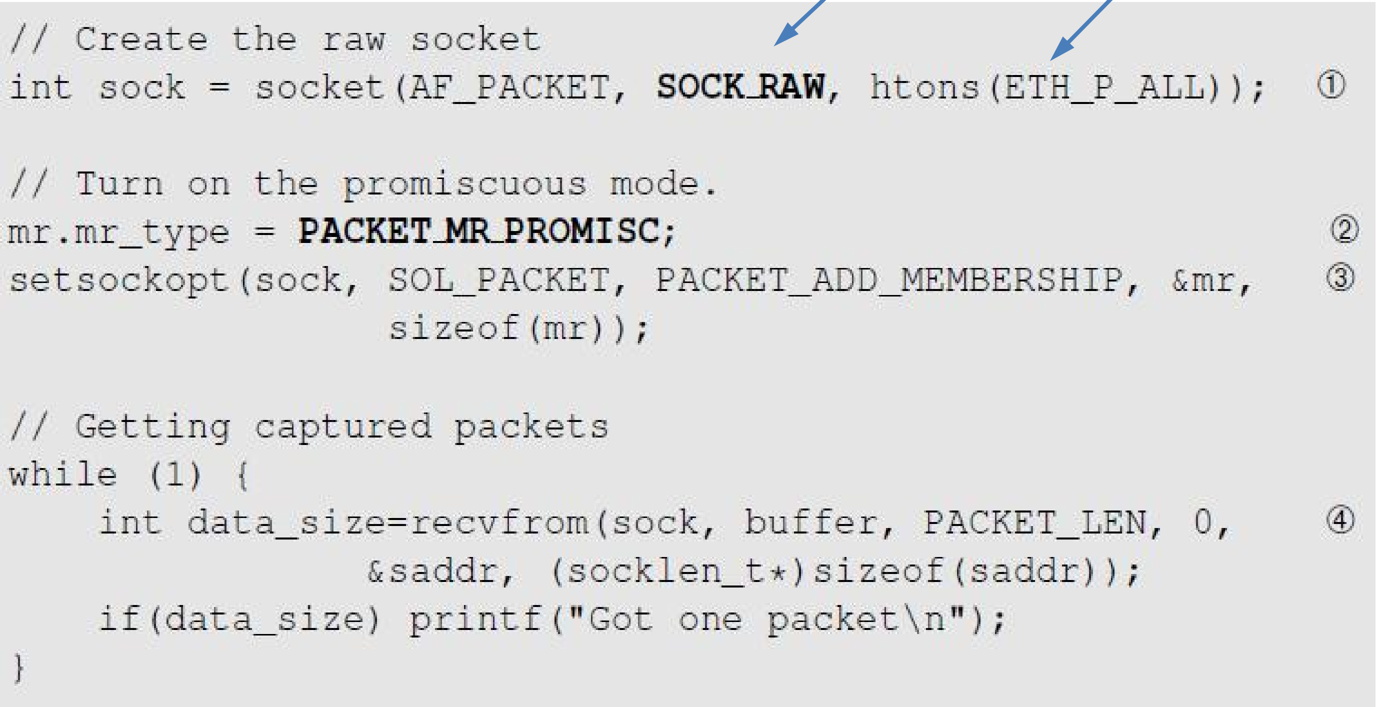


```
// Step ①
int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

// 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);
}
```

使用Raw Socket嗅探数据

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");
}
```

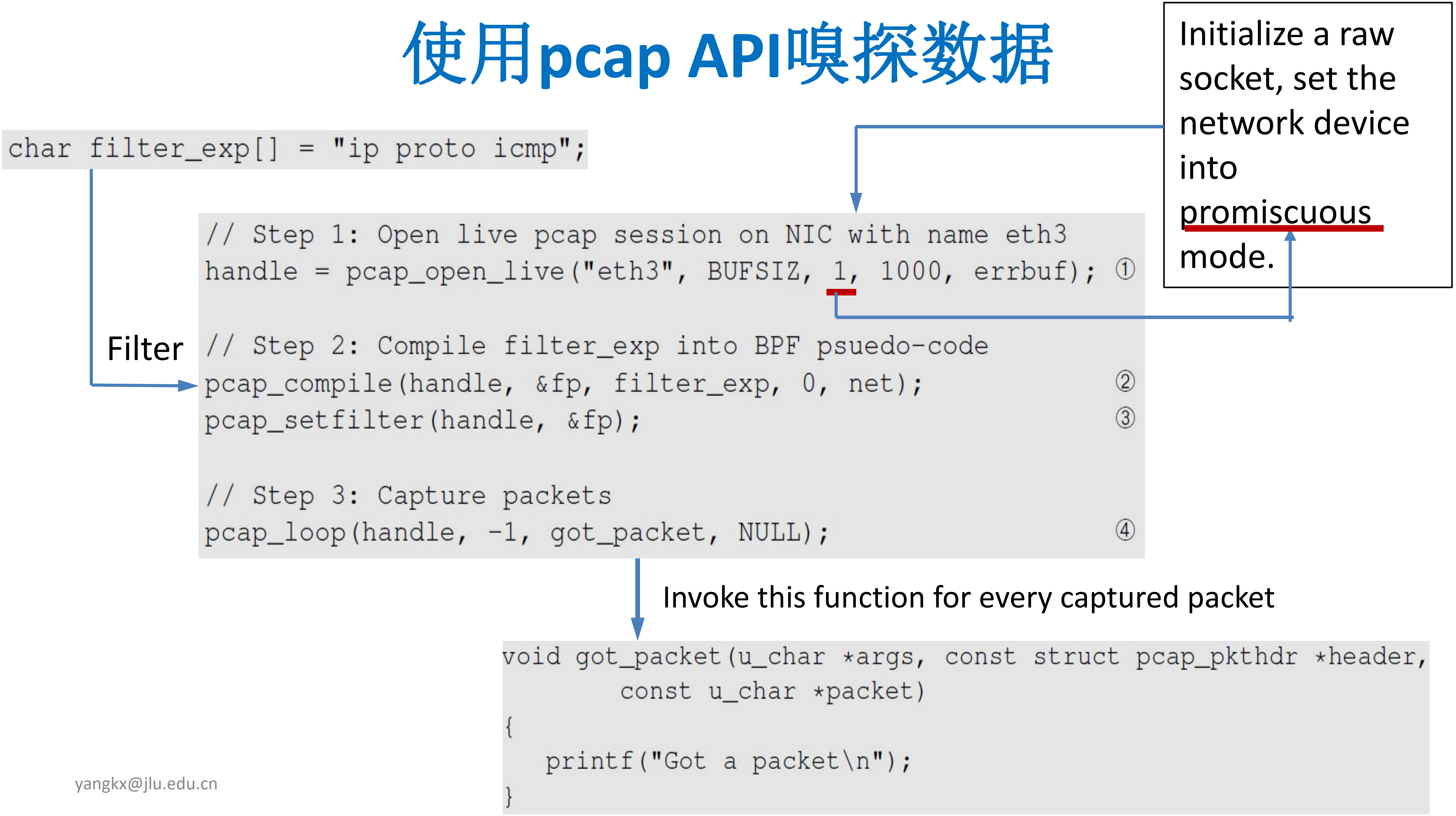
Enable the promiscuous mode

Wait for packets

本方案的缺点

- This program is not portable across different operating systems.
- Setting filters is not easy.
- This program does not explore any optimization to improve performance.
- The PCAP 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.

使用pcap API嗅探数据



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

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

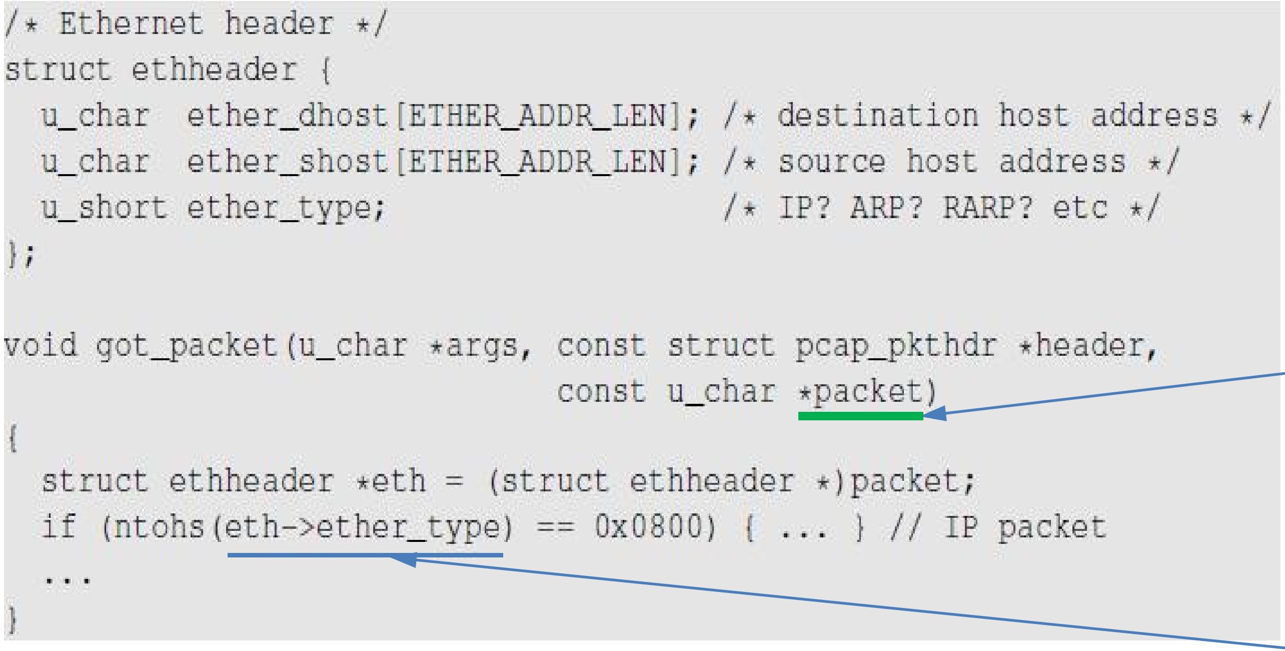
Filter

```
// 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
```

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");
}
```

捕获包处理：以太网帧头部

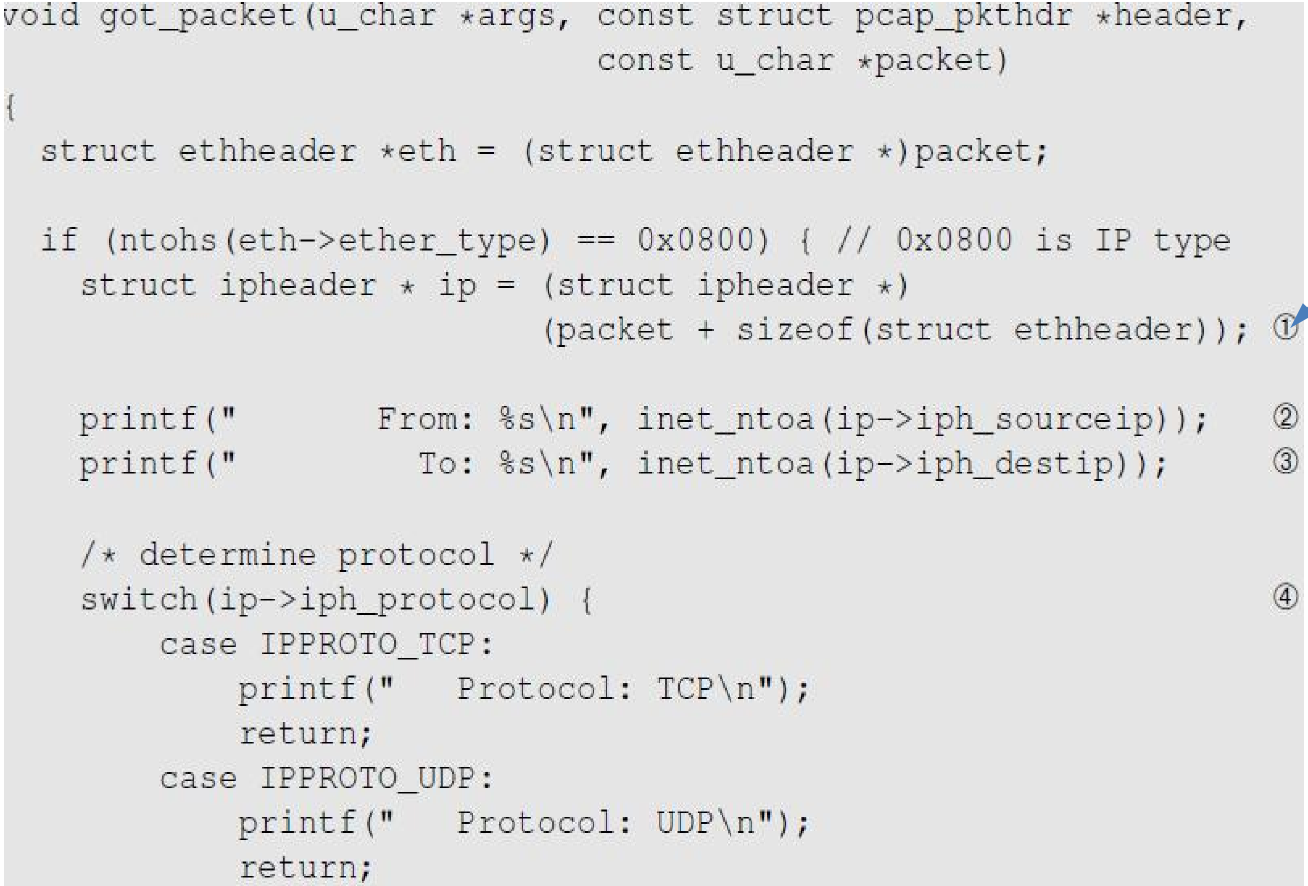


```
/* Ethernet header */
struct ethheader {
  u_char  ether_dhost[ETHER_ADDR_LEN]; /* destination host address */
  u_char  ether_shost[ETHER_ADDR_LEN]; /* source host address */
  u_short ether_type;                  /* IP? ARP? RARP? etc */
};

void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
{
  struct ethheader *eth = (struct ethheader *)packet;
  if (ntohs(eth->ether_type) == 0x0800) { ... } // IP packet
  ...
}
```

The **packet** argument contains a copy of the packet, including the Ethernet header. We typecast it to the Ethernet header structure.

Now we can access the field of the structure

捕获包处理：IP头部



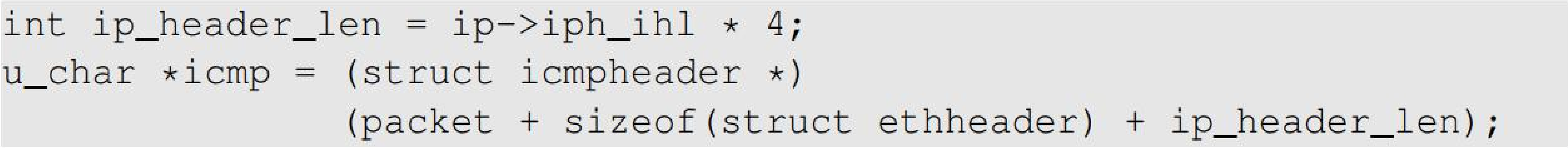
```
void got_packet(u_char *args, const struct pcap_pkthdr *header,
                              const u_char *packet)
{
  struct ethheader *eth = (struct ethheader *)packet;

  if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IP type
    struct ipheader * ip = (struct ipheader *)
                           (packet + sizeof(struct ethheader)); ①

    printf("       From: %s\n", inet_ntoa(ip->iph_sourceip));   ②
    printf("         To: %s\n", inet_ntoa(ip->iph_destip));     ③

    /* determine protocol */
    switch(ip->iph_protocol) {                                  ④
        case IPPROTO_TCP:
            printf("   Protocol: TCP\n");
            return;
        case IPPROTO_UDP:
            printf("   Protocol: UDP\n");
            return;
```

Find where the IP header starts, and typecast it to the IP Header structure.

Now we can easily access the fields in the IP header.

进一步完善

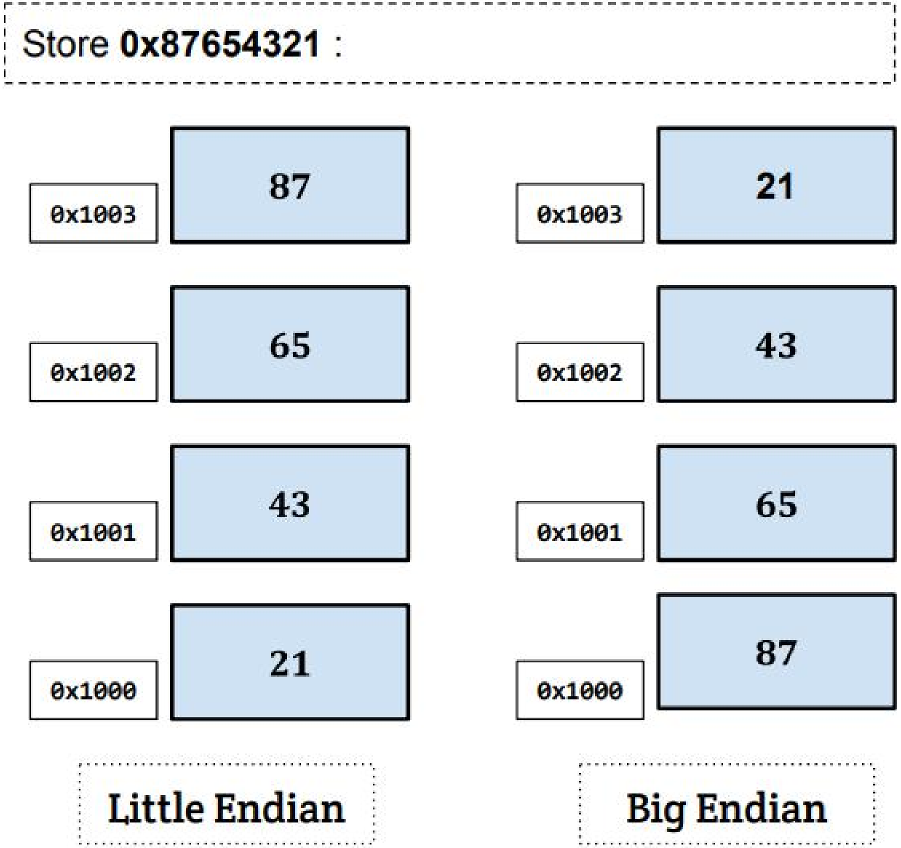
- If we want to further process the packet, such as printing out the header of the TCP, UDP and ICMP, we can use the similar technique.
  - We move the pointer to the beginning of the next header and type-cast
  - We need to use the header length field in the IP header to calculate the actual size of the IP header
- In the following example, if we know the next header is ICMP, we can get a pointer to the ICMP part by doing the following:



```
int ip_header_len = ip->iph_ihl * 4;
u_char *icmp = (struct icmpheader *)
               (packet + sizeof(struct ethheader) + ip_header_len);
```

机器字节序

- Endianness: a term that refers to the order in which a given multi-byte data item is stored in memory.
  - Little Endian: store the most significant byte of data at the highest address
  - Big Endian: store the most significant byte of data at the lowest address



Store **0x87654321** :

0x1003 87
0x1002 65
0x1001 43
0x1000 21

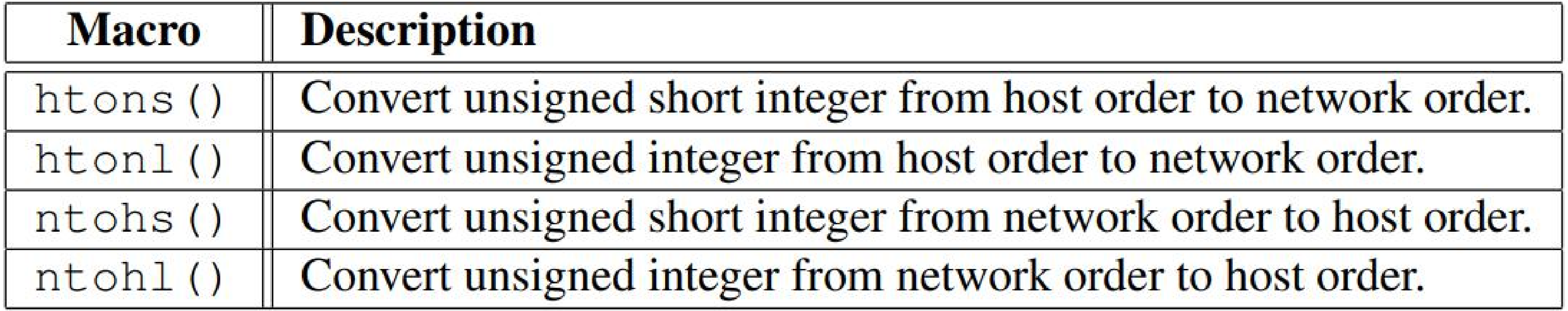
Little Endian

0x1003 21
0x1002 43
0x1001 65
0x1000 87

Big Endian

网络字节序

- 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.