实验7 数据包嗅探和伪造

王美珍
QQ：wer_sec@qq.com

主要内容

- □ 监听（嗅探）报文
- □ 伪造报文
- □ 监听并伪造
- □ **Scapy Vs. C**
- □ 字节序和校验和

1. 实验环境

- Ubuntu Seed虚拟机下载地址：
  - □ QQ群空间
- 虚拟机软件：vmware (15.5.0及兼容版本) + vmware tools
- ubuntu系统的用户密码

  普通用户： seed 密码:dees

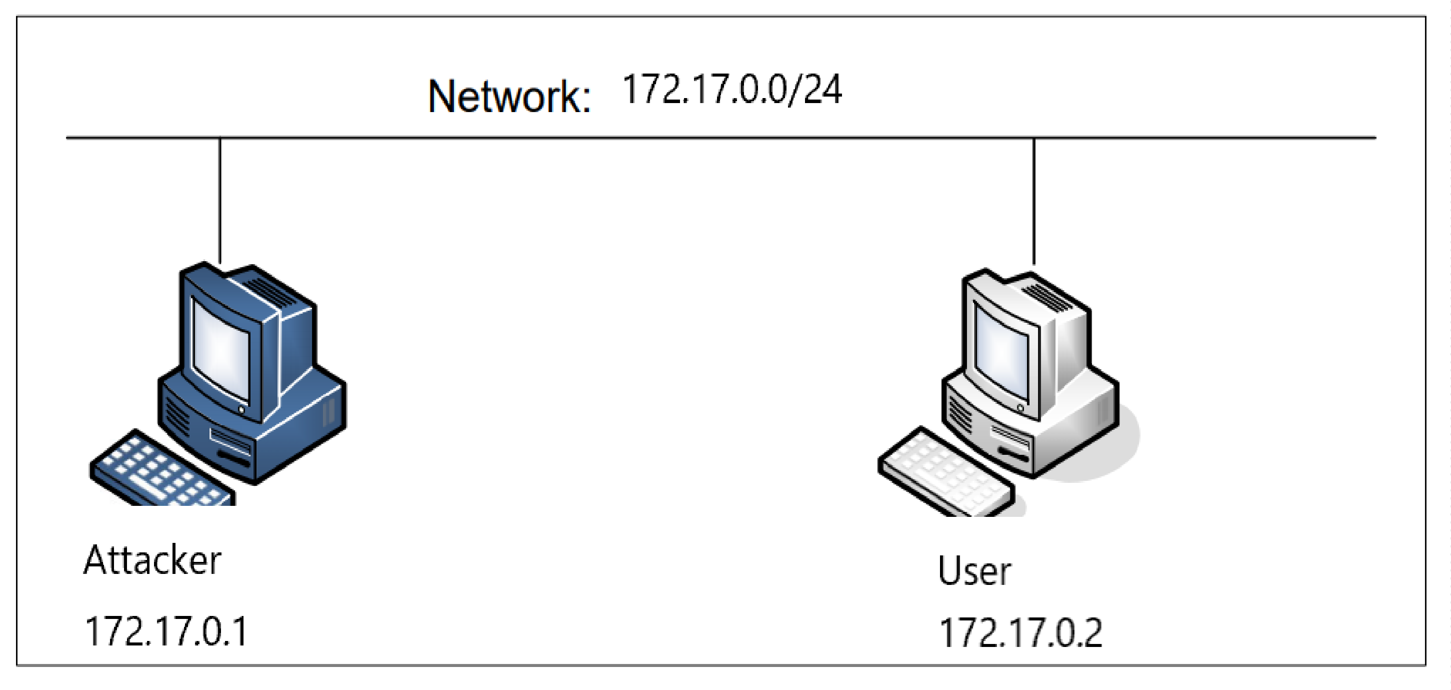
  超级用户：root 密码：seedubuntu

□ 实验采用一个虚拟机，多个容器来完成

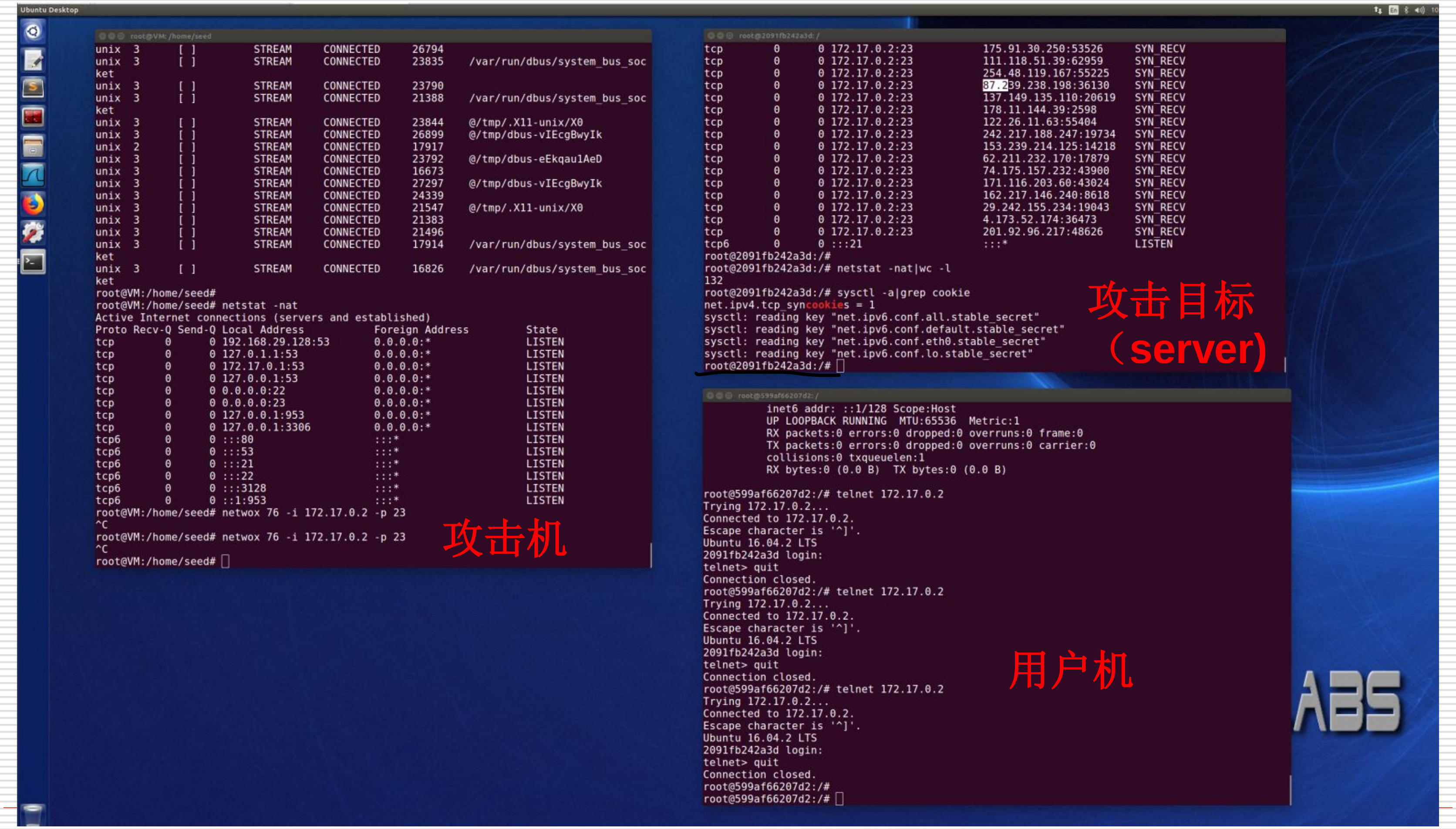
1. docker容器的使用

- 容器查看
  - docker ps –a，可以看到已有一个server
- 容器创建
  - docker run -it --name=user --hostname=user --privileged “seedubuntu” /bin/bash
- 容器启用/停止
  - docker start/stop 容器名
- 进入容器的命令行
  - docker exec –it 容器名 /bin/bash
- 删除容器(实验未完成前不要删除）
  - docker rm 容器名

1. 网络环境搭建



实验环境截图



```
unix  3      [ ]         STREAM     CONNECTED     26794
unix  3      [ ]         STREAM     CONNECTED     23835    /var/run/dbus/system_bus_soc
ket
unix  3      [ ]         STREAM     CONNECTED     23790
unix  3      [ ]         STREAM     CONNECTED     21388    /var/run/dbus/system_bus_soc
ket
unix  3      [ ]         STREAM     CONNECTED     23844    @/tmp/.X11-unix/X0
unix  3      [ ]         STREAM     CONNECTED     26899    @/tmp/dbus-vIEcgBwyIk
unix  2      [ ]         STREAM     CONNECTED     17917
unix  3      [ ]         STREAM     CONNECTED     23792    @/tmp/dbus-eEkqaulAeD
unix  3      [ ]         STREAM     CONNECTED     16673
unix  3      [ ]         STREAM     CONNECTED     27297    @/tmp/dbus-vIEcgBwyIk
unix  3      [ ]         STREAM     CONNECTED     24339
unix  3      [ ]         STREAM     CONNECTED     21547    @/tmp/.X11-unix/X0
unix  3      [ ]         STREAM     CONNECTED     21383
unix  3      [ ]         STREAM     CONNECTED     21496
unix  3      [ ]         STREAM     CONNECTED     17914    /var/run/dbus/system_bus_soc
ket
unix  3      [ ]         STREAM     CONNECTED     16826    /var/run/dbus/system_bus_soc
ket
root@VM:/home/seed#
root@VM:/home/seed# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 192.168.29.128:53       0.0.0.0:*               LISTEN
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 172.17.0.1:53           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
root@VM:/home/seed# netwox 76 -i 172.17.0.2 -p 23
^C
root@VM:/home/seed# netwox 76 -i 172.17.0.2 -p 23
^C
root@VM:/home/seed# 
```

攻击机

```
tcp        0      0 172.17.0.2:23           175.91.30.250:53526     SYN_RECV
tcp        0      0 172.17.0.2:23           111.118.51.39:62959     SYN_RECV
tcp        0      0 172.17.0.2:23           254.48.119.167:55225    SYN_RECV
tcp        0      0 172.17.0.2:23           87.239.238.198:36130    SYN_RECV
tcp        0      0 172.17.0.2:23           137.149.135.110:20619   SYN_RECV
tcp        0      0 172.17.0.2:23           178.11.144.39:2598      SYN_RECV
tcp        0      0 172.17.0.2:23           122.26.11.63:55404      SYN_RECV
tcp        0      0 172.17.0.2:23           242.217.188.247:19734   SYN_RECV
tcp        0      0 172.17.0.2:23           153.239.214.125:14218   SYN_RECV
tcp        0      0 172.17.0.2:23           62.211.232.170:17879    SYN_RECV
tcp        0      0 172.17.0.2:23           74.175.157.232:43900    SYN_RECV
tcp        0      0 172.17.0.2:23           171.116.203.60:43024    SYN_RECV
tcp        0      0 172.17.0.2:23           162.217.146.240:8618    SYN_RECV
tcp        0      0 172.17.0.2:23           29.242.155.234:19043    SYN_RECV
tcp        0      0 172.17.0.2:23           4.173.52.174:36473      SYN_RECV
tcp        0      0 172.17.0.2:23           201.92.96.217:48626     SYN_RECV
tcp6       0      0 :::21                   :::*                    LISTEN
root@2091fb242a3d:/#
root@2091fb242a3d:/# netstat -nat|wc -l
132
root@2091fb242a3d:/# sysctl -a|grep cookie
net.ipv4.tcp_syncookies = 1
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.eth0.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
root@2091fb242a3d:/# 
```

攻击目标
（server)

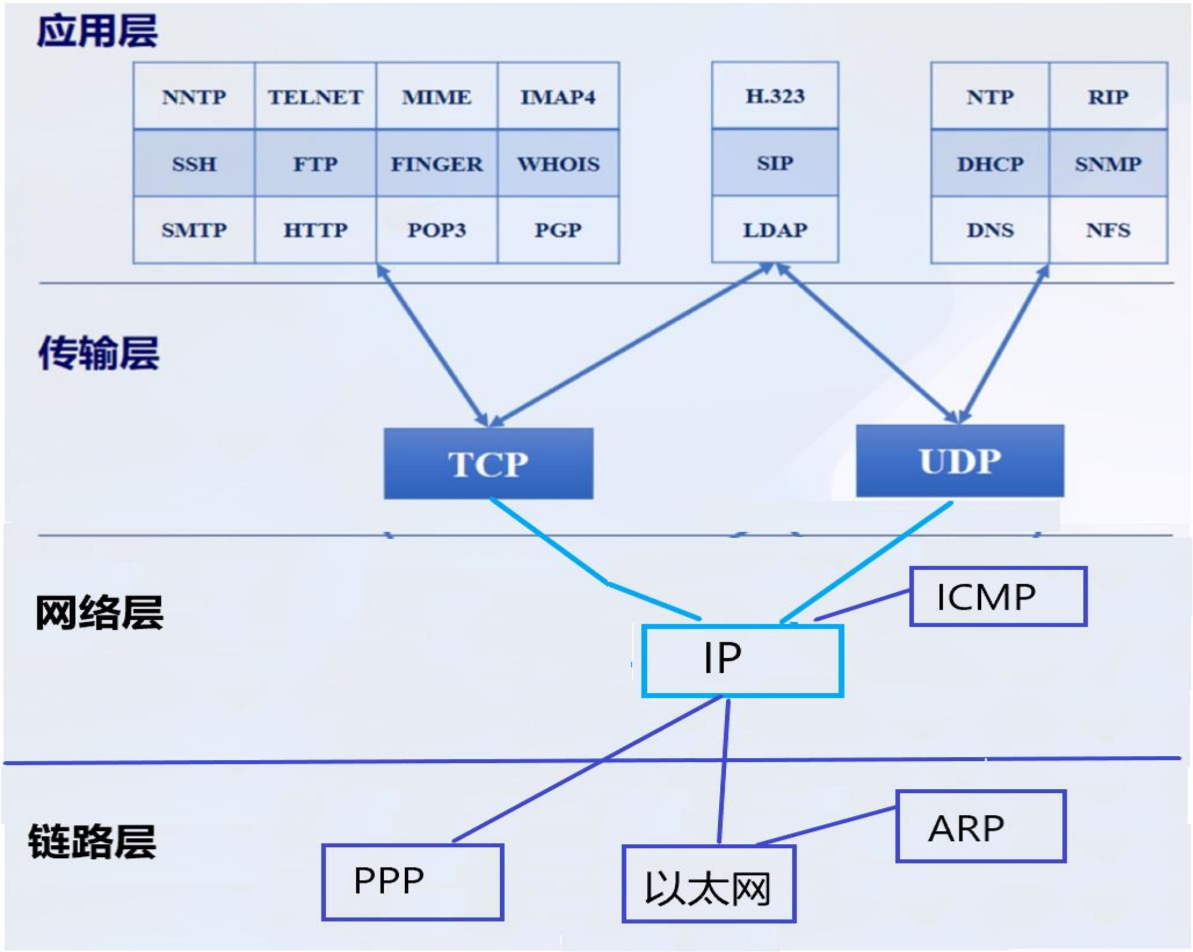
```
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@599af66207d2:/# telnet 172.17.0.2
Trying 172.17.0.2...
Connected to 172.17.0.2.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
2091fb242a3d login:
telnet> quit
Connection closed.
root@599af66207d2:/# telnet 172.17.0.2
Trying 172.17.0.2...
Connected to 172.17.0.2.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
2091fb242a3d login:
telnet> quit
Connection closed.
root@599af66207d2:/# telnet 172.17.0.2
Trying 172.17.0.2...
Connected to 172.17.0.2.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
2091fb242a3d login:
telnet> quit
Connection closed.
root@599af66207d2:/#
root@599af66207d2:/# 
```

用户机

2. 报文嗅探

- □ C语言：Raw Socket
- □ C语言：Pcap库
- □ Python: scapy

2 报文嗅探——网络协议概览



应用层

NNTP TELNET MIME IMAP4
SSH FTP FINGER WHOIS
SMTP HTTP POP3 PGP

H.323
SIP
LDAP

NTP RIP
DHCP SNMP
DNS NFS

传输层

TCP
UDP

网络层

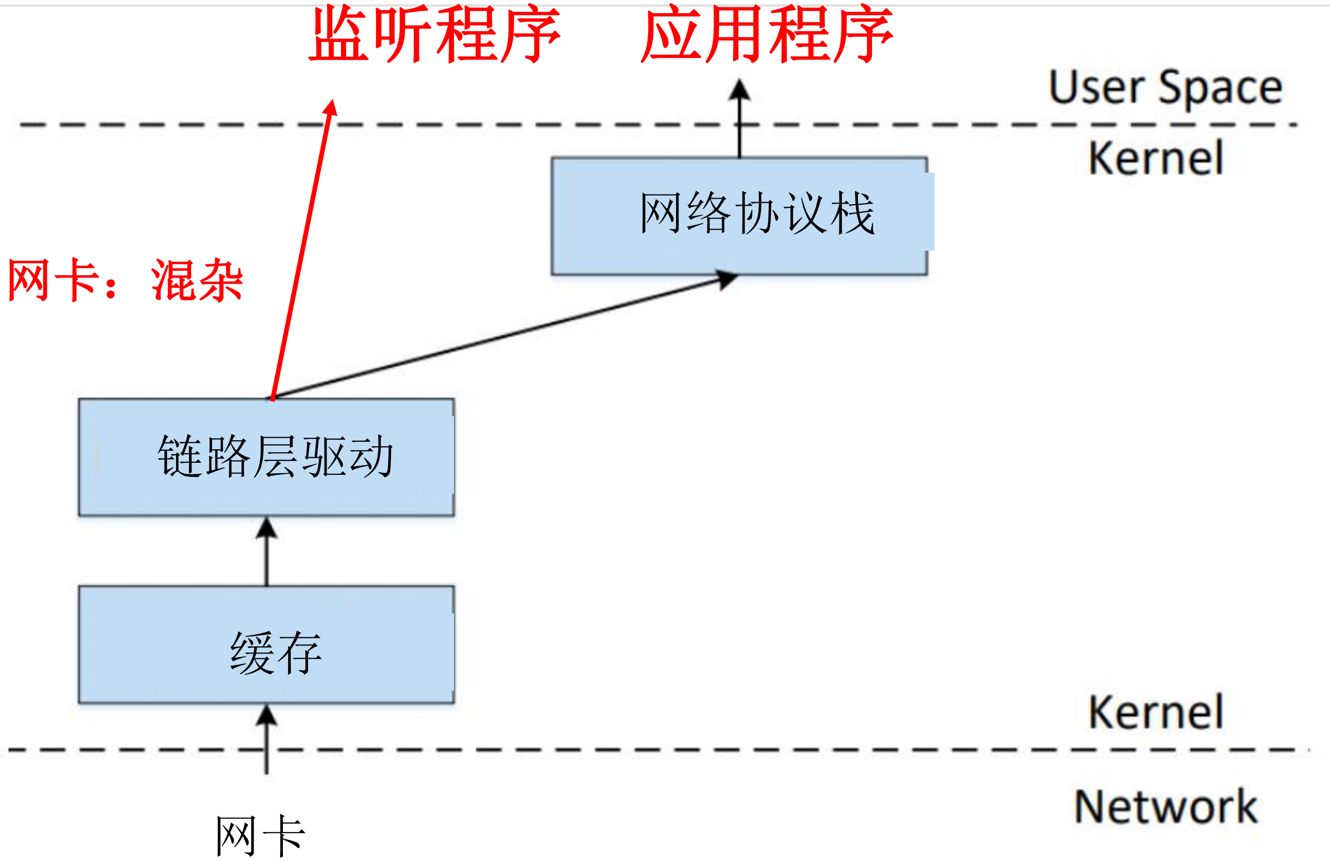
IP
ICMP

链路层

PPP
以太网
ARP

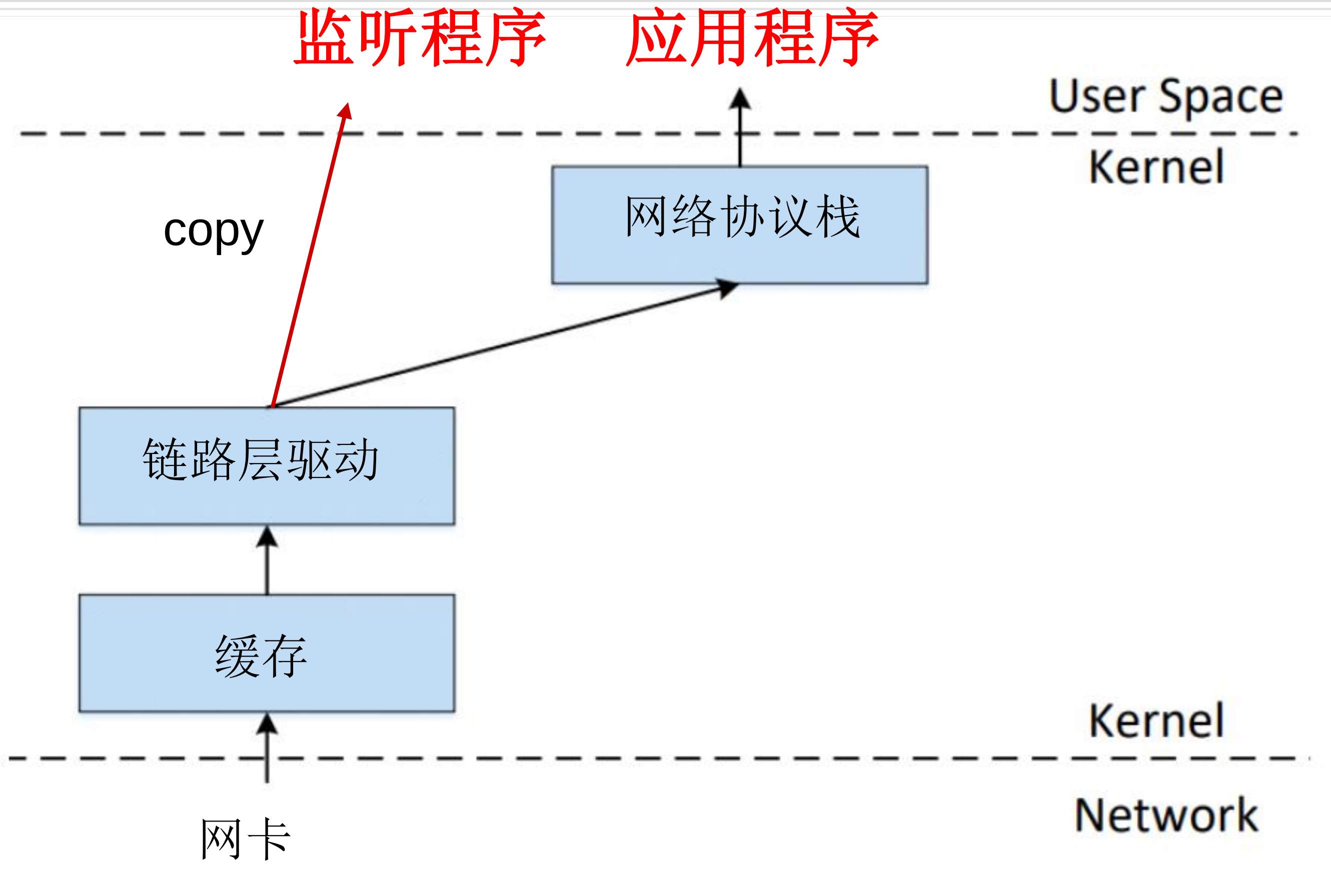
2. 报文嗅探

- 报文如何从底层接收上来？

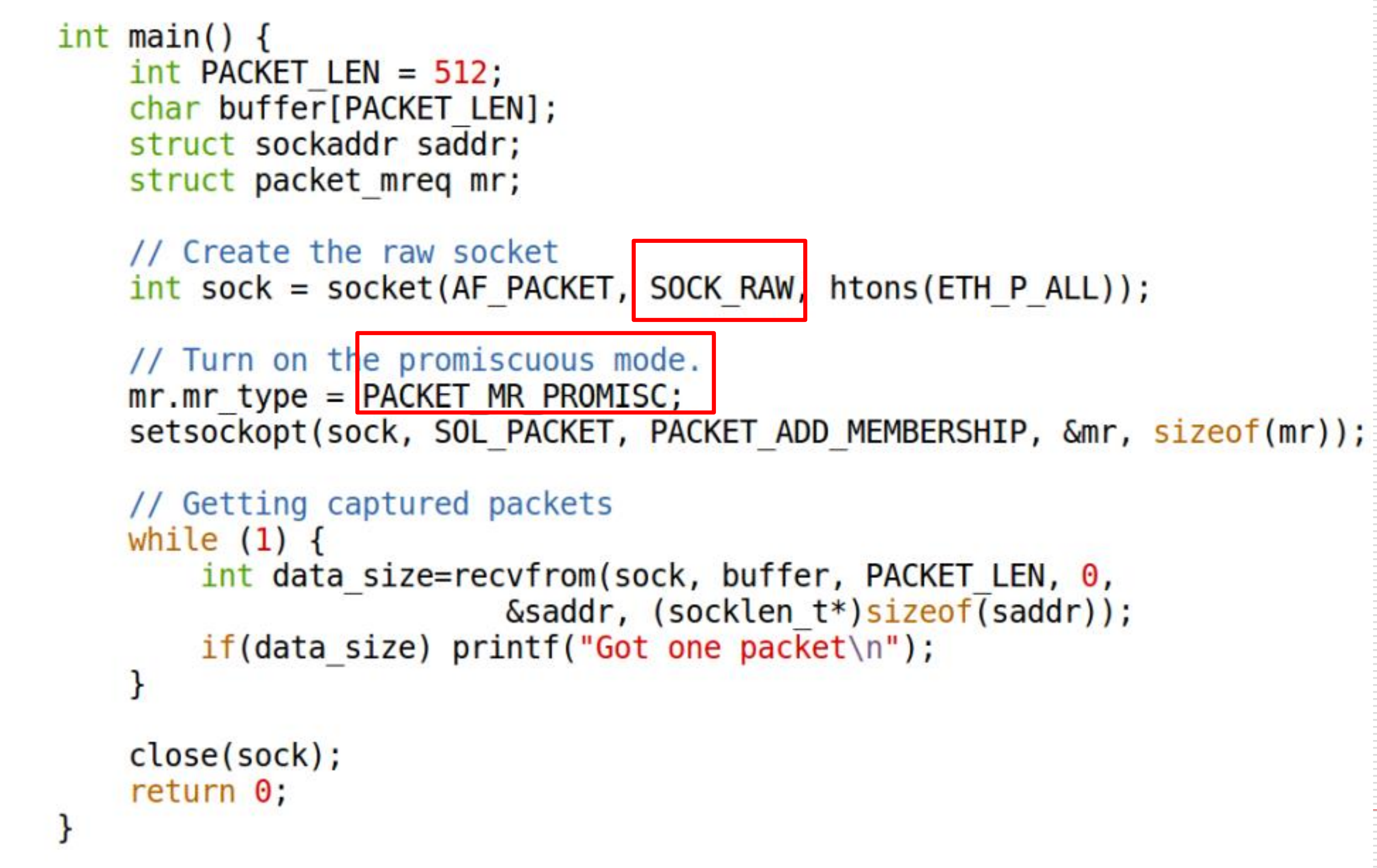


2. 如何获得报文的拷贝？

- □ **Raw Socket**

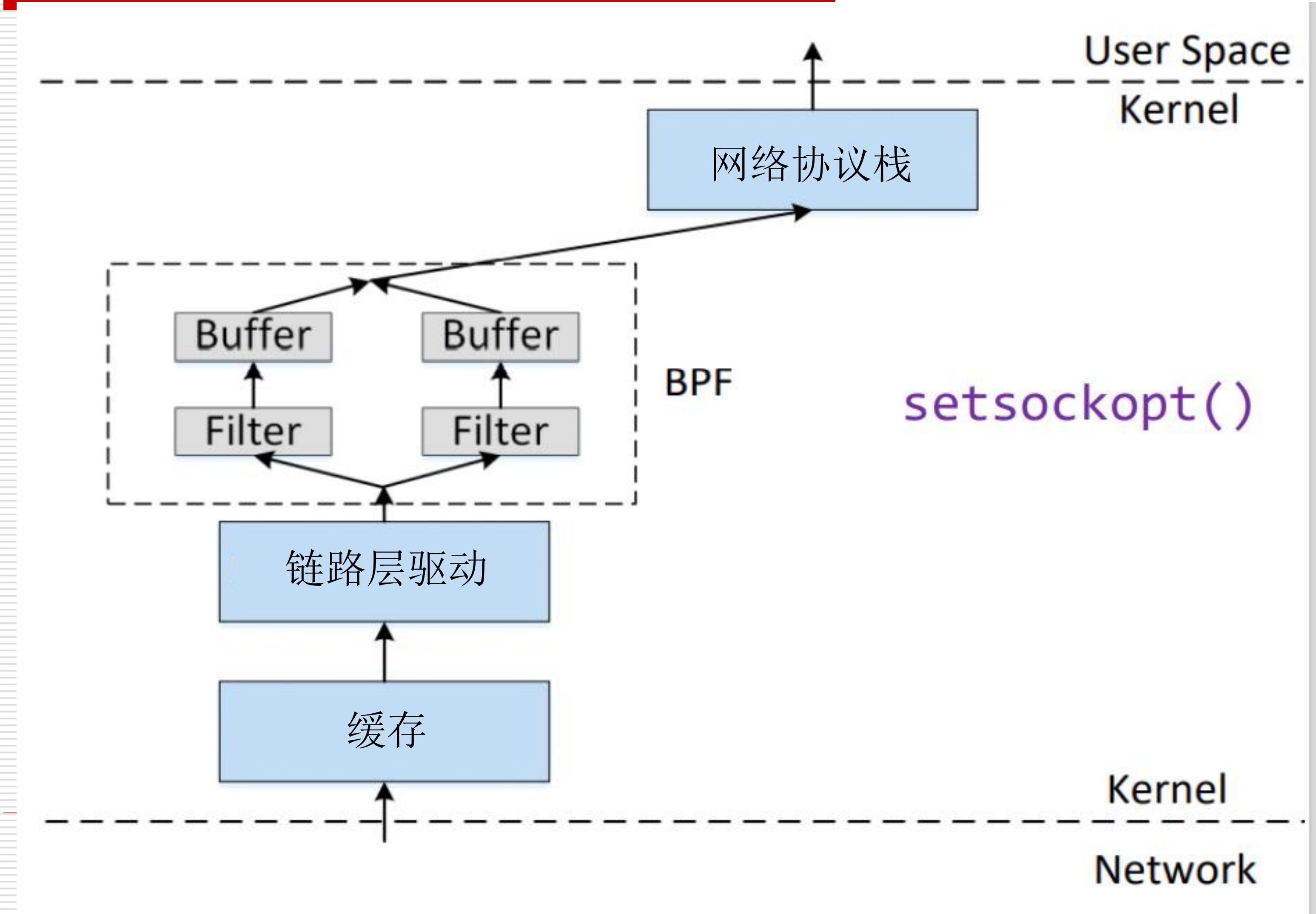


2. 用原始套接字进行报文捕获



```c
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;
}
```

2. 过滤出不想要的报文


User Space
Kernel
网络协议栈
Buffer
Buffer
Filter
Filter
BPF
setsockopt()
链路层驱动
缓存
Kernel
Network

2. PCAP: Packet Capture API

- 最开始来源于**tcpdump**
- 多平台支持:
  - Linux: libpcap
  - Windows: winpcap 和npcap
- **C**语言写的，实现语言实现封装
- 基于**pcap**的工具:
  - Wireshark,tcpdump，scapy,McAfee, nmap, snort

2. 用pcap写的监听程序

```
int main()
{
        pcap_t *handle;
        char errbuf[PCAP_ERRBUF_SIZE];
        struct bpf_program fp;
        char filter_exp[] = “udp or icmp ";
        bpf_u_int32 net;

        // Step 1: Open live pcap session on NIC with interface name
        handle = pcap_open_live("enp0s3", 8192, 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;
}
```

编译：gcc –o sniff sniff.c -lpcap

2. 用pcap写的监听程序

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
        printf("\nGot a packet\n");
        struct ethheader *eth=(struct ethheader *)packet;

        if(ntohs(eth->ether_type) == 0x800)
        {
                struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));
                printf("        From: %s\n",inet_ntoa(ip->iph_sourceip));
                printf("        To: %s\n",inet_ntoa(ip->iph_destip));

                switch(ip->iph_protocol) {
                        case IPPROTO_TCP:
                                printf("        Protocol: TCP\n");
                                break;
                        case IPPROTO_UDP:
                                printf("        Protocol: UDP\n");
                                break;
                        case IPPROTO_ICMP:
                                printf("        Protocol: ICMP\n");
                                break;
                        default:
                                printf("        Protocol: Others\n");
                                break;
                        }
}        }
```

2. pcap过滤器的例子

- □ **dst host 10.0.2.5:只捕获目的ip为10.0.2.5的数据包**
- □ **src host 10.0.2.6:只捕获源ip为10.0.2.6的数据包**
- □ **host 10.0.2.6 and src port 9090: 只捕获源或目的为10.0.2.6，并且源端口为9090的数据包**
- □ **proto tcp: 只捕获tcp数据包**

2. Python + scapy

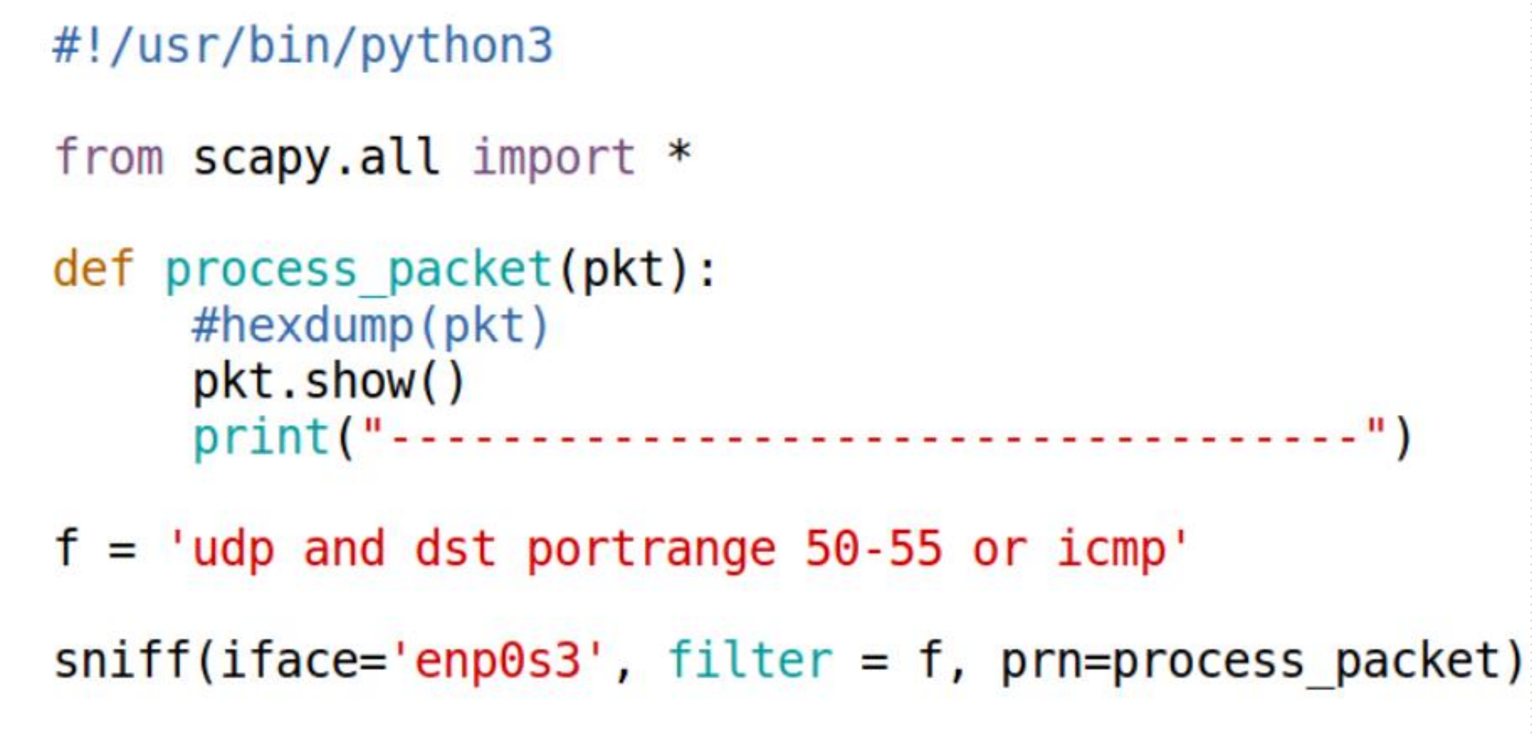
- **Scapy安装**
  - sudo apt install python-scapy
- **Python程序中加载scapy模块**
  - from scapy.all import *

2. Sniffer 例程1

```
#!/usr/bin/python3
from scapy.all import *
pkt = sniff(iface=‘docker0’,
            filter=‘icmp or udp’,
            count=10)
pkt.summary()
```

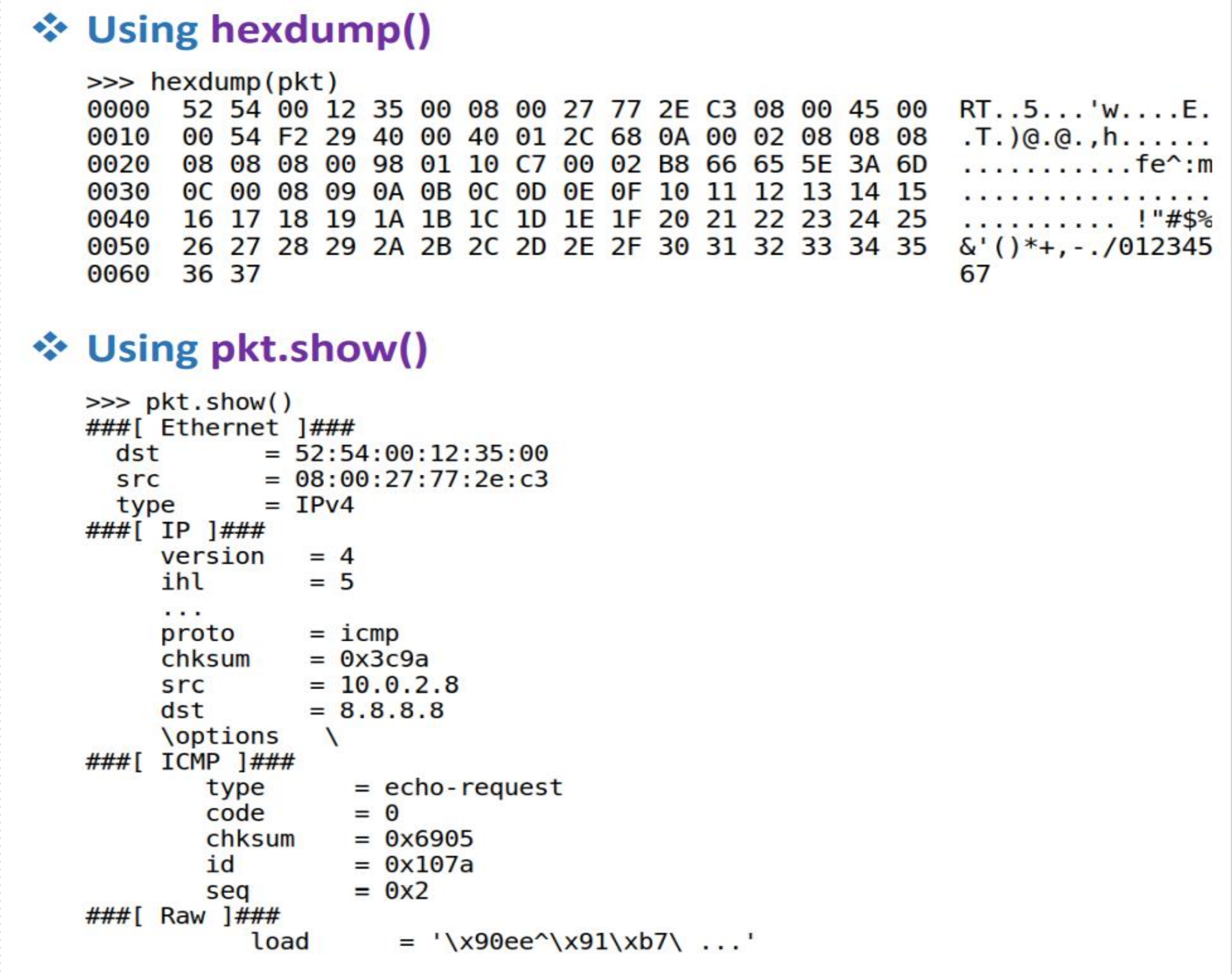
2. Sniffer 例程2

- 执行回调函数



```
#!/usr/bin/python3

from scapy.all import *

def process_packet(pkt):
     #hexdump(pkt)
     pkt.show()
     print("----------------------------------")

f = 'udp and dst portrange 50-55 or icmp'

sniff(iface='enp0s3', filter = f, prn=process_packet)
```

2. 显示报文的不同方式



❖ Using hexdump()

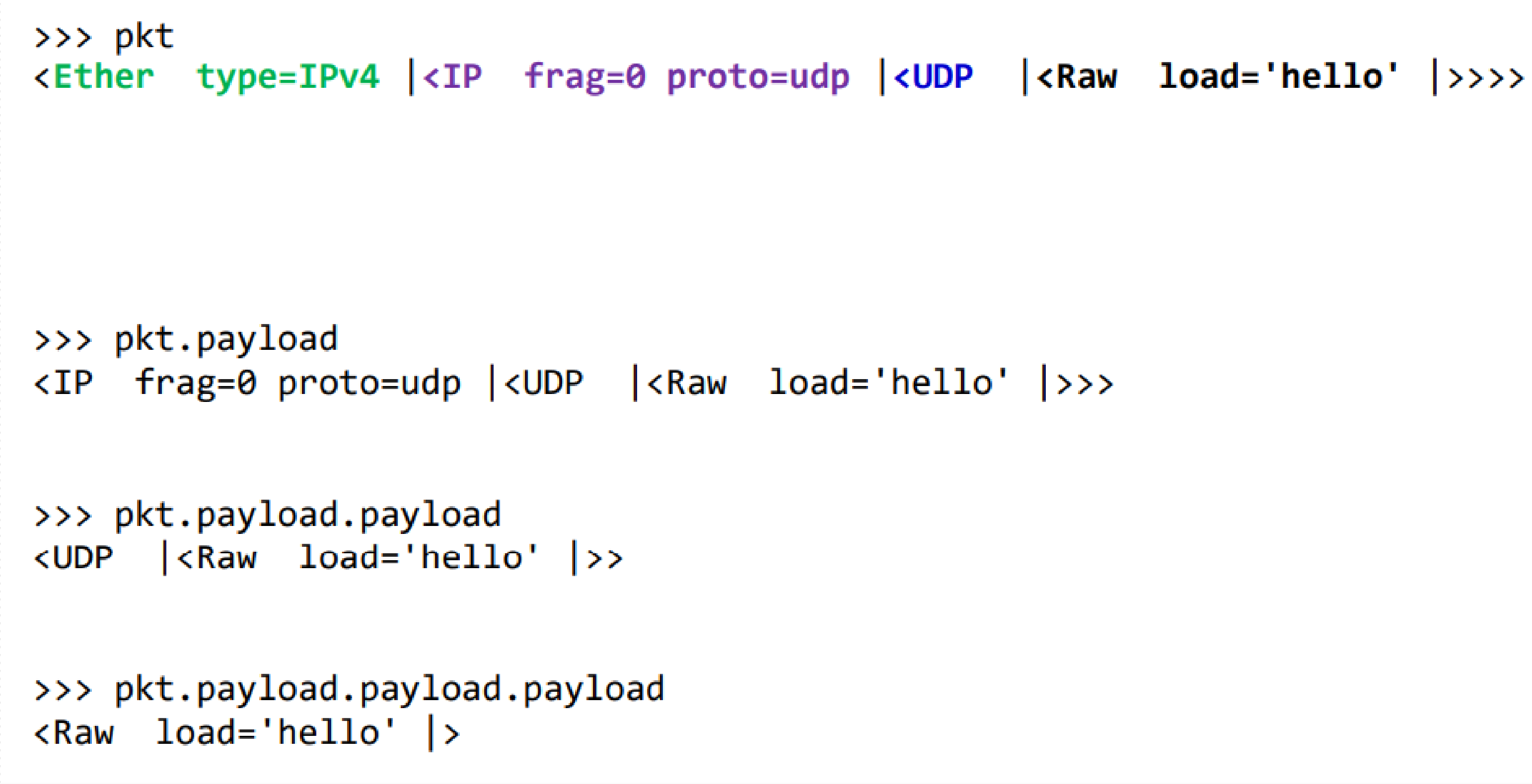
```
>>> hexdump(pkt)
0000   52 54 00 12 35 00 08 00 27 77 2E C3 08 00 45 00   RT..5...'w....E.
0010   00 54 F2 29 40 00 40 01 2C 68 0A 00 02 08 08 08   .T.)@.@.,h......
0020   08 08 08 00 98 01 10 C7 00 02 B8 66 65 5E 3A 6D   ..........fe^:m
0030   0C 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15   ................
0040   16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25   .......... !"#$%
0050   26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35   &'()*+,-./012345
0060   36 37                                             67
```

❖ Using pkt.show()

```
>>> pkt.show()
###[ Ethernet ]###
  dst         = 52:54:00:12:35:00
  src         = 08:00:27:77:2e:c3
  type        = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     ...
     proto     = icmp
     chksum    = 0x3c9a
     src       = 10.0.2.8
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x6905
        id        = 0x107a
        seq       = 0x2
###[ Raw ]###
           load      = '\x90ee^\x91\xb7\ ...'
```

2. 理解scapy的层次

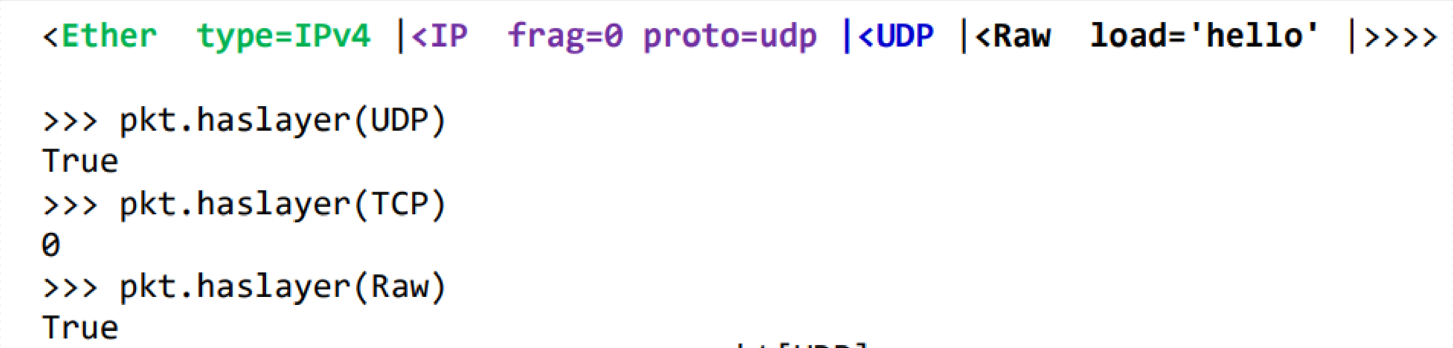
□ 通过层堆叠



```
>>> pkt
<Ether  type=IPv4 |<IP  frag=0 proto=udp |<UDP  |<Raw  load='hello' |>>>>




>>> pkt.payload
<IP  frag=0 proto=udp |<UDP  |<Raw  load='hello' |>>>


>>> pkt.payload.payload
<UDP  |<Raw  load='hello' |>>


>>> pkt.payload.payload.payload
<Raw  load='hello' |>
```

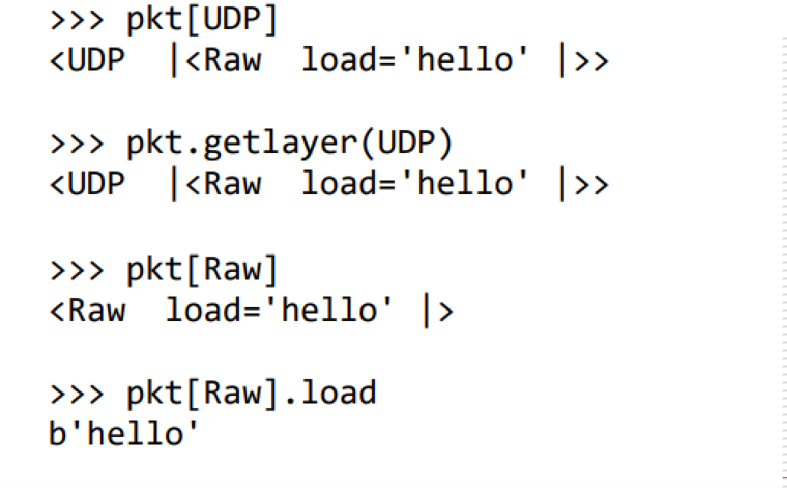
2. 访问层

- □ 检检查层的类型



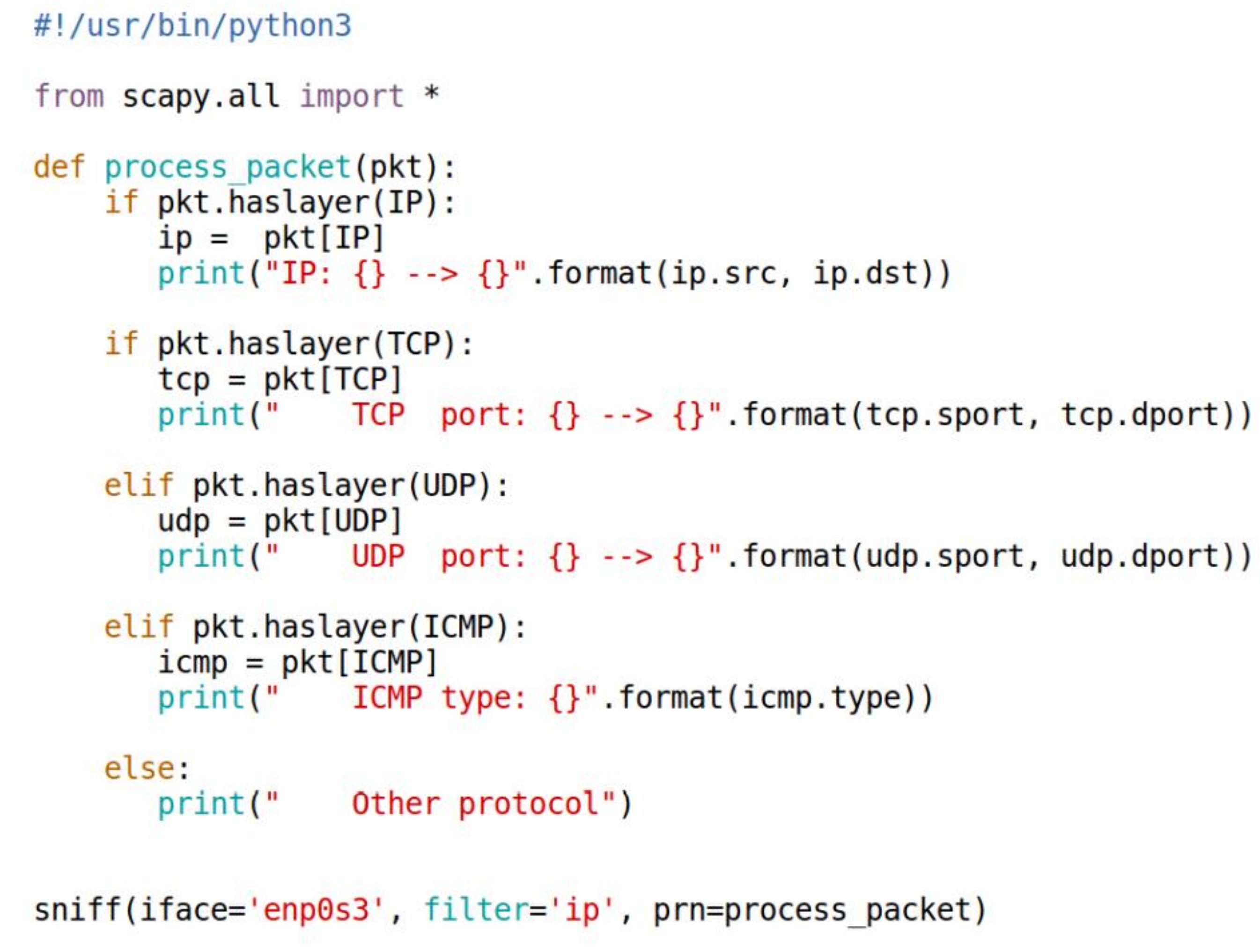
```
<Ether  type=IPv4 |<IP  frag=0 proto=udp |<UDP |<Raw  load='hello' |>>>>

>>> pkt.haslayer(UDP)
True
>>> pkt.haslayer(TCP)
0
>>> pkt.haslayer(Raw)
True
```

- □ 访问层



```
>>> pkt[UDP]
<UDP  |<Raw  load='hello' |>>

>>> pkt.getlayer(UDP)
<UDP  |<Raw  load='hello' |>>

>>> pkt[Raw]
<Raw  load='hello' |>

>>> pkt[Raw].load
b'hello'
```

2. Sniffer 例子3

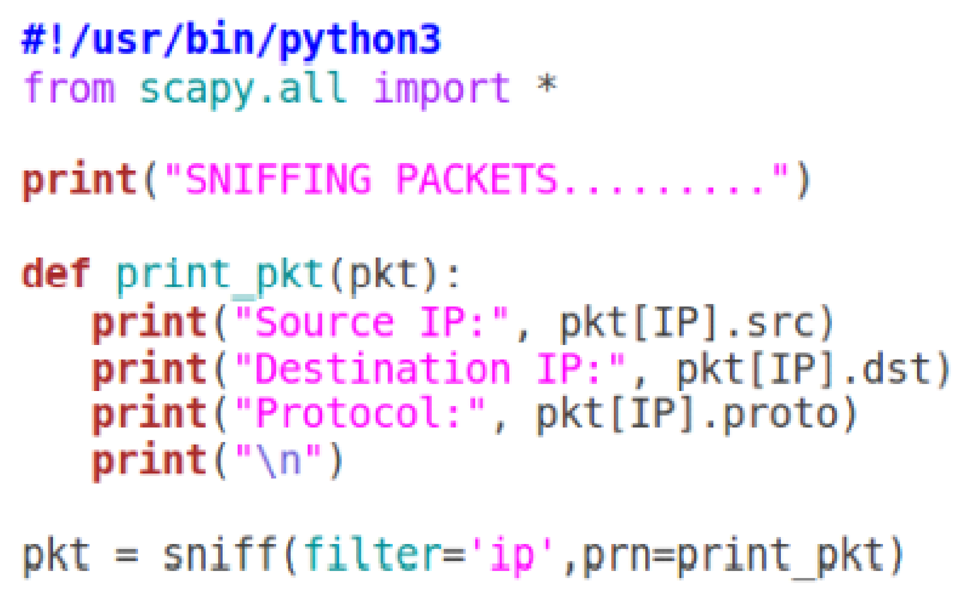


```
#!/usr/bin/python3

from scapy.all import *

def process_packet(pkt):
    if pkt.haslayer(IP):
       ip =  pkt[IP]
       print("IP: {} --> {}".format(ip.src, ip.dst))

    if pkt.haslayer(TCP):
       tcp = pkt[TCP]
       print("    TCP  port: {} --> {}".format(tcp.sport, tcp.dport))

    elif pkt.haslayer(UDP):
       udp = pkt[UDP]
       print("    UDP  port: {} --> {}".format(udp.sport, udp.dport))

    elif pkt.haslayer(ICMP):
       icmp = pkt[ICMP]
       print("    ICMP type: {}".format(icmp.type))

    else:
       print("    Other protocol")


sniff(iface='enp0s3', filter='ip', prn=process_packet)
```

2. 获得协议类的信息

- 获得属性名字
  - ls(IP)
  - ls(TCP)
  - ..
- 获得方法名字
  - help(IP)
  - help(TCP)

2. 任务1：用scapy监听报文



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

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

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

pkt = sniff(filter='ip',prn=print_pkt)
```

filter=‘ip dst 172.17.0.1’
filter=‘ip dst 172.17.0.1 or tcp port 23

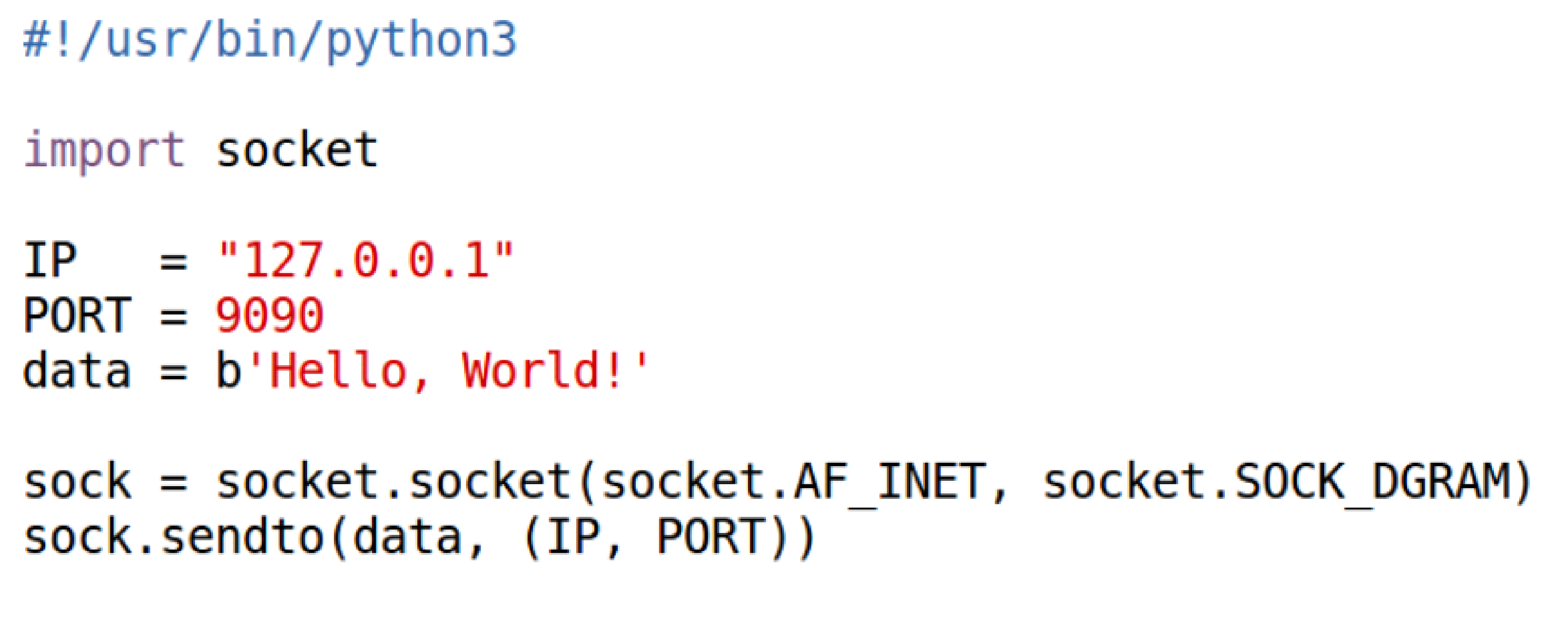
- □ **sudo python sniff.py**
- □ 是否能捕获到其它主机的流量?
- □ 试一试其它过滤器
  - ■ http://biot.com/capstats/bpf.html

3. 报文伪造

- Python：Scapy
- C语言：Raw socket
- C语言：pcap

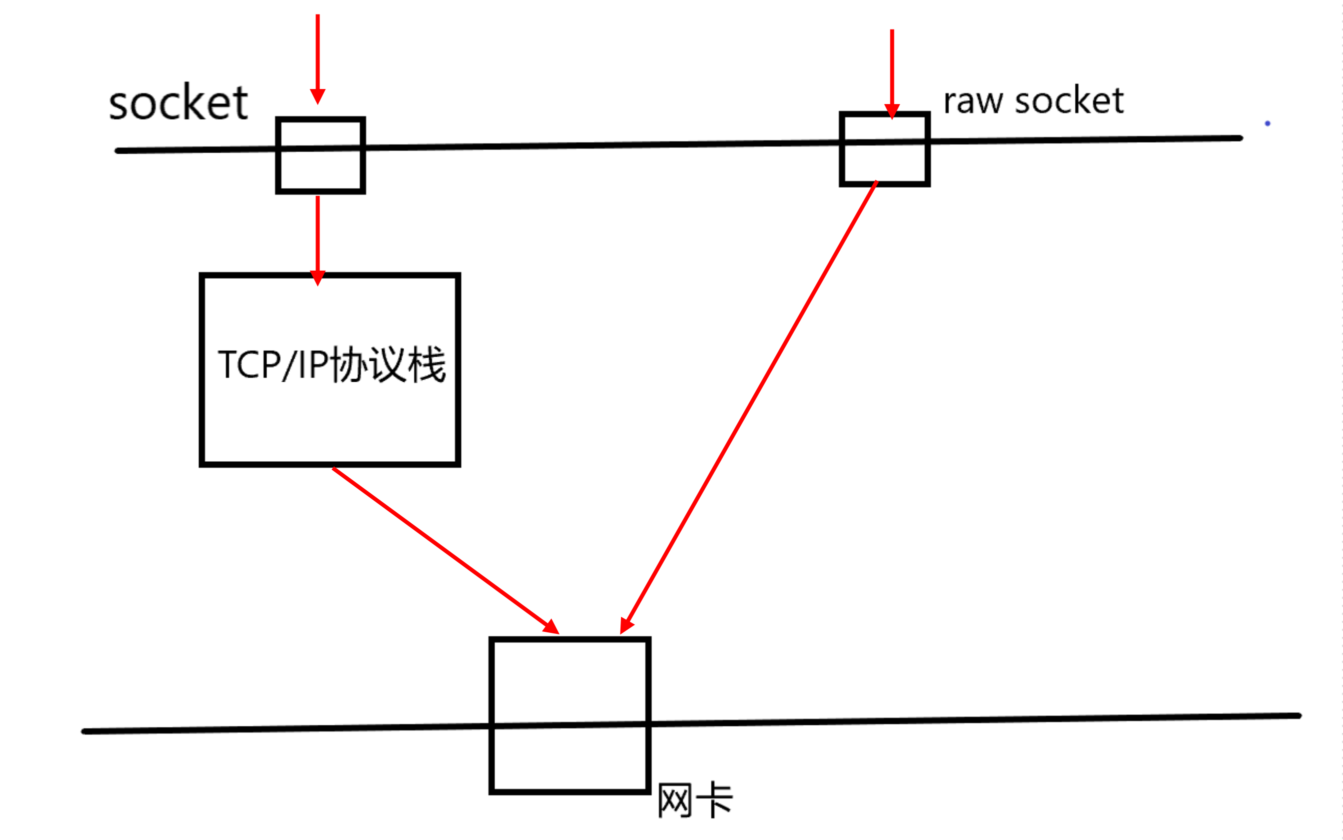
3. 报文伪造

□ 发送报文（python中的socket)

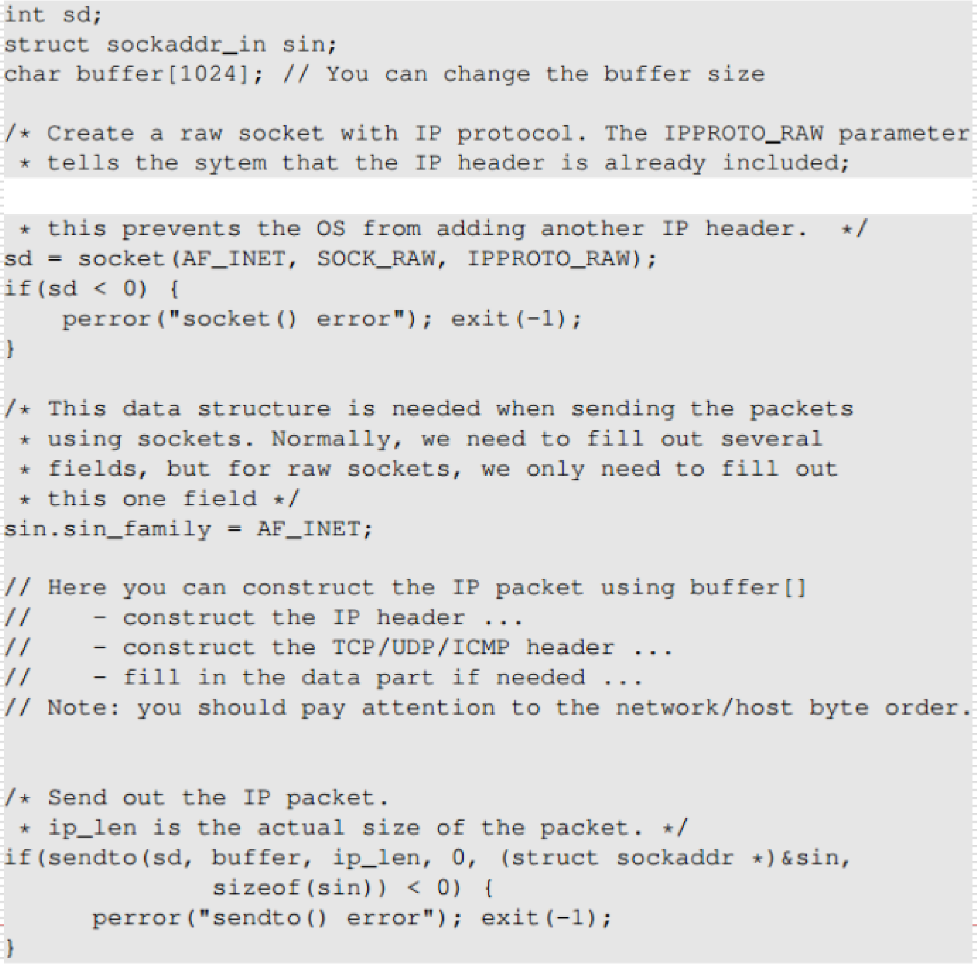


```
#!/usr/bin/python3

import socket

IP   = "127.0.0.1"
PORT = 9090
data = b'Hello, World!'

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(data, (IP, PORT))
```

3. 用原始套接字伪造报文



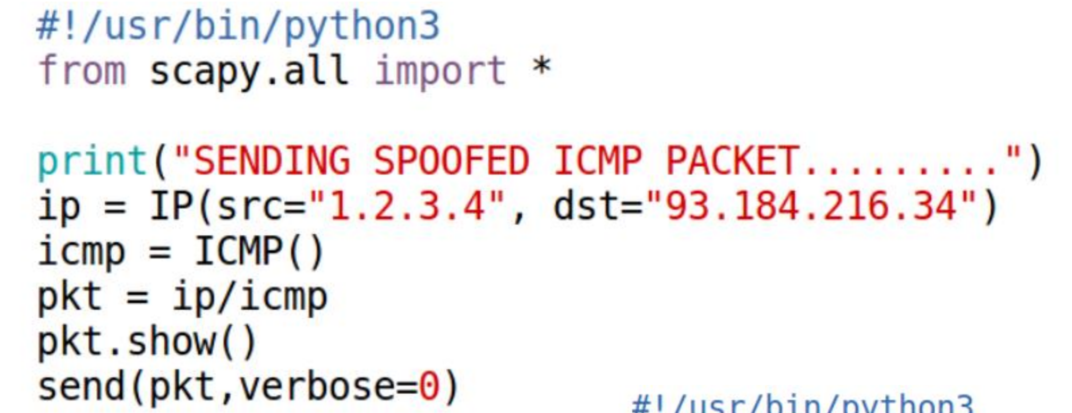
3. 原始套接字发送报文



```
int sd;
struct sockaddr_in sin;
char buffer[1024]; // You can change the buffer size

/* Create a raw socket with IP protocol. The IPPROTO_RAW parameter
 * tells the sytem that the IP header is already included;
 * this prevents the OS from adding another IP header.  */
sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
if(sd < 0) {
    perror("socket() error"); exit(-1);
}

/* This data structure is needed when sending the packets
 * using sockets. Normally, we need to fill out several
 * fields, but for raw sockets, we only need to fill out
 * this one field */
sin.sin_family = AF_INET;

// Here you can construct the IP packet using buffer[]
//    - construct the IP header ...
//    - construct the TCP/UDP/ICMP header ...
//    - fill in the data part if needed ...
// Note: you should pay attention to the network/host byte order.


/* Send out the IP packet.
 * ip_len is the actual size of the packet. */
if(sendto(sd, buffer, ip_len, 0, (struct sockaddr *)&sin,
              sizeof(sin)) < 0) {
      perror("sendto() error"); exit(-1);
}
```

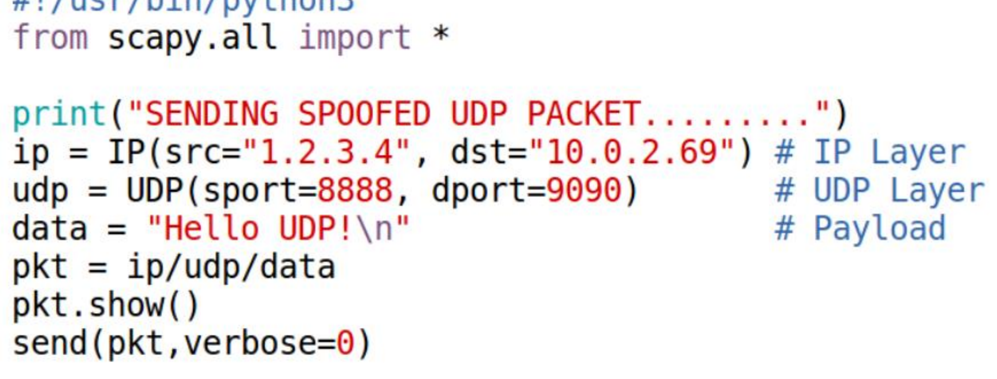
3. 用scapy伪造报文

□ 伪造ICMP报文



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

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

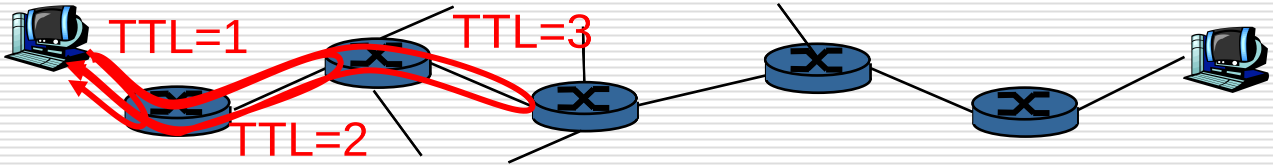
□ 伪造UDP报文



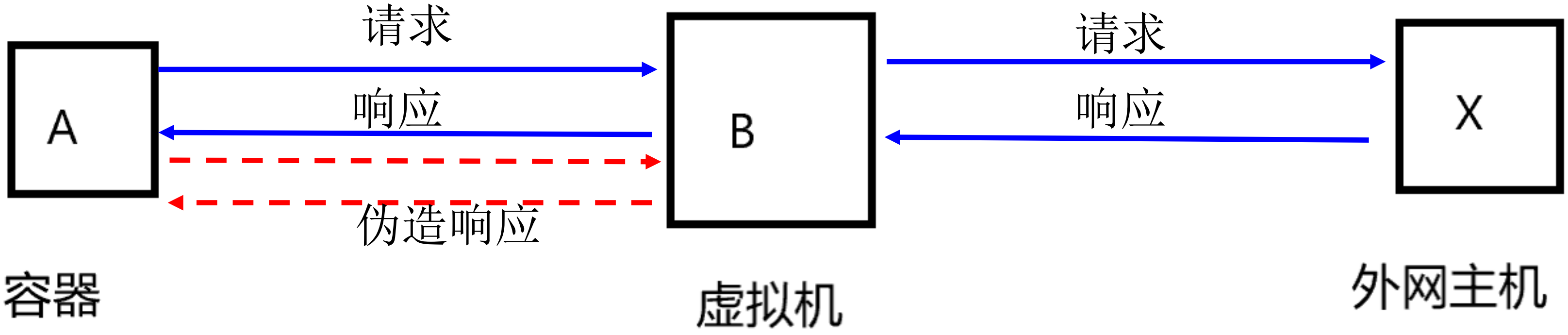
```
#!/usr/bin/python3
from scapy.all import *

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
pkt.show()
send(pkt,verbose=0)
```

3. Scapy构造报文实验：路由追踪

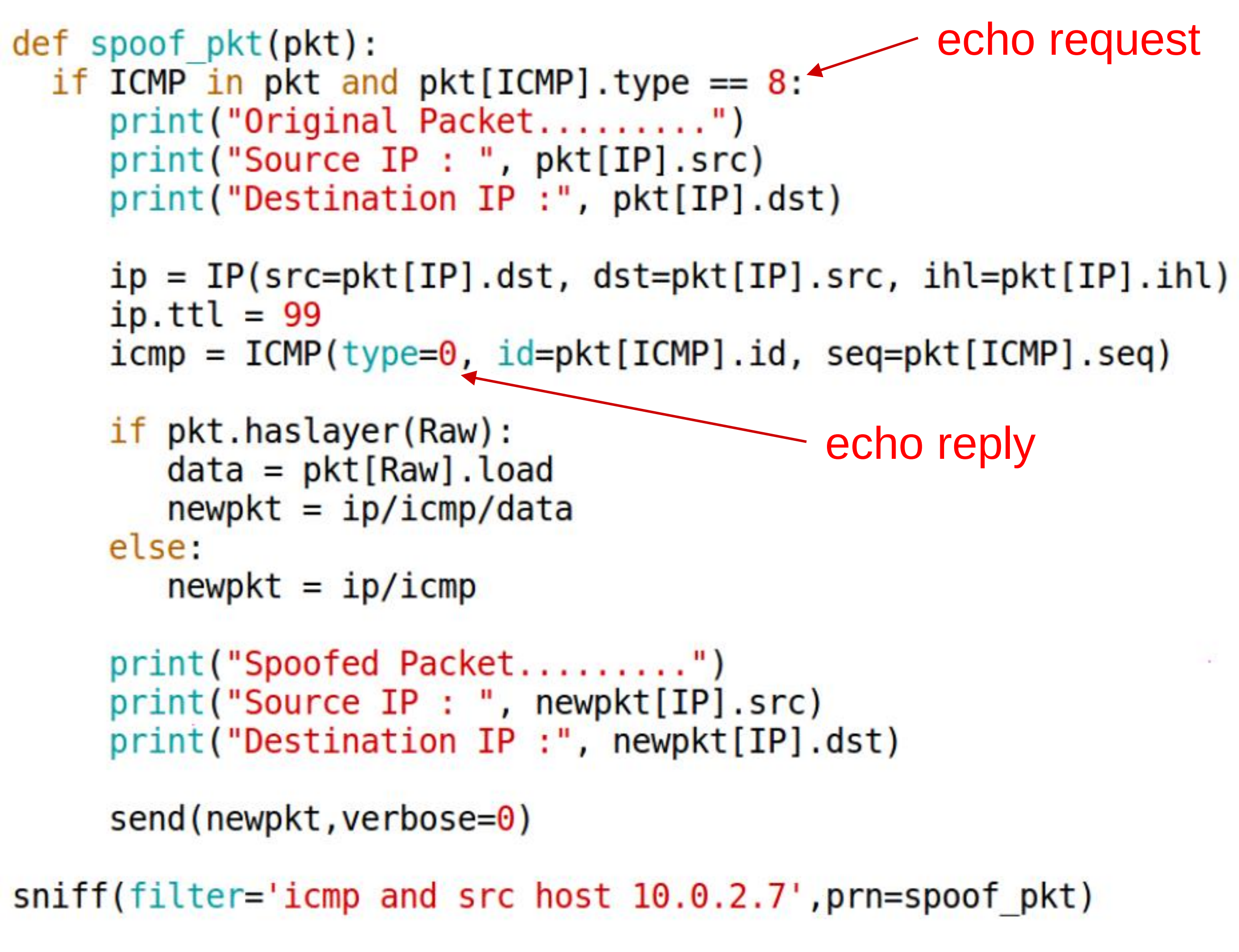
- **Traceroute 程序：为从源端到目的地的因特网端到端路径上的路由器提供时延计量方法。 对所有到目的地路径上的路由器 *i*:**
  - 发送分组,设定TTL=i
  - 路由器 *i* 将向发送者返回ICMP差错信息(TTL超时）
  - 发送者计算发送分组和收到响应之间的时间间隔



4. 监听请求伪造回应



4. Sniff-and-spoof例子



```
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)
     ip.ttl = 99
     icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)

     if pkt.haslayer(Raw):
        data = pkt[Raw].load
        newpkt = ip/icmp/data
     else:
        newpkt = ip/icmp

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

     send(newpkt,verbose=0)

sniff(filter='icmp and src host 10.0.2.7',prn=spoof_pkt)
```

echo request

echo reply

4. 任务3：

- 在虚拟机上运行
- **Python sniff_spoof_icmp.py**
- **在docker里面运行:**
  - ping 1.2.3.4
  - ping 8.8.8.8
  - ping 不存在的地址

是否都能收到回应？

5. Scapy 和 C比较

- □ 缺省值
- □ 包（字节数组）对象/结构
- □ 性能（实验）
  - Scapy：发包106个/秒
  - C：　发包4000个/秒
- □ 代码量

6. 字节顺序

- 大端（big-endian)和小端(little-endian)
  - 大端字节顺序意味着先从大的字节保存，即大的字节放在内存的低端；
  - 小端字节序意味着先从小的字节保存，即小的字节放在内存的低端；

0x87654321

在内存中的存放

0x1003	0x87
0x1002	0x65
0x1001	0x43
0x1000	0x21

小端

0x1003	0x21
0x1002	0x43
0x1001	0x65
0x1000	0x87

大端

6. 字节顺序转换函数

- □ **htons():** 无符号短整型数从主机字节序→网络字节序
- □ **htonl():**无符号长整型数从主机字节序→网络字节序
- □ **ntohs():**无符号短整型数从网络字节序→主机字节序
- □ **ntohl():**无符号短整型数从网络字节序→主机字节序

7. 实验任务

- **按照指导手册进行实验，完成问题，在微助教平台提交**
  - 设置过滤规则，嗅探数据包;
  - 构造报文，实现路由探测（可以程序发包，wireshark分析ICMP差错报告，记录路径上的路由器）（虚拟机设置成桥接模式）
  - 监听并伪造报文（ping 不存在的地址如1.2.3.4，能伪造回应）
  - C语言和python分别实现