

中山大学移动信息工程学院移动网络安全实  
验报告  
(2016学年春季学期)

刘爽  
班级：13M2  
ID:13354222  
专业：移动互联网

2016年4月24日

# 1 实验题目

## Packet Sniffing and Spoofing

### 1.1 实验描述

数据包的监听和假冒是网络完全中那个很重要的概念。它们是网络通信中的主要威胁。理解这两个概念对于理解网络安全措施有很重要的意义。现今市面上有数据包的监听和夹袄工具，如Wirshark, Tcpdump, Netox等。这其中有些工具在安全专家和攻击者之中被广泛使用着。学会使用这些工具对学生来说非常重要。更重要的是，他们能够在课堂上理解这些工具是如何工作的。比如，对数据包的监听和仿冒是如何用软件实现的。本次实验是要同学理解这两个功能在软件上是怎么实现的，读懂他们的源码，深入理解，然后根据具体要求编写自己的数据包监听和仿冒程序。

### 1.2 实验要求

完成题目所给予的两个Tasks。

Task1：Packet Sniffing

使用pcap库使得编写数据包监听的程序非常简单。pcap库让监听的任务变成了按顺序调用库中各种函数的简单步骤。在最后，被捕获到的数据包会被缓存起来以备后面的处理。

Task2：Packet Spoofing

通常情况下，当用户发送一个数据包时，操作系统不允许用户设置协议的头部全部内容，如TCP，UDP和IP的头部。操作系统自身会进行大部分的设置，而用户只能进行小部分的设置，如目的IP地址，目的端口号等。然而，如果用户有root privilege，他们则可以设置数据包头的任何内容。这种行为叫数据包的仿冒，并且可以通过raw socket来实现。raw socket给予程序员对数据包结构修改权利，它允许程序员生成随机包，以及设置包的包头和数据内容。

使用raw sockets包含四个步骤：

- I. 创建一个raw socket。
- II. 创建一个socket选项。
- III. 生成一个包。
- IV. 通过raw socket送出一个包。

### 1.3 实验目的

理解并学会使用pcap库。

理解使用pcap库进行数据包的sniffing的全过程。

理解使用pcap库进行数据包的spoofing的全过程。

## 2 实验原理

### 2.1 Task1

对于pcap库中监听包的整体过程大致分为下面的步骤：

- 1.确定监听哪个端口。
- 2.初始化pcap

告诉pcap我们需要监听哪些端口（支持多端口监听）。

### 3.制定监听规则

规则会被放在一个字符串里面，通过调用一个函数就可以完成规则的编译，然后再让pcap读取规则。

### 4.执行监听

可以设置pcap持续监控信道，只要信道一有信息就会被抓下，从而我们可以对接收到的内容进行后续的操作。

### 5.关闭pcap，结束监听。

## 2.2 Task2

通过socket编程自己去填写一个虚假的IP头部，从而生成一个仿冒一个虚假的数据包并发到网络上。

步骤如下：

- 创建一个raw socket。
- 设置socket。
- 构造一个数据包，填充包头。
- 利用raw socket把数据包发出去。

## 3 实验内容

### 3.1 Task1: Sniffing

#### 3.1.1 Task1a

Understanding sniffex. Please download the sniffex.c program from the tutorial mentioned above, compile and run it. You should provide screendump evidence to show that your program runs successfully and produces expected results.

I. Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial.

**Ans :**

#### 1. 设定设备：决定要监听的接口

有两种方法可以做到：

- 把设备的名字作为主函数参数argv的第一个参数传入。
- pcap库通过调用pcap\_lookupdev函数自动设定，将名字作为返回值存储在dev中。若该函数出错，则将错误信息存储在函数参数errbuf 中。

*dev = pcap\_lookupdev(errbuf);*

#### 2. 打开设备：告诉pcap要监听的设备

*pcap\_t \*pcap\_open\_live(char \*device, int snaplen, int promisc, int to\_ms, char \*ebuf)*

**char\* device:** dev, 在上一步中指定的设备名称。  
**int snaplen:** pcap可以监听捕获到的字节数。  
**int promisc:** 是否设置为混杂模式, 1为是, 0 为否。  
**int to\_ms:** time out (以毫秒为单位), 为0时表示在捕获到足够多的数据包之前会一直等待。  
**char\* ebuf:** errbuf, 当出现错误时存储错误信息。

3. 过滤

*pcap\_lookupnet()*

给定设备的名字, 返回它的一个IPv4的网络号和相应的网络掩码。

*int pcap\_compile(pcap\_t \*p, struct bpf\_program \*fp, char\*str, int optimize, bpf\_u\_int32 netmask)*

**pcap\_t \*p:** 上一步中指定的会话句柄pcap\_t\* handle。  
**struct bpf\_program \*fp:** 存储compile之后的filter程序的结构体指针fp。  
**char\* str:** 过滤语句表达式, 即正则表达式形式。  
**int optimize:** 过滤语句表达式是否需要优化, 1为是, 0为否。  
**pcap\_t \*p:** 需要执行过滤操作的网络的子网掩码。  
 运行过程中返回-1表示出现错误, 返回其余整数值表示成功。

*int pcap\_setfilter(pcap\_t \*p, struct bpf\_program \*fp)*

Compile结束后, 执行pcap\_setfilter()函数。

**pcap\_t \*p:** 类似地, 表示上一步中指定的会话句柄pcap\_t\* handle。  
**struct bpf\_program \*fp:** pcap\_compile函数生成的filter程序, 与pcap\_compile函数的参数2相同。

4. 实施监听

*u\_char \*pcap\_next(pcap\_t \*p, struct pcap\_pkthdr \*h)*

一次抓取一个包。

**pcap\_t \*p:** 指定的会话句柄pcap\_t\* handle。  
**struct pcap\_pkthdr \*h:** 指向包含数据包的信息的结构体的指针。  
 这个函数返回u\_char类型的指针, 指向h所描述的数据包。

*int pcap\_loop(pcap\_t \*p, int cnt, pcap\_handler callback, u\_char \*user)*

输入要抓取的包数n, 进入循环, 知道抓取包数达到n为止。

**pcap\_t \*p:** 指定的会话句柄pcap\_t\* handle。  
**int cnt:** >0 说明在返回前应该有多少包应该被抓取。<0 说明要一直到有错误发生为止。  
**pcap\_handler callback:** 回调函数的名字。  
**u\_char \*user:** 基本上=NULL。

6. 关闭会话

*pcap\_close()*

关闭监听。

II. Why do you need the root privilege to run sniffex? Where does the program fail if executed without the root privilege?

**pcap\_lookupdev()**函数需要root权限访问的原因是这个函数需要获得网络接口，而在Linux系统下没有root权限是不可能获得的。sniffer.c需要raw socket来让数据包能够直接发送。我们需要root权限来创建一个新的raw socket，因为没有root权限我们没法发现NIC。

当我们没有使用root权限的时候，运行结果如图1所示。程序会报"Couldn't find default device: no suitable device found."的错误。

```
sean@ubuntu:~/Lab5$ ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Couldn't find default device: no suitable device found
```

Figure 1: Execution without root privilege

III. Please turn on and turn off the promiscuous mode in the sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you demonstrate this.

非混杂模式：只监听与当前主机直接相关的包，如以当前主机为目的地址或源地址。

混杂模式：监听网络中的所有包。

#### Demonstration:

创建两个虚拟机，虚拟机A的ip地址为192.168.161.128，虚拟机B的ip地址为192.168.161.129。

我们让虚拟机B ping一个随机的ip地址，如192.168.161.

```
sean@ubuntu:~$ ping 192.168.161.1
PING 192.168.161.1 (192.168.161.1) 56(84) bytes of data.
64 bytes from 192.168.161.1: icmp_seq=1 ttl=128 time=0.382 ms
64 bytes from 192.168.161.1: icmp_seq=2 ttl=128 time=0.374 ms
64 bytes from 192.168.161.1: icmp_seq=3 ttl=128 time=1.32 ms
64 bytes from 192.168.161.1: icmp_seq=4 ttl=128 time=0.286 ms
64 bytes from 192.168.161.1: icmp_seq=5 ttl=128 time=0.321 ms
```

让虚拟机A在混杂模式和非混杂模式下进行sniffing，看捕获数据包的情况。修改数据包的格式为icmp。

```
char filter_exp[] = "icmp";          /* filter expression [3] */
struct bpf_program fp;               /* compiled filter program
ion) */
bpf_u_int32 mask;                    /* subnet mask */
bpf_u_int32 net;                     /* ip */
int num_packets = 10;                /* number of packets to capture
```

将pcap\_open\_live函数第二个参数变为0，开启非混杂模式。

非混杂模式:

结果如图2所示，在非混杂模式下，虚拟机A对于与自身ip地址无关数据包不会进行捕获，因而不会有任何数据包的信息出现。

混杂模式:

结果如Fig.3所示，在混杂模式下，虚拟机A对于与自身ip地址无关的数据包也会进行捕获，因而会抓取指定数量的(10)数据包之后程序才会停止。

```
sean@ubuntu:~/Lab5$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth0
Number of packets: 10
Filter expression: icmp
```

Figure 2: Result in non-promiscuous mode

### 3.1.2 Task1b

Writing Filters. Please write filter expressions for your sniffer program to capture each of the followings.

In your lab reports, you need to include screendumps to show the results of applying each of these filters.

- Capture the ICMP packets between two specific hosts.
- Capture the TCP packets that have a destination port range from to port 10-100.

```
sean@ubuntu:~/Lab5$ sudo ./sniffex
[sudo] password for sean:
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM

Device: eth0
Number of packets: 10
Filter expression: icmp

Packet number 1:
  From: 192.168.161.1
  To: 192.168.161.129
  Protocol: ICMP

Packet number 2:
  From: 192.168.161.129
  To: 192.168.161.1
  Protocol: ICMP
```

Figure 3: Result in promiscuous mode

#### 抓取ICMP包：

如Task1a中No.3 Problem所示，我们只需要将filter\_exp数组中的内容由"ip"改为"icmp"即可。

结果如下所示：

虚拟机B：负责随便ping一个ip地址。如Fig.4所示。

```
sean@ubuntu:~$ ping 192.168.161.1
PING 192.168.161.1 (192.168.161.1) 56(84) bytes of data.
64 bytes from 192.168.161.1: icmp_seq=1 ttl=128 time=0.382 ms
64 bytes from 192.168.161.1: icmp_seq=2 ttl=128 time=0.374 ms
64 bytes from 192.168.161.1: icmp_seq=3 ttl=128 time=1.32 ms
64 bytes from 192.168.161.1: icmp_seq=4 ttl=128 time=0.286 ms
64 bytes from 192.168.161.1: icmp_seq=5 ttl=128 time=0.321 ms
```

Figure 4: Operation of VM B

虚拟机A：负责抓取icmp数据包。

结果如Fig.3所示，在混杂模式下可以捕获到相应的icmp包。

#### 抓取TCP包：

将filter\_exp数组中的内容由"ip"改为"tcp dst portrang 10-100"。

```
char filter_exp[] = "tcp";          /* filter expression [3] */
struct bpf_program fp;              /* compiled filter program
sion) */
bpf_u_int32 mask;                   /* subnet mask */
bpf_u_int32 net;                   /* ip */
```

结果如下所示：

虚拟机B向虚拟机A进行telnet发送，TCP模式，端口为23。

```
sean@ubuntu:~$ telnet 192.168.30.128
Trying 192.168.30.128...
Connected to 192.168.30.128.
Escape character is '^]'.
Ubuntu 14.04 LTS
```

虚拟机A抓取端口号为10-100的包，数量为100，如Fig.5所示。

### 3.1.3 Task1c

Sniffing Passwords. Please show how you can use sniffex to capture the password when somebody is using telnet on the network that you are monitoring. You may need to modify the sniffex.c a little bit if needed. You also need to start the telnetd server on your VM. If you are using our pre-built VM, the telnetd server is already installed; just type the following command to start it.

将filter\_exp数组中的内容由"ip"改为"tcp and port 23"。这样只有telnet发送的包才能够被抓捕到。如Fig.6所示。

实验结果：

虚拟机B开启telnet服务之后，用telnet向虚拟机A发送，端口号为23。如Fig.7所示。

虚拟机A监听端口号为23发出的TCP包，并成功捕获。如Fig.8所示。

```

^Csean@ubuntu:~/Lab5$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth0
Number of packets: 100
Filter expression: tcp dst portrange 10-100

Packet number 1:
    From: 192.168.30.129
    To: 192.168.30.128
    Protocol: TCP
    Src port: 40546
    Dst port: 23

```

Figure 5: Result in TCP capturing

```

char filter_exp[] = "tcp and port 23";          /* filter expression */
struct bpf_program fp;                          /* compiled filter program */
ion) */
bpf_u_int32 mask;                               /* subnet mask */
bpf_u_int32 net;                                /* ip */
int num_packets = 100;                         /* number of packets to capture */

```

Figure 6: Modified monitoring port in sniffex.c

```

sean@ubuntu:~$ sudo service openbsd-inetd start
[sudo] password for sean:
 * Starting internet superserver inetd
sean@ubuntu:~$ telnet 192.168.30.128
Trying 192.168.30.128...
Connected to 192.168.30.128.
Escape character is '^]'.

telnet>
Ubuntu 14.04 LTS
ubuntu login: sean
Password:
Last login: Thu Apr 21 20:37:06 PDT 2016 from localhost on pts/6
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic i686)

```

Figure 7: Sending TCP packets by telnet



```

sean@ubuntu:~/Lab5$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: eth0
Number of packets: 100
Filter expression: tcp and port 23

Packet number 1:
  From: 192.168.30.128
  To: 192.168.30.129
  Protocol: TCP
  Src port: 23
  Dst port: 40547
  Payload (6 bytes):
000000  ff fa 21 01 ff f0

```

Figure 8: Capturing TCP packets by telnet

## 3.2 Task2:Spoofing

### 3.2.1 Task2.a

Write a spoofing program. You can write your own packet program or download one. You need to provide evidences (e.g., Wireshark packet trace) to show us that your program successfully sends out spoofed IP packets. 因为在TCP协议下，发送的是IP包，因为我们创建一个文件tcp.c，源地址和目的地址如下图所示。

```

close (sd);
printf ("Index for interface %s is %i\n", interface, ifr.ifr_ifindex);

// Source IPv4 address: you need to fill this out
strcpy (src_ip, "192.168.1.132");

// Destination URL or IPv4 address: you need to fill this out
strcpy (target, "www.google.com");

```

实验结果如下图所示：wireshark截取到的最后一个包源地址与所设置的相同，为192.138.1.132，目的地址为google域名对应的IP地址。

7931	1890.0279656	192.168.161.130	91.189.88.150	TCP	60	57507 >
7932	1890.0280836	91.189.88.150	192.168.161.130	TCP	60	http > 5
13464	44034.007787	192.168.1.132	216.58.197.100	TCP	54	60 > htt

此结果证明程序可以成功发送一个仿冒过的IP包。

### 3.2.2 Task2.b

Spoof an ICMP Echo Request. Spoof an ICMP echo request packet on behalf of another machine (i.e., using another machine's IP address as its source IP address). This packet should be sent to a remote machine on the Internet (the machine must be alive). You should turn on your Wireshark, so if your spoofing is successful, you can see the echo reply coming back

from the remote machine.

创建一个icmp.c文件，用以发送icmp包，模拟ping的效果，其中源地址和目的地址如下图所示。真正的主机地址是192.168.161.128，与设置的源地址是不同的。

```
// Source IPv4 address: you need to fill this out
strcpy (src_ip, "192.168.161.130");

// Destination URL or IPv4 address: you need to fill this out
strcpy (target, "www.yahoo.com");
```

实验结果如下图所示，Yahoo的域名对应的IP地址是116.216.12.74，源地址是之前设置好的192.168.161.130，第一个包是由程序发出的，模拟一个主机ping Yahoo域名的效果。第二个包是Yahoo的主机返回的一个reply，目的地址是192.168.161.130，而源地址是Yahoo域名对应的IP地址。这证明，Yahoo的主机被成功欺骗，ping成功。

13643	44867.537887	192.168.161.130	116.214.12.74	ICMP	46 Echo (pi
13646	44867.668987	116.214.12.74	192.168.161.130	ICMP	60 Echo (pi

实验结果显示Spoofing成功。

### 3.2.3 Task2.c

Spoof an Ethernet Frame. Spoof an Ethernet Frame. Set 01:02:03:04:05:06 as the source address. To tell the system that the packet you construct al-

ready includes the Ethernet header, you need to create the raw socket using the following parameters:

```
sd = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_IP))
```

When constructing the packets, the beginning of the buffer[] array should now be the Ethernet header.

创建一个icmp\_mac.c文件，发送修改过mac地址的ICMP包。如下图所示，主动设置为源mac地址为01:02:03:04:05:06。而源IP地址和目的IP地址都是Task2.b中的，并没有变。

```
src_mac[0] = 0x01;
src_mac[1] = 0x02;
src_mac[2] = 0x03;
src_mac[3] = 0x04;
src_mac[4] = 0x05;
src_mac[5] = 0x06;
```

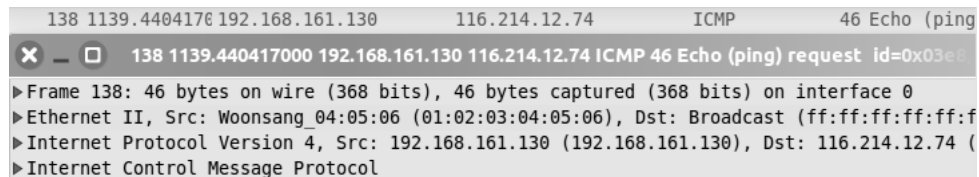
结果如图9所示，程序运行的包输出的源Mac地址为设置的01:02:03:04:05:06。如图10所示，wireshark截取到icmp包，源地址为192.168.161.130，在包的详细信息中，源Mac地址为设置好的01:02:03:04:05:06。由结果可知，仿冒一个帧是可行的。

### Questions.

Please answer the following questions.

```
sean@ubuntu:~/Lab5$ gcc -Wall -o icmp_mac icmp_mac.c -lpcap
sean@ubuntu:~/Lab5$ sudo ./icmp_mac
MAC address for interface eth0 is 01:02:03:04:05:06
Index for interface eth0 is 2
```

Figure 9: Output of icmp\_mac.c program



The image shows a Wireshark packet capture window. The top packet list shows a packet of 46 bytes, type ICMP Echo (ping) request, from 192.168.161.130 to 116.214.12.74. The packet details pane shows the following structure:

- Frame 138: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface 0
- Ethernet II, Src: Woonsang\_04:05:06 (01:02:03:04:05:06), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
- Internet Protocol Version 4, Src: 192.168.161.130 (192.168.161.130), Dst: 116.214.12.74 (116.214.12.74)
- Internet Control Message Protocol

Figure 10: Output of icmp\_mac.c program

1. Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

**Ans:**

不可以，当IP包很长的時候，需要进行Fragmentation，如果IP包的长度随便填一个值的话，Fragmentation不能够正确执行。这样在接收方接收到所有的Fragments并尝试组装的时候，会发现checksum不对，那么接收方会丢弃掉这个包，达不到Spoofing的目的。

2. Using the raw socket programming, do you have to calculate the checksum for the IP header?

**Ans:**

需要计算checksum。因为接受方需要检查checksum之后并确认包的数据没有出错才会接受。如果不计算checksum那么checksum的值就会是随机的，有很大可能接收方会发现checksum不对而丢弃这个包，这样达不到Spoofing的目的。

## 4 实验感想

此次实验需要实现sniff和spoof功能，实验内容较多，耗时较长，有以下感想和收获。

1. 在实现过程很复杂的实验中，理解原理相对更重要一些。此次实验不强制要求自己手动实现全部代码，只要能够理解相关的过程，在相关的代码上进行修改即可。换一种角度，只有当你在理解原理的基础上，你才能看懂代码，并且正确地修改达到自己的目的。

2. 注意全角和半角字符的使用。在实验过程中曾碰到过输入字符串语句编译出错的情况，但是从语法上看无论如何都是正确的。后来直接运行此行代码才发现全角和半角的引号混用，程序无法正确识别。因而出现在编译报错的情况。其实这从另一个角度提醒我们当自身觉得语法没有出错的时候，应该debug的方向。

3. 字符在计算机中是以ASCII码储存的。这是很早就知道的道理，但在日常的编码中总是会忘记。在实验过程中，有出现过将源MAC地址设为01:02:03:04:05:06而输出的则是31:32:34:35:36的情况。原因是因为直接设置字符串本质上是设置ASCII码，如果要真正设置为想要的数字，需要在字符前加0x表示二进制数。

4. 理解一个相对复杂的过程或代码，应先简后繁。意思是，先从大概的框架入手，整个过程大致分哪几个步骤，需要用到哪几个模块。然而尝试弄清楚每个步骤需要用到的函数，再尝试弄明白每个函数的具体实现。如此一来，整个过程会很清晰，理解起来更容易。写代码也是使用这个方法。

## 参考文献

- [1] <http://blog.csdn.net/httww/article/details/7521053>
- [2] [http://www.tcpdump.org/pcap3\\_man.html](http://www.tcpdump.org/pcap3_man.html)
- [3] <http://www.tcpdump.org/pcap.htm>
- [4] [http://www.cnblogs.com/yc\\_sunniwell/archive/2010/07/05/1771563.html](http://www.cnblogs.com/yc_sunniwell/archive/2010/07/05/1771563.html)
- [5] <http://www.tcpdump.org/manpages/pcap-filter.7.html>
- [6] <http://www.pdbuchan.com/rawsock/rawsock.html>