**信息安全实验报告**

**Lab 13 Packet Sniffing and Spoofing Lab**

**孙铁**

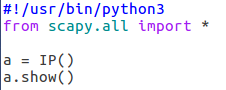
**SA20225414**

**Task 1**

安装Scapy：

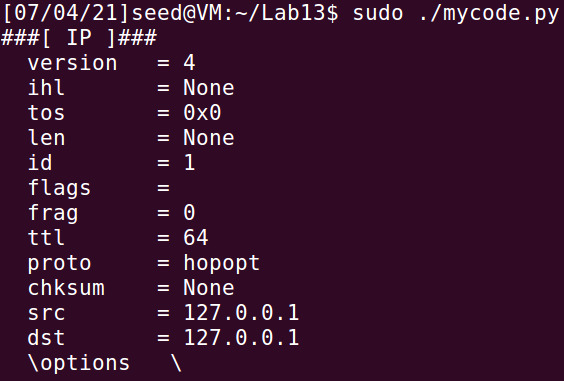


创建文件mycode.py：



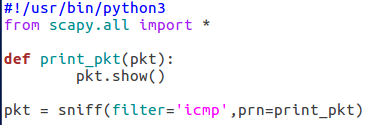
代码定义了一个IP报文，并用show函数输出了报文相关信息。

运行mycode.py：



**Task 1.1**

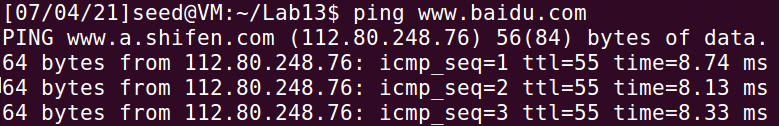
创建文件sniffer.py：



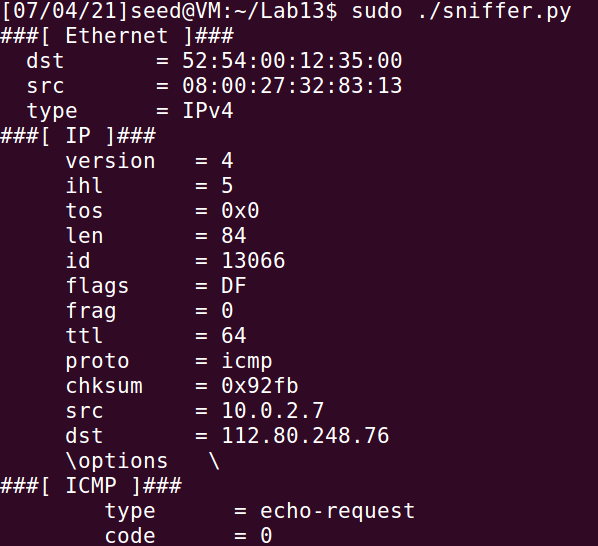
代码的作用是嗅探ICMP报文并打印出一些报文的信息。

**Task 1.1A**

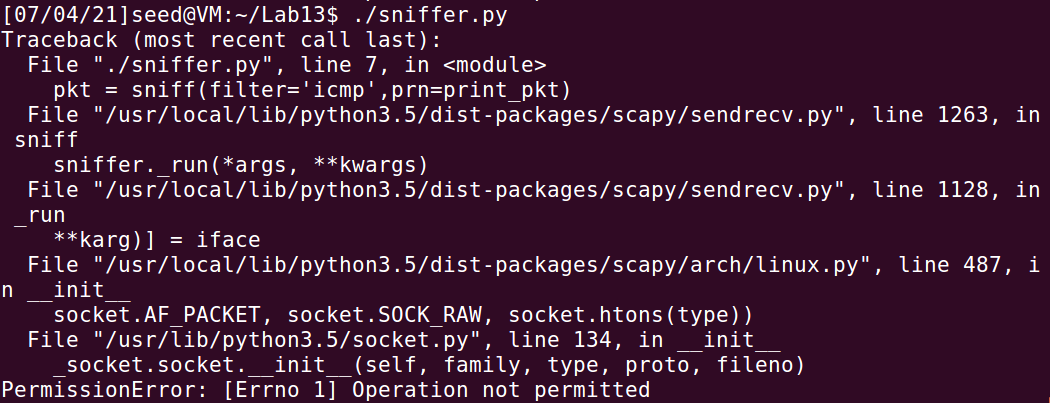
为了产生ICMP报文，向www.baidu.com使用PING指令：



在root权限下运行sniffer.py，捕获到的ICMP报文:



在普通用户权限下运行sniffer.py：



返回错误信息，提示权限不足以支持socket.\_\_init\_\_函数的调用。

**Task 1.1B**

1. 只捕获ICMP报文：

将sniffer.py文件中sniff函数的filter参数设为‘icmp’即可，如Task 1.1A所示。

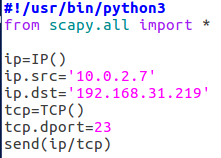
1. 捕获特定IP地址（本机IP地址10.0.2.7）23端口发出的TCP报文：

修改sniffer.py文件如下：

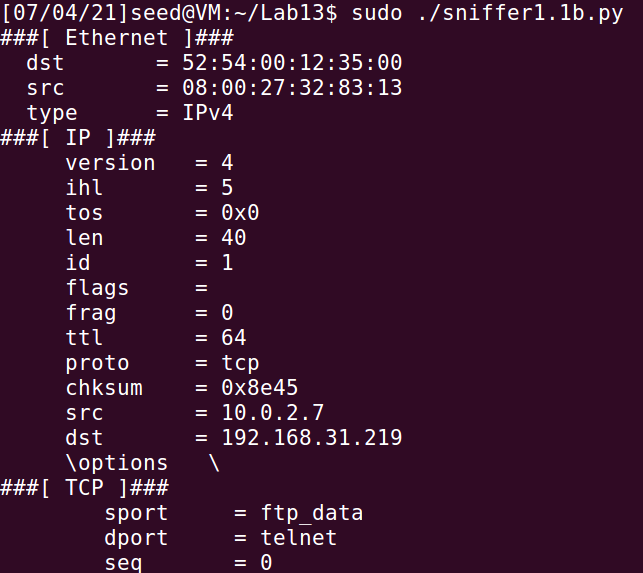


运行sniffer.py文件。

创建TCP报文发送脚本：



运行脚本之后sniffer.py捕获到指定的TCP报文：



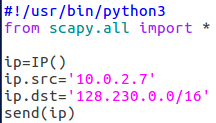
1. 捕获发往子网128.230.0.0/16的报文：

修改sniffer.py文件如下：

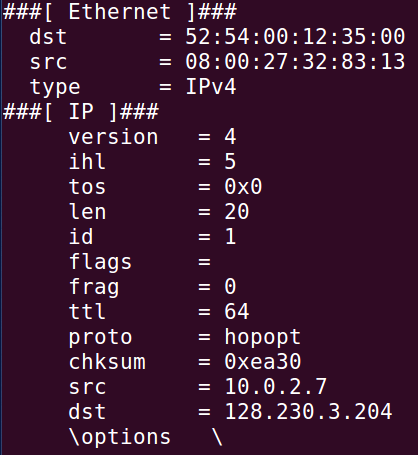


运行sniffer.py文件。

创建报文发送脚本：



运行脚本之后sniffer.py捕获到发往子网128.230.0.0/16报文：

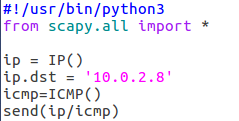


**Task 1.2**

在这个Task需要用到两个虚拟机：

* VMA：10.0.2.7；
* VMB：10.0.2.8；

在VMA创建报文发送脚本：



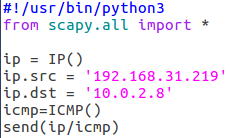
运行脚本，从VMA向VMB发送一个ICMP报文。

在VMB上用WireShark捕获到ICMP报文：



可以看到VMB收到了VMA的ICMP报文并返回了一条ICMP应答报文

修改VMA上的报文发送脚本，添加一任意IP地址作为伪造源地址（以主机地址192.168.31.219为例）。：



运行脚本发送报文。

在VMB上用WireShark捕获到ICMP报文：



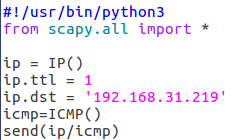
可以看到VMB捕获到的ICMP报文源地址为伪造地址。

此时VMA上用WireShark也成功捕获到VMB返回的ICMP应答报文：



**Task 1.3**

在VMA（10.0.2.7）创建报文发送脚本：



运行脚本发送一个TTL=1的ICMP报文。

在VMA上用WireShark捕获到ICMP报文：



报文在发送过程中TTL变为0，被路由器丢弃。

将脚本中ttl修改为2，再次运行报文发送脚本，在VMA上用WireShark捕获到ICMP报文：



成功收到192.168.31.219返回的ICMP应答报文。

由此可以得到VMA与192.168.31.219之间的路由数量为2。

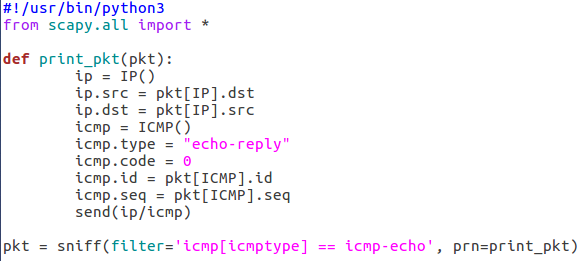
**Task 1.4**

在这个Task需要用到两个虚拟机：

- VMA：10.0.2.7；

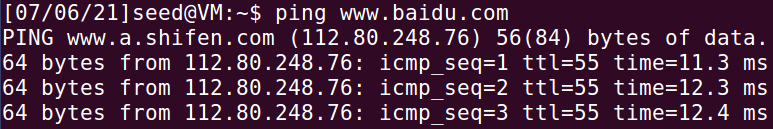
- VMB：10.0.2.8；

在VMA（10.0.2.7）创建文件sniffer.py：

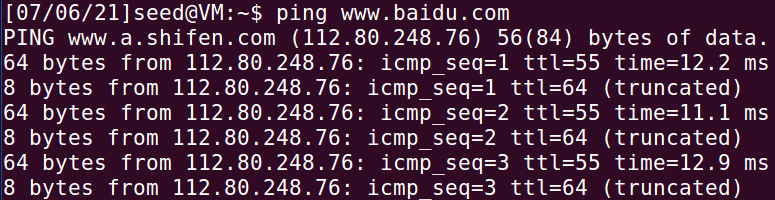


代码作用是嗅探ICMP回显请求报文，如果成功捕获回显请求报文，则调用print\_pkt函数伪造并发送一个ICMP回显应答报文。

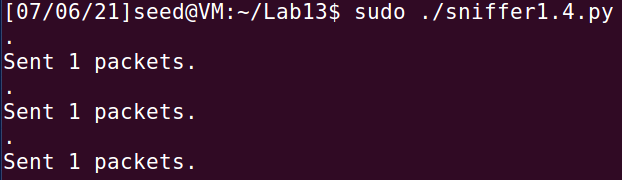
正常在VMB（10.0.2.8）向www.baidu.com使用PING指令：



运行sniffer.py，在VMB向www.baidu.com使用PING指令：



此时VMA上sniffer.py程序会不断发出伪造的回显应答报文：

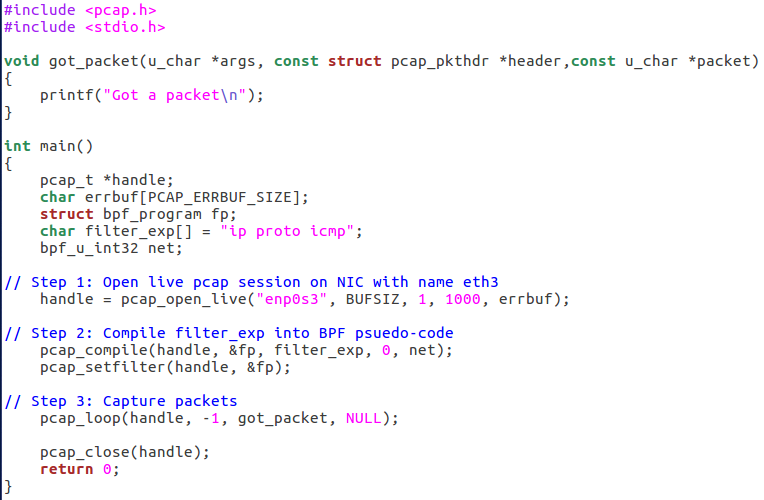


与正常的PING结果相比较可以发现sniffer.py运行之后VMB会收到两次ICMP回显应答，其中一次会被截断（truncated）。

**Task 2**

**Task 2.1**

在VMA（10.0.2.7）中创建文件sniffer2.1.c：

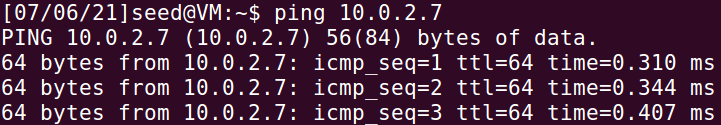


代码的作用是嗅探所有收到的ICMP报文，如果成功获取报文，则会输出“Got a packet”。

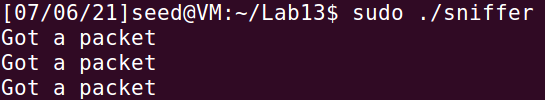
编译生成可执行文件sniffer，由于代码使用了pcap库，需要在编译命令后面增加 -lcap参数：



运行sniffer，并在VMB（10.0.2.8）向VMA（10.0.2.7）使用PING指令：

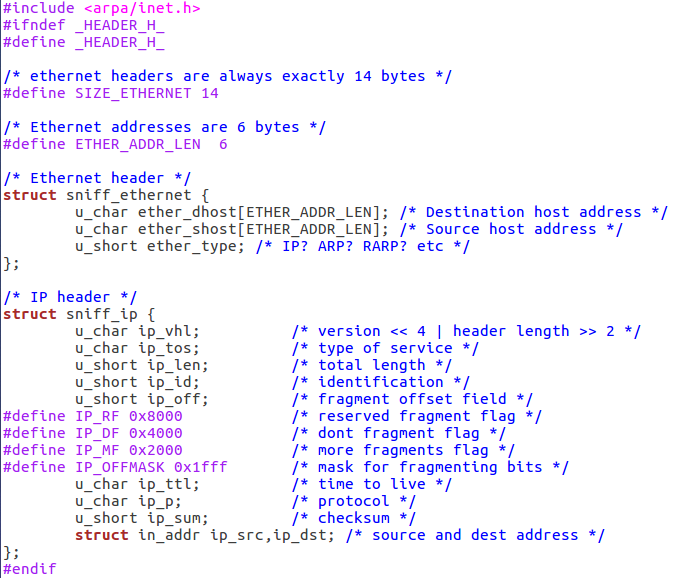


此时VMA上的sniffer成功嗅探到了ICMP报文：



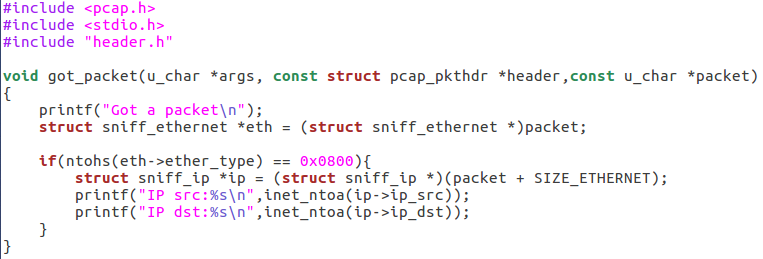
**Task 2.1A**

创建文件header.h：



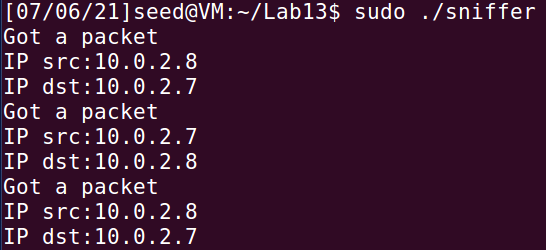
代码内容为IP和Ethernet数据包的头部结构体的定义。

为了输出嗅探到的报文的源地址和目的地址，对sniffer2.1.c进行修改：



将header.h引入sniffer2.1.c，然后在got\_packet函数中增加打印IP数据包的源地址和目的地址的语句。

重新编译sniffer2.1.c，运行sniffer并在VMB（10.0.2.8）向VMA（10.0.2.7）使用PING指令：



VMA（10.0.2.7）上sniffer成功打印出嗅探到的ICMP报文的源地址和目的地址。

**Question 1**

嗅探时依次调用了5种函数：

1. pcap\_open\_live函数打开指定网卡设备准备嗅探；
2. pcap\_compile函数将过滤规则编译为BPF格式；
3. pcap\_setfilter函数将BPF格式的过滤规则传递给内核；
4. pcap\_loop函数进入循环，开始捕获数据包，捕获到即会调用设置好的回调函数，对数据包进行处理；
5. pcap\_close函数释放网卡，停止嗅探。

**Question 2**

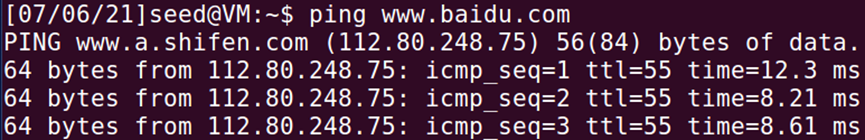
使用非root权限运行sniffer时会直接返回段错误：



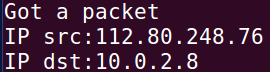
主要原因是因为访问网卡设备时权限不足。

**Question 3**

在VMA（10.0.2.7）上运行sniffer运行，并从VMB向www.baidu.com使用PING指令：



VMA上的sniffer成功嗅探到数据包：



将sniffer2.1.c中pcap\_open\_live函数的第三个参数从1改为0来关闭混杂模式：



重新编译，在VMA上运行sniffer，并从VMB向www.baidu.com使用PING指令：



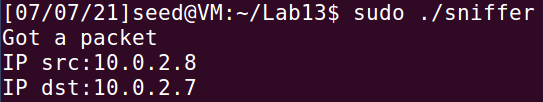
和混杂模式开启时不同，此时VMA上的sniffer不会嗅探到任何数据包。由此可以看出当混杂模式关闭时，VMA只能嗅探到发送给自己的数据包：

**Task 2.1B**

* 嗅探两个特定主机之间的ICMP数据包：

修改文件sniffer2.1.c中的filter\_exp字符串：



编译运行sniffer，从VMB向VMA或是从VMA向VMB使用PING指令：

成功嗅探到数据包。

从VMA或是VMB向www.baidu.com使用PING指令：



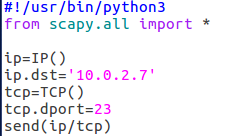
无法嗅探到数据包。

* 嗅探目的端口号为10-100的TCP数据包：

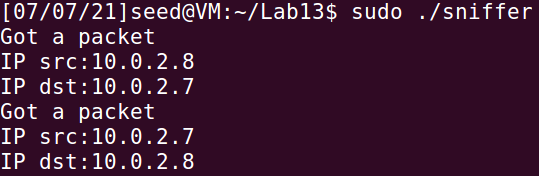
修改文件sniffer2.1.c中的filter\_exp字符串：



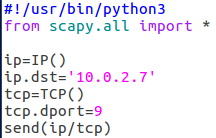
编译运行sniffer，在VMB运行如下报文发送脚本向VMA发送端口为23的TCP报文：



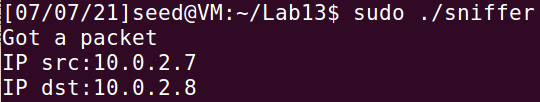
VMA上的sniffer成功嗅探到VMB发来的TCP数据包：



修改VMB上的报文发送脚本，将端口号改为9：



VMA上的sniffer成功没有嗅探到从VMB上发来的TCP数据包，而是首先嗅探到了VMA返回给VMB的TCP数据包：



**Task 2.1C**

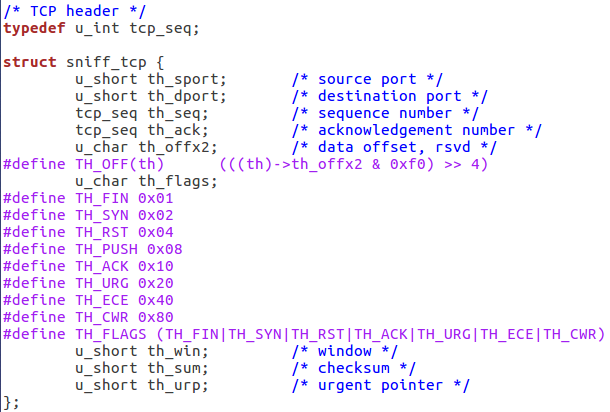
在这个Task需要用到三个虚拟机：

- VMA：10.0.2.7；

- VMB：10.0.2.8；

- VMC：10.0.2.9；

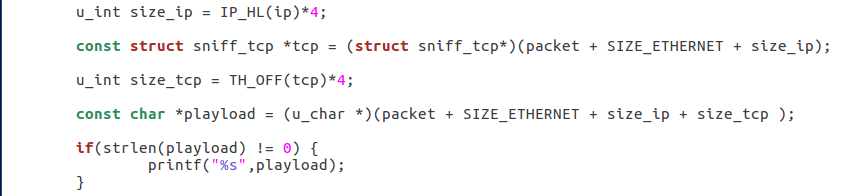
在header.h中加入TCP数据包的头部结构体的定义：



因为telnet使用TCP协议且固定端口号为23，所以为了针对地嗅探telnet的数据包，修改文件sniffer2.1.c中 filter\_exp字符串：

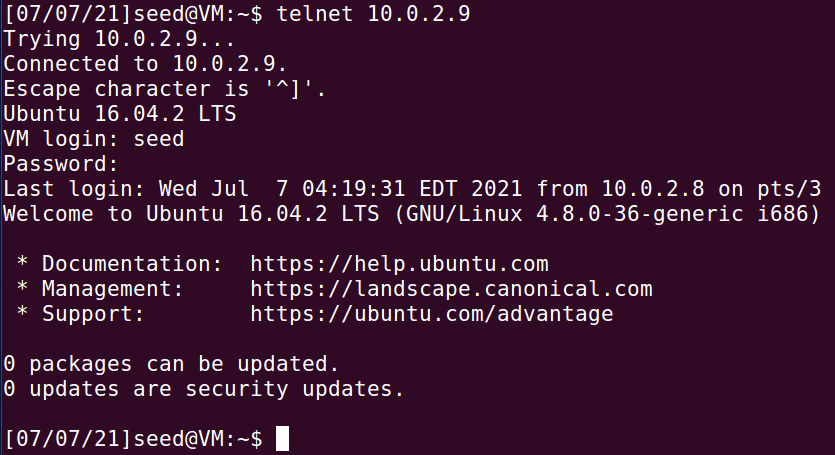


修改got\_packet函数：

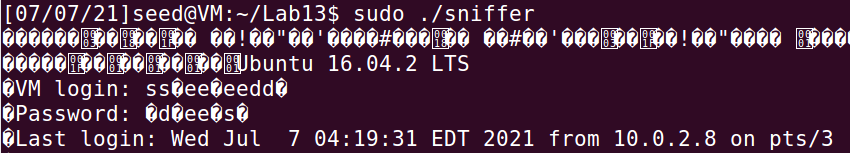


代码的作用是将数据包层层分解，找到TCP报文的位置并输出其数据部分。

编译运行sniffer，在VMB（10.0.2.8）向VMC（10.0.2.9）使用telnet指令：



此时VMA（10.0.2.7）上的sniffer成功嗅探到telnet发出的TCP报文：

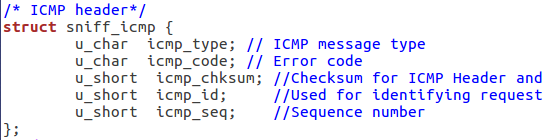


由此得到密码为dees。

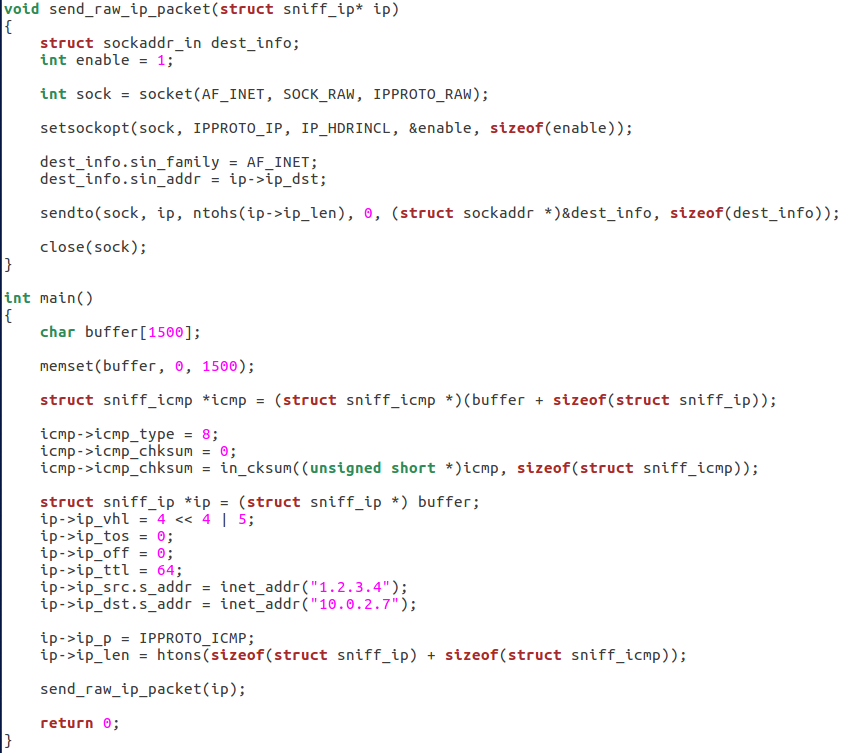
**Task 2.2**

Task 2.2A要求编写一个发送伪造报文的程序而Task 2.2B则要求伪造一个ICMP请求报文，为了节省篇幅，我选择将两者结合，直接伪造一个ICMP请求报文并发送。

在header.h中加入ICMP数据包的头部结构体的定义：



创建文件sniffer2.2.c：



代码的作用是伪造一个ICMP报文并使用raw socket发送。

要伪造数据包首先要创建一个缓冲区，然后依次填入ICMP头部信息和IP头部信息，其中ICMP请求报文头部只需要设置类型和校验和字段（校验和字段通过in\_cksum函数生成）；完成数据包的伪造之后，需要将缓冲区的指针传递给send\_raw\_ip\_packet函数，send\_raw\_ip\_packet函数的作用是创建并设置一个raw socket然后以此将数据包发送出去。

编译并运行sniffer2， WireShark成功捕获到ICMP请求和响应报文：



将目的地址修改为百度的IP地址112.80.248.75，重新编译运行sniffer2：



成功发出ICMP请求报文并收到了回应报文。

**Question 4**

1. 修改sniffer2.1.c中的IP数据包长度：



编译并运行sniffer2，WireShark没有捕获到任何报文。

1. 再次修改sniffer2.1.c中的IP数据包长度：



编译并运行sniffer2，成功捕获到ICMP报文，此时报文长度为116。



1. 再次修改sniffer2.1.c中的IP数据包长度：



编译并允许sniffer2，WireShark没有捕获到任何报文。

由上面实验可知，IP数据包长度不能过大或者是过小，在合法的IP数据包长度内，可以任意设置IP数据包长度。

**Question 5**

使用raw socket不需要计算IP报文的校验和，因为在数据包的发送过程中操作系统会设置该字段。

**Question 6**

只有root权限的进程才可以创建raw socket。使用非root权限运行程序会在调用套接字初始化函数socket函数时出错。

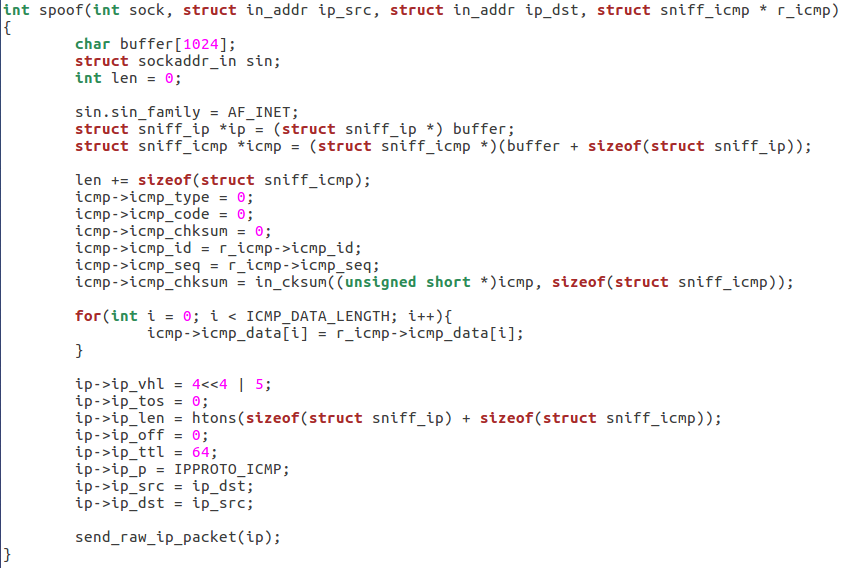
**Task 2.3**

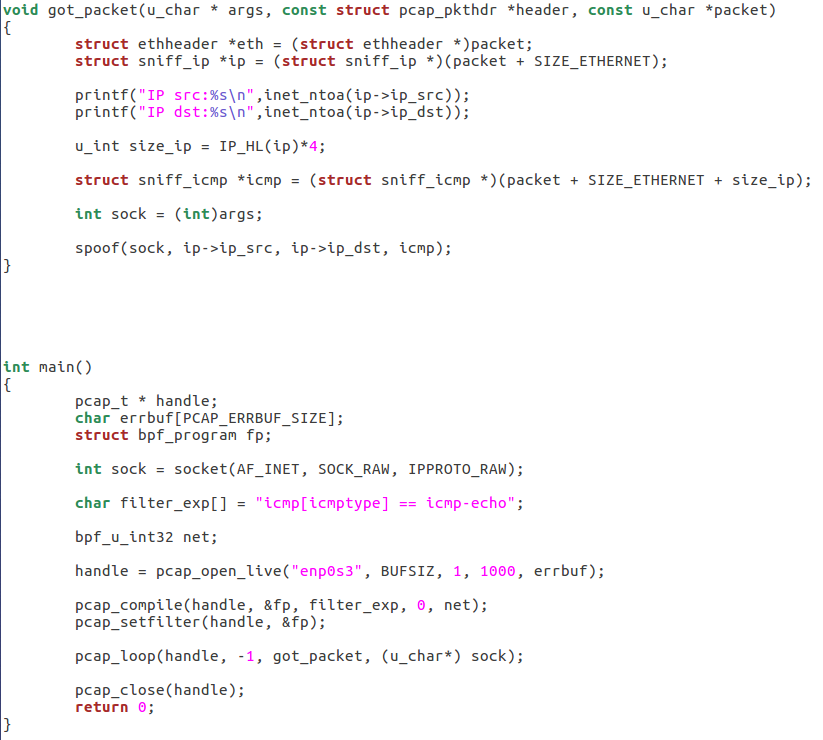
在这个Task需要用到两个虚拟机：

- VMA：10.0.2.7；

- VMB：10.0.2.8；

在VMA（10.0.2.7）创建文件sniffer2.3.c，核心代码如下：





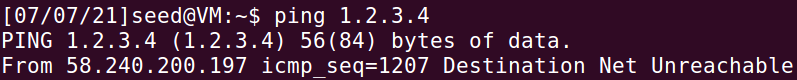
代码的作用是实现了报文的嗅探与伪造。首先不断尝试嗅探ICMP回显请求报文，如果成功则调用send\_raw\_ip\_packet函数以及spoof函数伪造并发送一个ICMP回显应答报文。

sniffer2.3.c整合了sniffer1.4.py、sniffer2.1.c与sniffer2.2.c，主要思路为：

1. 将伪造ICMP数据包的部分提取出来抽象整合为spoof函数；
2. 依旧使用got\_packet函数作为回调函数，输出嗅探到的数据包的源地址与目的地址，然后调用spoof函数；
3. 在proof函数中进行数据包的伪造，然后调用send\_raw\_ip\_packet函数；
4. 依旧使用send\_raw\_ip\_packet函数来进行raw socket的创建以及伪造报文的发送；
5. 依旧使用in\_cksum函数来得到ICMP报文的检验和字段；
6. 修改文件sniffer2.1.c中main函数中的filter\_exp字符串，从而实现针对ICMP请求报文进行嗅探：

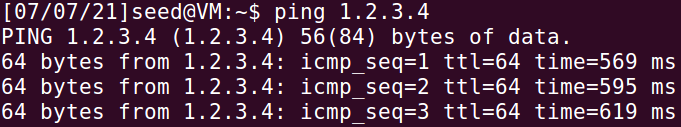


正常在VMB（10.0.2.8）向1.2.3.4使用PING指令：

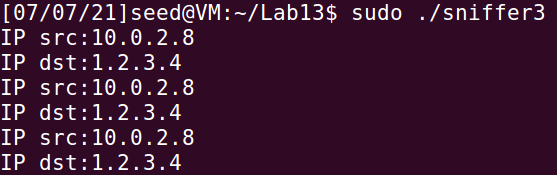


不会有任何数据包回复。

编译并运行sniffer3，在VMB向1.2.3.4使用PING指令：



此时VMA上sniffer.py程序嗅探到VMB发出的ICMP回显请求报文，打印出报文的源地址和目的地址：



与正常的PING结果相比较，可以发现sniffer3运行之后VMB会收到VMA伪造的ICMP回显应答。

VMB的WireShark也能够证明VMA成功嗅探并伪造了ICMP回显应答报文：

