**Lab3-report**

**Packet Sniffing and Spoofing Lab**

**Name:范心宇**

**Student Number:57117129**

**Lab Task Set 1: Using Tools to Sniff and Spoof Packets**

**Task 1.1: Sniffing Packets**

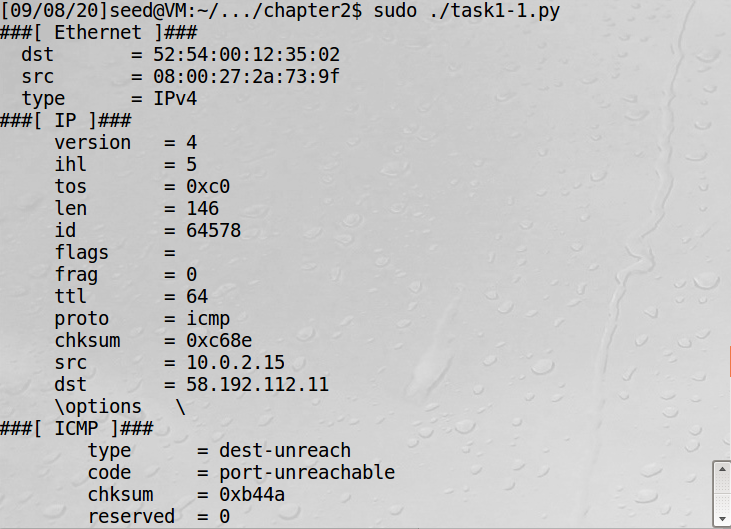
* **实验目的**

嗅探局域网上数据包。

* **实验过程**

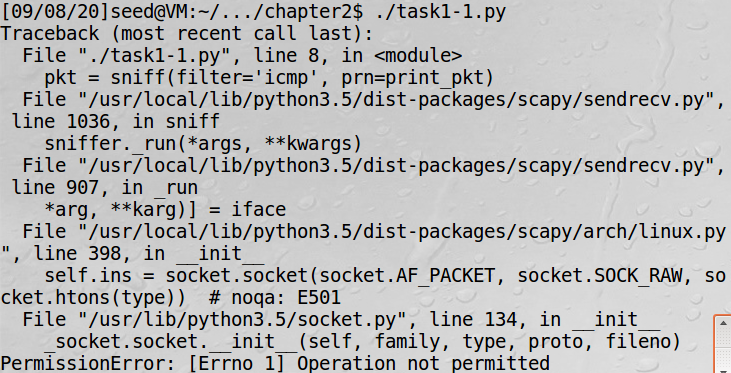
1. 嗅探数据包。

编写Python程序并在root权限下运行：



可以看到抓取到了数据包。

在普通用户权限下运行：

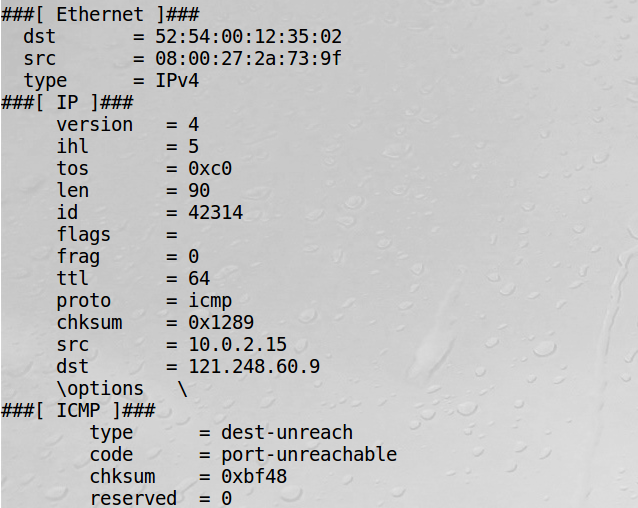


可以看到权限不够。

1. 抓取按要求过滤后的包。

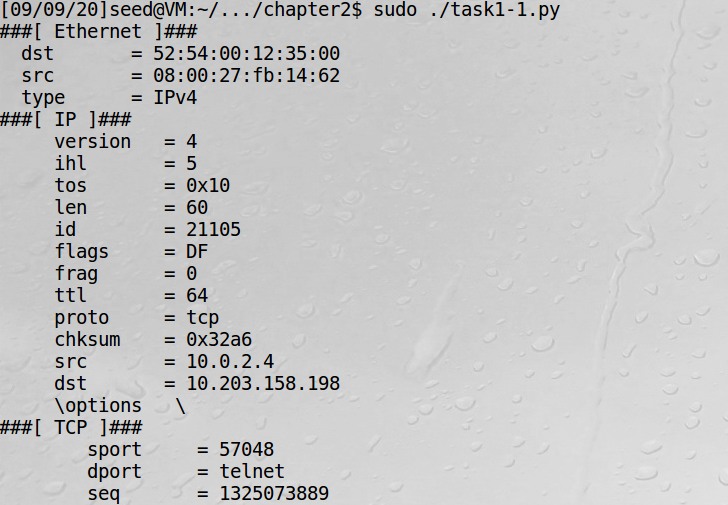
仅抓取ICMP报文：





抓取所有来自特定IP“10.0.2.4”且发往23号端口的TCP报文：

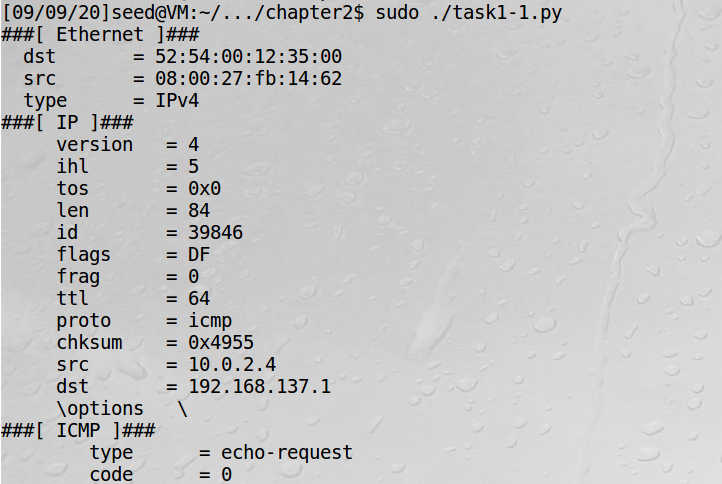




可以看到抓获的数据包源IP为10.0.2.4，目的端口为telnet。

抓取所有来自或发往特定子网“192.168.137.0/24”的数据包：





可以看到抓获了目的IP地址为192.168.137.1的数据包。

* **实验结论**

在Scapy工具中可以使用sniff()函数抓取网络上特定的数据包。

**Task 1.2: Spoofing ICMP Packets**

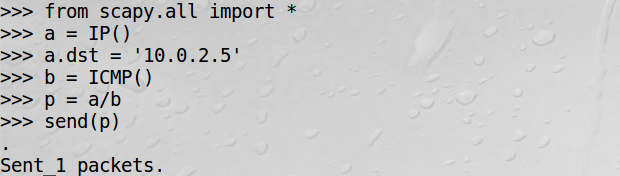
* **实验目的**

伪造ICMP数据报。

* **实验过程**

1. 伪造一个具有虚拟源IP地址的ICMP报文。

利用scapy伪造报文：



查看报文信息：



可以看到当前源IP地址为10.0.2.4，目的IP地址为10.0.2.5。

1. 尝试修改源IP地址。

我们需要伪造一个源IP地址：



查看此时的数据包信息：



此时IP源地址已经变成了我们修改后的地址。

1. 发送该数据包，在另一台虚拟机上通过wireshark观察。









可以看到wireshark抓到的ICMP数据包源IP为10.2.2.2，于是该虚拟主机返回的ICMP回送响应数据包，其目的IP地址即为10.2.2.2。

* **实验结论**

在Scapy中可以很方便地创建各种协议报文并设定其参数，从而方便我们伪造数据包。

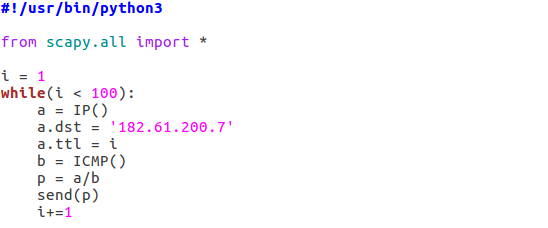
**Task 1.3: Traceroute**

* **实验目的**

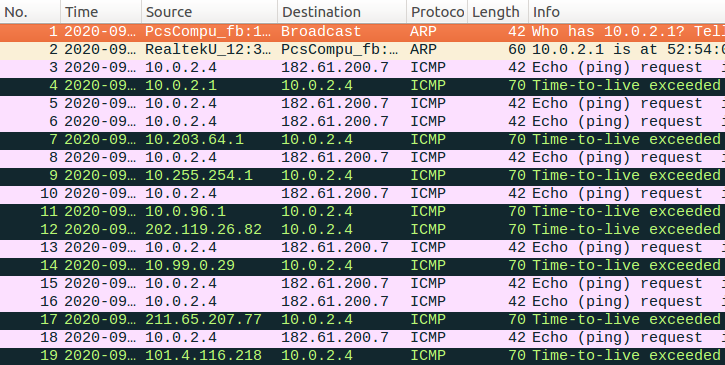
利用Scapy创建ICMP报文实现traceroute跟踪路由的功能。

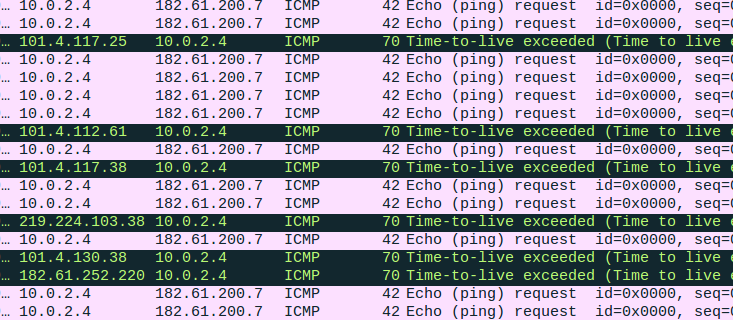
* **实验过程**

1. 编写构造ttl逐渐增加的ICMP报文的程序。



1. 运行该程序，并在wireshark上进行观察。





可以看到一路上返回的ICMP时间超过报文，根据这些报文可以找出途径的路由器的IP地址，如图所示为：10.0.2.1 - 10.203.64.1 - 10.255.254.1- 10.0.96.1 - 202.119.26.82 - 10.99.0.29 - 211.65.207.77 - 101.4.116.218 - 101.4.117.25 - 101.4.112.61 - 101.4.117.38 - 219.224.103.38 - 101.4.130.38 - 182.61.252.220，最终到达182.61.200.7。

需要注意的是由于IP层实现的是无连接的分组转发，实际的路由路径并不一定是这样的，并且由于每次发送报文的网络延时可能不同，收到ICMP时间超过报文的时间也可能不同，所以该路由路径是乱序的。

* **实验结论**

利用Scapy创建ICMP报文，改变其TTL的值，可以收到路径上路由器返回的ICMP时间超过报文，从而获得路由信息，这是一种粗略地估测路由路径的方法。

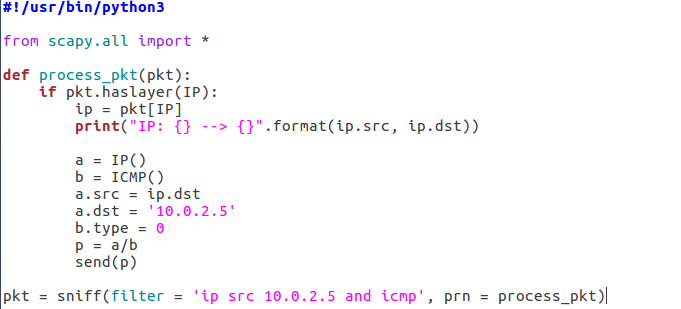
**Task 1.4: Sniffing and-then Spoofing**

* **实验目的**

利用Scapy进行报文嗅探与伪造，在虚拟机B上监听网络报文，一旦发现虚拟机A发送了ICMP回送请求报文，即伪造ICMP回送应答报文，作为对A的响应，使A始终能够收到响应报文。

* **实验过程**

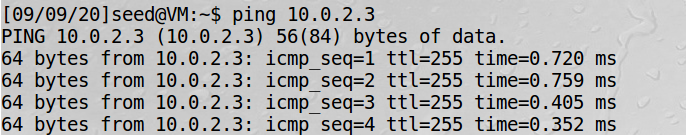
1. 在虚拟机B上编写嗅探与伪造程序的代码。



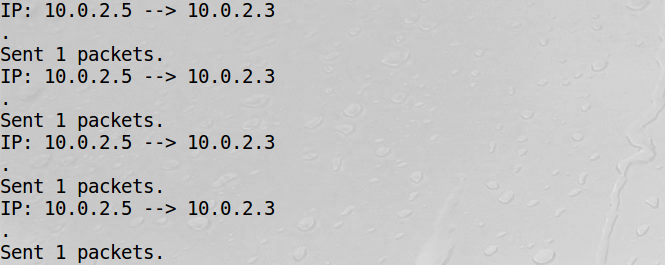
设置响应报文的IP目的地址为虚拟主机A的IP地址“10.0.2.5”，IP源地址为虚拟主机A发出的ICMP报文的目的地址，再将ICMP报文类型字段设为0，表示ICMP回送响应报文，过滤器设为“抓取来自源IP 10.0.2.5的ICMP报文”。

1. 运行虚拟机B上的程序，打开虚拟机A上的wireshark观察。

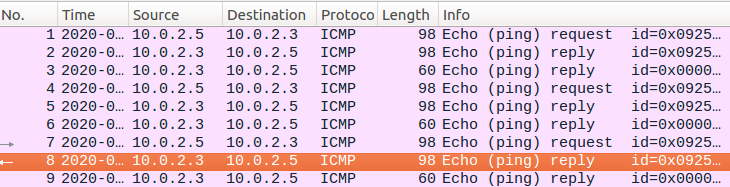
在虚拟机A的命令行输入ping命令，随意ping一个IP地址：



虚拟机B上显示：

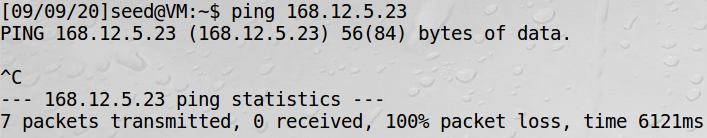


说明ICMP回送应答报文已发送，再观察虚拟机A的wireshark：

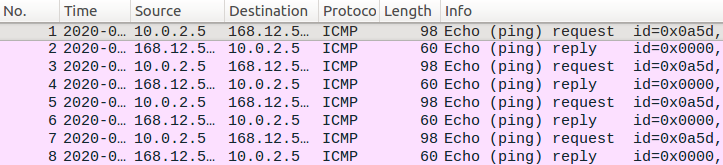


可以看到每个请求能得到两个响应，我们再ping其他IP地址观察结果。

关闭虚拟机B上运行的程序，在虚拟机A上ping“168.12.5.23”：



可以看到没有响应，打开虚拟机B上程序后再ping该地址：



可以看到wireshark中收到了应答报文。

* **实验结论**

使用Scapy能够方便地伪造并发送数据包，对网络上的其他主机发起攻击。

**Lab Task Set 2: Writing Programs to Sniff and Spoof Packets**

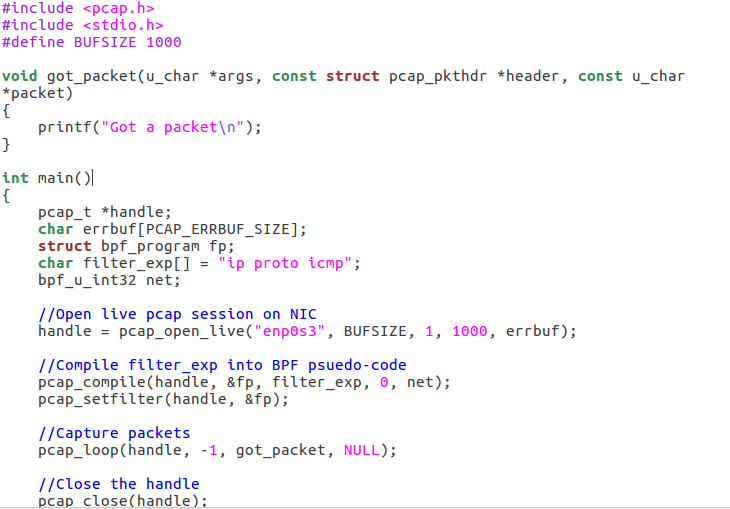
**Task 2.1: Writing Packet Sniffing Program**

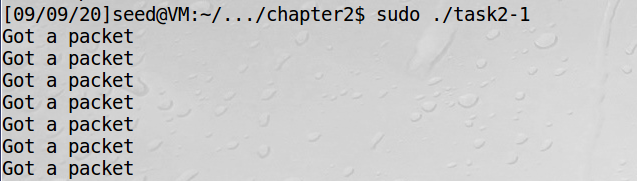
* **实验目的**

使用pcap库写一个数据包嗅探程序。

* **实验过程**

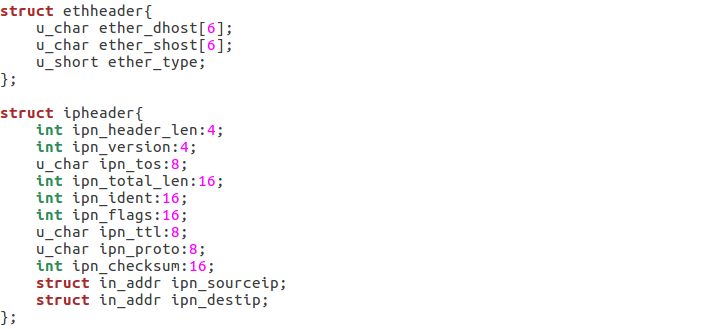
1. 编写给出的c程序并尝试运行。

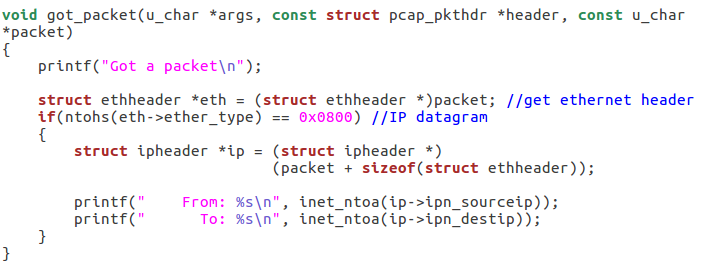




1. 添加代码打印IP源、目的地址。









可以看到能够成功分析出捕获的数据报的源、目的IP地址。

1. 回答问题。

3.1描述sniffer程序中重要函数的作用与抓包的流程。

答：首先pcap\_open\_live()函数打开给定的网卡端口，返回一个设备句柄，本主机是enp0s3，设置每次读取的报文的最大长度、返回函数等待时间，以及最重要的第三个参数设为“1”表示使用混杂模式，打开后即开始监听端口上报文流量；然后使用pcap\_compile()函数编译自定义过滤规则，其中第三个参数很重要，用于设定自己的过滤规则，其遵循BPF语法格式，该函数会将可读的规则编译成BPF底层二进制码；pcap\_setfilter()函数则运用过滤规则，将其添加到内核过滤程序中；再调用pcap\_loop()循环抓包函数开始抓包，其中最重要的是第三个参数，它传递一个回调函数，在回调函数里我们可以实现想要的分析与输出功能；最后通过pcap\_close()函数关闭打开的设备句柄。

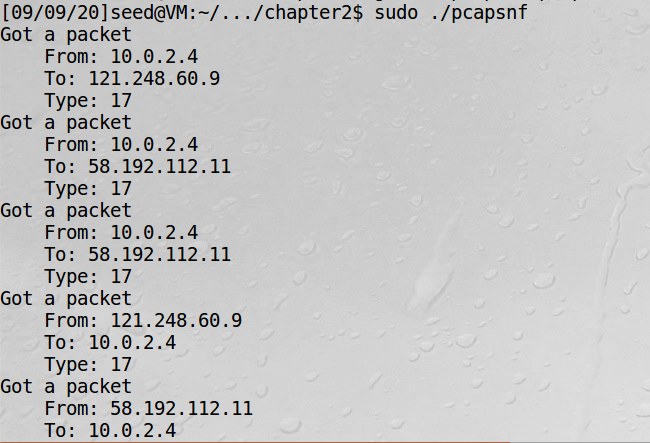
3.2为什么需要root权限执行sniffer程序？如果不使用root，哪一步不能完成？

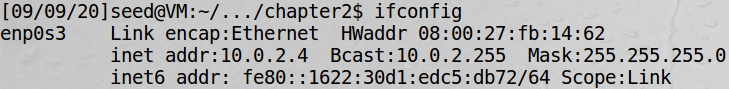
答：因为只有root权限才能通知操作系统将NIC设置为混杂模式，如果不使用root权限，那么pcap\_open\_live()函数在打开网络设备接口时无法启动混杂模式，只能监听到自己发送或接收的报文。

3.3改变混杂模式设置进行观察。

关闭混杂模式：



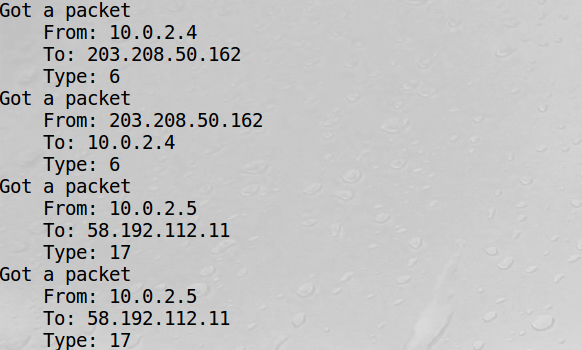




本虚拟主机IPv4地址为10.0.2.4，可以看到关闭混杂模式时，只能监听到本机相关数据包。

打开混杂模式：

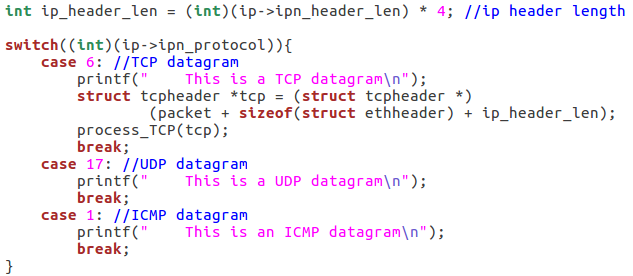




可以看到此时既能收到自身10.0.2.4的数据包，还能接收到源/目的IP为10.0.2.5的数据包。

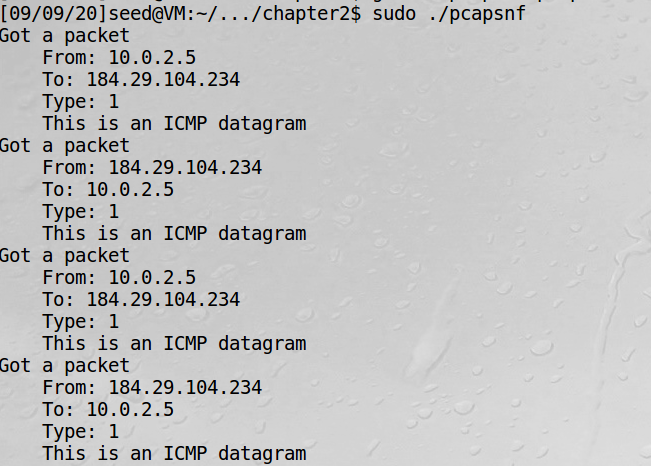
1. 设置过滤器。

在got\_packet()回调函数中增加进一步分析的代码：



抓取两个特定主机“10.0.2.5”和“184.29.104.234”通信的ICMP报文：





可以明显看到只抓取了特定报文。

抓取目的端口在10到100之间的TCP通信报文：





可以看到只抓取了目的端口在10~100间的TCP报文。

1. 嗅探密码。

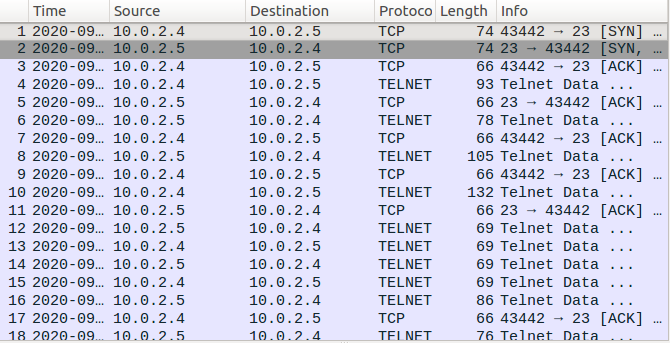
首先捕捉端口号为23的TCP报文：



将捕获的数据包保存到文件中：



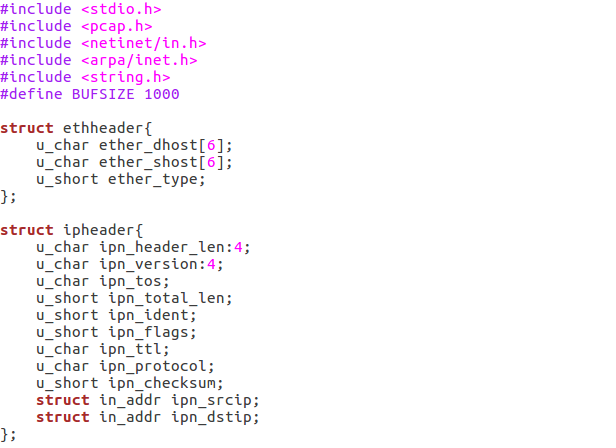
使用wireshark查看该文件：



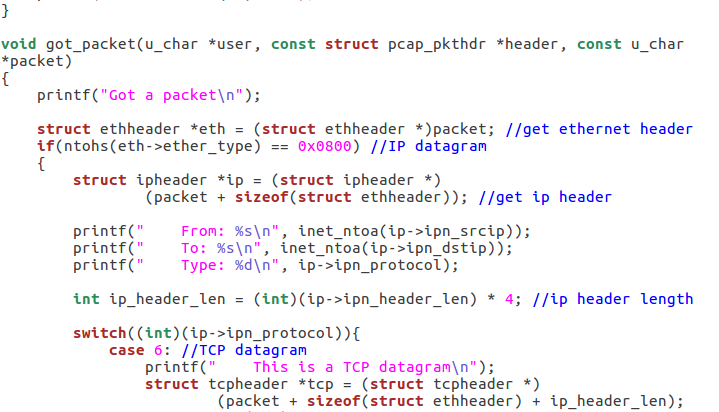
可以看到捕获了两台主机之间的telnet通信，但是以目前的时间和能力还不足以找到密码。。。

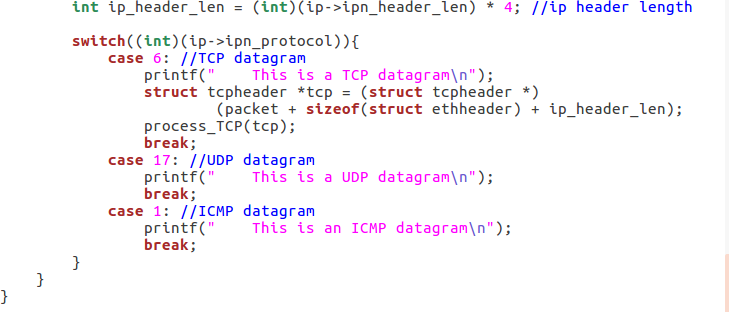
1. 附上完整代码。

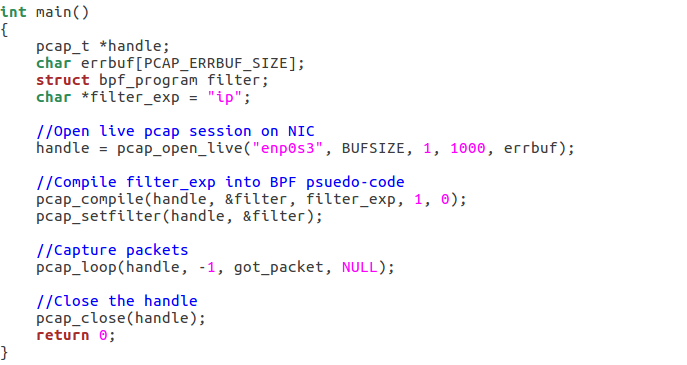
处理TCP及ICMP数据包的代码pcapsnf.c：



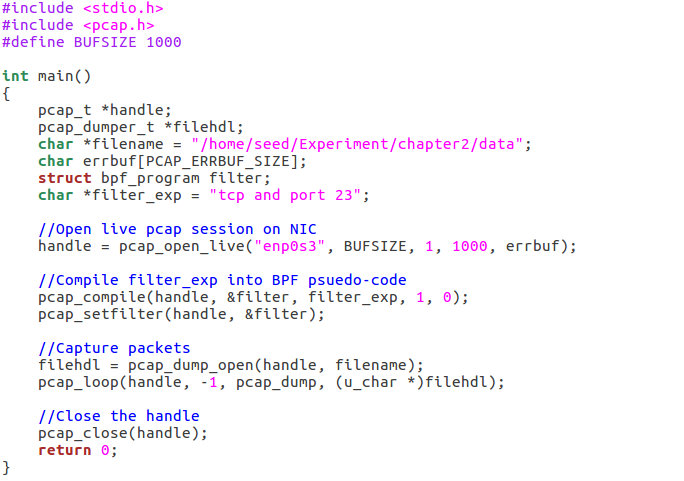








捕获telnet数据包并保存到文件中的代码sniffpwd.c：



* **实验结论**

可以看到，使用libpcap函数可以嗅探想要的数据包，并进行存储、分析等操作，其中重要的步骤有“设置混杂模式”“编译与设置过滤器”等。

**ARP Cache Poisoning Attack Lab**

**Task 1: ARP Cache Poisoning**

* **实验目的**

实施ARP缓存中毒攻击。

* **实验过程**

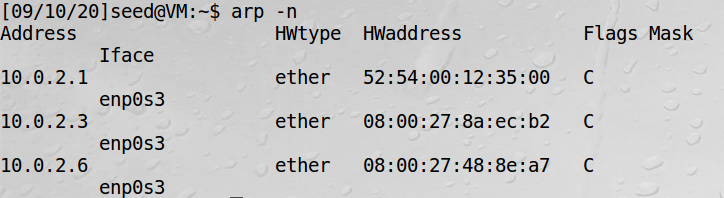
1. 查看基本信息。

虚拟机A的IP地址为10.0.2.5，MAC地址为08:00:27:be:40:8a，

虚拟机B的IP地址为10.0.2.6，MAC地址为08:00:27:48:8e:a7，

虚拟机M的IP地址为10.0.2.4，MAC地址为08:00:27:fb:14:62。

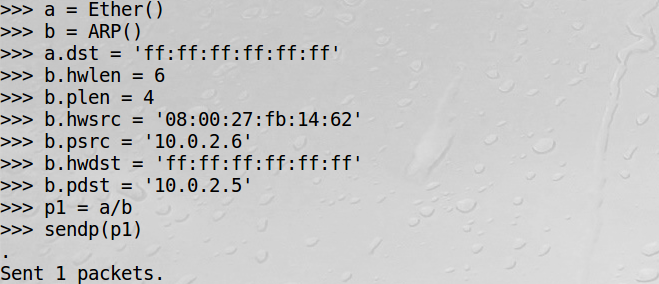
从A上ping B后查看ARP信息：



可以看到此时A的ARP缓存中保存了B的IP地址与MAC地址的映射信息。

1. 在虚拟机M上发送ARP请求报文给虚拟机A。

设置以太网帧头部和ARP报文中的目的MAC地址为广播地址，ARP报文中目的IP为虚拟机A的IP地址，源MAC为虚拟机M的MAC地址，源IP为虚拟机B的IP地址，并发送：



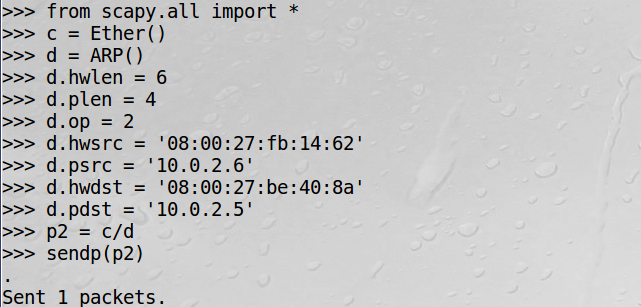
查看虚拟机A的ARP缓存信息：



可以看到虚拟机B的映射表项被修改，其MAC地址变成了M的MAC地址。

1. 在虚拟机M上发送ARP应答报文给虚拟机A。

删除上面建立的映射项目，设置ARP报文中选项字段为2，表示应答报文，目的MAC与IP都为虚拟机A的真实地址，源MAC为虚拟机M的MAC地址，源IP为虚拟机B的IP地址，并发送：



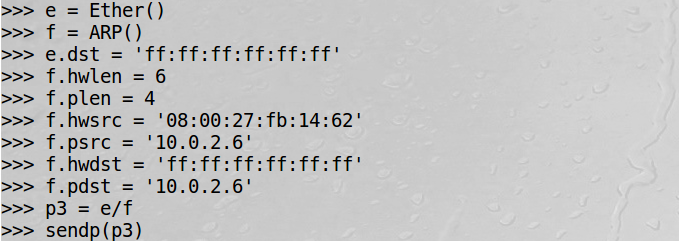
查看虚拟机A的ARP缓存信息：



可以看到虚拟机B的表项被修改，其MAC地址变成了M的MAC地址。

1. 在虚拟机M上发送ARP gratuitous报文给虚拟机A。

删除上面建立的映射项目，设置以太网帧头部和ARP报文中的目的MAC为广播地址，ARP报文中源MAC为虚拟机M的MAC地址，源/目的IP都为虚拟机M的IP地址，并发送：



查看虚拟机A中的ARP缓存信息：



可以看到虚拟机B的表项被修改，其MAC地址变成了M的MAC地址。

* **实验结论**

可以看到，每次收到ARP相关报文时，主机都会检查自身的ARP缓存，如发现有条目与新收到的ARP报文中包含的信息相违背，就会更新ARP缓存信息，这给ARP攻击带来了极大便利。

**Task 2: MITM Attack on Telnet using ARP Cache Poisoning**

* **实验目的**

在Telnet对话中实现中间人攻击。

* **实验过程**

1. 在虚拟机M上实现对虚拟机A和B的ARP缓存中毒攻击。

利用上一个实验的方法，将虚拟机A和B上关于对方的IP地址都映射到虚拟机M的MAC地址上，在虚拟机A上：



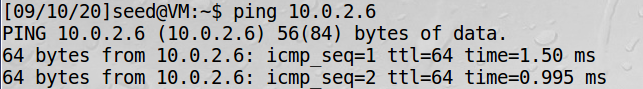
在虚拟机B上：



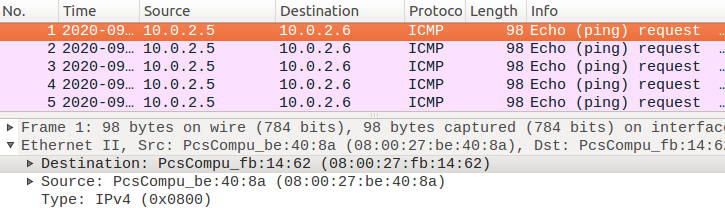
关于对方的IP地址都已映射到M的MAC地址。

1. 虚拟机A与B互相ping对方IP，观察wireshark结果。

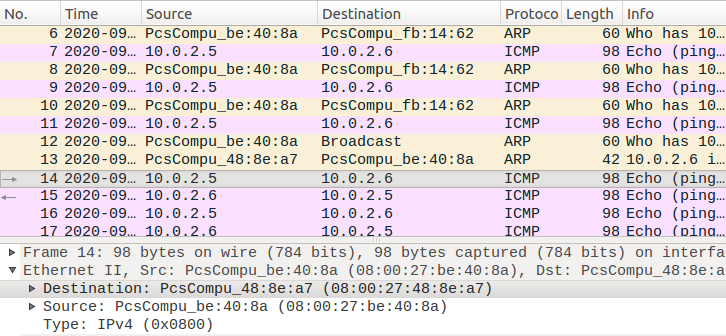
虚拟机A ping虚拟机B：



刚开始时没有响应，过段时间才收到ICMP应答报文。查看wireshark：



最初发送的ICMP请求报文，目的IP地址为10.0.2.6，而其帧首部目的MAC地址为08:00:27:fb:14:62，是虚拟机M的MAC地址，所以一直未能收到ICMP应答报文。再观察后面的报文：



可以看到在多次未收到ICMP应答报文后，虚拟机A发起了ARP查询请求，最初的三次仍向错误的MAC地址请求信息，未收到回应后，全网广播ARP查询请求，获得了虚拟机B的正确的MAC地址，随后再发送ICMP请求报文就得到了响应，这时ARP缓存里已保存了正确的映射信息，ARP缓存中毒攻击到此失效。

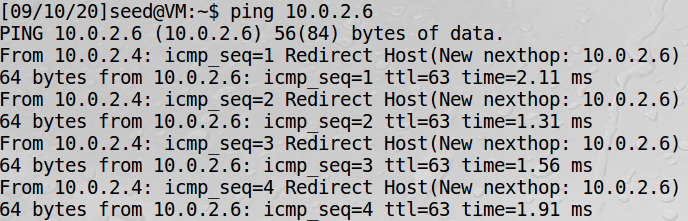
B ping A也是同样的结果。

1. 打开IP forwarding再观察。

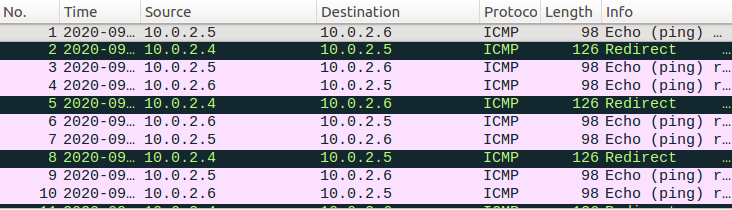
打开虚拟机M上的IP forwarding：



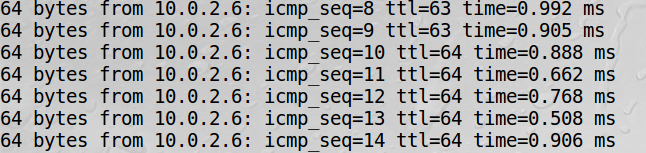
虚拟机A ping虚拟机B：



ping完后看到能够立即收到响应，并且结果显示ICMP应答报文从IP地址为10.0.2.4的主机上重定向转发而来。观察wireshark：

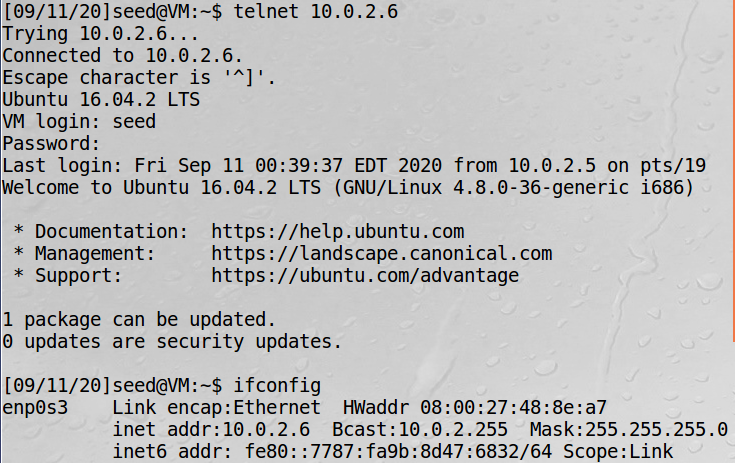


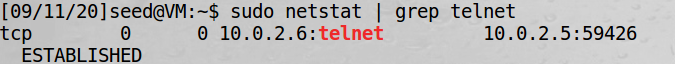
可以明确看到虚拟机M向A发送的ICMP重定向报文。过段时间后，虚拟机A与B恢复正常：



1. 实施中间人攻击。

首先由虚拟机A（10.0.2.5）向虚拟机B（10.0.2.6）发起Telnet请求，登录后测试其效果：





虚拟机A成功查看了虚拟机B的网络端口信息，同时可以看到虚拟机B上显示成功建立了telnet连接。

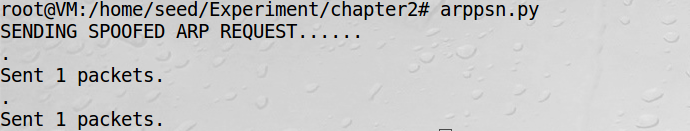
查看此时虚拟机A和虚拟机B的ARP缓存信息：





上述条目分别为A上关于B的地址映射与B上关于A的地址映射，此时的地址信息还是正确的。

实施ARP缓存中毒攻击（代码附在后面），再次查看：

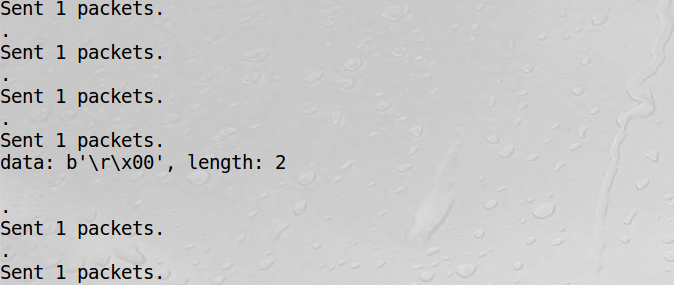






可以看到虚拟机A和B的ARP缓存中关于对方的条目被修改，MAC地址都指向了虚拟主机M。

此时实施MITM中间人攻击（代码附在后面），再使用telnet服务：

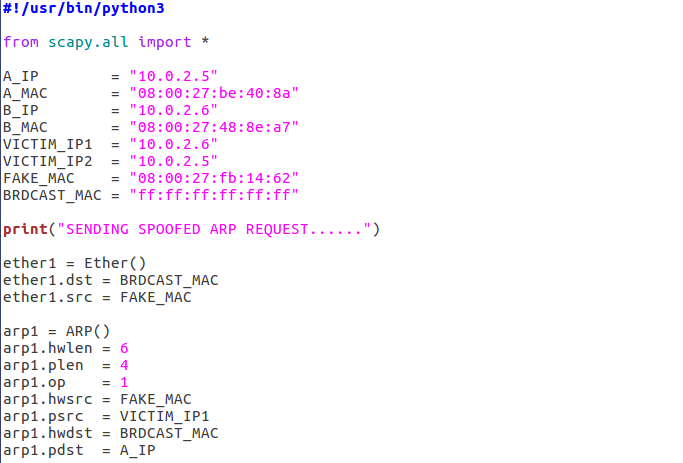


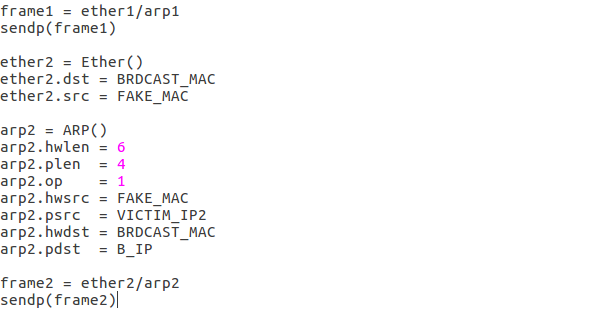
虚拟机M不断发送telnet报文。



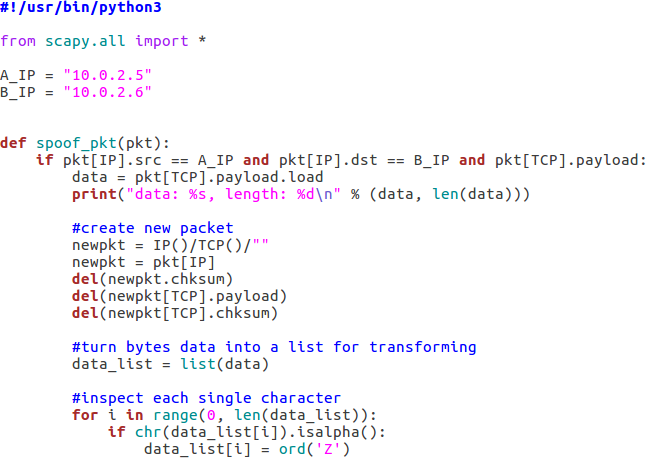
而在虚拟机A上，无论输入什么字符，都会变成“Z”，中间人攻击成功。

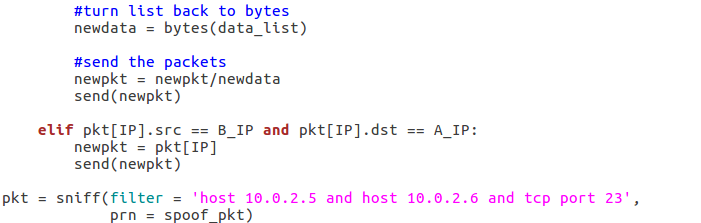
下面附上ARP缓存中毒攻击完整代码arppsn.py：





下面附上MITM中间人攻击完整代码MITM.py：





* **实验结论**

从本实验可以看出ARP协议虽然简单，但是存在很大的安全风险。由于ARP软件在收到ARP报文时，并不会检查其内容是否符合现实，而是直接采用其中信息更新自身的ARP缓存，这就给ARP攻击带来很大便利。当我们连接在局域网上时，嗅探到局域网上其他主机的IP与MAC映射信息，我们可以伪造ARP报文，使用受害者的IP地址和自身的MAC地址构造报文再发送，就可以攻击指定目的主机的ARP缓存，使其保存错误的映射信息，于是所有发往受害者的报文都写着我们的MAC地址，从而由我们来接收，这就是ARP缓存中毒。

进一步地，我们对两个通信的主机都使用这种攻击，使他们发送的报文都到达我们的主机，使用原始套接字或其他嗅探工具（如Scapy）收下这些报文，并伪造数据再发送给目的主机，这样可以使通信双方只能接收我们想要他们收到的数据。但需要注意的是，ARP攻击只能作用在局域网上，因为它本身工作在链路层并不会发送到外网，同时，使用加密数据的方式也可以一定程度地防范中间人攻击。

**IP/ICMP Attacks Lab**

**Tasks 1: IP Fragmentation**

* **实验目的**

使用Scapy构造分片的IP数据包。

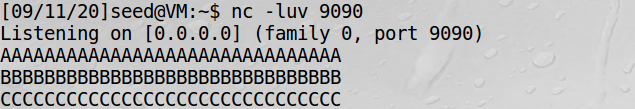
* **实验过程**

1. 进行IP分片。

编写三个分片的信息：



将第一个IP报文片偏移设为0，MF设为1，加上udp头部，设置检验和为0（否则Scapy会计算出一个错误的值），长度为udp报文总长度104，再加上负载。将第二个IP报文片偏移设为5，表示偏移了40个字节，也就是第一个IP报文中数据字段的长度，MF设为1，注意这里还需要设置协议类型字段为17，表示这是一个udp报文的数据（否则Scapy采用默认值0表示IPv6，这样会导致错误），最后加上负载。第三个IP报文与第二个类似，但是片偏移为9，MF设为0，表示这是最后一个分片。最后发送三个分片报文。运行该程序：

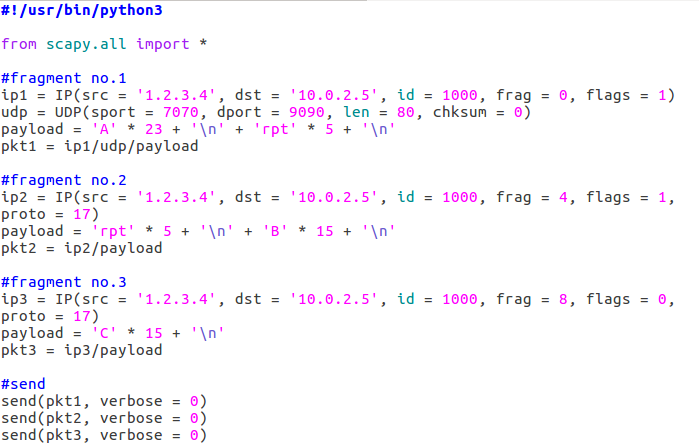


在虚拟机A上打开udp 9090端口监听，成功收到96bytes的数据。

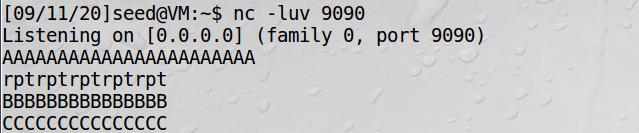
1. 进行有重叠的IP分片。

2.1第二个分片与第一个分片部分重叠

设置第一个IP分片包含8字节UDP头部和40字节数据，第二个IP分片包含32字节数据，其中有16字节重叠数据，第三个IP分片包含16字节数据：



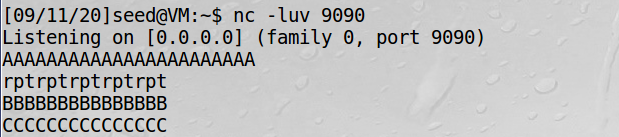
运行代码：



可以看到仍能运行成功，且重叠的字段rpt只打印了一份。

如果设置第二个IP分片与第一个IP分片的重叠部分取值不同，再测试：

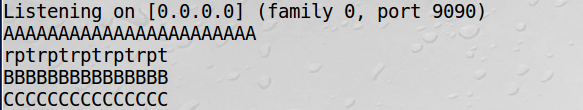




运行成功，且第二个分片的数据覆盖了第一个分片的数据。

下面调整发送分片的顺序，让第一个分片在第二个分片后面发送：



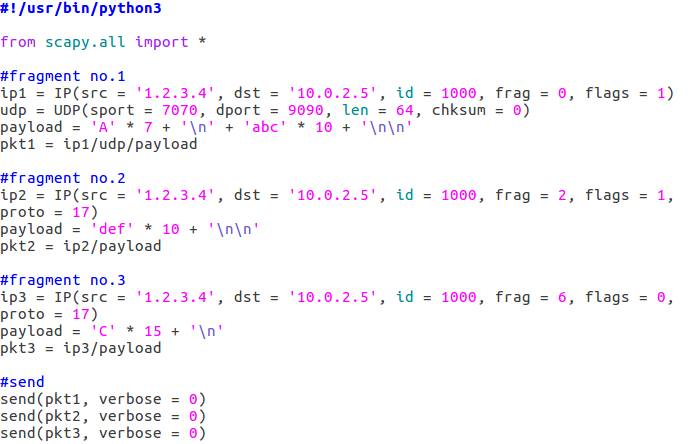


可以看到这一次打印的是第一个分片的重叠数据。

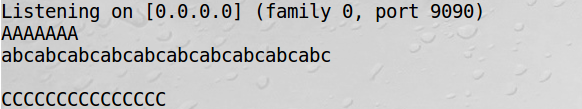
这说明了如果发送的IP分片中有重叠的数据部分，后到达的数据将覆盖掉前面的数据。

2.2第二个分片完全包含在第一个分片中

设置第一个IP分片包含8字节UDP头部和40字节数据，第二个IP分片包含32字节数据，其中32字节全部重叠，第三个IP分片包含16字节数据：

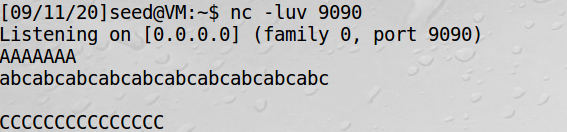


运行代码：



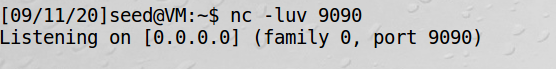
可以看到此时即使是第二个分片后发送，也没有覆盖第一个分片的数据。

下面调整发送分片的顺序，让第一个分片在第二个分片后面发送：



结果未发生改变，打印的仍是第一个分片的数据。

修改第二个IP分片为最后一个分片并取消第三个分片的发送：



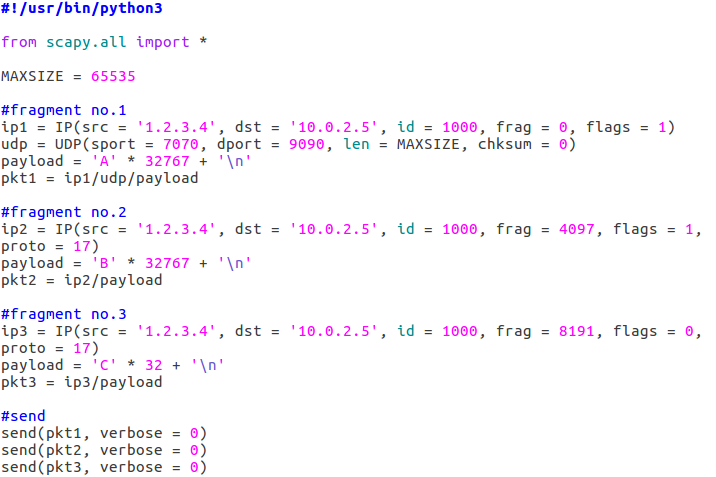


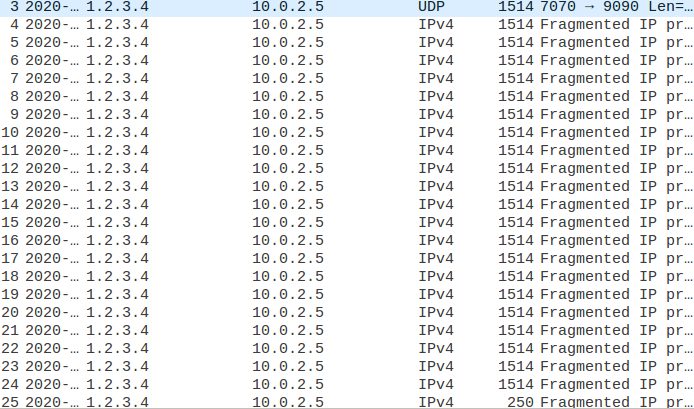
观察wireshark可以看到虚拟机A还发送了一个ICMP时间超过报文，说明第二个分片作为结尾的MF=0字段信息并未被IP层软件获取，该分片很可能被丢弃。

由此可以猜测，在本虚拟系统中，当报文段有重复数据时，如果只是部分重叠，那么IP层会收下重复报文，且后到达的报文数据将覆盖前面的数据；但如果一个报文完全被包含在了另一个报文中，那么IP层会认为该报文是一个错误的分片，不会收下其中的数据。

1. 发送一个超长报文。

分片处理超过65535字节的IP数据报，由于片偏移最多13位，所以最后一个分片的片偏移最大为8191，也就是8191\*8=65536-8个字节，那么算上IP数据报头部20字节，已经超过了IP数据报最大长度。且最后一个报文的长度最大也可以达到65535字节，所以完整的IP数据报最大可达到131063个字节长。

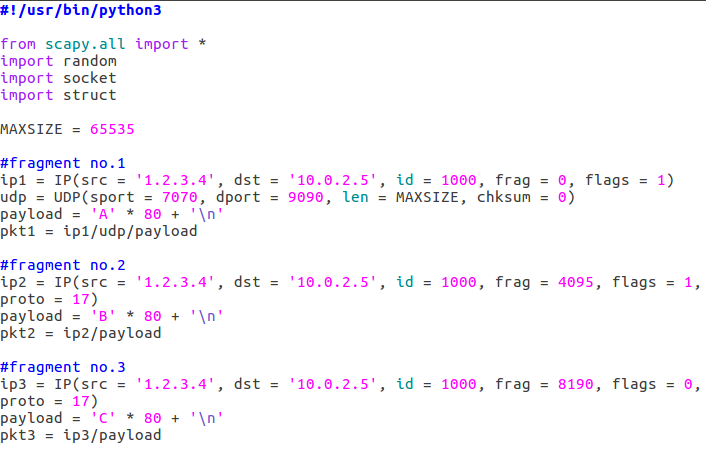


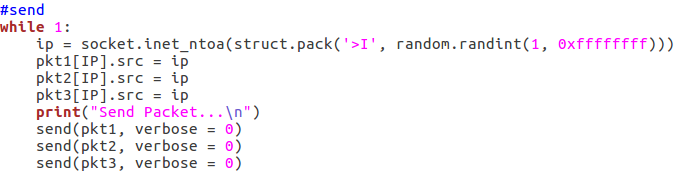


从wireshark可以看到过长的IP数据报又被分片处理后发送，但整个IP数据报共有65536+25=65561字节长。

1. 发送不完整的IP报文。

构造不完整的IP数据报分片：





不断改变源IP地址进行发送，但在此期间9090端口仍能够提供udp数据传输服务，拒绝服务攻击未能完成。

* **实验结论**

通过这一实验，我们验证了IP分片的各种情况，如分片有数据重叠时会发生什么，如何利用分片发送一个超大报文等等。利用这些知识，我们可以实施超长报文导致的缓冲区溢出攻击，以及大量IP数据报的分片缺失导致的拒绝服务攻击。