# 网络安全综合实践（二）

## 数据包嗅探和欺骗实验

华中科技大学网络空间安全学院

二零二一年十二月

# 实验 7 数据包嗅探和欺骗实验



## 1.1 实验目的

数据包嗅探和欺骗是网络安全中的两个重要概念；它们是网络通信中的两大威胁。能够理解这两种威胁对于理解网络中的安全措施至关重要。有许多数据包嗅探和欺骗工具，如 Wireshark，Tcpdump，Netwox，Scapy 等。其中一些工具被安全专家以及黑客广泛使用。能够使用这些工具对学生来说非常重要，但对于网络安全专业的学生来说，更重要的是了解这些工具的工作原理，即数据包嗅探和欺骗是如何在软件中实现的。

本实验的目标有两个：学习使用数据包嗅探和欺骗工具以及理解这些工具的底层实现技术。对于第二个目标，学生将编写简单的嗅探和欺骗程序，并深入了解这些程序的实现技术。本实验包含以下主题：

- Scapy
- 使用 pcap 库进行嗅探
- 原始套接字(Raw Socket)

## 1.2 实验环境

VirtualBox Workstation 虚拟机。

Ubuntu 16.04 操作系统（SEEDUbuntu16.04）。

**网络设置：**要进行此实验，需要 2 台机器。一台计算机用于攻击，第二台计算机用作受害者。可以在同一台主机上设置 2 台虚拟机，也可以使用 docker 容器。在本实验中，我们在 vmware 虚拟机中建立 docker 容器，搭建一个拓扑如图 1.1 所示的网络拓扑。

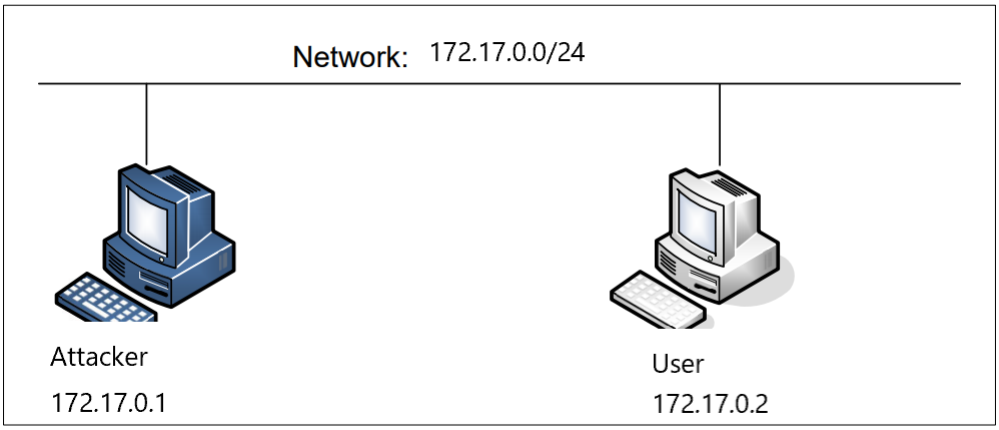


图 1.1 拓扑结构

## 1.3 实验内容

### 1.3.1 使用 scapy 来嗅探和欺骗数据包

许多工具可以用于进行嗅探和欺骗，但其中大多数只提供固定的功能。Scapy 是不同的：它不仅可以当作工具使用，还可以用于构建其他嗅探和欺骗工具，即我们可以将 Scapy 功能集成到我们自己的程序中。

在这组任务中，我们将使用 Scapy 执行每项任务。 要使用 Scapy，我们可以编写一个 Python 程序，然后使用 Python 执行该程序。请参阅以下示例。我们应该使用 root 权限运行 Python，因为欺骗数据包需要特权。在程序开始时（Line①），我们应该导入所有的 Scapy 模块。

```
$ view mycode.py
#!/bin/bin/python

from scapy.all import *    ①

a = IP()
a.show()

$ sudo python mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  ...
```

我们也可以进入 Python 的交互模式，然后在 Python 提示符下一次运行我们的程序。如果我们需要在实验中经常更改代码，这会更方便。

```
$ sudo python
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ IP ]###
  version   = 4
  ihl       = None
  ...
```

#### 1.3.1.1 数据包嗅探

Wireshark 是最受欢迎的嗅探工具，它易于使用。我们将在整个实验中使用到它。但是，

很难将 Wireshark 用于构建其他数据包嗅探和欺骗工具。我们将使用 Scapy 来达到这个目的。这项任务的目的是学习如何使用 Scapy 在 Python 程序中进行数据包嗅探。

使用 Scapy 进行嗅探操作，最核心的函数即是 sniff。它有一些常用的入参，如下表所示：

```
count   需要捕获的包的个数，0 代表无限
store   是否需要存储捕获到的包
filter  指定嗅探规则过滤，遵循 BPF （伯克利封包过滤器）
timeout 指定超时时间
iface   指定嗅探的网络接口或网络接口列表，默认为 None，即在所有网络接口上嗅探
prn 传入一个可调用对象，将会应用到每个捕获到的数据包上，如果有返回值，那么它不会显示
offline 从 pcap 文件读取包数据而不是通过嗅探的方式获得
```

以下提供示例代码：

```
#!/usr/bin/python3

from scapy.all import *

pkt = sniff(iface='docker0', filter='icmp or udp', count=10)

pkt.summary()
```

上面的代码在接口“docker0”上监听，通过过滤器的设置，只监听 icmp 或 udp 的报文，监听个数为 10，符合过滤规则的报文达到 10，sniff 函数就会返回。

### （1） 使用回调函数

我们再来看另一个程序示例。

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

def print_pkt(pkt):
  pkt.show()

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

上面的程序，对于每个捕获的数据包，将调用回调函数 print_pkt()；此函数将打印出有关数据包的一些信息。使用 root 权限运行程序并演示你确实可以捕获数据包。之后，再次运行程序，但不使用 root 权限； 描述和解释你观察到的结果。

```
// Run the program with the root privilege
$ sudo python sniffer.py

// Run the program without the root privilege
$ python sniffer.py
```

### （2）设置过滤器

通常，当我们嗅探数据包时，我们只对某些类型的数据包感兴趣。 我们可以通过在嗅探中设置过滤器来做到这一点。Scapy 的过滤器使用 BPF(Berkeley Packet Filter)语法；可以从

Internet 上找到 BPF 手册。请尝试设置以下过滤器并演示（每个过滤器应单独设置）：

- 仅捕获 ICMP 数据包；
- 捕获来自特定 IP 且目标端口号为 23 的任何 TCP 数据包；
- 捕获数据包来自或去往特定子网。可以选择任何子网，例如 128.230.0.0/16，不要选择 VM 所连接的子网。

#### 1.3.1.2 伪造 ICMP 数据包

作为数据包欺骗工具，Scapy 允许我们将 IP 数据包的字段设置为任意值。此任务的目标是使用任意源 IP 地址假冒 IP 数据包。我们将假冒 ICMP echo 请求数据包，并将它们发送到同一网络上的另一个 VM，使用 Wireshark 来观察我们的请求是否会被接收方收到，如果它被收到，则应答包将被发送到假冒的 IP 地址。以下代码显示了如何假冒 ICMP 数据包的示例。

```
>>> from scapy.all import *
>>> a = IP()                ①
>>> a.dst = '10.0.2.3'      ②
>>> b = ICMP()              ③
>>> p = a/b                 ④
>>> send(p)                 ⑤
.
Sent 1 packets.
```

在上面的代码中，Line①创建一个 IP 对象；为每个 IP 头字段定义一个属性。我们可以使用 ls(a)或 ls(IP)来查看所有属性名称/值。我们也可以使用 a.show()和 IP.show()来做同样的事情。Line②显示如何设置目标 IP 地址字段。如果未设置该字段，则将使用默认值。

```
>>> ls(a)
version    : BitField (4 bits)       = 4               (4)
ihl        : BitField (4 bits)       = None            (None)
tos        : XByteField              = 0               (0)
len        : ShortField              = None            (None)
id         : ShortField              = 1               (1)
flags      : FlagsField (3 bits)     = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField (13 bits)      = 0               (0)
ttl        : ByteField               = 64              (64)
proto      : ByteEnumField           = 0               (0)
chksum     : XShortField             = None            (None)

src        : SourceIPField           = '127.0.0.1'     (None)
dst        : DestIPField             = '127.0.0.1'     (None)
options    : PacketListField         = []              ([])
```

Line③创建一个 ICMP 对象。默认类型是 echo 请求。在 Line④中，我们将 a 和 b 堆叠在一起以形成一个新对象。/运算符由 IP 类重载，因此它不再代表除法； 相反，它意味着添加 b 作为 a 的有效载荷字段并相应地修改 a 的字段。结果，我们得到一个代表 ICMP 数据包的新对象。我们现在可以使用 Line⑤ send()发送此数据包。请对示例代码进行必要的更改，然后演示你可以使用任意源 IP 地址假冒 ICMP echo 请求数据包。

#### 1.3.1.3 路由追踪

此任务的目标是使用 Scapy 估计 VM 与所选目标之间的路由器数量距离。这基本上是 traceroute 工具实现的。在这个任务中，我们将编写自己的工具。这个想法很简单：只需将数据包（任何类型）发送到目的地，其生存时间（TTL）字段首先设置为 1。第一个路由器将丢弃

此数据包，并给发送方返回 ICMP 错误消息，告诉我们生存时间已超过。这就是我们获取第一个路由器的 IP 地址的方式。 然后我们将 TTL 字段增加到 2，发送另一个数据包，并获取第二个路由器的 IP 地址。我们将重复此过程，直到我们的数据包最终到达目的地。应该注意的是，该实验仅获得估计结果，因为理论上并非所有这些包都采用相同的路径（但实际上，它们可能在短时间内采用相同的路径）。以下代码显示了该过程中的一轮。

```
a = IP()
a.dst = '1.2.3.4'
a.ttl = 3
b = ICMP()
send(a/b)
```

通过 wireshark 抓包，查看 ICMP 错误报告的源 IP 地址，来得到从源到目的主机的路径上的路由器信息。

如果你是一位经验丰富的 Python 程序员，可以编写工具来循环自动执行整个过程。 如果你不熟悉 Python 编程，可以通过手动更改每轮中的 TTL 字段来完成，并根据你在 Wireshark 中的观察记录 IP 地址。无论哪种方式都可以接受，只要你得到结果。

#### 1.3.1.4 嗅探然后欺骗

在此任务中，将结合嗅探和欺骗技术来实现以下 sniff-and-then-spoof 程序。实验环境如下：A 为 user 容器，B 为虚拟机，X 为 Internet 上的一台主机。

A（容器）--------------------B（VM）-------Internet---------X

在 A 上，你 ping IP X。这将生成 ICMP echo 请求数据包。如果 X 处于活动状态，ping 程序将收到 echo 回复，并打印出响应信息。

你的 sniff-and-then-spoof 程序在虚拟机 B 上运行，虚拟机 B 通过数据包嗅探来监控 LAN。每当它监测到 ICMP echo 请求时，无论目标 IP 地址是什么，你的程序都应立即使用数据包欺骗技术发出 echo 响应。因此，无论机器 X 是否处于活动状态，A 上的 ping 程序将始终收到回复，指示 X 处于活动状态。

你需要使 Scapy 来完成此任务，在作业中，需要提供证据证明你的数据包欺骗技术有效。(比如随便 ping 地址 1.2.3.4， 3.4.5.6 等，B 在没有运行你的程序时，A ping 不通；但是 B 上运行欺骗程序后，A 就能 ping 通了)。

### 1.3.2 使用 pcap 库编写数据包嗅探程序

可以使用 pcap 库轻松编写嗅探器程序。使用 pcap，编写嗅探器的任务变成在 pcap 库中调用一系列简单的过程。在调用序列结束时，数据包被捕获后将立即放入缓冲区进行进一步处理。捕获数据包的所有细节都由 pcap 库处理。下面的示例代码展示了如何使用 pcap 编写简单的嗅探器程序。

```c
#include <pcap.h>
#include <stdio.h>

/* This function will be invoked by pcap for each captured packet.
   We can process each packet inside the function.
 */
void got_packet(u_char *args, const struct pcap_pkthdr *header,
        const u_char *packet)
{
   printf("Got a packet\n");
}

int main()
{
  pcap_t *handle;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct bpf_program fp;
  char filter_exp[] = "ip proto icmp";
  bpf_u_int32 net;

  // Step 1: Open live pcap session on NIC with name eth3
  //         Students needs to change "eth3" to the name
  //         found on their own machines (using ifconfig).
  handle = pcap_open_live("eth3", BUFSIZ, 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;
}

// Note: don't forget to add "-lpcap" to the compilation command.
// For example: gcc -o sniff sniff.c -lpcap
```

Tim Carstens 还编写了一个关于如何使用 pcap 库编写嗅探器程序的教程。该教程可从 http://www.tcpdump.org/pcap.htm 获得。

#### 1.3.2.1 了解嗅探器的工作原理

在此任务中，学生需要编写嗅探器程序以打印出每个捕获的数据包的源和目标 IP 地址。可以输入上述代码。学生应提供屏幕截图作为证据，证明他们的嗅探程序可以成功运行并产生预期结果。另外，请回答以下问题：

- **问题 1**. 请用自己的话来描述用于嗅探程序的至关重要的库函数调用顺序。 可以简单描述，不需要像教程或书中那样详细的解释。
- **问题 2.** 为什么需要 root 权限才能运行嗅探程序？如果在没有 root 权限情况下执行程序，程序在哪里失败？
- **问题 3.** 请打开并和关闭嗅探程序中的混杂模式。这种模式开启和关闭时，你能证明其中的差异吗？请描述一下如何证明这一点。

#### 1.3.2.2 编写过滤器

嗅探器程序编写过滤器表达式以捕获下列每个数据包。可以找到 pcap 过滤器的在线手册。在实验报告中，需要包含屏幕截图来展示在应用每个过滤器后的结果。

- 捕获两个特定主机之间的 ICMP 数据包。
- 捕获目标端口号在 10 到 100 范围内的 TCP 数据包。

#### 1.3.2.3 嗅探密码

当某人在你监视的网络上使用 telnet 时，如何使用你的嗅探器程序抓取密码？你可能需要修改嗅探器代码以打印出捕获的 TCP 数据包的数据部分 （telnet 使用 TCP）。如果打印出整个数据部分，然后手动标记密码（或部分密码） 的位置，也可以接受。

### 1.3.3 使用原始套接字构造报文

当普通用户发送数据包时，操作系统通常不允许用户设置协议头中的所有字段（例如 TCP，UDP 和 IP 头）。操作系统将设置大多数字段，同时仅允许用户设置一些字段，例如目标 IP 地址，目标端口号等。但是，如果用户具有 root 权 限，则可以在数据包中设置任意字段。这称为数据包欺骗，通过原始套接字可以完成该目标。

原始套接字为程序员提供了对数据包构造的绝对控制权，它允许程序员构造任意数据包，包括设置头字段和有效负载。使用原始套接字非常简单；

它包括四个步骤：（1）创建一个原始套接字，（2）设置套接字选项，（3）构造数据包，以及（4）通过原始套接字发送数据包。有许多在线教程可以教你如何在 C 编程中使用原始套接字。

下面的代码片段我们展示了这样一个程序的简单框架。

```
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);
}
```

#### 1.3.3.1 编写一个欺骗程序

请在 C 中编写自己的数据包欺骗程序。需要提供证据（例如，Wireshark 数据包跟踪）以显示程序成功发送了假冒的 IP 数据包。

#### 1.3.3.2 伪造 ICMP echo 请求

代表另一台机器假冒 ICMP echo 请求数据包（使用另一台机器的 IP 地址作为其源 IP 地址）。应该将此数据包发送到 Internet 上的远程计算机（计算机必须处于活动状态）。 在假冒的机器上打开 Wireshark，如果欺骗成功，可以看到从远程计算机返回的 ICMP echo 回复。

#### 1.3.3.3 嗅探然后欺骗

在此任务中，将结合嗅探和欺骗技术来实现以下 sniff-and-then-spoof 程序。 需要在同一局域网上安装两个虚拟机（VM）。在 VM A 上， ping IP X。这将生成 ICMP echo 请求数据包。如果 X 处于活动状态，ping 程序将收到 echo 回复，并打印出响应信息。sniff-and-then-spoof 程序在 VM B 上运行，VM B 通过数据包嗅探来监视局域网。每当它监测到 ICMP echo 请求时，无论目标 IP 地址是什么，你的程序都应立即使用数据包欺骗技术发出 echo 回复。因此，无论机器 X 是否处于活动状态，ping 程序将始终收到回复，指示 X 处于活动状态。 需要使用 C 语言编写这样一个程序，并在报告中包含屏幕截图以显示你的程序是否有效。请在报

告中附上代码（含有足够的注释）

#### 1.3.3.4 问题思考

**问题 4.** 无论实际数据包有多大，你都可以将 IP 数据包长度字段设置为任意值吗？

**问题 5**. 使用原始套接字编程，是否必须计算 IP 头的校验和？

**问题 6.** 为什么需要 root 权限才能运行使用原始套接字的程序？ 如果在没有 root 权限的情况下执行，程序在哪里失败？

## 1.4 实验指南

### 1.4.1 填充原始数据包中的数据

当使用原始套接字发送数据包时，基本上是在缓冲区内构建数据包，因此当需要将数据包发送出去时，只需向操作系统提供缓冲区和数据包的大小。 直接在缓冲区上工作并不容易，因此常见的方法是将缓冲区（或缓冲区的一部分） 转换为结构体，例如 IP 头结构体，这样就可以使用这些结构体的字段来引用缓 冲区的数据。可以在程序中定义 IP，ICMP，TCP，UDP 和其他协议头结构。 以下示例说明如何构造 UDP 数据包：

```
struct ipheader {
   type  field;
   .....
}

struct udpheader {
   type field;
   ......
}

// This buffer will be used to construct raw packet.
char buffer[1024];

// Typecasting the buffer to the IP header structure
struct ipheader *ip = (struct ipheader *) buffer;

// Typecasting the buffer to the UDP header structure
struct udpheader *udp = (struct udpheader *) (buffer
                            + sizeof(struct ipheader));

// Assign value to the IP and UDP header fields.
ip->field = ...;
udp->field = ...;
```

### 1.4.2 网络/主机字节顺序和转换

编写 C 程序的时候需要注意网络和主机字节顺序。如果使用 x86 CPU，则主机字节顺序使

用 Little Endian，而网络字节顺序使用 Big Endian。无论放入数据包缓冲区的数据是什么，都必须使用网络字节顺序；如果不这样做，数据包将是不正确的。 实际上不必担心机器正在使用什么样的 Endian，实际上不应该担心程序是否可移植。需要做的是始终记住在将数据放入缓冲区时将数据转换为网络字节顺序，并在将数据从缓冲区复制到计算机上的数据结构时将它们转换为主机字节顺序。 如果数据是单个字节，则无需担心顺序，但如果数据是 short，int，long 或包含多个字节的数据类型，则需要调用以下转换数据的函数：

```
htonl(): convert unsigned int from host to network byte order.
ntohl(): reverse of htonl().
htons(): convert unsigned short int from host to network byte order.
ntohs(): reverse of htons().
```

可能还需要使用 inet_addr()，inet_network()，inet_ntoa()，inet_aton()将 IP 地址从点分十进制形式（字符串）转换为 32 位整数的网络/主机字节顺序。可以从 Internet 上获取它们的使用手册。

## 1.5 实验提交

实验结果提交到微助教平台，本实验不需上机检查和提交报告。