# 1 蜜罐和网络背景

本章介绍互联网协议的简要背景。描述最重要的 TCP（Transmission Control Protocol，传输控制协议）和 IP（Internet Protocol，因特网协议）路由协议。其中，关于 ARP（Address Resolution Protocol，地址解析协议）链路层的一些信息对于理解数据包是如何到达终端主机是必要的。此外，还介绍了蜜罐的基本概念，所有不同蜜罐解决方案的基本思想，给出了一些简要的背景以及各自的优缺点。已经了解这些基本知识的读者可以跳过本章。

## 1.1 TCP/IP 协议简介

所谓的 Internet 协议簇就是通信协议集合，它们实现了互联网运行的协议栈。它的命名源于两个最重要的协议 TCP 和 IP，下面简要介绍它们及相关协议，这应足以提供网络的基本概念，并帮助你了解后续章节介绍的网络知识。根据需要，在通篇的适当位置引入网络的其他概念。对于想深入了解 Internet 协议簇的读者，请查阅参考文献[8,92,97]。这些书籍综览了网络的所有方面，并关注于它们实际的实现。

Internet 协议簇可以看作是一组不同层的集合，每一层负责一个具体任务，各层联合工作实现整个网络通信。每一层与其上层或下层都有一个明确定义的接口来具体说明希望处理的数据。Internet 协议簇共包括 5 个不同的层：

（1）应用层：这一层包含应用程序实现服务所使用的所有协议。重要的例子如万维网使用的 HTTP 协议，文件传输使用的 FTP 协议，接收电子邮件使用的 POP3、IMAP 协议，发送邮件使用的 SMTP 协议，以及远程登录使用的 SSH 协议。作为用户，通常与应用层进行交互。

（2）传输层：这一层响应来自应用层的服务请求，并向网络层发出服务请求。传输层提供两台主机之间透明的数据传输，通常用于端到端连接、流量控制或错误恢复。这一层的两个最重要的协议是 TCP 和 UDP（User Datagram Protocol，用户数据报协议），后面将简单介绍它们。

（3）网络层：网络层提供端到端的数据包交付，换言之，它负责数据包从源发送到目的地，

任务包括诸如网络路由、差错控制和 IP 编址。这一层包括的重要协议有 IP（版本 4 和版本 6）、ICMP（Internet Control Message Protocol，Internet 控制报文协议）和 IPSec（Internet Protocol Security，Internet 协议安全）。

（4）数据链路层：链路层是从路由器到终端主机必需的桥梁，它负责同一个本地网络中节点之间的数据传输，也负责广域网中相邻节点之间的数据传输。它包括 ARP、ATM（Asynchronous Transfer Mode，异步传输模式）和以太网等协议。

（5）物理层：Internet 协议簇中的最底层，定义了在两个连接的网络节点间如何发送原始比特。其中，协议规定了比特流如何编码以及如何将比特转换为物理信号。这一层的一些协议和技术有 ISDN（Integrated Services Digital Network，综合服务数字网）、Wi-Fi（wireless LAN，无线局域网）和调制解调。

IP、ARP、UDP 和 TCP 是必须了解的协议，下面给出简要概述。

IPv4[①]是一个面向数据的协议，设计用于分组交换网络（例如以太网），是一个尽最大努力交付协议。这意味着它并不保证一个主机发送的 IP 数据包能够被目的地主机接收到。此外，它并不能保证 IP 数据包被正确接收：一个数据包可能会被乱序接收或根本接收不到。这些问题由传输层协议解决，特别是 TCP 实现的几个机制保证了在 IP 之上的可靠数据传输。IP 通过所谓的 IP 地址实现寻址，互联网中的每个主机都有一个 IP 地址，可以把它想象为一个地址，在这一地址下主机是可达的。通常，一个 IP 地址用点分十进制法表示——也就是 4 个十进制表示的字节由圆点分隔：192.0.2.1，通过这个 IP 地址，网络中的其他主机可以联络这个主机。此外，IP 实现了分片概念：由于不同类型的网络发送数据报文有不同的最大数据量限制，可能需要将一个报文分解成若干较小的报文，这就是 IP 分片，而且由于端主机不得不将不同分片重新组装起来，因此需要 IP 重组。

MAC 地址是网络适配器的物理地址，可以查看 IP 和网络适配器的 MAC 地址，在 UNIX 系统中使用命令/sbin/ifconfig，而在 Windows 系统中使用命令 ipconfig/all。由于数据链路层通常应用自己的寻址方案，必须有一个协议能够把一个 IP 地址（网络层）映射成相对应的 MAC 地址（数据链路层）。如果你的计算机想与本地网络上的另一台计算机通信，只能使用数据链路层，因为远程计算机的网络适配器只监听带有它的 MAC 地址的网络数据包。数据链路层接收到网络层的一个 IP 数据包，只知道目的 IP 地址，因此，它必须找出哪个 MAC 地址属于给定的 IP 地址，ARP 所做的就是把 IP 地址解析为 MAC 地址。下面的例子显示了这一协议的会话过程。IP 地址为 10.0.1.6 的主机发送一个数据包给 IP 地址为 10.0.1.91 的主机，由于发送者不知道目的主机的物理地址，它会发出一个广播，即向网络中所有主机发送一个请求，IP 地址为 10.0.1.91 的主机接受这个请求并向发送者发送一个包含其 MAC 地址的答复：

19:34:35.54 arp who-has 10.0.1.91 tell 10.0.1.6

19:34:35.54 arp reply 10.0.1.91 is-at 00:90:27:a0:77:9b

现在我们知道，ARP 用于把一个 IP 地址映射为一个物理地址，而 IP 负责从源到目的地的 IP

[①]我们只关注 IPv4 是因为两个主要原因：IPv6 还没有广泛应用，而且几乎没有针对 IPv6 网络的蜜罐解决方案。

数据包的路由。ARP 允许我们从一台主机到另一台主机透明地重定向流量，它也允许一台主机接收来自许多不同 IP 地址主机的流量。在后面的章节中，我们将谈论 Honeyd，以及它如何使用 ARP 在网络上创建数百个虚拟蜜罐。

现在我们来快速浏览一下两个最重要的传输层协议：UDP 和 TCP。使用 UDP，两个运行在通过网络连接的两台不同计算机上的应用程序可以交换消息，通常称为数据报（利用所谓的数据报套接字）。UDP 是 IP 上面的一个层，是无状态的，即发送主机发送 UDP 消息后不保留消息状态。这是一个几乎没有开销的非常简单的网络协议，基本上 UDP 仅提供应用程序多路复用（换言之，为运行在同一台计算机上的多个并发应用程序产生的多个连接区分数据）、报文头和载荷的校验和。UDP 协议最主要的缺点是，它不提供任何的可靠性和数据报有序性，数据报到达时可能是无序的、重复出现的，甚至根本没有到达目的主机。它不直接处理报文丢失或报文重新排序。没有检查每一个数据报是否都到达的开销，对许多轻量级或时间敏感的应用，UDP 更快更有效。因此，这一协议通常用于流媒体（IP 语音或视频聊天）和在线游戏，对于这些应用，丢失一些数据报不是至关重要的。UDP 另一个重要应用是域名系统（Domain Name System，DNS），用来把一个给定 URL 解析为一个 IP 地址。

另一方面，TCP 是面向连接的，通过所谓的数据流，在两个网络主机之间提供一个可复用的、可靠的通信信道。TCP 保证数据从发送者到接收者可靠和有序地交付，相比较地，UDP 不能保证这些属性。TCP 从应用层接收字节流，把它们分成适当大小的片段，然后这些片段被交给网络层（通常为 IP），对它们进行进一步的处理。TCP 给每个报文一个序列号，通过检查序列号以确保没有报文丢失。后面我们讨论如何建立 TCP 会话时，将会仔细讨论序列号。运行在接收主机上的 TCP，为所有已成功接收到的报文返回一个确认。与序列号一起，这个确认号用于检查是否收到所有报文，需要的话可以对它们重新排序。如果在合理的时间范围内没有接收到确认，发送主机上的定时器将产生一个超时，根据这一信息，如有需要可以重传丢失报文。同时，TCP 使用校验和来控制是否正确接收一个给定的报文段。此外，TCP 应用拥塞控制，实现高性能和避免网络链路拥塞。如上所述，TCP 有些复杂，但与 UDP 相比它有很多优势。如果两台主机需要可靠的网络通信，通常使用 TCP。例如，对于万维网使用的 HTTP 协议、电子邮件相关应用程序使用的 SMTP 和 POP3/IMAP 协议，以及数据传输使用的 FTP 协议，这是必要的。

下面将介绍报文头，并解释如何建立 TCP 连接，但我们不会介绍得过细。你可以找到许多书籍，集中讨论 TCP / IP 网络、所有相关协议以及这些协议是如何交互的[8,92,97]。

图 1.1 显示了 TCP 头部结构，这是一个简化版本，但它包含了理解 TCP 主要内容的足够细节。一开始，两个 16 位字段指定源和目的端口，在传输层中端口用于多路复用，通过网络端口，不同的应用程序仅在一个 IP 地址上监听成为可能。例如，典型地，一个 Web 服务器监听 TCP 端口 80，而一个 SMTP 服务器使用 TCP 端口 25，通过复用两个服务器“共享”主机 IP 地址。TCP 头接下来的两个字段包含 32 位序号和确认号。序号有两个重要的作用，首先用于配置连接时设置初始序号，如果连接建立，有效载荷中的第一个数据字节就是序号。如果 ACK 标志被设置，确认号指定发送者期望的下一个序号。

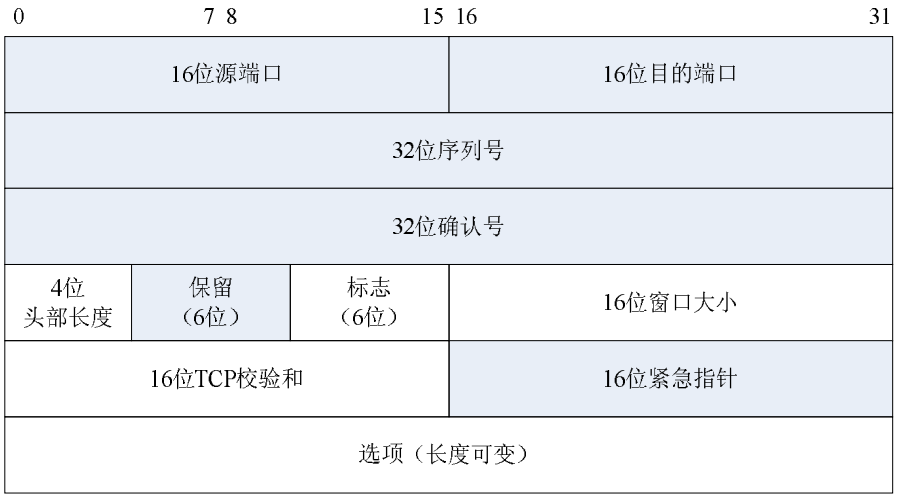


图 1.1　TCP 头部不同字段和选项概述



头部长度字段指定 TCP 头部的 32 位字的长度，最小为 5 个字，最大为 15 个字，进而该字段指定了从 TCP 报文段开始到数据的偏移量。接下来的 6 位保留，用于将来 TCP 需要扩展时使用，这些保留位中有两位已经被最新的 TCP 协议栈使用，但出于简单介绍的目的，我们跳过这些内容。对 TCP 更重要的是接下来的 6 个比特：即所谓的 TCP 标志，它们用于提供有关当前 TCP 报文段状态的信息：

- SYN：用于连接设置期间同步序号。
- ACK：确认号是否重要的标识，如果设置该位，那么确认号就是发送者期望接收的下一个序号。
- RST：重置连接，主要用于错误状态中。
- FIN：用于拆除连接，因此没有更多的数据要发送。
- PSH：推送功能。
- URG：指示紧急指针字段是否重要。

窗口大小字段用于指定发送者希望接收的从确认号开始的字节数。TCP 头还包含一个校验和，用于在目的地检验到达的报文段是否未被篡改。只有设置 URG 标志时，才使用紧急指针字段，它指定了从序号开始的偏移量，指明 TCP 有效载荷中从哪个点开始的数据应立即移交给应用层。TCP 头还有其他可选字段，这里就不再讨论了，因为它们几乎与蜜罐部署不相关。

因为 TCP 在发送者和接收者之间建立了一个连接，它需要在通信刚开始时创建一个连接。这是通过所谓的 TCP 握手完成的，这个握手主要通过交换序号和确认号在两个主机之间同步状态，这些数字稍后用于确认目的主机是否正确收到给定的报文段，以及用于重传和拥塞控制。TCP 握手过程需要在发送者 S 和接收者 R 之间交换三个协议消息：

（1）S→R：发送方发送一个带有 SYN 标志位（置 1）和一个序号为 x 的报文段。

（2）S←R：接收者应答一个带有 SYN 和 ACK 标志位（都置 1）的 TCP 报文段，确认号被设置为主机希望接收的下一个序号，这个例子中是 x + 1。此外，接收者将他的序号设置为 y，因为他也希望与另一方同步这个序号。

（3）S→R：发送方发送一个带有 ACK 标志位（置 1）的 TCP 报文段，他响应的报文序号为 x + 1，同时增加确认号为 y+1。

这一握手过程之后，双方都知道对方的序号和确认号的当前值，之后，此信息被用于 TCP 所有目标，例如，无差错数据传输和拥塞控制。

Internet 协议簇的另一个重要方面是 IP 路由，理解 IP 路由的重要性有多种原因：它是主机之间可以相互通信的基本方法，并提供对互联网拓扑结构和小型网络（一个企业网络）拓扑结构的深入理解。为了成功地创建非常复杂的蜜网，基本了解互联网路由是很重要的。

## 1.2 蜜罐背景

在我们从高级技术层面开始讨论蜜罐技术之前，这一主题的背景知识对我们是有帮助的。为了激发我们使用蜜罐技术，首先看一下网络入侵检测系统（NIDS）[64]是有必要的。面对越来越复杂的逃避技术[70,105]，以及采用加密技术来保护网络通信，防止被窃听的协议越来越多，入侵检测系统能提供的有用信息数量越来越少。入侵检测系统也受到高误报率的困扰，从而进一步降低了它们的作用。蜜罐技术能够帮助解决一些问题。

蜜罐是一个被严密监控的计算资源，希望被探测、攻击或者攻陷。更准确的描述是，蜜罐是“一个信息系统资源，其价值在于未经授权或非法使用该资源”（该定义来自蜜罐邮件列表 SecurityFocus，http://www.securityfocus.com/archive/119）。一个蜜罐的价值可以由从它可获得的信息来衡量。通过监测进出蜜罐的数据来收集 NIDS 无法获得的信息。例如，即使使用了加密技术保护网络流量，我们仍然能够记录一个交互式会话中的按键。为了检测恶意行为，入侵检测系统需要已知攻击特征，而通常检测不到未知的攻击。另一方面，蜜罐可以检测未知的攻击，例如，通过观察离开蜜罐的网络流量，我们可以检测到漏洞威胁，即使是使用从未见过的漏洞利用手段。

因为蜜罐没有生产价值，任何连接蜜罐的尝试都被认为是可疑的。因此，分析蜜罐收集的数据所产生的误报比从入侵检测系统收集到的数据导致的误报少。在蜜罐的帮助下，我们所收集的大部分数据可以帮助我们了解攻击。

蜜罐可以运行任何操作系统和任意数量的服务，配置的服务决定了敌手可用的损害和探测系统的媒介。高交互蜜罐提供了一个攻击者可以交互的真实系统，相反，低交互蜜罐只能模拟一部分功能，例如网络堆栈[67]。高交互蜜罐可以完全被攻陷，允许敌手获得对系统的完全访问，并实施进一步的网络攻击。与此相反，低交互蜜罐只是模拟一些服务，敌手不能利用这些服务获得对蜜罐的完全访问。低交互蜜罐有更多的限制，但它们有助于在更高层面上收集信息，例如，了解网络探测或蠕虫活动，它们也可以用于分析垃圾邮件，或主动防范蠕虫，参见第 10 章如何使用不同种类蜜罐的案例研究综述。这两种方法的任一方法都不优于对方，它们有各自独特的优缺点，将在这本书

中进行介绍。

密罐还分为物理蜜罐和虚拟蜜罐。一个物理蜜罐是网络上一台拥有自己 IP 地址的真正机器，一个虚拟蜜罐由另一台机器模拟，它响应发送给虚拟蜜罐的网络流量。

当收集有关网络攻击或探测的信息时，蜜罐部署的数量会影响收集数据的数量和精确性。测量基于 HTTP 的蠕虫[68]是一个很好的例子，当蠕虫完成 TCP 握手并发送有效载荷之后，我们可以识别这些蠕虫。然而，由于它们随机选择 IP 地址进行连接，所以绝大多数的连接请求不会被响应。通过把蜜罐配置为一个 Web 服务器，或模拟成一个有漏洞的网络服务，蜜罐可以捕获蠕虫的有效载荷。我们部署的蜜罐越多，蠕虫连接它们的可能性越大。

一般来说，蜜罐有几种不同的类型，除此之外，我们可以混合并匹配出不同的类型，将在后续的章节中详细解释和讨论。在深入探究虚拟蜜罐这一领域之前，首先纵览一下不同类型的蜜罐。

### 1.2.1 高交互蜜罐

一个高交互蜜罐是一个常规的计算机系统，如商用现货（commercial off-the-shelf ，COTS）计算机、路由器或交换机。该系统在网络中没有常规任务，也没有固定的活动用户，因此，除了运行系统上的正常守护进程或服务，它不应该有任何不正常的进程，也不产生任何网络流量。这些假设帮助检测攻击：每个与高交互蜜罐的交互都是可疑的，可以指向一个可能的恶意行为。因此，所有出入蜜罐的网络流量都被记录下来。此外，系统的活动也被记录下来备日后分析。



也可以将几个蜜罐组合成为一个蜜罐网络——蜜网（honeynet）。通常，一个蜜网由多个不同类型的蜜罐组成（不同的平台和/或操作系统），这允许我们同时收集不同类型的攻击数据，通常我们能够了解更全面的攻击信息，并因此得到攻击者行为的定性结论。

蜜网创建了一个玻璃鱼缸的环境，允许攻击者与系统交互，同时给操作者捕捉攻击者所有活动的能力。这个鱼缸还控制了攻击者的行动，减少了他们破坏蜜罐系统的风险。部署蜜罐的一个关键部件称为蜜墙（Honeywall）——一个用于分离蜜网与网络其他部分的二层桥接设备，这个设备通过数据控制和捕捉分析数据降低蜜网风险，蜜墙上的工具考虑到对攻击者活动的分析，任何出入蜜罐的流量必须通过蜜墙。捕获信息有不同的方法，包括被动网络嗅探器、IDS 警报、防火墙日志和被称为 Sebek 的内核模块，将在 2.5.1 节中详细介绍。攻击者的活动被控制在网络层，通过入侵防御系统和连接限制器过滤所有出站连接。

高交互蜜罐的缺点之一是较高的维护量：你必须小心监测你的蜜罐，并密切观察所发生的事情，分析危险还需要一些时间，从我们的经验来看，分析一个完整的事件可能花费数小时甚至数天，直到你完全明白攻击者想干什么！

高交互蜜罐可以完全被攻陷，它们运行着带有所有漏洞的真实的操作系统，没有使用仿真，攻击者可以与真实的系统和真实的服务交互，允许我们捕获大量的威胁信息。当攻击者获得非授权访问时，我们可以捕捉他们的漏洞利用，监视他们的按键，找到他们的工具，或者搞清他们的动机。高交互解决方案的缺点是它们增加了风险：由于攻击者可能完全地访问操作系统，他们就有可能用它来损害其他非蜜罐系统。挑战之一就是将它们扩大到大量计算机规模时所带来的开销和各种问

题。本书将在第 2 章更详细地介绍高交互蜜罐。

### 1.2.2 低交互蜜罐

与此相反，低交互蜜罐模拟服务、网络堆栈或一台真实机器的其他功能，它们允许攻击者与目标系统有限交互，允许我们了解关于攻击的主要的定量信息，例如，一个模拟的 HTTP 服务器，可以只响应对某个特定文件的请求，只实现整个 HTTP 规范的一个子集。交互的层次应该“刚好够用”欺骗攻击者或自动化工具——如一个寻找特定文件而危害服务器的蠕虫。低交互蜜罐的优点是简单和易维护，通常你可以只是配置你的低交互蜜罐，让它为你收集数据，这些数据可以是正在传播的网络蠕虫，或者是垃圾邮件发送者对开放式网络中继的扫描等信息。此外，安装这类蜜罐通常很容易：你只需要安装和配置一个工具就完成了。相比之下，高交互蜜罐只不过是你需为你的环境定制的一般方法。

低交互蜜罐可以主要用于收集统计数据，收集关于攻击模式的高级别信息。进一步地，它们可以作为一种提供预警的入侵检测系统，也就是对新的攻击提供自动报警（见第 10 章）。此外，它们还可以用于引诱攻击者远离生产机器[19,67,87]。另外，低交互蜜罐还可用于检测蠕虫、干扰敌手，或者了解正在进行的网络攻击。本书中将介绍多种不同类型的低交互蜜罐。低交互蜜罐也可以组成一个网络，形成一个低交互蜜网。

因为攻击者只与一个模拟系统交互，不会完全攻陷系统，低交互蜜罐构造了一个可控环境，因此风险有限：攻击者不能完全攻陷系统，因此你不必担心他滥用你的低交互蜜罐。

有许多不同的低交互蜜罐可用，在第 3 章中将介绍几种解决方案，并说明如何使用它们。此外，后续章节重点关注具体的工具，并非常详细地介绍它们。

表 1.1 总结了高交互蜜罐和低交互蜜罐，并对比它们各自重要的优缺点。

**表 1.1 高交互蜜罐和低交互蜜罐的优缺点**

| 高交互蜜罐 | 低交互蜜罐 |
|---|---|
| 真实的服务、操作系统或应用程序 | 模拟的 TCP/IP 协议栈、弱点等 |
| 高风险 | 低风险 |
| 难以部署和维护 | 容易部署和维护 |
| 捕获大量的信息 | 捕获有关攻击的定量信息 |

### 1.2.3 物理蜜罐

蜜罐的另一种可能的分类方法是可以分为物理蜜罐和虚拟蜜罐。物理蜜罐意味着蜜罐运行在一个物理计算机上，物理通常暗指高交互，从而允许系统被完全攻陷。安装和维护它们通常是代价昂贵的。对于大的地址空间，为每个 IP 地址部署一个物理蜜罐是不切实际的，也是不可能的，这种情况下，我们需要部署虚拟蜜罐。

### 1.2.4 虚拟蜜罐

在本书中我们重点关注虚拟蜜罐。为什么这类蜜罐如此有趣呢？主要原因是其可度量性和易维护性，在一台机器上可以有数以千计的蜜罐，部署它们代价低，并且几乎每个人都可以容易地使用它们。

与物理蜜罐相比，这种方法更轻便。我们也可以在一台物理计算机上部署多个虚拟机作为蜜罐，而不是将一个物理计算机系统配置为一个蜜罐，这使得更容易维护和较低的物理需求。通常使用VMware[103]或用户模式 Linux（UML）[102]建立这种虚拟蜜罐，这两个工具允许我们在一台物理机器上并发运行多个操作系统和应用程序，便于收集数据，我们将在第 2 章中详细介绍这两个工具。此外，在其他章节中我们会介绍其他类型的虚拟蜜罐，重点介绍这些蜜罐的不同点。由于本书的重点在虚拟蜜罐，在这里就不详细介绍了。应该记住的主要一点是：一个虚拟蜜罐是由另一台机器模拟的，响应发送给虚拟蜜罐的网络流量。

对于工作的任何蜜罐，外部 Internet 需要能够访问它。我们中许多人通过 DSL 或有线调制解调器连接 Internet，这些设备通常使用网络地址转换（NAT），即使调制解调器后面可能有一个完整的网络，从 Internet 仍无法访问你的内部网络。因此，在采用 NAT 的网上部署蜜罐，你不会得到有价值的数据。某些 NAT 设备允许你改变端口转发配置，至少允许你一点点暴露于 Internet。对于较严格的实验，你应该找一个提供实时非过滤 IP 连接的 ISP。



### 1.2.5 法律方面

蜜罐有一些风险，如果一个攻击者设法攻陷你的一个蜜罐，他可能试图攻击不在你控制之下的其他系统。这些系统可位于 Internet 的任何地方，攻击者可以使用你的蜜罐作为跳板攻击敏感系统，这意味着运行蜜罐时会涉及一些法律问题，但在不同国家有不同的法律，对法律状况很难给出一致观点。我们将不讨论运行一个蜜罐系统的法律问题，因为如果你生活在美国，你可以从《Know Your Enemy》（http://www.honeynet.org/book/）的 Richard Salgado 章节中获得相关法律信息。在大多数国家这些法律是相似的，特别是欧洲和美国。你必须考虑明确的问题是，你的 ISP 可能明确地禁止在你的 IP 地址上运行蜜罐，或者由于无法预料敌手的步骤，可能危及其他机器的安全。如果你不确定你所做的，请咨询律师。也可以联系当地的蜜罐组织，他可以给你本国法律状况的概述。你可以在蜜罐项目的网站（http://www.honeynet.org/）上得到世界各地的不同蜜罐组织纵览。

## 1.3 商业工具

在详细讨论各种蜜罐解决方案之前，我们必须研究一些用于探测和扫描蜜罐的工具。本节中讨论的工具主要用于边界勘测和网络流量监测。攻击者往往想知道他们将面对什么样的目标，攻击一个主机之前，他们收集有关操作系统和运行其上的是什么服务等信息，搞清操作系统对了解主机可

能存在的漏洞是非常重要的，有关服务及其版本的信息允许敌手策划一条攻击路线。就我们而言，真正重要的是了解什么样的邪恶想法可能会试图入侵我们的蜜罐，这就是我们要详细介绍这些工具的原因。每个人的工具箱中都有一个称为 Nmap 的工具，它已经非常流行，甚至在影片《The Matrix Reloaded》中出现特写。

理解一些这样的工具的输出是相当困难的。如果你不熟悉 TCP，也没有阅读有关网络的部分，你可能想跳过 tcpdump 和 Ethereal 的讨论，回头再看。

### 1.3.1 tcpdump

tcpdump 是用来了解网络上正在发生的事情的主要工具，tcpdump 能够嗅探网络，并以可以理解的形式呈现这些数据。使用一种功能强大的过滤语言，tcpdump 可用于展现主机的流量和我们感兴趣的事件。该工具的官方网站是 http://www.tcpdump.org/，从那里可以下载它。大多数基于 UNIX 的系统内置有 tcpdump 的一个版本，也可以通过包管理系统简单地安装它。WinDump 是 Windows 版本的 tcpdump，可以从http://www.winpcap.org/windump/上得到。

为了捕获进出 10.0.1.91 的流量，运行以下命令：

```
# tcpdump -n -s 1500 host 10.0.1.91
```

图 1.2 显示了运行命令后得到的一些样本流量结果。首先，是一些 ARP 信息，用于把 IP 地址映射成 MAC 地址。接着，主机 10.0.1.91 和 10.0.1.6 通过 TCP 握手建立 TCP 连接。在通过 FIN 包终止连接之前发送少量数据包。为了充分地利用 tcpdump，需要很好地理解 TCP/IP。然而，仅是想断定一个蜜罐是否启动并接受连接，你不需要太多的技巧，读了本节之后就很容易做到。

本节简要地评述 tcpdump 最重要的参数和 pcap 过滤语言的一些特征。如果你想知道所有可能的命令，如标志，以及关于过滤语言的更多细节，tcpdump 提供一个实用手册。下面给出你可能最常用的命令标志。

- -i interface：选择 tcpdump 捕捉网络流量的网络接口，默认是你的主要以太网接口，如 en0，但有时你的机器可能有多于一个的网络接口，你可能会想监听不同接口。
- -n：关闭 DNS 解析，显示 IP 地址。如果你正在监视一个接受世界各地连接的繁忙的服务器，此标志特别有用。如果 tcpdump 需要为所有新的 IP 地址解析 DNS 名称，输出会大大延迟。此外，如果你没有连接到互联网，而是在本地网络上进行实验，这个选项就派上用场了。
- -s snaplen：默认地，tcpdump 只查看数据包的前 64 个字节。这使得处理速度非常快，但对一些协议，这些数据不足以完全解码数据包。设置捕捉长度为 1500，可以捕获和检查整个数据包。
- -X：当开启这个参数，tcpdump 将以十六进制和可打印 ASCII 字符显示数据包的内容。这有助于我们快速看清一个数据包的内容，看到正在传输哪类数据。
- -S：打印序号的绝对值。默认情况下，tcpdump 显示序号的差值，是从捕获的一个连接的第一个数据包开始计算的累积差值。该标志有时帮助关闭这个功能。

```
代码浏览：
reading from file /tmp/dump, link-type EN10MB (Ethernet)
19:34:35.54 arp who-has 10.0.1.91 tell 10.0.1.6
19:34:35.54 arp reply 10.0.1.91 is-at 00:90:27:a0:77:9b
19:34:39.40 IP 10.0.1.6.2809 > 10.0.1.91.25: S 317611736:317611736(0)
        win 5840 <mss 1460,sackOK,timestamp 111830 0,nop,wscale 0>
19:34:35.55 arp who-has 10.0.1.6 (ff:ff:ff:ff:ff:ff) tell 10.0.1.91
19:34:35.55 arp reply 10.0.1.6 is-at 00:09:5b:be:66:92
19:34:39.40 IP 10.0.1.91.25 > 10.0.1.6.2809: S 2544916314:2544916314(0)
        ack 317611737 win 5752
        <mss 1460,nop,nop,timestamp 213261 111830,nop,wscale 0>
19:34:39.40 IP 10.0.1.6.2809 > 10.0.1.91.25: . ack 1 win 5840
        <nop,nop,timestamp 111833 213261>
19:34:39.40 IP 10.0.1.91.25 > 10.0.1.6.2809: . 1:90(89) ack 1 win 5752
19:34:39.40 IP 10.0.1.6.2809 > 10.0.1.91.25: . ack 90 win 5840
        <nop,nop,timestamp 111836 213261>
19:34:42.39 IP 10.0.1.6.2809 > 10.0.1.91.25: P 1:17(16) ack 90 win 5840
        <nop,nop,timestamp 114826 213261>
19:34:42.39 IP 10.0.1.91.25 > 10.0.1.6.2809: . ack 17 win 5736
19:34:42.40 IP 10.0.1.91.25 > 10.0.1.6.2809: . 90:163(73) ack 17 win 5752
19:34:42.40 IP 10.0.1.6.2809 > 10.0.1.91.25: . ack 163 win 5840
        <nop,nop,timestamp 114830 213261>
19:34:44.63 IP 10.0.1.6.2809 > 10.0.1.91.25: P 17:23(6) ack 163 win 5840
        <nop,nop,timestamp 117066 213261>
19:34:44.63 IP 10.0.1.91.25 > 10.0.1.6.2809: . ack 23 win 5746
19:34:44.64 IP 10.0.1.91.25 > 10.0.1.6.2809: F 163:163(0) ack 23 win 5752
19:34:44.64 IP 10.0.1.6.2809 > 10.0.1.91.25: F 23:23(0) ack 164 win 5840
        <nop,nop,timestamp 117071 213261>
19:34:44.64 IP 10.0.1.91.25 > 10.0.1.6.2809: . ack 24 win 5752
```

图 1.2　捕获一个连接虚拟蜜罐 tcpdump 的输出样本，可以看到三次握手建立连接和交换 FIN 报文段终止连接

- -w filename：将捕获的数据包转存到指定文件中。如果你想日后彻底检查捕获的数据包，或者保存它们以备法庭鉴定，这个功能是非常有用的。如果你监控一个繁忙的网络，该文件尺寸会迅速增大。
- -r filename：读取先前保存的转储文件和打印网络数据，似乎它们是直接从网络上捕获的。从 stdin 读取数据，可以使用文件名“-”。

### 1.3.2　Wireshark

为了有一个更方便的界面，你也可以使用工具 Wireshark（以前称为 Ethereal）。Wireshark 像 tcpdump 一样提供给你相同数量的信息，但它有一个图形用户界面，更容易分析一个给定的数据包的踪迹。

Wireshark 的官方网站是 http://www.wireshark.org/，在那里可以下载许多用于不同平台的

Wireshark 工具，有二进制和源代码两种格式的 Windows 版本，有用于不同 Linux 的发行版，以及不同的 BSD 版本，此外还有用于 Mac OS X 的版本。

图 1.3 给出了一个 Wireshark 的屏幕截图。可以看到两台主机之间发送的 ICMP 数据包，在网络中捕获的其他数据包，每一个数据包在一个独立的部分显示，可以详细分析所有数据包。

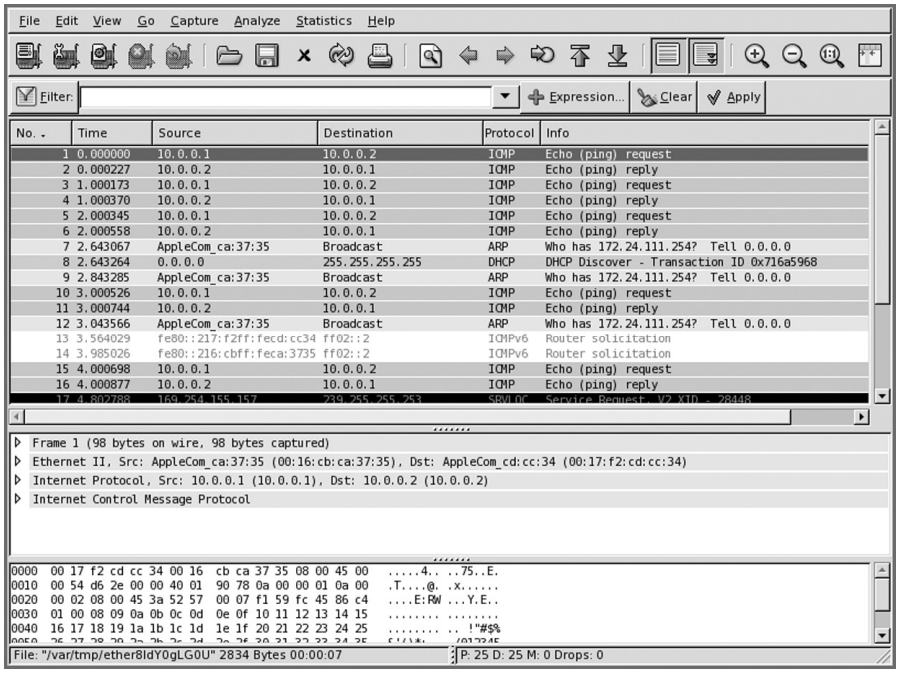


图 1.3　Wireshark 屏幕截图，一个网络嗅探器

### 1.3.3　Nmap

Nmap 的作者 Fyodor 把它定义为一个网络探测工具和安全扫描器，它确实是一个“操作系统指纹”的婉转说法。借助 Nmap，我们可以快速扫描互联网上的许多主机，识别出它们使用的操作系统和正在提供的服务。要确定在 192.168.1.1 上运行的操作系统和服务，可以 root 用户身份按如下命令调用 Nmap：

```
nmap -sS -O -F 192.168.1.1
```

图 1.4 中的输出告诉我们，192.168.1.6 上运行的是最近版本的 Linux 内核，Nmap 不能很清楚地告诉我们具体是哪一个版本，因此它给出 Linux 2.4.18 和 Linux 2.6.7 两个选择，版本号对应着主机上运行的内核，每个内核可能选择 TCP 的实现略有不同，而 Nmap 利用这些差异来确定主机上正在运行的操作系统。在这个例子中，被扫描的 Linux 主机的确运行的是 2.6.7 版本，有趣的是还看到其他的 TCP 结果，如 TCP 时间戳，允许 Nmap 确定机器的运行时间，在这个例子中，主机已经运行了将近一个小时。

```
代码浏览：
Starting nmap 4.11 ( www.insecure.org/nmap/ ) at 2007-01-16 17:45 PDT
Interesting ports on 192.0.2.1:
(The 1208 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
9/tcp open discard
13/tcp open daytime
21/tcp open ftp
22/tcp open ssh
23/tcp open telnet
25/tcp open smtp
37/tcp open time
79/tcp open finger
80/tcp open http
111/tcp open rpcbind
113/tcp open auth
139/tcp open netbios-ssn
445/tcp open microsoft-ds
MAC Address: 00:09:5B:AF:34:11 (Netgear)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.18 - 2.6.7
Uptime 0.033 days (since Tue Jan 16 16:57:58 2007)

Nmap finished: 1 IP address (1 host up) scanned in 3.883 seconds
```

图 1.4　针对 Linux 主机上运行多个服务的 Nmap 示例输出



可以从http://insecure.org/nmap/install/上获得 Nmap 的详细安装手册。该工具可用于所有主流平台，包括 Windows、Linux 和 BSD 等不同版本。可以以二进制形式安装，或编译其源代码。

Nmap 是一个十分复杂的工具，我们只给出一些最常用的命令行标志的简单介绍。

- -p port-ranges：指定扫描的端口。
- -sV: 启用版本检测，也就是说，在一个给定的端口上 Nmap 试图识别正运行的服务及其版本。
- -O：启用远程操作系统检测。
- -A：启用版本检测和操作系统检测。
- -T[0-5]：设置时间选项，一个较大的数字意味着两次探测之间较小的时间间隔。
- -oN/-oX/-oG file：指定输出格式为正常、XML 或 grepable 格式，把扫描报告写入文件，便于以后分析使用。

这 6 个命令行标志应足以应付 Nmap 的日常使用。这个工具对于核实蜜罐的启动和运行非常有用，也可以测定是否所有服务正在运行。

# 4

# Honeyd——基础篇

Honeyd 是一个框架——把数千个虚拟蜜罐及对应的网络集成到一起。通常，我们配置 Honeyd 集成现有网络上未分配的 IP 地址。对于每一个 IP 地址，我们可以告诉 Honeyd 我们希望如何模拟计算机的行为。例如，我们可以建立一个虚拟 Web 服务器，服务器看似运行 Linux 和监听 80 端口。我们可以在另一个 IP 地址上建立一个带有类似 Windows 网络栈的虚拟蜜罐，它上面的所有 TCP 端口似乎都正在运行服务。这将使我们接收到蠕虫或探测的初始 TCP 有效载荷。Honeyd 可用于在现有网络上建立一些诱饵，或者只使用一台计算机创建包含数百个网络和上千台主机的路由拓扑。本章详细介绍 Honeyd 如何工作、如何配置以及如何部署。

## 4.1 概述

你的第一个蜜罐将是一次激动人心的体验，你会好几个小时观察它的日志，等着你感兴趣的流量出现，或者等待远程攻击的探测。最后，有人破门而入。不幸的是，当只使用一个 IP 地址时，这可能需要一段时间。但是，也有其他方法，可以显著地提高你在互联网上的曝光度。显然，如果一个地址被探测和攻击需要花费很长时间，那么你同时观察 100 个或者可能 1000 个 IP 地址，则可能花费较少的时间就能观察到你感兴趣的活动。

这就是 Honeyd 发挥作用的地方，它是一个低交互虚拟蜜罐的框架，可以在一个网络上或者甚至整个 Internet 上创建数千个虚拟蜜罐，如图 4.1 所示。Honeyd 支持 IP 协议簇[92]，根据虚拟蜜罐上配置的服务，响应网络发送给蜜罐的网络请求。当发送一个响应数据包时，Honeyd 的个性引擎产生与所配置的操作系统个性相匹配的网络行为。它是一款遵循 GNU 通用公共许可证（GPL）的开源软件，可以在绝大多数操作系统上运行。

Honeyd 不仅可以充分利用未分配的网络地址，让你更多地观察互联网上的恶意活动，也可以用来阻止敌手攻击你的真实系统。一个很好的例子是一年一度的 Cyberdefense（网络防御）演习——一个美国军事院校与来自国家安全局（NSA）的红队之间的竞赛。每个学院有一个学生组成的小组，负责保护他们网络，而红队试图攻陷他们，或者造成各种类型的破坏。几年前，当 Honeyd

首次发布时，一些学生通过配置 Honeyd 创建几百个虚拟蜜罐以加固他们的网络，这些蜜罐打算用来阻碍敌手，不让其攻击真正的机器。这一策略取得了出乎意料的成功，军事院校的学生们饶有兴趣地看着 NSA 小组花费几个小时试图攻破实际上并不存在的机器。

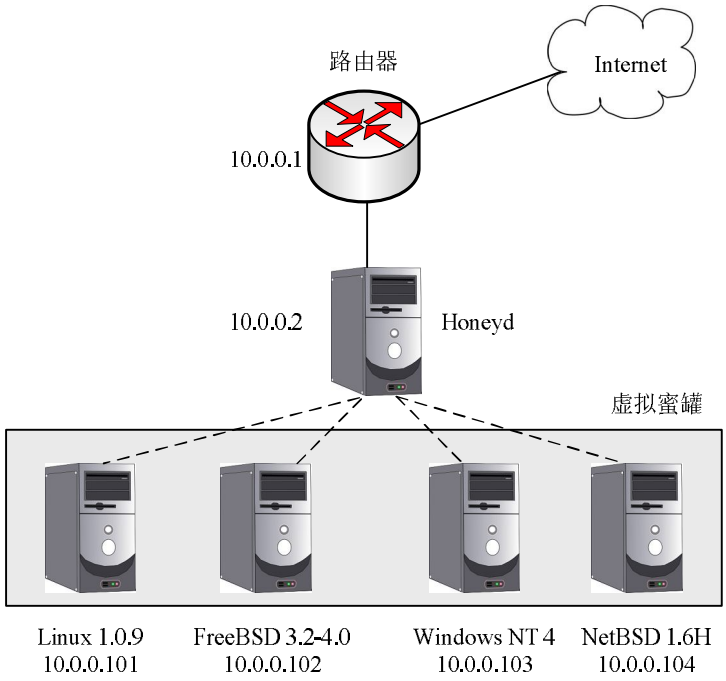




图 4.1　Honeyd 通过路由器或代理 ARP 为其虚拟蜜罐接收流量。对每一个蜜罐，Honeyd 可以模拟不同操作系统的网络栈行为

同样，我们可以利用 Honeyd 让互联网上的攻击者感到混乱和迷惑。本章介绍配置 Honeyd 的基本知识，接下来的章节将解释如何诱捕垃圾邮件发送者，如何建立自己的系统捕捉数百万的垃圾邮件，如何捕获蠕虫。请别走开！

### 4.1.1　特性

Honeyd 有很多有趣的特性。

- 同时模拟数千的虚拟主机：使用 Honeyd 的主要原因就是它能够同时创建数千个虚拟蜜罐。敌手可以通过网络与每一个单独的主机交互，体验每个主机不同的行为，这些行为依赖于它的配置。
- 通过配置文件配置任意服务：你可以提供与对手交互的任意程序，无论何时 Honeyd 收到一个新的网络连接，它将启动你为这一连接指定好的程序，回应攻击者。如果不运行程序，你也可以使用 Honeyd 作为代理将连接转到其他机器上，或者采用类似被动指纹识别的功能，识别远程主机和负载的随机采样。

- 在 TCP/IP 协议栈层面上模拟操作系统：这一特性允许 Honeyd 欺骗 Nmap 和 Xprobe，使它们相信一个虚拟蜜罐上正运行着各种配置的操作系统。为了进一步提高真实性，处理分片重组和 FIN 扫描机制策略也可以做调整。
- 模拟任意路由拓扑：路由拓扑结构可以任意复杂。可以配置迟延、丢包和带宽等特征。Honeyd 支持非对称路由、物理机器集成到一个虚拟拓扑中，以及通过 GRE 隧道的分布式操作。
- 子系统虚拟化：利用子系统，Honeyd 可以在一个蜜罐的虚拟命名空间下执行真正的 UNIX 应用，如 Web 服务器、FTP 服务器等。这一特征也允许在虚拟地址空间内进行动态端口绑定和网络连接的后台初始化。

### 4.1.2 安装和设置

在使用 Honeyd 进行实验和尝试它的各种功能之前，首先在你的计算机上安装它，我们希望你运行的是 Linux、Mac OS X 或 FreeBSD 之类的操作系统，因为 Windows[①]不能真正提供 UNIX 系统的灵活性。如果你运行是 Debian，那么以 root 用户身份使用下列命令就可以简单地安装 Honeyd。

```
apt-get install honeyd
```

另一方面，如果你喜欢冒险且不介意编译软件，你总是可以自己得到源代码，并编译最新和功能最强大的版本。下面的步骤将帮助你安装和运行该软件：

（1）确认你已安装了所有的从属软件，Honeyd 需要 libevent、libdnet 和 libpcap。可以从 www.monkey.org/~provos/libevent/上下载最新版本的 libevent，从 libdnet.sourceforge.net/上下载 libdnet，从 www.tcpdump.org/下载 libpcap。

（2）使用命令 tar -xzf <package>.tar.gz 解压源码包。

（3）对于每一个包，进入该包目录，执行./ configure、make 和 sudo make install 命令。

（4）从www.honeyd.org/release.php上查找并下载最新版本的 Honeyd。如果你已经安装了 gpg，也应该下载 Honeyd 软件包的数字签名，验证其完整性。

（5）使用命令 tar -xzf honeyd-<version>.tar.gz 解压 Honeyd 软件包。

（6）进入源目录配置该软件包，执行./configure。如果你没有安装 Python 开发库，配置脚本可能会失败。可以选择安装 Python 开发库，使你能够使用一些有趣的脚本功能和 Honeyd 的内部 Web 服务器，也可以决定跳过 Python 的功能执行：

```
./configure --without-python
```

如果仍然不能成功，可以查阅 www.honeyd.org/faq.php 常见问题解答。

（7）使用 make 命令编译二进制代码，然后使用 sudo make install 命令安装软件。如果你没有

[①] 喜欢冒险的 Mike Davis 将一个较早版本的 Honeyd 移植到了 Windows 上，可以从www.securityprofiling.com/honeyd/honeyd.shtml上下载它。然而，值得注意的是 Windows 的二进制文件不支持 UNIX 版本的许多高级特性。

安装 sudo，那么切换到 root 用户执行该命令。

现在应该安装好了二进制文件，在使用预备的配置文件尝试运行 Honeyd 之前，必须配置你的主机以禁止它转发 IP 数据包，在 Linux 上执行以下命令：

```
echo 0 > /proc/sys/net/ipv4/ip_forward
```

在 BSD 系统中，systctl 命令可用于关闭 IP 转发：

```
sysctl -w net.inet.ip.forwarding=0
```

开启 IP 转发功能，操作系统内核对其接收到的任何发给虚拟蜜罐的 IP 数据包都会尝试进行转发，这会导致大量的数据包重复，甚至数据包风暴。另一种不需要禁用 IP 转发的方法，是在 Honeyd 的主机上配置一个防火墙，阻止所有发送给蜜罐的数据包，Honeyd 仍然能够响应它们，但操作系统本身将忽略它们。

如果一切顺利，Honeyd 应该能启动并运行了，如图 4.2 所示。不过，在讨论有关配置这个守护进程之前，要想真正理解它是如何工作的，有必要先简要讨论下它的整体设计和局限性。

代码浏览：

```
$ sudo ./honeyd -d -f config.sample
Password:
Honeyd V1.0 Copyright (c) 2002-2004 Niels Provos
honeyd[8222]: started with -d -f config.sample
Warning: Impossible SI range in Class fingerprint "IBM OS/400 V4R2M0"
Warning: Impossible SI range in Class fingerprint "Microsoft Windows NT 4.0"
honeyd[8222]: listening promiscuously on fxp0: (arp or ip proto 47 or (udp
  and src port 67 and dst port 68) or (ip ))
honeyd[8222]: HTTP server listening on port 80
honeyd[8222]: HTTP server root at /usr/local/share/honeyd/webserver/htdocs
honeyd[8222]: Demoting process privileges to uid 32767, gid 32767
```

图 4.2　在示例配置文件上首次运行 Honeyd

## 4.2　设计概述

为了理解 Honeyd 如何工作以及如何使用它，首先需要了解它的基础设计。为了满足你对技术细节的需求，我们会在第 5 章讨论一些专门功能和应用细节，在这里只介绍一些核心功能。Honeyd 体系结构的基本概况如图 4.3 所示。尽管通过配置可以控制 Honeyd 的每一个方面，但是三个重要特征决定了 Honeyd 的整体行为：一是敌手只能从网络中与 Honeyd 交互；二是配置多少 IP 地址，Honeyd 就可以模拟多少；三是通过改变每个输出数据包与配置的操作系统特征相匹配，从而欺骗指纹识别工具。通过了解软件设计产生的局限性，对 Honeyd 是否是解决你的问题所需要的恰当工具，可以做出有根据的选择。下面详细介绍这些设计的每一个选择。

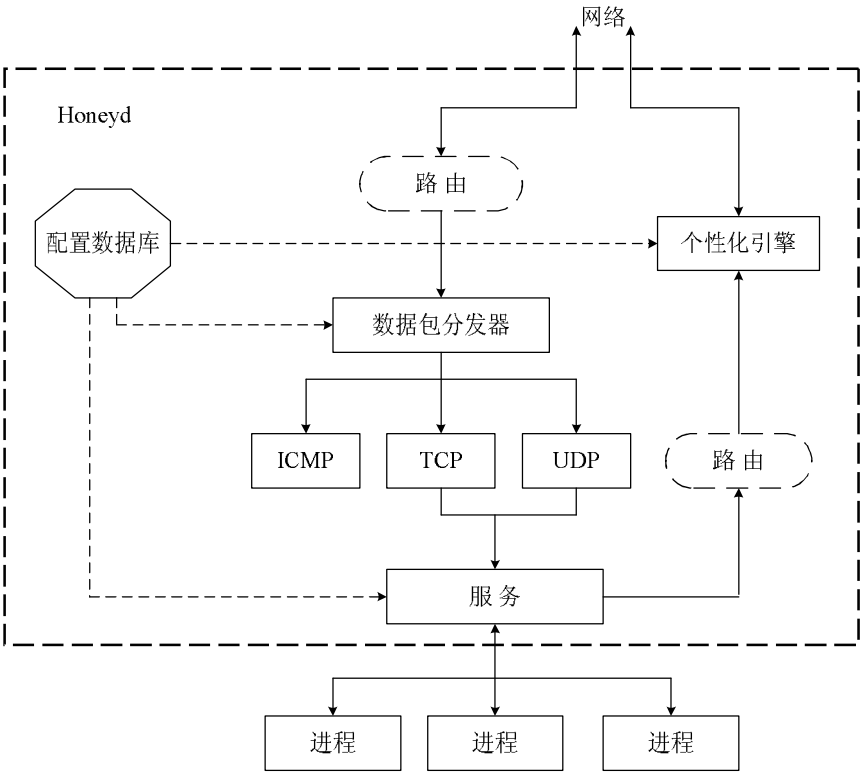


图 4.3 Honeyd 使用一个简单体系结构。一个中央包分发器接收所有感兴趣的网络流量，基于明确的配置，创建不同的服务进程处理流量，发送网络的数据包被个性引擎修改，以匹配所配置操作系统的特点



### 4.2.1 仅通过网络交互

我们的基本假设是，一个敌手只可以在网络层面上与我们的蜜罐交互，这意味着他或者她不能走近计算机并通过键盘登录，因为任何 Honeyd 模拟蜜罐没有与之对应的物理计算机。我们并不是模拟操作系统的每一个方面，而是选择只模拟其网络堆栈。这种方法的主要缺点是，一个敌手永远无法获得对一个完整系统的访问，即使他攻陷模拟的服务。另一方面，我们仍然能够捕捉连接和攻击的尝试。可以通过与虚拟机如 VMware[94]相结合，减轻 Honeyd 的这些缺点。我们将在第 7 章讨论如何将不同的蜜罐组成一个混合系统。现在，重要的是理解 Honeyd 是一个模拟 TCP 和 UDP 服务的低交互蜜罐，当然它也能正确地理解和响应 ICMP 消息。

### 4.2.2 多 IP 地址

作为一个功能强大且灵活的解决方案，Honeyd 可以同时处理在多 IP 地址上虚拟蜜罐，这允许在网络上植入大量虚拟蜜罐，模拟出不同的操作系统和服务。为了更贴近实际，Honeyd 也可以模拟任意的网络拓扑结构。为了模拟拓扑上分散的地址空间和负载均衡，Honeyd 支持网络隧道。

### 4.2.3 欺骗指纹识别工具

还记得我们在 1.3 节中讨论的指纹识别工具吗？为了向攻击者呈现更逼真的蜜罐，我们需要欺骗指纹识别工具，使它们报告我们所预期的操作系统。Honeyd 通过反转指纹识别工具的数据库来实现这个，当一个蜜罐需要发送一个网络数据包时，Honeyd 会修改数据包，匹配对应数据库中配置的操作系统指纹（见 5.1 节）。

## 4.3 接收网络数据

图 4.4 给出了 Honeyd 处理网络数据包的概念性描述。一个中央计算机拦截发送给配置为蜜罐的 IP 地址的网络流量，模拟它们的响应。在描述 Honeyd 的内部运作之前，我们需要配置网络，使发送给虚拟蜜罐的数据包到达 Honeyd 主机。对于任何需要快速复习网络知识的人，第 1 章解释和描述了最重要的网络协议。

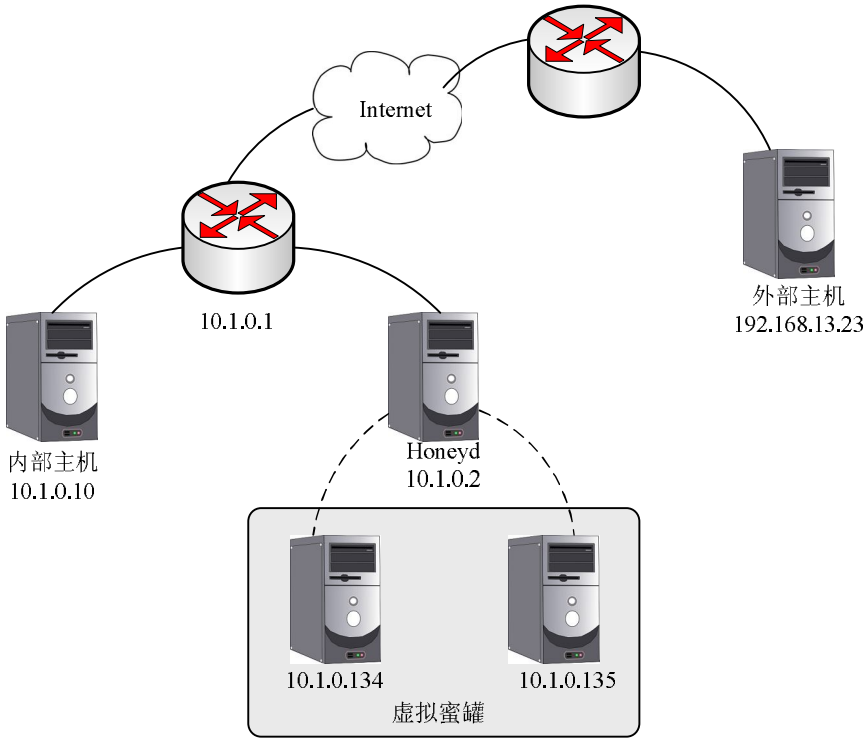


图 4.4 一个内部网络上的主机通过本地网络直接到达虚拟蜜罐，它需要通过一个 ARP 请求把 IP 地址映射为 MAC 地址，然后将数据包发送到虚拟蜜罐。外部主机不需要知道 MAC 地址，因为路由器将发送 ARP 请求

Honeyd 应答那些目的 IP 地址属于一个模拟蜜罐的网络数据包。然而，由于缺乏适当的网络配

置，Honeyd 将永远也看不到任何对应答的响应数据包。我们应该把网络配置成这种形式——Honeyd 接收发送给其模拟的 IP 地址的数据包。有几种方式可以做到这一点，例如，可以为虚拟 IP 地址建立特殊的路由，把它们指向 Honeyd 主机，也可以使用代理 ARP[7]，或者使用网络隧道。

假设路由器的 IP 地址是 10.1.0.1，而 Honeyd 主机地址是 10.1.0.2，在这个简单的例子中，虚拟蜜罐的 IP 地址属于本地网络，假设本地网络地址是 10.1.0/24，则虚拟蜜罐将有形如 10.1.0.x 的地址，其中 x 是介于 3 和 254 之间的数字，例如 10.1.0.134，如图 4.4 所示。

当攻击者从互联网上发送一个数据包到蜜罐 10.1.0.x 时，路由器接收到该数据包，并尝试转发它。路由器查询其路由表，找到转发地址为 10.1.0.x。由于蜜罐位于本地网络范围之内，路由器可以直接访问它。在更复杂的情况下，还有其他的结果：因为没有到达该蜜罐的路由，路由器可能丢弃该数据包，或者路由器转发该数据包到另一个路由器。

不过，当路由器试图将数据包转发到蜜罐时，它需要把蜜罐的 IP 地址翻译成链路层或 MAC 地址。地址解析协议（ARP）负责这一翻译过程，对于未分配的 IP 地址，ARP 请求一直得不到回复，最终路由器丢弃该数据包，因为目的主机看起来已经停用。为了让路由器把数据包转发到 Honeyd 机器，我们仿造了回复，并返回主机 10.1.0.2 的 MAC 地址。建立这样一个重定向有几种方法，一个简单的方法是使用 arp 命令，比方说 10.1.0.2 的 MAC 地址是 1A:1A:60:F0:19:07，下面的命令将目的地址为 10.1.0.34 的数据包重定向到 Honeyd 主机：

```
/usr/sbin/arp -s 10.1.0.134 1A:1A:60:F0:19:07
```

如果我们有路由器 10.1.0.1 的管理控制权，可以为虚拟蜜罐插入一条静态路由，指向 Honeyd 的主机 10.1.0.2，而不使用代理 ARP。例如，如果路由器运行的是 NetBSD，下面的命令会实现这个重定向：

```
route -n add -net 10.1.0.132/30 10.1.0.2
```

当路由器接收到从 10.1.0.132 到 10.1.0.135 范围内的任何 IP 地址的数据包，它把完整的数据包转发给 Honeyd 主机，而无需 ARP 重定向。在更复杂的环境下，有可能通过隧道把网络地址空间聚集到 Honeyd 主机，Honeyd 支持通用路由封装（GRE）[34,35]隧道协议，在 5.6 节中会详细介绍。

## 4.4 运行时标志

虽然我们刚才看到运行 Honeyd 相当简单，但它是一个极其复杂的工具，可以把它调整做几乎任何你想得到的与网络相关的任务。虽然大部分复杂性在于其配置（4.5 节中描述），但是 Honeyd 还有一些能影响其行为的命令行参数。

如果已经在你的系统上正确安装了 Honeyd，可以执行下面的命令得到命令行参数的概况与描述：

```
man honeyd
```

当阅读帮助页面时，第一段如图 4.5 所示，可能看起来相当不清晰。这是所有的命令行标志和相应参数的概括，参数的名称说明了 Honeyd 期望的变量类型。

```
honeyd [-dP] [-l logfile] [-s servicelog] [-p fingerprints] [-0 p0f-file]
  [-x xprobe] [-a assoc] [-f file] [-i interface] [-u uid] [-g gid]
  [-c host:port:username:password] [--webserver-port port]
  [--webserver-root path] [--rrdtool-path path]
  [--disable-webserver] [--disable-update]
  [--fix-webserver-permissions] [-V|--version] [-h|--help]
  [--include-dir] [net ...]
```

图 4.5　Honeyd 支持的运行时标志

下面详细解释这些参数。介绍的顺序与帮助列出的有些差别，首先介绍比较重要的参数，后面再介绍较少使用到的参数。

- -f configfile：可能是最重要的标志，它告诉 Honeyd 在哪儿可以找到它的配置文件。该配置文件包含关于你的虚拟蜜罐的所有信息，以及它们能为网络提供的服务，更多信息参见 4.5 节。
- -i interface：默认情况下，Honeyd 使用第一个网络接口监听入站流量。不过，如果你的 Honeyd 机器有多个接口，你需要使用命令行手工指定接口，接收发向你的虚拟蜜罐的流量。例如，假设你的主机有三个接口：eth0 在 192.168.1.0/24 中，eth1 在 10.1.0.0/24 中，eth2 在 10.2.0.0/24 中。如果你想在后两个网络上创建虚拟蜜罐，需要在命令行中指定-i eth1 -i eth2。
- -d：这个标志使 Honeyd 运行在调试模式，所有的状态信息都被显示到当前终端上。这对于测试配置文件和了解 Honeyd 如何工作是有帮助的。一旦一切都似乎运行正常后，应该忽略这个标志，让 Honeyd 作为一个守护进程在后台运行。即使没有-d，仍然可以通过 syslog 获得日志消息，通常可以从/var/log/messages 中读取。
- -l logfile：启用此标志使 Honeyd 把数据包级别的日志写入到指定的日志文件中，重要的是，这个文件和相关目录对于 Honeyd 是可写的。Honeyd 通常使用用户 ID 为 nobody 运行，所以确认对于该用户，该目录对用户或用户组是可写的，更多信息参见 4.8 节。此标志默认是关闭的。
- -s servicelog：类似数据包级别日志，启用这一标志使 Honeyd 记录模拟服务提供的信息，这是一个服务脚本写到 stderr 的所有数据，更多信息参见 4.8 节。此标志默认是关闭的。
- -p fingerprints：Nmap 指纹数据文件的路径名。如果使用 Honeyd 的默认安装目录，这个文件会在/usr/local/share/Honeyd/nmap.print 上。在这种情况下，不需要使用此标志。另一方面，如果你刚刚安装了 Nmap 的一个新版本，你可能想使用该文件的最新版本，可以使用- p 标志告诉 Honeyd 它的位置。
- -0 p0f-file：被动指纹识别数据库的路径名。这个数据库可以让 Honeyd 识别远程主机的操作系统，这对以后的分析是相当有用的。如果已经正确安装 Honeyd，则不需要指定此标志。

- -x xprob：Xprobe 指纹数据库的路径名。它允许 Honeyd 向 ICMP 指纹识别工具返回正确的 ICMP 应答。
- -a assoc：Xprobe 和 Nmap 的指纹联合数据库的路径名。此文件用于联合两种指纹数据库的好处，对于一个给定的虚拟蜜罐，该联合文件使 Honeyd 能向指纹识别工具返回正确的 TCP 和 ICMP 应答。

## 4.5 配置

在任何事情发生之前，需要正确配置 Honeyd。Honeyd 使用一个简单的基于文本的配置文件，指定在哪些 IP 地址上运行虚拟蜜罐，以及指定每个主机可用的服务。配置语言是上下文无关语法，可以用 BNF（巴科斯－诺尔范式）描述，如图 4.6 所示。对于每一个配置命令，完整的语法以紧凑形式描述。如果对一个命令的具体选项有疑问，可以把图 4.6 作为一个快速参考表。在详细解释所有的命令之前，首先对主要的配置命令做一个简要概述。

代码浏览：

```
config = creation | addition | delete | binding | set |
         annotate | route [config] | option
creation= "create" template-name | "create" "default" |
   "dynamic" template-name
addition= "add" template-name proto "port" port-number action |
   "add" template-name "subsystem" cmd-string ["shared"] ["restart"] |
   "add" template-name "use" template-name "if" condition
delete= "delete" template-name |
   "delete" template-name proto "port" port-number
binding = "bind" ip-address template-name |
   "bind" condition ip-address template-name |
   "bind" ip-address "to" interface-name |
   "dhcp" template-name "on" interface-name ["ethernet" cmd-string] |
   "clone" template-name template-name
set = "set" template-name "default" proto "action" action |
   "set" template-name "personality" personality-name |
   "set" template-name "personality" "random" |
   "set" template-name "ethernet" cmd-string |
   "set" template-name "uptime" seconds |
   "set" template-name "droprate" "in" percent |
   "set" <template-name> "maxfds" <number> |
   "set" template-name "uid" number ["gid" number] |
   "set" ip-address "uptime" seconds
annotate= "annotate" personality-name [no] finscan |
   "annotate" personality-name "fragment" ("drop" | "old" | "new")
route = "route" "entry" ipaddr |
   "route" "entry" ipaddr "network" ipnetwork |
```

```
    "route" ipaddr "link" ipnetwork |
    "route" ipaddr "unreach" ipnetwork |
    "route" ipaddr "add" "net" ipnetwork \\
                "tunnel" ipaddr(src) ipaddr(dst) |
    "route" ipaddr "add" "net" ipnetwork ipaddr \\
                ["latency" number"ms"] ["loss" percent] \\
                ["bandwidth" number["Mbps"|"Kbps"] \\
                ["drop" "between" number "ms" "-" number "ms" ]
proto = "tcp" | "udp" | "icmp"
action = ["tarpit"] ("block" | "open" | "reset" | cmd-string | \\
    "internal" cmd-string \\
    "proxy" ipaddr":"port )
condition = "source os =" cmd-string |
    "source ip =" ipaddr | "source ip =" ipnetwork |
    "time " timecondition
```

图 4.6　Honeyd 配置语言规范。虽然起初你可能不知所措，但是它提供了针对所有 Honeyd 配置选项的一个有用参考。将在本节和下一章中详细解释配置语言中的每一部分

### 4.5.1　create

模板是指一个完全配置的计算机系统，创建虚拟蜜罐的第一步是为每个不同的计算机系统配置一个模板。随后，一个模板可以分配到一个 IP 地址，在该地址上建立一个虚拟蜜罐。虚拟蜜罐具有所指定模板的所有特征，包括操作系统的行为和每个端口应该运行的服务。

使用 create 命令创建新模板，create 可能的参数使用 BNF 描述：

```
creation:="create" <template-name>|"create default"|"dynamic" <template-name>
```

该语法支持三种不同的选项，我们只介绍前两种选项的用途，第 5 章讨论第三个选项。在第一种选项下，需要指定一个任意的模板名，每个模板的命名应该是不同的，因为我们必须通过它们的名字唯一地识别它们。如果你尝试两次创建相同的模板名称，Honeyd 将无法加载配置并输出一个信息。一个模板名称对应一个特定模板，该模板可以做进一步配置，如增加其他服务，最终将它绑定到一个 IP 地址创建一个蜜罐。

第二种选项更有趣，因为它引用默认模板。当 Honeyd 接收到一个 IP 地址的数据包，它会尝试查找一个名称与目标 IP 地址一致的模板。例如，当 Honeyd 接收到一个目标地址为 10.1.0.135 的数据包，它会寻找一个具有相同名称（10.1.0.135）的模板，如果找不到这个模板，Honeyd 会使用默认模板。

使用默认模板，很容易快速装配地址空间。例如，如果我们想为在一个 C 类网络中所有可用的地址指定一个相同的配置，我们只需创建和配置默认模板，没有必要为单个 IP 地址分配任何模板。

### 4.5.2　set

set 和 add 命令用于改变一个模板的配置。set 命令从 Nmap 指纹文件中选取一个特征赋给一个

模板，特征决定网络栈的行为，可以选自于多种流行操作系统，后面会讨论一个节选的流行操作系统列表。set 命令也可以定义所支持的网络协议的默认行为。set 的可能参数如下：

```
set ::= "set" <template-name> "default" <proto> "action" <action> |
        "set" <template-name> "personality" <personality-name> |
        "set" <template-name> "personality" "random" |
        "set" <template-name> "ethernet" <cmd-string> |
        "set" <template-name> "uptime" <seconds> |
        "set" <template-name> "droprate in" <percent> |
        "set" <template-name> "uid" <number> ["gid" <number>]
```

一个模板定义的默认行为决定了主机对未分配端口或协议的反应，协议可能是 TCP、UDP 或 ICMP，行为决定了主机对每个协议的默认反应，行为可以是 block（阻塞）、reset（重置）或 open（打开）。

- open：指定所有端口在默认状态是打开的。这个设置只影响 UDP 或 TCP 包，例如，可用于捕获蠕虫攻击的第一个有效载荷。
- block：意味着默认丢掉所有指定协议的数据包。当设置 block 时，蜜罐不响应该协议或端口的数据包，这一特性可以用来模拟防火墙的行为。
- reset：指定默认状态下关闭所有端口。如果一个 TCP 端口被关闭，对于发给该端口的 SYN 包，虚拟蜜罐应答 TCP RST；如果一个 UDP 端口被关闭，虚拟蜜罐使用 ICMP 端口不可达响应。

有了这些信息，现在可以轻松地创建一个使用如图 4.7 所示配置的不可见蜜罐。

```
create invisible
set invisible default tcp action block
set invisible default udp action block
set invisible default icmp action block
```

图 4.7　一个不响应任何网络流量的不可见蜜罐的配置

一个不可见蜜罐看似没有任何意义，因为它的目的就是通过网络不可达。另一方面，如果你回想一下默认模板的作用——只有当找不到其他模板的时 Honeyd 才使用它，一个不可见主机可以选择禁止默认模板。例如，假设我们要利用蜜罐装配一个 C 类网络，但该网络上已有两个生产系统，我们可以给它们分配不可见模板，对于这两个 IP 地址禁用 Honeyd。

大概 set 命令最重要的作用就是为模板设置网络特征，特征决定了 Honeyd 如何产生响应包。其他方面，特征还影响 TCP 序列号产生、IP 标识码改变，以及 TCP 时间戳增长速度。

可用的特征依赖于 Nmap 指纹数据库的版本。如果在/usr/local/下安装了 Honeyd，可以在/usr/local/share/honeyd/nmap/prints 下找到该数据库，否则，需要根据安装目录对应地改变路径。也可以使用 Nmap 提供的指纹数据库。使用下面的 shell 脚本产生一个可用的特征名列表：

```
grep "^Fingerprint" /usr/local/share/honeyd/nmap.prints | cut -f2- -d" | sort -u
```

在写这本书的时候，Nmap 已经有大约一千个不同的指纹了，图 4.8 给出了一个包含一些常用特征名的列表。可以使用数据库中的任何一个特征名，分配给一个操作系统的网络堆栈，并赋值到模板中，例如：

```
create linux
set linux personality "Linux 2.6.6"
```

| | |
|---|---|
| FreeBSD 4.6 | Microsoft Windows NT 4.0 SP3 |
| FreeBSD 5.0-RELEASE | Microsoft Windows XP Pro |
| Linux 2.4.20 | Microsoft Windows XP SP1 |
| Linux 2.6.6 | Microsoft Windows XP SP2 |
| Linux 2.6.7 | NetBSE 1.6 |

图 4.8　Nmap 指纹数据库中流行特征名节选

如果你不能确定一个名字，可以让 Honeyd 通过随机指定模板名随机地为你选择一个模板。这通常不是一个好办法，因为 Nmap 数据库包含很多由于毫无结果的网络行为而产生的不实用的指纹。例如，如果你打算使用“Apple Newton MessagePad 2100,Newton OS 2.1”作为特征，则无法建立 TCP 连接，因为即使试图建立一个连接时，这个特征也总是返回一个 TCP RST。

set 命令有很多选项，这里我们只讨论 uptime 和 Ethernet 选项。其他选项参见 5.1 节，那里将讨论高级配置选项。一个主机的 uptime 指从它启动到现在经历的时间。依赖于操作系统，它可能是由 TCP 时间戳选项确定的[55]。通常，系统启动时，它被初始化为 0，如果你知道时间戳的更新频率，可以猜出系统运行了多长时间。Netcraft 使用这个信息来测算出一个 Web 服务器的运行时间，参见http://uptime.netcraft.com/，例如，Honeyd 可以使用 set 命令初始化时间戳，模拟这种行为：

```
set linux uptime 259200 # three days
```

对于不支持时间戳选项或者随机初始化时间戳的操作系统，uptime 选项不起作用。

Ethernet 选项显式地将以太网 MAC 地址分配给一个模板。回想一下，向一台主机发送一个数据包必须知道该主机的 MAC 地址，前面提到，我们使用代理 ARP 把 Honeyd 主机的 MAC 地址发送给路由器或者局域网中的其他主机。不使用代理 ARP 的原因是：局域网中任何人可以很容易地查明所有虚拟蜜罐都指向同一个 MAC 地址，如果我们想欺骗访问本地网络的入侵者，这可能是一个彻底的泄漏。使用 set 命令的 Ethernet 选项，我们可以为每个虚拟蜜罐分配独立的 MAC 地址，Honeyd 处理所有 ARP 交互请求，这样，我们就不再需要设置代理 ARP 了。可以显式地指定 MAC 地址，例如：

```
set 10.1.0.134 ethernet "3f:12:4e:14:d0:32"
```

另一方面，Honeyd 通过指定设备生产商名也支持速记，如果我们想模拟一个 Cisco 路由，我们可能想把以太网地址设置成“cisco”，之后 Honeyd 在为 Cisco 预留的地址空间中随机产生一个 MAC 地址。

当一个有关联 IP 的模板被复制或绑定到一个新的 IP 地址时，Honeyd 为作为结果的 IP 地址或模板自动创建一个新的以太网地址，新的以太网地址的后三个字节随机产生。

**注意**

为了使一个网络正常运行，非常重要的一点是，所有 MAC 地址都必须不同。如果 MAC 地址冲突，会发生糟糕的事情。使用这个选项要求你确信没有冲突发生。

### 4.5.3 add

add 命令是任何模板应用的核心，它允许我们指定可远程访问的服务及每个端口上运行的应用程序。该命令的语法如下：

```
addition ::= "add" template-name proto "port" port-number action |
             "add" template-name "subsystem" cmd-string ["shared"] |
             "add" template-name "use" template-name "if" condition
```

add 命令通常针对我们想要配置的模板，如果模板不存在，需要通过 create 命令初始化模板。add 命令的 BNF 描述给出了三个不同的版本，后两个（指定子系统和配置动态模板）的作用在这里不做解释，在第 5 章介绍。

add 命令的第一个版本是最常见的，要求我们为每个服务指定协议、端口号和执行命令。例如，可以通过下面的命令在 22 端口上为 TCP 连接启动一个 SSH 模拟器。

```
add linux proto tcp port 22 "./scripts/ssh-emul.py"
```

当一个远程主机与带有 Linux 模板的 22 端口建立 TCP 连接时，Honeyd 启动一个新的进程执行服务脚本./scripts/ssh-emul.py。该脚本通过它的 stdin 接收网络输入，反过来，脚本的 stdout 通过网络发送到远程主机。除了配置 TCP 服务，add 命令也可用于运行在 UDP 上的服务，例如，一个模拟域名服务（DNS）的脚本。对于每个新建连接，Honeyd 需要创建一个新进程，如果 Honeyd 模拟的虚拟蜜罐收到大量网络流量，这可能导致性能瓶颈。

你可能已经注意到，add 命令语法定义了一个以 action 命名的标志。在前面的例子中，我们已经使用 UNIX shell 脚本路径取代了这个标志，对于指定端口上的新连接执行该脚本。图 4.6 详细展现了 action 标志的完整语法，它更为复杂：

```
action ::= ["tarpit"] ("block" | "open" | "reset" | cmd-string |
                       "internal" cmd-string |
                       "proxy" ipaddr":"port )
```

通过研究上面的 BNF，我们看到除了指定 action 为一个 UNIX shell 脚本，Honeyd 还能够识别我们先前已经使用的用于模板默认行为的标志。现在我们看到，可以基于每个端口使用 block、reset 和 open。例如，一个模板的默认行为可能接受所有端口上的 TCP 连接，但是需要关闭 Nmap 的操作系统检测所使用到的一些端口。可以使用下面的命令模拟一个关闭端口，该端口使用一个 TCP RST 报文段拒绝 TCP 连接：

```
add linux proto tcp port 23 reset
```

关键字 internal 告诉 Honeyd 加载一个 Python 模块，该模块对于每个连接无需创建新进程就可以在 Honeyd 内部执行。当一个虚拟蜜罐需要处理大流量时，创建新进程可能导致性能瓶颈。想了解更多有关内部 Python 服务的信息，请参见 5.4 节。

另一个有趣的特征是 proxy 关键字，它允许我们将网络连接转发到另一个不同的主机。例如，我们可能希望有一个域名服务器作为虚拟蜜罐，但又不想实际模拟一个域名服务器，这时，我们只需配置模板转发 UDP 数据包或 TCP 连接到一个确实运行域名服务器的主机。假设该域名服务器的 IP 地址为 10.0.0.2，该模板的形式如下：

```
add linux proto tcp port 53 proxy 10.0.0.2:53
add linux proto udp port 53 proxy 10.0.0.2:53
```

为了提供更多的动态行为，Honeyd 可以在服务参数上做变量扩展。对于服务和代理表达式，可以扩展以下四个变量：$ ipsrc、$ ipdst、$sport 和$ dport。变量扩展使得服务的行为适应它正处理的具体网络连接，它也可以重定向网络探测回送到正在进行探测的主机。例如，我们可能已经注意到有人通过 SSH（Evil laugh）攻击我们网络上的主机。只是为了好玩，我们决定把攻击发送回给敌手：

```
add linux proto tcp port 22 proxy $ip_src:22
```

在这种情况下，敌手建立一个到虚拟蜜罐 22 号端口的连接，而 Honeyd 又连接到敌手正在实施攻击机器上运行的 SSH 服务，任何发送到蜜罐的命令直接返回给她。

当然，重定向攻击返回给源主机只是一个游戏，而在实际中没有用。然而，对于常规服务脚本，这一灵活性是有用的，正如我们可以把 IP 源地址和源端口号作为命令参数。在一个脚本中，通过下面的环境变量这些信息也是可用的：HONEYD_IP_SRC，HONEYD_IP_DST，HONEYD_DST_PORT，HONEYD_SRC_PORT 和 HONEYD_PERSONALITY。在第 5 章将详细讨论如何创建脚本模拟互联网服务。



### 4.5.4 bind

完成一个虚拟蜜罐配置需要的最后一个命令是 bind 命令，它用于给一个 IP 地址分配模板。如果一个数据包到达一个 IP 地址，该 IP 地址还没有分配到模板，Honeyd 使用默认模板代替。bind 命令完整的语法如下：

```
binding ::= "bind" ip-address template-name |
            "bind" ip-address "to" interface-name |
            "bind" condition ip-address template-name |
            "dhcp" template-name "on" interface-name
              ["ethernet" cmd-string] |
            "clone" template-name template-name
```

第一个命令的语义非常直观，它告诉 Honeyd 对一个给定 IP 地址使用一个特定的模板。每当

Honeyd 接收到一个发给该 IP 地址的数据包，守护进程会参考模板配置，以决定是否一个端口是打开的，以及当连接建立时启动什么服务。其他配置更复杂：把一个 IP 地址绑定到一个接口上，允许 Honeyd 把真实机器集成到一个虚拟路由拓扑中，而且有条件的绑定可以作为一个速写用于配置动态模板，这些额外的选项会在第 5 章中详细描述。

当 Honeyd 为一个 IP 地址绑定一个模板时，本质上它是创建了一个名字为该 IP 地址的新的模板，并把原始模板的配置复制到对应该 IP 地址的模板中。这有很多优点：可以在该 IP 地址上使用所有的配置命令而不用改变模板，或者可以改变这个模板而无需改变以前配置的 IP 地址的配置。

clone 命令是 bind 命令更一般的形式，无需为一个 IP 地址复制一个模板配置，而可以使用 clone 复制一个模板配置给一个新的模板名称。例如，我们可以创建一个基础模板，指定所有协议的默认行为，然后从它克隆新的模板。由此产生的配置文件比较小，因为我们不用一遍又一遍重复协议的默认值。

dhcp 命令用于创建通过 DHCP 自动分配 IP 地址的虚拟蜜罐，这对于整合虚拟蜜罐到现有的生产网络中是非常有用。dhcp 命令有两个参数：模板的名称和发送 DHCP 请求的接口。在启动时，如果在本地网络中有一个 DHCP 服务器监听，Honeyd 自动地为每个 dhcp 语句自动索取 IP 地址。dhcp 命令只能用于分配有以太网地址的模板，当一个虚拟蜜罐等待来自 DHCP 服务器的应答时，它被分配一个 169.254.1.1 到 169.254.255.255 范围内的私有 IP 地址，这就把一个 Honeyd 实例限制到大约 65000 个虚拟蜜罐，可以通过 DHCP 获取地址。在实际中，任何 DHCP 服务器都不可能会分发那么多的租约。可选的 Ethernet 变量可以用于防止分配给虚拟蜜罐一个随机的 MAC 地址。



### 4.5.5 delete

delete 命令可以用来重新配置运行中的虚拟蜜罐，它的语法如下：

```
delete ::= "delete" <template-name>|\\
           "delete" <template-name> <proto> "port" <port-number>
```

该命令可用于完全删除一个模板，或者从模板中删除特定的服务。当一个模板被删除后，Honeyd 就不再知道它了，例如，如果你删除了名称为 10.1.0.1 的模板，试图到该 IP 地址的后续连接可以由默认模板处理。现存的到一个模板的连接不受影响，即使它被删除，仍能保持到一个服务的连接。

当试图建立一个更动态的虚拟蜜罐环境时，此命令最有用。使用 honeydctl 应用程序，重新配置正在运行中的 Honeyd 是可能的，参见 5.8 节。一个实例方案是一个 Bait 和 Switch 网络[72]，在那里我们有一个影子蜜网，复制了一个现有的生产网络。当入侵检测系统检测到一个敌手，可疑流量被重定向到影子蜜网。借助 honeydctl，通过创建一个工具，经常地扫描生产网络，并动态地复制拓扑和主机，能够实现影子网络。当生产网络中的一个主机从网络上断掉，我们可以删除相应的模板，当它恢复网络连接时，我们可以重新为它创建一个虚拟蜜罐。

### 4.5.6 include

include 命令不影响蜜罐的配置，但它可以用于把一个复杂的 Honeyd 配置组织成几个文件。include 命令带一个单一文件名作为参数，本质上是使用该文件内容替代 include 命令。includes 允许多达 10 级嵌套。

## 4.6 Honeyd 实验

在接下来的两节中，我们将探讨建立 Honeyd 两种常用的方案。第一种是描述如何本地运行 Honeyd，而不需要网络。第二种展示了一种简单的方法，利用 Honeyd 把虚拟蜜罐集成到现有生产网络中。本地设置非常适合对 Honeyd 的首次探索，因为失误不会导致对你的网络灾难性的破坏。通过本地接口，除了那些与以太网级模拟相关的功能，几乎所有的功能都可用。

### 4.6.1 本地 Honeyd 实验

图 4.9 显示了在私有 IP 网络 10.1.0.0/24 上包括两个虚拟蜜罐的一个完整的 Honeyd 配置。配置指定了通过网络可以访问的两个模板，分别对应 IP 地址 10.1.0.1 和 10.1.0.134。请记住，在 Honeyd 上下文中，一个模板是一个完整的蜜罐配置，可以被配置成在多个 IP 地址上可用。在一个 IP 地址上建立一个模板的过程，在 Honeyd 术语中称为 binding（绑定）。我们有一个用于路由器的模板，模拟了 Cisco 7206 路由器的网络堆栈，它只能通过 telnet 访问。另一个模板指定一个运行两个服务的蜜罐：一个简单的 Web 服务器和一个 SSH 连接的转发器，在这个例子中，转发器重定向 SSH 连接到连接发起者。仅当你的本地网络不包括在 10/8 地址范围内时，这个例子才可以在你的方案中工作。如果是在上述网络中，只需用 192.168 或 127.0 替换所有的 10.1。

```
create routerone
set routerone personality "Cisco 7206 running IOS 11.1(24)"
set routerone default tcp action reset
add routerone tcp port 23 "scripts/router-telnet.pl"

create netbsd
set netbsd personality "NetBSD 1.5.2 running on a Commodore Amiga
(68040 processor)"
set netbsd default tcp action reset
add netbsd tcp port 22 proxy \$ipsrc:22
add netbsd tcp port 80 "scripts/web.sh"

bind 10.1.0.1 routerone
bind 10.1.0.134 netbsd
```

图 4.9 Honeyd 一个示例配置。配置语言是上下文无关语法，这个例子定义了两个模板：一个可以通过 telnet 访问的路由器和一个运行 Web 服务器的主机

让我们在回环接口上用这个配置实例启动 Honeyd。在与虚拟蜜罐交互之前，必须执行以下几个步骤：

（1）必须确保到网络 10.0.0.0/8 的路由指向回环接口。在 BSD 上，通过以下命令完成这一工作：

```
route -n add -net 10.0.0.0/8 127.0.0.1
```

在 Linux 上，需要输入：

```
route -n add -net 10.0.0.0/8 gw 127.0.0.1
```

当然，这需要 root 权限，sudo 命令在这里会对你有所帮助。

（2）启动 Honeyd，在回环接口上监听 10/8 网络。不同操作系统之间的主要区别是回环接口的名称，BSD 系统可以有多个回环接口，127.0.0.1 通常分配给 lo0，而 Linux 只支持一个称为 lo 的回环接口。假设读者根据她的操作系统替换了正确的名称，则使用下面的命令启动 Honeyd：

```
honeyd -d -i lo -f config.book 10.0.0.0/8
```

（3）向一个虚拟蜜罐发送 ping 命令，以验证 Honeyd 正在运行，而且接收到了我们的网络流量：

```
ping -n -c1 10.1.0.1
```

如果一切工作正确，应该看到如下信息：

```
PING 10.1.0.1 (10.1.0.1): 56 data bytes
64 bytes from 10.1.0.1: icmp_seq=0 ttl=64 time=1.294 ms
--- 10.1.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 1.294/1.294/1.294/0.000 ms
```

否则，核对 Honeyd 调试日志的输出，看看 Honeyd 是否收到了 ping 命令的 ICMP 数据包。通常，可能是路由表设置不正确。

如果你做完了这些步骤，现在应该能够与虚拟蜜罐交互了。例如，尝试 telnet 10.1.0.1，你应该看到路由器的登录界面，警告你未授权访问是被禁止的；之后从 Honeyd 的控制台输出，告诉你一个连接已经建立，也会通知你失败的登录尝试：

```
Connection established: tcp (127.0.0.1:4245 - 10.1.0.1:23) <->
        scripts/router-telnet.pl
E(127.0.0.1:4245 - 10.1.0.1:23): Attempted login: root/test
```

当然，我们也感兴趣这个配置在欺骗 Nmap 的工作上有多好，nmap -sS -o -F 10.1.0.1 的输出应该如下所示：

```
代码浏览：
(The 1216 ports scanned but not shown below are in state: filtered)
PORT STATE SERVICE
23/tcp open telnet
```

```
Device type: router
Running: Cisco IOS 11.X
OS details: Cisco 7206 running IOS 11.1(24), Cisco 7206 router (IOS 11.1(17)
```

确实，这验证了 Honeyd 正确模拟了分配给路由器模板的网络堆栈行为。对 netbsd 模板做同样的事情，它运行在 IP 地址 10.1.0.134 上，再一次地，Nmap 应该证实了我们在配置文件中所指定的，它应列出开放的 Web 和 SSH 端口，也应告诉我们，在 Amiga 平台上运行操作系统 NetBSD。

### 4.6.2 把 Honeyd 整合到生产网络中

在很多状况下，并不容易做到把对一个不用的网络的访问路由到一台 Honeyd 机器上。取而代之，我们更愿意使用 Honeyd 在现存的生产网络中创建虚拟蜜罐，我们希望真实的机器和 Honeyd 虚拟蜜罐和谐共处，也就是说，Honeyd 最好不要扰乱生产网络的流量。我们可以通过创建一个不响应任何网络流量的默认的模板实现这一目标。为了避免代理 ARP 的复杂配置，我们决定使用 Honeyd 内置的以太网功能。通过为每一个模板分配一个以太网地址，Honeyd 将自动响应模板已经绑定的任何 IP 地址的 ARP 请求。我们仍然需要知道哪些 IP 地址没有使用，以便可以选择一个空闲的。然而，如果没有对整个网络的管理控制权，给 Honeyd 分配静态 IP 地址是不现实的。

幸运的是，我们可以使用 dhcp 命令获取动态 IP 地址，语法已经在 4.5.4 节中介绍了。当通过 DHCP 配置 IP 地址时，就不会出现意外地使用已经分配给其他机器的 IP 地址的情况了。图 4.10 给出了一个带动态 IP 地址的虚拟蜜罐实例配置。

```
create default
set default default tcp action block
set default default udp action block
set default default icmp action block

create linux
set linux personality "Linux 2.4.20"
set linux ethernet "dell"
set linux default tcp action reset
add linux tcp port 80 "scripts/web.sh"

dhcp linux on eth0
```

图 4.10 将 Honeyd 整合到一个生产网络非常简单。我们需要确保不会干扰现存机器，然后使用 DHCP 为该蜜罐取得一个 IP 地址。为了 DHCP 能够工作，我们需要为每一个想要使用 DHCP 的模板分配一个以太网地址

使用 Honeyd 的 DHCP 功能可能是启动和运行一个虚拟蜜罐的最简单的方法，这个方法最大的缺陷就是我们不知道该蜜罐的 IP 地址是多少。尽管 DHCP 服务器通常是尽量为同一主机分配同一地址，但这个特征要求该主机的 MAC 地址不能改变。不幸的是，当 Honeyd 重新启动时，每个虚拟蜜罐接受一个新的随机 MAC 地址。为了避免这种情况发生，可以使用可选的以太网变量为蜜罐

分配静态 MAC 地址。这个 MAC 地址在 Honeyd 重新启动后不会改变，DHCP 服务器就更可能为蜜罐又重新分发相同的 IP 地址。当已知一台主机 MAC 地址时，很多 DHCP 服务器支持将固定 IP 地址分配给该主机。当通过 DHCP 获得 IP 地址时 Honeyd 调试输出如图 4.11 所示。

```
honeyd[12915]: [fxp0] trying DHCP
honeyd[12915]: [fxp0] got DHCP offer: 192.168.1.38
honeyd[12915]: Updating ARP binding: 00:10:11:b1:b1:97 -> 192.168.1.38
```

图 4.11　当通过 DHCP 获得 IP 地址时 Honeyd 调试输出。当一个模板收到来自 DHCP 服务器的 IP 地址时，Honeyd 自动更新 ARP 表，以使得可以在新 IP 地址下访问蜜罐

在实例化多个蜜罐时，dhcp 命令可以重复使用。尽管 Honeyd 内部限制 65535 个 DHCP 主机，但是在正常使用情况中很难达到这个上限，大多数 DHCP 服务器不能处理这么多租约。在使用这种特性做下一步实验前，一定要保证为其他正常用户预留了可用 IP 地址。

## 4.7　服务

尽管 Honeyd 已经提供了复杂的方法响应网络流量，但是一个蜜罐的现实功能还是体现在敌手能够对话的服务上。为了提供一个现实的服务所付出的努力，可以直接通过接收来自敌手的更多信息得到回报。下面我们给出一些简单的例子，说明如何配置和编写自己的服务。

在最简单的情况下，一个服务就是一个应用程序，它从 stdin 中读取输入并把输出写到 stdout 中。通过 Inetd 开始的 Internet 服务就是一个例子。

假设我们只想创建一个非常简单的服务，对用户说“hello”，并且回送用户发送的所有输入。可以使用如图 4.12 所示的 shell 脚本实现这一服务。

```
#!/bin/sh
echo "Hello you!"
while read data
do
  echo "$data"
done
```

图 4.12　Honeyd 简单服务脚本，回送连接用户的网络输入。该脚本通过 stdin 接收网络输入，通过 stdout 发送网络输出

将该文件命名为 hello.sh 并保存在 Honeyd 的脚本目录下。接下来，需要已经建立了 Honeyd，使得你通过回环接口可以访问虚拟蜜罐。

如图 4.13 所示的配置文件创建了一个带有 hello.sh 的模板，配置运行在 TCP 端口 23 上，这是 telnet 使用的端口。保存该配置文件为 test.config，然后使用以下命令启动 Honeyd：

```
honeyd -d -i lo -f test.config
```

```
create test
add test tcp port 23 "scripts/hello.sh"

bind 10.1.0.2 test
```

图 4.13　简单的 Honeyd 配置，测试 hello.sh 服务脚本。该配置指示 Honeyd 创建一个虚拟蜜罐响应 telnet 连接

重要的是脚本必须是可执行的，并在 Honeyd 配置文件中指定正确的路径，否则会收到错误信息。如果所有都正确建立，应该可以连接到 10.1.0.2 上的虚拟蜜罐了，只要输入以下命令：

```
telnet 10.1.0.2
```

现在应该可以看到一行信息——“Hello you!”，接下来你键入控制台的每一行输入信息都会被回显。因为我们正在调试模式下运行 Honeyd，在正在运行 Honeyd 的终端窗口中你将看到有关建立的连接的额外信息。如果看不到任何信息，通常情况下表明 Honeyd 看不到你的网络数据包，那样的话，你需要回到前面的一节，确保你的路由建立正确。

无论何时 Honeyd 收到该端口的连接，它启动一个新进程执行指定脚本。如果你打算将 Honeyd 部署在一个繁忙的网络上，可能导致数百个新建进程被启动，从而可能使你的系统性能显著下降。幸运的是，有一些其他办法创建服务，而且有更好的性能，可以在第 5 章中找到更多有关这方面的信息以及如何创建更真实的服务。

需要提醒的是：Shell 脚本因命令注入攻击问题而臭名昭著，执行脚本前适当地去除和引用输入是很重要的。



## 4.8　日志

Honeyd 的框架支持几种记录网络活动的方法，它可以创建连接日志，报告对所有协议尝试和完成的连接。要获得更详细的信息，通过 stderr 服务可以记录到 Honeyd 的任意信息。该框架也使用 syslog 日志记录通信警告或系统级错误。在大多数情况下，我们期望 Honeyd 与网络入侵检测系统（NIDS）或者一组自定义脚本配合使用，用于解析和分析日志文件。

### 4.8.1　数据包级日志

通过-l 命令行选项可以启用数据包级日志，它将一个文件名作为输入参数。该文件被创建的目录，对 Honeyd 运行用户（通常是 nobody）是可写的。分析数据包日志是观察蜜罐收到的流量的最简单的方法，该日志文件包含有关源和目标 IP 地址、正在使用的协议和端口号等信息。如果一个连接被建立，日志文件还包含该连接何时开始、何时结束以及传输了多少字节等信息。图 4.14 包含来自一个日志文件的例子，以表格格式描述。

| 日期 | 协议 | T | 源 | | 目的 | | 信息 | 注释 |
|---|---|---|---|---|---|---|---|---|
| | | | IP | 端口 | IP | 端口 | | |
| 2005-04-02-15:35:15 | tcp(6) | S | 10.3.6.139 | 1827 | 10.1.2.124 | 3128 | | [Windows XP SP1] |
| 2005-04-02-15:35:56 | tcp(6) | - | 10.3.6.139 | 4378 | 10.1.2.84 | 8080: | 48 S | [Windows XP SP1] |
| 2005-04-02-15:36:11 | tcp(6) | - | 10.4.7.196 | 2671 | 10.1.2.175 | 2380: | 40 RA | [FreeBSD 5.0-5.1] |
| 2005-04-02-15:39:47 | tcp(6) | E | 10.3.6.139 | 1827 | 10.1.2.123 | 3128: | 9950 240 | |
| 2005-04-02-15:40:18 | icmp(1) | - | 10.3.5.182 | | 10.1.3.99: | | 11(0): 56 | |

图 4.14　Honeyd 数据包级日志文件实例输出。显示连接建立、终止以及探测数据包

日期列包含 Honeyd 接收到数据包时的时间戳，下一列包含有关 Internet 协议的信息，通常是 TCP、UDP 或 ICMP 协议。然而，当接收到罕见网络探测时，它也可能是任何其他的 Internet 协议。第三列以 T 标识，包含连接类型：S 代表连接开始，E 代表连接结束，“-”表示数据包不属于任何连接。接下来的两列显示了有关源 IP 地址和源端口、目的 IP 地址和目的端口的信息，对于一些协议，如 ICMP 协议，端口是空的，因为这些协议不使用端口。信息列包含与一个连接或一个数据相关的信息，当一个连接结束时，它包含 Honeyd 接收和发送的字节数。对于一个探测数据包，它包含附加协议信息：

- Tcp：协议头内设置的数据包大小和标志。Honeyd 识别下列标志：F — Fin、S — Syn、R — Rst、P — Push、A — Ack、U — Urg、E — ECE 和 C — CWR。
- ICMP：数据包的 ICMP 代码、类型和大小。参阅 Steven 的《TCP/IP 图解》一书了解更多信息[38]。
- UDP：数据包大小。

信息列包含额外的用户可读信息，在多数情况下，它至少包含一个基于被动指纹识别的远程操作系统的猜测。

对于支持连接的协议，如 TCP，Honeyd 不会记录所有的数据包，而是以类似 Netflow 方式记录连接开始和结束。其主要优点是减少了日志的混乱，例如，如果某人打算从蜜罐下载一个大文件，记录下载的每个数据包没有任何意义。Honeyd 使用 S 来标识一个连接的开始，当连接结束时（使用 E 标识），计算信息交换的总量。

数据包日志对于数据挖掘是非常有用的，一个简单的 Python 脚本可以用于计算每天探测我们蜜罐的不同 IP 地址的数量、操作系统的分布或最受欢迎端口的名单。图 4.15 给出了一个 Python 脚本的例子，用于计算每天访问我们蜜罐的不同 IP 地址的数量，测量 IP 地址的数量对于了解你的蜜罐遭受到的扫描活动十分有益，这个数量可能会随着时间的推移而增长。

这些日志文件会随着时间的推移变得非常大，这取决于你的蜜罐接收的流量。为了防止文件系统溢出，一个很好的办法是循环使用这些日志文件。通过 USR1 信号，Honeyd 支持日志循环使用，当 Honeyd 接收到这个信号，关闭所有当前日志文件，打开新创建的文件。要手动循环使用日志文件，可以使用如图 4.16 所示的脚本。

```
import sys
old_day = ''
ips = {} # Dictionary containing each unique IP once
for line in sys.stdin:
    (date, _, _, srcip, _) = line.split('  ', 4) # Extract date and source IP
    day = '-'.join(date.split('-')[0:3])
    if day != old_day:
        if old_day:
            print old_day, len(ips)
        old_day = day
        ips = {}
    ips[srcip] = 1
print day, len(ips)
```

图 4.15　一个 Python 脚本，从 Honeyd 数据包日志文件中计算每天源 IP 地址的数量

```
mv logfile logfile.0
mv logfile.srv logfile.srv
kill -USR1 $(cat /var/run/honeyd.pid)
```

图 4.16　通过向 Honeyd 发送 SIGUSR1 来循环日志文件。假设 Honeyd 被配置为记录数据包日志到 logfile 文件及记录服务级日志到 logfile.srv 文件

### 4.8.2　服务级日志

通过-s 命令行选项可以启用服务级日志，此标志把一个文件名作为输入参数。创建该文件所处的目录，对 Honeyd 运行用户（通常是 nobody）是可写的。正如数据包日志给我们提供全部流量的概况，服务日志提供给我们关于当前流量的更详细的信息，每个服务脚本可以使 Honeyd 通过打印信息到 stderr 来把信息写入此日志文件，因此，不同的服务产生的日志文件的格式也不同。如果你自己写服务模拟脚本，可以自由选择格式，以便于日后分析。

图 4.17 显示了一个远程 IP 地址相信了我们伪造的代理和 SMTP 服务器，该 IP 地址试图通过一个打开的代理连接到一个邮件服务器，匿名发送电子邮件。对于这个远程邮件服务器，似乎邮件来自于代理的 IP 地址。可以从这些日志中收集到更多有趣的例子，图中显示了试图侵入俄罗斯石油公司安全 Web 服务器或即时通讯公司的登录服务器。

| 日期 | 协议 | 源 | | 目的 | | 数据 |
|---|---|---|---|---|---|---|
| | | IP | 端口 | IP | 端口 | |
| 2005-04-10-00:56:48 | tcp(6) | 10.3.23.14 | 3259 | 10.1.3.222 | 3128: | CONNECT 10.4.228.113:25 HTTP/1.0 |
| 2005-04-10-00:59:12 | tcp(6) | 10.3.23.14 | 3343 | 10.1.3.124 | 8000: | CONNECT 10.5.167.5:25 HTTP/1.0 |
| 2005-04-10-01:04:20 | tcp(6) | 10.3.23.14 | 4116 | 10.1.3.209 | 3128: | some@net.em→schan@net.em |

图 4.17　Honeyd 服务日志文件实例输出。数据部分的格式取决于每个独立的服务模拟脚本。这个 E-mail 显示了一个伪造的代理服务器和一个伪造的开放邮件中继

## 4.9 小结

本章介绍了 Honeyd——一个创建低交互蜜罐的框架，可以监听数千个 IP 地址。需要配置你的网络，以便运行 Honeyd 的计算机可以接收所有必要的流量。这可以通过配置一个路由器或代理 ARP，或者通过 Honeyd 自己配置 MAC 地址来完成。由于 Honeyd 是一个非常复杂的应用软件，我们给出了一个安装和创建基本配置的例子。一个单一的虚拟蜜罐的行为受控于一个模板，可以为一个模板配置服务并绑定多个 IP 地址。通过本章的学习，可以把 Honeyd 配置为不同的方案。在下一章将要讨论一些 Honeyd 更高级的功能。