Ipables 指南 1.2.2

English version: Oskar Andreasson <oan@frozentux.net>
中文版: Yijun Zhu <yijunzhu@live.com>

致谢

英文的感谢信我就不再翻译了.但是我还是想借这个机会感谢国内参与或者关注 Linux 的筒子们，不管是工作需要还是兴趣使然，有了你们的努力，才有今天 Linux 在国内的普及

另外还得感谢 1.1.19 版本的 sllscn，我首先接触的就是 1.1.19 中文版，而且本文里面的翻译摘抄了部分内容。

目录

译者序

首先申明，本文翻译并未取得原作者的同意，但是本文也不是以商业为目的。故原文作者有完全版权，译文作者放弃所有版权 ☺。该系列文档是自己学习 iptables/netfilter 的重要文档，特别是国内翻译好了的 1.1.19 版本.鉴于 iptables 的发展最近很迅猛，我就在自己学习 iptables 1.2.2 之余，尝试翻译一下，当然受限于自己的英文和 iptables 的水平，一定存在很多地方不准确甚至是错误的，敬请指正。

另外就是我们（奥卡科技）针对 Linux 的转发性能做了革命性的改革，推出了 HPOS（High performance OS），**在本公司的优化 OS 上面，可以达到小包（64 bytes）的双向线速。**不要相信宣传彩页，请相信您的眼睛。

本 OS 针对个人客户免费提供，同时会提供技术支持，特别欢迎网吧/酒店/小企业使用。同时寻求企业以及集成商合作。鉴于资源问题，目前本版本主要针对 X86/IXP425 平台进行优化，其他平台暂时请不要尝试，热烈欢迎硬件设备商进行新平台移植的合作。

本公司网站：www.orcal.cn

公司 Logo：

如何阅读

本文可以仅仅当作一个参考手册，也可以选择从头到位的通读。本文的原来目标是写一个 iptables 以及 netfilter 扩展的介绍材料，但是随着时间的推移，这个目标已经发生了变化。它现在瞄准的是让它成为一个完备的参考手册，至少能够成为带你迈进 iptables/netfilter 门槛的引领者。需要指出的是这个文档既不能够处理特定的软件 bug，也不能教你怎么规避 bug。

在你使用或者阅读代码过程中，发现一些bugs或者异常行为，请先联系Netfilter的邮件列表并且告诉他们你的问题，这样他们就能够告诉你这是不是一个真正的问题甚至这个问题可能已经被解决了。Iptables/Netfilter尽量保证没有bug，但是，事情没有那么理想，总会有一两个安全方面的bugs，这些已知的bugs可以通过Netfilter main page查询到，同样这个网页也是你获取这类信息应该去的地方。

这也意味着文中的脚本不是写来解决 netfilter 软件 bug 的，之所以写他们是为了演示如何构造一个规则集来解决我们的实际问题。例如本指南不会覆盖“因为 Apache 1.2.12 版本易受攻击，我们如何关闭 HTTP 端口问题”这类的话题。（事实上，这个内容已经被覆盖，虽然不是为了这个目的）

本文档是为了让大家有一个好的而且简单的指南，让用户能够知道如何开始使用iptables，但同时本指南也尽可能的覆盖完善。因为有太多的match/target补丁，所以本文不会包含所有的match/target，需要完整列表的同学请自行去Netfilter main page上面去寻找。

假如你有任何额外的建议或者你认为你发现本文档没有包含的一些问题，请直接和我联系(原作者)。我会很乐意的查看你的建议，也许它就会在下一个版本中出现。

前提

阅读本文档需要一些 Lunix/Unix 的脚本知识，同时你要知道怎么编译自己的内核以及对于内核内部机制有一些了解。

我尝试着尽可能使读者不需要这些知识也能完全弄懂这篇文章，但是从某种程度上来说，这几乎是不可能的。

本文约定

当我们谈及命令，文件或者其他信息的时候，如下约定会被应用。

- 程序或者屏幕输入用如下的格式显示。

```
[blueflux@work1 neigh]$ ls
default  eth0  lo
[blueflux@work1 neigh]$
```

- 所有的命令和程序名都用**粗体。**
- 所有的系统部件，如硬件、内核部件、loopback 使用*斜体。*
- 计算机文本输出用这样的格式。
- 文件名和路径名象这样 /usr/local/bin/iptables。

第一章．介绍

为什么编写本文档

我发现在 HOWTO 里面缺少 2.4.x 内核的 iptables/netfilter 的函数信息，另外我尝试回答一些问题，例如状态匹配。大部分都会通过 rc.firewall.txt 例子来说明，这个例子你可以在/etc/rc.d/下面使用。这个例子是基于你可能熟悉的 masquerading HOW 文档。

还有一个小脚本*rc.flush-iptables.txt*，我写它只是为使你在把配置搞砸的时候使用。

本文如何组织

最初我是为我的 boingworld.com 编写一个很小的指南，结果有大量的读者，同时我收到了很多反馈，所以我继续编写它。最初的原始稿件大约 10-15 张 A4 纸，但是它稳定的发展着。许多人给我指出问题，例如拼写，bug 修正等等。在我写本文的时候，该篇文档独立点击数已经达到了 60w。

本文一步一步教你编写 setup，当然我衷心希望你能够理解更多关于 iptables 的软件包。我的大部分工作都是基于 rc.firewall 这个例子，因为我发现这个例子是一个很好的途径学习如何使用 iptables。我决定自顶向下地跟

随 rc.firewall 文件来学习 iptables，使然这样会困难一些，但是更有逻辑。任何时候你发现难于理解某些东西，请回来再参考本文档。

术语

本文档包含一些术语，应该有所了解。下面是部分解释，并说明了本文中如何使用它们。

连接(Connection) - 本文主要指相互之间有关联的一系列数据包。所有的这些报文链接在一起形成一种稳定的连接。换句话讲，一个连接就是一系列报文交换。例如 TCP，在三次握手之后就构成了一个连接，这样在关闭之前他们都可以被看作一个连接。

DNAT - 目的地址转换。DNAT 是一种转换报文目的地址的技术，它常常和源地址转换联合起来让多台机器共享一个网络可路由地址，并且提供服务。这个通常是利用这个同一地址匹配不同的端口来实现。

IPSEC - 安全网络协议是一种加密IPv4 报文，并且通过internet把他们安全分发的协议。更多的信息请参考文末的其他资源和链接。

内核地址空间(Kernel space) - 某种程度上来说它是用户空间的相对面，这也就是说动作发生在内核里，而不会超出这个范围。

报文(Packet) - 包含包头和数据部分，并且在网络上传输的单一结构。例如 IP 报文或者 TCP 报文。但是在 RFC 里面，报文的定义不像这么宽泛，相反 IP 报文叫做数据包，TCP 报文叫做数据段。但是为了简单起见，本文所有结构都叫做报文。

服务质量（QOS）- QOS定义了如何处理报文的一种方式以及收发报文时候应该提供的服务质量。更多信息，请参加文中的TCP/IP repetition和文末的其他资源和连接。

数据段(Segment) - 一个 TCP 数据段等同于一个报文，是一种更正式的称呼。

流(Stream) - 流就是一条连接上面收发的相关数据包。根本上来说，我用流来描述有双向收发的任何类型的连接。对于 TCP，流意味着发送一个 SYN 收到 SYN/ACK 的连接，它也可能是发送 SYN 但是接收到 ICMP error 的连接。换句话讲，这个词我用的很宽泛。

SNAT - 源地址转换，这是一种把报文的源地址转换为另外一个的技术。这样就能够让多台机器共享一个可路由地址，这个主要是为了解决 IPv4 地址不足的问题。

状态(State) - 这个术语主要指报文的状态，要么根据 RFC793，要么根据 Netfilter/iptables 自己定义的状态，需要说明的是，实现并没有完全依据 RFC793.这么做的主要原因是 Netfilter 必须对于连接和报文做出一些假设。

用户空间(User space) - 这个词语我主要指内核外的一些，例如 iptables -h 在内核外执行，而 iptables -A FORWARD -p tcp -j ACCEPT 部分在内核内执行，因为新的规则被添加到了内核的规则库。

Userland - 同上用户空间.

VPN(virtual private network) - 虚拟私有网络是一种在公共网络上创建虚拟私有网络的技术，例如 IPSec 是一种创建 VPN 的技术，而 OpenVPN 是另外一种技术。

下章预览

本章主要介绍了为什么以及如何组织本文，同时还介绍了文中常用的几个名词。

下一章会用相当长的篇幅来描述 TCP/IP。主要原因在于 IP 协议以及它的子协议都在 iptables/netfilter 里得到了广泛的应用。他们分别是 TCP/UDP/ICMP 以及 SCTP。SCTP 和其他协议比起来相当的新，所以这部分有大量的篇幅为那些不是很熟悉的人描述这个协议。下一章也会顺便介绍一下基本的和高级的路由技术。

第二章. TCP/IP协议族

Iptables 是一个极度知识敏感的工具，这也意味着 iptables 将会需要你掌握大量的知识才能够充分利用它的功能，更具体的说就是你需要深刻理解 TCP/IP 协议。

本章主要讲解一些你能够使用 iptables 所必须掌握的知识。在这些知识里面，我们会逐个学习 ip/tcp/udp/icmp 协议，他们的报文头以及他们的通常用法和互相关系。Iptables 工作在网络层和传输层，本章的关注点也正是如此。

Iptables当然也能够工作在更高的层次，例如应用层。但是这不是本章的工作，我在IP报文过滤章节会继续讨论这个话题。

TCP/IP层

正如我们已经提及的，TCP/IP 是多层的。这也意味着每一层都有一个相应的函数负责处理相应的事情，这样每一层功能都很简洁明了。

更主要的原因是整个体系结构是高度可扩张的，我们能够在不需完全重写 tcp/ip 协议栈的前提下，添加新的应用层协议。另外我们添加一个完整的 tcp/ip 协议栈到现有的应用里面，也不需重写每一个应用。当我们写一个新的网卡驱动时候，每一层都应该尽可能少的知道其他层信息。

当我们讨论内核的 tcp/ip 编程时候，我们常常谈及的仅仅是 tcp/ip 协议栈。而说到 tcp/ip 协议栈则意味着所有子层，从链路层到应用层。

在我们谈论层次的时候，主要有两种基本的架构。其中一种是 OSI（开放系统互联），也就是我们常常说的 7 层模型。这个模型我们仅仅需要了解，相比而言我们更关心 TCP/IP 协议栈。但是，从历史的观点来看，OSI 是有必要了解的，特别是你工作于不同的网络类型时候。

人们常常讨论哪一种模型更流行，但是看起来 OSI 稍占上风，当然这取决于你在哪儿生活。通常来说，在美国和欧洲，当你和销售或者技术人员讨论时候，人们默认讨论的就是 OSI 模型。

但是，在文中的剩余章节中，除非特别指出，否则我们主要谈论 TCP/IP 模型。

1. 应用层
2. 表示层
3. 会话层
4. 传输层
5. 网络层
6. 数据链路层
7. 物理层

我们所发送的报文，从 OSI 模型的最上层传输到最底层，每一层都加上自己的头到报文上面，这个就是我们所谓的封装过程。当这个报文发送到了目的地址后，它的解封装过程和上面恰恰相反，每一次都剥离自己的头部。每一个报文头部都包含了它最终需要到达的应用程序或者其他它应该到得地方。

第二要介绍的同时也更让我们产生兴趣的是我们所熟知的 TCP/IP 协议模型。如下图的 TCP/IP 列表所示，人们并没有对 TCP/IP 架构的层次达成一致共识，但是人们通常认为 3-5 层是较为流行的。在大部分图示或者解释里面，4 层经常被讨论。为了简单起见，我们主要讨论人们普遍关注的 4 层。

1. 应用层
2. 传输层
3. 网络层
4. 链路层

正如你所见的，TCP/IP 的架构和 OSI 的模型很像，但是不同。和 OSI 模型工作流程一样，在报文进入或者离开的时候，我们增加或者减少每一层的报文头。

例如，我们用最为常见的写信为例来介绍这个过程。

你想发一封信给别人，问问他们近况如何或者他们最近在忙什么。为了完成这个目的，你必须首先设计问题，这个问题就是你邮件里面所写的内容，它存储在应用层。

当我们把邮件内容写完后，接着就是写上我们要把这封信发给谁，也就是接收者是谁，可能像下面这个例子。

Attn: John Doe

上面的过程等同于传输层所作的工作，它包含在众所周知的 TCP/IP 协议族里面。在传输层，假如我们用 TCP 的话，那么这个就相当于一些特定端口，例如 25.

此时，我们就应该写上如下的地址，表明我们想把这封信发送到哪儿。

V. Andersgardsgatan 2 41715 Gothenburg

上述过程在 TCP/IP 里面就是网络层的功能，网络层包含了我们所需要到达的地址。正如我们在信封上面所写的地址，IP 层的 ip 地址等同于我们的目的地址。（IP 有如下格式 192.168.0.4）

最后一步就是把这封信放入放入邮箱，这个过程大约等同于把一个报文放到链路层。链路层能够把这个报文放到合适的网卡上进行发送。

总结一下，上面作者就是用一个发信的例子来类比 TCP/IP 每一层的功能。他把写信分成如下几个阶段：

1. 写信的内容 ---- 应用程产生数据
2. 写信的接收者 ---- 表明发给哪个应用程序
3. 收信人地址 ---- 接收者的 IP 地址是多少

4. 把信投入邮筒 ---- 链路层负责数据发送

当接收者最终获得这封信，他会从信封里取出信件来，他所收到的这封信也许要求他回信，也许不用。在每一种情况下，接收者自己决定是否回复邮件。如果他要回复邮件，那么收件人和发件人的地址就需要颠倒过来，这样接收者就变成了发送者。

有一点很重要，就是 iptables 设计的初衷就是为了工作于网络层以及传输层。Iptables 也可以完成应用层或者链路层的过滤，但是这个不是它的强项，因为它本来就不是为了这个目的而被构建的。

例如，假如你用一个字符匹配来完成报文过滤，我们就以“index.html”为例，iptables 能够工作么？通常来说，没有问题。但是，假如报文的尺寸非常小，也许就不能正常工作了，原因就在于 iptables 是基于规则的逐包处理过程，假如字符串被分割在几个报文里，iptables 就不能够看到完整的字符串，从而工作不正常。

基于这个原因，你最好安装一台带有snort的代理，通过snort来过滤应用层的数据，这些内容在IP过滤再细论述。

因为 iptables/netfilter 特别适合工作于网络层以及传输层，所以这两层也是我们后续章节的着眼点。在网络层，我们几乎全部讲述 IP 协议，当然这些还有其他的协议，例如 GRE，但是他们在网络上应用的比较少，而且 iptables 也没有重点关注过这些协议。基于上述的因素，我们主要关注网络层的 IP 协议以及传输层的 TCP/UDP 以及 ICMP。

ICMP准确的说是传输层和网络层的混合物，它运行在网络层，它在有网络层报文头的同时还有额外的头部信息。ICMP后面我们会详细论述，见ICMP 作用。

IP作用

我们前面已经说了，IP 协议属于网络层。IP 协议是这样的一种协议，它主要负责告诉你的机器，路由器，交换机以及其他设备一个报文到底往哪儿转发。该协议是 TCP/IP 协议栈的核心组件，它是构成 internet 的基础。

IP 协议用 IP 报文头封装传输层报文，IP 报文头包含了如下的信息：这个报文从哪个传输层协议过来，它准备被发送到哪台机器，它从哪儿来以及一些

其他有用的信息。所有的这些字段都是精心定义的，细化到每一个 bit，当然这一原则同样适用于我们后面将要谈论的协议。

IP 协议有许多基本功能要完成，首先它必须要能够定义数据包的传输层类型，其次 IP 协议定义了我们今天所使用的网络地址系统。我们所说的地址系统就是 IP 地址，我们常常用点分式表示，例如 127.0.0.1 这么做主要是为了便于阅读，其实 IP 地址就是 32bit 组成的整数。例如 127.0.0.1 其实就是 *01111111000000000000000000000001.*

IP 协议要能够封装/去封装 IP 数据包，即从链路层接受报文或者发送传输层的报文。这个过程看来来很明显，但是在许多情况下，它就没有那么简单了。所有功能中，有两个功能是必须的，而且对于安全或者路由社区的人有相当大的吸引力。首先 IP 协议的责任就是负责把一个报文从一台机器路由到另外一台机器或者相反。大部分时间，在单网卡机器上，这个过程都是比较简单的。因为你只有两个选择，要么送给本地进程，要么送给默认网关。但是你一定启用了转发功能，并且加载了防火墙和安全选项，这样就会给网络管理员带来很多问题。其次 IP 协议的一个功能就是分片和重组，重组收到的分片包以及按照硬件要求分片报文。当每一个分片都足够小的时候，他们就会带来严重的问题，因为报文头都不能够正确的读取，更别提里面的数据了。

在 2.4 内核以后，分片就不再是个大问题，因为 netfilter 连接跟踪模块首先会重组报文，接着再完成其他功能的处理。

IP 协议是一个非面向连接的协议，也就是说 IP 不需要协商一个连接。面向连接的协议需要协商一个连接，然后在这个连接上面收发数据，最后关闭这个连接。TCP 就是这样的协议，但是它建立在 IP 基础之上的。IP 不是面向连接的原因有很多种，其中一个原因就是它不会为很多应用增加不必要的开销。对于报文丢失的情况，通过简单的重传就可以解决问题，而不需新建连接，等待对端确认等等额外开销。

IP 也是众所周知的不可靠协议，或者简单的说就是你不知道报文是否被接收到。IP 的工作就是简单的从传输层接受报文，接着做它该做的事情，接着把加工后的报文传递给链路层，它所做的就是这些了。IP 也许会接收到一个返回报文，就是从链路层接收到，接着传递给网络层，网络层完成它的工作，最后交给传输层。IP 不关心接收到的是发送报文还是返回报文，非面向连接的原则同样适用于 IP 的不可靠性。因为不可靠也许会要求一个额外的应答报文。举个例子，我们为 servername.com 发送一个 DNS 请求。假如我们没有收到一个应答，那么我们就知道出问题了，从而重发一个请求。但是正常情况下，我们只需要发送一个请求就能够得到一个应答。但是假如我们加上可靠性到 IP 协议上来，那就意味着请求报文需要两个报文（一个请求，另外一个是请求确认），接着两个报文为完成应答（应答报文和应答确认）。换句话讲，增加了可靠性，我们所发送的报文都是以前的两倍，明显增加了额外的开销。

IP报文头

根据前面我们对IP协议的介绍，我们应该能够明白IP报文的头里面包含了好几个部分，但是IP实现的时候，每一部分都足够的小，只要能够工作就好。这样IP报文头的总体开销就会比较小。IP报文的详细内容可以参考下面的IP报文头图片。

对于报文头的解释是很简洁明了的，而且我们也只会讨论他们的最基本概念，对于每一个报文头，我们都会列出相应的 RFC，如果想了解更多的内容，请参考这些文档。RFC 全称是请求评论，但是现在他们在网络世界里有了完全不同的意义，他们是整个 internet 的定义和标准，而在以前，RFC 仅仅是一种大家互相交流讨论的一种文件罢了

The IP protocol is mainly described in *RFC 791 - Internet Protocol*. However, this RFC is also updated by *RFC 1349 - Type of Service in the Internet Protocol Suite*, which was obsoleted by *RFC 2474 - Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, and which was updated by *RFC 3168 - The Addition of Explicit Congestion Notification (ECN) to IP* and *RFC 3260 - New Terminology and Clarifications for Diffserv*.

正如你所看到，标准总在发展，有时这些标准有点难于追踪。给你一个小提示，你可以通过 RFC-editor.org 来比较这些 RFC 之间的差异。例如 IP，RFC791 是最为基本的定义，而另外的一些 RFC 只是简单的更新或者微小的改动。在讲述报文头的时候，我们会再详细的说明这些 RFC 之间的变动。

有件事情需要提醒一下，有时候，一个 RFC 可能会被废弃。通常这个说明这个 RFC 已经被彻底的进行了更新，并且最为简单的就是直接替换，而不是细微修改。当一个 RFC 被废弃的时候，在老的 RFC 里面会增加一个指向新 RFC 的链接。

Table 1-11. Internet Protocol headers

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Version				IHL				TOS/DSCP/ECN												Total Length											
Identification																				Flags			Fragment Offset							
Time To Live								Protocol												Header Checksum											
Source Address																															
Destination Address																															
Options																													Padding		

版本(Version) - 第 0 到 3bit。版本字段是二进制表示的，IPV4 为 0100，而 IPV6 为 0110.这个字段在报文过滤中很少用到。

网络报文头长(Internet Header Length) - 第 4-7bit，它表示以 4 字节为单位的报文头长，例如没有选项的报文长度为 20byte，这样这个字段就为 5.

QOS 字段-第 8 到 15bit，这个字段是 IP 报文头里面最复杂的部分之一。原因很简单，这个字段已经被更新 3 次了。他们的基本含义没有发生变化，但是他们的具体实现却改变了。最初，这个字段被称为服务类别（type of service）,0-2bit 称为优先权字段，第 3bit 表示时延，第 4bit 表示吞吐量。最后两个 bit 预留。这种实现方式在很多老的硬件里面还能够看到。但是在后来更新的 ECN 版本里面，第 6-7bit 被使用了，这样他们就被设置了值而不是原先的预留值 0.但是很多老的防火墙和路由器在检查报文时候，发现这些字段是 1 就会把这个报文丢掉。现在看来，这个和 RFC 是冲突的，但是很多时候你除了抱怨，不能做的更多的去解决问题。

第二次修改是 RFC2474，这些字段变成了 DS 的域，DS 表示差分服务。根据标准，第 0-5bit 用于差分服务编码，而第 6-7 仍然是保留的。DSCP 和最初的 TOS 工作比较相近，但是其中一个比较大的改动就是设备必须忽略没有使用的两个字段，这样根据之前 RFC 做的修改又被废弃了。

第三版本，RFC3168 定义，在这个版本里面，两个预留的字段被 ECN 所使用，ECN 就是明确早期丢弃，他的主要目的就是在实际发生拥塞之前，通过这个 bit 通知端点设备减低报文的发送速度，从而一定程度上减低速率，从而避免报文的丢弃。这两个字段分别叫做 ECT（ECN Capable Transport）和 CE（Congestion Experienced）。ECT 主要表示是否支持 ECN，而 CE 则表示是否拥塞。

最后一个版本就是最新的 RFC3260，这个版本没有太多的修改，只是引入了一些新的术语。这个版本里面也阐明了一些开发者所讨论的问题。

总长度(Total Length) - 这个字段用来表示整个报文的总长度，包括报文头和数据区。单个报文能够支持的最打长度为 65535 字节，最小的报文是 576 字节，这儿不考虑分片与否。根据 RFC791，假如你确保报文能够被正确接收，我们推荐你发送的报文大于这个长度。但是现在比较常见的是 1500 字节长度的报文。（译者注：我没有仔细看过这个 RFC，但是我的常识是报文要大于 64 字节，不知道这儿为什么是 576.）

标志(Identification) - 这个域用于在重组的时候唯一的确定同一个报文

标记(Flags) - 这些字段主要用于分片控制，其中第一个 bit 预留，而且必须设置为 0.第二个 bit 设置为 0 表示可以被分片，1 则表示不能被分片。如果这个是最后一个分片，则设置第三个 bit 为 0，否则设置 1.

分片偏移(Fragment Offset) - 就是这个报文相对于第一个分片的偏移，第一个分片报文的偏移为 0，这样判断一个分片报文的时候，如果上面 MF 设置了或者这儿的 offset 设置了都表示为分片报文。

生存时间(Time to live) - 生存时间表明这个报文最多能够存活多长时间，换句话讲就是这个报文能够经过多少跳。每一个处理该报文的设置都要对 TTL 减一，这么做主要是为了防止网络环路。当报文 TTL 为 0，被丢弃的时候，主机需要给这个报文的发送者返回一个 ICMP 超时错误。

协议(Protocol) - 这个表明上层的协议是什么。例如可能是TCP/UDP/ICMP，详细的定义可以参考Internet Assigned Numbers Authority

报文头校验和(Header checksum) - 这个是报文的报文头校验和，这个检验和在报文头被修改的时候就需要重新计算，这样的话基本上每个设备都需要重新计算，因为总存在 TTL 或者其他的修改。

源地址(Source address) - 源地址，4 个字节。主要表明这个报文从哪儿来。

目的地址(Destination address) - 表明这个报文的目的。

选项(Options) - 这是一个可变长的字段。可选项目包含以下内容：松散源路由（Loose source routing）：给出一连串路由器接口的 IP 地址。IP 包必须沿着这些 IP 地址传送，但是允许在相继的两个 IP 地址之间跳过多个路由器。严格源路由（Strict source routing）：给出一连串路由器接口的 IP 地址。IP 包必须沿着这些 IP 地址传送，如果下一跳不在 IP 地址表中则表示发生错误。路由记录（Record route）：当 IP 包离开每个路由器的时候记录路由器的出站接口的 IP 地址。时间戳（Timestamps）：当 IP 包离开每个路由器的时候记录时间。.

所有的可选项都在TCP选项列表里面，在RFC的附录里面，更多的内容参见RFC，想得到最近的列表，请检查Internet Assigned Numbers Authority。

填充(Padding) - 这个区域主要是为了让整个 IP 报文是 32bit 的整数倍。

TCP作用

TCP 协议位于 IP 协议层之上，TCP 是一个有状态的协议，并且通过内部机制能够确认报文是否被对方正确的接收。TCP 主要有如下几个作用：

5. 最主要就是确认双方的的可靠数据收发
6. 数据在网络层和应用层之间正确传输
7. 数据报文能够正确的被应用层接收
8. 报文在传输过程中不会乱序

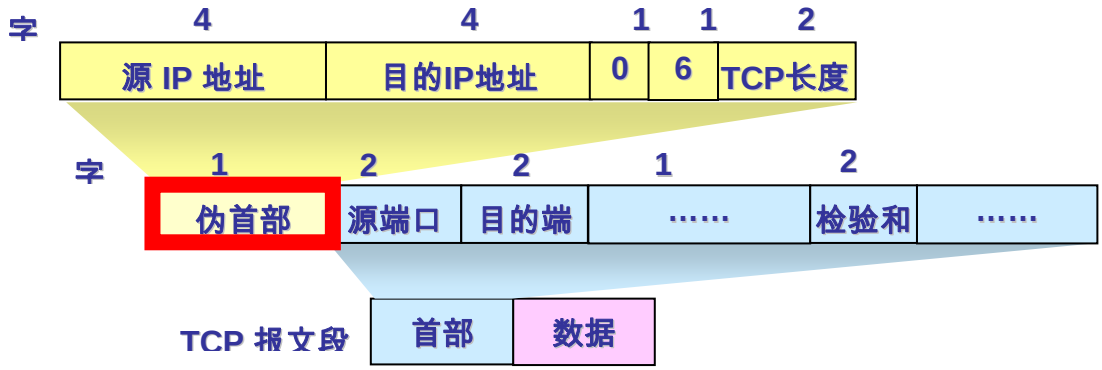
所有上面的这几点主要都是通过 TCP 报文头来得到实现。

TCP 协议工作方式就像一个拥有开始和结束信号的连续数据流，开始的信号就是 TCP 的三次握手，大致过程就是首先初始化一端发送 SYN，对方回应 SYN/ACK 或者 SYN/RST 分别表示接受或者拒绝，最后发起端发送 ACK 表示确认。这样，一条连接就正式建立了，后续的报文可以收发。

当连接建立起来之后，我们有另外一套机制来确保数据被正确收发。这个就是 TCP 的可靠性，这个功能其实实现也很简单，就是通过 TCP 的序列号和确认号。当我们发送一个报文的时候，我们赋予这个报文一个新的序列号，接着对端受到报文后，发送 ACK 对于这个报文予以确认。TCP 序列号也还可以被当作报文排序的一个参考。

一个连接的断开通过双发发送 FIN 来完成，当需要断开一个连接的时候，双方都需要发送 FIN，然后都得到对端的 FIN/ACK 确认后，这个连接才真正断开。当一段发送了 FIN 之后，这一端就不能够再发送数据，但是它可以仍然接受数据，直到对端停止发送。

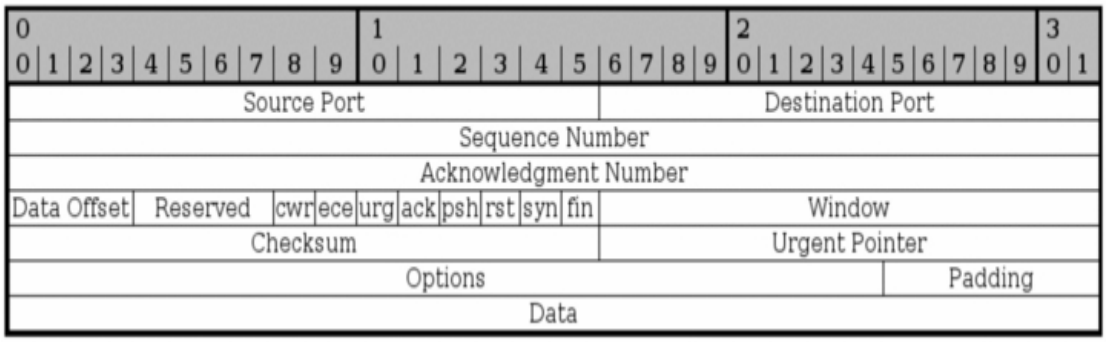
正如你将看到的，TCP 报文头里面也有一个校验和，而且 TCP 头部校验和比较有特点，他需要增加一个 12 字节的伪头部参与校验和的计算。



TCP报文头

TCP 的报文必须能够完成我们前面提到的那些功能，我们已经解释了头部里面一些字段的使用，但是我们并没有深入接触，下面这张图就是一张完整的 TCP 报文头，它以 4 字节为单位格式化了。

Table 1-31. Transmission Control Protocol headers



源端口(Source port) - 表示这个报文的发送端口，这个源端口最初和发送进程进行了直接绑定，但是现在我们的实现，通过 IP 地址，源端口和目的端口进行 hash，然后映射到相应的应用程序。

目的端口(Destination port) - 这个是 TCP 报文的远端目的端口。这个和源端口一样，原来是和应用直接绑定的，现在也是通过 hash 映射到应用程序，这样容许系统能够打开更多的连接。

序列号(Sequence Number) - 序列号主要被赋予每一个报文，这样可以对报文进行排序。最后这个序列号在回应报文的 ACK 字段予以确认。

确认号(Acknowledgment Number) - 当我们确认一个报文被接收的时候，我们就发送 ACK 确认这个报文。例如，我们接收到一个带有序列号的报文，而且这个报文是一个正常报文，那么我们就回应一个 ACK 报文，在 ACK 字段里面填写的正是过来的序列号。

数据偏移(Data Offset) - 这个字段表明 TCP 的报文头有多长，或者说数据从哪儿开始。它也是以 4 字节为单位的，这样即使不是 4 字节对齐，也可以通过尾部的填充字段来添加。

预留(Reserved) - 这些 bits 暂时预留，在 RFC793 里面，这也宝航接下来哦 CWR 和 ECN 字段，根据 RFC793，从 100 到 105 这两个 bit 必须全部被置零。后来我们引入了 ECN，这样带来了很多麻烦，因为老式的防火墙或者路由器会把这些字段置位的报文丢弃。

拥塞窗口减小(CWR) - 这个是 RFC3268 新增的标志，它主要在发送方接收到 ECE 标致的报文之后，发出的报文会被置上这个标志，表明发送窗口已经被减小，这样我们就会减慢自己 ide 发送速度。

ECN 响应(ECE) - 这个也是 RFC3268 里面添加的，主要作用就是接收方想让报文的发送方意识到网络已经拥塞了。下面添加一个 ECN 的描述文档。

TCP对ECN的支持使用TCP中预先定义的保留位。ECN定义两个新的标志，如下图所示：

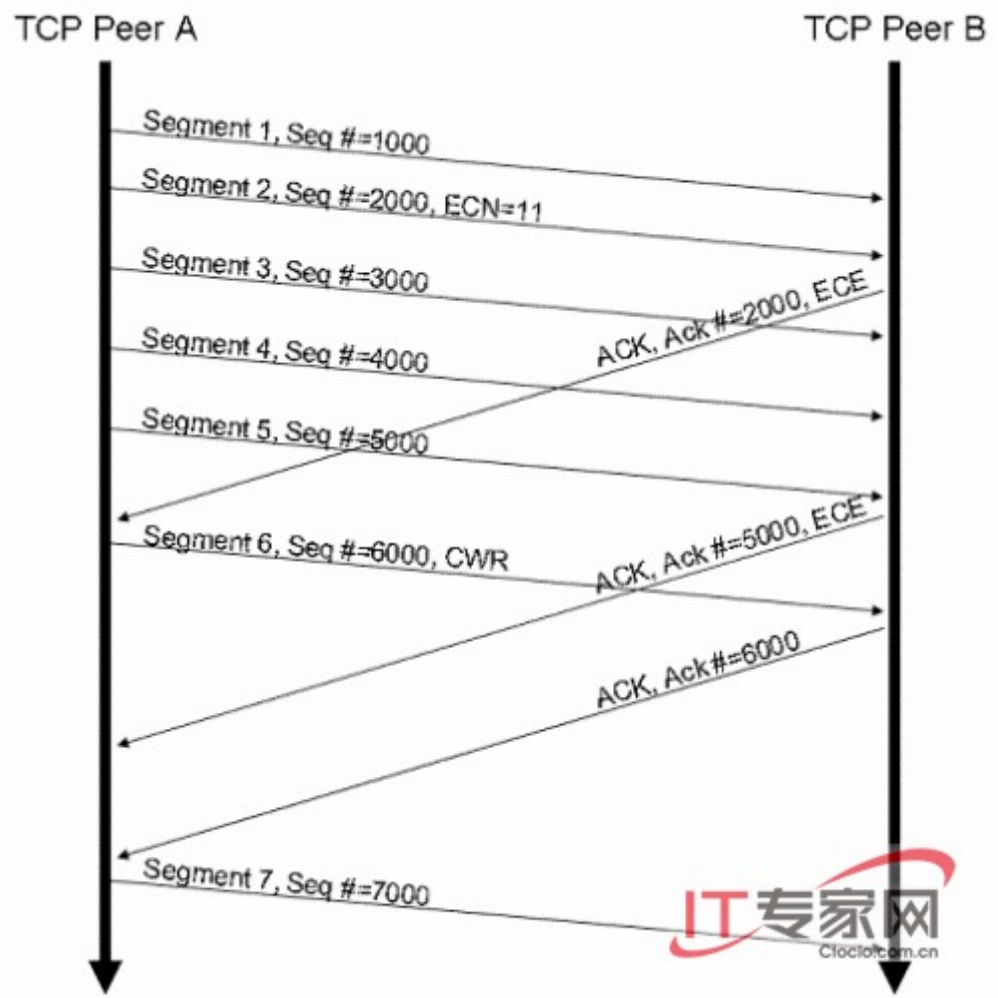
ECE：ECN 响应标志被用来在 TCP3 次握手时表明一个 TCP 端是具备 ECN 功能的，并且表明接收到的 TCP 包的 IP 头部的 ECN 被设置为 11。更多信息请参考 RFC793。

CWR：拥塞窗口减少标志被发送主机设置，用来表明它接收到了设置 ECE 标志的 TCP 包。拥塞窗口是被 TCP 维护的一个内部变量，用来管理发送窗口大小。

当两个支持ECN的TCP端进行TCP连接时，它们交换SYN，SYN-ACK和ACK包。对于支持ECN的TCP端来说，SYN包的ECE和CWR标志都被设置了。SYN-ACK只设置ECE标志。

一个支持ECN的TCP主机在支持ECN的TCP连接上发送设置了IP头部为 10 或者 01 的TCP包。支持ECN的路由器在经历拥塞时设置IP头部的ECN域为 11。当一个TCP接收端发送针对收到的一个设置ECN位为 11 的TCP包的响应时，它设置TCP包头中的ECE，并且在接下来的ACK中也做同样设置。

当发送主机接收到设置了 ECE 标志的 ACK 时，它就像感知到包丢失一样，开始减少发送窗口，运行慢启动过程和拥塞避免算法。在下一个数据包中，发送者设置 CWR 标志。在接收到新的设置 CWR 标志的包时，接受者停止在接下来的 ACK 中设置 ECE 标志。



图展示了一个在支持ECN的TCP端节点之间的一个TCP连接的例子，它们之间的一个支持ECN的路由器正在经历拥塞。

在这个例子中，TCP端A发送数据给TCP端B。TCP端A一次性发送 5 个包。包 2 通过一个拥塞的支持ECN的路由器转发，将IP包头的ECN位设置为 11。当TCP端B接收到这个包，它发送设置了ECE标志的ACK。当TCP端A接收到第一个设置了ECE的ACK以后，它降低发送速率，并且在发送下一个包(6)时设置其CWR标志。通过接收包 6，TCP端将不对接下来的ACK包设置ECE标志。详情请参考RFC 3168。

紧急指针(URG) - 这个区域表明我们是否需要使用紧急指针，设置为 0 表示不使用，反之则使用。关于紧急指针的具体含义，请参见 tcp/ip 详解的 20.8 章节。

确认(ACK) - 这个位置表明这个报文是对我们收到报文的一个回应，我们确认一个报文表明我们不仅仅收到报文，而且这个报文没有错误。发送方收到 ACK 之后就知道哪个报文已经被确认了，从而从发送缓冲释放该报文。

上送(PSH) - “推”标志位表示立刻把数据上送给用户程序，不管用户窗口情况或者是否还有后续报文。

重置(RST) - 重置标志告诉对端关闭这个链接。这个标志主要有着几个作用。例如这个连接因为某些原因已经不能工作了，或者说这个连接不存在甚至可能是报文走错了路径。

同步序列号(SYN) - 同步序列号用来初始化一个连接。这个字段在两个方向都被使用，连接发起端打开连接，并且设置 SYN 标志，而返回报文则设置 SYN/ACK。除此之外，其他地方不能使用这个标志。

完成(FIN) - 完成标志意味着这个机器没有更多的报文发送了。对端接收到 FIN 位后，就会回复 FIN/ACK。这样发送 FIN 的机器就再也不能在这个连接上发送数据了。但是另外一端可以继续发送数据直到它停止发送，这个时候对端再发送一个 FIN，接收到 FIN/ACK 的应答后，这个连接就彻底的关闭了，我们把之前的状态就做半关闭。

窗口(Window) - 窗口大小常常被接收者用来告诉发送者还有多少空闲缓冲可以使用。这个是通过发送 ACK 确认报文来完成的，这个报文里面包含了 ACK 值以及可以容纳的缓冲大小。每一次 ACK 报文都会更新这个窗口大小。

校验和(Checksum) - 这个是整个 TCP 报文头的校验和。

紧急指针(Urgent Pointer) - 这个指针指向紧急数据的尾部，如果这个连接内有紧急数据需要处理，那么发送方会设置 URG 标志以及把紧急指针指向紧急数据的尾部。

选项字段(Options) - 这个可选字段简单的来说，总是包含三方面内容。一个是初始域，通过它我们可以知道可选项的长度。第二是可选项的类别，让我们知道是那个可选项。最后就是实际的可选项数据。可选项的完全列表可以通过文末的TCP options链接找到。

填充(Padding) - 填充字段确保整个 TCP 报文头是 4 字节对齐的。填充字段总是用全 0 来填充。

UDP作用

UDP 是 IP 之上一个简单而又基本的协议，它遵循一种简单而没有任何错误检测的工作方式，并且 UDP 也是无状态的。但是，它非常适合查询/响应这种工作方式，例如 DNS 查询等等。因为我们知道一旦我们没有接收到应答，那么查询报文可能丢失了，这样我们仅仅只需重新发送一个即可，这样就减少了 TCP 的很多额外开销。另外有些时候即使我们需要一些错误检测，但是对于顺序不敏感，UDP 也比 TCP 更为高效一些，我们只需在 UDP 之上增加错误检测即可。

UDP协议由RFC768定义，这份协议的定义和协议本身一样简单可依赖。

UDP headers

UDP报文头

UDP 可以看作一个叫简化的 TCP 报文头，它主要包括源/目的端口，报文头长以及校验和。如此而已。

Table 1-33. User Datagram Protocol headers

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Source Port																Destination Port															
Length																Checksum															
Data																															

源地址（Source port）- 源端口表明这个报文是从哪儿发送出来的，但是假如你不需要对端的回复的话，这个字段也可以设置为全 0，但是在大部分实现里面，这个字段都是填充合理值。

目的端口（Destination port）- 这个表明报文的目的端口，这是必选项，和源端口不同。

报文长度（Length）- 这个长度是整个报文的长度，包括报文头和数据区。

校验和（Checksum）- 和 TCP 的校验和一样。

ICMP作用

ICMP 主要用来在主机和主机以及主机和网关之间报告错误信息，至于网关和网关之间，应该用 GGP 来完成同样的功能。正如我们已经讨论的，IP 协议没有完美的解决错误问题，但是 ICMP 可以用来协助解决一些问题，唯一的问题在于 ICMP 的报文头有点复杂。

最基本的 ICMP 报文头包含了类型，码值以及校验和，所有的 ICMP 消息都有这几个字段。类型字段表明了这是哪种类型的错误码或者是应答消息，例如目的不可达，ping 以及 ping 的回应，甚至是重定向消息。

假如需要的话，码值字段细化了更多的信息，例如是一个目的不可达的报文，这儿会有很多种原因，例如网络不可达，主机不可达，端口不可达等等，这些都通过码值来区分。

最后就是校验和，它就是简单的对整个报文校验。

可能你已经注意到了，IP 报文头里面明确的定义了 ICMP，这么做是因为 ICMP 是 IP 不可或缺的一部分。虽然看起来 ICMP 承载于 IP 之上，是一个更高层的协议，但同时又是平行的。ICMP 是 IP 协议的一个重要组成部分，有 IP 的地方一定需要 ICMP 作为一个补充。

ICMP报文头

我们前面已经说过了，ICMP 的报文头在不同类型的消息里面结构都不一样。但是大部分 ICMP 还是可以根据他们的报文头进行类别划分的，所以我们首先会介绍基本的格式，然后仔细的查看每一种格式的具体含义。

Table 1-2. Internet Control Message Protocol - Basic Headers

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Version				IHL				TOS/DSCP/ECN										Total Length													
Identification																		Flags		Fragment Offset											
Time to Live								Protocol										Header Checksum													
Source Address																		Destination Address													
Type								Code										Checksum													

每一个 ICMP 报文都包含了 IP 报文的基本格式，很多字段前面我们已经介绍了。所以我们不再赘举。

当然，ICMP 还有一些自己的新增字段，这儿我们仔细说一下啊。

- 类别（Type）- 这个字段表明ICMP的类型，每一种不同的ICMP报文，这个字段都不一样，例如目的不可达的标号为 3，完整列表请参见附录的 ICMP types，这个区域总共包含 8bits。
- 编码（Code）-每一个ICMP都有自己的编码，一些类型的所有ICMP报文都共享一个编码，但是另外一些类型的ICMP报文有多种编码，例如类型为 3 的端口不可达，他的编码就可能为 0/1/2/3/4/5，每一个编码都有其自己的含义。完整的列表请参见附录的ICMP types.
- 校验和（Checksum）- 校验和字段 16bit，它是整个 ICMP 报文头的一个校验。

这个时候，我们可以看到不同类型报文的看起来差异相当大。我们会逐一的讨论一些通用的 ICMP 类型。

ICMP Echo Request/Reply

Table 1-4. Internet Control Message Protocol - Echo/Echo Reply Message

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Type										Code										Checksum											
Identifier																				Sequence Number											
Data																															

我首先选择这个报文来介绍，是因为它和我们关联的很紧密。通俗的说，它就是我们经常所说的 ping/ping reply.echo 请求类别是 8，而请求回应的类别则是 0.当一台机器收到一个类别为 8 的 ICMP 报文是，需要回应一个类型为 0 的应答包。

当我们发送应答报文的时候，源地址和目的地址之间需要互换，这个工作做完后，我们需要重新计算校验和。在该种类型的 ICMP 报文里，只有一种编码，所以我们总是设置为 0.

- 标志（Identifier）- 这个用于唯一的确定一对请求和应答，同一连接的报文这个字段相等。
- 序列号（Sequence number）- 每一台主机的初始序列号都是 1，然后逐包递增。

该报文包含一个数据区域，默认情况下，这个数据为空，但是我们可以添加一些随机报文。

ICMP Destination Unreachable

Table 1-3. Internet Control Message Protocol - Destination Unreachable Message

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Type										Code										Checksum											
Unused																															
Internet header + 64 bits of original data datagram																															

前三个区域我们已经介绍过了，但是目的不可达报文的编码有很多种，下面我们分别介绍一下。

- Code 0-网络不可达（Network unreachable），意思是一个具体的网络当前不可达。
- Code 1-主机不可达（Host unreachable），告诉我们一个具体的主机不可达。
- Code 2-协议不可达（Protocol unreachable），告诉我们一个具体的协议不可达（例如 TCP/UDP 等等）。
- Code 3-端口不可达（Port unreachable），告诉我们一个具体的端口不可达（例如 ssh，http，ftp，等等）
- Code 4-分片需要 DF 位设置（Fragmentation needed and DF set），假如一个报文需要分片才能发送，但是报文头里面设置了 DF（不能分片）位，那么网关需要报告这个错误消息。
- Code 5-源路由错误（Source route failed），假如一个源路由发生某种错误，我们会得到这个值。
- Code 6-目的网络不可达（Destination network unknown），假如没有到某个网络的路由，就会发送这个消息。
- Code 7-目的主机不知道（Destination host unknown），如果没有到某主机路由，就会发送这个响应。
- Code 8-源主机废除（Source host isolated），这个字段现在不再使用。
- Code 9-目的网络管理禁止（Destination network administratively prohibited），如果一个网络被网关隔离，你的报文到不了那个网络，就会得到这样的回复。
- Code 10-目的主机管理禁止（Destination host administratively prohibited），如果一台主机被管理员禁止访问了，你的报文到达不了就会返回这样的错误。
- Code 11-网络不可达因为 TOS（Network unreachable for TOS），假如因为报文的 TOS 设置而不能访问网络，我们就会得到一个这样的 ICMP 错误报文。
- Code 12-主机不可达因为 TOS（Host unreachable for TOS），假如因为报文的 TOS 而不能访问目的主机，你就会得到这个错误。

- Code 13-防火墙禁止（Communication administratively prohibited by filtering），如果你的报文因为过滤被丢弃，你就会得到这个错误。
- Code 14-主机优先权错误（Host precedence violation），这个主要被第一条路由器使用，通知主机所使用的优先权被拒绝。

报文的末尾，仍然有一个数据区，这里面是完整的 IP 报文头以及原来 IP 报文的 64bits 数据。假如里面的协议包含有端口，这样端口信息就会在这个 64bits 里面。

源抑制（Source Quench）

Table 1-8. Internet Control Message Protocol - Source Quench Message

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Type										Code										Checksum											
Unused																															
Internet header + 64 bits of original data datagram																															

源抑制报文被接收端发送，告诉这个报文或者流的发起端减缓报文的发送速度。需要注意的是网关或者目的主机也有可能静默的把报文丢掉，而不是发送抑制报文。

这个报文只有 IP 报文头和 64bits 的数据区，这些数据用来匹配报文的正确发送进程。

源抑制报文的类型是 4，它的编码都是 0.

现在我们有很多种办法通告发送方，网关或者目的主机已经过载了。其中一种可行的方法就是设置报文头的 ECN 标识。

重定向（Redirect）

Table 1-7. Internet Control Message Protocol - Redirect Message

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Type										Code										Checksum											
Gateway Internet address																															
Internet header + 64 bits of original data datagram																															

ICMP 重定向只会在一个情况下产生，考虑如下的情形，你有一个 192.168.0.0/24 的网络，里面有几台主机以及两台网关。在其中一个网关到 10.0.0.0/24，另外一个网关是默认网关，它到除了 10.0.0.0 之外的所有网络。接着网络里面的一台机器没有到 10 网段的路由，但是它有默认网关。这样这台机器就把这个报文发送给默认网关，但是默认网关发现通过另外一台网关发送这个报文会更直接。所以默认网关会发送一个 ICMP 重定向报文给这台主机，告诉他到 10 网段的路由应该发给另外一台网关。这样主机收到重定向报文后，就知道距离 10 网段最近的网关了。

这个报文头里面一个很重要的字段就是网关地址，这个区域告诉主机正确的网关地址，主机在发送的时候应该用这个网关。这个报文也包含了 IP 报文头和 64bit 的数据字段。

重定向报文有 4 个不同的编码，他们分别是：

- Code 0-网络重定向（Redirect for network），对于一个网络的地址都需要重定向。
- Code 1- Redirect for host 仅仅对于一个主机重定向
- Code 2- Redirect for TOS and network，因为 TOS 和网络重定向，这个意思就是不仅仅因为网络重定向，而且要基于 TOS 字段。
- Code 3- Redirect for TOS and host，基于主机和 TOS 重定向，就是除了主机之外还要考虑 TOS.

TTL equals 0

Table 1-9. Internet Control Message Protocol - Time Exceeded Message

0									1									2									3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Type									Code									Checksum													
Unused																															
Internet header + 64 bits of original data datagram																															

TTL 为 0 的 ICMP 消息也称为时间过期消息，它的类型是 11，而且有两个编码值。假如报文经过一个网关或者在在目的主机分片重组，它的 TTL 为 0，这个报文要被丢弃。我们通过发送一个 TTL 为 0 的错误消息给发送者，通知他报文已经被丢弃了。这样发送方就能够以更发的 TTL 发送报文。

这个报文含有数据字段，数据字段包含了 IP 首都和 64bits 的数据区，这样发送者就能够匹配到发送进程。我们前面说过 TTL 为 0 的编码有两种。

- Code 0-TTL 在传输过程中为 0（TTL equals 0 during transit），假如网关在转发报文的时候，它的 TTL 值为 0 了就返回这样一个错误通知。

- Code 1-在重组过程中 TTL 为 0（TTL equals 0 during reassembly），这个是目的主机在重组分片报文的时候，发现 TTL 为 0，返回一个错误消息。

Parameter problem

Table 1-6. Internet Control Message Protocol - Parameter Problem Message

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Type										Code										Checksum											
Pointer										Unused																					
Internet header + 64 bits of original data datagram																															

ICMP 的参数错误消息的类别为 12，并且他也有 2 个编码。这个消息主要是通知发送端，网关或者接收端识别不了报文的部分内容或者说一些必须的可选项丢失了。

参数错误类型有自己的报文头，里面有一个指针，指向了出问题报文的地方。

- Code 0-IP 报文头错误（IP header bad (catchall error)），这个错误消息和指针一起合用，就能够定位到错误。
- Code 1-必须的选项丢失（Required options missing），假如一个必须的可选项丢失了，这个字段就告知发送者。

Timestamp request/reply

Table 1-10. Internet Control Message Protocol - Timestamp/Timestamp Reply Message

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Type										Code										Checksum											
Identifier																				Sequence Number											
Originate Timestamp																															
Receive Timestamp																															
Transmit Timestamp																															

时间戳类型在现在已经很少用了，但是我们还是简单的说一下，他们的编码都是 0，发送的消息类别为 13，而应答消息类别为 14.时间戳消息包含了 3 个 4 字节的计数器。

Information request/reply

Table 1-5. Internet Control Message Protocol - Information Request/Information Reply Message

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Type										Code										Checksum											
Identifier																				Sequence Number											

消息获取类型现在也被废弃不用了，主要是我们有很多上层应用可以完成同样的功能，例如 DHCP 等等，他的工作方式就是请求应答式的。

SCTP Characteristics

SCTP作用

SCTP 协议，首先因为我本人对其了解比较少，第二是目前我所接触的领域用的还比较少，第三就是想了解 SCTP 估计通过下面的内容，显得还是不足。所以要么不理会这个协议，要么重新找本书好好学习它。至少在中译本里面，SCTP 是被我剔除的。

TCP/IP 目的地址驱动路由

TCP/IP 和路由结合在一起的话，事情就会变得很复杂了。最初人们认为根据报文的目的地址路由就足够了，但是最近，事情变得越来越复杂，linux 可以根据 ip 报文的头里面的单一区域甚至是某些 bit，更为神奇的是可以通过 TCP/UDP/ICMP 报文头路由，这些都叫做策略路由或者高级路由。

我们首先简单的说一下目的地址路由，首先我们创建一个把握恩，接着本级价差这个报文的目的地址并且在路由表里面进行查找，假如目的地址是本地，这个报文就被直接送往本机，不然就送给路由表里面的出口。接着下一条收到这个报文后也做同样的事情，知道这个报文到最终目的地或者 TTL 为 0 被丢弃。

正如你看到的，这个路由策略很基本而且简单，但是引入策略路由后，世界就变得完全不一样了，我们基本可以根据报文的任何字段进行路由。

下章预览

本章带领你简单的过一遍下面的章节：

- TCP/IP 结构
- IP 协议的功能和报文头
- TCP 协议的功能和报文头
- UDP 协议的功能和报文头
- ICMP 协议的功能和报文头
- TCP/IP 的基于目的地址的路由

当你开始写防火墙规则集的时候，这些东西你马上就用得到，把所有这些零散的信息聚集起来，你就能够构建一个完美的防火墙。

第三章 IP过滤

介绍

这一章主要介绍 IP 过滤的基本理论，例如它是什么，它是怎么工作，以及应用它的一些基本知识（哪儿放置防火墙/策略等等）

这一章我们可能问的问题是：

防火墙到底放置在哪儿？大部分情况下，这个问题都很容易回答，但是在一个很大的网络拓扑里面，它可能就变得很复杂了。

我们应该应用哪些策略？什么人应该有访问哪些资源的权利？IP 过滤到底是一个什么东西。所有的这些问题在下面的章节中你都能够完美的找到答案。

IP过滤是什么

你明白 IP 过滤是什么相当的重要。Iptables 就是一个 IP 过滤器，假如你没有完全弄懂这个概念，在你以后布置防火墙的时候你就会遇到很严重的麻烦。

IP 过滤主要工作于 TCP/IP 协议栈的第二层，当然 Iptables 有工作于第三次的能力。但是古板的定义 IP 过滤器就是工作于第二层的。

假如 IP 过滤器严格的按照定义来实现，那么换句话讲就是 IP 过滤只能基于 IP 报文头来实现，例如源/目的地址，TOS/DSCP/ECN，TTL 协议号等等。但

是 Iptalbes 并没有严格的遵从那古板的协议，它做的更多，例如我们可以更深入的到报文内部过滤，例如 TCP/UDP 头，甚至 MAC 地址或者状态等等。

有一点需要提一下，Iptables 严格和和 TCP/IP 协议栈一样处理报文流,但是 iptables 是基于逐包处理这个原则之上的，他们跟踪的序列号，端口号等等确定了一个唯一的流，这个就是我们所谓的连接跟踪，在连接跟踪的基础之上，我们就能够做一些其他的操作，例如 NAT，或者是基于状态的包过滤。

正如我前面所说，Iptables 不能把不同的包连接在一起检查，默认是逐包检查。因此你有的时候就不能看到报文的完整内容。我之所以多次强调这个问题，是因为在多个邮件列表里面多次提到这个问题。例如每一次发生一个新的病毒，这儿就会有大量的人在问能否丢掉包含一个特殊字符的所有报文。关于这个问题的我们举个例子。例如我们想匹配 cmd.exe 这个字符串。

现在，假如病毒的作者他足够的聪明，他就会写一个足够小的报文，然后把 cmd 和.exe 分别放置在两个不同的报文里面。因为我们的匹配函数不能够跨包匹配，所以这些报文全部能够通过。

有些人也许会说，我们为什么不该进这个匹配算法，例如我们匹配的时候能够跨包。这种想法看起来很简单，但是他的成本很高，不管是 CPU 时间片还是内存消耗。连接跟踪已经消耗了大量的系统资源，如果我们再加重系统负担的话，这样的防火墙可怕就不是我们所期望的了，更不要说我们仅仅为了这一个功能而增加那么多的资源消耗。

我们不开发这个功能还有另外一个原因，代理比起我们来可能更适合完成这个需求，代理工作于应用层，它主要用来帮助我们加快低速页面的访问。例如 Squid 是一个 web 代理，用户首先把页面请求发送给 web 代理，接着 web 代理帮助我们完成页面的下载然后把页面传递给请求者，现在，如果再有一个用户请求同样的页面，web 代理就不再去服务器下载，可能仅仅是简单的比较一下修改时间，如果一样则直接把自己缓存的页面下载给请求者。

现在你可能理解了，代理拥有检查文件内容的内置功能，这样，它就比 Iptables 更适合更称跨包检查的任务。

最后，警告你们那么多了，不要在Iptables上面做应用层的过滤，但是总有人愿意尝试。通过下面这个网址*http://l7-filter.sourceforge.net/*，你可以下载到一个补丁，它主要完成应用层的过滤，尽管如此，它的主要目的也是为了完成QOS和流量计费，当然它也可以完成你所期望的报文过滤。应用层过滤现在仍然处于试验阶段，并且没有被netfilter的核心开发团队所接纳，因此本文不会讨论它。

IP过滤的术语和表达式

为了完全理解我们接下来要讨论的章节，有一些术语和表达式我们有必要先了解。其中包含大量的我们在 TCP/IP 章节讨论的细节。下面是一个 IP 过滤经常用到的术语和表达式。

- Drop/Deny- 当一个报文被丢弃或者拒绝的时候，它只是被简单的释放，没有更多的善后工作。不会发送一个回应告诉主机这个报文被丢弃了，接收主机也不会被告知这个报文丢弃了，这个报文从此消失。
- Rejecy-从根本上来说，这个动作和上面的 Drop/Deny 一样，只是 Reject 还会发送一个应答给发送主机，通知他这个报文被丢弃了。这个应答可能是固定的格式，也可能是根据报文里面的一些字段从新构造的一个报文。（不幸的是，对于接收端，还没有 Reject 一样的函数，通知接收主机报文被丢弃了，在某种情况下，这个可能也是好事，例如 DOS 报文被悄悄的丢弃，主机根本不就需要知道。）
- State-一个报文的状态是相对于一组报文来说的。例如，一个报文是防火墙第一次见到或者是已知的，那么他们的状态分别就是新建或者稳定流。状态可以通过连接跟踪系统获取到，在连接跟踪系统里面保存了所有的会话。
- Chain-一个链其实就是一组规则集合，这个规则会被应用到所有经过这个链的报文。每一个链都有其特定的含义，例如它关联到哪个表，它能够做什么事情。在 Iptables 里面，有许多这样的不同链，我们在后面会详细介绍。
- Table-每一个表都有特定的目的，在 iptables 里面有 4 个表，分别是 raw/nat/mangle/filter，例如 filter 表专门用来完成报文过滤，nat 表专门用来完成地址转化等等。
- Match-匹配在这儿有两个含义，第一个含义是需要匹配什么内容，例如—source 表示匹配源端口；另外一个含义是整个规则链是否匹配上了一个报文，如果匹配了就需要采取相应的动作，例如丢弃等等。
- Target-通常而言，在规则集合里面的每一条规则都有一个自己的目标，换句我们更容易理解的话就是动作。假如一个报文匹配这个规则，那么这个动作就要被执行。例如这个动作可能是丢弃或者接收报文，还有可能是地址转换。还有另外一个名词叫做跳转，我们在后面会接触到。需要说明的是，每一个规则的 Target 或者 jump 都是可选的。
- Rule-在 Iptables 的实现里面，一个规则就是一个基本匹配加上可能存在的几个扩展匹配以及唯一的一个动作。我们已经提供了很多的动作或者匹配，他们都可以被应用到规则里面。
- Ruleset-规则集，我们常常用来指 Iptables 里面所有表上面的规则集合总和。他们常常被作为配置文件保存下来。
- Jump-跳转指令和 target 很像，他们唯一的不同在于，你需要指令跳转的目的链。假如这个规则匹配了，那么这个报文就会被送到你指定的链上面进行处理。
- Connection tracking-防火墙主要通过连接跟踪来完成对于连接/流的跟踪。当然，为了完成这个功能，需要花费大量的系统资源，例如 CPU，内存。Netfilter 也不例外，但是好的一方面是通过 netfilter 的正确设置，我们就能够得到一个相对安全的防火墙。

- Accept-接受一个报文并且让他从防火墙的规则里面透过，它和 drop/deny/reject 是相对的。
- Policy-我们讲策略的时候，主要有两重含义。第一是假如报文通过防火墙规则链检查的时候，不匹配任何规则我们应该怎么做，这个是我们讲的主要关注点。另外一个策略的含义就是我们通常讲的宽泛的安全策略，例如为了部署公司的网络，我们需要怎么网络分段这些。

怎么部署IP过滤

在我们部署防火墙的第一步就是考虑把它放哪儿。假如你的网络划分的很合理的话，这应该是一个很简单的事情。首先就是在网关的位置-即主机和 internet 之间，这儿需要放置一台防火墙。同时，在大的公司里面，不同部门之间通过防火墙切割也许是一个好的主意。例如，为什么研发要能够访问人事呢，为什么不把财务和其他部门隔开呢。简单的说一下，你可能不希望调皮的工程师自己去修改自己的工资单。

简单所以下，上面的方法都意味着你应该规划你的网络越完善越好，并且保持他们之间尽可能的隔离。特别是你的网络规模在中等到大型之间，记住应该只让你所喜欢的报文进来。

假如你有服务需要对外开放的话，创建一个非军事区（DMZ）可能是一个更好的主意。非军事区就是有服务器的一小片物理网络，这儿外部网络和内部网络都可以直接访问，这样即使有人侵入了服务器，也不会对内网造成太大的伤害。

我们有很多方法来配置防火墙的策略和默认行为，这一节我们主要讨论一下实际的理论，记住在开始规划你的网络或者配置设备前，请三思。这样可以让你更加深刻的回顾自己的决定。

在我们实际开始前，你应该明白，大部分的防火墙都是有默认行为，例如一个报文没有被任何一个规则匹配，那么他可能默认的被丢弃或者接受。不幸的是每一个链只有一个策略，但是这样也方面我们依据网络接口制定出不同的策略。

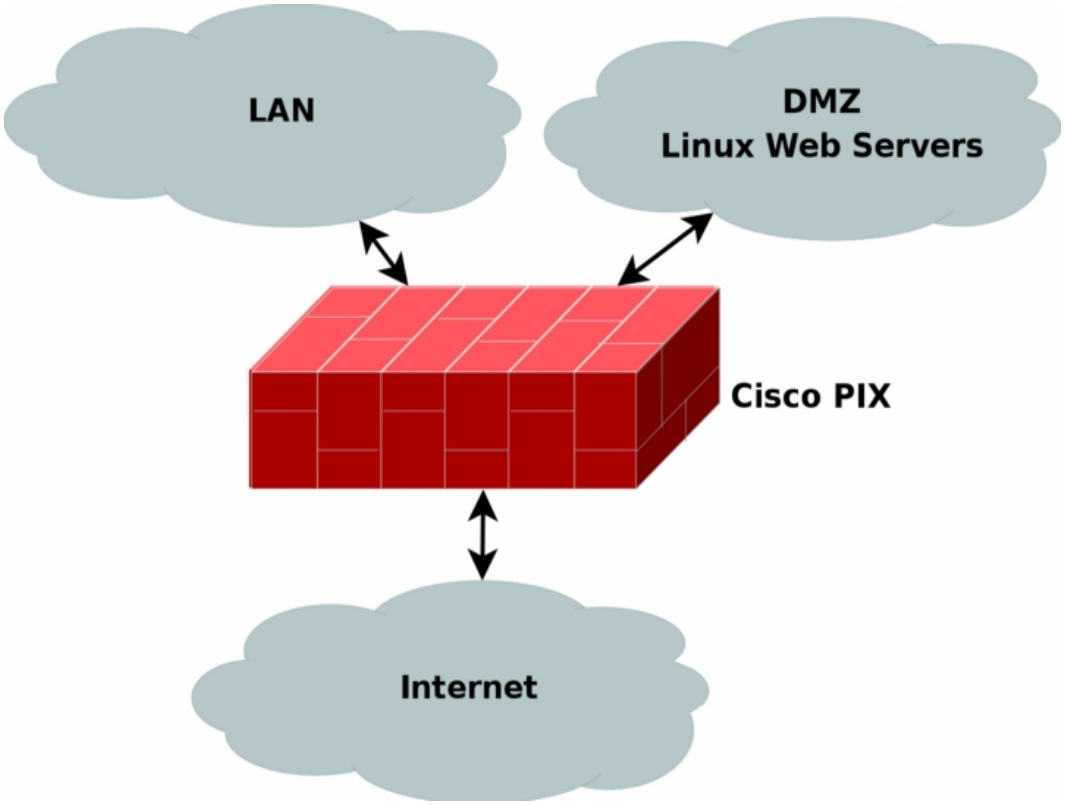
我们常用的主要就是两个策略，要么丢弃者接受所有的异常报文（这儿的异常报文指的是没有被规则所匹配上的报文）。大部分时候，我们倾向于使用丢弃策略，这样我们在上面的规则链里面添加自己想要选择的报文，这样的防火墙默认就很安全，但是也意味着你需要在前面制定大量的规则。

你要做的第一个决定就是你想使用什么类型的防火墙，你的安全概念范围到底有多大，什么样的应用程序必须要能够穿透防火墙。一些特定的应用程序对于防护墙来说就是噩梦，他们的数据协商包含在控制会话里面，这样防火墙就很难知道哪些端口是需要开放的。常见的这类应用在 iptables 里面工作良

好，但是不幸的是，对于一些不常见的应用程序，你可能就需要自己编写相应的匹配模块了。

有一些应用程序只能部分的和 Iptables 一起正常工作，通常的 ICQ 聊天是没有问题的，但是交谈或者文件传输功能就可能不正常，他们需要专门的代码来处理这个流程。因为 ICO 协议是非标准的，所以大部分的 IP filter 选择有条件支持或者仅仅提供一些补丁。

一个不错的想法是我们的安全是层次化的，我的意思是我们每一次的安全策略应该是尽可能的多条，千万不要仅仅依靠一条策略来达到全面的安全。我们只要牢记这个原则，那么我们的安全就会加深很多。下面来看一个例子。



正如你所见到的，在这个例子中，我们放置了一台 cisco 的 IPX 防火墙在网络中间，它可能做 NAT，同时她也可能构建一个 DMZ 区域。当然 IPX 还能够把除了 http/ftp/ssh 回应包之外的出去报文给隔离，它还能够容许来自内网和外网的 http 请求，还有来自内网的 ftp 和 ssh 报文。除此之外，我们的每一台服务器都是基本 linux，所以我们在每一台机器上面安装了 iptables 和 netfilter。这样即使有人把 IPX 攻陷了，我们仍然是相对安全的，这样的网络就叫做层次化安全网络。

除此之外，我们还可以在机器上面安装 Snort，Snort 是一个极好的 NIDS，它通过检查每个报文的签名，然后和特征库进行匹配，如果是一场报文就发送邮件或者通知网络管理员。当然我们也可以开启主动防御，这样它就能

够把这个地址的连接给隔离，但是我们要小心，因为 Snort 的主动防御误报的比较多。

另外在每一台服务器之前我们还可以安装一个 web 代理，好处。。。。。，我推荐的 web 代理就是 Squid。

另外一个建议安装的就是 Tripwire，它是一个优秀的主机入侵检测系统。它所做的就是给所有文件生成校验，然后定期的检查文件是否发生变化。

最后需要提醒的就是我们尽可能的选用标准化的产品，正如前面已经提到过的 ICQ 这个例子，假如你没有用标准系统，问题会层出不穷。假如这个东西只是你一个人使用，那无所谓，不过万一你运行的是一个宽带接入服务器，这个就显得尤为重要。你不能指定用户使用什么操作系统，有可能是 windows/linux，甚至是 VMS。假如你基于自己的私有系统，那么你就有可能正处于麻烦中。

下章预览

本章带你简单的浏览了一下基本的 IP 过滤知识和安全测量手段，所有的这些知识都能够加固你的网络/工作站/服务器。下面这些主题我们都有所涉及：

- IP 过滤的使用方法
- IP 过滤的策略
- 网络规划
- 防火墙规划
- 层次化安全技术
- 网络分段

在下一章我们快速的浏览一下网络地址转换（NAT），在下面一节我们就开始亲密接触 Iptables 和 netfilter 了，从那儿开始我们就正式开始接触这个“洪水猛兽”。

第四章 网络地址转换

NAT 可能是 linux 和 Iptables 最为吸引人的地方了，你不需要购买昂贵的 cisco IPX，很多小公司或者个人往往都选择 Iptables 完成这个功能。一个很重要的原因就是便宜，安全。它只需要你有一台旧电脑，一个比较新的 linux 发行版，另外还需要两张网卡以及线缆，仅此而已。

本章我们会讲解一些 NAT 的基本理念，它能够用来做什么，它是如何工作的以及在开始应用它之前，你应该考虑哪些内容。

NAT用途以及基本术语和表达式

从根本上来说，NAT 就是容许一台或者多台机器共享一个 IP 地址。例如我们有一个网络，里面有 5-10 台机器。我们把这些机器的默认网关设置为 NAT 服务器。这样所有的报文都会被发送给网关，但是 NAT 服务器的处理有些特殊。

NAT 服务器能够改变报文的源和目的地址成另外一个不同的地址。NAT 服务器接收到一个报文，接着就改写他的源/目的地址，其次在重新计算这个报文的校验和。其中很重要的一个用途就是源地址 NAT（SNAT），源地址 NAT 其实就是解决我们上面提到的那个问题，我们只有一个公网 IP，但是有 5-10 台计机器需要上网，这样我们就打开 SNAT 功能，5-10 计算机就能够共享这一个公网 IP 地址上网了。

当然还有另外一些叫做目的 NAT 的（DNAT），DNAT 在我们假设自己的服务器时候特别有用。首先它能够帮助我们节约 IP 地址（多台服务器共享一个 IP），其次在 NAT 服务器和实际的服务器之间相当于增加了一个防护体系，增强了服务器的安全系数。例如一些公司可能同时运行 web 和 ftp 服务器，他们可能在同一台机器上面，也有可能是物理上分开的机器，但是上面运行着不同聊天或者远程接入服务器（主要便于员工在家或者出差的时候，和公司内员工互相联系）。我们可以借助 DNAT，通过同一个 IP 提供上述的所有服务。

上面的例子我们经常称之为 PNAT（端口地址转换，即一个 IP 地址，不同端口区分连接），在本书中我们不会经常提及这个概念，因为他被 SNAT 或者 DNAT 所包含。

在 linux 世界里面，我们有两个武器来完成 NAT 功能，分别是 Fast-NAT 和 Netfilter-NAT.Fast-NAT 在 linux 内核的 IP 路由代码里面实现，而 Netfilter-NAT 也是在 linux 内核里面实现，但是是在 Netfilter 里面完成。我们这本书不准备深入的谈论和 IP 路由相关的内容，所为我们所说的到此为止，除了下面我所增加的一些注意点。Fast-NAT 这个名字的由来就是因为它比 netfilter-NAT 更快，fast-NAT 没有连接跟踪相关的代码。连接工作需要消耗大量的 CPU 时间和内存，所以它比较慢。但是也正如我们说的，不好的一方面就是 Fast-NAT 它没有连接跟踪，所有对于整个网络做 SNAT 它就实现的不是很完备，另外更致命的是它对于复杂动态协议的支撑相当的弱。不是说 Fast-NAT 不能做复杂协议的 NAT，而是和 netfilter-NAT 比起来，他们需要花费的精力要大得多。

我们还经常接触的另外一个 SNAT 就是伪装（Masquerade），它主要应用于动态地址分配的场合，对于 Masquerade 它会自动用出接口上面的地址作为报文的新地址。

NAT使用说明

我们前面已经解释过了，在我们使用 NAT 的时候，有一些问题需要我们注意。这其中最重要的问题就是一些特定的协议和应用程序不能在 NAT 的环境里面工作正常。希望在你所管理的网络里面，这些应用不是很常见，这样我们也不会有太多问题。

第二个问题就是一些应用程序和协议只能部分工作正常。这些应用比我们之前所说的完全不能工作要更普及一些，但是不幸的是，对于这样的结果我们也没有太多的办法。假如一些新的复杂协议不断的被开发，而且他们都是私有的，那么我们就只有那么忍受他们。

作为最后一个注意点，NAT 只是我们解决 IPV4 地址不足的一种补救手段，它不是最终方案，真正的解决方案是加速 IPV6 的推广和应用。

理论上的NAT机器配置

这是一个理论上的情景，我们需要在两个网络之间搭建一台 NAT 服务器，是他们之间能够互相访问。我们会讨论机器的硬件配置以及另外一些你搭建 NAT 服务器之前应该考虑的问题。

搭建一台NAT服务器你需要什么

在我们讨论更多的内容前，我们应该首先看一下搭建基于 Linux 的 NAT 需要什么样的硬件。假如你所需要搭建的是一个小型网络，这不会有什么问题。但是假如你开始着眼大型网络，它可能就会是一个问题。NAT 的最大问题就是它对于资源的消耗，对于一个 1-10 个人的小型私有网络，一台带有 32M 内存的 486 应该就足够了。但是假如你考虑的是 100 以上的大型网络。硬件就会是一个比价重要的问题。当然你同时还的考虑带宽，并发数。一般而言，任意一台废弃的电脑都应该足以应付这个问题，这个也是基于 Linux 的防火墙优势，你能够随意的找一台破旧机器作为服务器，这个也是他和其他防火墙比起来的优点。

你也要考虑一下网卡的情况，这个和你的实际应用相关。在挑选网卡的时候，我们最好选择被 linux 内核默认支持的网卡。我多次遇到一些 linux 没有支持的网卡，他们的驱动在 linux 上面工作的不是很理想。所以我们宁愿花多一点钱买一个好点的网卡，这值得。

需要指明一点，如果你的确需要在非常旧的机器上面搭建自己 ide 防火墙，那么我还是建议你选用 PCI 或者更好的网卡。首先网卡在以后还可以重用，另外基于 ISA 的网卡太慢了，并且它严重的加重 CPU 负担。这也就意味着你把重担放在 ISA 上面，那么下一个倒下的就会是你的机器。

最后，我们应该考虑一下一台 NAT/firewall 的机器应该分配多大的内存给他，比较理想的做法是分配至少 64M 的内存。尽管可以运行于 32M 的机器上，但是我们并不推荐那么用，NAT 不是特别占用内存，但是我们还是最好明智的尽可能多的分配内存，以防有突发流量。

正如你所见的，关于硬件我们有大量的可讨论话题，但是老实说，大部分情况下都不需要考虑硬件问题，除非你为公司或者一个大型网络构建 NAT，对于家庭用户而言，经常都是有什么硬件就是使用什么，很少有人进行测试或者比较。

NAT机器的放置

这个话题看起来相当的简单，但是在大型网络里面它可能就远远超出你的想象。一般而言，NAT 机器应该和其他过滤机器一样，放置在网络边缘。这同样也很容易想象，过滤和 NAT 运行在同一台机器上。当然，如果你的网络足够大，我们还是推荐进行网络划分，然后一个网络一台 NAT/filter 机器.

在我们前面介绍的例子里面，我们应该有两个网络以及一个 internet 连接。我们需要根据网络的大小来决定如何处理网络，例如网络足够的大，我们就应该把它们进行切分，然后每一个网络都加上一台 NAT 机器，当然我们要有足够的公网地址，最后所有的这些 NAT 服务器连接到同一台路由器上面。

怎么放置代理

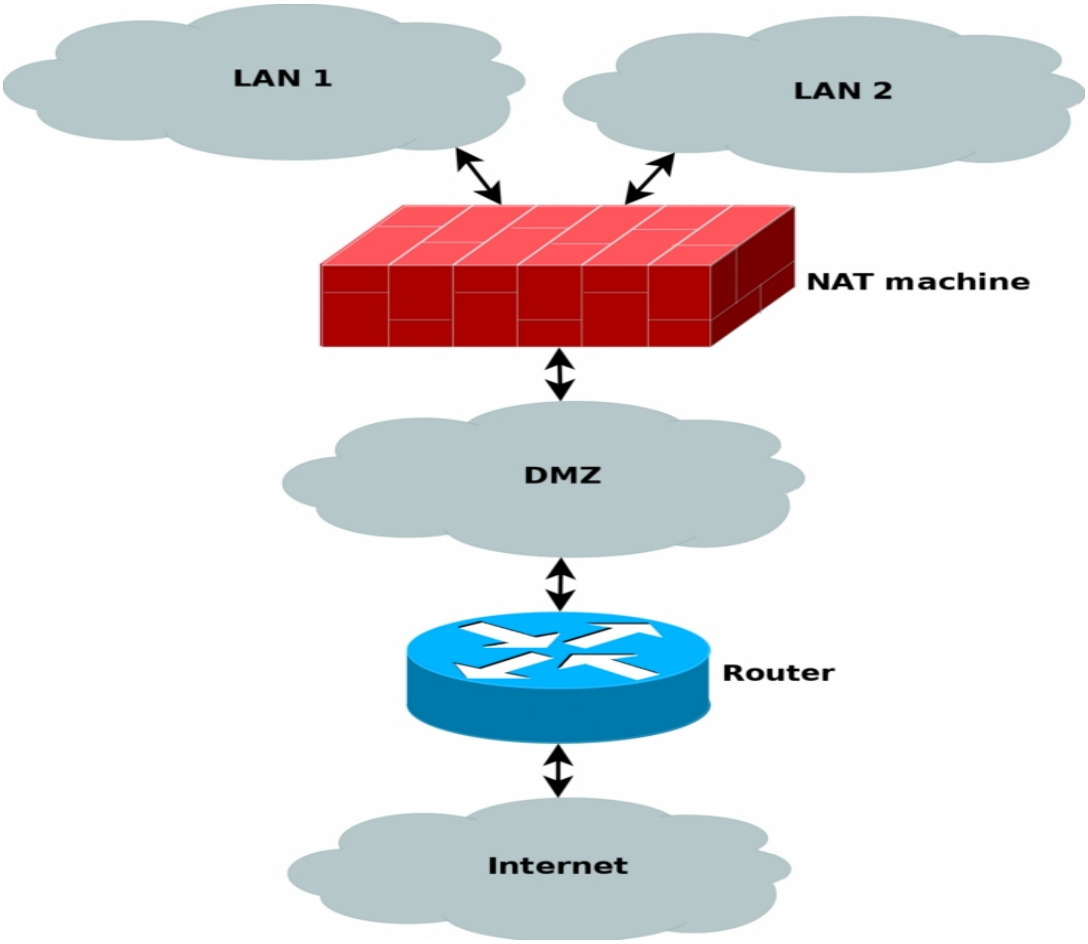
代理和 NAT 一起合用的话就会产生一些问题，特别是透明代理。一般的代理到没有什么大问题，但是在网络里面使用透明代理会让我们头疼。首先代理也会占用大量的 CPU 时间，这样把代理和 NAT 放在同一台机器就不是很明智。第二个问题就是你进行 SNAT/DNAT，这样代理就得不到报文的实际情况，这是个问题。假如他们在同一个机器上面，Squid 能够在处理流程里面参与运算，从而获取相关信息。

正如你看到的，对于这种问题我们没有太多的办法，你肯定可以运行一个代理，但是我们不推荐这么做。其中一种可行的办法就是在防火墙之前架设一个代理。然后让所有的 web 流量都到这台机器上面，本地完成 NAT。

第二个可行的办法就是在防火墙外面搭建一个代理，但是要隔断所有不通过代理的 web 流量，这个方法虽然粗鲁，但是能够保证工作。

完成NAT的最后一步

最后一步，我们要把前面的所有知识融合到一起，然后再看看如何解决 NAT 问题。我们要根据网络的规模以及它准备怎么工作，从而决定把代理放在防火墙前面还是后面。下面是我们推荐的一张图。



所有的正常流量都会被直接送到 DMZ 区域，然后发送给路由器，接着路由器把这些流量送到网络。正如你猜测的，有一些例外，web 流量会在 netfilter 内部被打上标记，接着根据标记路由到代理服务器。我们在仔细看一下过程，假如一台 NAT 机器发送一个 http 报文，在 mangle 表里面会被打伤 netfilter 的标记，接着当我们路由这个报文给路由器的时候，我们在路由表里面会检查到刚才的标记，根据这个标记，我们可以选择把这个报文交给代理服务器。接着代理服务器就会对这个报文完成后续工作。我们在文档的后续部分能够接触到这些内容，但是大部分这些工作都是和高级路由相关的。

NAT 机器和其他路由器或者机器一样，有一个在公网上可用的 IP 地址。但是所有在 NAT 网络内的机器，只有他们的私有网络地址，这样我们就可以节约一大笔钱已经 IP 地址空间。

下章预览

这一章我们主要围绕 NAT 详细介绍了相关概念，具体的说我们讨论了在使用 NAT 时候会遇到的问题以及把代理和 NAT 混用会带来的问题。本章覆盖了如下的主题：

- NAT 用法
- NAT 组件
- NAT 历史
- NAT 使用的术语
- 使用 NAT 的硬件考虑
- 使用 NAT 的问题

所有我们讨论的内容在你使用 iptables/netfilter 的时候都会遇到，NAT 在现有的网络里面普遍布置，尽管 NAT 只是一个过渡方案。在后续的章节里面我们会深入探讨 NAT 的知识。

第五章 准备工作

本章的目标就是帮助你开始开始使用 iptables/netfilter，以及让你明白在 linux 里面，iptables/netfilter 的作用.我希望你能够在这一章的指导下，安装一个属于你自己的防火墙并且完成你的试验。保持足够的时间并且坚持，你就能够让它按照你期望的方式工作。

哪儿获取Iptables

Iptables的用户侧代码可以从*http://www.netfilter.org/*下载，iptables内核代码在配置编译选项的时候可以加入，其他必要的步骤在下文会深入探讨。

安装内核

想运行最为基本的 iptables，你需要按照如下的选项配置内核，当然不管你是用 make menuconfig 还是其他命令：

CONFIG_PACKET-这个选项容许应用程序或者其他公共工具直接访问网络设备，这样典型的工具就是 tcpdump 或者 snort

CONFIG_PACKET 严格来说并不是 iptables 工作所必须的，但是考虑到它有很多用处，所以我选择把它包括在内。当然你不喜欢可以不选。

CONFIG_NETFILTER-假如你想让你的电脑作为一台连接到网络的防火墙或者网关，你就必须选择此项。换句话讲，这个选项是本手册里面其他组件能够正常工作的必须条件，我猜测你会需要它，因为你正阅读本手册。

当然为了让你的接口能够正常工作，你还需要安装合适的驱动，例如以太网或者 ppp 等。上面的操作仅仅只是增加 iptables 最为基本的组件，老实说目前为止你还不能另外它来做任何事情，它只不是是一个空的框架罢了。加入你想使用 iptables 里面更多的高级功能，你就需要在内核里进行适当的配置，这儿我基于 2.4.9 内核给你讲解可用的配置以及他们的含义。

CONFIG_IP_NF_CONNTRACK-这个模块主要用来完成连接跟踪，连接跟踪主要完成NAT以及Masquerading，当然还有其他功能。加入你想建立一个Lan上的防火墙，那么你就应该选这个项。例如*rc.firewall.txt*这个脚本也必须有这个模块才能正常工作。

CONFIG_IP_NF_FTP-假如你想做 FTP 的连接跟踪，那么这个选项就是必须的。因为在正常情况下，FTP 的连接很难做连接跟踪，连接跟踪模块需要一个叫做 helper 的帮助函数，这个选项就负责编译这个 helper。假如你没有加入这个模块，那么在穿过防火墙或者网关的时候，你的 FTP 就可能工作不正常。

CONFIG_IP_NF_IPTABLES-假如你想支持任何过滤，伪装或者 NAT 业务，这个选项都是必须的。它会增加全部的 iptables 标记框架到内核。没有这个功能，你就不能任何基于 iptables 的业务。

CONFIG_IP_NF_MATCH_LIMIT-准确的说，这个模块不是必须的，但是我们在*rc.firewall.txt*这个例子里面使用了它。这个选项提供一个LIMIT的match函数，通过这个函数我们就能够控制每分钟多少个报文被过滤，这个操作是通过一个合适的规则来完成的。例如-m linit - limit 3/minute就会每分钟最多处理 3 个报文。这个选项也能够被用来规避DOS攻击。

CONFIG_IP_NF_MATCH_MAC-这个容许我们基于MAC地址完成报文匹配。每一个以太报文都有它自己MAC地址，这样我们就能够基于MAC地址阻塞报文，并且这是

一个很有效的办法，因为一台机器的MAC地址很少变化。在*rc.firewall.txt*这个例子里面我没有使用这个功能。

CONFIG_IP_NF_MATCH_MARK-这个选项允许我们启用 MARK 的匹配器（match），例如我们首先对报文进行 MARK，然后在表中基于这个 MARK 完成报文的匹配。后面我们会详细论述 MARK 动作。

CONFIG_IP_NF_MATCH_MULTIPORT-这个选项容许我们用源或者目的端口的范围来完成报文匹配，没有这个功能，我们无法做到。

CONFIG_IP_NF_MATCH_TOS-借助这个 match，我们能够完成基于报文 TOS 字段的匹配。TOS 全称是服务类别。TOS 可以通过 mangle 表或者 ip/tc 命令进行设置。

CONFIG_IP_NF_MATCH_TCPMSS-有了这个选项，我们就能够根据报文的 MSS(max segment size)进行 TCP 报文的匹配。

CONFIG_IP_NF_MATCH_STATE-相比较ipchains，这是最大的变化之一。有了这个模块，我们就能够对报文进行状态匹配。例如，我们已经有了双向的收发报文，那么这个连接上的报文就可以被算作已建立状态（ESTABLISHED）。这个功能在*rc.firewall.txt*里面被广泛使用。

CONFIG_IP_NF_MATCH_UNCLEAN-这个模块让我们能够对一些特殊的 IP、TCP、UDP 以及 ICMP 报文进行匹配，他们或者类型不对，或者是非法的。例如我们可以选择丢弃所有这样的报文，但是我们自己也不知道对不对。请注意这个 match 还处于试验阶段，所以它也许不是在所有情况下都能够正常工作的。

CONFIG_IP_NF_MATCH_OWNER-这个选项给予我们基于 socket 的 owner 匹配的能力。例如我们可以仅仅让 root 拥有网络访问能力，这个模块最初仅仅是为了实验 iptables 能够做什么的一个例子。请注意这个 match 也还是处于试验阶段，使用请谨慎。

CONFIG_IP_NF_FILTER-这个模块增加基本过滤表，这个过滤表能够帮助你完成 IP 过滤。在过滤表里面，你能够看到 INPUT、FORWARD 以及 OUTPUT 链。假如你想对接收或者发送的报文进行任何过滤，这个选项都是必须的。

CONFIG_IP_NF_TARGET_REJECT-这个 target 让我们能够对收到的报文发送一个 ICMP 错误应答包，而不是简单的丢弃。我们需要时刻留意的是，和 ICMP 或者 UDP 不同，TCP 的连接需要通过发送 TCP 的 RST 报文才能够拒绝或者复位。

CONFIG_IP_NF_TARGET_MIRROR-这个选项让报文原样的返回给发送者。例如我们在 INPUT 链上，针对目的端口为 HTTP 的报文设置一个 MIRROR 的 target，这样一旦有人尝试访问这个端口，我们就会把它发送的报文原样返回，这样他也许就会看到自己的主页了。

MIRROR target 不被推荐使用，它最初被作为一个测试例子而编写，而且它可能会变得相当的危险。（和其他东西混在一起，可能形成 DDOS 攻击）

CONFIG_IP_NF_NAT-这个模块可能完成网络地址转换。这个选项就给了我们能够访问NAT表的能力。如果我们想实现端口转发，伪装等功能，这个选项是必须被是能的。假如你的每一台机器都有一个合理的IP地址，那么对于报文过滤或者LAN伪装，这个选项不是必须的。所以，对于我们*rc.firewall.txt*这个例子，该选项是必须选择的。

CONFIG_IP_NF_TARGET_MASQUERADE-这个模块主要用于在我们事先不知道接口地址的情况下做 NAT，例如我们通过 DHCP/PPP 等获取接口地址。伪装功能比 NAT 占用稍多一些的系统资源，但是它可以让我们在事先不知道接口地址的情况下工作。

CONFIG_IP_NF_TARGET_REDIRECT-这个 target 和应用程序代理一起合用就比较有效果，例如我们不是让报文直接通过，而是把它们重新映射到本地主机，其实通过这种方式我们就可以实现透明代理。

CONFIG_IP_NF_TARGET_LOG-这个模块给 iptables 增加一个 LOG target，我们可以把特定报文的 log 信息记录并且发送给 syslogd，这样我们就能够清楚的知道这个报文到底发生了什么，这个功能对于安全审计以及调试脚本都是非常宝贵的。

CONFIG_IP_NF_TARGET_TCPMSS-这个选项可以对付一些阻塞 ICMP 分段信息的 ISP（服务提供商）或服务。没有 ICMP 分段信息，一些网页、大邮件无法通过，虽然小邮件可以，还有，在握手完成之后，ssh 可以但 scp 不能工作。我们可以用 TCPMSS 解决这个问题，就是使 MSS（Maximum Segment Size）被钳制于 PMTU（Path Maximum Transmit Unit）。

CONFIG_IP_NF_COMPAT_IPCHAINS-增加和老的 ipchains 兼容的功能，不要把这个看成是一个长期的解决办法，很有可能在 2.6 内核里面就把这个功能去掉，赶紧升级吧。

CONFIG_IP_NF_COMPAT_IPFWADM-和上面一样，都是一个和老版本兼容的工具。

上面我看到了有很多选项，我对于每一种行为都进行了简短的介绍。他们只是 2.4.9 内核里面已经包容的一些选项。假如你想看到更多的功能，我建议你在 netfilter 网站上面查看 patch-o-matic(POM).POM 将来有可能会被加入内核，但是他们至少现在还不足够稳定。

如果你想*rc.firewall.txt*这个脚本能够正常工作，那么下面的这些选项都需要编译进内核或者模块。

- CONFIG_PACKET
- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_FTP
- CONFIG_IP_NF_IRC
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_TARGET_LOG
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_TARGET_MASQUERADE

上面所列只是*rc.firewall.txt*这个脚本能够运行的最小配置要求，其他脚本还需要更多配置我会专门说。从现在开始，我们我们只需注意要学习的这个脚本。

用户空间设置

首先我们来看看如何编译 iptables 软件包。需要注意的一点的是 iptables 的编译和配置与内核的编译和配置息息相关，另外一些发行版里面已经预装了 iptables，例如 Red Hat。但是在老的 Red Hat 里面默认是禁止的，我们会密切关注如何使能以及在其他发行版上面的安装情况。

编译用户空间应用程序

首先解压 iptables 软件包，这儿我们以 2.4 内核以及 iptables1.2.6a 为例。和其他软件一样，通过如下命令解压。

```
bzip2 -cd iptables-1.2.6a.tar.bz2 | tar -xvf  -
```

上面的命令等同于 tar -xjvf iptables-1.2.6a.tar.bz2，只是这个命令在一些老的版本上面工作不正常。

通过上面的命令，我们就解压出文件，并且创建了一个名字为 iptables-1.2.6 为目录，想了解更多编译和运行中的问题信息，请查看目录下的 INSTALL 文件。

这一步，你将配置、安装一些额外的模块，也可以为内核增加一些选项。我们这里只是检查、安装一些未被纳入内核的标准的补丁。当然，更多的在实验阶段的补丁，仅在进行其他某些操作时才会用到。

一些补丁仅仅是一些实验性质的，并且不是很好的主意去安装这些补丁。但是这些有更多的有意思的 match 和 target，不要害怕，至少看看他们是干什么用的。

为了完成这一步，我们要在 iptables 的目录内用到如下一些命令：

```
make pending-patches KERNEL_DIR=/usr/src/linux/
```

KERNEL_DIR 变量应该指向你实际放置内核源码的位置，通常情况下会是 /usr/src/linux，但是这个变化很大，至少你自己应该知道内核源码在哪儿。

上面的命令只会询问一些补丁它们是否需要被加入内核，而 Netfilter 的开发者们有大量的补丁或附件想要加入内核，通过下面的命令可以加入这些补丁。

```
make most-of-pom KERNEL_DIR=/usr/src/linux/
```

上面命令所安装的补丁，在 netfilter 的世界里被称之为 patch-o-matic，但是一些会破坏你内核的补丁还是被跳过了。你要知道这个命令的作用，要了解它们对内核原码的影响，好在在你选用之前，会有所提示。下面的命令可以安装所有的 patch-o-matic。

```
make patch-o-matic KERNEL_DIR=/usr/src/linux/
```

加入你把所有 patch-omatci 的补丁都安装，他们之间可能会互相影响，甚至会破坏内核，所以在使用先请认真的阅读帮助文件。

加入你不想给内核打补丁的话可以完全跳过上面的步骤，因为这些都不是所必需的功能。但是这些补丁里面有一些真的很有意思，所以你打开看看他们都是些什么也不会有什么坏处。

到这儿你就完成了 patch-o-matic 的补丁安装，现在你就可以用这些新加的补丁编译一个新内核。别忘了重新配置一下内核，因为有些补丁可能并没有被加入到内核。你也可以选择先编译用户空间程序然后再编译内核。

编译 iptables 用户空间程序，你只需简单的敲入这个命令。

```
make KERNEL_DIR=/usr/src/linux/
```

用户空间程序这样就应该可以正常编译完了，加入没有的话，那就看你自己了，当然你也可以加入 Netfilter mailing list，在这儿你有机会向别人请

教你的问题。安装 iptables 的时候，很多地方都可能导致错误，所以别慌张，冷静下来仔细再思考一边，或者请别人帮助都是一个不错的方法。

假如一切顺利的话，你现在已经编译完成了，接下来通过下面这个命令就可以完成安装。

```
make install KERNEL_DIR=/usr/src/linux/
```

希望你的软件现在可以工作了。使用 Iptables 用户空间程序的修改功能，你都得重新编译和安装内核和模块。更多的安装指南，请参见源码顶级目录的 INSTALL 文件。

在Red Hat7.1 上面安装iptables。

RedHat 7.1 上面默认已经安装了 2.4 内核，并且在这个内核里面 Netfilter 和 Iptables 都默认内建，同样用户空间的配置管理程序都也被包含在内。但是 RedHat 默认禁止了所有的这一切，而是用让老的 ipchain 工作。我们经常在 mail list 里面看到有人问为什么 iptables 不能工作，那下面我们来看看怎么样关闭 ipchain 以及打开 iptables。

默认 RedHat 7.1 安装的是一个基本没有用的旧版本用户空间程序，所以你也许想编译一个新的程序，并且自己定制一个新的内核，这样才能完全利用 iptables 的功能。

首先你就需要永远关闭 ipchains 模块，按照如下的方式修改位于 /etc/rc.d/里面的文件名，我们首先执行下列命令：

```
chkconfig --level 0123456 ipchains off
```

这个命令把所有指向/etc/rc.d/init.d/ipchains 的软连接改名为 K92ipchains。以 S 开头表示，在启动时会由初始化脚本运行此脚本。改为 K 开头后，就表示终止服务，或以后在启动时不再运行。这样，ipchains 以后不会再开机就运行了。

但是为了停止正在运行中的程序，你需要另外一个命令。这个命令可以用于正在运行中的服务。我们通过下面命令就能够终止 ipchains 服务：

```
service ipchains stop
```

最后，开启 iptables 服务。首先我们应该知道我们想让服务在哪一个级别运行，通常是 2-5，他们分别对应如下：

- 2.不带 NFS 支持的多用户，假如这台机器没有网络连接那么和 3 一样。
- 3.完全多用户模式，这个是我们正常启动的情况。
- 5.X11.假如你想默认进入图形界面，你就应该选择这个级别。

为了让 iptables 运行在我们所期望的级别，我们需要执行下列命令。

```
chkconfig --level 235 iptables on
```

上面的命令其实就是让 iptables 服务运行在 2-5 运行级别，假如你想让它运行在其他级别，请运行同样的命令。但是对于 iptables，其他级别是不被推荐的。级别 1 是单用户模式，你可以用来修复系统。级别 4 暂时没有使用，而级别 6 是为了关闭电脑。

通过下面的命令我们可以开启 iptables 命令：

```
service iptables start
```

在脚本 iptables 里还没有定义规则。在 Red Hat 7.1 中添加规则的方法有二种：

第一个方法是编辑/etc/rc.d/init.d/iptables，要注意在用 RPM 升级 iptables 时，已有的规则可能会被删除。

另一个方法是先装载规则，然后用命令 iptables-save 把规则保存到文件中，再由目录 rc.d 下的脚本（/etc/rc.d/init.d/iptables）自动装载。

我们先来说明如何利用“剪切粘贴”的方法设置 /etc/rc.d/init.d/iptables。为了能在计算机启动 iptables 时装载规则，可以把规则放在“start)”节或函数 start()中。

注意：如果把规则放在“start)”节里，则不要在“start)”节里运行 start()，还要编辑“stop)”节，以便在关机时或进入一个不需要 iptables 的运行级别时，脚本知道如何处理。还应检查“restart”节和“condrestart”节的设置。一定要注意，我们所做的改动在升级 iptables 时可能会被删除，而不管是通过 Red Hat 网络自动升级还是用 RPM 升级。

第二种设置的方法需要下列步骤：

首先，在脚本文件里面编写规则集，或者直接通过 iptables 命令。这样也许更适合你的需要，别忘了先做些测试。当你发现这些规则工作没有问题或者根据你的经验，没有 bugs 的时候，用 iptables-save 命令。在 shell 里面就直接直接运行 iptables-save 即可。例如

```
Iptables-save >/etc/sysconfig/iptables
```

上面的命令就保持配置到/etc/sysconfig/iptalbes 文件。下次开机的时候就会被 iptables rc.d 脚本直接加载。

另外你也可以通过 service iptables save 命令来保持文件，它会自动把规则保持为/etc/sysconfig/iptalbes，下次开机也可自动加载。

但是请记住，这两个命令不可以混用。

当所有的这些操作做完后，你就可以卸载当前的 ipchains 和 iptables 软件包了。我们这么做是不想新的应用程序和老的版本混淆。假如你从源代码安装 iptables，这一步骤就是必须的。但其实新旧软件包也不容易混淆，因为基于 RPM 的软件包不使用原码的缺省目录，删除请使用下面命令：

```
rpm -e iptables
```

既然你都不会再使用 ipchians 了，为什么还保留呢。用下面的命令删除：

```
rpm -e ipchains
```

所有这些步骤完成后，你就完成了从源码开始的 iptables 升级过程。老的可执行文件，库，头文件等等应该全部删除。

下章预览

本章讨论了在一些平台上怎么获取以及怎么安装 iptables 和 netfilter。在一些比较新的 linux 发行版里面 iptables 已经默认安装了，但是有些时候我们需要自己编译内核和应用程序来获取最新版本。本章的讲解希望能够帮得上忙。

下一章主要讨论表和链是怎么遍历的以及基于什么样的顺序访问这样的问题。这些知识的理解，对于你构建自己的规则集相当重要。不同的表会被分开介绍，因为他们本来就是为了不同的目的而创建的。

第六章 遍历表和链

这一章我们来讨论数据包是以什么顺序、如何遍历不同的链和表的。稍后，在你自己写规则时，就会知道这个顺序是多么的重要。一些组件是 iptables 与内核共用的，比如，数据包路由的判断。了解到这一点是很重要的，尤其在你用 iptables 改变数据包的路由时。这会帮助你弄明白数据包是如

何以及为什么被那样路由，一个好的例子是 DNAT 和 SNAT，不要忘了 TOS 的作用。

概述

当一个报文进入防火墙时候，首先触发硬件，接着被传递给内核中的合适设备。在这个报文到最终目的之前，它会在内核中会经历一系列步骤，

首先我们来看一下目的是本机的报文，在被实际送到最终的应用程序前，它会通过如下的步骤。

表 6-1 目的为本机的报文（就是我们自己的机器）

Step（步骤）	Table（表）	Chain（链）	Comment（注释）
1			在线路上
2			到了接口上（例如 eth0）
3	raw	PREROUTING	这个链在连接跟踪之前处理报文，它能够设置一条连接不被连接跟踪处理。
4			这儿是连接跟踪处理的点，我们在后面的状态匹配章节再说。
5	mangle	PREROUTING	这个链主要用来修改报文，例如修改 TOS 等等。
6	nat	PREROUTING	这个链主要用来处理 DNAT，我们应该避免在这条链里面做过滤，因为可能有一些报文会漏掉。
7			路由决定，例如决定报文是上本机还是转发或者其他地方。
8	mangle	INPUT	到了这点，mangle 表的 INPUT 链被使用，在把这个报文实际送给本机前，路由之后，我们需要再次修改报文。
9	filter	INPUT	在这儿我们对所有送往本级的报文进行过滤，要注意所有收到的并且目的地址为本机的报文都会经过这个链，而不管哪个接口进来的或者它往哪儿去。
10			本地进程或者应用程序，例如服务器或者客户端程序。

现在数据包是由 INPUT 链过，而不是 FORWARD 链。这样更符合逻辑。刚看上去可能不太好理解，但仔细想想就会恍然大悟的。

现在我们来看看源地址是本地器的包要经过哪些步骤：

表 6-2. 本地主机发出报文

Step	Table	Chain	Comment
1			本地进程或者应用程序（例如服务器或者客户端程序）
2			路由选择，用哪个源地址以及从哪个接口上出去，当然还有其他一些必要的信息。
3	raw	OUTPUT	这儿是你能够在连接跟踪生效前处理报文的点，这儿你可以标记某个连接不被连接跟踪处理。
4			这儿就是本地发出报文进行连接跟踪处理的地儿，更详细的信息我们在下一章状态匹配介绍。
5	mangle	OUTPUT	这儿是我们修改报文的地方，在这儿做报文过滤是不被推荐的，因为它可能有副作用。
6	nat	OUTPUT	这儿对于本级发送的报文做目的 NAT（DNAT）
7			路由决定，因为前面的 mangle 和 nat 表可能修改了报文的路由信息。
8	filter	OUTPUT	这儿是对发送报文做过滤的地方。
9	mangle	POSTROUTING	这条链可能被两种报文遍历，一种是转发的报文，另外就是本级产生的报文。
10	nat	POSTROUTING	在这儿我们做源 NAT（SNAT），我们建议你不要在这儿做报文过滤，因为有副作用。即使你设置了默认策略，一些报文也有可能溜过去。
11			在接口上发出（例如 eth0）
12			到了线路上（例如 internet）

下面的例子我们假设报文的目的地址是另外一个网络，在我们设备上只是穿过，那么它经过如下几个步骤。

表 6-3. 转发报文

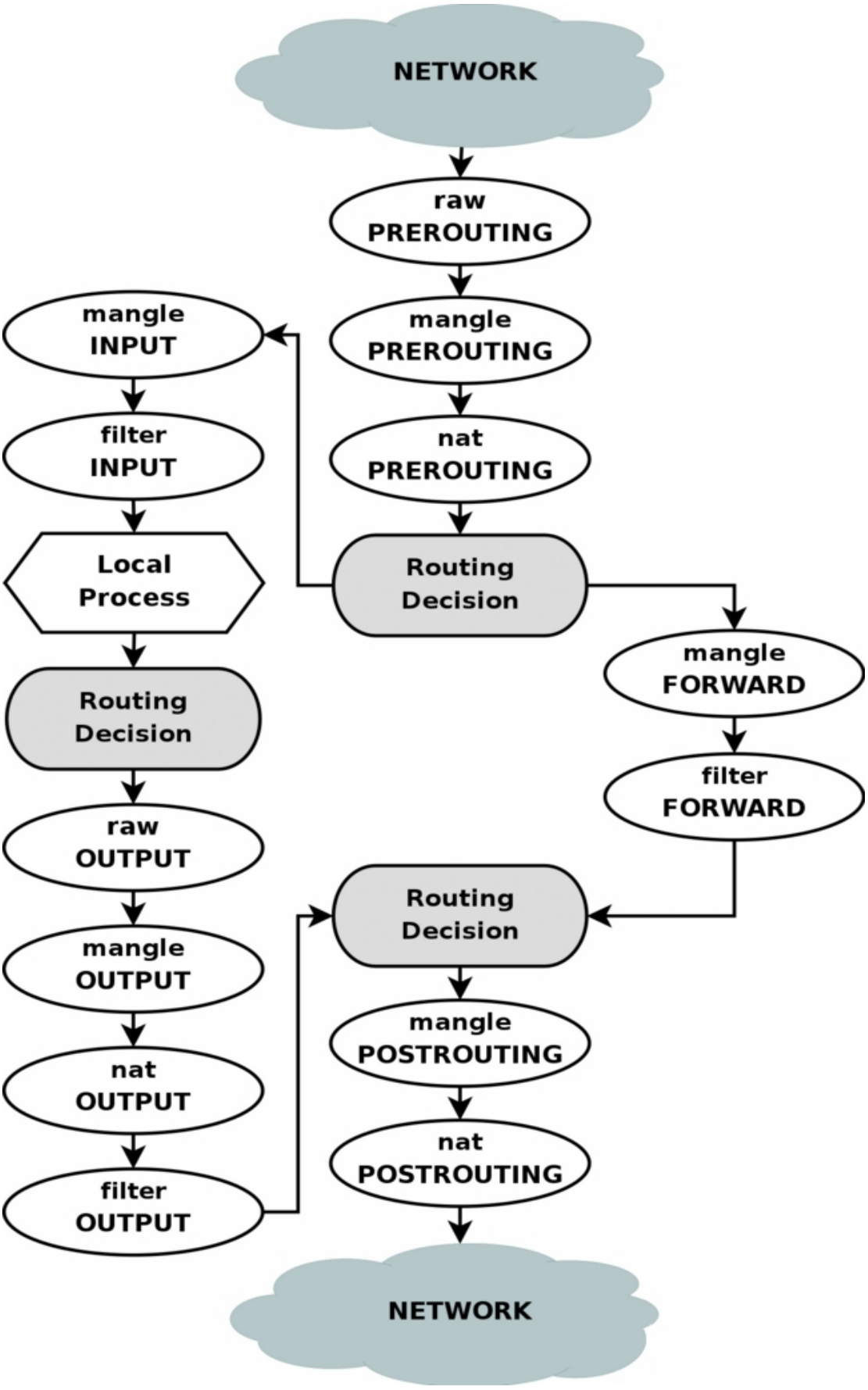
Step	Table	Chain	Comment
1			在线路上（例如 internet）
2			到了接口（例如 eth0）
3	raw	PREROUTING	在这儿你可以设置不想被连接跟踪系统处理的连接。
4			这儿做的是非本地报文的连接跟踪系统处理，还在后面的状态匹配里面详细解释。

Step	Table	Chain	Comment
5	mangle	PREROUTING	这条链主要用来修改报文，例如改变 TOS 等等。
6	nat	PREROUTING	这条链主要完成 DNAT，SNAT 后面处理。在这条链上面不要做报文过滤。
7			路由决定，例如转发报文还是上送本机。
8	mangle	FORWARD	包继续被发送至 mangle 表的 FORWARD 链，这是非常特殊的情况才会用到的。在这里，包被 mangle（还记得 mangle 的意思吗）。这次 mangle 发生在最初的路由判断之后，在最后一次更改包的目的之前（译者注：就是下面的 FORWARD 链所做的，因其过滤功能，可能会改变一些包的目的地，如丢弃包）。--摘录自 1.09 版本的中文翻译。
9	filter	FORWARD	The packet gets routed onto the FORWARD 包被继续转发至 filter 表的 FORWARD 链，只有转发报文才会到这儿，因此这儿我们做所有报文的过滤。在你自己制定规则的时候，请考虑这一点。
10	mangle	POSTROUTING	这个链也是针对一些特殊类型的包（译者注：参考第 6 步，我们可以发现，在转发包时，mangle 表的两个链都用在特殊的应用上）。这一步 mangle 是在所有更改包的目的地址的操作完成之后做的，但这时包还在本地上。--摘录自 1.09 版本的中文翻译。
11	nat	POSTROUTING	这个链只能作为 SNAT 作用，千万不要做报文过滤，伪装（Masquerading）也是在这儿工作的。
12			到了报文的出接口（例如 eth1）
13			有一次到了线路上（例如本地网络）

正如你看到的，这儿有太多的步骤需要处理，而且报文在 iptables 的任何一个地方都可能随时终止。但是我们感兴趣的还是 iptables 全貌。请一定注意这儿没有为某一个接口或者与之类似的链，FORWARD 链会被本机的所有转发报文遍历。

在前面的例子里面，请不要在 INPUT 链上面做过滤，因为 INPUT 所针对的仅仅是目的为本机的报文。

在前面三个不同的例子里面，我们看到了不同的链是如何被使用的。如果我们画一个完整的全景图，它应该看起来想下面这个样子：


NETWORK
raw
PREROUTING
mangle
PREROUTING
nat
PREROUTING
Routing
Decision
mangle
INPUT
filter
INPUT
Local
Process
Routing
Decision
raw
OUTPUT
mangle
OUTPUT
nat
OUTPUT
filter
OUTPUT
mangle
FORWARD
filter
FORWARD
Routing
Decision
mangle
POSTROUTING
nat
POSTROUTING
NETWORK

简单解释一下上面这个图，例如一个报文到了第一个需要做路由决定的点，如果它的目的不是本机，则会被转发到 FORWARD 链，反之，目的地址正是本机所监听的，那么我们就会通过 INPUT 链把报文送给本机。

还需要指出的是，报文的目的地址是本级，但是它的地址可能会在 PREROUTING 链里面被 NAT 修改。因为这个发生在第一次路由前，报文在修改后才能路由。请注意，所有报文都会经过上面的至少一条链。如果你把你一个报文做 DNAT，然后把它发回它原来来的网络，这样它就会沿着余下的链表，直到它被发送回原来的网络。

如果你想了解更多的信息，你可能想使用*rc.test-iptables.txt* 脚本，这里面制定了一些规则让我们测试表和链是怎么遍历的。

Mangle表

这个表主要用来进行报文修改，你可以自由的修改报文的 TOS 之类的。

在这个表里面，千万不要做过滤/NAT/伪装这类的事情。

下面这些 target 只能在 mangle 表里面使用，其他地方不能用。

- TOS
- TTL
- MARK
- SECMARK
- CONNSECMARK

TOS target 主要用来修改报文的 TOS 字段，这个字段可以设置一个网络的报文如何被路由这样的策略。需要注意的是这个实现并不完善，网络里面的很多路由器都不处理这个字段。换句话讲，不要设置发完网络报文的 TOS 字段，除非你准备用 iprouter2 来做路由。

TTLtarget 可以用来设置所有的报文都有一个系统的 TTL 这类的操作，一个很重要的理由就是我们可以欺骗 ISP。一些 ISP 不愿意看到用户几台计算机共享同一个连接，那些 ISP 会查找一台单独的计算机是否使用不同的 TTL，并且以此作为判断连接是否被共享的标志。

MARK target 可以给报文设置一些其他应用程序可识别的标记，例如 iprouter2 程序。这样它就能够给予标记完成路由。利用这个标记我们也可以做带宽限制或者基于类的入队。

SECMARK target 主要用在针对报文设置安全标记，这个标记可以被 selinux 或者其他安全系统使用。SCEMARK 和 CONNSECMARK 一起合用可以针对连接设置标记。

CONNSECMARK target 用来把一个连接的标记拷贝给单个报文或者相反，然后这些标记被 selinux 或者其他安全子模块使用。

NAT表

这个表只能用来完成 NAT 功能，换句话讲，它只能用来完成源/目的地址的转换。需要注意的是，只有第一个报文会经过这个表，这个连接的其他报文会自动完成转换。这个表的 target 有：

- DNAT
- SNAT
- MASQUERADE
- REDIRECT

DNAT target 主要用在我们只有一个公网 IP，然后我们需要把用户访问数据重定向到不同的服务器。换句话讲我们改变了报文的目的地址，然后重路由到实际主机。

SNAT target 改变包的源地址，这在很大程度上可以隐藏你的本地网络或者 DMZ 等。一个很好的例子是我们知道防火墙的外部地址，但必须用这个地址替换本地网络地址。有了这个操作，防火墙就能自动地对包做 SNAT 和 De-SNAT(就是反向的 SNAT),以使 LAN 能连接到 Internet。如果使用类似 192.168.0.0/24 这样的地址，是不会从 Internet 得到任何回应的。因为 IANA 定义这些网络（还有其他的）为私有的，只能用于 LAN 内部

MASQUERADE target 的作用和 SNAT 完全一样，只是计算机的负荷稍微多一点。因为对每个匹配的包，MASQUERADE 都要查找可用的 IP 地址，而不象 SNAT 用的 IP 地址是配置好的。当然，这也有好处，就是我们可以使用通过 PPP、PPPOE、SLIP 等拨号得到的地址，这些地址可是由 ISP 的 DHCP 随机分配的。

RAW表

RAW表的一个主要用途就是为了一件事情，那就是对报文设置一个标志，让这个报文不被连接跟踪系统所跟踪。我们通过NOTRACK target来实现这个功能。一个连接在NOTRACK里面被处理，接着连接跟踪系统就不会再处理这个报文。不增加一个表是解决不了前面的问题的，因为我们其他表都是在连接跟踪处理完之后再处理报文，更多信息请阅读后面的状态匹配章节。

这个表只有 PREROUTING 和 OUTPUT 两个链，只是因为这两个地方是他们 hit 连接跟踪的唯一地方。

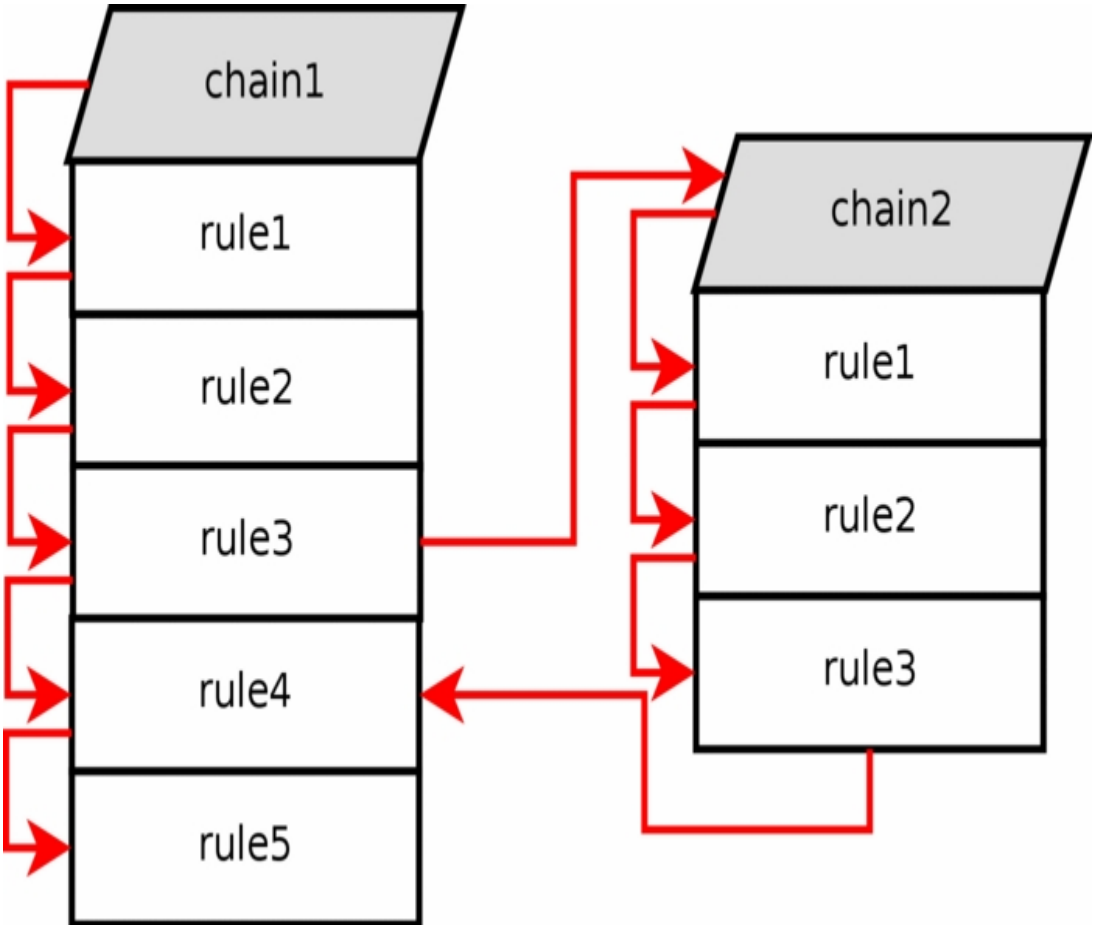
要让这个表能够工作，iptables_raw 模块需要被加载，加入我们显式的制定-t raw 标志，只要这个模块可用，它就会被自动加载。

Filter表

这个表主要是用来报文过滤的，我们可以在任何时候匹配包并过滤它们。我们就是在这里根据包的内容对包做 DROP 或 ACCEPT 的。当然，我们也可以预先在其他地方做些过滤，但是这个表才是设计用来过滤的。几乎所有的 target 都可以在这儿使用。大量具体的介绍在后面，现在你只要知道过滤工作主要是在这儿完成的就行了

用户自定义链

一个报文在处理的时候能够通过 jump 规则跳转到本表的另外一个链，但是需要指出的是这个新的链表必须是用户自定义的，不能是任何内建的链表。这个规则的查询是逐条的，自上向下的。遍历结束的条件是被一个 target 终止或者内建的链表结束。



如果报文匹配都一个 jump target，他就跳转到新的链里面自顶向下逐条匹配，直到遇到 target 终止或者 return 会主链。例如上面的图，在链 1 里面到 tule3 的时候匹配成功，跳到转链 2 从头开始执行，最后通过链 2 返回链 1 继续 rule4 的执行。

用户自定义的链没有默认策略，只有内建的才可以配置默认策略。在用户自定义的链最空有一个空白的规则，它的动作的 return 返回到主链。

用户自定义链的每个条规则都必须逐一的被检查，直到有一个规则匹配这个报文，接着调用这个规则的 target 处理这个报文，处理的结果当然可能结束或者继续，另外还有一种可能就是达到了规则链的末尾。加入到达了规则链的末尾，那么报文就会被送回父链，父链可能是用户自定义链，也有可能是内嵌的链。

下章预览

在这一章，我们讨论了好几种链和表以及他们是怎么遍历的，这里面不仅仅有系统内建的链，也有我们用户自定义的链。这一块内容非常重要，它可能看起来很简单，但是如果你没有完全理解，那么就有可能引发致命的错误。

下一章我们主要讨论状态匹配，以及状态是如何变化的和这种变化是如何反应到单个报文上面的。下一章的内容和本章一样重要。

第七章 状态匹配

这一章我们主要从细节上讲解状态匹配的工作原理，读完本章，你应该清醒的知道状态匹配是如何工作的。在这一节，我们还会通过一系列的例子来说明它的工作机制，希望这样能够加深你对状态的理解。

介绍

状态机制是 iptables 中特殊的一部分，其实它不应该叫状态机制，因为它只是一种连接跟踪机制。但是，很多人都认可状态机制这个名字。文中我也或多或或少地用这个名字来表示和连接跟踪相同的意思。这不应该引起什么混乱的。连接跟踪可以让 Netfilter 知道某个特定连接的状态。运行连接跟踪的防火墙称作带有状态机制的防火墙，以下简称为状态防火墙。状态防火墙比非状态防火墙要安全，因为它允许我们编写更严密的规则。

在 iptables 里面，报文适合连接跟踪的四种状态相关，分别是 NEW, ESTABLISHED, RELATED and INVALID。我们后面对一种状态都会深入探讨，通过使用—state 参数我们就能够轻易的控制谁或者什么能够发起新的会话。

所有连接跟踪都是通过内核中一个叫做 conntrack 的框架来实现，conntrack 可以被编译进内核，也可以作为一个模块存在。大部分情况下，我们都需要也想要一个详细的连接跟踪而不是默认的这个。因此，这儿就会有更多的功能模块分别处理 TCP/UDP/ICMP 等协议。这些功能模块从报文里面提取一些特定信息来标志一个特定的连接。例如 UDP 流通过用他们的目睹地址，源地址，目的端口以及源端口组合来表示。

在之前老内核里面，我们能够关闭/使能碎片重组。但是因为 iptables 和 netfilter 的引入，这个功能被取消了。没有重组功能，连接跟踪就不能正常的工作，所以重组默认是自动开启的。如果你想禁止重组功能，那你就得关闭连接跟踪功能。

连接跟踪的状态主要在两个地方被触发，一个是 PREROUTING，另外一个数 OUTPUT，他们分别对应外来报文和本级产生报文。例如我们从本级发送一个报

文，那么在 OUTPUT 链里面，它的状态会变成 NEW，当我们接收到回应报文的时候，连接状态就 PREROUTING 处更改为 ESTABLISHED，以此类推。加入第一个报文不是我们尝试的，那么在 PREROUTING 出就设置为 NEW，然后我们发送回复的时候，在 OUTPUT 设置为 ESTABLISHED。

连接跟踪实体

我们先来简要的看一下连接跟踪的实体以及从/proc/net/ip_conntrack 里面读出来的值。这里面包含了当前你连接跟踪数据库的所有实体。如果你的连接跟踪模块已经加载了，那么我们查看/proc/net/ip_conntrack 可能会得到如下结果：

```
tcp      6 117 SYN_SENT src=192.168.1.6 dst=192.168.1.9 sport=32775 \
     dport=22 [UNREPLIED] src=192.168.1.9 dst=192.168.1.6 sport=22 \
     dport=32775 [ASSURED] use=2
```

conntrack 模块维护的所有信息都包含在这个例子中了，通过它们就可以知道某个特定的连接处于什么状态。首先显示的是协议，这里是 tcp，接着是十进制的 6（译者注：tcp 的协议类型代码是 6）。之后的 117 是这条 conntrack 记录的生存时间，它会有规律地被消耗，直到收到这个连接的更多的包。那时，这个值就会被设为当时那个状态的缺省值。接下来的是这个连接在当前时间点的状态。上面的例子说明这个包处在状态 SYN_SENT，这个值是 iptables 显示的，以便我们好理解，而内部用的值稍有不同。SYN_SENT 说明我们正在观察的这个连接只在一个方向发送了一 TCP SYN 包。再下面是源地址、目的地址、源端口和目的端口。其中有个特殊的词 UNREPLIED，说明这个连接还没有收到任何回应。最后，是希望接收的应答包的信息，他们的地址和端口和前面是相反的。

连接跟踪记录的信息依据 IP 所包含的协议不同而不同，所有相应的值都是在头文件 linux/include/netfilter-ipv4/ip_conntrack*.h 中定义的。IP、TCP、UDP、ICMP 协议的缺省值是在 linux/include/netfilter-ipv4/ip_conntrack.h 里定义的。具体的值可以查看相应的协议，但我们这里用不到它们，因为它们大都只在 conntrack 内部使用。随着状态的改变，生存时间也会改变。

最近 patch-o-matic 里有一个新的补丁，叫做 tcp-window-tracking，可以把上面提到的超时时间也作为系统变量，这样我们就能够在系统运行时改变它们的值。以后，我们就不必为了改变这些值而重编译内核了。

这些可通过/proc/sys/net/ipv4/netfilter 下的一些特殊的系

统调用来改变。仔细看看
/proc/sys/net/ipv4/netfilter/ip_ct_*里的变量吧。

当一个连接在两个方向上都有传输时，conntrack 记录就删除[UNREPLIED]标志，然后重置。在末尾有 [ASSURED]的记录说明两个方向已没有流量。这样的记录是确定的，在连接跟踪表满时，是不会被删除的，没有[ASSURED]的记录就要被删除。连接跟踪表能容纳多少记录是被一个变量控制的，它可由内核中的 ip- sysctl 函数设置。默认值取决于你的内存大小，128MB 可以包含 8192 条目录，256MB 是 16376 条。你也可以在
/proc/sys/net/ipv4/ip_conntrack_max 里查看、设置。

另外一种更有效的设置方法是模块加载的时候，设置 hashsize 的大小，连接跟踪的最大连接数等于这个值的 4 倍，我们举下面的例子来说明。

```
work3:/home/blueflux# modprobe ip_conntrack hashsize=4096
work3:/home/blueflux# cat /proc/sys/net/ipv4/ip_conntrack_max
32768
work3:/home/blueflux#
```

用户空间状态

正如你已经看到的，根据协议的不同，报文在内核里面可能经历几个完全不同的状态。但是出了内核及用户空间，我们就只有前面介绍的 4 种状态了。这 4 种状态分别是 NEW, ESTABLISHED, RELATED and INVALID. 它们主要是和状态匹配一起使用。下面就简要地介绍以下这几种状态：

表 7-1 用户空间状态

State	Explanation
NEW	NEW 状态表示这个报文是我们所看到的第一个报文，也意味着在连接跟踪系统里面，它马上就被匹配。例如我们发送一个 SYN 报文，它就是连接跟踪系统所看到的第一个报文，接着它将会被匹配。但是一些不是 SYN 的报文也有可能被认为是 NEW 状态，这样在某些情况下可能会导致一些问题，但是它对于我们重新恢复被其他防火墙关闭的连接很重要，或者说一个连接已经超时了，但是没有真正关闭。
ESTABLISHED	ESTABLISHED 状态表示在双向都看到流量了，并且会持续匹配 。ESTABLISHED 状态其实很容易理解，要想进入 ESTABLISHED 状态，一个机器只需要发送一个报文，然后收到相应的应答报文。只要接受到一个应答，状态就会从 NEW 变成 ESTABLISHED，即使接收到的是 ICMP 的错误报告。

State	Explanation
RELATED	RELATED 是个比较麻烦的状态。当一个连接和某个已处于 ESTABLISHED 状态的连接有关系时，就被认为是 RELATED 的了。换句话说，一个连接要想是 RELATED 的，首先要有一个 ESTABLISHED 的连接。这个 ESTABLISHED 连接再产生一个主连接之外的连接，这个新的连接就是 RELATED 的了，当然前提是 conntrack 模块要能理解 RELATED。ftp 是个很好的例子，FTP-data 连接就是和 FTP-control 有 RELATED 的。还有其他的例子，比如，通过 IRC 的 DCC 连接。有了这个状态，ICMP 应答、FTP 传输、DCC 等才能穿过防火墙正常工作。注意，大部分还有一些 UDP 协议都依赖这个机制。这些协议是很复杂的，它们把连接信息放在数据包里，并且要求这些信息能被正确理解。
INVALID	INVALID 意味着这个报文不能被识别或者它没有任何状态。导致这个的原因可能有很多种，例如内存不足或和返回的 ICMP 没有匹配上任何连接。一般而言，在这种状态下丢包是一个很好的选择。
UNTRACKED	UNTRACKED 状态，简单的说，报文在 raw 表里面被 NPTRACK target 进行了标记，然后这个报文在连接跟踪表里面的状态就是 UNTRACKED。这也就是说所有 RELATED 的链接都看不到，一些特殊情况我们需要仔细考虑，例如相关的 ICMP 报文需要特殊处理。

这些状态可以—state 选项和在一起完成基于状态的报文匹配，这么做让我们的连接跟踪系统不可思议的强健和高效。以前，我们需要打开所有 1024 以上的端口让报文进行我们本地网络，但是有了连接跟踪系统之后，就再也没有必要打开所有的端口了。因为我们只需要打开我们所希望的返回端口。

TCP连接

这一节以及下一节，我们都是关注连接状态以及他们和三个基本协议是怎么配合工作的。我们也仔细了解一下这三者之外的报文，连接是怎么建立的。我们选择 TCP 开始，是因为 TCP 本身就是一个有状态的协议，并且 iptables 里面的连接跟踪也是一个很有意思的模块。

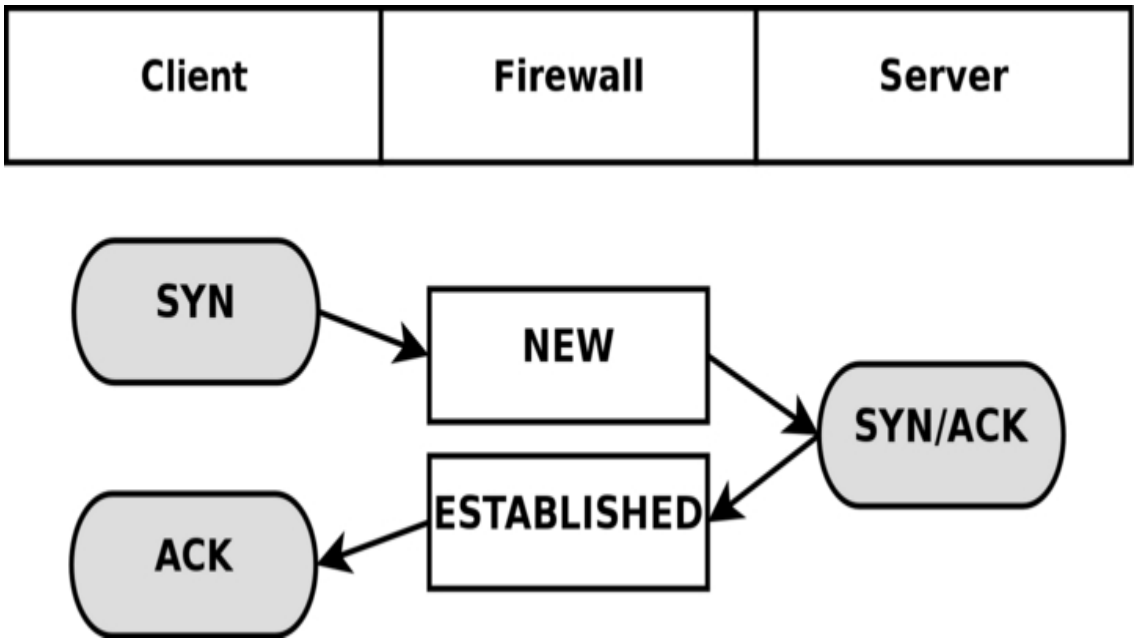
TCP 连接通过三次握手来初始化，这样我们就建立和协商了一个实际数据发送的连接。整个会话是通过 SYN 报文开始的，接着回应一个 SYN/ACK 报文，最后给予 ACK 报文通知连接已经建立起来。现在我们就能够通过这个连接发送数据了，最大的问题是连接跟踪系统是怎么参与这个问题，答案马上就来。

正如用户所想的，连接跟踪对于所有的连接类型工作流程都是一致的，我们仔细看一下下面的图片，并且观察连接的状态变化。你可能已经看出来了，连接跟踪系统并没有严格的遵从用户观点的 TCP 状态变化。一旦它看到第一个

报文，那么连接状态就是 NEW，接着一旦收到回应包，连接的状态就变成 ESTABLISHED。

假如你稍微多思考一会，你就应该理解为什么这么做了。通过这个特殊的实现方式，我们就可以让 NEW 和 ESTABLISHED 状态的报文离开本地主机，但是只有我们希望的报文才能发送到我们主机，即转化成 ESTABLISHED 状态，其他报文都被丢掉，这样的工作方式很完美。反过来，如果在连接建立的整个过程中，连接的状态都是 NEW，那么我们就不能够阻止外部网络发送给我们的报文。因为我们必须要能够容许 NEW 状态的报文一次次的回复本机，从而完成三次握手，这样我们就不能起到防火墙的作用。

在内核的实现中，TCP 的状态相当复杂，但是很多都是用户空间所看不到的，但是基本来说，他们遵从 RFC793 协议定义的 TCP 标准状态，我们会在相应的章节详细说明。



你可能已经看到了，从用户角度看，连接跟踪的实现相当简单。但是，从内核的角度看待这个问题，可能就稍稍有点不同了。下面我们看个例子，观察在/proc/net/ip_conntrack 里面，连接的状态是怎么变化的。第一个状态是受到连接上的第一个 SYN 报文。

```
tcp      6 117 SYN_SENT src=192.168.1.5 dst=192.168.1.35 sport=1031 \
     dport=23 [UNREPLIED] src=192.168.1.35 dst=192.168.1.5 sport=23 \
     dport=1031 use=1
```

通过上面的连接情况我们能够看到，SYN_SENT 标识已经被设置但是还没有收到对端的回复报文（UNREPLIED）。当我们在相反方向上受到另一个报文的，内部状态会变成另外一个。

```
tcp      6 57 SYN_RECV src=192.168.1.5 dst=192.168.1.35 sport=1031 \
     dport=23 src=192.168.1.35 dst=192.168.1.5 sport=23 dport=1031 \
     use=1
```

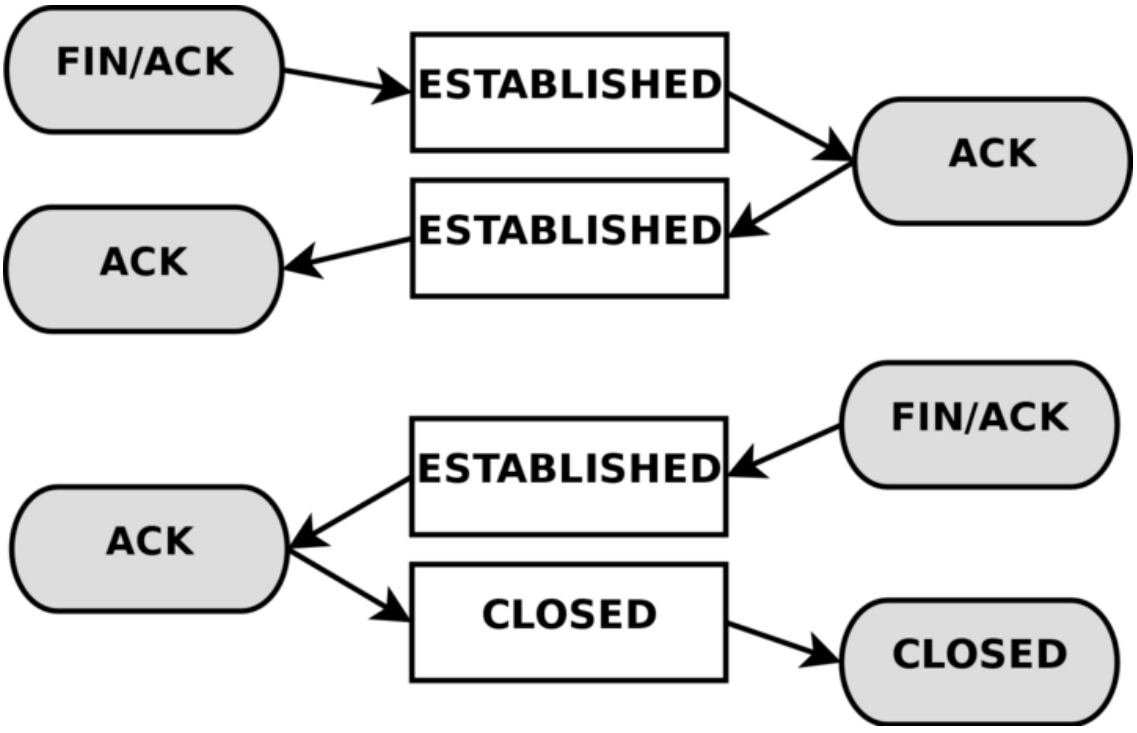
现在我们已经收到回复的 SYN/ACK 报文了，那么状态再次发生变化，我们可以看到是 SYN_RECV。SYN_RECV 状态告诉我们 SYN 报文已经被传递同时我们也收到对端发送回来的 SYN/ACK。另外这个连接现在在双方都已经看到报文收发了，当然这个只是我们的猜测。一旦我们收到最后的 ACK 报文，最后一个状态就达到了。

```
tcp      6 431999 ESTABLISHED src=192.168.1.5 dst=192.168.1.35 \
     sport=1031 dport=23 src=192.168.1.35 dst=192.168.1.5 \
     sport=23 dport=1031 [ASSURED] use=1
```

在最后一个例子里，我们得到了 3 次握手里面的最后 ACK 报文，所以连接变成 ESTABLISHED 状体啊。一般而言，现在的连接就是有保障的了。

一个连接也有可能进入 ESTABLISHED 状态，但却不是有保障的。在我们启动 tcp-window-tracking 补丁之后，我们就可能尝试重新恢复会话。

一个连接关闭的时候，他会经历下面的状态变迁。



你可能已经看出来，这个连接只有在收到最后一个 ACK 的时候才真正关闭。需要指出的是，这张图描述的仅仅是正常情况下如何关闭一个连接。一个

连接也有可能被 RST 关闭，或者说一个连接被拒绝的时候，我们会立刻关闭连接。

当一个 TCP 连接关闭的时候，这个连接会进入 TIME_WAIT 状态，这个状态的默认时间是 2 分钟。这样确保报文在连接关闭的时候，哦能够完全通过我们的规则检查。另外就是可以看作一个缓冲，这样报文即使在某几个拥塞的路由器里面也能够最终到达我们防火墙或者连接的另一端。

假如一个连接被 RST 复位，那么状态直接变成 CLOSE。在这个状态下，我们还有 10 秒钟处理这个链接上的报文。RST 报文不会等待任何确认，直接关闭这个连接。当然还有其他的一些我们没有谈及的状态，完整列表以及他们的超时时间请看下表。

Table 7-2. Internal states

表 7-2 内部状态

State	Timeout value
NONE	30 minutes 30 分钟
ESTABLISHED	5 days 5 天
SYN_SENT	2 minutes 2 分钟
SYN_RECV	60 seconds 60 秒钟
FIN_WAIT	2 minutes 2 分钟
TIME_WAIT	2 minutes 2 分钟
CLOSE	10 seconds 10 秒钟
CLOSE_WAIT	12 hours 12 小时
LAST_ACK	30 seconds 30 秒钟
LISTEN	2 minutes 2 分钟

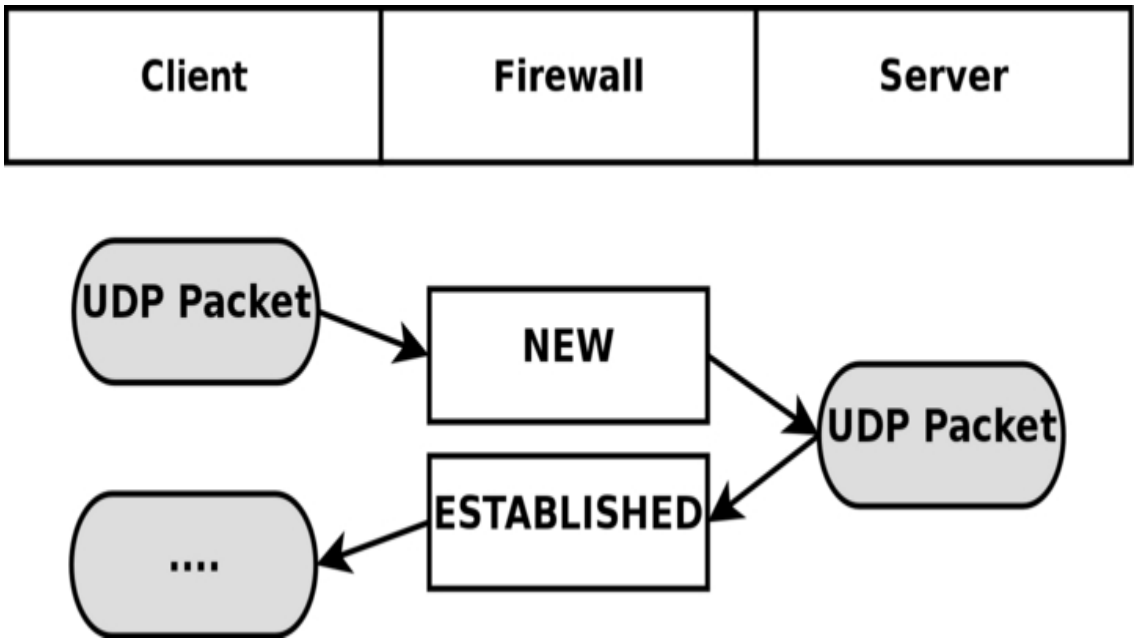
这些值都是可定义的，我们能够通过编译内核来修改这些值，另外我们也可以通过 proc 文件系统来修改，具体在这个目录下

/proc/sys/net/ipv4/netfilter/ip_ct_tcp_*。但是需要说明的是，这些默认值都是在实践中得到充分验证的，假如你实在需要修改，这些值的单位是秒。

状态机制在用户空间里的部分不会查看TCP包的标志位（也就是说TCP标志对它而言是透明的）。如果我们想让NEW状态的包通过防火墙，就要指定NEW状态，我们理解的NEW状态的意思就是指SYN包，可是iptables又不查看这些标志位。这就是问题所在。有些没有设置SYN或ACK的包，也会被看作NEW状态的。这样的包可能会被冗余防火墙用到，但对只有一个防火墙的网络是很不利的（可能会被攻击哦）。那我们怎样才能不受这样的包的影响呢？你可以使用未设置SYN的NEW状态包 里的命令。还有一个办法，就是安装patch-o-matic里的tcp-window-tracking扩展功能，然后设置 /proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_loose 为0，这样我们就能包防火墙会把出SYN之外的所有NEW报文丢弃。

UDP连接

UDP 连接本身是无状态的。他们不存在连接的建立和关闭的概念，同时大部分 UDP 报文都是无序的。接收到的两个 UDP 报文先后顺序不等同于他们的发送时间。但是尽管如此，我们能够能够在内核里面设置连接的状态，下面我们就看看连接跟踪系统是怎么实现的。



正如你看到的，也就是说从用户的观点来看，UDP 连接的建立和 TCP 连接基本是一致的。其实他们的内部实现，连接的信息却存在较大的差异。但是本质上是一样的，我们首先看看初始 UDP 报文发送后，连接情况。

```
udp       17 20 src=192.168.1.2 dst=192.168.1.5 sport=137 dport=1025 \
      [UNREPLIED] src=192.168.1.5 dst=192.168.1.2 sport=1025 \
      dport=137 use=1
```

从前两个值可知，这是一个 UDP 包。第一个是协议名称，第二个是协议号，第三个是此状态的生存时间，默认是 30 秒。接下来是包的源、目地址和端口，还有期待之中回应包的源、目地址和端口。[UNREPLIED]标记说明还未收到回应

```
udp       17 170 src=192.168.1.2 dst=192.168.1.5 sport=137 \
      dport=1025 src=192.168.1.5 dst=192.168.1.2 sport=1025 \
      dport=137 [ASSURED] use=1
```

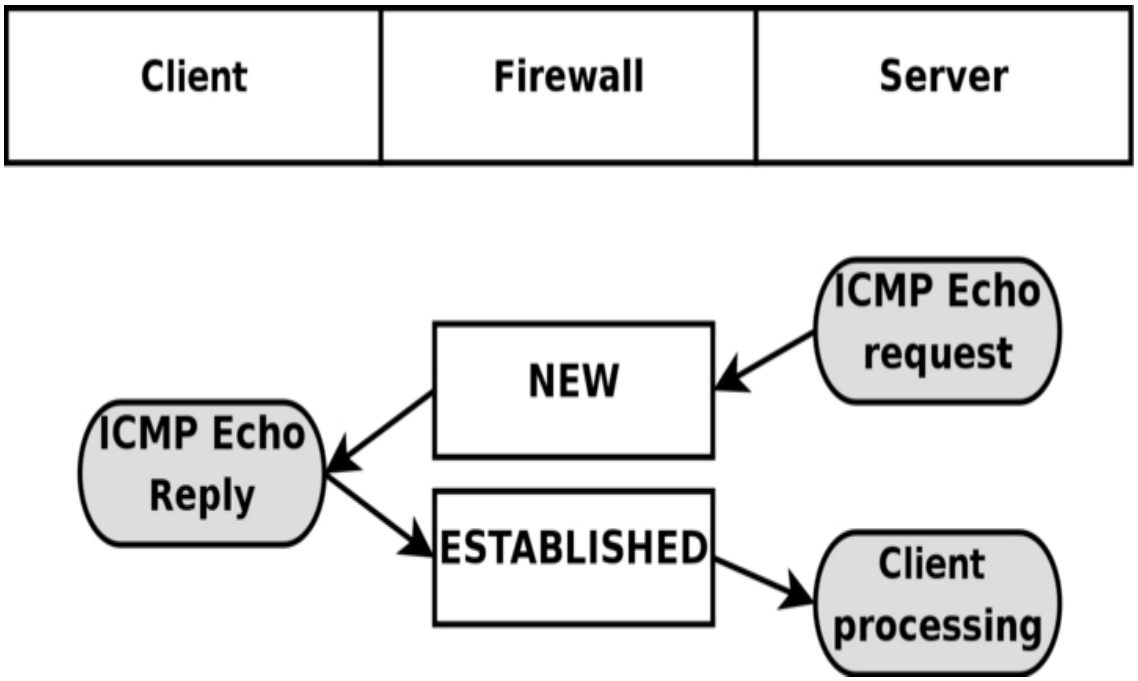
到此为止，服务器已经对收到的报文发送了一个回复，这样这个连接状态就可以认为是 ESTABLISHED，虽然这个状态我们在上面看不出来。最主要的差别就 UNREPLIED 是标记被清空了，另外就是默认的超时被设置为 180 秒-但是在这个例子中他们被减为 170 秒，呵呵，在 10 秒后，这个值就会变为 160 秒，还有一件事情我们忘记了，那就是 ASSURED 标记。如果这个标记向北设置上，那么我们只需要对处于 NEW 状态的连接发送一个合法的回应报文。

```
udp       17 175 src=192.168.1.5 dst=195.22.79.2 sport=1025 \
      dport=53 src=195.22.79.2 dst=192.168.1.5 sport=53 \
      dport=1025 [ASSURED] use=1
```

现在这个连接就是有保障的了。如果这个连接在 180 秒内都没有被使用，那么它就会超时。180 秒钟看起来比较短，但是对于大部分应用而言，这个值足够了。每一个报文通过这个连接穿透防火墙的时候，这个值都会被重置。

ICMP连接

ICMP 报文更谈不上算是一个有状态的流了，他们仅仅用于控制并却从来不建立任何连接，但是还是有 4 种 ICMP 的报文会期望收到应答报文，所以他们会有两种状态。这两种状态分别是 NEW 和 ESTABLISHED。我们所说的四种特殊 ICMP 报文就是 Echo/Timestamp/Information/Fianly address mask，另外 timestamp 和 information request 都已经被废弃了，所以这些的请求报文有可能被丢弃。但是 Echo 和 Address mask 这两种报文都应该得到足够的支持。下面我们来看看他们是怎么工作的。



上图你可以看出来，主机对着目标机发送一个 echo 请求，防火墙就会为这个报文新建一个连接，目标主机会送一个应答给防火墙，这样防火墙就认为这是一个处于 ESTABLISHED 状态的连接了。当地一个报文被发出去，连接状态如下：

```
icmp     1 25 src=192.168.1.6 dst=192.168.1.10 type=8 code=0 \
     id=33029 [UNREPLIED] src=192.168.1.10 dst=192.168.1.6 \
     type=0 code=0 id=33029 use=1
```

这个格式和我们看到的TCP/UDP有点差异，除了我们在TCP/UDP里面看到的，这儿还有其他三个新的字段，分别是type/code/ID。Type和code分别是ICMP的类型和编码，具体的可以看附录的ICMP types一节。最后是ID字段，每一个ICMP保额为你都有一个属于自己的ID，当我们收到一个ICMP报文的时候，我们就通过ID来关联应答。

下一个字段，就是我们熟悉的 UNREPLIED 标记。还和前面一样，这个标记告诉我们的信息是我们只在一个方向上有流量。最后，我们看到了期望的应答报文信息，里面的地址信息以及 type/code/ID 都做了相应的修改。

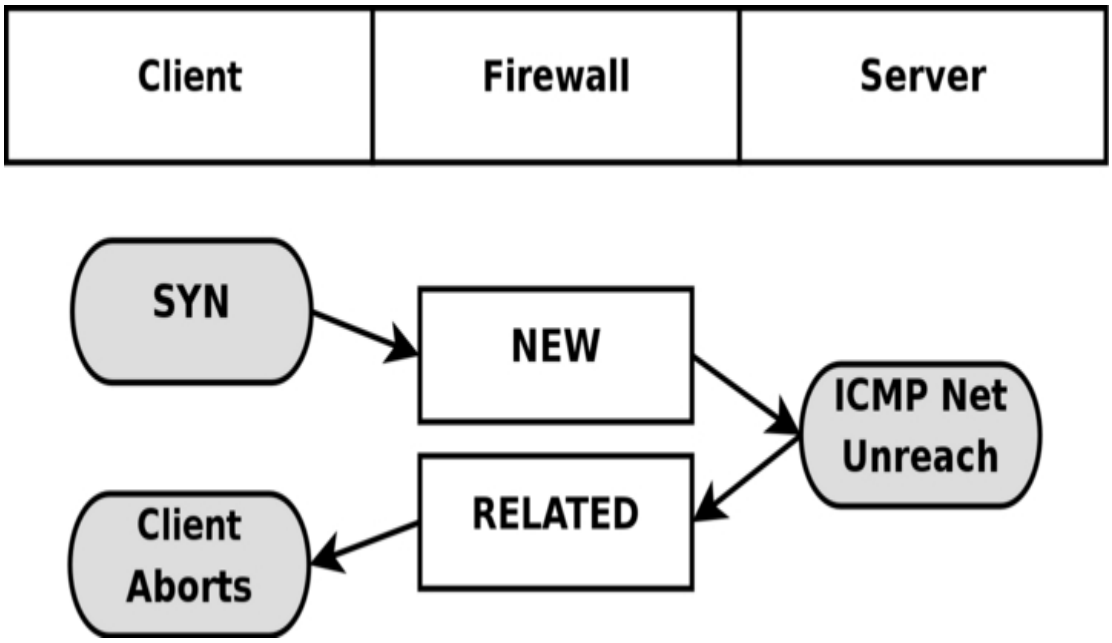
还和前面一样，应答报文被认为是 ESTABLISHED 的状态。但是我们可以确保在 ICMP 应答报文之后不会有更多的报文，所以一旦收到应答报文，这个连接就终止。

在上面的每一个例子里面，请求报文都被看作 NEW，而应答报文都被看作 ESTABLISHED。那么我们看深入的看一下这个情况吧，当防火墙看到一个请求报文，它认为状态是 NEW 的，接着主机针对这个请求回复一个应答，这样防火墙就认为这个连接的状态已经是 ESTABLISHED 了。

主要应答包必须符合连接跟踪表的特定信息才会被认为是 ESTABLISHED，每一种报文都是这样的。

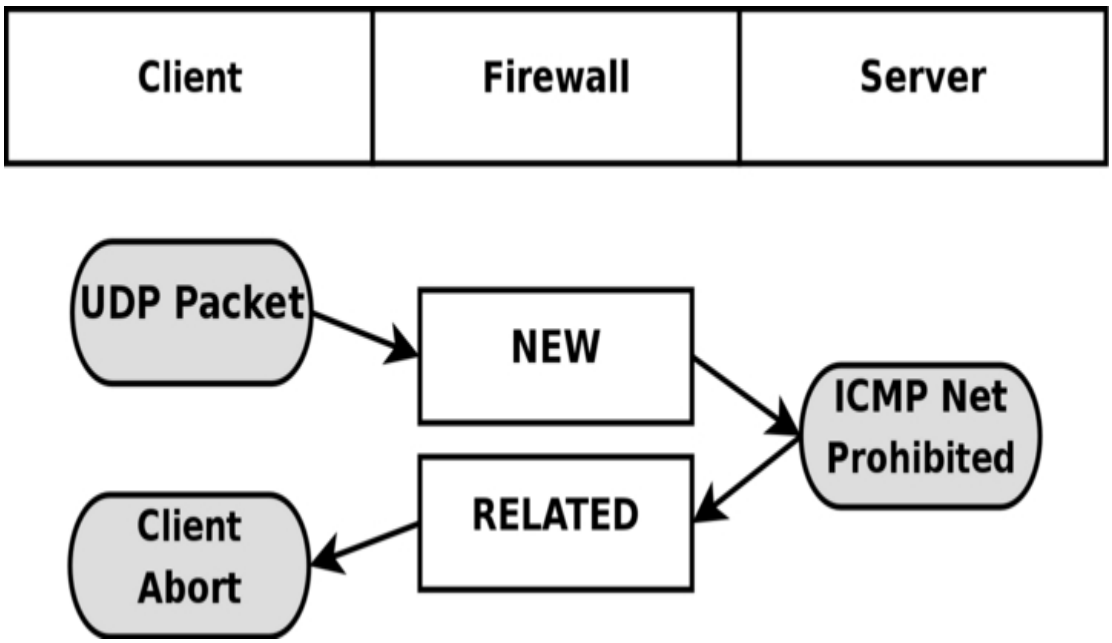
ICMP 请求的默认超时时间是 30 秒钟，你也可以在 /proc/sys/net/ipv4/netfilter/ip_ct_icmp_timeout 里面修改，这个值其实是比较理想，它应该能够满足大部分情况。

ICMP 的另外一个很重要的功能就是用于连接的错误通知，因为这个原因，ICMP 的应答报文更多时候会被当作 RELATED 关联到一个已经存在的连接。一个简单的例子就是 ICMP 主机不可达或者网络不可达。假如我们尝试去连接一台主机或者网络，但是他们却可能已经关闭了，这样最后一台路由器就会给我们回复一个 ICMP 错误消息，这样这个 ICMP 回应消息就被认为是处于 RELATED 状态的。我们看下图：



上面的例子中，我们发送一个 SYN 报文，防火墙就会新建一个连接。但是目的网络或者主机却是不可到达的，因此路由器会返回一个 ICMP 的错误消息。这样连接跟踪代码就会认为这个 ICMP 消息是 RELATED 的。多亏我们之前建立的连接，这样 ICMP 回应报文就能够被发送给正确的接收者，从而让他从容的关闭这个连接。同时，防火墙也会关闭这个连接，因为它知道这是一个错误消息。

对于 UDP 连接而言，如果他们碰到了错误报文，行为也是一样的。所有针对 UDP 连接的 ICMP 错误报文都被认为是 RELATED。



这次一个 UDP 报文发给一台主机，这样我们为这个报文新建一条连接。但是这个网络被人为的禁止访问，所以我们的防火墙就会收到一个 ICMP 错误消息。防火墙知道这个 ICMP 错误消息关联到已经打开的 UDP 连接，所以就给他发送一个错误消息。此时，防火墙也会把连接跟踪表里面的这条连接删除。最终，发送端收到错误消息并且关闭这个连接。

默认连接

在一些特定情况下，连接跟踪不知道如何处理一个未知协议，在我们不认识或者我们不了解这个报文的协议工作机制时候就会发生这种情况。在这些情景里，我们用一个默认当作来处理这个报文。例如 NETBLT、MUX、EGP 等等，默认行为看起来很像 UDP 连接，第一个报文认为是 NEW，收到应答就会设置连接为 ESTABLISHED。

当我们是能默认行为时候，所有这些报文都会有一样的默认超时时间，他们可以通过/proc/sys/net/ipv4/netfilter/ip_ct_generic_timeout 修改。默认时间是 600 秒或者 10 分钟，以便适应你的通信量，尤其是在耗时较多、流量巨大的情况下，比如使用卫星等。

不被跟踪的连接和raw表

当第一次在连接跟踪里面出现的时候，UNTRACK 相当的奇特。简单的说，他主要就是说针对在 raw 表被标记的报文不被跟踪。

Raw 主要也就是为了这个目的而创建的，在这个包里面，你可以针对不想被跟踪的报文打上 NOTRACK 标记。

注意我这儿说的是报文而不是连接，因为这个 mark 是针对每个报文打的，否则我们还得对连接进行某种操作来确定是否需要跟踪。

我们在本章开头已经说了，连接跟踪和状态匹配都是相当消耗系统资源的，因为这个原因，如果不是必须的，最好还是关闭这个功能。

其中一个例子就是在一台流量很大的路由器上面你想针对转发报文做连接跟踪，而对于路由报文则不管。这样你就可以在 raw 里面对针对非目的地址为本机的报文设置 NOTRACK 标记，我们就可以不对转发报文做连接跟踪，而只对本机的报文设置，从而节约了大量的系统资源。

另外一个例子就是你有一个流量相当大的 web 服务器，你想对 web 连接之外的报文做连接跟踪。这样你就可以在 raw 表里面对 web 流量做 NOTRACK。

当然 NOTRACK 还有一些值得你考虑的问题，例如一个连接被打上 NOTRACK 标记，接着这个连接的相关连接也不会被跟踪。换句话讲连接跟踪和 nat 的 helper 就不能正常工作，ICMP 错误报文也是，你只有手动打开相应的端口。但是我们遇到复杂协议的时候，例如 FTP 或者 SCTP 等等，这样可能就很难控制。

复杂协议和连接跟踪

有一些协议比其他的要麻烦一些，我这么说的意思是当我们对这种协议做连接跟踪的时候，可能很难跟踪正确。这样反面例子就是 ICQ/IRC/FTP 等等，他们这些协议在报文的数据区携带有其他信息，因此我们需要 helper 模块来帮助我们寻找连接关系。

下面是一个简单的复杂协议列表，包括在哪个版本被包含。

Table 7-3. Complex protocols support

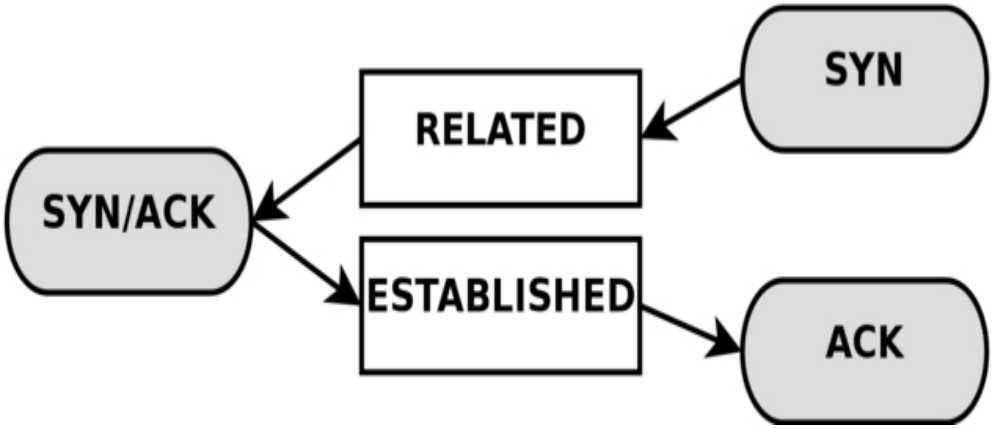
Protocol name	Kernel versions
FTP	2.3
IRC	2.3
TFTP	2.5
Amanda	2.5

- FTP
- IRC
- TFTP

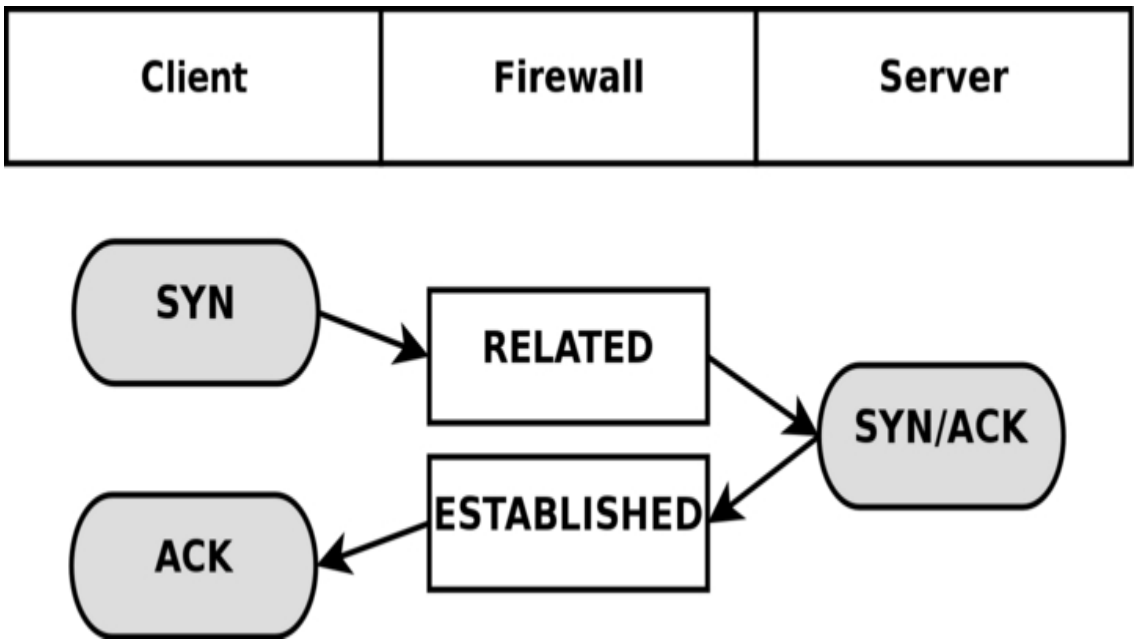
我们首先以 FTP 为例子为介绍，FTP 协议首先打开一个称之为控制会话的连接，当我们通过这个会话发送命令的时候，其他的端口打来来完成剩余的数据传输。这些数据连接可以通过两种方式完成，分别是主动和被动。当一个连接是主动类型时，FTP 客户端告诉服务器自己的地址和端口，然后自己打开响应的端口，接着服务器用一个随机端口最源端口来连接客户端的指定端口。

问题在于防火墙不知道他们额外的连接，因为他们是通过报文的数据区来协商这些信息的，所以防火墙就不知道服务器要连接客户端的哪一个端口。

解决方案就是增加一个 helper，它主要就是检查控制连接的数据，发现一个要建立一个新的连接，就把这个连接关联到主链接。这样服务器就能够被跟踪到了，我们看下图。



被动 FTP 工作在相反的模式，FTP 客户端告诉服务器它要什么数据，服务器告诉客户端地址和连接端口。客户端收到信息就通过自己的 20 端口连接到服务器的特定端口。如果 FTP 服务器在防火墙后面，或你对用户限制的比较严格，只允许他们访问 HTTP 和 FTP，而封闭了其他所有端口，为了让在 Internet 是的客户机能访问到 FTP，也需要增加上面提到的 helper。下面是被动模式下 data 连接的建立过程：



内核里面已经有一些helper可以用了，更为具体的是，FTP和IRC协议都已经能够正常工作了。假如你在内核里面没有找到你所需要的helper，那么建议你到patch-o-matic里面查找一下。假如这个里面也没有，那么你还有其他一些方法，首先下载最新的iptables代码，看里面是否包含；另外就是到mail list里面问一下是否已经有人做好了。假如这些方法都没有得到你想要的，那么你就只有阅读一下Rusty Russell's Unreliable Netfilter Hacking HOW-TO（附件的其他资源列表里面），然后自己动手写一个。

连接跟踪的 helper 可以编译进内核，也可以编译成模块，如果你编译成模块，那么可以用过下面的命令加载这些模块。

```
modprobe ip_conntrack_ftp
modprobe ip_conntrack_irc
modprobe ip_conntrack_tftp
modprobe ip_conntrack_amanda
```

需要注意的是连接跟踪并不处理 NAT，所以你需要更多的模块来完成 NAT 功能。例如你想做 NAT 并且对 FTP 做连接跟踪，那么你除了 FTP 的连接跟踪模块，还需要 NAT 模块。所有 NAT helper 以 ip_nat_开头，这是命名规范。例如 FTP NAT 模块的 helper 应该命名为 ip_nat_ftp，IRC 模块的 helper 应该是 ip_nat_irc。连接跟踪有同样的命名规范，因此 IRC 的连接跟踪 helper 名字为 ip_conntrack_irc,而 FTP 连接跟踪的 helper 名字为 ip_conntrack_ftp。

下章预览

本章介绍了 netfilter 里面状态是如何工作的以及不同的协议状态如何变化。本章还讨论了怎么修改他们的行为，以及复杂协议连接跟踪是如何做的，在本节我们还列举了不同连接模块的 helper。

下一章我们将要讨论怎么保存和恢复规则集，当然任何一个东西都是有人赞同有人反对的，我们分别讨论。

第八章 保存和恢复规则集

Iptables 软件包里面有很多非常有用的工具，特别是在你处理大量规则集的时候，iptables-save 和 iptables-restore 可能就很有用处，他们把规则保存到一个特殊格式的文件里面。

Iptables-restore 可以和脚本语言一起混用，方法就是你把脚本的输出重定向到 iptables-restore 的输入。假如你有大量的规则，那么这个方法相当的快。例如 make_rules.sh|iptables-restore.

速度方面的考虑

用 iptables-save 和 iptables-restore 的一个重要原因就是他们能够加速恢复和保存大的规则。使用脚本更改规则的问题是，改动每个规则都要调运命令 iptables，而每一次调用 iptables，它首先要把 Netfilter 内核空间中的整个规则集都提取出来，然后再插入或附加，或做其他的改动，最后，再把新的规则集从它的内存空间插入到内核空间中。这会花费很多时间

解决这个问题的武器，就是 iptables-save 和 iptables-restore。Iptables-save 命令用来把规则保存到一个特殊格式的文本文件中，而 iptables-restore 则能够加载这个文件，并且在内核中恢复规则。他们的最大优点就是对于规则的操作只需要一次请求。Iptables-save 能够一次性从内核里面获取全部的规则，并且保持到一个文件里面。而 iptables-restore 则能够一次性把所有规则加载并且同步到内核。换句话说，对于一个很大的规则集，如果用脚本来设置，那这些规则就会反反复复地被卸载、安装很多次，而我们现在可以把整个规则集一次就保存下来，安装时则是一次一个表，这可是节省了大量的时间

如果你需要把大量的规则插入内核，这两个工具是明显的选择。当然，它们也有不足之处，下面的章节会详细说明

恢复的不足

可能你已经猜测了，iptables-restore 能够处理所有的脚步吗？到目前为止，答案是否定的，而且很可能永远都不能。iptables-restore 的主要不足是不能用来做复杂的规则集。例如，我们想在计算机启动时获取连接的动态分配的 IP 地址，然后用在脚本里。这一点，用 iptables-restore 来实现，或多或少是不可能的。

一种可能的办法就是你自己写一个小脚本，这个脚本用于获取 ip 地址，然后把手动写的规则里面的关键字换成刚刚得到的地址了，然后保存为一个临时文件，这个时候再用 iptables-restore 来恢复这个规则。这样虽然很罗嗦但是至少可以解决你的问题。但是我们还是得承认，这个方法很笨。

第二个办法和前面所是的差不多，写一个脚本，这个基本能够输出 iptables-restore 的格式，然后重定向给 iptables-restore 的输入。这个方法在规则集比较多的时候很浪费系统资源。

另一个办法是先装入 iptables-restore 文件，再运行一个特定的脚本把动态的规则装入。其实，这也是较笨的方法。iptables-restore 并不适合于使用动态 IP 的场合，如果你想在配置文件里使用选项来实现不同的要求，iptables-restore 也不适用。

iptables-restore 和 iptables-save 还有一个不足，就是功能不够齐全。因为使用的人不是太多，所以发现这个问题的人也不多，还有就是一些 match 和 target 被引用时考虑不细致，这可能会出现我们预期之外的行为。 尽管存在这些问题，我还是强烈建议你使用它们，因为它们对于大部分规则集工作的还是很好的，只要在规则中别包含那些新的都不知如何使用的 match 和 target。

---

iptables-save

我们前面已经解释了，iptables-save 是一个用来保存当前规则到一个 iptables-restore 可以使用的文件。这个命令实在很简单，参数只有两个，我们看看这个命令的语法。

```
iptables-save [-c] [-t table]
```

-c 参数告诉 iptables 在保存的时候保存包和字节数量。例如我们需要重启，那么这个选项就相当重要了。需要注意的是默认情况下，这个参数不被使用。

-t 参数告诉 iptables 哪一个表的配置需要保存，没有这个参数所以表的配置都会被保存。下面是没有任何规则的 iptables-save 的输出。

```
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:17 2002
*filter
:INPUT ACCEPT [404:19766]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [530:43376]
COMMIT
# Completed on Wed Apr 24 10:19:17 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:17 2002
*mangle
:PREROUTING ACCEPT [451:22060]
:INPUT ACCEPT [451:22060]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [594:47151]
:POSTROUTING ACCEPT [594:47151]
COMMIT
# Completed on Wed Apr 24 10:19:17 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:17 2002
*nat
:PREROUTING ACCEPT [0:0]
:POSTROUTING ACCEPT [3:450]
:OUTPUT ACCEPT [3:450]
COMMIT
# Completed on Wed Apr 24 10:19:17 2002
```

我们来稍微解释一下上面的配置。

以#开头表示这行是注释。

每一个表都以*标志，例如*mangle。

接着在每一个表里面，我们都有链和规则，一个链看起来像如下的格式:

<链名><链策略>[<报文计数>:<字节计数>]，链名可能是 PREROUTING，而默认策略可能是我们前面说过的 ACCEPT。最后报文和字节计数和 iptables -L -v 输出是一致的。

每一个表的结束都用一个 COMMIT 关键字，COMMIT 关键字表明此刻我们应该把所有规则发送到内核。

上面是最简单的例子，因此我想用一个简明的例子来说明这个过程可能更好。其中包含一个非常小的规则集*Iptables-save ruleset*。iptables-save的输出如下：

```
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:55 2002
*filter
```

```
:INPUT DROP [1:229]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth1 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
COMMIT
# Completed on Wed Apr 24 10:19:55 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:55 2002
*mangle
:PREROUTING ACCEPT [658:32445]
:INPUT ACCEPT [658:32445]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [891:68234]
:POSTROUTING ACCEPT [891:68234]
COMMIT
# Completed on Wed Apr 24 10:19:55 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:55 2002
*nat
:PREROUTING ACCEPT [1:229]
:POSTROUTING ACCEPT [3:450]
:OUTPUT ACCEPT [3:450]
-A POSTROUTING -o eth0 -j SNAT --to-source 195.233.192.1
COMMIT
# Completed on Wed Apr 24 10:19:55 2002
```

你可能看到了，每一个命令前都有字节和报文统计值，这是因为我们用了-c 参数。除了有计数器，其他的都和普通的脚本一样。现在的问题是怎么把输出保存到文件中。非常简单，既然使用 linux，你应该早就知道了，用重定向就可以了。

```
iptables-save -c > /etc/iptables-save
```

上面的命令就是把所有的 iptables 的配置保存到/etc/iptables-save 文件里。

---

iptables-restore

Iptables-restore 命令用来恢复通过 iptables-save 保存的配置。不幸的是，它只能从标准输入接受输入，而不能从文件接受。下面是它的使用语法：

iptables-restore [-c] [-n]

-c 参数表明我们要恢复我们之前保存的字节以及报文统计，这个参数还有另外一种长格式写法—counter。

-n 参数表明不会覆盖之前已经存在的配置，默认情况下 iptables-restore 会刷新和破坏所有之前已经存在的配置。-n 参数当然也可以通过—noflush 来替代，他们是一样的。

我们有很多种办法来加载配置文件，这儿我们只想介绍一种最简单也是最常见的办法。

```
cat /etc/iptables-save | iptables-restore -c
```

下面这个命令也能够工作。

```
iptables-restore -c < /etc/iptables-save
```

这个命令会把/etc/iptables-save 文件里面的内容读出，然后通过标准输入传递给 iptables-restore.

这样规则就能够被正确的加载到内核并且应该一切工作正常，假如没有，那应该是你使用有问题。

下章预览

本章我们一定程度上讨论了 iptables-save 以及 iptables-restore 这两个程序以及他们的用法。这两个应用程序都在 iptables 软件包里面能够找到，他们主要都用来快速保存和恢复大量规则。

下一章我们主要看一下 iptables 规则的语法以及如何写 iptables 的规则，并且我们还描述了什么是好的编写风格。

第九章 怎么编写规则

本章以及接下来的三章主要都是讨论如何编写你自己的规则。规则可以这么描述，它能够指导防火墙何时阻塞或者允许链上的连接或者报文通行。你所书写的每一行就被认为是一个规则，我们将会讨论基本的 matches/target 以及如何使用他们，另外我们如何构建自定义的链。

本章我们将讨论在 raw 表里面怎么创建一个可以被 iptables 用户空间程序接受的命令，以及不同的表的情况，还有我们能够发送给 iptables 的命令。在这之后我们，我们会用一章来介绍所有可用的 matches，最后再深入理解每一种 target 和 jump。

Iptables命令的基础

我们前面已经说过了，一条规则占用一行，内核通过这条规则来决定怎么处理报文。假如所有的判定条件或者 matches 被匹配上，那么我们执行 target 或者 jump 指令。通常我们所写的规则看起来有如下的格式：

iptables [-t *table*] command [match] [target/jump]

没有人强制要求 target 必须放在命令的最后面，但是你放在最后面符合一般人阅读习惯。不管怎么说，我们所写的或者所看到的大部分都遵从这个格式，因此，这样你看别人的脚步，就很容易理解了。

假如你使用了非标准 target，你需要指定你操作哪一张表，但是通常情况下我们不需要指定表名称，因为默认的操作是在 filter 表上面。也没有必要非得在这里指定表名，实际上几乎可在规则的任何地方。当然，把表名在开始处已经是约定俗成的标准。

命令总是放在开头或者紧挨着表名，我们用命令告诉应用程序我们想干什么。例如增加一个规则，在链尾增加一个规则或者是删除一条规则，下面我们还会说这个问题。

match 细致地描述了包的某个特点，以使这个包区别于其它所有的包。在这里，我们可以指定包的来源 IP 地址，网络接口，端口，协议类型，或者其他什么。下面我们将会看到许多不同的 match

最后我们说说 target，假如一个报文匹配所有的 matches，那么 target 就告诉这个报文应该做什么。例如让内核把报文送到我们自定义的链上，我们也可以告诉内核直接丢掉这个报文，甚至可以让内核给发送者回复一个特定报文。在本段的后续章节，我们会详细的论述。

表（Tables）

-t选项指明你准备操作哪个表，默认情况下，filter表会被使用。注意我们下面介绍的内容只是遍历表和链一个摘要。

表 9-1. 表（Tables）

Table	Explanation
nat	nat 表的主要用处是网络地址转换，即 Network Address Translation，缩写为 NAT。做过 NAT 操作的数据包的地址就被改变了，当然这种改变是根据我们的规则进行的。属于一个流的包只会经过这个表一次。如果第一个包被允许做 NAT 或 Masqueraded，那么余下的包都会自动地被做相同的操作。也就是说，余下的包不会再通过这个表，一个一个的被 NAT，而是自动地完成。这就是我们为什么不应该在这个表中做任何过滤的主要原因，对这一点，后面会有更加详细的讨论。PREROUTING 链的作用是在包刚刚到达防火墙时改变它的目的地址，如果需要的话。OUTPUT 链改变本地产生的包的目的地址。POSTROUTING 链在包就要离开防火墙之前改变其源地址。
mangle	这个表主要用来 mangle 数据包。我们可以改变不同的包及包头的内容，比如 TTL，TOS 或 MARK。注意 MARK 并没有真正地改动数据包，它只是在内核空间为包设了一个标记。防火墙内的其他的规则或程序（如 tc）可以使用这种标记对包进行过滤或高级路由。这个表有五个内建的链： PREROUTING，POSTROUTING， OUTPUT，INPUT 和 FORWARD。PREROUTING 在包进入防火墙之后、路由判断之前改变包，POSTROUTING 是在所有路由判断之后。 OUTPUT 在确定包的目的之前更改数据包。INPUT 在包被路由到本地之后，但在用户空间的程序看到它之前改变包。FORWARD 在最初的路由判断之后、最后一次更改包的目的之前 mangle 包。注意，mangle 表不能做任何 NAT，它只是改变数据包的 TTL，TOS 或 MARK，而不是其源目地址。NAT 是在 nat 表中操作的。
filter	Filter 明确的就是用来完成报文过滤的，和操作其他表一样，我们在 filter 里面可以丢弃/记录/接受/拒绝报文。FORWARD 链主要用来处理非本机报文，INPUT 链处理所有目的地址为本机的报文，而 OUTPUT 则处理所有本机发送的报文。
raw	Raw 表和它的链都在 netfilter 的模块之前，他主要就是用来完成 NOTRACK 功能，如果我们想使用这个功能，需要 2.6 以及之后的内核。Raw 表里面有 PREROUTING 和 OUTPUT 两条链，在这两条链上他们早于 netfilter 处理报文。PREROUTING 用于到本机的所有报文，而 OUTPUT 链用于本机发送的所有报文。

上面大致介绍了可用的 4 张表，他们完全是为了不同的目的而构建的，另外你还需要知道每一条链是做什么的。假如你没有理解他们的用法，就可能会在防火墙上留下漏洞，给人以可乘之机。在章节表和链 中，我们已详细地讨论了这些必备的的表和链。如果你没有完全理解包是怎样通过这些表、链的话，我建议你回过头去再仔细看看

命令（Commands）

本节我们会覆盖所有不同的命令（Commands）以及用他们能够做什么。命令（Commands）告诉 Iptables 要对我们的规则做什么，通常情况下，我们也许是增加或则删除一些信息，下列命令（Commands）我们可以使用。

Table 9-2. Commands

Command	-A, --append
Example	iptables -A INPUT ...
Explanation	这个命令在链尾追加一条规则，换句话讲这条规则会被放置在规则集的尾部，也就是最后才被检查，除非你后来又追加了另一条规则。☺
Command	-D, --delete
Example	iptables -D INPUT --dport 80 -j DROP, iptables -D INPUT 1
Explanation	这条命令删除链里面的一条规则，我们有两种方式输出一条规则，第一是输入一条完整的规则，另外就是通过序号删除。假如你用第一种方法删除，那么你输入的规则必须和你想删除的规则完全一致。如果你想用第二种方法来删除规则，你必须制定删除的序号，链的序号从顶部开始从 1 编号。
Command	-R, --replace
Example	iptables -R INPUT 1 -s 192.168.0.1 -j DROP
Explanation	这条命令替换指定位置的旧规则，它和 delete 工作方式一样的，但是不同在于它不完全删除这条规则，而是替换上一条新的规则。这个主要用于你做一些测试，实际工作中很少使用。
Command	-I, --insert
Example	iptables -I INPUT 1 --dport 80 -j ACCEPT
Explanation	在链里面插入一条规则，这条规则会在我们指定的地方插入。换句话讲，上面的这条命令会把这条规则插入到第一个位置，所有它就是第一条规则。
Command	-L, --list
Example	iptables -L INPUT
Explanation	这条命令列出指定链的所有规则，在上面的例子里面，我们会列出 INPUT 链里面的所有规则。如果你不指定链名称，那么就列出这张表里面所有的链上规则，上面没有指定表名称，默认就是 filter 表，如果想查看特定表用-t 参数。他的输出还受到其他参数的影响，例如-n 和-v。
Command	-F, --flush
Example	iptables -F INPUT
Explanation	这个命令情况链上的所有规则，它等同于逐条删除所有规则，只是它更快。没有指定参数的话，他会把指定表上面所有链的规则

	删除干净。
Command	-Z, --zero
Example	iptables -Z INPUT
Explanation	这个命令告诉应用程序情况链上的所有计数器值，没有指定链就是所有链。假如你用-v 参数加上-L 查看，你就会在开头看到统计值。清空这个计数你可以用-Z 参数。-Z 参数的使用和-L 一个样，除了-Z 没有输出。当然-L 和-Z 可以合用，这样首先列出规则，但是报文的计数都被清空。
Command	-N, --new-chain
Example	iptables -N allowed
Explanation	这个命令让我们在指定的表上面创建一条新的用户自定义链，在上面的例子里面，我们创建了一条叫做 allowed 的用户自定义链。请注意，名字不能和系统内建的重合。
Command	-X, --delete-chain
Example	iptables -X allowed
Explanation	这条命令会从表上删除指定的链，要想删除这个链，这儿必须没有任何规则关联到这个链。换句话讲，你要删除或者替换所有相关的规则，然后才能删除这条链。另外这个命令只能删除用户自定义链，而且没有指定链名称的时候，会尝试把指定表中所有的用户自定义链删除。
Command	-P, --policy
Example	iptables -P INPUT DROP
Explanation	为链设置默认的 target（可用的是 DROP 和 ACCEPT），这个 target 称作策略。所有不符合规则的包都被强制使用这个策略。只有内建的链才可以使用规则。但内建的链和用户自定义链都不能被作为策略使用，也就是说不能象这样使用：iptables -P INPUT allowed（或者是内建的链）。
Command	-E, --rename-chain
Example	iptables -E allowed disallowed
Explanation	这条命令会把这个链的第一个名字改成第二个，例如上面，我们会把 allowed 改成 disallowed。注意我们所修改的仅仅是名称罢了，所有的规则都不会有任何变化。

除了你想看内建的帮助或者看命令的版本，否则都应该输入完整的命令。想获取帮助，你可以使用-v 参数，而想得到帮助，请使用-h 参数。如果你想获得帮助，请在-h 后面加上你想获取帮助的命令名称。下面我们介绍的部分选项，还有它们的作用

Table 9-3. Options

Option	-v, --verbose
Commands used with	--list, --append, --insert, --delete, --replace
Explanation	这个选项主要和—list 一起使用。假如两个一起用的话，输出信息就包含接口地址/规则选项以及 TOS 的标记。 --list 命令还会列出字节和报文统计。其中计数器是以 K、M、G（这里用的是 10 的幂而不是 2 的幂哦）为单位的。如果想知道到底有多少个包、多少字节，还要用到选项-x，下面会介绍。如果-v 和--append、--insert、--delete 或--replace 连用，iptables 会输出详细的信息告诉你规则是如何被解释的、是否正确地插入等等
Option	-x, --exact
Commands used with	--list
Explanation	使--list 输出中的计数器显示准确的数值，而不用 K、M、G 等估值。注意此选项只能和--list 连用。
Option	-n, --numeric
Commands used with	--list
Explanation	使输出中的 IP 地址和端口以数值的形式显示，而不是默认的名字，比如主机名、网络名、程序名等。注意此选项也只能和--list 连用
Option	--line-numbers
Commands used with	--list
Explanation	--line-number 命令和—list 一起合用，他们用来控制输出数字。用了这个选项后，每一条规则都带有序号，这样我们就很方便的知道每条规则的位置。这条命令只能和—list 合用。
Option	-c, --set-counters
Commands used with	--insert, --append, --replace
Explanation	这个选项可以让我们在创建或者修改规则的时候，设置字节或者报文的计数值。语法大致是这个样子的，--set-counters 20 4000,这个命令设置报文计数值为 20 而字节计数值为 4000.
Option	--modprobe
Commands used with	All
Explanation	此选项告诉 iptables 探测并装载要使用的模块。这是非常有用的一个选项，万一 modprobe 命令不在搜索路径中，就要用到

	了。有了这个选项，在装载模块时，即使有一个需要用到的模块没装载上，iptables 也知道要去搜索。

下章预览

这一章我们主要讨论了 iptables 的基本命令以及简要介绍了 netfilte 的表。这些命令让我们能够对加载进内核的模块做大量的不同操作。

下一章我们就讨论 iptables 以及 netfilter 里面可用的 matches。这一章内容很多，我建议你不要每一个 matches 都研究的很深透，除非你要用他。一个不错的注意是你知道每一个 match 的基本用法，在你需要的时候在深入学习。

第十章 Iptables的匹配（matches）

本章我们主要说说 matches，我粗略的把他们分为五类。第一个是通用匹配，它可以用在任何规则里，第二类的只能用于 TCP 的 TCPmatches，第三类是能够 UDP 的 UDP matches，第四类是只能用于 ICMP 的 ICMP matches，第五类是一些特殊匹配，例如基于状态(state)，所有者(owner)或者限制(limit)等等。最后一类还会被细化为更小的 matches，尽管他们本身并没有什么不同，我这么划分只是为了让你更容易理解罢了。

通过我们前面几章的介绍，你可以已经明白了。Match 其实就是一个判断一个报文 true/false 的条件。一条规则可以包含多条 matches，例如我们想匹配从本地网络的某台主机发出的报文，不仅仅如此我们还想细化到它从这台主机的哪个端口上发出。我们接着可以通过 matches 的结果来告诉本条规则是否对这个报文应用相应的 target 挥着 jump。假如一条规则里面的任何一个 match 不匹配，整条规则都失败，例如除了源地址外其他都满足条件，我们还是认为本条规则匹配失败，接着用下一条规则进一步匹配。假如所有的 matches 都满足，那么这条规则对应的 target 或者 jump 就会被调用。

通用匹配

本节主要介绍通用匹配。通用匹配就是不管你处理什么协议或者你加载了什么扩展匹配（matches），都能够使用。另外还有一些 match 需要有特定前提才能够使用，例如 TCP match，我们必须首先通过—protocol 制定是 TCP 才能接着进行 TCP 协议的匹配。下面所列的都是通过匹配。

表 10-1.通用匹配

Match	-p, --protocol
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp
Explanation	这个匹配器（match）主要用来检查特定的协议，例如 TCP/UDP/ICMP 等等，它主要有以下几种使用方式： 1. 制定协议的名称，它必须在/etc/protocols 里面定义，不然就会报错。 2. 2.你可以指定一个整数值，例如 ICMP 就是 1，TCP 就是 6 而 UDP 是 17. 3. 另外你可以指定为 ALL，ALL 表示仅仅匹配 ICMP/TCP/UDP 协议，这个是默认配置，数值为 0. 4. 可以是协议列表，以英文逗号为分隔符，如：udp,tcp 5. 最后我们还可以对协议取反，例如!TCP 表示匹配 UDP/ICMP，当然从这儿我们也可以看到取反只能针对 TCP/UDP/ICMP 协议。
Match	-s, --src, --source
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -s 192.168.1.1
Explanation	这个匹配器（match）主要用来匹配报文的源地址，它不仅可以匹配单台主机，还以匹配一个网络。 1、单个地址，如 192.168.1.1，也可写成 192.168.1.1/255.255.255.255 或 192.168.1.1/32 2、网络，如 192.168.0.0/24，或 192.168.0.0/255.255.255.0 3、在地址前加英文感叹号表示取反，注意空格，如--source ! 192.168.0.0/24 表示除此地址外的所有地址 4、缺省是所有地址
Match	-d, --dst, --destination
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -d 192.168.1.1
Explanation	匹配报文的目的地址，用法和源地址匹配一致。
Match	-i, --in-interface
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -i eth0

Explanation	以包进入本地所使用的网络接口来匹配包。要注意这个匹配操作只能用于 INPUT，FORWARD 和 PREROUTING 这三个链，用在其他任何地方都会提示错误信息。指定接口有一下方法： 1、指定接口名称，如：eth0、ppp0 等 2、使用通配符，即英文加号，它代表字符数字串。若直接用一个加号，即 iptables -A INPUT -i +表示匹配所有的包，而不考虑使用哪个接口。这也是不指定接口的默认行为。通配符还可以放在某一类接口的后面，如：eth+表示所有 Ethernet 接口，也就是说，匹配所有从 Ethernet 接口进入的包。 3、在接口前加英文感叹号表示取反，注意空格，如：-i ! eth0 意思是匹配来自除 eth0 外的所有包。
Match	-o, --out-interface
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A FORWARD -o eth0
Explanation	以包离开本地所使用的网络接口来匹配包。使用的范围和指定接口的方法与--in-interface 完全一样。
Match	-f, --fragment
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -f
Explanation	用来匹配一个被分片的包的第二片或及以后的部分。因为它们不包含源或目的地址，或 ICMP 类型等信息，其他规则无法匹配到它，所以才有这个匹配操作。要注意碎片攻击。这个操作也可以加英文感叹号表示取反，但要注意位置，如：！-f 。取反时，表示只能匹配到没有分片的包或者是被分片的包的第一个碎片，其后的片都不行。现在内核有完善的碎片重组功能，可以防止碎片攻击，所以不必使用取反的功能来防止碎片通过。如果你使用连接跟踪，是不会看到任何碎片的，因为在它们到达任何链之前就被处理过了。

隐含匹配（Implicit matches）

本节我们主要讨论隐含匹配器（match），他们被自动加载。这也意味着隐含匹配可以直接使用而不需其他任何额外的动作，例如我们可以直接使用—protocol TCP。现在系统里面有三种隐含匹配，分别是：TCP matches，UDP matches 以及 ICMP matches。每一种 matches 都会针对特定的协议有特殊的匹配条件。而与之相对的是显示匹配，它就表示说我们需要明确的用-m 或者—match 来指明我们要做匹配，这部分内容下一章再说。

TCP匹配器（matches）

这些匹配器是协议相关的，他们只能工作于 TCP 报文流。所以我们在使用这个匹配器（match）的时候需要指定协议—protocol TCP。请注意—protocol tcp 必须在 TCP 匹配器（match）的左边，TCP 以及 UDP 还有 ICMP 都是隐含匹配的，我们首先说说 TCP，其他的分别介绍。

表 10-2 TCP 匹配器（matchs）

Match	--sport, --source-port
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp --sport 22
Explanation	基于 TCP 包的源端口来匹配包，端口的指定形式如下： 1、不指定此项，则暗示所有端口。 2、使用服务名或端口号，但名字必须是在/etc/services 中定义的，因为iptables从这个文件里查找相应的端口号。从这可以看出，使用端口号会使规则装入快一点儿，当然，可读性就差些了。但是如果你想写一个包含 200 条或更多规则的规则集，那你还是老老实实地用端口号吧，时间是主要因素（在一台稍微慢点儿地机子上，这最多会有 10 秒地不同，但要是 1000 条、10000 条呢）。 3、可以使用连续的端口，如：--source-port 22:80 这表示从 22 到 80 的所有端口，包括 22 和 80。如果两个号的顺序反了也没关系，如：--source-port 80:22 这和 --source-port 22:80 的效果一样。 4、可以省略第一个号，默认第一个是 0，如：--source-port :80 表示从 0 到 80 的所有端口。 5、也可以省略第二个号，默认是 65535，如：--source-port 22:表示从 22 到 65535 的所有端口 6、在端口号前加英文感叹号表示取反，注意空格，如：--source-port ！ 22 表示除 22 号之外的所有端口；--source-port ！ 22:80 表示从 22 到 80（包括 22 和 80）之外的所有端口。 注意：这个匹配操作不能识别不连续的端口列表，如：--source-port ！ 22, 36, 80 这样的操作是由后面将要介绍的多端口匹配扩展来完成的。
Match	--dport, --destination-port

Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp --dport 22
Explanation	目的端口用来匹配报文的目的端口，用法和源端口一样。
Match	--tcp-flags
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -p tcp --tcp-flags SYN,FIN,ACK SYN
Explanation	匹配指定的 TCP 标记。有两个参数，它们都是列表，列表内部用英文的逗号作分隔符，这两个列表之间用空格分开。第一个参数指定我们要检查的标记（作用就象掩码），第二个参数指定“在第一个列表中出现过的且必须被设为 1（即状态是打开的）的”标记（第一个列表中其他的标记必须置 0）。也就是说，第一个参数提供检查范围，第二个参数提供被设置的条件（就是哪些位置 1）。这个匹配操作可以识别以下标记：SYN， ACK，FIN，RST ，URG，PSH。另外还有两个词也可使用，就是 ALL 和 NONE。顾名思义，ALL 是指选定所有的标记，NONE 是指未选定任何标记。这个匹配也可在参数前加英文的感叹号表示取反。例如： 1、iptables -p tcp --tcp-flags SYN,FIN,ACK SYN 表示匹配那些 SYN 标记被设置而 FIN 和 ACK 标记没有设置的包，注意各标记之间只有一个逗号而没有空格。 2、--tcp-flags ALL NONE 匹配所有标记都未置 1 的包。 3、iptables -p tcp --tcp-flags ! SYN,FIN,ACK SYN 表示匹配那些 FIN 和 ACK 标记被设置而 SYN 标记没有设置的包，注意和例 1 比较一下。
Match	--syn
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -p tcp --syn
Explanation	这个匹配或多或少算是 ipchains 时代的遗留物，之所以还保留它，是为了向后兼容，也是为了方便规则在 iptables 和 ipchains 间的转换。它匹配那些 SYN 标记被设置而 ACK 和 RST 标记没有设置的包，这和 iptables -p tcp --tcp-flags SYN,RST,ACK SYN 的作用毫无二样。这样的包主要用在 TCP 连接初始化时发出请求。如果你阻止了这样的包，也就阻止了所有由外向内的连接企图，这在一定程度上防止了一些攻击。但外出的连接不受影响，恰恰现在有很多攻击就利用这一点。比如有些攻击黑掉服务器之后安装会一些软件，它们能够利用已存的连接到达你的机子，而不要再新开一个端口。这个匹配也可用英文感叹号取反，如：！ --syn 用来匹配那些 RST 或 ACK 被置位的包，换句话说，就是状态为已建立的连接的包。
Match	--tcp-option
Kernel	2.3, 2.4, 2.5 and 2.6

Example	iptables -p tcp --tcp-option 16
Explanation	根据选项匹配包。TCP选项是TCP头中的特殊部分，有三个不同的部分。第一个 8 位组表示选项的类型，第二个 8 位组表示选项的长度（这个长度是整个选项的长度，但不包含填充部分所占的字节，而且要注意不是每个TCP选项都有这一部分的），第三部分当然就是选项的内容了。为了适应标准，我们不必执行所有的选项，但我们可以查看选项的类型，如果不是我们所支持的，那就只是看看长度然后跳过数据部分。这个操作是根据选项的十进制值来匹配的，它也可以用英文感叹号取反。所有的选项都可在Internet Engineering Task Force里找到。

UDP匹配器（matches）

这一节介绍的匹配器只对 UDP 报文有效，请注意我们在用—protocol UDP 参数的时候，UDP 匹配器就被隐含的加载了。UDP 是一种无连接协议，所以在它打开、关闭连接以及在发送数据时没有多少标记要设置，它也不需要任何类型的确认。假如一个 UDP 报文丢了，那它就丢了（也不会有 ICMP 的错误通告）。这也意味着和 TCP 比起来，UDP 的匹配要少得多，但是对于连接跟踪模块而言，他们工作起来基本是一样的。

Table 10-3. UDP matches

Match	--sport, --source-port
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p udp --sport 53
Explanation	基于 UDP 包的源端口来匹配包，端口的指定形式和 TCP matches 中的--sport 完全一样。
Match	--dport, --destination-port
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p udp --dport 53
Explanation	基于 UDP 包的目的端口来匹配包，端口的指定形式和 TCP matches 中的--sport 完全一样。

ICMP匹配器（matches）

这一节我们要说的是 ICMP 的匹配器（matches），需要说明的是这些报文的生存期非常的短暂，他们是 UDP 一样是无连接的，但是生存时间更短。ICMP 协议主要用来错误报告和连接控制之类的，ICMP 不是 IP 的子协议，而是和 IP 并列的一个协议，它主要用来增强 IP 功能和帮助 IP 进行错误处理。ICMP 的报

文头和 IP 很像，但是还存在一些差异。其中最有特色的就是类型字段，通过它我们就知道这个报文的主要作用是什么。例如我们不能访问 IP 地址，那么我们就会得到一个主机不可达的 ICMP 返回包。完整的 ICMP 类型列表，请参见附录的 ICMP types。ICMP 只有一个匹配器（matches），但愿足够了，另外就是前面介绍的通用匹配器我们仍然可以使用。

Table 10-4. ICMP matches

表 10-4. ICMP matches

Match	--icmp-type
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p icmp --icmp-type 8
Explanation	根据ICMP类型匹配包，类型的指定可以使用十进制数值或相应的名字，数值在RFC792 中有定义，名字可以用iptables --protocol icmp --help 查看，或者在附录ICMP类型中查找。这个匹配也可用英文感叹号取反，如：--icmp-type ！ 8 就表示匹配除类型 8 之外的所有ICMP包。要注意有些ICMP 类型已经废弃不用了，还有一些可能会对无防护的主机带来“危险”，因为它们可能把包重定向到错误的地方。  请注意 255 被 netfilter 用来匹配所有的 ICMP 报文，所有如果你想匹配这种类型的报文，那么你将能够获取到所有的 ICMP 报文。

SCTP匹配器（matches）

流控制传输协议（SCTP）是一个相当新的协议，关于 SCTP 的详细介绍请自己查找资料或者看英文版的 iptables 指南。SCTP 通过-p sctp 来加载相关的匹配器。

SCTP 协议是被一些大的电信公司以及交换机/网络制造商发明的，SCTP 特别适合具有高可靠和高吞吐要求的大量并发事务。因为我没有使用过 SCTP，所以本小节也不翻译。

显示匹配（Explicit matches）

显式匹配必须用-m 或--match 装载，比如要使用状态匹配就必须使用-m state。有些匹配还需要指定协议，有些就不需要，比如连接状态就不要。这些状态是 NEW（还未建立好的连接的第一个包）， ESTABLISHED（已建立的连接，也就是已经在内核里注册过的），RELATED（由已经存在的、处于已建立状态的连接生成的新连接），等等。有些匹配还处在开发阶段，或者还只是为了说明 iptables 的强大能力。这说明不是所有的匹配一开始就是实用的，但以后你可能会用到它。随着 iptables 新版本的发布，会有一些新的匹配可用。隐含匹配和显式匹配最大的区别就是一个是跟随协议匹配自动装载的，一个是显式装载的。

地址类别匹配器（match）

地址类别匹配基于地址类别来处理报文，地址类别被内核用来把不同的报文进行分类，通过这个匹配器你就能够根据内核的地址类别对报文进行匹配，需要指明的是这儿的地址列表主要是三层含义，我会给一个简要的介绍，但是更多更具体的内容还是请参见*Linux Advanced Routing and Traffic Control HOW-TO* and *Policy Routing using Linux*.下面把所有的类别都列出来。

表 10-6 地址类别

Type	Description
ANYCAST	任意播：这是一个点到多点的连接类别，通俗的说就是网络里面有许多接收者，但是只有一个能够实际接收到数据，典型的例子是 DNS。你有一个服务器地址，但是实际上这个地址指向多台主机，但是你的报文只会被最近的主机获得。请注意这个在 linux 的 IPV4 里面没有实现。
BLACKHOLE	黑洞：黑洞地址简单的说这个地址专门用于悄悄的删除报文，它在 linux 的路由表里面配置。
BROADCAST	广播：广播就是一个报文发送给特定网络里面的每一个机器，ARP 是一个很好的例子。
LOCAL	本地：本地地址就是我们工作的机器上面任一地址，例如 127.0.0.1 或者其中某一接口地址。
MULTICAST	多播：多播就是一个报文发送给特定的一组订阅者，典型的例子是视频或者声音报文。
NAT	NAT：被内核做过地址转换的报文。
PROHIBIT	禁止：和黑洞的行为模式一样的，只是它要向发送主机回复一个 ICMP 错误消息。
THROW	扔（THROW）：翻译的肯定不准确，它是 linux 内核的一条特殊路由。假如一个报文查找到的是 throw，那么结果和没有查找到路由式样的。正常情况下，它的处理和无路由一样，但是在有策略

Type	Description
	路由的情况下，在另外一个路由表里面可能查找到路由。
UNICAST	单播：一个真实可路由的地址，这是我们看到的最常见类型。
UNREACHABLE	不可到达：这表明是一个我们不知道如何送达的地址，这个报文会被丢弃然后回复一个 ICMP 主机不可达消息。
UNSPEC	UNSPEC：没有定义的地址类型，当然也没有任何含义。
XRESOLVE	解析：这个地址被内核用来把路由的查新放松到用户空间程序，这个可能是想把恶心的查询放到内核外，或者也有可能是有专门的查询程序。Linux 里面没有实现这个功能。

地址类型匹配器（match）通过-m addrtype 加载，加载完之后，下面这些选项就可以使用了。

表 10-7.地址匹配选项

Match	--src-type
Kernel	2.6
Example	iptables -A INPUT -m addrtype --src-type UNICAST
Explanation	这个选项主要用来匹配报文的源地址类型，它既可以是一个单一的地址类别，例如--src-type UNICAST；也可能是几个类型的组合，之间通过逗号隔开，例如--src-type BROADCAST,MULTICAST。当然我们也可以对这些类型取反，但是！符号要放在变量前面，例如！ --src-type BROADCAST,MULTICAST
Match	--dst-type
Kernel	2.6
Example	iptables -A INPUT -m addrtype --dst-type UNICAST
Explanation	这个选项匹配报文的目的地址类别，用法和源地址分类匹配一样。

AH/ESP匹配器（match）

这两个匹配器主要用于 IPSEC 的 AH/ESP 协议，IPSEC 主要用来在不安全的网络上建立安全的隧道，AH/ESP 就是被用来建立这些安全连接的，其实 AH/ESP 是两个独立的匹配器，但是他们的工作方式很相似，所以我们放在一起说。

我不会更多的深入描述 IPSEC，想了解更多信息请看下列链接：

- *RFC 2401 - Security Architecture for the Internet Protocol*
- *FreeS/WAN*

- *IPSEC Howto*
- *Linux Advanced Routing and Traffic Control HOW-TO*

资源当然不局限于上面的，更多的信息自己到网络上面搜寻。

想使用 AH/ESP 匹配器，请使用-m ah 加载 AH 匹配器，或者-m esp 加载 ESP 匹配器。

在 2.2 以及 2.4 的内核里面，linux 用 FreeS/WAN 来实现 IPSEC，但是在 2.5.47 以及的版本里面，Linux 内核直接实现了 IPSEC，这是对 IPSEC 的完全重写。

表 10-8 AH 匹配器选项

Match	--ahspi
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p 51 -m ah --ahspi 500
Explanation	这个匹配器主要匹配 AH 报文的 SPI 蚕食，请注意这儿需要制定我们需要匹配的协议为 51.SPI 利用源和目的地址以及安全密钥来创建一个安全联盟（SA）。安全联盟（SA）能够唯一的识别主机的每一个 IPSEC 隧道。在两个相同的主机之间，通过 SPI 能够区分出不同的连接。利用—ahspi 参数我们能够借助报文的 SPI 来匹配上相应的报文，当然我们还可以通过 a:sign 的格式来匹配一段 SPI，例如 500：520.

Table 10-9. ESP match options

表 10-9 ESP 匹配器选项

Match	--espspi
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p 50 -m esp --espspi 500
Explanation	用法和 AH SPI 一致。

Comment match

这个匹配器主要用来在内核和规则集里面增加注释，这样就能够让人更加容易规则的作用并且易于调试。需要说明的是这并不是一个真正的匹配器，要使用它可以通过-m comment 加载。

Table 10-10. Comment match options

Match	--comment
Kernel	2.6
Example	iptables -A INPUT -m comment --comment "A comment"
Explanation	--comment 选项用来添加实际的注释内容，每一个注释最大长度为 256 个字符。

Connmark匹配器（match）

Connmatk 匹配器（match）用起来很像 MARK/mark 的 target 和 match 结合体。Connmark 匹配器（match）用来匹配一个连接通过 CONNMARK target 设置的 marks，它只有一个参数。

你想用这个来匹配报文，并且在 CONNMARK 设置了报文的 mark 之后才能工作。

表 10-11. Connmark match 选项

Match	--mark
Kernel	2.6
Example	iptables -A INPUT -m connmark --mark 12 -j ACCEPT
Explanation	Mark 选项用来匹配一个连接关联的 mark，这个 mark 可以使用掩码，不使用默认为 0xFFFFFFFFUL。例如你设置了 -mark 1/1,这个时候一个连接的 mark 为 B10001（33），那么这个连接是可以匹配上的，因为 10001&&00001 等于 1。

Contrack匹配器（match）

Conntrack match 是状态匹配的扩展功能，它让我们能够在更细的颗粒度上匹配报文。它让我们具有了直接读取连接跟踪表的能力，而不需要用一个前端层，例如状态匹配。

Conntrack match 的选项比较多，他们都是针对连接跟踪系统里面的不同选项，下面我们分别介绍。

Table 10-12. Conntrack match options

Match	--ctstate
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m conntrack --ctstate RELATED
Explanation	这个 match 用来根据连接跟踪的状态匹配报文状态，它和以前的状态匹配功能是一样的，可以使用的状态如下： • INVALID • ESTABLISHED • NEW • RELATED • SNAT • DNAT 我们一次可以匹配多个状态，但是这些状态要通过逗号隔开。例如-m conntrack --ctstate ESTABLISHED,RELATED。 我们也可以使用！来取反，例如-m conntrack ！ --ctstate ESTABLISHED,RELATED, 这条命令匹配除了 ESTABLISHED 和 RELATED 之外的所有状态。
Match	--ctproto
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m conntrack --ctproto TCP
Explanation	这个 match 匹配协议，它和—protocol 用法一样。
Match	--ctorigsrc
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m conntrack --ctorigsrc 192.168.0.0/24
Explanation	这个 match 匹配正向连接的源地址，用法和源地址匹配一样。
Match	--ctorigdst
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m conntrack --ctorigdst 192.168.0.0/24
Explanation	匹配正向连接的目的地址，用法同上。
Match	--ctreplsrc
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m conntrack --ctreplsrc 192.168.0.0/24
Explanation	这个 match 匹配反向连接的源地址，用法如上。

Match	--ctrepldst
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m conntrack --ctrepldst 192.168.0.0/24
Explanation	这个 match 匹配反向连接的目的地址，用法如上。
Match	--ctstatus
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m conntrack --ctstatus RELATED
Explanation	这个match匹配连接的状态，我们在状态匹配（状态匹配）一节介绍过，可用的值如下： • NONE-该连接没有状态 • EXPECTED-这个连接是被 expectation 创建的期望连接。 • SEEN_REPLY-本连接已经得到方向报文，但是还没有保障。 • ASSURED-这个连接已经有保障了，除非它的超时或者一种一种关闭，这个连接才会被删除。 当然我们也接受取反（!）操作符，例如-m conntrack ! --ctstatus ASSURED 匹配除了 ASSURED 之外的所有报文。
Match	--ctexpire
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m conntrack --ctexpire 100:150
Explanation	这个 match 用来匹配连接的连接跟踪系统里面的超时时间，时间单位是秒。 1，我们可以输入一个数字，例如 iptables -A INPUT -p tcp -m conntrack --ctexpire 100 2，我们还可以匹配一个范围，例如 iptables -A INPUT -p tcp -m conntrack --ctexpire 100:150 3，这个 match 还支持取反，例如-m conntrack ! --ctexpire 100 匹配所有超时时间不是 100 秒的

DSCP 匹配器（match）

这个match基于DSCP匹配报文，DSCP可以用过下面文档了解更多信息。*RFC 2638 - A Two-bit Differentiated Services Architecture for the Internet* RFC。

Table 10-13. Dscp match options

表 10-13 Dscp 匹配器选项

Match	--dscp
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m dscp --dscp 32
Explanation	这个选项指定匹配的数值，我们可以输入 10 进制或者 16 进制。 1. 如果我们以 10 进制输入，直接输入数字即可 2. 如果我们以 10 进制输入，那么数字以 0x 开头，例如 0x20 3. 本选项支持取反(!)操作。
Match	--dscp-class
Kernel	2.5 and 2.6
Example	iptables -A INPUT -p tcp -m dscp --dscp-class BE
Explanation	--dscp-class match 用来基于报文的 diffServ 类别来匹配报文，这个值可以是 RFC 里面定义的 BE,EF,AFxx,CSxx 等等，本选项也支持取反(!)

请注意—dscp 和—dscp-class 两个选项是互斥的，他们不能同时使用。

ECN匹配器（match）

Ecn match用来基于TCP的ECN字段匹配报文，ECN的内容请见*RFC 3168 - The Addition of Explicit Congestion Notification (ECN) to IP。*下面介绍选项。

表 10-14. Ecn match 选项

Match	--ecn
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m ecn --ecn-tcp-cwr
Explanation	如果报文的 CWR 比特被设置，那么--ecn-tcp-cwr 用来匹配这个报文。CWR 主要用来通知对端本段已经收到 ECE，而且做出了响应。默认情况下，我们匹配 CWR 设置了的报文，但是我们可以通过取反(!)来修改本行为。
Match	--ecn-tcp-ece

Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m ecn --ecn-tcp-ece
Explanation	--ecn-tcp-ece 用来匹配报文的 ECE 比特。如果主机收到路由器设置的 CE 报文，那么主机就发送 ECE 的回复报文，通过这个报文来通知对端减低发送速度，对端最后发送 CWR 报文来确认。默认情况下 ECE 比特被设置的报文会被匹配，但是我们可以通过取反(!)来修改。
Match	--ecn-ip-ect
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m ecn --ecn-ip-ect 1
Explanation	--ecn-ip-ect 用来匹配 ECT 报文，ECT 有如下几种用法。 1.它们通过设置相应比特来协议 ECN 能够被支持 2.ECT 被路由器用来通知拥塞发生 ECT的值可以通过*ECN Field in IP*查看。

Table 10-15. ECN Field in IP

Iptables value	ECT	CE	[Obsolete] RFC 2481 names for the ECN bits.
0	0	0	Not-ECT, ie. non-ECN capable connection.
1	0	1	ECT(1), New naming convention of ECT codepoints in RFC 3168.
2	1	0	ECT(0), New naming convention of ECT codepoints in RFC 3168.
3	1	1	CE (Congestion Experienced), Used to notify endpoints of congestion

---

Hashlimit 匹配器（match）

它是 Limit match 的改进版本，它为每一个目的地址，源地址，目的端口以及源端口都建立 hash 表项。例如你可以设置每一个 IP 地址每秒钟最多接收 1000 个报文，或者你可以设置每一种业务报文每秒钟最多 200 个。

每一条规则创建一个单独的 hashtable，每一个 hashtable 都有最大尺寸以及最大数量的桶。Hash 表里面是一个或者多个源地址，目的地址，源端口，目的端口的组合。

Table 10-16. Hashlimit match options

Match	--hashlimit
Kernel	2.6
Example	iptables -A INPUT -p tcp --dst 192.168.0.3 -m hashlimit --hashlimit 1000/sec --hashlimit-mode dstip,dstport --hashlimit-name hosts
Explanation	--hashlimit 指定每一个桶的限度，在这个例子里面是 1000.这个数值可以是/sec， /minute，/hour 或者/day.默认是/sec。  这个选项是必须的。
Match	--hashlimit-mode
Kernel	2.6
Example	iptables -A INPUT -p tcp --dst 192.168.0.0/16 -m hashlimit --hashlimit 1000/sec --hashlimit-mode dstip --hashlimit-name hosts
Explanation	--hashlimit-mode 选项指定了哪些值用来作为 hash 的 key。这个例子里面，我们用 dstip，所以 192.168.0.0/16 网络里面的每一台主机都会被限制 1000/sec.我们可以在源/目的地址，源/目的端口里面选择一个或者多个。  这个选项是必须的。
Match	--hashlimit-name
Kernel	2.6
Example	iptables -A INPUT -p tcp --dst 192.168.0.3 -m hashlimit --hashlimit 1000 --hashlimit-mode dstip,dstport --hashlimit-name hosts
Explanation	这个选项指定了具体 hash 的名称，我们可以在目录 /proc/net/ipt_hashlimit 里面看到。

	 这个选项也是必须的。
Match	--hashlimit-burst
Kernel	2.6
Example	iptables -A INPUT -p tcp --dst 192.168.0.3 -m hashlimit --hashlimit 1000 --hashlimit-mode dstip,dstport --hashlimit-name hosts --hashlimit-burst 2000
Explanation	这个选项和—limit-burst 一样设置桶的最大尺寸，每一个桶都有一个突发大小，令牌桶的工作原理请参考 Limit match
Match	--hashlimit-htable-size
Kernel	2.6
Example	iptables -A INPUT -p tcp --dst 192.168.0.3 -m hashlimit --hashlimit 1000 --hashlimit-mode dstip,dstport --hashlimit-name hosts --hashlimit-htable-size 500
Explanation	这个选项设置最多可用的桶的数目，这个例子里面，它意味着最多 500 个端口可以同时使用。
Match	--hashlimit-htable-max
Kernel	2.6
Example	iptables -A INPUT -p tcp --dst 192.168.0.3 -m hashlimit --hashlimit 1000 --hashlimit-mode dstip,dstport --hashlimit-name hosts --hashlimit-htable-max 500
Explanation	--hashlimit-htable-max 设定 hash 表项的最大值，这里面包含所有的连接，当然包含那些暂时没有激活的连接。
Match	--hashlimit-htable-gcinterval
Kernel	2.6
Example	iptables -A INPUT -p tcp --dst 192.168.0.3 -m hashlimit --hashlimit 1000 --hashlimit-mode dstip,dstport --hashlimit-name hosts --hashlimit-htable-gcinterval 1000
Explanation	垃圾回收多久运行，一般而言，这个值比超时时间小，数值单位为毫秒。假如太小，那么大量的系统资源就会浪费，但是如果太大则会有大量的无用连接专占用 hash 表项，让新的连接无法使用。这个例子里面垃圾回收每秒钟运行一次。
Match	--hashlimit-htable-expire
Kernel	2.6
Example	iptables -A INPUT -p tcp --dst 192.168.0.3 -m hashlimit -

	-hashlimit 1000 --hashlimit-mode dstip,dstport --hashlimit-name hosts --hashlimit-htable-expire 10000
Explanation	这个值设定多久一个空闲的 hash 表项会超时，假如一个桶长于这个时间没有使用，它就会超时，接着下一次的垃圾回收就会把它从 hash 表项里面移除。

Helper 匹配器（match）

这个 match 和我们之前介绍的 match 有一些差异，这个 match 是基于连接的 helper 模块名称来匹配报文的。我们看一下 FTP 的会话过程，首先打开控制会话，接着在控制会话里面协商数据会话的参数，ip_conntrack_ftp helper 模块就会发现这个协商数据，接着创建一个关联的连接到控制连接上，这样一个报文进来我们就能够知道它管理到哪一个协议（就是连接跟踪的复杂协议过程）。

Table 10-17. Helper match options

Match	--helper
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m helper --helper ftp-21
Explanation	--helper 选项的参数是一个字符串，它告诉 match 哪一个 helper 被匹配，例如—helper irc。下面就是这个 match 和其他不一样的地方了，我们可以指定在哪一个端口上面这个期望连接被创建，例如通常情况下，FTP 都是在 21 端口上，但是它也可能改成 954 或者其他，那么我们就可以这样匹配—helper ftp-954

IP范围匹配器（match）

IP 范围匹配用来匹配一整段的 IP，和—source 以及—destination 匹配工作原理一样，只是 IP 范围匹配更为灵活，它可以匹配一整段的地址。

Table 10-18. IP range match options

Match	--src-range
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m iprange --src-range 192.168.1.13-192.168.2.19
Explanation	--src-range 用来匹配一整段的源地址。 1. 匹配整段地址，--src-range 192.168.1.13-192.168.2.19

	匹配从 192.168.1.13 到 192.168.2.19 之内的所有地址 2. 可以对地址取反，例如！--src-range 192.168.1.13-192.168.2.19 匹配除了 192.168.1.13 到 192.168.2.19 之外的所有地址
Match	--dst-range
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m iprange --dst-range 192.168.1.13-192.168.2.19
Explanation	匹配一整段的目的地址，用法和源一致。

长度匹配器（match）

长度匹配就是一个简单的根据报文长度匹配的 match，假如你因为一些特殊的原因或者想阻止 ping-of-death-like 这类的攻击，就用这个 match 吧。

Table 10-19. Length match options

Match	--length
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m length --length 1400:1500
Explanation	1. 指定一个整数值，例如--length 1400，匹配长度为 1400 的报文。 2. 指定一个范围，例如--length 1400:1500，匹配 1400-1500 之内的报文。 3. 取反操作，例如！--length 1400:1500,匹配所有非 1400-1500 之内的报文。

限速匹配器（match）

这个匹配操作必须由-m limit 明确指定才能使用。有了它的帮助，就可以对指定的规则的日志数量加以限制，以免你被信息的洪流淹没哦。比如，你可以事先设定一个限定值，当符合条件的包的数量不超过它时，就记录；超过了，就不记录了。我们可以控制某条规则在一段时间内的匹配次数（也就是可以匹配的包的数量），这样就能够减少 DoS syn flood 攻击的影响。这是它的主要作用，当然，还有很多其他作用（译者注：比如，对于某些不常用的服务可以限制连接数量，以免影响其他服务）。limit match 也可以用英文感叹号取反，如：-m limit ！ --limit 5/s 表示在数量超过限定值后，所有的包都会被匹配。

1.1.19 里面自己举了保安的例子，这儿我翻译原文的例子。

1. 我们首先设定一条规则-m limit --limit 5/second --limit-burst 10/second，limit-burst 令牌首先被初始化为 10，每一条满足这条规则的报文会消耗一个令牌。
2. 接着我们收到 1-2-3-4-5-6-7-8-9-10 十个报文，这 10 个报文之间的间隔满足 1 毫秒，而且这 10 个报文都能够匹配上这条规则。
3. 这样我们的令牌都发完了，一旦令牌用完，那么这个报文就不能被这条规则匹配，那么它就继续下一条规则的匹配，如果没有规则就是默认行为，例如 ACCEPT 或者 DROP。
4. 每隔 200 毫秒没有匹配到报文，令牌的数量就会加 1，当然这个数量是有上限的，就是我们通过—limit-burst 指定的数值，这儿是 10.，
5. 当然，我们每收到一个报文，令牌的数量也会减少 1，当减少为 0 的时候，本条规则就匹配不上。

Table 10-20. Limit match options

Match	--limit
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -m limit --limit 3/hour
Explanation	This sets the maximum average match rate for the limit match. You specify it with a number and an optional time unit. The following time units are currently recognized: /second /minute /hour /day. The default value here is 3 per hour, or 3/hour. This tells the limit match how many times to allow the match to occur per time unit (e.g. per minute).
Match	--limit-burst
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -m limit --limit-burst 5
Explanation	这里定义的是limit match的峰值，就是在单位时间（这个时间由上面的--limit指定）内最多可匹配几个包（由此可见，--limit-burst的值要比--limit的大）。默认值是 5。为了观察它是如何工作的，你可以启动“只有一条规则的脚本”Limit-match.txt，然后用不同的时间间隔、发送不同数量的ping数据包。这样，通过返回的 *echo replies*就可以看出其工作方式了。

MAC地址匹配器（match）

MAC match 能够用来根据报文的 MAC 地址匹配报文，到目前为止，这个模块实现还有一些限制，只能根据报文的源 MAC 进行匹配，但是将来肯定会有很多改动让它更加实用。

请注意这个模块是通过-m mac 选项来加载的，我之所以提及是很多人总想当然的认为是用-m mac-source 加载。

表 10-21. MAC 匹配的选项

Match	--mac-source
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -m mac --mac-source 00:00:00:00:00:01
Explanation	基于包的 MAC 源地址匹配包，地址格式只能是 XX:XX:XX:XX:XX:XX，当然它也可以用英文感叹号取反，如--mac-source ! 00:00:00:00:00:01，意思很简单了，就是除此之外的地址都可接受嘛。注意，因为 MAC addresses 只用于 Ethernet 类型的网络，所以这个 match 只能用于 Ethernet 接口。而且，它还只能在 PREROUTING，FORWARD 和 INPUT 链里使用。

Mark匹配器（match）

Mark match 用来基于报文上面被设置的 marks 来匹配报文，marks 是一个特殊的字段，而且它只在本机内核里有效。Marks 可以被不同的内核模块使用，例如流量整形和过滤。到目前为止，只有一种方式设置报文的 marks 字段，那就是 MARK target，在以前 ipchain 里面叫做 FWMARK，这也就是为什么很多人在高级路由里面仍然提及 FWMARK 了。Marks 字段是一个 32bit 的整数，所以它的最大值为 4294967296，现在看来它足够用了。

Table 10-22. Mark match options

Match	--mark
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -t mangle -A INPUT -m mark --mark 1
Explanation	以包被设置的 mark 值来匹配包，这个值是是由下面将要介绍的 MARK target 来设置的，它是一个无符号的整数。所有通过 Netfilter 的包都会被分配一个相关联的 mark field 。但要注意 mark 值可不是在任何情况下都能使用的，它只能在分配给它值的那台机子里使用，因为它只是由内核在内存里分配的和包相关的

	几个字节，并不属于包本身，所以我们不能在本机之外的路由器上使用。mark 的格式是--mark value[/mask]，如上面的例子是没有掩码的，带掩码的例子如--mark 1/1。如果指定了掩码，就先把 mark 值和掩码取逻辑与，然后再和包的 mark 值比较。

多端口匹配器（match）

多端口 match 用来匹配多个不连续的端口，没有这个模块我们就只能创建多条规则来完成这个功能了。

你不能在一条规则里面把标准端口匹配和多端口匹配混用，例如--sport 1024:63353 -m multiport --dport 21,23,80， 它可能就不是按照你所想的方式工作，相反只有第一条规则生效，后面的多端口匹配会被忽略。

Table 10-23. Multiport match options

Match	--source-port
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m multiport --source-port 22,53,80,110
Explanation	最多能够匹配 15 个不连续的源端口，它使用的前提是通过-p tcp 或者-p udp 指定协议，其实它就是增强版的—source-port 功能。
Match	--destination-port
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m multiport --destination-port 22,53,80,110
Explanation	匹配报文的目的端口，用法如上。
Match	--port
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m multiport --port 22,53,80,110
Explanation	同端口多端口匹配，意思就是它匹配的是那种源端口和目的端口是同一个端口的包，比如：端口 80 到端口 80 的包，110 到 110 的包等。使用方法和源端口多端口匹配一样。

属主匹配器（match）

基于包的生成者（也就是所有者，或称作拥有者，owner）的 ID 来匹配包，owner 可以是启动进程的用户的 ID，或用户所在的组的 ID，或进程的 ID，或会话的 ID。这个扩展原本只是为了说明 iptables 可以做什么，现在发展到实用阶段了。但要注意，此扩展只能用在 OUTPUT 中，原因显而易见：我们几乎不可能得到发送端程序的 ID 的任何信息，或者在去往真正目的地的路上哪儿有路由。甚至在 OUTPUT 链里，这也不是十分可靠，因为有些包根本没有 owner，比如 ICMP responses，所以它们从不会被这个 match 抓到。

Table 10-24. Owner match options

Match	--cmd-owner
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m owner --cmd-owner httpd
Explanation	这个选项匹配命令的属主，也就是发送这个报文的进程的名称，在这个例子里面，httpd 就会被匹配。另外这个选项支持取反操作(!)，例如-m owner ! --cmd-owner ssh。
Match	--uid-owner
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m owner --uid-owner 500
Explanation	按生成包的用户的 ID（UID）来匹配外出的包。使用这个匹配可以做这样一些事，比如，阻止除 root 外的用户向防火墙外建立新连接，或阻止除用户 http 外的任何人使用 HTTP 端口发送数据。
Match	--gid-owner
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m owner --gid-owner 0
Explanation	按生成包的用户所在组的 ID（GID）来匹配外出的包。比如，我们可以只让属于 network 组的用户上 Internet，而其他用户都不行；或者只允许 http 组的成员能从 HTTP 端口发送数据。
Match	--pid-owner
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m owner --pid-owner 78
Explanation	按生成包的进程的ID（GID）来匹配外出的包。比如，我们可以只允许PID为 94 的进程（http进程当然不能是多线程的）使用http端口。这个匹配使用起来有一点难度，因为你要知道进程的ID号。当然，你也可以写一个小小的脚本，先从ps的输出中得到PID，再添加相应的规则，这儿有个例子Pid-owner.txt。
Match	--sid-owner
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m owner --sid-owner 100

Explanation	按生成包的会话的ID（SID）来匹配外出的包。一个进程以及它的子进程或它的多个线程都有同一个SID。比如，所有的HTTPD进程的SID和它的父进程一样（最初的 HTTPD进程），即使HTTPD是多线程的（现在大部分都是，比如Apache和Roxen）也一样。这里有个脚本*Sid-owner.txt*可以反映出这一点。

PID/SID 以及命令匹配在 SMP 的内核上工作不正常，因为他们有不同的进程列表，但是在后续版本里这个问题可能被修复。

报文类型匹配器（match）

报文匹配是基于报文类别来匹配报文，例如报文的目的地址是特定用户还是所有人或者一组人，他们对于的报文类型就是单播，广播以及组播。详细的介绍请看*TCP/IP repetition*章节。

Table 10-25. Packet type match options

Match	--pkt-type
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m pkttype --pkt-type unicast
Explanation	--pkt-type 告诉报文类型匹配器什么样的报文将被匹配，它的可能值是 unicast , broadcast or multicast，这儿也可以取反(!),例如-m pkttype --pkt-type ! broadcast 就能够匹配到所有的非广播报文。

Realm匹配器（match）

Realm match 用来基于路由 realm 来匹配报文，路由 realm 在复杂的路由场景里面使用，例如 BGP 等等。

路由 realm 被 linux 用来把路由划分成逻辑的路由组。在一台路由器上面，路由表和转发表互相之间是可以直接访问的，但是在 linux 系统上面，因为它不是完全的路由器，所以路由表在用户空间，而转发表在 linux 内核里面，这样查找路由表就很消耗资源了。路由 realm 就是为解决这个问题而诞生的，从而让系统灵活和高效。

Reals 被 BGP 等有大量路由条目的路由协议引用，路由守护进程能够他们的前缀/源等等把它们放置在不同的 realm 里面。Realm 是一个数字，但是我们也可以通过/etc/iproute2/rt_realms 设置名称。

Table 10-26. Realm match options

Match	--realm
Kernel	2.6
Example	iptables -A OUTPUT -m realm --realm 4
Explanation	这个选项匹配报文的 realm，我们还可以增加一个掩码，但是掩码是可选的。假如我们输入的不是一个数字，那么系统就会尝试通过/etc/iproute2/rt_realms 文件来解析。假如我们使用的是名称，那么我们就不能设置掩码了。这个选项也支持取反操作。

Recent match

Recent 匹配是一个相当大而且复杂的匹配器，它让我们能够根据我们之前匹配的内容再做一次匹配。例如我们看到一个 IRC 的出连接，我们能够把这个地址加入主机列表，并且我们还有另外一条规则能够识别出来返回的报文。

在我们具体看匹配选项之前，我们先尝试解释一下它是如何工作的。首先，我们用几条不同的规则来完成 recent 匹配，recentmatch 用几条不同的列表来记录事件，默认的列表是 DEFAULT，我们通过 set 选项创建一个新的实体，这样当这个规则被匹配时候，我们就把这个实体假如指定列表。列表上的实体包含一个时间戳/源地址。这样我们接着就可以用其他不同的选项来基于这些信息进行匹配或者更新时间戳等等。

最后，假如我们为了一些原因需要删除一个实体，我们可以通过—remove 选项来完成。在我们举实际例子前，我们先认识一下所以的选项。

Table 10-27. Recent match options

Match	--name
Kernel	2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m recent --name examplelist
Explanation	默认的列表名称是 DEFAULT，但是如果我们有对于一个列表时候，这可能就不合适了。
Match	--set
Kernel	2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m recent --set
Explanation	这个选项在指定的列表里面增加一个实体（相当于在指定列表上增加一个元素）。这个 match 总是会返回 ture，除非我们使用了取反操作。
Match	--rcheck

Kernel	2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m recent --name examplelist --rcheck
Explanation	--rcheck 选项是在指定的列表里面查找是否有含有这个源地址的实体（根据 ip 地址寻找链表里面的元素）。假如找到了，那么就返回 true，否则返回 false。 我们也支持取反操作(!),这样如果查找到了就返回 false，负责返回 true。
Match	--update
Kernel	2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m recent --name examplelist --update
Explanation	假如源地址能够在指定列表里面找到就返回 true，并且更新 last-seen 时间。这个选项也支持取反操作。
Match	--remove
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -m recent --name example --remove
Explanation	这个 match 会首先尝试用源地址在链表里面查找实体，查找到了就返回 true，当然最后这个实体就会被删除。这个操作也支持取反操作。
Match	--seconds
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -m recent --name example --check --seconds 60
Explanation	这个选项只能与—check 和—update 一起使用，这个选项指定了自从上次”last seen”被更新以来到现在的时间窗。假如 last-seen 比这个时间旧，那就返回 false，负责就和上面的工作机制一样。
Match	--hitcount
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -m recent --name example --check --hitcount 20
Explanation	--hitcount 必须与—check 和—update 一起使用。这就会限制报文的数量。假如它和—seconds 合用，那就表示在这个时间窗内最多允许的报文数量。
Match	--rttl
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -m recent --name example --check --rttl
Explanation	--rttl 选项让你能够根据现在的报文 TTL 和列表上面的实体里面

	TTL 数值比较。他们必须要一致，这样就避免了很多挟持会话攻击。
Match	--rsource
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -m recent --name example --rsource
Explanation	--rsource 选项告诉 match 要在列表里记录报文的源地址和端口，这是默认行为。
Match	--rdest
Kernel	2.4, 2.5 and 2.6
Example	iptables -A INPUT -m recent --name example --rdest
Explanation	--rdest 选项和—rsource 相对应，它需要建立报文的目的地址和端口。

我写了一个脚本来描述recent是如何使用的，你可以在Recent-match.txt里面发现。

简单的说，这个例子是一个 netfilter 里面状态机制的一个替代，但是它的确很差。首先我们创建两条链，一个叫做 http-recent，另外一个叫做 http-recent-final.http-recent 链用在一个连接的开始以及实际的数据传输阶段，而 http-recent-final 用于最后的连接关闭。

这个例子是一个很糟糕的状态匹配替代品，首先它不能处理状态匹配的所有功能，另外它很慢，特别是处理一些极端例子。但是它仍然是一个我们学习 recent 匹配的好例子。

那么我们跟随一个报文穿透这些规则集，首先报文到达 INPUT 链，我们把它送到 http-recent 链。

1. 第一个报文应该是一个 SYN 报文，并且不能是 ACK/FIN 或者 RST 报文。所以我们通过—tcp-flags SYN,ACK,FIN,RST SYN 来匹配报文。在这儿我们把这个报文加入列表-m recent --name httplist －set，最后当然我们接受(ACCEPT)这个报文。
2. 在第一个报文后，我们应当收到一个 SYN/ACK 报文，这样我们增加这个匹配--tcp-flags SYN,ACK,FIN,RST SYN,ACK。 此刻，我们更新列表里面的实体，-m recent --name httplist －update，最后接受(ACCEPT)这个报文。
3. 此刻我们应该从连接的发起端得到一个 ACK 确认报文，这儿我们也是更新列表里面的实体，并且接受(ACCEPT)报文

4. 此刻，数据发送可以开始了，从此开始报文里面不能包含任何 SYN 标记，但是可以有 ACK 标记来确认收到的数据。每一次我们收到这样的报文，我们就是更新实体时间以及接受(ACCEPT)它。
5. 发送可以通过两个方向来终止，最简单的就是发送 RST 报文，RST 就是简单的复位连接这样它就终止了。通过 FIN/ACK，对端确认 FIN，这样发起者就不能再发送数据了。但是对端仍然能够收发数据，所以我们把连接状态设置为 final 来处理接下来的事务。
6. 在 http-recent-final 链里面，我们坚持这个报文是否仍然在 httplist 里面，假如是我们就把它发送到 http-recent-fianl1 链里面。在最初链里面我们把这个连接从 httplist 里面删除并且假如 http-recnet-fianl。如果连接已经被删除并且移到 http-recent-final 链表里，我们就把这个报文发送到 http-recent-final2 链上。
7. 在最后的 http-recent-final2 链上，我们等待没有关闭的一段结束发送数据，并且也关闭这个连接。一段这个操作完成，这个连接就完全移除了。

正如你看到的，recent 链表变得相当复杂，但是你能够给你提供很多方法。我们需要记住我们从来不需要发明轮子。假如已经有现成的功能，重用他而不是创造自己的。

状态匹配器（match）

状态匹配和连接跟踪一起协作，因为它是从连接跟踪机制中得到包的状态的。这样我们就可以了解连接所处的状态。它几乎适用于所有的协议，包括那些无状态的协议，如ICMP和UDP。针对每个连接都有一个缺省的超时值，如果连接的时间超过了这个值，那么这个连接的记录就被会从连接跟踪的记录数据库中删除，也就是说连接就不再存在了。这个match必须有-m state作为前提才能使用。状态机制的详细内容在章节状态匹配中。

表 10-28. 状态匹配选项

Match	--state
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -m state --state RELATED,ESTABLISHED
Explanation	指定要匹配包的的状态，当前有 4 种状态可用：INVALID，ESTABLISHED，NEW和RELATED。 INVALID意味着这个包没有已知的流或连接与之关联，也可能是它包含的数据或包头有问题。ESTABLISHED意思是包是完全有效的，而且属于一个已建立的连接，这个连接的两端都已经有数据发送。NEW表示包将要或已经开始建立一个新的连接，或者是这个包和一个还没有在两端都有数据发送的连接有关。RELATED说明包正在建立一个新的连接，这个

	连接是和一个已建立的连接相关的。比如，FTP data transfer，ICMP error 和一个TCP或UDP连接相关。注意NEW状态并不在试图建立新连接的TCP包里寻找SYN标记，因此它不应该不加修改地用在只有一个防火墙或在不同的防火墙之间没有启用负载平衡的地方。具体如何使用，你就再看看章节状态机制吧

Tcpmss匹配器（match）

Tcpmss匹配主要用来基于TCP的最大分段大小来匹配报文。这个match只对SYN和SYN/ACK报文有效。想要完整的MSS解释，请看附录的TCP options或者*RFC 793 - Transmission Control Protocol* and the *RFC 1122 - Requirements for Internet Hosts - Communication Layers*。

表 10-29. Tcpmss 匹配选项

Match	--mss
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp --tcp-flags SYN,ACK,RST SYN -m tcpmss --mss 2000:2500
Explanation	--mss 选项告诉匹配器匹配多大的数值，它可以是一个整数也可以是一个范围，例如 -m tcpmss --mss 2000 或者 -m tcpmss --mss 2000:2500 这个选项也支持取反操作(!),例如 -m tcpmss ! --mss 2000:2500

TOS匹配

根据 TOS 字段匹配包，必须使用-m tos 才能装入。TOS 是 IP 头的一部分，其含义是 Type Of Service，由 8 个二进制位组成，包括一个 3 bit 的优先权子字段（现在已被忽略），4 bit 的 TOS 子字段和 1 bit 未用位（必须置 0）。它一般用来把当前流的优先权和需要的服务（比如，最小延时、最大吞吐量等）通知路由器。但路由器和管理员对这个值的处理相差很大，有的根本就不理会，而有的就会尽量满足要求。

表 10-30. Tos 匹配选项

Match	--tos
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A INPUT -p tcp -m tos --tos 0x16
Explanation	根据 TOS 字段匹配包。这个 match 常被用来 mark 包，以便后用，除此之外，它还常和 iproute2 或高级路由功能一起使用。它的参数可以是 16 进制数，也可以是十进制数，还可以是相应的名字（用 iptables -m tos -h 能查到）。到写这篇文章时，有以下参数可用： Minimize-Delay 16 (0x10)，要求找一条路径使延时最小，一些标准服务如 telnet、SSH、FTP-control 就需要这个选项。 Maximize-Throughput 8 (0x08)，要求找一条路径能使吞吐量最大，标准服务 FTP-data 能用到这个。 Maximize-Reliability 4 (0x04)，要求找一条路径能使可靠性最高，使用它的有 BOOTP 和 TFTP。 Minimize-Cost 2 (0x02)，要求找一条路径能使费用最低，一般情况下使用这个选项的是一些视频音频流协议，如 RTSP（Real Time Stream Control Protocol）。 Normal-Service 0 (0x00)，一般服务，没有什么特殊要求。

TTL匹配

根据 IP 头里的 TTL (Time To Live，即生存期)字段来匹配包，此必须由-m ttl 装入。TTL field 是一个字节（8 个二进制位），一旦经过一个处理它的路由器，它的值就减去 1 它的值。当该字段的值减为 0 时，报文就被认为是不可转发的，数据报就被丢弃，并发送 ICMP 报文通知源主机，不可转发的报文被丢弃。这也有两种情况，一是传输期间生存时间为 0，使用类型为 11 代码是 0 的 ICMP 报文；二是在数据报重组期间生存时间为 0，使用类型为 11 代码是 1 的 ICMP 报文。这个 match 只是根据 TTL 匹配包，而对其不做任何更改，所以在它之后可使用任何类型的 match。

表 10-31.TTL 匹配选项

Match	--ttl-eq
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m ttl --ttl-eq 60
Explanation	根据 TTL 的值来匹配包，参数的形式只有一种，就是十进制数值。它可以被用来调试你的局域网，比如解决 LAN 内的主机到 Internet 上的主机的连接问题，或找出 Trojan（Trojan）可能的入口。这个 match 的用处相对有限，但它其实是很有用的，这就看你的想象力如何了。比如可以用它来发现那些 TTL 具有错误缺省值的机子（这可能是实现 TCP/IP 栈功能的那个程序本身的错

	误，或者是配置有问题）。
Match	--ttl-gt
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m ttl --ttl-gt 64
Explanation	用法同上，只是要求大于指定的数值。
Match	--ttl-lt
Kernel	2.3, 2.4, 2.5 and 2.6
Example	iptables -A OUTPUT -m ttl --ttl-lt 64
Explanation	用法同上，只是要求小于指定的数值。

非正常报文匹配

这个匹配没有任何参数，也不需要显式地装载。注意这应该被看作是一个实验性的匹配，它不总是能正常工作的，对有些不正常的包（unclean package，就是所谓的脏包）或问题，它是视而不见的。这个 match 试图匹配那些好象畸形或不正常的包，比如包头错或校验和错，等等。它可能常用来 DROP 错误的连接、检查有错的流，但要知道这样做也可能会中断合法的连接。

下章预览

本章主要介绍了 iptables 里面所使用的 matches 以及他们的作用。Iptables 和 netfilter 的 match 很好开发以及具有相当的灵活性。下一章我们主要讨论 targets 以及他们的作用，以及 linux 防火墙的能力。

第 11 章. Iptables的动作（targets）和跳转(jumps)

Target/jump 告诉一条规则当报文完全匹配该条规则的时候，应该怎么处理此报文。系统里面有很多基本的 target，例如 ACCEPT 和 DROP 等等。但是在我们介绍这些 target 之前，我们一起简要的看一下 jump 是如何工作的。

Jump 其实和 target 工作方式完全一样，只是 jump 需要在同一表内指定一条链。我们之前已经解释过，用户自定义的链通过-N 命令创建，例如我们首先创建一个 tcp_packets 链，用如下命令：

```
iptables -N tcp_packets
```

接着我们在增加一个到这条链的跳转。

```
iptables -A INPUT -p tcp -j tcp_packets
```

接着我们从INPUT链跳转到tcp_packets链，并且开始tcp_packets链的遍历。假如报文到达链的尾部（即未匹配上任何工作），就返回到父链的下一条规则继续执行。假如一个报文在子链里面被一条规则匹配上，那就相当于在父链里面匹配，它不会在返回父链继续遍历规则。请注意这个报文会和其他正常流程一样在所有其他的链里面遍历，详细情况请查看表和链的遍历章节。

target 指定我们要对包做的操作，比如 DROP 和 ACCEPT，还有很多，我们后面会介绍。不同的 target 有不同的结果。一些 target 会使包停止前景，也就是不再继续比较当前链中的其他规则或父链中的其他规则，最好的例子就是 DROP 和 ACCEPT。而另外一些 target 在对包做完操作之后，包还会继续和其他的规则比较，如 LOG，ULOG 和 TOS。它们会对包进行记录、mangle，然后让包通过，以便匹配这条链中的其他规则。有了这样的 target,我们就可以对同一个包既改变它的 TTL 又改变它的 TOS。有些 target 必须要有准确的参数（如 TOS 需要确定的数值），有些就不是必须的，但如果我们想指定也可以（如日志的前缀，伪装使用的端口，等等）。本节我们会尽可能全面地介绍每一个 target。现在我们就来看看有哪几种 target。

ACCEPT target

ACCEPT target 不需要任何才是，一旦一个报文满足了匹配规则，它就会被 ACCEPT。这样本条规则被匹配上之后，它就不会继续在本链或者本表的其他链中遍历。但是请注意，系统中可能还有其他的表，报文也有可能在那儿被丢弃。

在 2.3/2.4/2.5/2.6 内核里面工作。

CLASSIFY target

CLASSIFY target用来给报文分类，它可以根据不同的qdiscs来分类报文，例如atm/sbq/dsmark/pfifo_fast/htb以及prio_qdiscs。更多的信息请参考Linux Advanced Routing and Traffic Control HOW-TO。

CLASSIFY target 只在 mangle 表的 POSTROUTING 链里面有效。

Table 11-1. CLASSIFY target options

Option	--set-class
Example	iptables -t mangle -A POSTROUTING -p tcp --dport 80 -j CLASSIFY --set-class 20:10
Explanation	CLASSIFY target只有一个—set-class参数。这个参数告诉target如何分类报文，class需要两个参数，他们通过冒号隔开，例如MAJOR:MINOR。再一次，想更多信息你需要参考Linux Advanced Routing and Traffic Control HOW-TO。

工作于 2.5 和 2.6 内核。

CLUSTERIP target

CLUSTERIP target 基于轮询的方式创建一组具有相同 IP 和 MAC 的负载分担主机。实现很简单，你需要在这个 cluster 里面的所有机器上面配置一个相同的虚拟 IP(VIP)，接着在每一台机器上面匹配 CLUSTERIP。它不需要专门的负载均衡硬件或者机器，只是在 cluster 里面的每一台机器上面完成这个工作。注意这是一个非常简单的解决方案，并不适合大型和复杂网络。它同时也没有实现“心”跳功能。

所有运行的主机都有一个系统的组播 MAC，这样通过的主机都能够收到请求消息，然后通过一个 hash 算法来决定 cluster 里面的哪一台机器需要对请求作出应答。组播 MAC 以 01:00:5e 开头，例如一个组播 MAC 可能是 01:00:5e:00:00:20。虚拟 IP 地址可以是任何地址，但是所有主机的 IP 地址必须相同。

需要注意的是 CLUSTERIP 可能对 SSH 这类的协议产生影响，连接可以正常建立，但是当你再一次连接同一主机时候，可能会连接到另外一台主机，这样它就可能有不同的密钥，所有你的 ssh 客户端就会解决连接。所有解决方案主要有两个：

1. 对于管理用的地址，单独设置一台机器，而不是放在 cluter 里面
2. 对于 cluster 里面的所有机器，配置相同的密钥

Cluster 能够通过三种方式来实现 hash，分别是只有源地址，源地址+源端口以及源地址+源端口+目的端口。第一种方式对于你需要记住连接状态很重要，例如有购物车的 web 服务器，但是这种方式会有一些副作用，有一些机器会比另外一些负载更大，因为同一台主机的所有请求都会到同一台服务器。

源地址+源端口在你不需要保持连接状态的时候可能是一个不错的主机。例如带有搜索引擎的网页。

最后一种，源地址+源端口+目的端口在你有几个服务同时运行同时也不需要保持连接状态时候很有效。这样的例子 NTP/DNS 等等。这种工作方式就是一个轮询-即每一台主机每次收到一个请求。

每一个 cluter 里面的主机在/proc/net/ipt_CLUSTERIP 目录下都会有一个文件，这个文件和 VIP 相关。例如你的 VIP 是 192.168.0.5，那么这个文件就是 192.168.0.5. 打开这个文件你就能够看到这台机器回应哪个节点。你可以修改让机器回应另外一个节点，例如节点 2，那么我们通过如下命令。

```
echo "+2" >> /proc/net/ipt_CLUSTERIP/192.168.0.5
```

移除通过下面命令：

```
echo "-2" >> /proc/net/ipt_CLUSTERIP/192.168.0.5
```

Table 11-2. CLUSTERIP target options

Option	--new
Example	iptables -A INPUT -p tcp -d 192.168.0.5 --dport 80 -j CLUSTERIP --new ...
Explanation	--new 选项创建一个新的 CLUSTERIP 的实体，它必须放在规则的第一条。
Option	--hashmode
Example	iptables -A INPUT -p tcp -d 192.168.0.5 --dport 443 -j CLUSTERIP --new --hashmode sourceip ...
Explanation	--hashmode 选项来用决定才有哪种 hash 模式，分别可以是 • sourceip • sourceip-sourceport • sourceip-sourceport-destport
Option	--clustermac
Example	iptables -A INPUT -p tcp -d 192.168.0.5 --dport 80 -j CLUSTERIP --new --hashmode sourceip --clustermac 01:00:5e:00:00:20 ...

Explanation	The MAC address that the cluster is listening to for new connections. This is a shared Multicast MAC address that all the hosts are listening to. See above for a deeper explanation of this. --clustermac 用来指定所有节点共享的组播 MAC
Option	--total-nodes
Example	iptables -A INPUT -p tcp -d 192.168.0.5 --dport 80 -j CLUSTERIP --new --hashmode sourceip --clustermac 01:00:5e:00:00:20 --total-nodes 2 ...
Explanation	--total-nodes 表明这个 cluster 里面有多少台主机参与工作。
Option	--local-node
Example	iptables -A INPUT -p tcp -d 192.168.0.5 --dport 80 -j CLUSTERIP --new --hashmode sourceip --clustermac 01:00:5e:00:00:20 --total-nodes 2 --local-node 1
Explanation	--local-node 参数指定本机在 cluster 里面的编号，它用于轮询的索引。
Option	--hash-init
Example	iptables -A INPUT -p tcp -d 192.168.0.5 --dport 80 -j CLUSTERIP --new --hashmode sourceip --clustermac 01:00:5e:00:00:20 --hash-init 1234
Explanation	--hash-init 产生一个 hash 种子。

这个target和RFC 1812 - Requirements for IP Version 4 Routers相冲突，所以请意识到可能带来的风险。特别是，3.3.2一节里面说一台路由器千万不要信任另外一台有组播MAC的主机或者路由器。

在最近的 2.6 内核里面引入，还处于试验阶段。

CONNMARK target

CONNMARK target 用来标记一条完整的连接，和 MARK target 的工作方式很像。它主要用来和 connmark match 一起配合使用。例如我们想匹配报文头的一个字段，但是我们不想仅仅标记一个报文而是整条连接，这个时候就是 CONNMARK 派上用场的时候了。

CONNMARK target 可以在所有的表和链上面使用，但是请记住 NAT 表只有第一个报文会被匹配，所有对于后续报文 NAT 上面的 CONNMARK 不生效。

Table 11-3. CONNMARK target options

Option	--set-mark
Example	iptables -t nat -A PREROUTING -p tcp --dport 80 -j CONNMARK --set-mark 4
Explanation	--set-mark 选项用来在连接上面设置标记(mark)，标记(mark)是一个无符号整数，数值范围为 0-42949672951.每一个 bit 可以通过—set-mark 12/8 的形式来设置，这样就只容许 mask 设置的位被赋值。例如—set-mask 12/8 就是只有第 4 位能够被赋值。
Option	--save-mark
Example	iptables -t mangle -A PREROUTING --dport 80 -j CONNMARK --save-mark
Explanation	--save-mark target 选项用来把通过 MARK target 添加到报文上的标记(mark)赋值给整条连接，这个标记(mark)可以通过下面的—mask 进行掩码。
Option	--restore-mark
Example	iptables -t mangle -A PREROUTING --dport 80 -j CONNMARK --restore-mark
Explanation	--restore-mark 和—save-mark 相反，它从连接里面恢复标记(mark)给报文。这个选项也可以和 mask 合用，但是需要注意的是这个标记只在 mangle 表里面有效。
Option	--mask
Example	iptables -t mangle -A PREROUTING --dport 80 -j CONNMARK --restore-mark --mask 12
Explanation	--mask 只能和—save-mark 或者—restore-mark 合用，--mask 表明了哪些位需要保存或者恢复。例如恢复出来的值是 11，B1011，但是它的 mask 为 12，B1100，最终就只有 1000 是有效的。

工作于 2.6 内核。

CONNSECMARK target

CONNSECMARK target用来在报文的mark和SELinux安全上下文的mark之间互相拷贝。更多关于SELinux文档，请阅读*Security-Enhanced Linux*主页。本target只能在mangle表里面和SECMARK合用。SECMARK用来对报文进行mark设置。

SELinux 的描述超出本文描述的范围，但是简单的说这个功能是 Linux 访问控制的增强功能。这个功能比现在大部分的 Linux/Unix 上面都要更安全，每一个对象都有一个安全属性或者说安全上下文，在允许或者拒绝一个任务执行之前，需要匹配这些安全属性。

Table 11-4. CONNSECMARK target options

Option	--save
Example	iptables -t mangle -A PREROUTING -p tcp --dport 80 -j CONNSECMARK --save
Explanation	--save 选项用来把报文上的安全标记拷贝到连接上。
Option	--restore
Example	iptables -t mangle -A PREROUTING -p tcp --dport 80 -j CONNSECMARK --restore
Explanation	--restore 用来从连接上恢复出安全比较到报文上。

DNAT target

DNAT target 用来做目的地址转换，换句话讲就是重写报文的目的地址。假如一个报文匹配了这个规则，而且这条规则的 target 是 DANT，那么这个报文以及这条连接上的所有后续报文都会被转换地址。这个功能相当有用，例如你有一个主机在 LAN 内运行 web 服务，但是没有可以在 internet 上路由的 IP，这个时候你可以告诉防火墙把所有访问 HTTP 端口的报文转发到这台主机。目的地址还可以是一个地址段，这样做就起到了负载分担的效果。

注意 DNAT 只在 NAT 表里面的 PREROUTING 和 OUTPUT 链上有效，或者是被这两条链调用的链里。但还要注意的是，包含 DANT target 的链不能被除此之外的其他链调用，如 POSTROUTING。

Table 11-5. DNAT target options

Option	--to-destination
Example	iptables -t nat -A PREROUTING -p tcp -d 15.45.23.67 --dport 80 -j DNAT --to-destination 192.168.1.1-192.168.1.10
Explanation	指定要写入 IP 头的地址，这也是包要被转发到的地方。上面的例

	子就是把所有发往地址 15.45.23.67 的包都转发到一段 LAN 使用的私有地址中，即 192.168.1.1 到 192.168.1.10。如前所述，在这种情况下，每个流都会被随机分配一个要转发到的地址，但同一个流总是使用同一个地址。我们也可以只指定一个 IP 地址作为参数，这样所有的包都被转发到同一台机子。我们还可以在地址后指定一个或一个范围的端口。比如：--to-destination 192.168.1.1:80 或 --to-destination 192.168.1.1:80-100。SNAT 的语法和这个 target 的一样，只是目的不同罢了。要注意，只有先用--protocol 指定了 TCP 或 UDP 协议，才能使用端口。

因为 DNAT 需要很多工作才能工作正常，所以我决定用较多的篇幅来解释它是怎么工作的。我们用一个简单的例子来说明这个工作过程，我们想通过 internet 发布我们的 web 服务，但是我们只有一个 IP 地址，并且 HTTP 服务器放置在我们的内网中，这条防火墙就有了唯一的 IP 地址，我们称之为 $INET_IP，而 HTTP 服务器有一个内网地址，我们称之为$HTTP_IP。最后防火墙增加一个内网地址$LAN_IP。那么我们在 NAT 表的 PREROUTING 链里面要做的第一件事情如下：

```
iptables -t nat -A PREROUTING --dst $INET_IP -p tcp --dport 80 -j
DNAT \
--to-destination $HTTP_IP
```

现在所有从 Internet 上面访问本机 80 端口的报文都会被转发到内部的 HTTP 服务器上面。假如你通过 Internet 做上面的测试，一切工作非常完美。假如假如你通过同一内网的其他主机访问这台 web 服务器呢？那么它就工作不正常。这其实是路由的问题。下面我们来好好分析这个问题。为了容易阅读，我们把在外网上访问我们服务器的那台机子的 IP 地址记为$EXT_BOX。

1. 包从地址为$EXT_BOX 的机子出发，去往地址为$INET_IP 的机子
2. 报文到达防火墙
3. 防火墙对这个报文做地址转换，而且这个报文还要通过其他很多链。
4. 报文离开防火墙并被转发到$HTTP_IP
5. 报文到达 HTTP 服务器，并且 HTTP 服务器对这个报文作出应答，应答包问要再次通过防火墙。当然，这要求把防火墙作为 HTTP 到达$EXT_BOX 的网关。一般情况下，防火墙就是 HTTP 服务器的缺省网关
6. 防火墙复原返回报文，这样这个报文看起来就像是防火墙自己发出的。
7. 返回报文和往常一样返回$EXT_BOX

现在我们来看看假如是和 HTTP 服务器一个网络的主机访问会出现什么问题。这儿我们定义访问的主机地址为$LAN_BOX，其他不变。

1. 报文离开$LAN_BOX 到达$INET_IP
2. 报文到达防火墙

3. 报文做 DNAT 以及其他所需的操作，但是这个报文没有做 SNAT，所以源地址仍然是一样的。
4. 报文离开防火墙并且到达 HTTP 服务器。
5. HTTP 服务器尝试回应这个把握恩，查找路由表得知是同一个网络的主机，所以就直接给这台主机发送回应报文。
6. 回复包到达客户机，但它会很困惑，因为这个包不是来自它访问的那台机子。这样，它就会把这个包扔掉而去等待“真正”的回复包，

针对这个问题有个简单的解决办法，因为这些包都要进入防火墙，而且它们都去往需要做 DNAT 才能到达的那个地址，所以我们只要对这些包做 SNAT 操作即可。比如，我们来考虑上面的例子，如果对那些进入防火墙而且是去往地址为$HTTP_IP、端口为 80 的包做 SNAT 操作，那么这些包就好象是从$LAN_IP 来的了，也就是说，这些包的源地址被改为$LAN_IP 了。这样，HTTP 服务器就会把回复包发给防火墙，而防火墙会再对包做 Un-DNAT 操作，并把包发送到客户机。解决问题的规则如下：

```
iptables -t nat -A POSTROUTING -p tcp --dst $HTTP_IP --dport 80 -j
SNAT \
--to-source $LAN_IP
```

要记住，按运行的顺序 POSTROUTING 链是所有链中最后一个，因此包到达这条链时，已经被做过 DNAT 操作了，所以我们在规则里要基于内网的地址$HTTP_IP（包的目的地）来匹配包。

警告：我们刚才写的这条规则会对日志产生很大影响，这种影响应该说是很不好的。因为来自 Internet 包在防火墙内先后经过了 DNAT 和 SNAT 处理，才能到达 HTTP 服务器（上面的例子），所以 HTTP 服务器就认为包是防火墙发来的，而不知道真正的源头是其他的 IP。这样，当它记录服务情况时，所有访问记录的源地址都是防火墙的 IP 而不是真正的访问源。我们如果想根据这些记录来了解访问情况就不可能了。因此上面提供的“简单办法”并不是一个明智的选择，但它确实可以解决“能够访问”的问题，只是没有考虑到日志而已。

其他的服务也有类似的问题。比如，你在 LAN 内建立了 SMTP 服务器，那你就要设置防火墙以便能转发 SMTP 的数据流。这样你就创建了一个开放的 SMTP 中继服务器，随之而来的就是日志的问题了。

一定要注意，这里所说的问题只是针对没有建立 DMZ 或类似结构的网络，并且内网的用户访问的是服务器的外网地址而言的。（译者注：因为如果建立了 DMZ，或者服务器和客户机又被分在不同的子网里，那就不需要这么麻烦了。因为所有访问的源头都不在服务器所在的网里，所以就没必要做 SNAT 去改变包的源地

址了，从而记录也就不是问题了。如果内网客户是直接访问服务器的内网地址那就更没事了）

较好的解决办法是为你的 LAN 在内网建立一台单独的 DNS 服务器（译者注：这样，内网的客户使用网站名访问 HTTP 服务器时，DNS 就可以把它解析成内网地址。客户机就可以直接去访问 HTTP 服务器的内网地址了，从而避免了通过防火墙的操作，而且包的源地址也可以被 HTTP 服务器的日志使用，也就没有上面说的日志问题了。），或者干脆建立 DMZ 得了（这是最好的办法，但你要有钱哦，因为用的设备多啊）。

对上面的例子应该考虑再全面些，现在还有一个问题没解决，就是防火墙自己要访问 HTTP 服务器时会发生什么，能正常访问吗？你觉得呢:)很可惜，现在的配置还是不行，仔细想想就明白了。我们这里讨论的基础都是假设机子访问的是 HTTP 服务器的外网地址，但这个外网地址其实就是防火墙对外的地址，所以当防火墙访问这个外网地址时，就是访问它自己。防火墙上如果有 HTTP 服务，那客户机就会看到页面内容，不过这不是它想看到的（它想要的在 DNAT 上了），如果没有 HTTP 服务，客户就只能收到错误信息了。前面给出的规则之所以不起作用是因为从防火墙发出的请求包不会经过那两条链。还记得防火墙自己发出的包经过哪些链吧:)我们要在 nat 表的 OUTPUT 链中添加下面的规则：

```
iptables -t nat -A OUTPUT --dst $INET_IP -p tcp --dport 80 -j DNAT \
--to-destination $HTTP_IP
```

有了最后这条规则，一切都正常了。和 HTTP 服务器不在同一个网的机子能正常访问服务了，和它在一个网内的机子也可以正常访问服务了，防火墙本身也能正常访问服务了，没有什么问题了

我想大家应该能明白这些规则只说明了数据包是如何恰当地被 DNAT 和 SNAT 的。除此之外，在 filter 表中还需要其他的规则（在 FORWARD 链里），以允许特定的包也能经过前面写的（在 POSTROUTING 链和 OUTPUT 链里的）规则。千万不要忘了，那些包在到达 FORWARD 链之前已经在 PREROUTING 链里被 DNAT 过了，也就是说它们的目的地址已被改写，在写规则时要注意这一点。

工作于 2.3/2.4/2.5/2.6

DROP target

顾名思义，如果包符合条件，这个 target 就会把它丢掉，也就是说包的生命到此结束，不会再向前走一步，效果就是包被阻塞了。在某些情况下，这个 target 会引起意外的结果，因为它不会向发送者返回任何信息，也不会向路由器返回信息，这就可能会使连接的另一方的 sockets 因苦等回音而亡:）解决这个问题的较好的办法是使用 REJECT target，（译者注：因为它在丢弃包的同时还会向发送者返回一个错误信息，这样另一方就能正常结束），尤其是在阻止端口扫描工具获得更多的信息时，可以隐蔽被过滤掉的端口等等（译者注：因为扫描工具扫描一个端口时，如果没有返回信息，一般会认为端口未打开或被防火墙等设备过滤掉了）。还要注意如果包在子链中被 DROP 了，那么它在主链里也不会再继续前进，不管是在当前的表还是在其他表里。总之，包死翘翘了。

工作于 2.3/2.4/2.5/2.6

DSCP target

这个target修改报文里面的DSCP mark，DSCP target能够设置TCP报文的DSCP值为任意你想要的。更多信息参考*RFC 2474 - Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*。

简单来说，DSCP 是一种把服务分类的方法，并且基于这些分类方法在通过路由器的时候给予不同的优先级。通过这个方法，你可以对交互性强的 TCP 会话一个交互性强的连接，例如（SSH，telnet，POP 等）。换句话讲，对于一些优先级较低的连接，例如 SMTP 等等，你可以通过一些慢速网络传递。

Table 11-6. DSCP target options

Option	--set-dscp
Example	iptables -t mangle -A FORWARD -p tcp --dport 80 -j DSCP --set-dscp 1
Explanation	--set-dscp 设置 DSCP 值为一个特定的值。这个值可以通过下面的 class 来设置，或者是一个十进制或者十六进制的数字。
Option	--set-dscp-class
Example	iptables -t mangle -A FORWARD -p tcp --dport 80 -j DSCP --set-dscp-class EF
Explanation	--set-dscp-class选项根据预定于好的服务分类设置DSCP的数

	值，一些可用的值为EF/BE/CSxx以及AFxx等等，你可以通过 *Implementing Quality of Service Policies with DSCP*网站查看更多。请注意--set-dscp-class 和 --set-dscp是互斥的，一次只能使用一种来描述。

工作于 2.3/2.4/2.5/2.6

ECN target

这个 target 主要用来修改 IPV4 报文头的 ECN 字段，例如你可以把它们复位或者设置相应的数字。因为 ECN 是一个相对新的事物，所以互联的时候可能有一些问题。例如它利用了之前 TCP 被预留的两位，而一些路由器和应用程序不会转发这两位预留字段被置位的报文。

请注意我们不要在连接的中途打开 ECN 字段，这是被 RFC 所禁止的。因为双方都需要协商 ECN，假如我们直接打开，这样其中一端主机就感知不到，所以可能就完成不了本次协商。

Table 11-7. ECN target options

Option	--ecn-tcp-remove
Example	iptables -t mangle -A FORWARD -p tcp --dport 80 -j ECN --ecn-tcp-remove
Explanation	ECN 只有一个参数，--ecn-tcp-remove 告诉 target 移除 TCP 头的 ECN 字段。

工作于 2.5/2.6

LOG target选项

这个 target 是专门用来记录包地有关信息的。这些信息可能是非法的，那就可以用来除错。LOG 会返回包的有关细节，如 IP 头的大部分和其他有趣的信

息。这个功能是通过内核的日志工具完成的，一般是 syslogd。返回的信息可用 dmesg 阅读，或者可以直接查看 syslogd 的日志文件，也可以用其他的什么程序来看。LOG 对调试规则有很大的帮助，你可以看到包去了哪里、经过了什么规则的处理，什么样的规则处理什么样的包，等等。当你在生产服务器上调试一个不敢保证 100%正常的规则集时，用 LOG 代替 DROP 是比较好的（有详细的信息可看，错误就容易定位、解决了），因为一个小小的语法错误就可能引起严重的连接问题，用户可不喜欢这样哦。如果你想使用真正地扩展日志地话，可能会对 ULOG target 有些兴趣，因为它可以把日志直接记录到 MySQL databases 或类似的数据库中。

注意，如果在控制台得到的信息不是你想要的，那不是 iptables 或 Netfilter 的问题，而是 syslogd 配置文件的事，这个文件一般都是/etc/syslog.conf。有关这个问题的更多信息请查通过 man syslog.conf 查看。

另外你也需要检查 dmesg 设置，dmesg 用来决定哪些错误会被内核显示在控制台上。 Dmesg -n 1 会阻止除了 panic 之外的所有消息。Dmesg 的消息级别和 syslogd 一样，它只能工作于内核组件的 log 消息。

LOG 现在有 5 个选项，你可以用它们指定需要的信息类型或针对不同的信息设定一些值以便在记录中使用。选项如下：

Table 11-8. LOG target options

Option	--log-level
Example	iptables -A FORWARD -p tcp -j LOG --log-level debug
Explanation	告诉 iptables 和 syslog 使用哪个记录等级。记录等级的详细信息可以查看文件 syslog.conf，一般来说有以下几种，它们的级别依次是：debug，info，notice，warning，warn，err，error，crit，alert， emerg，panic。其中，error 和 err、warn 和 warning、panic 和 emerg 分别是同义词，也就是说作用完全一样的。注意这三种级别是不被赞成使用的，换句话说，就是不要使用它们（因为信息量太大）。信息级别说明了被记录信息所反映的问题的严重程度。所有信息都是通过内核的功能被记录的，也就是说，先在文件 syslog.conf 里设置 kern.=info /var/log/iptables，然后再让所有关于 iptables 的 LOG 信息使用级别 info，就可以把所有的信息存入文件/var/log/iptables 内。注意，其中也可能会有其他的信息，它们是内核中使用 info 这个等级的其他部分产生的。有关日志的详细信息，我建议你看看 syslog 和 syslog.conf 的帮助，还有 HOWTO，等等。

Option	--log-prefix
Example	iptables -A INPUT -p tcp -j LOG --log-prefix "INPUT packets"
Explanation	告诉 iptables 在记录的信息之前加上指定的前缀。这样和 grep 或其他工具一起使用时就容易追踪特定的问题，而且也方便从不同的规则输出。前缀最多能有 29 个英文字符，这已经是包括空白字符和其他特殊符号的总长度了。
Option	--log-tcp-sequence
Example	iptables -A INPUT -p tcp -j LOG --log-tcp-sequence
Explanation	把包的 TCP 序列号和其他日志信息一起记录下来。TCP 序列号可以唯一标识一个包，在重组时也是用它来确定每个分组在包里的位置。注意，这个选项可能会带来危险，因为这些记录被未授权的用户看到的话，可能会使他们更容易地破坏系统。其实，任何 iptables 的输出信息都增加了这种危险
Option	--log-tcp-options
Example	iptables -A FORWARD -p tcp -j LOG --log-tcp-options
Explanation	记录 TCP 包头中的字段大小不变的选项。这对一些除错是很有价值的，通过它提供的信息，可以知道哪里可能出错，或者哪里已经出了错。
Option	--log-ip-options
Example	iptables -A FORWARD -p tcp -j LOG --log-ip-options
Explanation	记录 IP 包头中的字段大小不变的选项。这对一些除错是很有价值的，还可以用来跟踪特定地址的包。

工作于 2.3/2.4/2.5/2.6

MARK target

用来设置mark值，这个值只能在本地的mangle表里使用，不能用在其他任何地方，就更不用说路由器或另一台机子了。因为mark比较特殊，它不是包本身的一部分，而是在包穿越计算机的过程中由内核分配的和它相关联的一个字段。它可以和本地的高级路由功能联用，以使不同的包能使用不同的队列要求，等等。如果你想在传输过程中也有这种功能，还是用TOS target吧。有关高级路由的更多信息，可以查看Linux Advanced Routing and Traffic Control HOW-TO。

Table 11-9. MARK target options

Option	--set-mark
Example	iptables -t mangle -A PREROUTING -p tcp --dport 22 -j MARK --set-mark 2
Explanation	设置 mark 值，这个值是一个无符号的整数。比如，我们对一个流或从某台机子发出的所有的包设置了 mark 值，就可以利用高级路由功能来对它们进行流量控制等操作了。

Works under Linux kernel 2.3, 2.4, 2.5 and 2.6.

MASQUERADE target

这个 target 和 SNAT target 的作用是一样的，区别就是它不需要指定--to-source 。MASQUERADE 是被专门设计用于那些动态获取 IP 地址的连接的，比如，拨号上网、DHCP 连接等。如果你有固定的 IP 地址，还是用 SNAT target 吧。

伪装一个连接意味着，我们自动获取网络接口的 IP 地址，而不使用--to-source 。当接口停用时，MASQUERADE 不会记住任何连接，这在我们 kill 掉接口时是有很大好处的。如果我们使用 SNAT target，连接跟踪的数据是被保留下来的，而且时间要好几天哦，这可是要占用很多连接跟踪的内存的。一般情况下，这种处理方式对于拨号上网来说是较好的（这有利于已有那连接继续使用）。如果我们被分配给了一个不同于前一次的 IP，不管怎样已有的连接都要丢失，但或多或少地还是有一些连接记录被保留了（真是白痴，是吧）。

即使你有静态的 IP，也可以使用 MASQUERADE，而不用 SNAT 。不过，这不是被赞成的，因为它会带来额外的开销，而且以后还可能引起矛盾，比如它也许会影响你的脚本，使它们不能用。

注意，MASQUERADE 和 SNAT 一样，只能用于 nat 表的 POSTROUTING 链，而且它只有一个选项（不是必需的）：

Table 11-10. MASQUERADE target options

Option	--to-ports
Example	iptables -t nat -A POSTROUTING -p TCP -j MASQUERADE --to-ports 1024-31000

Explanation	在指定TCP或UDP的前提下，设置外出包能使用的端口，方式是单个端口，如--to-ports 1025，或者是端口范围，如--to- ports 1024-3000。注意，在指定范围时要使用连字号。这改变了SNAT中缺省的端口选择，详情请查看 *SNAT target*。

Works under Linux kernel 2.3, 2.4, 2.5 and 2.6.

MIRROR target

注意 MIRROR 很危险，它只是开发用来作为一个新的连接跟踪的 NAT 的例子。它可能带来危险，而且没有正确设置的话，可能导致严重的 DOS/DDOS。所以它从 2.5 版本开始被移除。

这个 target 是实验性的，它只是一个演示而已，不建议你使用它，因为它可能引起循环，除此之外，还可能引起严重的 DoS。这个 target 的作用是颠倒 IP 头中的源目地址，然后再转发包。这会引起很有趣的事，一个骇客最后攻破的可能就是他自己的机子。看来，使用这个 target 至少可以使我们的机子更强壮:) 我们如果对机子 A 的 80 端口使用了 MIRROR，会发生什么呢？假设有来自 yahoo.com 的机子 B 想要访问 A 的 HTTP 服务，那他得到的将是 yahoo 的主页，因为请求是来自 yahoo 的。

注意，MIRROR 只能用在 INPUT、 FORWARD、 PREROUTING 链和被它们调用的自定义链中。还要注意，如果外出的包是因 MIRROR target 发出的，则它们是不会被 filter、nat 或 mangle 表内的链处理的，这可能引起循环或其他问题。比如，一台机子向另一台配置了 MIRROR 且 TTL 值为 255 的机子发送一个会被认为是欺骗的数据包，同时这台机子也欺骗自己的数据包，以使它被认为好像是来自第三个使用了 MIRROR 的机子。这样，那个包就会不间断地往来很多次，直到 TTL 为 0。如果两台机子之间只有一个路由器，这个包就会往返 240-255 次。对骇客来说，这是不坏的情况，因为他只要发送一个 1500 字节的数据（也就是一个包），就可以消耗你的连接的 380K 字节。对于骇客或者叫做脚本小子（不管我们把他们称作什么）来说，这可是很理想的情况。

只能用于 2.3 和 2.4 版本，更高版本都因为它的危险性而被移除。

NETMAP target

NETMAP 是 SNAT 和 DNAT 的新实现，在转化过程过程里面做的是 1：1 的映射，即一个私有地址映射到一个公网地址，他们的主机地址部分不做更改。例如我们之前有一个 254 太主机的私有网络，后来我们有了 254 个公网地址，这时候我们不想每台机器都更改 IP 地址，那么我们在防火墙上面可以增加如下规则：

-j NETMAP -to 10.5.6.0/24，这样从防火墙上面出来的报文都变成了公网地址。

Table 11-11. NETMAP target options

Option	--to
Example	iptables -t mangle -A PREROUTING -s 192.168.1.0/24 -j NETMAP --to 10.5.6.0/24
Explanation	这是唯一的参数，在上面例子里，192.168.1.x 会直接变成 10.5.6.x。

工作于 2.5 和 2.6

NFQUEUE target

NFQUEUE target 是 QUEUE 的扩展，它允许把报文发送到不同的队列，队列通过 16bit 的数字来标识。

这个 target 需要内核的 nfnetlink_queue 运行，更多 NFQUEUE 的信息，请参见 QUEUE target。

Table 11-12. NFQUEUE target options

Option	--queue-num
Example	iptables -t nat -A PREROUTING -p tcp --dport 80 -j NFQUEUE --queue-num 30
Explanation	--queue-num 选项指定了使用哪个队列，假如没有指定这个参数，那么默认的是使用队列 0.最多可用的队列数目是 65535，队列 0 也被 QUEUE target 使用。

工作于 2.6.14 以及后续版本

NOTRACK target

NOTRACK target

这个target用来关闭连接跟踪，这个target在状态匹配的*Untracked connections and the raw table*一节有介绍。

这个 target 没有参数，所以很容易使用。首先匹配上你不想跟踪的报文，然后在这个报文上应用 NOTRACK target。

本 target 只能用在 raw 表里面。

工作于 2.6 内核

QUEUE target

这个target为用户空间的程序或应用软件管理包队列。它是和iptables之外的程序或工具协同使用的，包括网络计数工具，高级的数据包代理或过滤应用，等等。讨论程序的编码已超出了本文的范围。即使讨论，也要花很多时间，而且在这样一篇文章之内也无法说清有关Netfilter和iptables的编程。具体的信息请查看Netfilter Hacking HOW-TO。

在 2.6.14 内核里面，netfilter 的行为已经发生了变化，我们引入了一个新的处理模块叫做 nfnetlink_queue。最新的实现 QUEUE 的 target 是指向 NFQUEUE 的默认队列。

工作于 2.3/2.4/2.5/2.6

REDIRECT target

在防火墙所在的机子内部转发包或流到另一个端口。比如，我们可以把所有去往端口 HTTP 的包 REDIRECT 到 HTTP proxy（例如 squid），当然这都发生在我们自己的主机内部。本地生成的包都会被映射到 127.0.0.1。换句话说，这个 target 把要转发的包的目的地址改写为我们自己机子的 IP。我们在做透明代理（LAN 内的机子根本不需要知道代理的存在就可以正常上网）时，这个 target 可是起了很大作用的。

注意，它只能用在 nat 表的 PREROUTING、OUTPUT 链和被它们调用的自定义链里。 REDIRECT 只有一个选项：

Table 11-13. REDIRECT target options

Option	--to-ports
Example	iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080
Explanation	在指定 TCP 或 UDP 协议的前提下，定义目的端口，方式如下： 1、不使用这个选项，目的端口不会被改变。 2、指定一个端口，如--to-ports 8080 3、指定端口范围，如--to-ports 8080-8090

工作于 2.3/2.4/2.5/2.6

REJECT target

REJECT 和 DROP 基本一样，区别在于它除了阻塞包之外，还向发送者返回错误信息。现在，此 target 还只能用在 INPUT、FORWARD、OUTPUT 和它们的子链里，而且包含 REJECT 的链也只能被它们调用，否则不能发挥作用。它只有一个选项，是用来控制返回的错误信息的种类的。虽然有很多种类，但如果你有 TCP/IP 方面的基础知识，就很容易理解它们

Table 11-14. REJECT target options

Option	--reject-with
Example	iptables -A FORWARD -p TCP --dport 22 -j REJECT --reject-with tcp-reset
Explanation	告诉REJECT target应向发送者返回什么样的信息。一旦包满足了设定的条件，就要发送相应的信息，然后再象DROP一样无情地抛弃那些包。可用的信息类型有：1、icmp-net-unreachable 2、icmp-host-unreachable 3、 icmp-port-unreachable 4、icmp-proto-unreachable 5、icmp-net-prohibited 6、icmp-host-prohibited 。其中缺省的是port-unreachable。你可以在附录ICMP类型中看到更多的信息。还有一个类型——echo-reply，它只能和匹配 ICMP ping包的规则联用。最后一个类型是tcp-reset，（显然，只能用于TCP协议）它的作用是告诉REJECT返回一个TCP RST包（这个包以文雅的方式关闭TCP连接，有关它的详细信息在RFC 793 - Transmission Control Protocol里）给发送者。正如iptables的 man page中说的，tcp-reset主要用来阻塞身份识别探针（即 113/tcp，当向被破坏的邮件主机发送邮件时，探针常被用到，否则它不会接受你的信）。

工作于 2.3/2.4/2.5/2.6

RETURN target

顾名思义，它使包返回上一层，顺序是：子链——>父链——>缺省的策略。具体地说，就是若包在子链中遇到了 RETURN，则返回父链的下一条规则继续进行条件的比较，若是在父链（或称主链，比如 INPUT）中遇到了 RETURN，就要被缺省的策略（一般是 ACCEPT 或 DROP）操作了。

我们来举个例子说明一下，假设一个包进入了 INPUT 链，匹配了某条 target 为--jump EXAMPLE_CHAIN 规则，然后进入了子链 EXAMPLE_CHAIN。在子链中又匹配了某条规则，恰巧 target 是--jump RETURN，那包就返回 INPUT 链了。如果在 INPUT 链里又遇到了--jump RETURN，那这个包就要交由缺省的策略来处理了。

工作于 2.3/2.4/2.5/2.6

SAME target

SAME target 基本工作方式很像 SNAT，但是唯一的不同是它会尽量让同一台主机出去的 IP 保持一致。例如有一个/24 网段（192.168.1.0），外出 IP 有三个（10.5.6.7-9），这时 192.168.1.30 通过.7 公网地址做了 NAT 映射，那么防火墙会尽量保持这台机器后续的所有报文都通过.7 地址出去。

Table 11-15. SAME target options

Option	--to
Example	iptables -t mangle -A PREROUTING -s 192.168.1.0/24 -j SAME --to 10.5.6.7-10.5.6.9
Explanation	--to 参数指定了公网地址，其中地址段通过“-”号指定了开始和结尾地址。
Option	--nodst
Example	iptables -t mangle -A PREROUTING -s 192.168.1.0/24 -j SAME --to 10.5.6.7-10.5.6.9 --nodst
Explanation	正常情况下，SAME 通过源地址和目的地址两者来查找相应的连接，但是指定了—nodst 之后，就会只有源地址来查找连接。不使用这个参数，默认是源地址和目的地址的联合查找。

工作于 2.5/2.6

SECMARK target

SECMARK target用来在报文上面打上安全标记（mark），这个功能主要用来实现SELinux和安全系统，但是他们在Linux上面还是刚起步，但是前景可观。SELinux的讲述超出本书范围，详情请访问Security-Enhanced Linux。

简单的说 SELinux 是一种新的增强型强制认证功能，它是 NSA 提出并且实现的。SELinux 大致就是对于系统里面的每一个对象都设置一个安全属性，在

访问他们之前进行匹配。SECMARK 就是用来设置报文的安全上下文，接着用来进行比对。

SECMARK 只能用在 mangle 表。

Table 11-16. SECMARK target options

Option	--selctx
Example	iptables -t mangle -A PREROUTING -p tcp --dport 80 -j SECMARK --selctx httpcontext
Explanation	--selctx 用来实行这个报文处于哪个安全上下文，这个上下文将在 linux 的安全系统里面被匹配。

SNAT target

这个 target 是用来做源网络地址转换的，就是重写包的源 IP 地址。当我们有几个机子共享一个 Internet 连接时，就能用到它了。先在内核里打开 ip 转发功能，然后再写一个 SNAT 规则，就可以把所有从本地网络出去的包的源地址改为 Internet 连接的地址了。如果我们不这样做而是直接转发本地网的包的话，Internet 上的机子就不知道往哪儿发送应答了，因为在本地网里我们一般使用的是 IANA 组织专门指定

SNAT 只能用在 nat 表的 POSTROUTING 链里。只要连接的第一个符合条件的包被 SNAT 了，那么这个连接的其他所有的包都会自动地被 SNAT,而且这个规则还会应用于这个连接所在流的所有数据包。

Table 11-17. SNAT target options

Option	--to-source
Example	iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to-source 194.236.50.155-194.236.50.160:1024-32000
Explanation	指定源地址和端口，有以下几种方式： 1、单独的地址。 2、一段连续的地址，用连字符分隔，如 194.236.50.155-194.236.50.160，这样可以实现负载平衡。每个流会被随机分配

	一个 IP，但对于同一个流使用的是同一个 IP。 3、在指定-p tcp 或 -p udp 的前提下，可以指定源端口的范围，如 194.236.50.155:1024-32000，这样包的源端口就被限制在 1024-32000 了。 注意，如果可能，iptables 总是想避免任何的端口变更，换句话说，它总是尽力使用建立连接时所用的端口。但是如果两台机子使用相同的源端口，iptables 将会把他们的其中之一映射到另外的一个端口。如果没有指定端口范围， 所有的在 512 以内的源端口会被映射到 512 以内的另一个端口，512 和 1023 之间的将会被映射到 1024 内，其他的将会被映射到大于或等于 1024 的端口，也就是说是同范围映射。还要注意，这种映射和目的端口无关。因此，如果客户想和防火墙外的 HTTP 服务器联系，它是不会被映射到 FTP control 所用的端口的。

工作于 2.3/2.4/2.5/2.6

TCPMSS target

TCPMSS target 主要用来更改报文的 TCP 的 SYN 报文的 MSS 数值，MSS 数值用来控制特定连接的最大报文大小，正常情况下，就是 MTU 数值，减去 40 字节。引入 MSS 主要是用来解决一些 ISP 或者服务提供商组织 ICMP 需要分片报文，这样会导致很多奇怪的问题，从而从路由器/防火墙的观点来看，一些正常，但是防火墙后面的主机却不能交换大块数据。这样就可能导致例如邮件服务器能够发送小邮件，但是缺不能发送大邮件，web 浏览器能够连接上网页，但是却接收不到数据，或者 ssh 工作正常，但是 SCP 在握手成功后就挂起了。总而言之就是任何大报文都工作不正常。

TCPMSS target 就是用来解决这些问题的，主要是修改通过本条连接的报文的大小。请注意我们只需要在 SYN 报文里面设置 MSS 数值，因为只有在这之后，主机才关心 MSS 数值。

Table 11-18. TCPMSS target options

Option	--set-mss
Example	iptables -t mangle -A POSTROUTING -p tcp --tcp-flags SYN,RST SYN -o eth0 -j TCPMSS --set-mss 1460
Explanation	--set-mss 参数显式的指定外出报文的 MSS 大小，在上面的例子

	中，我们指定通过 eth 出去的 SYN 报文的 MSS 为 1460，因为没人的 MTU 是 1500，减去 40 就是 1460.MSS 只需要在 SYN 报文里面正确设置，这样对端就会自动处理 MSS。
Option	--clamp-mss-to-pmtu
Example	iptables -t mangle -A POSTROUTING -p tcp --tcp-flags SYN,RST SYN -o eth0 -j TCPMSS --clamp-mss-to-pmtu
Explanation	--clamp-mss-to-pmtu 选项自动设置 MSS 数值大小，这样你就不需要显式指定了。它会自动设置数值为 PMTU 减去 40 字节。

工作于 2.5/2.6

TOS target

TOS 是用来设置 IP 头中的 Type of Service 字段的。这个字段长一个字节，可以控制包的路由情况。它也是 iproute2 及其子系统可以直接使用的字段之一。值得注意的是，如果你有几个独立的防火墙和路由器，而且还想在他们之间利用包的头部来传递路由信息，TOS 是唯一的办法。前面说过，MARK 是不能用来传递这种信息的，它只能在本机内核内使用。如果你需要为某个包或流传递路由信息，就要使用 TOS 字段，它也正是为这个而被开发的。

Internet 上有很多路由器在这一方面并没有做好工作，因此，在发送包之前改变其 TOS 没有什么大用处。最好的情况是路由器根本不理它，最坏的情况是路由器会根据 TOS 处理，但都是错误的。然而，如果你是在一个很大的 WAN 或 LAN 里，而且有很多路由器，TOS 还是能有很好的作为的。总的来说，基于 TOS 的值给包以不同的路由和参数还是可能的，即使在网络里是受限制的

TOS target 只能设置为指定数值或者命名数值。这些预定义好的 TOS 数值可以在 ip.h 文件里面找到，原因有很多种，所以你不需要设置文件之外的其他数值。但是还有一些规避的方法，其中一个比较有名的是 Matthew G. Marsh 维护的一个补丁，但是不建议使用。

请注意这个 target 只能在 mangle 表里面只用。

在一些老的版本 1.2.2 以及更低，iptables 使用有一些限制，但是不推荐用这么低的版本。

Table 11-19. TOS target options

Option	--set-tos
Example	iptables -t mangle -A PREROUTING -p TCP --dport 22 -j TOS --set-tos 0x10
Explanation	设置 TOS 的值，值的形式可以是名字或者使相应的数值（十进制或 16 进制的）。一般情况下，建议你使用名字而不使用数值形式，因为以后这些数值可能会有所改变，而名字一般是固定的。TOS 字段有 8 个二进制位，所以可能的值是 0-255（十进制）或 0x00-0xFF（16 进制）。如前所述，你最好使用预定义的值，它们是： 1、Minimize-Delay 16 (0x10)，要求找一条路径使延时最小，一些标准服务如 telnet、SSH、FTP- control 就需要这个选项。 2、Maximize-Throughput 8 (0x08)，要求找一条路径能使吞吐量最大，标准服务 FTP-data 能用到这个。 3、Maximize-Reliability 4 (0x04)，要求找一条路径能使可靠性最高，使用它的有 BOOTP 和 TFTP。 4、Minimize-Cost 2 (0x02)，要求找一条路径能使费用最低，一般情况下使用这个选项的是一些视频音频流协议，如 RTSP（Real Time Stream Control Protocol）。 5、Normal-Service 0 (0x00)，一般服务，没有什么特殊要求。这个值也是大部分包的缺省值。 完整的列表可以通过命令 **iptables -j TOS -h** 得到。在 1.2.5 版时，这个列表就已经是完整的了，而且会保持很长一段时间。

工作于 2.3/2.4/2.5/2.6

TTL target

TTL 可以修改 IP 头中 Time To Live 字段的值。它有很大的作用，我们可以把所有外出包的 Time To Live 值都改为一样的，比如 64，这是 Linux 的默认值。有些 ISP 不允许我们共享连接（他们可以通过 TTL 的值来区分是不是有多个机子使用同一个连接），如果我们把 TTL 都改为一样的值，他们就不能再根据 TTL 来判断了。

关于任何设置Linux的TTL默认值，请参阅附录其他资源和链接 内的ip-sysctl.txt。

TTL 只能在 mangle 表内使用，它有 3 个选项：

Table 11-20. TTL target options

Option	--ttl-set
Example	iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-set 64
Explanation	设置 TTL 的值。这个值不要太大，也不要太小，大约 64 就很好。值太大会影响网络，而且有点不道德，为什么这样说呢？如果有些路由器的配置不太正确，包的 TTL 又非常大，那它们就会在这些路由器之间往返很多次，值越大，占用的带宽越多。这个 target 就可以被用来限制包能走多远，一个比较恰当的距离是刚好能到达 DNS 服务器。
Option	--ttl-dec
Example	iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-dec 1
Explanation	设定 TTL 要被减掉的值，比如--ttl-dec 3。假设一个进来的包的 TTL 是 53，那么当它离开我们这台机子时，TTL 就变为 49 了。为什么不是 50 呢？因为经过我们这台机子，TTL 本身就要减 1，还要被 TTL target 再减 3，当然总共就是减去 4 了。使用这个 target 可以限制“使用我们的服务的用户”离我们有多远。比如，用户总是使用比较近的 DNS，那我们就可以对我们的 DNS 服务器发出的包进行几个--ttl-dec。（译者注：意思是，我们只想让距离 DNS 服务器近一些的用户访问我们的服务）当然，用--set-ttl 控制更方便些。
Option	--ttl-inc
Example	iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-inc 1
Explanation	设定TTL要被增加的值，比如--ttl-inc 4。假设一个进来的包的

	TTL是 53，那么当它离开我们这台机子时，TTL应是多少呢？答案是 56,原因同--ttl-dec。使用这个选项可以使我们的防火墙更加隐蔽，而不被trace-routes发现，方法就是设置--ttl-inc 1。原因应该很简单了，包每经过一个设备，TTL就要自动减 1，但在我们的防火墙里这个 1 又被补上了，也就是说，TTL的值没变，那么trace-routes就会认为我们的防火墙是不存在的。Trace-routes让人又爱又恨，爱它是因为在连接出问题时，它可以给我们提供极有用的信息，告诉我们哪里有毛病；恨它是由于它也可以被黑客或骇客用来收集目标机器的资料。怎么使用它呢？这里有个很好的例子，请看脚本*Ttl-inc.txt*。

工作于 2.3/2.4/2.5/2.6

ULOG target

ULOG可以在用户空间记录被匹配的包的信息，这些信息和整个包都会通过 netlink socket被多播。然后，一个或多个用户空间的进程就会接受它们。换句话说，ULOG是至今iptables和Netfilter下最成熟、最完善的日志工具，它包含了很多更好的工具用于包的记录。这个target可以是我们把信息记录到MySQL或其他数据库中。这样，搜索特定的包或把记录分组就很方便了。你可以在*ULOGD project page*里找到ULOGD用户空间的软件。

Table 11-21. ULOG target options

Option	--ulog-nlgroup
Example	iptables -A INPUT -p TCP --dport 22 -j ULOG --ulog-nlgroup 2
Explanation	指定向哪个 netlink 组发送包，比如-- ulog-nlgroup 5。一个有 32 个 netlink 组，它们被简单地编号位 1-32。默认值是 1。
Option	--ulog-prefix
Example	iptables -A INPUT -p TCP --dport 22 -j ULOG --ulog-prefix "SSH connection attempt: "
Explanation	指定记录信息的前缀，以便于区分不同的信息。使用方法和 LOG 的 prefix 一样，只是长度可以达到 32 个字符。
Option	--ulog-cprange
Example	iptables -A INPUT -p TCP --dport 22 -j ULOG --ulog-cprange 100

Explanation	指定每个包要向“ULOG 在用户空间的代理”发送的字节数，如--ulog-cprange 100，表示把整个包的前 100 个字节拷贝到用户空间记录下来，其中包含了这个包头，还有一些包的引导数据。默认值是 0，表示拷贝整个包，不管它有多大。
Option	--ulog-qthreshold
Example	iptables -A INPUT -p TCP --dport 22 -j ULOG --ulog-qthreshold 10
Explanation	告诉 ULOG 在向用户空间发送数据以供记录之前，要在内核里收集的包的数量（译者注：就像集装箱），如--ulog-qthreshold 10。这表示先在内核里积聚 10 个包，再把它们发送到用户空间里，它们会被看作同一个 netlink 的信息，只是由好几部分组成罢了。默认值是 1，这是为了向后兼容，因为以前的版本不能处理分段的信息。

工作于 2.3/2.4/2.5/2.6

下章预览

本章详细讨论了 linux 里面可用的 target，这个列表随着参与者的增长，正在急剧膨胀。本章还讨论了不同 target 的选项。

下一章我们主要就讲述如何调试你的脚步以及通过什么技术来调试。它同时提供一些基本调试技术例如 bash 和 echo，还有一些高级技巧例如 nmap 和 nessus 等。

第十二章 调试你的脚本

编写自己的脚本一个重要的环节就是怎么调试自己的脚本以及怎么找出其中的问题。本章将会介绍一些基本的调试以及定位手段，还有万一因为脚本问题不不能连接上防火墙的时候，应该怎么处理。

调试是必须的

一般而言，对于 iptables/netfilter 甚至是许多防火墙，调试或多或少都是必须的。99%的防火墙问题都是因为认为的策略造成的，我可以保证，你写规则的时候及其容易犯错误，而且很多时候这些错误通过肉眼很难定位。

除了这些，你可能还会犯另外一个错误，那就是编写的规则让你自己不能连接上防火墙。这个问题在使用规则前我们也可以通过一些办法来发现。充分利用基本语言以及系统提供的功能，我们就能够充分的调试这些脚本。

Bash调试技巧

通过 bask 你有多种方法来调试你的脚本，发现 bug 的第一步就是知道问题出在哪一行，这可以通过 bash -x 或者增加打印语句来实现。例如：

```
...
echo "Debugging message 1."
...
echo "Debugging message 2."
...
```

在上面的例子里，我们可以用过观察打印语句来定位基本问题在哪儿，例如我们只看到 Debugging message 1.，那么问题就可能在下面的两句。通过打印至少可以缩小定位的范围。根据我的经验，我会打印一些无用的信息，接着通过 grep 或者类似的搜索程序来定位。

另外一个方法就是通过-x 选项。 Bash -x 用来在标准输出上面打印执行的每一行语句。

```
#!/bin/bash

Into the line below:

#!/bin/bash -x
```

通过上面的设置，你就可以看到执行的每一行命令。另外一点很不错的是，bash 执行的每一行语句前面都有一个“+”号，这样就和实际执行的错误/告警输出区别开。

-x 还可以用来解决一些复杂的问题，例如：

```
#!/bin/bash
iptables="/sbin/iptables"
$iptables -N output_int_iface
cat /etc/configs/machines | while read host; do
```

```
    $iptables -N output-$host
    $iptables -A output_int_iface -p tcp -d $host -j output-$host

    cat /etc/configs/${host}/ports | while read row2; do
      $iptables -A output-$host -p tcp --dport $row2 -d $host -j
ACCEPT
    done
  done
```

这些规则看起来足够简单了，但是我们总会发现运行有问题。通常情况下我们运行会得到下面的错误：

```
work3:~# ./test.sh
Bad argument `output-'
Try `iptables -h' or 'iptables --help' for more information.
cat: /etc/configs//ports: No such file or directory
```

这次我们通过-x 选项再次运行这些脚本。我们再看看输出，我们看到很多$host 和$row2 变量没有指定值，接着分析一下可能是编程问题或者数据错误。在我们这个例子里面是数据错误，修正即可。但是我想说明的是通过这个-x 选项我们就能够清楚的知道程序执行过程。

```
work3:~# ./test.sh
+ iptables=/sbin/iptables
+ /sbin/iptables -N output_int_iface
+ cat /etc/configs/machines
+ read host
+ /sbin/iptables -N output-sto-as-101
+ /sbin/iptables -A output_int_iface -p tcp -d sto-as-101 -j output-
sto-as-101
+ cat /etc/configs/sto-as-101/ports
+ read row2
+ /sbin/iptables -A output-sto-as-101 -p tcp --dport 21 -d sto-as-101
-j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-101 -p tcp --dport 22 -d sto-as-101
-j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-101 -p tcp --dport 23 -d sto-as-101
-j ACCEPT
+ read row2
+ read host
+ /sbin/iptables -N output-sto-as-102
```

```
+ /sbin/iptables -A output_int_iface -p tcp -d sto-as-102 -j output-
sto-as-102
+ cat /etc/configs/sto-as-102/ports
+ read row2
+ /sbin/iptables -A output-sto-as-102 -p tcp --dport 21 -d sto-as-102
-j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-102 -p tcp --dport 22 -d sto-as-102
-j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-102 -p tcp --dport 23 -d sto-as-102
-j ACCEPT
+ read row2
+ read host
+ /sbin/iptables -N output-sto-as-103
+ /sbin/iptables -A output_int_iface -p tcp -d sto-as-103 -j output-
sto-as-103
+ cat /etc/configs/sto-as-103/ports
+ read row2
+ /sbin/iptables -A output-sto-as-103 -p tcp --dport 21 -d sto-as-103
-j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-103 -p tcp --dport 22 -d sto-as-103
-j ACCEPT
+ read row2
+ /sbin/iptables -A output-sto-as-103 -p tcp --dport 23 -d sto-as-103
-j ACCEPT
+ read row2
+ read host
+ /sbin/iptables -N output-
+ /sbin/iptables -A output_int_iface -p tcp -d -j output-
Bad argument `output-'
Try `iptables -h' or 'iptables --help' for more information.
+ cat /etc/configs//ports
cat: /etc/configs//ports: No such file or directory
+ read row2
+ read host
```

最后一种情况就是你通过 ssh 等工具登陆到防火墙上运行配置，但是在中途控制台挂住了，这大部分情况下都是因为规则太多，你通过-x 就可以看到运行到哪一行。但是不幸的是，这不是所有的可能，另外一种可能是我们的防火墙规则配置错误，但是服务器通过 echo 发送会回应包，所以这条连接被成功建立，所以只有 incoming 的流量还是能够发送。

通过上面你可能已经看到，在我们最后调试的时候已经变得很复杂了，另外就是我们还经常需要通过 SSH 等远程调试工具连接防火墙，这样一旦一条规则配置错误，而且没有做好预防措施的话，我们可能就得走到 50 公里外的设备旁直接调试。

系统调试工具

其中一种预防措施就是设定一个周期性的脚本来重置防火墙的策略，只到你确认这些脚本工作正常。我们一般通过 cron 来运行周期性的任务。Cron 的命令行像下面一样，而且常常和 crontab -e 合用。

```
*/5 * * * * /etc/init.d/rc.flush-iptables.sh stop
```

请再次确认，上面的脚本的确如你期望的周期性工作。

另外一个调试脚本经常用的工具是 syslog，这个工具可以记录所有的信息日志，事实上，基本上所有的大型软件都支持 syslog，内核也包括在内。所有发给 syslog 的信息都有两个基本变量，分别是设备和信息的级别/优先级。

设备告诉 syslog 是哪一个程序发送的日志，系统里面有很多这样的程序，但是我们关注的仅仅是 kernel 设备，因为所有 netfilter 发送的消息都是通过这个设备发出。

消息级别告诉 syslog 这条信息的优先级，系统里有如下预定义好的级别：

1. debug
2. info
3. notice
4. warning
5. err
6. crit
7. alert
8. emerg

Depending on these priorities, we can send them to different log files using the syslog.conf. For example, to send all messages from the kern facility with warning priority to a file called /var/log/kernwarnings, we could do as shown below. The line should go into the /etc/syslog.conf.

根据这些优先级我们通过 syslog.conf 文件，可以让它们发送到不同文件。例如我们让所有内核发出的 warning 级别消息保存到/var/log/kernelwarnings 文件。我们可以通过把下面的配置增加到 syslog.conf 文件来完成这个工作。

kern.warning /var/log/kernwarnings

正如你所见，它很简单。这样你就可以用过/var/log/kernelwarnings 查看消息了，当然这还的依赖 netfilter 发出消息的级别，这个可以用过—log-level 选项来调节。

满足你设定的 log 规则的报文都会被记录到这个文件，有了这些信息你就能够查看是否有错误发生。例如你可以在每条链的最后设定 log rules，观察有多少报文满足默认策略。一条 log 信息具有下面所列的格式：

```
 Oct 23 17:09:34 localhost kernel: IPT INPUT packet died: IN=eth1
OUT=
MAC=08:00:09:cd:f2:27:00:20:1a:11:3d:73:08:00 SRC=200.81.8.14
DST=217.215.68.146
LEN=78 TOS=0x00 PREC=0x00 TTL=110 ID=12818 PROTO=UDP SPT=1027 DPT=137
LEN=58
```

你可能已经理解了，syslog 是你调试规则的有力帮手。通过这些 logs 你就能够明白为什么你打开的一些端口工作不正常。

Iptables调试

Iptables 很多时候只能做简单调试，因为 iptables 的输出错误不是很友好。基于这个理由，也许我们可以看一些 iptables 的常见错误以及理解为什么会得到这些错误。

首先说说"Unknown arg"错误，这个错误的原因有很多种，例如：

```
work3:~# iptables -A INPUT --dport 67 -j ACCEPT
iptables v1.2.9: Unknown arg `--dport'
Try `iptables -h' or 'iptables --help' for more information.
```

这个错误很容易解决，因为我们只用了一个参数，但是通常情况下我们可能一次输入多个参数。本问题就是我们没有用—protocol 指定协议，这样—dport 不是对于所有协议都是有效的，所以增加协议即可解决这个问题。

另外一个常见的错误是遗漏了“-”符号，下面就是一个例子，我们只需要增加“-”即可。

```
work3:~# iptables -A INPUT --protocol tcp -dport 67 -j ACCEPT
Bad argument `67'
Try `iptables -h' or 'iptables --help' for more information.
```

最后，就是简单的拼写错误，当然也相当常见。这个错误消息和我们之前的没有输入提前错误是一样的，所以你需要仔细辨别。

```
work3:~# iptables -A INPUT --protocol tcp --destination-ports 67 -j
ACCEPT
iptables v1.2.9: Unknown arg `--destination-ports'
Try `iptables -h' or 'iptables --help' for more information.
```

另外还有一些原因可能导致"Unknown arg"错误，假如你的拼写正确而且规则提前也都完备，那么另外一个可能的错误原因就是应用程序和内核版本不匹配，例如我们忘记在内核编译 filter 表格，这样我们就可能遇到的错误。

```
work3:~# iptables -A INPUT -j ACCEPT
iptables v1.2.9: can't initialize iptables table `filter': Table does
not exist
(do you need to insmod?)
Perhaps iptables or your kernel needs to be upgraded.
```

通常而言，iptables 会自动加载特定的模块到内核，所以要么你重启内核后没有做 depmod，要么你忘记了编译这个模块。假如这个出问题的模块是一个 match 模块，那么错误消息就会稍微难于理解了。

```
work3:~# iptables -A INPUT -m state
--state ESTABLISHED -j ACCEPT
iptables: No chain/target/match by that name
```

在这种情况下，我们忘记了编译 state 模块，但是错误消息不是很容易理解。不过它的确给你了正确的错误提示。最后我们还是说说这个错误，只不过这一次是 target 没有编译。通过这个错误你可能就明白了，对于这种错误很难追寻，因为他们都是完全一样的。

```
work3:~# iptables -A INPUT -m state
--state ESTABLISHED -j REJECT
iptables: No chain/target/match by that name
```

最简单的办法就是查看我们是否忘记了 depmod，或者这个模块在下面目录下是否丢失。目录如下/lib/modules/2.6.4/kernel/net/ipv4/netfilter。所有 ipt_*开头的文件，大写的都是 targets，而所有的小写字符都是 matches。例如 ipt_REJECT.ko 就是一个 target，而 ipt_reject.ko 则是一个 match。

在 2.4 以及老的内核里面，内核模块的扩展名为.o，而在新的 2.6 内核里面变成.ko

另外一个解决问题的办法就是注释掉整条链，然后查看问题是否解决，在你不知道问题在哪儿的情况下，这可能是一个很高效的办法。删除整条链并且设置默认策略为 ACCEPT，接着测试，如果报文仍然不同就是别的链有错误，按照这个办法继续排除。反之如何报文通了，那就是这条链有问题，再深入研究看看问题在哪儿。

其他调试工具

在调试防火墙脚本的时候，当然还有其他很多有用的工具。本节简要介绍这些常见的调试工具，我下面要说的主要是 nmap 和 nessus。

Nmap

Nmap 是一个检查防火墙的强有力工具，它能够查看端口使用情况以及读取等多的底层信息。它还支持操作系统识别，端口扫描，支持 IPV4 以及 IPV6 等等。

The basic form of scanning is done with a very simple commandline syntax. Don't forget to specify which ports to scan through with the -p option, for example -p 1-1024. As an example, take a look below.

最基本的扫描通过下面的命令就可以检查，别忘记-p 指定扫描端口，例如-p 1-1024，看个例子：

```
blueflux@work3:~$ nmap -p 1-1024 192.168.0.1

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-03-18
17:19 CET
Interesting ports on firewall (192.168.0.1):
(The 1021 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
587/tcp open  submission

Nmap run completed -- 1 IP address (1 host up) scanned in 3.877
seconds
```

Nmap 还能够进行操作系统识别，操作系统识别需要特权用户才能使用，看个例子：

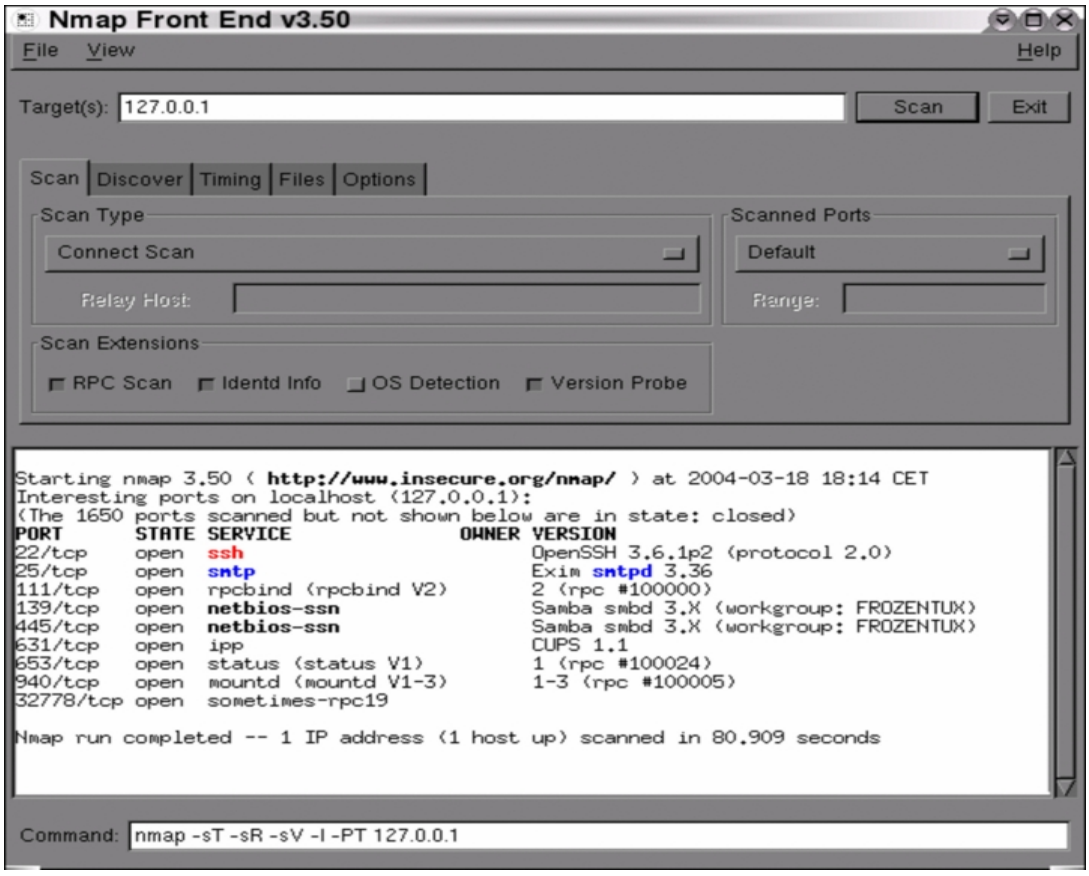
```
work3:/home/blueflux# nmap -O -p 1-1024 192.168.0.1

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-03-18
17:38 CET
Interesting ports on firewall (192.168.0.1):
(The 1021 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
587/tcp open  submission
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux Kernel 2.4.0 - 2.5.20
Uptime 6.201 days (since Fri Mar 12 12:49:18 2004)

Nmap run completed -- 1 IP address (1 host up) scanned in 14.303
seconds
```

操作系统识别不是很完美，但是它能够帮助你缩小范围。它也能够被黑客所利用，所以要注意给你的操作系统加固。

Nmap 还有一个图形化的用户界面，我们称之为 nmapfe。它是一个很不错的界面，假如你想用一些复杂的查找，它也许能够帮得上忙。看个例子：



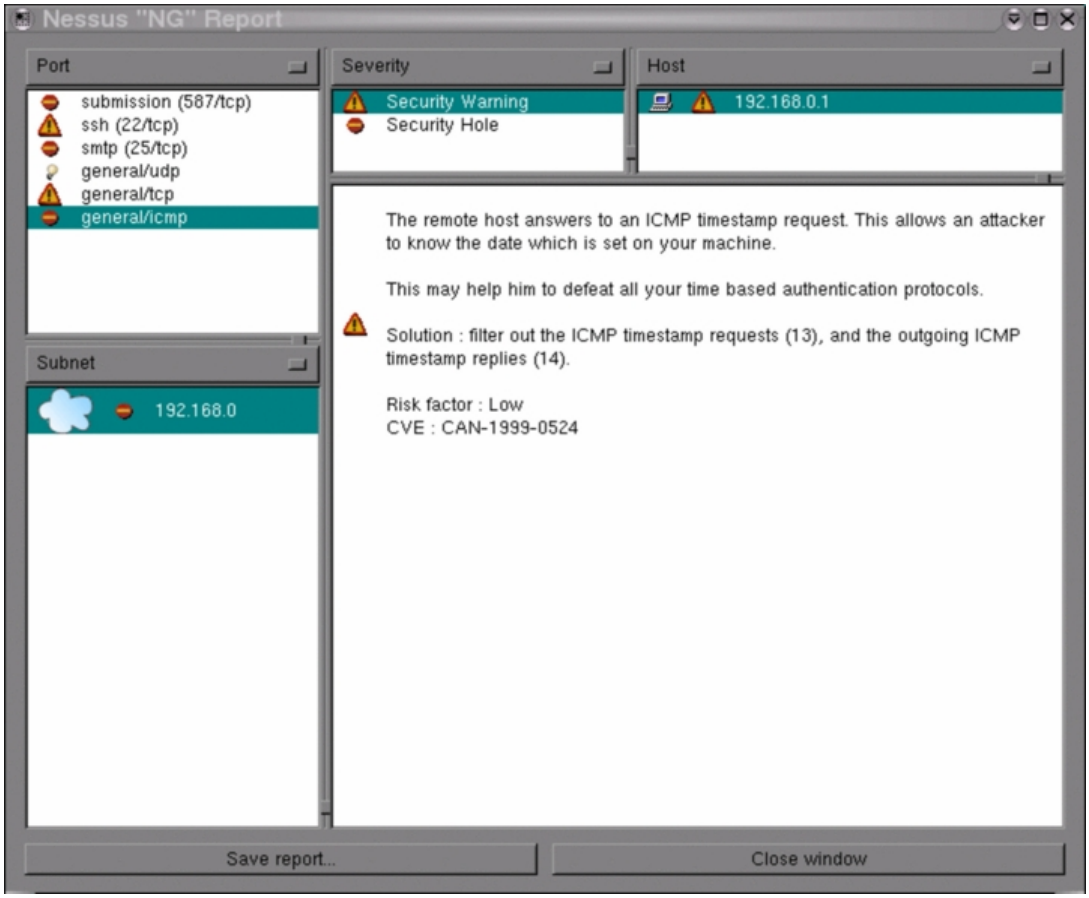
当然nmap的功能远远不止这一些，更多请参考nmap主页。

可能你已经知道了，这是一个极好的主机测试工具，用它可以观察机器的端口情况。例如，配置结束后，用 nmap 扫描一下机器看看配置是否生效。

Nessus

Nmap 是一个底层的扫描器，用来查看端口，操作系统等等信息，nmap 会尝试连接端口来发现更多信息。Nessus 则更进一步扫描机器，发现所有打开的端口，端口上的服务，程序的版本以及对该程序进行安全测试，最终输出一个报文。

现在明白了吧，这个工具对于发现你机器的安全隐患更为有用。这个程序是服务器/客户端模式，通过外部的 nessus 守护进程更为容易获得更多信息，内部也是如此。客户端是一个图形化的工具，你通过它能够登陆到守护进程，增加你的配置以及设定扫描端口等等。下面是一个报告：



Nessus 使用时候需要小心，因为一些攻击脚本可能会把机器搞死，但是幸运的是这些脚本默认是关闭的。

下章预览

本章我们介绍了一些调试脚本的基本方法。调试脚本是一个乏味漫长但是又是必须的过程。我们介绍了下面这些主题：

- Bash 在调试中的作用
- 系统调试工具
- Iptables 调试手段
- 一些其他调试工具

第十三章 rc.firewall文件

可能另外还有更为便捷的方法来创建脚本，但是这些脚本都是便于读者的阅读而写的，这样就不需要知道太多的 bash 脚本知识。

---

rc.firewall例子

现在所有东西都准备好了，rc.firewall.txt 非常的大而且没有太多的注释，在继续本文前，我建议你通读脚本，大致知道他的功能，接着在继续阅读本文。

---

rc.firewall解释

配置选项

首先要注意的是 rc.firewall.txt 里面的配置一节，它里面包含了和你实际环境相关的一些变量，所以使用前需要更改。例如你的 IP 地址总是不一样的，所以$INET_IP 总是一个合法的地址，如果你没有可能需要的脚本是

rc.DHCP.firewall.txt。另外$INET_IFACE 变量总是指向你使用的实际数据，这个可能是 eth0,eth1,ppp0,tr0 等等。

本脚本没有包含 DHCP 或者 PPPoE 的任何配置，所以这些章节都是空白。但是留着这些空白，这样你能够看到脚本之间的差异。另外如果你的确需要这些信息，请把其他脚本嵌在这儿或者甚至是从空白重新创建一个。

本地网络章节包含了你的 LAN 大部分配置选项，当然这些都是必须的。例如你需要指定连接上 LAN 的物理接口的 IP 地址。

另外这儿有一个本地主机匹配，请确认没有特殊需求不要更改这里面的内容。另外这一节还有一个$IPTABLES 变量，它指向 iptables 应用程序的实际位置，这个值会有些变化。编译 iptables 的默认路径是/usr/local/sbin/iptables，但是有一些发行版放置在/usr/sbin/iptables。

加载外部模块

首先我们通过/sbin/depmod -a command 命令查看模块依赖文件是否最新，接着我们加载脚本需要的模块。例如我们需要加载 LOG, REJECT and MASQUERADE 这几个 target。用如下命令：

```
/sbin/insmod ipt_LOG
/sbin/insmod ipt_REJECT
/sbin/insmod ipt_MASQUERADE
```

在这些脚本里面，我们强制加载模块，这样可能会导致加载模块的失败。假如模块加载失败，它可能因为很多因素并且产生一个错误信息。最大的可能性是模块或者功能已经被静态编译进内核，这时候加载模块也会产生错误。

下一个选项是加载ipt_owner模块，这个模块用来让特定用户访问特定连接。我不会使用这个模块，但是你可以只让root用户能够执行连接到redhat.com的ftp和http连接，其他都会被DROP。你也可以只让你和root连接到internet。也许对于其他人来说很恶心，但是这比较安全的拒绝黑客攻击。更多信息，请查看构建规则章节的Owner match。

我们加载状态匹配模块，所有状态匹配和连接工具代码个别比ip_conntrack_*和 ip_nat_*开头。连接跟踪 helper 是一些特殊的模块，它们能够告诉内核怎么跟踪一些特殊的连接。没有这些 helper，内核就不知道怎么跟踪一些特殊协议。NAThelper 是连接跟踪 helper 的扩展，它告诉内核怎么检查报文以及如何做转换让这些连接工作正常。例如 FTP 是一个复杂协议，它通过数据发送控制信息。

所以假如一个做了 NAT 的主机连接到 FTP 服务器，它就会把自己的本地地址放置在报文的荷载里面，并且告诉 FTP 服务器连接到这个地址。因为这个本地地址在外网无效，所以 FTP 服务器就不知道怎么处理这个连接，这样连接就中断了。FTP 的 NAT helper 模块能够做这些转换，从而 FTP 服务器知道连接哪台机器。另外 FCC/IRC 和 FTP 一样，都需要一个 helper 模块才能正常的通过 NAT 网关。

假如你在自己的防火墙上遇到mRIC DCCs问题，但是所有其他IRC客户端工作正常，请阅读附录Common problems and questions里面mIRC DCC problems 章节。

到目前位置，想让这些模块工作正常主要加载相应的协议模块，更多的解释请参阅附录的Common problems and questions。在patch-o-matic补丁里面，还有更多的协议模块，如何编译这些补丁，请阅读准备一节。

Note that you need to load the ip_nat_irc and ip_nat_ftp if you want Network Address Translation to work properly on any of the FTP and IRC protocols. You will also need to load the ip_conntrack_irc and ip_conntrack_ftp modules before actually loading the NAT modules. They are used the same way as the conntrack modules, but it will make it possible for the computer to do NAT on these two protocols.

---

proc设置

此时我们开启转发功能，命令如下：

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

是开启转发功能前三思，因为在应用规则前可能已经有成百上千的报文通过我们设备了，所以正常情况下，请先应用规则，然后再开启转发。

万一你需要动态申请地址，例如你用 SLIP/PPP/DHCP 等等，命令如下：

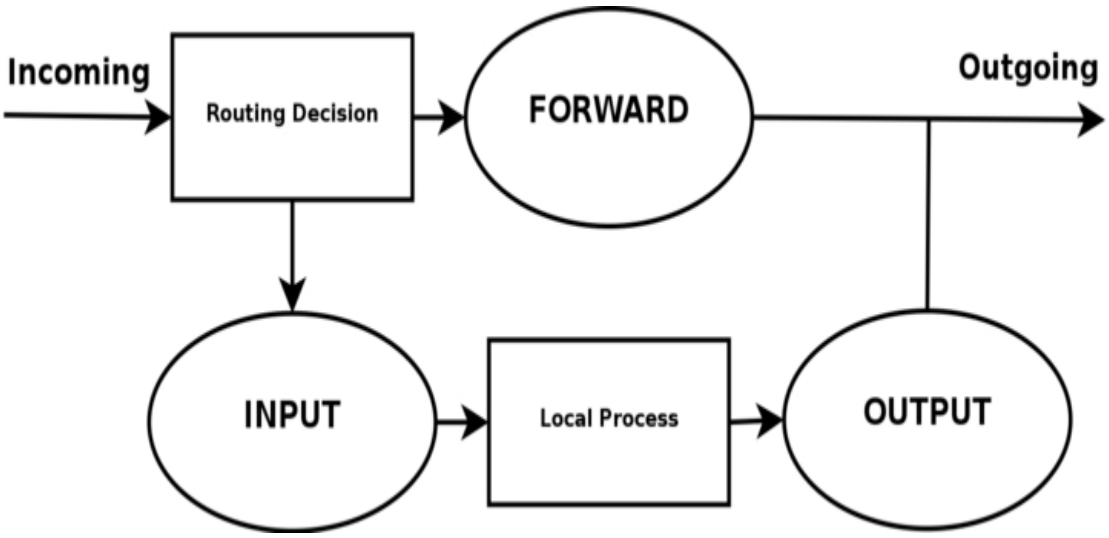
```
echo "1" > /proc/sys/net/ipv4/ip_dynaddr
```

Rc.firewall.txt 脚本以及本指南里面的其他脚本都包含 proc 设置章节，在脚本工作不正常时候可以检查里面的设置，但是在没有清楚他们用法前不要尝试修改。

规则位置的优化

本节简要地描述了针对脚本rc.firewall.txt，我将如何选择、使用内建的链链和自定义的链。我选择过的一些路径从这个或那个角度看可能是错误的，我会指出这些情况和问题发生在何时何地。这里还对章节 *表和链* 做了简要的回顾，希望能给你一点儿提醒，以使你能想起在实际应用中包是如何表和链的。

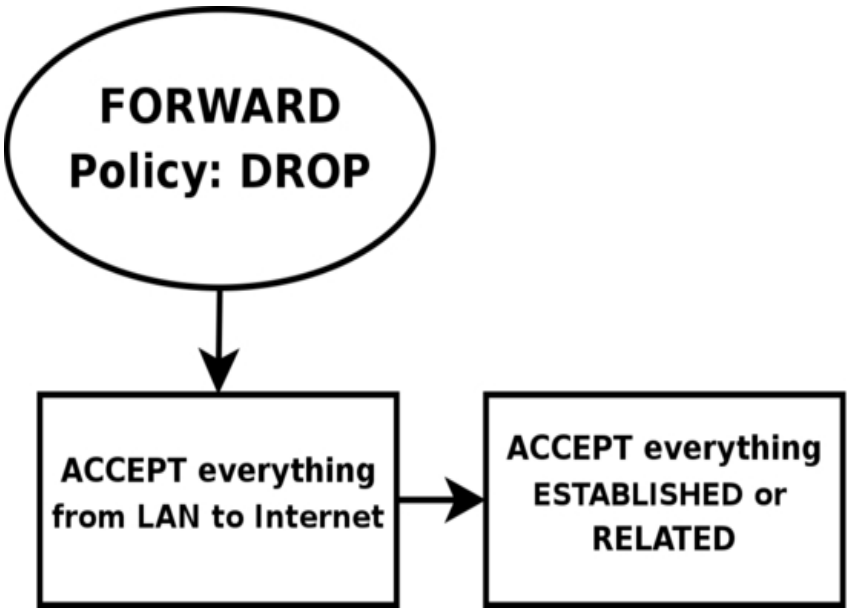
为了尽可能地少占用CPU，我们已经替换了所有不同的自定义链，与此同时，我把主要的精力放在了安全性和易读性上。我不让TCP包去经历ICMP、UDP和TCP规则的洗礼，而是简单地匹配所有的TCP包，然后让它去一个自定义链中旅行。这种方法并不比让它经历所有的规则开销大。下图可以解释在Netfilter中，外来的包是如何被处理的（相对于章节*表和链*的深入讨论，这个图形太粗糙了）。我希望通过上面的解释和下面的图形能让大家明白写这个脚本的目的，详细的注释在后面几节。



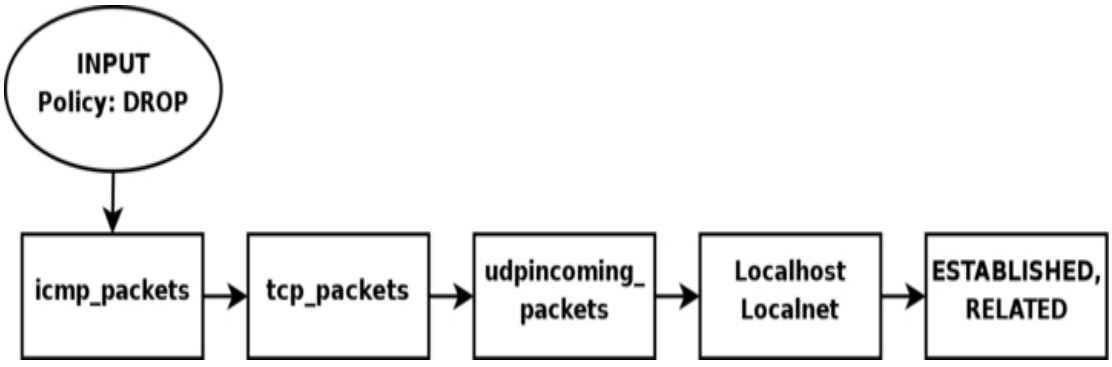
利用这个图形，我们可以弄清楚脚本的目的。整个脚本基于这样一种假设，我们有一个局域网，一个防火墙及一个 Internet 连接，且有一个静态 IP 地址（相对的是动态地址，它们使用的连接是 DHCP、PPP、 SLIP，等等），还想把防火墙作为 Internet 上的一台服务器来运行某些服务。我们完全信任局域网，因此不能阻塞任何从局域网发出的数据传输。还有一个要优先考虑的事，我们只允许那些被明确说明为可以接受的数据通过。为了做到这一点，我们就

要把缺省策略设为 DROP。这样，那些没有被明确标识为允许进入的数据就都被阻塞了。

在上面的假设里，我们想让局域网能够访问 Internet。因为局域网是被完全信任的，所以我们应该允许所有来自局域网的数据通过。但 Internet 是不被信任的，所以我们想阻塞从 Internet 向我们的局域网发起的连接。根据上面的所有假设，我们来考虑考虑需要做什么、不需要做什么以及我们想做什么。



首先，我们解决的是局域网要能连接到 Internet 的问题。那我们就要对所有数据包做 NAT 操作，因为局域网内的机子都没有真实的 IP 地址。NAT 是在 PREROUTING 链中完成的，这也是脚本最后创建的那个规则所在的链。这意味着我们必须要在 FORWARD 链中做过滤工作，否则我们就是允许所有外部的机子都能完全访问局域网了。因为我们完全信任局域网，所以允许所有由内向外的数据通过。由于我们假设 Internet 上的机子都不可以访问局域网内的机子，所以要阻塞所有由外向内的连接，但已经建立的或相关的连接除外，因为它们只是用来回应内网对外网的访问，而不是建立对内网的新连接。

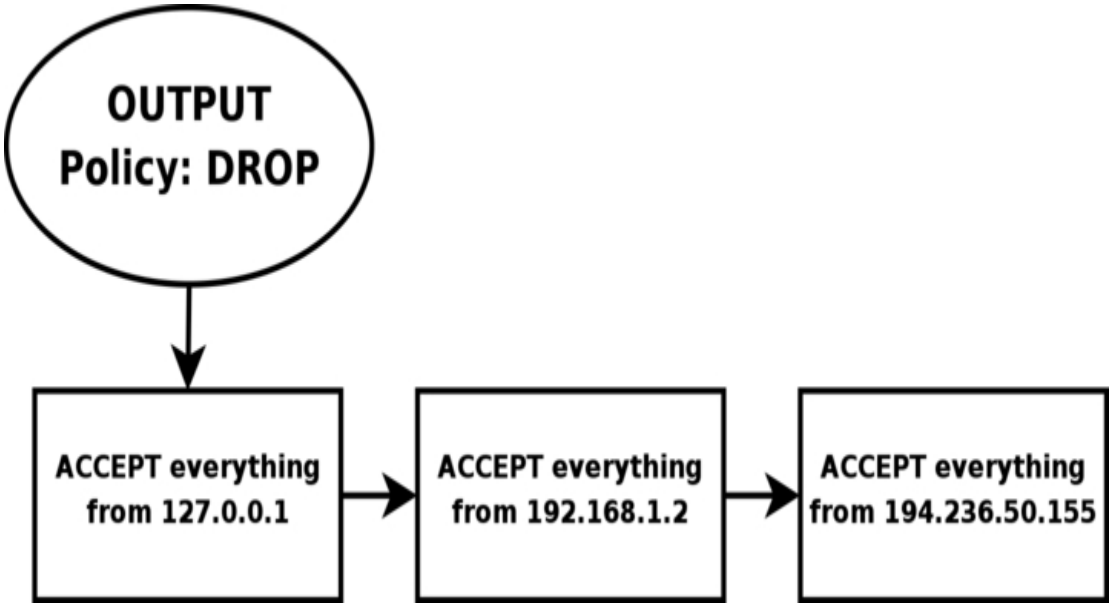


由于资金有限，我们的防火墙只提供了有限的几个服务：HTTP、 FTP、SSH和IDENTD。因此，我们要在INPUT链里允许这些协议通过，还要在 OUTPUT链里允许返回的数据通过。我们除了完全信任局域网，也信任loopback和它的IP地址，因此我们要有相应的规则来允许所有来自局域网和loopback的数据通过。但是我们不会允许一些特殊的包或包头通过，也不会接受Internet上某一段IP的访问。比如，网段 10.0.0.0/8 是为局域网而保留的，一般来说，我们不允许来自它们的包进入，因为这样的包 90%都是用来进行欺骗的。不过，在实现这条标准之前，还要注意一个问题，就是有一些ISP在他们的网络里使用的恰恰就是这些地址。 在附录常见问题与解答里有这个问题的进一步说明。

因为我们在防火墙上运行 FTP 服务，而且想让包经历最少的规则，所以要把处理 established 和 related 状态的规则放到 INPUT 链的顶部。基于同样的原因，我们把这些规则分到子链中。这样，包就可以尽量少地穿越规则，从而节省时间，也可以降低网络的冗余。

在这个脚本里，我们依据不同的协议（如 TCP、 UDP 或 ICMP）把包分到子链中。用来匹配 TCP 包的链叫做 tcp_packets，它可以匹配所有我们允许通过的 TCP 端口和子协议（如 FTP、HTTP 等）。我们还要建立一个名为 allowed 的子链，以便在真正接受“那些使用有效端口来访问防火墙的 TCP 包”之前，对它们进行附加的检查。至于 ICMP 包，自有称作 icmp_packets 的链来处理。在决定如何建立这个链时，我考虑到如果我们同意接受 ICMP 包的类型和代码，就没有必要对它们做附加的检查，所以直接接受它们就行了。最后，UDP 包由谁处理呢？当然就是 udp_packets 了。如果包是那种允许被接收的类型，就直接放行了。

因为我们的网络很小，所以防火墙也要作为工作站来用。这就要求我们要允许一些特殊的协议能和它通信，比如 speak freely 和 ICQ。



现在，我们来考虑考虑 OUTPUT 链。因为很信任防火墙，所以我们允许几乎所有离开它的包通过，而没有阻塞任何用户和协议。但我们也不想让人利用这台机子进行 IP 欺骗，因此我们只放行那些从防火墙本身的 IP 发出的包。为了实现这一点，我们很可能在 ACCEPT 链中加入这样一条规则：如果包是由防火墙的 IP 发出的，就放行，否则，它们就会被 OUTPUT 链的缺省策略 DROP 掉。

设置默认策略

在我们创建规则前，需要设置内建链的默认策略，命令很简单，如下：

```
iptables [-P {chain} {policy}]
```

默认策略在链里面所有规则都没有被匹配的时候引用，例如在一个报文不满足规则集里面的任何策略时候，怎么处理这个报文，这就是默认策略存在的理由。

请注意默认策略的副作用，它不做过滤直接就处理报文，很多时候可能会带来一些奇怪的行为。

在filter表里面创建用户链

现在，你对我们的防火墙应该已经有了一个很清晰的印象，心动了吧。心动不如行动，让我们把它变为现实吧。这一节我们就要小心仔细地创建所有自定义链和链内的规则。

如前所述，我们要建立这几条自定义链：icmp_packets、tcp_packets、udp_packets 和allowed，其中allowed链是由tcp_packets链调用的。所有进入$INET_IFACE的ICMP包都会被重定向到icmp_packets链，TCP包是到tcp_packets链，那UDP包自然就是udp_packets链了，详细的解释都在 INPUT chain里。创建自定义链的命令还记得吗？很简单哦，只要使用选项-N ，再指定链的名字即可（不要忘了，新建的链都是空的），如下：

```
iptables [-N chain]
```

在下面的几节里，我们会详尽地介绍上面创建的每一条链，以使你了解它们包含哪些规则、有什么作用。

The bad_tcp_packets chain

这条链包含的规则检查进入包（incoming packet）的包头是否不正常或有没有其他问题，并进行相应地处理。但事实上，我们使用它只是为了过滤掉一些特殊的包：没有设置 SYN 位但又是 NEW 状态的 TCP 包，还有那些设置了 SYN/ACK 但也被认为是 NEW 状态的 TCP 包。这条链可以用来检查所有可能的不一致的东西，比如上面的包或者 XMAS port-scans 等。我们还可以为 INVALID 状态的包增加一条规则的。

如果你想完全了解无SYN位的NEW状态（NEW not SYN），可以去附录常见问题与解答里看看未设置SYN的NEW状态包一节，它介绍了未设置SYN的NEW状态包通过其他规则的情况。在某些情况下可以允许这种包通过，但 99%的情况是我们不想让它们通过。因此，我们会先记录这种包，然后再扔掉它们。

我们拒绝SYN/ACK包以NEW状态进入的原因也是非常简单的，深入的说明在附录常见问题与解答的NEW状态的SYN/ACK包里。基本上，我们这样做是出于对其他主机的好意，因为我们为他们预防了序列号预测攻击（sequence number prediction）。

---

allowed链

如果包是从$INET_IFACE 进入的，而且是 TCP 包，那它就要经过 tcp_packets 链的检验。如果这个连接就是冲着被允许通过的端口来的，我们还要对它进行一些检查，以确定是否真的要接受它。这些“最后的审判”都是在 allowed 链里进行的。

首先，我们看看这个包是否是 SYN 包，如果是，它很可能是新连接的第一个包，我们当然接受了。如果不是，那就看看包是否来自某个 ESTABLISHED 或 RELATED 状态的连接，是的话，就接受。ESTABLISHED 状态的连接是那种在两个方向上都有流量存在的连接。依据状态机制的观点，这个连接一定处于是 ESTABLISHED 状态的，因为我们现在能看到这个包，说明以前肯定收到了相应的 SYN 包。最后一条规则将 DROP 所有其他的包。当包到达最后这条规则，就几乎意味着所有连接都不会有双向的交流，也就是说，我们不会回应 SYN 包。当试图用非 SYN 包开始新的连接时，包也会走到这条规则。不用 SYN 包建立新连接没有什么实际的用处，当然，端口扫描要排除在外。就我知道的而言，现在没有什么有用的 TCP/ IP 程序会使用 SYN 包以外的包来打开一个 TCP 连接。因此，我们要把这样的包 DROP 掉，我有 99%的把握说它们是端口扫描用的。

处理TCP的链

tcp_packets 链指定了哪些端口可从 Internet 访问。但我们还要对进入的包做更多的检查，因此，每个包都会被发送到上面提到的 allowed 链。

-A tcp_packets 告诉 iptables 要在哪条链里增加规则，规则被放在指定链的末尾。-p TCP 指定要匹配的是 TCP 包，-s 0/0 说明要匹配的源地址是从网络掩码为 0.0.0.0 的地址 0.0.0.0 开始的，换句话说，就是所有的地址。这实际上是默认值，我写出来只是尽可能使你更明白。--dport 21 指定目的端口，也就是说如果包是发往端口 21 的，就会被匹配。如果所有的标准都匹配了，包就要被送往 allowed 链。

TCP 的 21 号端口也是允许访问的，也就是 FTP 的控制端口，它可以控制 FTP 连接，前面提到过，我还允许所有 RELATED 状态的连接通过。这样，我们也就可以使用 PASSIVE（主动）和 ACTIVE（被动）的连接了，当然，要事先装载 ip_conntrack_ftp 模块。如果我们不想再提供 FTP 服务，就卸载 ip_conntrack_ftp 模块，并把$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed 这一行从文件 rc.firewall.txt 里删掉。

22 号端口是 SSH 使用的。如果你允许外网的如何人都能通过 telnet（使用 23 号端口）访问你的机子，那你还是使用 SSH 吧，它的安全性要好很多。注意，你操作的是防火墙，把任何访问权分配给除你自己之外的人都不是什么好主意。防火墙总是应该尽量少地暴露自己。

80 是 HTTP 端口，也就是说你在防火墙上运行了网页服务。如果你不提供网页服务，就删掉这条规则吧。

最后我们还提供了 IDENTD 服务，端口是 113。这个服务对某些协议是必须的，如 IRC。注意，如果你 NAT 一些在内网里的主机的话，软件 oidentd 也值得一用，它会把 IDENTD 请求中继给内网里正确的机子。

如果没有匹配上面任何一条规则，包就会被送回 tcp_packets 链的父链，也就是把它发到 tcp_packets 链的那条规则所在的链。如果你想打开更多的端口，只要对 tcp_packets 链里的任何一行使用“复制、粘贴大法”，再修改一下端口号即可。

处理UDP的链

如果我们在 INPUT 链中遇到了 UDP 包，就把它发送到 udp_packets 链。在那里，我们只处理 UDP 包，所以要用-p UDP 来指定相应的协议。我们接受来自任何地址的包，故有-s 0/0，这其实就是源地址选项的默认值，但为了更明确，我们还是把它写出来了。此外，我们只接受发往特定端口的包，这些端口是我们想对 Internet 开放的。注意，我们不需要依据发送端的源端口来决定是否打开某个端口，这个工作是由状态机制完成的。如果我们想运行某个使用 UDP 端口的服务（如 DNS），只要开放相应的端口，其他的端口不需要打开。那

些处于 ESTABLISHED 状态、正在进入防火墙的包在到达包含--state ESTABLISHED,RELATED 的规则（这是 INPUT 链里那些“处理来自 Internet 的包的规则”中的第一条规则）之后就会被接受了。

我们不接受外来的以 53 号端口为目的的 UDP 包，也就是说，我们不想接受外来的 DNS 查询。其实，规则已经写好了，我只是把它给注释掉了。如果你想把防火墙作为一台允许 Internet 访问的 DNS 服务器，那就把注释符号去掉。

就我个人而言，我会打开 123 号端口，它对应的协议是 network time protocol，简称 NTP 。我们可以利用这个协议与某台具有精确时间的时间服务器联系，以设置本机的时间。你们中的大部分可能用不到此协议，所以我也把它注释掉了，虽然我已经写出了规则。

我打开了 2074 号端口，它是某些实时的多媒体应用程序使用的。比如 speak freely ，你可以用这个程序通过音箱、麦克风或耳麦与其他人进行实时交谈。如果你不需要，就把这条规则注释掉吧。

端口 4000 相应的协议是 ICQ 协议，由 ICQ 使用，世界上使用最广泛的聊天程序之一，“地球人都知道”。Linux 上至少有 2-3 种不同的 ICQ 克隆。我想不必解释为什么要开放这个端口了吧。（译者注：国产的聊天程序，常见的是 QQ（端口 8000），现在又有了 UC（端口 3001）等。）

如果你正在经历日志满天飞的苦恼，这里有两个额外的规则可以使用，当然，这要看是因为什么引起的了。我们这里的第一条规则会阻塞目的端口是 135 到 139 的广播包。大部分 Microsoft 使用者会用到 NetBIOS 或 SMB，而它们就使用这些广播包。这条规则可以阻塞所有位于外网的那些 Microsoft Network 产生的广播包，我们的日志也就可以简洁一些了。第二条规则也是解决日志问题，不过问题的产生者变了，这回是外网的 DHCP 查询。如果你的外网是由非交换的以太网组成的，在那里客户机可以通过 DHCP 得到 IP 地址，也就是说，如果外网里会有很多 DHCP 查询广播包，那就启用这条规则吧。

注意，我把最后这两条规则也注释掉了，因为有些人可能想看看相关的记录。如果你正经历“合法日志过多”的痛苦，就试试丢弃那些包吧。其实，在 *INPUT* 链中，就在这两条有关日志的规则之前，还有更多这种类型的规则。

处理ICMP的链

现在，我们该决定可以接受哪些 ICMP 类型了。在 INPUT 链中，如果 ICMP 包是从 eth0 （即本例的 Internet 接口）进入的，我们就要把它重定向到

icmp_packets 链（前面提到过），以检查是否是可以接受的类型。目前，我只接受三种 ICMP 包：ICMP Echo requests，TTL equals 0 during transit 和 TTL equals 0 during reassembly。默认不接受其他任何 ICMP 类型的原因是，几乎所有其他类型的 ICMP 包都是 RELATED 状态的，也就是说它们都会被处理 RELATED 状态的规则放行。

如果一个ICMP包是用来回应“已经存在的包或流”的，那它就是与那些流相关的，也就是说，它的状态是RELATED。更多的信息在章节*状态机制*里。

现在来解释一下我为什么只接受上面提到的三种 ICMP 包。Echo Request 用来请求 echo reply，这个操作主要被用来 ping 其他的机子，以确定那些机子是否可用。如果没有这一条规则，其他机子将不能通过 ping 来确定我们是否可用。注意，有些人倾向于删掉此规则，只是因为他们不想被 Internet 看到。删掉这个规则将会使任何来自 Internet 的、对我们防火墙的 ping 都无效，因为防火墙对他们完全没有回应。

允许超时（Time Exceeded，如 TTL equals 0 during transit 传输期间生存时间为 0 和 TTL equals 0 during reassembly 在数据报组装期间生存时间为 0）信息进入，我们就可以追踪从本地到某台主机的路径，或者在包的 TTL 为 0 时，我们能得到回应信息。比如，在我们追踪到某台主机的路径时，会以 TTL = 1 的包开始。当它得到第一个路由时，TTL 减为 0，我们也会得到第一个路由返回的超时信息。然后是 TTL = 2 的包，我们就会得到第二个路由器返回的超时信息。如此，直到得到我们的目的主机返回的信息。这样，我们就可以从路径上的每一台主机得到一个回应，从而我们可以看到路径上的每一台主机，也就可以知道路径是断在哪台机子了。

完整的ICMP类型列表在附录ICMP类型里。关于ICMP类型的更多信息与用法，我建议你看看下面的文章：

Ralph Walden的The Internet Control Message Protocol。

J. Postel的RFC 792 - Internet Control Message Protocol。

注意，我阻塞了所有我不想接受的 *ICMP* 包，这对你的网络来说可能会有问题，但在我这里，一切工作正常。

INPUT链

我写的 INPUT 链大部分是使用其他链来完成这个艰难的工作的。这样做，我们就不需要从 iptables 装载太多的规则（译者注：这是针对装载 INPUT 链的规则说的，因为这时其他规则已经装载好了），而且它在较慢的机子上也可以工作得很好，但另一方面，这样的机子在高负载时还是会丢弃很多包（译者注：机子慢，就是不行）。之所以能做到这一点，是因为对于大量不同的包，我们通过检查特定的细节来确定它们的类别，再把这些包发送到相应的自定义链去处理。这样，我们可以分割规则集使之包含很少的规则，每个包要经历的规则也就少了。从而，在过滤包时，防火墙的开销也就小多了。

首先，我们要检查进入的tcp包的形态是否是不正常的或我们不想要的。这个工作是这样完成的，我们把所有的tcp包送到bad_tcp_packets链，由其中的规则进行检查，具体的描述在小节bad_tcp_packets链里。

然后，我们开始处理被信任的网络的数据传输。这包括所有来自“连接内网的网卡”的流量，所有来自和发往 loopback 的流量（要注意，和 loopback 相对应的 IP 地址包括了所有分配给防火墙的地址，其中也包括连接 Internet 的地址）。我们把处理 LAN 的流量的规则放在防火墙的上部，因为我们的局域网产生的流量要远远多于 Internet 连接。这样，规则会更有效率，防火墙就能以较小的开销去匹配包，从而网络阻塞的可能性也就减小了，而且也便于我们查看经过防火墙的包主要是什么类型。

下面的一些规则会处理来自 Internet 的信息，在接触这些规则之前，有一个相关的规则，我们可用它来减少一些开销。这是一个处理状态的规则，它允许所有处于状态 ESTABLISHED 或 RELATED 且发往 Internet 接口的包进入。在 allowed 链中有一个与此类似的规则（译者注：实在是多余，建议大家把它拿掉吧）。顺序上，当然是 INPUT 链里的规则先处理包了。然而，在 allowed 链里保留--state ESTABLISHED,RELATED 规则还是有一些原因的，比如，方便某些人想剪切此功能，粘贴到其他地方。

在 INPUT 链里，我们会把从$INET_IFACE 进入的所有 TCP 包发往 tcp_packets 链，类似地，把 UDP 包发往 udp_packets 链，把 ICMP 包发往 icmp_packets 链。一般说来，防火墙遇到的最多的包是 TCP 包，其次是 UDP 包，最后是 ICMP 包。但要注意，这只是一般情况，对你可能不适用。一样的规则因为顺序不同，或者说逻辑不同，效率会有很大的差别。如果规则集写得不好，即使只有 100 条规则，而且有 100mbit 的网卡，就算是 Pentium III 的机子也会吃不消的。所以你自己写规则集时一定要注意这一点。

这里有一条被注释掉了规则，万一在我们的防火墙外部有一些Microsoft网络，我们可以启用它来解除日志过多的烦恼。Microsoft的客户机有个坏习惯，就是向地址 224.0.0.0/8 发送大量的多播包。因此我们要有这条规则来阻塞那些包，以免我们的日志被它们填满。还记得吗？udp_packets链里也有两条类似的规则。忘了的话，就到处理UDP的链看看吧。

在其他的包被 INPUT 链的策略处理之前，我们会把它们记录下来，以便查找可能的问题或 bug：它可能就是我们不想允许它进入的那种包，也可能是对

我们做了什么坏事的用户，还可能是防火墙的问题，如我们阻塞了应该被放行的包。我们要了解所有的情况，这样问题才能得以解决。我们每分钟最多记录3个包，因为我们可不想让日志里记载的都是废话。为了容易辨别包的来源，我们还对所有的记录设置了前缀。

所有没被上面的规则处理的包都会被策略DROP掉。策略的设置在本章的小节缺省策略的设置里，距离我们已经很远喽。

FORWARD链

在本例中，FORWARD 链包含的规则很少。首先，我们会把所有的包发往 bad_tcp_packets 链。此链我们前面提到过多次，它可以被多条链调用，其实它也就是为这个目的而设计的。

之后就是 FORWARD 链的主要规则了。第一个允许所有来自$LAN_IFACE 的数据通过，没有任何限制，也就是说，我们的 LAN 可自由地访问 Internet。第二个允许 ESTABLISHED 和 RELATED 状态的包能通过防火墙。换句话说，就是所有对我们的内网发出的连接的回应都可以返回局域网。为了使我们的内网能访问 Internet，这些规则是必须的，因为我们在前面已经把 FORWARD 链的策略设为 DROP 了。这样设置规则也是很聪明的，因为它在保证局域网可以访问 Internet 的同时阻止了 Internet 对局域网的访问。

最后我们也有一个处理日志的规则，用来记录没被上面任何规则匹配的包。这样的包很可能是形态不正常的或者是其他问题，比如可能是黑客攻击。这个规则与 INPUT 链中的那个类似，只是前缀不同，这里用的是："IPT FORWARD packet died: "。前缀主要用来分离日志的记录，便于我们查找包的来源和包头的一些信息。

OUTPUT链

除了我几乎没有人把防火墙还当作工作站来使用，但正因为这样，我允许几乎所有从防火墙的 IP（包括 LOCALHOST_IP，$LAN_IP 或$STATIC_IP ）出发的数据，而阻塞其他情况。因为其他任何情况都可能被人以某种方式欺骗。最后的规则还是用来记录那些要被策略 DROP 掉的包。这样，我们就可以了解它们，继而可以对产生的问题（可能是具有威胁性的错误，或者是用来进行欺骗的包）采取行动。

PREROUTING链

顾名思义，PREROUTING链（nat表的）是在路由之前做网络地址转换工作的。然后，包再经过路由，就会被送到filter表的INPUT或FORWARD链。我们在这里讨论这个链的唯一原因是，我们觉得有责任再次指出你不应该在此链中做任何过滤。PREROUTING链只会匹配流的第一个包，也就是说，这个流的所有其他的包都不会被此链检查。事实上，在这个脚本中，我们根本没有用到 PREROUTING 链。如果你想对一些包做DNAT操作，例如，你把web server放在了局域网内，这里就是你放置规则的地方。有关PREROUTING链的详细信息在章节表和链中。

千万注意，*PREROUTING*链只能做网络地址转换，不能被用来做任何过滤，因为每个流只有第一个包才会经过此链。

POSTROUTING链

我们最后的任务应该是构造网络地址转换，对吧？至少对我来说是的。我们在 nat 表的 POSTROUTING 里只加入了一条规则，它会对所有从 Internet 接口（对我来说，这是 eth0）发出的包进行 NAT 操作。在所有的例子脚本里，都有一些变量，它们要给以正确的配置。选项-t 指定要在那个表里插入规则，这里是 nat 表。命令-A 说明我们要把规则添加到 POSTROUTING 链末尾。-o $INET_IFACE 指定要匹配所有从接口 INET_IFACE 出去的包，这里我们使用的是 eth0。最后，我们把 target 设置为 SNAT。这样，所有匹配此规则的包都会由 SNAT target 处理，之后，它们的源地址就是 Internet 接口的地址了。不要忘了 SNAT 可是一定要有 IP 地址的，用--to-source 来设置哦。

在这个脚本中，我们选择SNAT而不用MASQUERADE是有原因的。主要的原因是我们的防火墙有静态IP地址，使用SNAT会更快更有效。还有一个原因是我们要在这个例子中展示它的作用以及怎样使用它。如果你没有静态的IP地址，要想实现SNAT，还是使用MASQUERADE为好，因为它简单易用，而且它可以自动获得IP地址。当然，计算机的消耗会多一点，但如果你使用DHCP，这样做是很值得的。如果你想了解MASQUERADE target的表现，应该看看脚本 rc.DHCP.firewall.txt。

下章预览

本章介绍了不同脚本的作用，特别是 rc.firewall.txt。每个人都有自己的编码风格，文中的例子只是我自己的爱好罢了。

下一章介绍其他一些不同的例子，他们告诉你在不同的场景下使用哪个例子。这些例子也可以从本文档的服务器上面下载。

十四章 例子脚本

本章介绍本指南里面的每一个例子，并且全面浏览一下这些基本的结构和他们提供的服务。这些脚本可能不是很完美，而且也可能不适合你的网络。换句话讲，这些例子只是模板，需要你自己按照实际需求进行修改。

---

rc.firewall.txt 脚本结果

本指南所有的脚本都是依据一个特定的结构来写的。理由嘛，就是这样可以使它们彼此相似，便于我们查找不同之处。本章将要对这个结构做一个很好的说明，而且还会简单地阐述这些脚本为什么会按照现在这种样子来写，以及我为什么选择一直使用这种结构。

注意，即使我选择了这种结构，你也不一定非要用它，对你来说，它可能并不是最好的。我选择它只是因为它易读，而且能很好地符合我的逻辑。

脚本结构

这就是本指南所有脚本遵循的脚本结构。如果有不同于此的地方，可能就是我出错了，除非我特意说明为什么要打破这种结构。

1. *Configuration* -首先是一个配置选项区，里面的变量在脚本中会用到。几乎任何脚本（shell-script）的第一部分都是配置选项区。
    1. *Internet* -有关 Internet 连接的配置。如果我们没有任何 Internet 连接，这一部分就可以跳过去。注意，相比我们列出来的，这一部分可能会包含更多小节，虽然我们这里只有了了几个，但足够应对我们已有的各种 Internet 连接了。
        1. *DHCP* -如果脚本用到了 DHCP，我们就要在此添加相应的配置。
        2. *PPPoE* -如果想把脚本用于 PPPoE 连接，就要在此添加相应的配置。
    2. *LAN* -如果防火墙后有局域网，就要使用这里的配置了。大部分情况下都会用到这儿，因为局域网几乎总是存在的。

3. *DMZ* -对非军事区（DMZ zone）的配置。大部分脚本不会用到这个设置，因为这些脚本针对的主要是一些普通的家庭网络，或小企业的网络。
4. *Localhost* -本地（local-host）的有关设置。虽然我把它们写成变量的形式，但一般不会被改变，也不应该有什么理由要改变它们。
5. *iptables* -有关 iptables 的设置。大部分情况下，这里只设置一个变量，用来指向 iptables 程序的位置。
6. *Other* -如果还有什么信息，首先应该把它们放在相应的小节里，实在没有相应的小节，就放这儿吧。

2. *Module loading* -脚本应该维护一个模块列表。它分为两部分，第一部分包含必需的模块，同时第二部分要包含不必要的模块的列表。

注意，这些模块可能会提高安全性，或为管理者、客户添加某些服务，还有一些模块不是必需的，但它们可能也被加入了列表。不过，在本例中，我已经注意了这个问题。

最新的 iptables 里面，模块已经是自动加载而且大部分模块不需要手动再次加载。但是从控制的角度看问题，最后你自己加载，例如 conntrack helper 模块就是手动加载的。

   1. *Required modules* -这里装载的是必要的模块，它们可能会提高安全性或为管理者、客户增加某些服务。
   2. *Non-required modules* -这里列出的是不必要的模块，所以它们都被注释掉了。如果你用到了它们提供的功能，就可以启用它们。

3. *proc configuration* -这儿关心的是有关 proc 系统的设置。如果一些选项是必须的，我们就启用它，如果不是，就把它注释掉。大部分有用的 proc 配置都列在这儿了，但远远不是全部的。
   1. *Required proc configuration* -包含了使脚本能正常工作的所有必需的 proc 配置，它也可以包含一些能提高安全性或为管理者、客户增加特定服务的选项。
   2. *Non-required proc configuration* -这里提到的选项不是必需的，虽然它们可能很有用。因此，我把它们都注释掉了。当然，这里并没有包括所有这样的选项。

4. *Rules set up* -现在，应该添加规则了。我把所有的规则都明确地分配到了与表、链相应的小节里。所有自定义的规则都写在系统内建的链之前（译者注：当然要写在前面了，因为后面要调用它们哦）。另外，我是按照命令 iptables -L 输出的顺序来安排此脚本里表与链的出现顺序的（译者注：这样，便于我们查看哦）。
   1. *Filter table* -首先是 filter 表，而且我们先要设置策略。

1. *Set policies* -为所有系统内建的链设置策略。通常，我会设置 DROP，对于允许使用的服务或流会在后面明确地给以 ACCEPT。这样，我们就可以方便地排除所有我们不想让人们使用的端口。
2. *Create user specified chains* -在这里创建所有以后会用到的自定义链。如果没有事先建立好，后面是不能使用它们的，所以我们要尽早地建立这些链。
3. *Create content in user specified chains* -建立自定义链里使用的规则。其实你也可以在后面的某个地方写这些规则，之所以写在这儿，唯一的原因是这样做规则和链会离得近些，便于我们查看。
4. *INPUT chain* -创建 INPUT 链的规则。

从这里开始，我就是遵循 iptables -L 的输出格式来创建规则的，这样做的唯一原因就是为了便于阅读，避免混淆。

5. *FORWARD chain* -为 FORWARD 链创建规则。
6. *OUTPUT chain* -为 OUTPUT 链创建规则。其实，在这里要建的规则很少。

2. *nat table* -在处理完 filter 表之后，该设置 nat 表了。我们这样做是有一定原因的。首先，我们不想太早地打开转发机制（译者注：注意，filter 表设定的是过滤机制，而不是转发）和 NAT 功能，因为它们可能导致数据会在错误的时间（也就是这样的时刻：我们打开了 NAT，但过滤规则还没有运行）通过防火墙。还有，我把 nat 表看作是围绕 filter 表的一个层。也就是说，filter 表是核心， nat 表是它外部的一个层，mangle 表是第二层。从某些观点来看，这可能有点不对，但也八九不离十了。
    1. *Set policies* -与 filter 一样，我们先来设置策略。一般说来，缺省的策略，即 ACCEPT，就很好。这个表不应该被用来做任何过滤，而且我们也不应该在这儿丢弃任何包，因为对我们假设的网络情况来说，可能会发生一些难以应付的事情。我把策略设为 ACCEPT，因为没有什么原因不这样做。
    2. *Create user specified chains* -在这儿创建 nat 表会用到的自定义链。一般情况下，我没有任何规则要在这儿建立，但我还是保留了这个小节，以防万一罢了。注意，在被系统内建链调用之前一定要建好相应的自定义链。
    3. *Create content in user specified chains* -建立自定义链的规则。
    4. *PREROUTING chain* -要对包做 DNAT 操作的话，就要用到此链了。大部分脚本不会用到这条链，或者是把里面的规则

注释掉了，因为我们不想在不了解它的情况下就在防火墙上撕开一个大口子，这会对我们的局域网造成威胁。当然，也有一些脚本默认使用了这条链，因为那些脚本的目的就是提供这样的服务。

5. *POSTROUTING chain* -如果使用 SNAT 操作，就要在此建立规则。你可能有一个或多个局域网需要防火墙的保护，而我就是依据这样的情况来写此脚本的，所以这个脚本中使用的 POSTROUTING 链是相当实用的。大部分情况下，我们会使用 SNAT target，但有些情况，如 PPPoE，我们不得不使用 MASQUERADE target。
6. *OUTPUT chain* -不管什么脚本都几乎不会用到这个链。例如可以限定只有本机接口地址的报文才能发出等等。

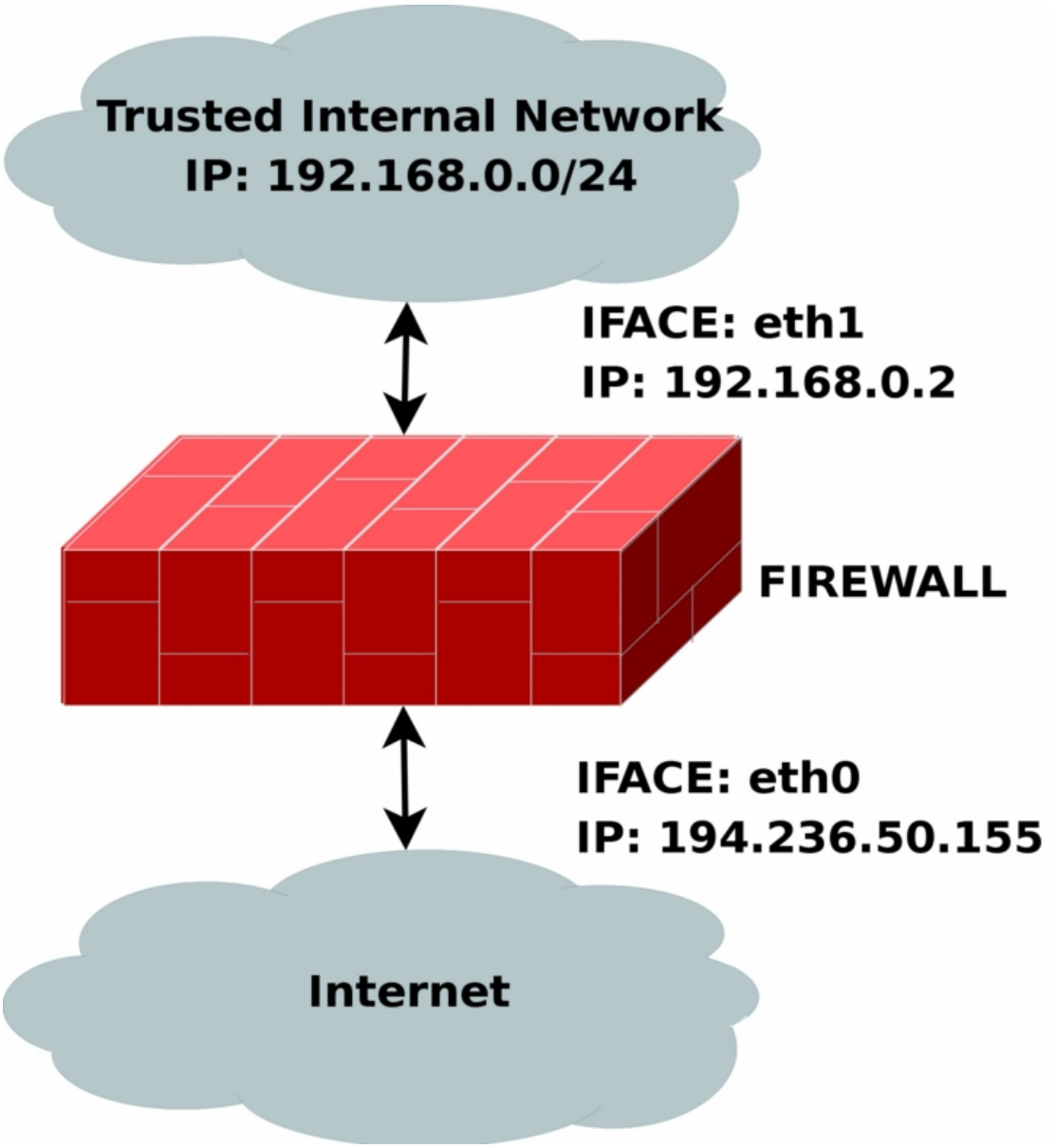
3. *mangle table* -最后要做的就是处理 mangle 表了。通常，我不会使用这个表，因为一般情况下，它不会被任何人要到，除非他们有什么特殊的需要，比如为了隐藏一条连接后的多台机子，我们要统一设置 TTL 或 TOS 等。在这个脚本里，此表是空白的。但在此指南中还是有个小小的例子说明了 mangle 表的用处。
    1. *Set policies* -设置策略。这里的情形和 nat 表几乎完全相同。这里不应该做过滤，也不应该丢弃任何包。在任何脚本里我都不会把 mangle 表的策略设为其他的值，也不鼓励你这样做。
    2. *Create user specified chains* -建立自定义链。我几乎不会用到这个链，所以没有建立任何规则。保留此小节，只是已备后用。
    3. *Create content in user specified chains* -建立自定义链的规则。
    4. *PREROUTING* -本指南的所有脚本都未在此链建立规则。
    5. *INPUT chain* -本指南的所有脚本都未在此链建立规则。
    6. *FORWARD chain* -本指南的所有脚本都未在此链建立规则。
    7. *OUTPUT chain* -本指南的所有脚本都未在此链建立规则。
    8. *POSTROUTING chain* -本指南的所有脚本都未在此链建立规则。

这应该可以较详细地解释每个脚本的结构是怎样的以及它们为什么要使用这个结构了。

注意，上面的描述其实还是非常简单的，应该被看作一个摘要，它简要地解释了脚本为什么要按照这种松散的结构来写。千万注意，我可没有说过这种结构是唯一的、最好的。

---

rc.firewall.txt



脚本rc.firewall.txt是核心，防火墙配置实例 rc.firewall对它已经做了很详细的解释，其他的脚本都是以它为基础得到的。这个脚本主要是针对具有两个连接的家庭网络而设计的，如一个局域网连接，一个Internet连接。我们假设的情况是你有一个静态IP地址，不需要*DHCP*，*PPP*， *SLIP*或其他什么协议为你动态分配IP。如果你想要的恰恰是使用这些协议的脚本，就到rc.DHCP.firewall.txt看看吧。

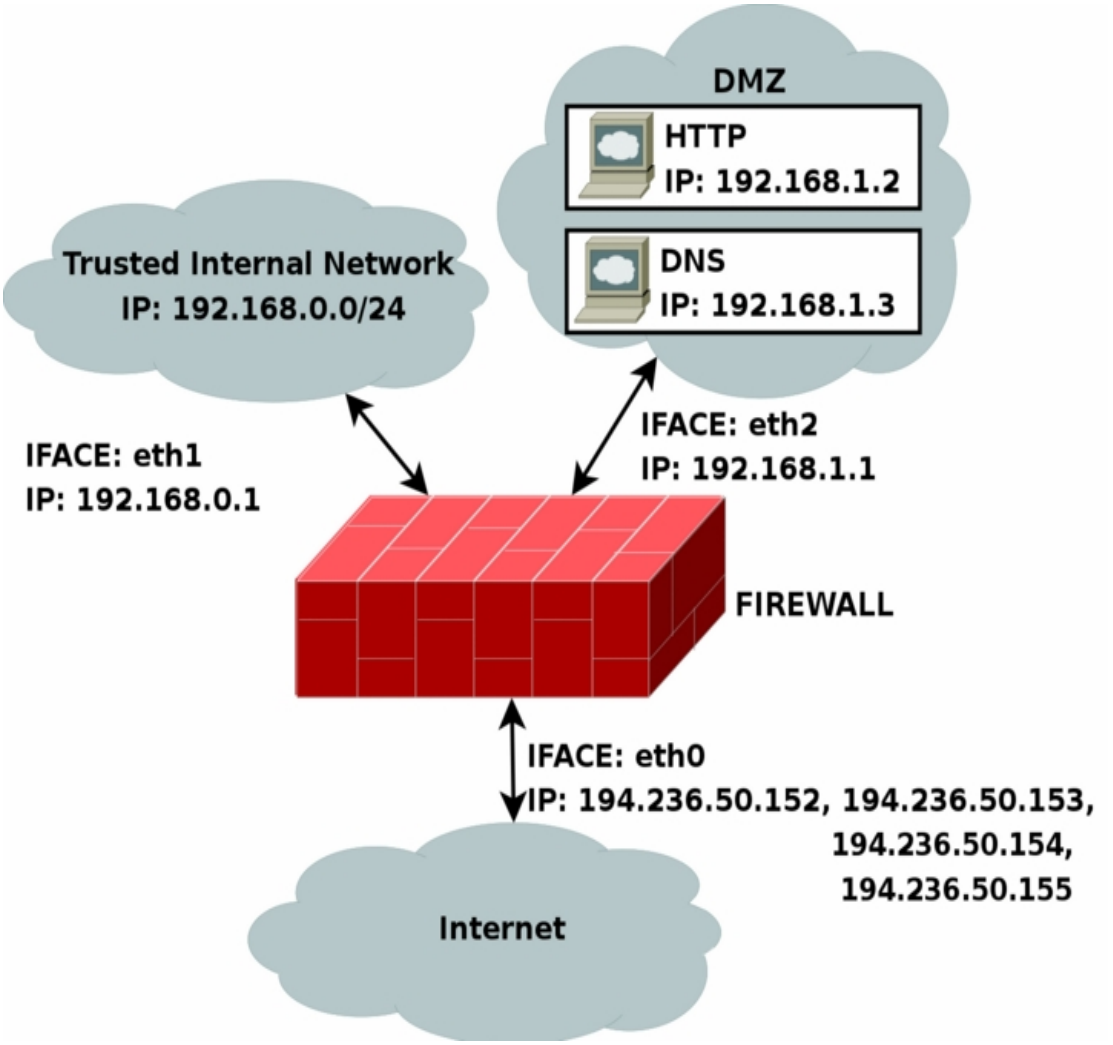
脚本 rc.firewall.txt 要完全发挥作用，系统必需要有下面列出的功能，你可以把它们编译进内核，也可以编译成模块。如果你改变了脚本，就要加入相应的功能模块或把它们编进内核。

- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_MATCH_LIMIT

- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_LOG

---

rc.DMZ.firewall.txt



脚本rc.DMZ.firewall.txt所针对的情况是这样的：有一个可信任的内网，一个DMZ，还有一个Internet连接。这里的DMZ是通过设置一对一的*NAT* 操作得到的，它需要IP别名（就是在一块网卡上设置多个IP地址）的支持。我们还有其他的方法来实现 DMZ：如果你有一个整个的网段，可划分子网，然后把某个子网分给DMZ，再为防火墙配置相应的内网与外网 IP地址（译者注：第一种方法是针对有多个网段的情况，即内网一个网段，DMZ一个网段，第二种方法是把一个网段划分成几个子网，这样就和第一种情况一样了）。注意，这种方法会多消耗两个IP，一个是网络地址，一个是广播地址（译者注：具体细节请上网搜索子网划分的相关信息，这个指南并不包含此类信息）。以上两种方法用哪

一个就要你自己决定了。本指南会给你实现防火墙与NAT的手段或叫做技术，但具体如何去做，没有完全的说明，因为这已经超出本文的范围了。

这个脚本需要以下模块，也可能它们已被编译进内核了。

- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_LOG

从图中可以看出，此脚本假设你有两个内网，一个是可信任的内网，使用地址192.168.0.0/24，另一个是 DMZ（我们正是对它做一对一的NAT），使用地址192.168.1.0/24。如果有人从 Internet 向我们的DNS_IP 发送一个包，我们就要对它使用 *DNAT*，之后，此包的目的地址就指向DMZ 里的DNS 服务器了，它也就可以到达真正的DNS 服务器。否则，DNS 服务器不会看到这个包，也就没有应答之说了。下面是实现上述 *DNAT* 功能的语句：

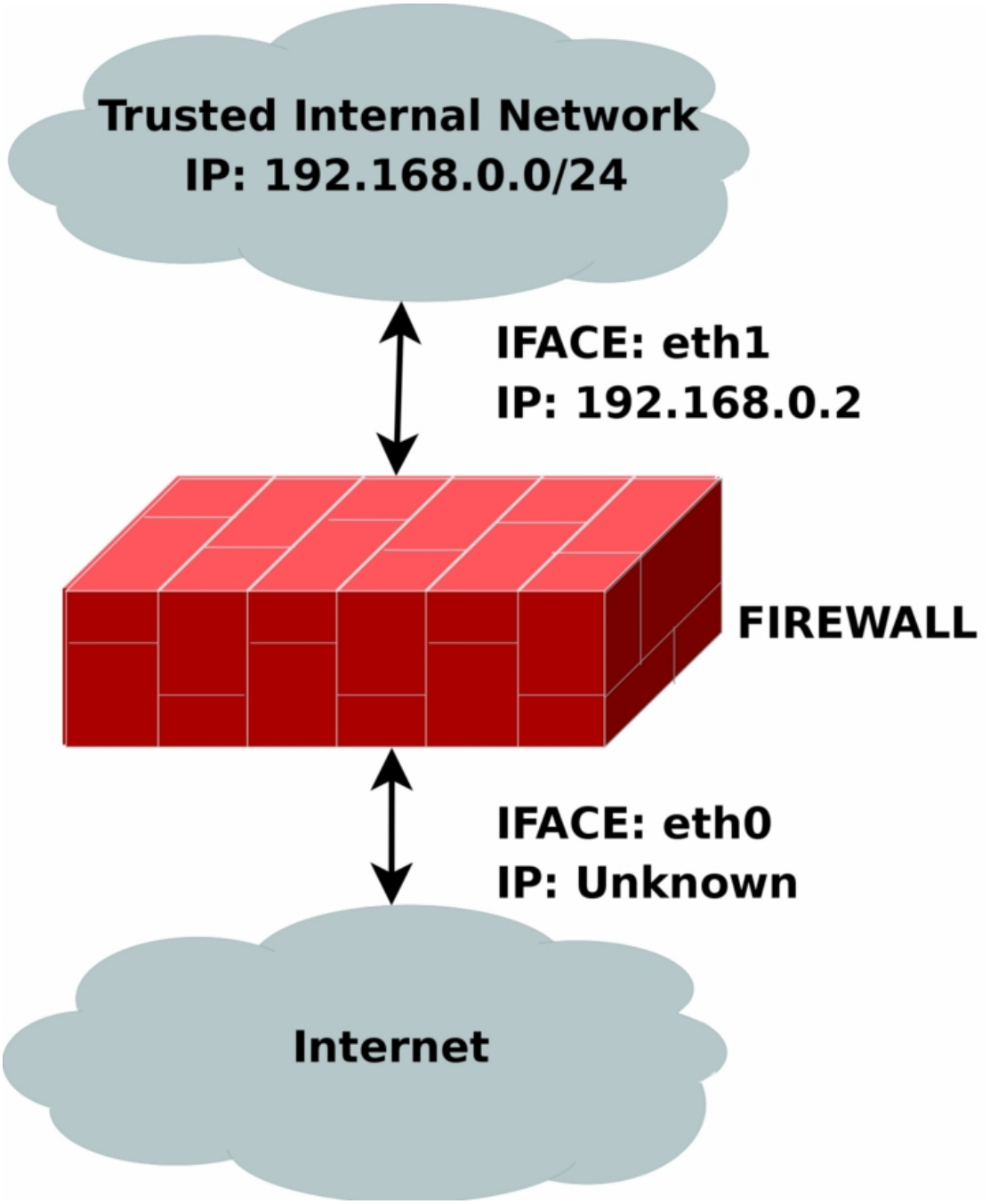
```
$IPTABLES -t nat -A PREROUTING -p TCP -i $INET_IFACE -d
$DNS_IP \
--dport 53 -j DNAT --to-destination $DMZ_DNS_IP
```

我们可以看出，这个规则要放在 *nat* 表的 *PREROUTING* 链中，包要满足的条件是：使用 *TCP* 协议且使用53 号端口，从接口$INET_IFACE 进入，而且要以$DNS_IP 为目的。被匹配的包要交给 *DNAT* target 来处理，它会把包的目的地址改为由--to-destination 指定的地址$DMZ_DNS_IP。这就是 *DNAT* 的工作流程。当相应的应答包被发送到防火墙时，会自动地被un-DNAT。

现在，你应该完全可以读懂这个脚本了。如果有什么你不明白的东西在脚本的其他部分没有被用到，那可能就是我的错误了，要告诉我哦。

---

rc.DHCP.firewall.txt



脚本rc.DHCP.firewall.txt适用于那些使用*DHCP*、*PPP*或*SLIP*连接Internet的情况，它和原始的脚本rc.firewall.txt几乎一样，主要的区别在于这里不再使用变量STATIC_IP。原因很简单了，就是它不能和动态的IP一起使用。此脚本相对于原始脚本的改变是很少的，但还是有一些人发信问我做了什么改变。经过大家的考验，这个脚本应该是一个很好的解决方案了。

它需要如下功能模块。

- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_IPTABLES

- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_MASQUERADE
- CONFIG_IP_NF_TARGET_LOG

我做的改变主要是删除了变量 STATIC_IP 以及和它相关的所有东西。以前，主要的过滤工作是基于变量 STATIC_IP 的，现在是 INET_IFACE 了。也就是说，在这个脚本里，我们不再把-d $STATIC_IP 作为过滤的条件，而是用 -i $INET_IFACE。这几乎是唯一的改变，也是必需的改变。

可还是有一些问题要考虑。现在，我们不能再在 INPUT 链依据某些条件，比如--in-interface $LAN_IFACE --dst $INET_IP 来进行过滤（译者注：因为这时已无固定的 INET_IP）。这强迫我们只能基于 Internet 接口进行包的过滤，在这种情况下，内网必须访问那个可变的 Internet 的 IP。这会出现一些问题，有个例子可以说明这一点，就是我们在防火墙上运行 HTTP 服务。如果我们访问这个网站（其中，主页包含了一个指向 HTTP 服务器的静态连接，这可能是某个动态的 DNS 解决方案），问题就暴露了。经过 NAT 操作的机子会向 DNS 查询 HTTP 服务器的 IP，然后再试着访问这个 IP。万一我们是基于接口和 IP 做的过滤，这台机子就不能访问到 HTTP 了，因为 INPUT 链会 DROP 掉这个包（译者注：还是因为 Internet 接口的 IP 不固定）。在某种情况下，这也会发生在你有静态 IP 的时候，但在那种情况下，我们可以增加一条规则，以检查 LAN 接口的包是否是发往 INET_IP 的，若是，则 ACCEPT。

如果你看过以前的内容，得到或写一个可以获取动态IP的脚本可能会是个解决问题的好办法。比如，我们可以写一个脚本，它紧随着Internet连接的启动而运行，而且它能从命令ifconfig的输出中提取IP，再把这个IP赋给某个变量。较好的办法是使用一些程序自带的脚本，如 pppd带的脚本ip-up。也有一些网站，如linuxguruz.org，提供了很多有用的脚本，你可以在附录其他资源和链接找到它的链接

这个脚本比 rc.firewall.txt 的安全性要差一点。我明确地建议你尽可能使用后者，因为前者的开放性大了点，所以外部攻击的威胁就大了。

还有一种方法可获得 IP，就是在脚本里加上类似这样的语句：

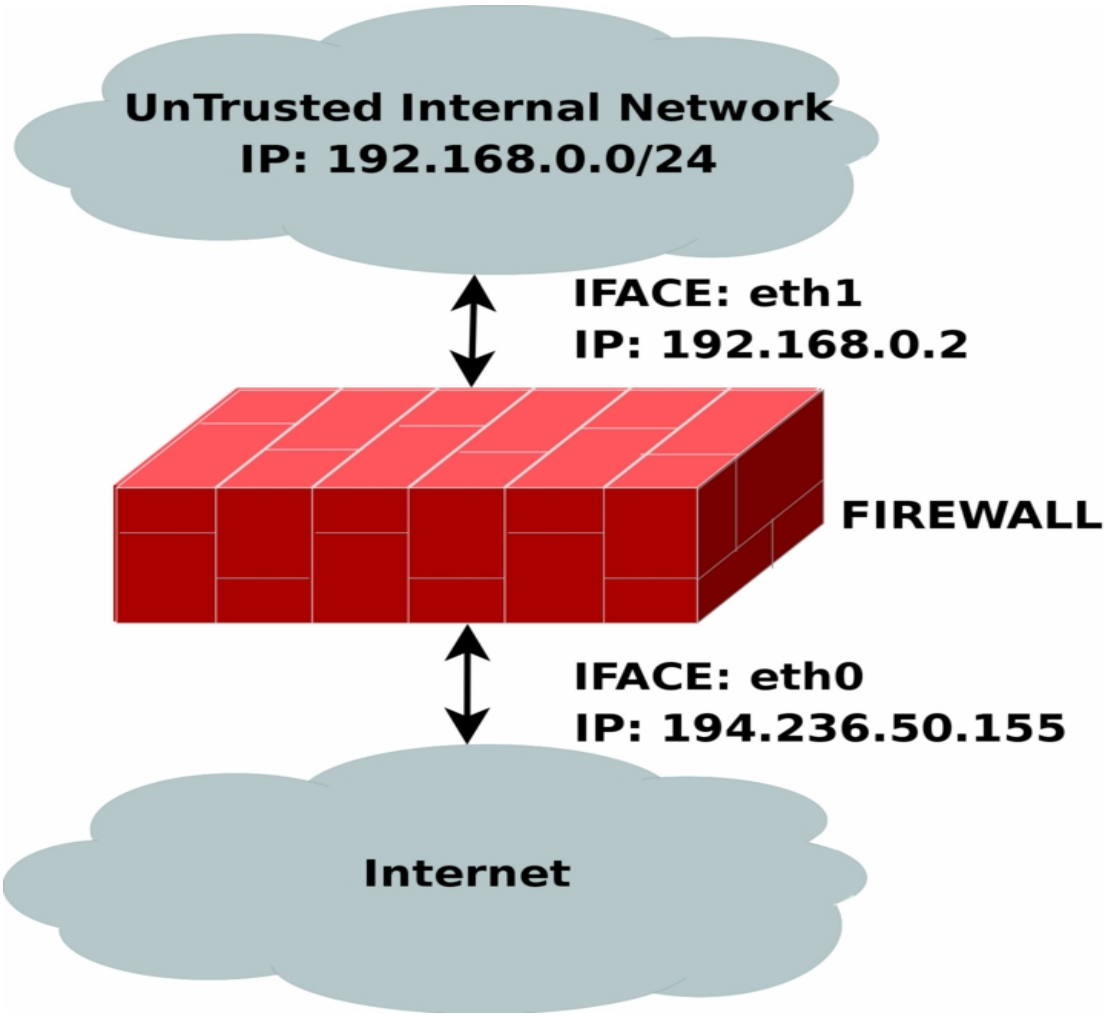
```
INET_IP=`ifconfig $INET_IFACE | grep inet | cut -d : -f 2 |
\
cut -d ' ' -f 1`
```

上面这句话的作用是从ifconfig的输出里提取接口$INET_IFACE的IP，再赋给$INET_IP。更好的办法是使用脚本 retreiveip.txt。但要注意，这个方法可能会引起一些不正常的情况，比如使防火墙和内网之间已有的连接停止。下面就说明一下最常见的问题。

1. 如果这个脚本的代码是在另一个脚本内运行的，而那个脚本又是由*PPP daemon*启动的，就会因为NEW not SYN rules（具体信息查看未设置SYN的NEW状态包）的原因而挂起所以当前活动的连接。如果你删掉那个规则，可能会没有事，但还是不保险。
2. 如果你不想改动已有的规则，而又要添加或删除规则，还要不损害已有的规则，这就没法做了。比如，你又想阻塞所有局域网里的机子访问防火墙，又想让它们能控制防火墙上的 *PPP daemon*，如果不删除那个用来阻塞的规则，怎么能完成这样的事？
3. 事情可能也不必这么复杂，就像上面说的，这会导致一些安全问题。但如果这个脚本能保持简单，维持规则的顺序与发现问题都是很容易的。

---

rc.UTIN.firewall.txt



脚本rc.UTIN.firewall.txt适用于这样的情况：我们不信任任何与防火墙连接的网络，包括内网。我们只允许内网使用*POP3*、*HTTP* 和*FTP*。至于从Internet来的连接，权限和其他脚本一样。

此脚本需要以下功能模块

- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_LOG

这个脚本遵循的原则是不要相信任何人，包括我们自己的员工。这是个令人悲痛的现实，大部分破坏和攻击确实是来自我们内部的。这个脚本只是在加强防火墙方面给了你一个例子。它和 rc.firewall.txt并没有太多的不同，只是少了一些允许通行的规则。

---

rc.test-iptables.txt

这个脚本用来测试 iptables 里所有的链。当然，这要根据你的配置情况做些操作，如打开 ip_forwarding 或是设置 masquerading，等等。只要你的安装了基本的 iptables，就可以使用它。其实这个脚本只使用了 LOG，以便能记录所有的 ping 请求与应答。通过这种方式，我们就可以了解哪些链被穿越了以及被穿越的顺序。使用方法如下，先运行这个脚本，再发布一个 ping 命令，如：

```
ping -c 1 host.on.the.internet
```

然后用命令 `tail -n 0 -f /var/log/messages` 就可看到用了哪些链以及是什么顺序，除非记录因某些原因被替换了。

此脚本仅仅是为测试而写的。也就是说，不要使用类似这样的规则，它记录某一类包的所有信息，这会很快地占满你的日志分区，而且它会成为一个有效的 DoS 攻击。它还可能导致在最初的 DoS 攻击之后，无法记录真正的攻击信息。

---

rc.flush-iptables.txt

rc.flush-iptables.txt不应该被称作脚本，它只是重置并清空所有的表、链。它先把filter表的INPUT、OUTPUT和 FORWARD链的策略设为缺省的ACCEPT，然后是nat表的PREROUTING 、POSTROUTING和OUTPUT链。我们这样做就不必为被关闭的连接和没有通过的包而操心。这个脚本就是为防火墙设置和除错用的，因此我们只关心打开所有的东西并恢复它们的缺省值就行了。

之后，我们清空 filter 表里所有的链，紧接着是 NAT 表的。这样，就不会有什么不应该存在的规则了。这个做完后就该删除 filter 表和 NAT 表里的自定义链了。这时，脚本的工作就应该完成了。当然，如果你用到了 mangle 表，可以在这个脚本里添加相应的清空规则。（译者注：其实作者已经这样做了）

最后再说明一下，有些人写信建议我把这个脚本放到 rc.firewall 脚本里面，而且是用 Red Hat Linux 脚本的语法，这样当 rc.firewall 启动时，这个脚本也可以启动。但我不会这样做的，因为这是一个指南，主要是用来学习 iptables 的使用方法的，不应该有过多的 shell 脚本特有的语法。加入 shell 脚本特有的语法会使阅读的难度大大增加，这就远离了我的初衷。这个指南是按照易读的标准来写的，以后我会继续这样做。

Limit-match.txt

这个脚本是用来测试limit match的，也会让你明白limit match是如何工作的。装入这个脚本，再用不同的时间间隔发送ping数据包，可以看出哪个包可以通过，这些包又是以什么频率通过的。你应该可以看出，在limit的burst值再次到达之前，所有的*echo replies*都会被阻塞。

Pid-owner.txt

这个脚本说明了如何使用PID owner match。它其实什么都没做，但你可以运行一下，命令**iptables -L -v**的输出会说明它确实匹配了些东西。

Recent-match.txt

说明Recent-match如何使用的一个例子，完全了解这个例子请看iptables匹配章节的Recent match。

Sid-owner.txt

说明SID owner match如何使用的一个例子。同样，它也是什么都没做，但你可以运行一下，命令**iptables -L -v**的输出会说明它确实匹配了些东西。

Ttl-inc.txt

一个小小的例子，说明了如何隐藏我们的防火墙或路由器，以使跟踪路由程序看不到，这样就可以对可能的攻击者隐藏很多信息。

Iptables-save ruleset

这只是一个输出的例子，它在规则的保存与恢复里被用来说明 iptables-save 命令是如何使用的。所以，它没有任何用处，只是一个参考而已。

下章预览

本章简单介绍了本指南里面的几个脚本例子，希望通过这些例子能够让你对于 iptables/netfilter 有一些深入的认识。

下一节主要讨论 iptables/netfilter 的不同图形化用户界面。当然上面所列不是所有的图形化工具，这些工具都是想简化你的创建工作。但是假如你需要定制个人的复杂化需求，你多少还是得做一些修改。

第十五章 iptables/netfilter图形化界面

下面我们列出了常用的工具，但是具体介绍我就不翻译了，有需要的自行比较你真正需要的是哪一个，然后到相关的网站上下载详细资料。

Fwbuilder http://www.fwbuilder.org/.

Turtle Firewall Project http://www.turtlefirewall.com/

IPMenu http://users.pandora.be/stes/ipmenu.html

Easy Firewall Generator http://easyfwgen.morizot.net/

Integrated Secure Communications System http://iscs.sourceforge.net/

下章预览

本章我们介绍了常见的防火墙图形界面，其中大部分是完全开源的，只有很小一部分需要花钱购买完全功能版本。下一章我们介绍基于 iptables/netfilter 的商业化产品。

第十六章 商业化产品

本节描述目前基于 iptables/netfilter 比较成熟的商业化产品，它们都是经过充分测试而且得到用户充分认可的。 对于这些产品的任何评论或者采购意图，请于厂商直接联系。

HPOS Firewal express series

简单来说奥卡科技的 express 系列产品（www.orcal.cn）主要就是针对 Linux 用户推出的高性能多业务防火墙。首先从性能上讲，它是目前业界最高转发性能的 Linux 防火墙，另外从价格上来看，有大量的免费版本以及部分高端收费定制版本。

奥卡科技的 express 系列防火墙有两种授权模式。一种是软件 license，奥卡科技提供软件版本，个人或者企业用户自己准备特定硬件平台，根据使用性质决定 license 的范围。另外一种是硬件设备，直接购买奥卡科技现成的防火墙，这些平台和软件都是经过充分测试的，能够保证更高的兼容性的发挥硬件性能。

奥卡科技对防火墙所作的修改对于用户是透明的，一个全新的高效协议栈内核被构建在 HPOS express 系列防火墙里面，但是所有的系统接口和应用层接口都和现在保持不变，保证了足够的软件兼容性。

如果想进一步了解该系列产品或者有试用需求，请访问 www.orcal.cn.

What's next?

下章预览

本章介绍了成熟的基于 iptables/netfilter 的防火墙，同时提供了试用的联系方式。

好了，这些就是本指南的所有正文部分了，我希望看完之后你有立刻想使用的冲动。另外文中的任何问题或者 bug 都请直接和我联系。

附录A 常用命令详解

查看当前规则集的命令

查看当前正在使用的规则集是一个十分常用的操作，使用iptables的什么命令还记得吗？我们可是在构建防火墙规则这一章里介绍过啊，虽然到时说得简单了点。再复习一下吧，命令语法如下：

```
iptables -L
```

这个命令会尽可能地以易读的形式显示当前正在使用的规则集。比如，它会尽量用文件/etc/services 里相应的名字表示端口号，用相应的 DNS 记录表示 IP 地址。但后者可能会导致一些问题，例如，它想尽力把 LAN 的 IP 地址（如 192.168.1.1）解析成相应的名字。但 192.168.0.0/16 这个网段是私有的，也就是说，它只能用在局域网里，而不能在 Internet 里使用，所以它不会被 Internet 上的 DNS 服务器解析。因此，当解析这个地址时，命令就好像停在那儿了。为了避免这种情况的发生，我们就要使用选项：

```
iptables -L -n
```

如果你想看看每个策略或每条规则、每条链的简单流量统计，可以在上面的命令后再加一个 verbose 标志，如下：

```
iptables -L -n -v
```

不要忘了，iptables -L 命令还可以查看 nat 表和 mangle 表的内容哦（更不要忘了，默认的表是 filter），只需要使用-t 选项,比如我们只想看 nat 表的规则，就用下面的命令：

```
iptables -L -t nat
```

在/proc 里，可能还有一些文件你会感兴趣。比如，你可以在连接跟踪记录表里看到当前有哪些连接。这个表包含了当前的所有连接，你还可以通过它了解到每个连接处于什么状态。要注意，这个表是不能编辑的，即使可以，也不应该更改它。可以用下面的命令查看这个表：

```
cat /proc/net/ip_conntrack | less
```

此命令会显示当前所有被跟踪的连接，但要读懂那些记录可是有些难度哦。

修正和清空iptables的命令

即使你把 iptables 弄的一塌糊涂，我们也有非常有效的命令来处理，而不必重新启动计算机。我接到过很多关于这个问题的询问，所以我想最好在这儿回答一下。如果你增加的规则有问题，要想删掉它，只要把命令中的-A 改为-D 即可。这样，iptables 就会找到那个错误的规则并删掉它，但如果在你的规则里有好几条同样的规则，它只能删掉找到的第一条。如果你不想这样的事情发生，那就试试用序号来删除。如，你想删除 INPUT 链的第 10 条规则，可以使用 iptables -D INPUT 10。

还有一种情况，就是要清空整个链，这就要使用选项-F。比如，我们要清空整个 INPUT 链，使用的命令就是 iptables -F INPUT。但是要注意，选项-F 并不改变链的缺省策略。所以，如果被我们清空的那条 INPUT 链的策略是 DROP，它还是会阻塞所有的包。那怎么才能重置策略呢？还记得策略 DROP 是如何设置的吧，还是用那个方法啊。比如，我们把 INPUT 链的策略改为 ACCEPT，就用 iptables -P INPUT ACCEPT。

我已经写了一个用来清空并重置iptables的脚本，叫做rc.flush-iptables.txt（附录里有它的代码），在你写自己的防火墙脚本时，很可能会用到。但如果你在mangle表里乱试乱改而导致问题的话，这个脚本就帮不上忙了。因为在脚本rc.firewall.txt里，我没有用到mangle表，所以在 rc.flush-iptables.txt里也就没有添加相应的恢复功能。

Appendix B. 常见问题与解答

模块装载问题

装载模块时，你可能会遇到几个问题，比如，有错误提示说明没有你指定名字的那种模块：

```
insmod: iptable_filter: no module by that name found
```

这个提示是无关紧要的，因为那些模块很有可能已经被静态地编译进内核了。当你遇到这个信息时，这是你应该首先想到的。至于是否真的如我们所想，最简单的测试方就是敲一个用到那个模块功能的命令试试。对于上面的情况，可能是 filter 表没有装入，从而就没有相应的功能，当然不能使用 filter 表了。为了检查 filter 表是否装入，可以用下面的命令来试试：

```
iptables -t filter -L
```

这个命令会输出 filter 表里所有的链，或者是运行失败，给出错误提示信息。如果一切正常，输出结果类似下面的情况，当然，这还要看你是否已经在 filter 表里加入了规则（译者注：在这个例子里，表是空的）。

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

如果你确实没有装载 filter 表，得到的就是如下信息：

```
iptables v1.2.5: can't initialize iptables table `filter': Table \
     does not exist (do you need to insmod?)
Perhaps iptables or your kernel needs to be upgraded.
```

这个问题就有些严重了，从此提示中我们能得到两个信息：第一，我们确实没有把相应的功能编译进内核里；第二，在模块一般应在的目录中没有找到这个模块。这意味着问题是，你或者忘记了装载想用的模块，或者没有用 depmod -a命令更新模块数据库，或者没有把相应的功能编译进内核（不论是静态的还是作为模块）。当然还可能是其他原因，但这些是主要的，不管怎样，大部分原因是很容易解决的。比如，第一个问题可以简单地通过在内核源码目录里运行make modules_install 命令来解决，这当然是有前提的，就是源码已

经编译（compile）而且模块已经构建（build）。第二个问题的解决办法也很简单，只要运行一下depmod -a命令，之后再看看能否正常工作即可。第三个问题有点超出我们的范围了，而且这个问题或多或少会让你感到发晕。更多的信息可以在Linux文档计划里找到。

在运行 iptables 时，你还可能得到另外一个错误信息：

```
iptables: No chain/target/match by that name
```

这说明你要用的链或 target、或 match 不存在，原因有很多，但最普遍的是你拼错了名字。当你想使用一个不可用的模块时也会产生这种错误。模块之所以不可用，可能是因为你没有装载正确的模块，或者内核里不包含那个模块，或者是 iptables 自动装载模块时失败了。通常，你不止应该考虑上面提到的所有解决办法，还要考虑规则中 target 的拼写错误，或者其他的原因。

未设置SYN的NEW状态包

iptables 有个“特点”没有被很好地给以说明，所以很多人（当然，也包括我）都忽视了它。这个“特点”就是：如果你使用状态 NEW，那么未设置 SYN 的包也会通过防火墙。之所以有这个特点，是因为在某些情况下，我们想把那样的包看作某个（比如是和另一个防火墙有关的）已处于 ESTABLISHED 状态的连接的一部分。这个特点使拥有两个或更多的防火墙协同工作成为可能，而且可使数据在服务器间无丢失的传输，如辅助防火墙可以接受子网的防火墙的操作。但它也会导致这样的事情：状态 NEW 会允许几乎所有的 TCP 连接进入，而不管是否有 3 次握手。为了处理这个问题，我们需要在防火墙的 INPUT 链、OUTPUT 链和 FORWARD 链加入如下规则（译者注：此规则作者称为“NEW not SYN rules”，下一小节还会提到）：

```
$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j LOG \
     --log-prefix "New not syn:"
$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
```

警告，在 Netfilter/iptables 项目中，这个特点所拥有的行为缺少文档说明，更明确的说，在你的防火墙上，它是一个很不安全的因素。

注意，这个规则用于 microsoft 的 TCP/IP（微软实现的 TCP/IP 就是不行，至少现在不行）产生的包时还是有些问题。如果包是由 microsoft 的产品生成的，且被标为状态 NEW，那么就会被此规则记录然后丢弃。看起来规则工作很正常啊，是吧。但问题就出在这儿了，因为连接无法中断了。这个问题出

现在关闭连接时，在最后一个包即 FIN/ACK 包发出后，Netfilter 的状态机制就会关闭连接、删除连接跟踪表里的相应记录。但就在这时，Microsoft 那不完善的程序会发送另外一个包，这个包就是那种未设置 SYN 且被认为是 NEW 状态的包，因此它就会被上面的规则匹配。换句话说，就是对这个规则不需要过于关注，如果你很在意它，就在规则里加入选项--log-headers 吧。这样，你就可以把包头记录下来，从而可以更好地了解相应的包。

对于这个规则，还有一些已知的问题。比如，某个连接（比如是从 LAN 发出的）已经连接到防火墙，而且有个脚本要在启动 PPP 时激活。当你启动 PPP 连接时，刚才提到的那个连接可能就会被干掉（be killed）。当然，这只会在特定的情况下才能发生，就是你把 conntrack 和 nat 作为模块运行，并且每次运行那个脚本时这两个模块都要被装入和卸载。如果你在防火墙之外的机子上运行 telnet，而且又通过这个 telnet 连接运行脚本 rc.firewall.txt，也会导致上面的问题。为了能简单地表达这个问题，你先准备一个 telnet 连接，或其他的流连接，再运行连接跟踪模块，然后装入上面的规则，最后，试着用 telnet client 或 daemon 发送一些数据。效果应该出来了，连接跟踪代码会认为这个连接是非法的，因为在此之前，它没有看到任何方向有包发出，更为严重的是现在连接上有了未设置 SYN 的包，因为刚才由 telnet client 或 daemon 发出的包肯定不是这个连接的第一个包。因此，上面的规则就起作用了，也就是说，这个包会被记录下来，然后被无情地扔掉，从而连接就会中断。

NEW状态的SYN/ACK包

某些，TCP 欺骗攻击所用的技术叫做序列号预测（Sequence Number Prediction）。在这类攻击中，攻击者利用另一台机子的 IP 访问攻击对象（译者注：这就是为什么叫欺骗的原因了，攻击者是想假冒另一台被攻击对象信任的机子，以达到欺骗攻击对象的目的），然后再试着预测攻击对象使用什么序列号。

我们来看看典型的使用序列号预测技术的欺骗是如何实现的，参与者：攻击者[A]（attacker）试图假装另一台机子[O]（other host）向受害者[V]（victim）发送数据。

1. [A]以[O]的 IP 为源地址向[V]发 SYN。
2. [V]向[O]回应 SYN/ACK。
3. 现在，若[O]以 RST 回应这个未知的 SYN/ACK，攻击就失败了，但如果[O]已经没有这个能力了呢？比如它早已被另外的攻击（如 SYN flood）降服，或者被关闭，或者它的 RST 包被防火墙拒绝。
4. 如果[O]没能破坏这条连接，而且[A]猜对了序列号，那它就能以[O]的身份和[V]交谈了。

只要我们没能在第三步以 RST 回应那个未知的 SYN/ACK 包，[V]就会被攻击，而且我们还会被连累（译者注：因为我们本身也被攻击了，而且还可能会

成为攻击者的替罪羊被起诉，呜呜，好惨）。所以，为安全起见，我们应该以正确的方式向[V]发送一个 RST 包。如果我们使用类似“NEW not SYN rules”（译者注：在上一小节中）的规则，SYN/ACK 包就可以被丢弃了。因此，我们在 bad_tcp_packets 链中加入了如下规则：

```
iptables -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
```

这样，你想成为上面那个[0]的机会就很少了（译者注：作者好幽默啊，我们可不想成为被别人利用的对象），而且这条规则在绝大部分情况下是安全的，不会有什么副作用，但多个防火墙要协同工作的情况要除外。那种情况下，防火墙之间会经常传递、接受包或流，有了这条规则，有些连接可能会被阻塞，即使是合法的连接。这条规则的存在还产生了另外一问题，就是有几个 portscan（端口扫描器）会看到我们的防火墙，但好在仅此而已。

使用私有IP地址的ISP

我的一位朋友告诉我说有些事我完全忘记了，从那时起，我就把这一节加上了。你刚上网时连接的网络是 ISP 提供的，但某些愚蠢的 ISP 在那个网络里使用的是私有地址，而那是 *IANA* 专门分配给局域网使用的。Swedish Internet Service Provider 和电话垄断企业 Telia 就是这样做的，例如在 DNS 服务器上，他们使用的 IP 地址段就是 10.x.x.x。我们最容易遇到的问题是，在这个脚本里，为了防止被欺骗，不允许从 10.x.x.x 发出的连接来访问我们。不幸的是，对于上面的例子，为了 DNS 能正常地被访问，我们不得不把规则的放宽松一些。也就是说，我们或者在刚才提到的那条防止欺骗的规则上面增加一条规则（如下），或者是把那条规则注释掉：

```
/usr/local/sbin/iptables -t nat -I PREROUTING -i eth1 -s \
     10.0.0.1/32 -j ACCEPT
```

我愿意对这些 ISP 再多费些唇舌。这些 IP 地址不是为了让你象这样愚蠢的使用而分配给你的，至少我知道不是这样的。对于一个大集团的站点或者是我们自己的家庭网络来说，这样用是很合适的，但你不能只因为你们的一些原因就强迫我们把自己公示于天下。

放行DHCP数据

一旦你了解 DHCP 是如何工作的，就会知道这其实是一个很简单的任务。但你必须小心处理到底让谁进入、不让谁进入。首先，我们要明白 DHCP 是工作在 UDP 协议之上的，所以，UDP 协议是我们期望的第一个条件。其次，我们应该检查是从那个接口接收和发送请求的。例如，如果我们设置了 DHCP 使用接口 eth0，那就要阻塞 eth1 上的 DHCP 请求。为了让规则再详细些，我们只需打开（allow）DHCP 实际使用的 UDP 端口，一般都是 67 和 68。这两个端口是标准定义，我们就用它们来匹配被允许的包。现在，规则应该是这个样子的：

```
$IPTABLES  -I INPUT -i $LAN_IFACE -p udp --dport 67:68 --sport \
     67:68 -j ACCEPT
```

注意，现在我们能够接受所有来自和发往 UDP 端口 67、68 的数据，好像不太安全，但这并不是多大的问题，因为这条规则只允许从 67 或 68 端口连接的主机才能访问。当然，此规则还可以更严谨一些，但也应该足够接受所有的 DHCP 请求和更新，而不至于需要在防火墙上开一个大洞。如果你很在意现在的规则是否很宽松，你当然可以写一个限制条件更紧的

关于mIRC DCC的问题

mIRC 使用一个特殊的设定，它可以使 mIRC 连接穿过防火墙，也可以使 DCC 连接能在防火墙不了解它的情况下正常工作。如果此选项和 iptables 还有 ip_conntrack_irc 模块与 ip_nat_irc 模块一起使用，那 mIRC 就不能工作了。问题在于 mIRC 会自动对包进行 NAT 操作，这样当包到达防火墙后，防火墙就完全不知道该对包做什么了，也不知道该怎么做。如果是防火墙来处理，它只是简单地用自己的 IP 去询问 IRC 服务器，然后用那个地址发送 DCC 请求。mIRC 不希望防火墙自作聪明地以这种方式代替自己来处理这个包。

打开“I am behind a firewall”（我在防火墙后）这个配置选项并且使用 ip_conntrack_irc 和 ip_nat_irc 模块，会导致 Netfilter 建立包含“Forged DCC send packet”的记录。

最简单的解决办法是不要选中 mIRC 的那个选项而让 iptables 来做这些工作。意思就是要明确地告诉 mIRC，它不是在防火墙后面的。

附录C ICMP类型

这是一个完整的 ICMP 类型的列表：

Iptables 和 netfilter 用 ICMP 类型 255 来匹配所有的 ICMP 报文，例如-A INPUT -p icmp --icmp-type 255 -j DROP，这样所有的 ICMP 报文都会被丢弃。

Table C-1. ICMP types

TYPE	CODE	Description	Query	Error	Reference
0	0	Echo Reply	x		RFC792
3	0	Network Unreachable		x	RFC792
3	1	Host Unreachable		x	RFC792
3	2	Protocol Unreachable		x	RFC792
3	3	Port Unreachable		x	RFC792
3	4	Fragmentation needed but no frag. bit set		x	RFC792
3	5	Source routing failed		x	RFC792
3	6	Destination network unknown		x	RFC792
3	7	Destination host unknown		x	RFC792
3	8	Source host isolated (obsolete)		x	RFC792
3	9	Destination network administratively prohibited		x	RFC792
3	10	Destination host administratively prohibited		x	RFC792
3	11	Network unreachable for TOS		x	RFC792
3	12	Host unreachable for TOS		x	RFC792
3	13	Communication administratively prohibited by filtering		x	RFC1812
3	14	Host precedence violation		x	RFC1812
3	15	Precedence cutoff in effect		x	RFC1812
4	0	Source quench			RFC792
5	0	Redirect for network			RFC792
5	1	Redirect for host			
5	2	Redirect for TOS and network			RFC792
5	3	Redirect for TOS and host			RFC792
8	0	Echo request	x		RFC792
9	0	Router advertisement - Normal			RFC1256

TYPE	CODE	Description	Query	Error	Reference
		router advertisement			
9	16	Router advertisement - Does not route common traffic			RFC2002
10	0	Route selection			RFC1256
11	0	TTL equals 0 during transit		x	RFC792
11	1	TTL equals 0 during reassembly		x	RFC792
12	0	IP header bad (catchall error)		x	RFC792
12	1	Required options missing		x	RFC1108
12	2	IP Header bad length		x	RFC792
13	0	Timestamp request (obsolete)	x		RFC792
14		Timestamp reply (obsolete)	x		RFC792
15	0	Information request (obsolete)	x		RFC792
16	0	Information reply (obsolete)	x		RFC792
17	0	Address mask request	x		RFC950
18	0	Address mask reply	x		RFC950
20-29		Reserved for robustness experiment			Zaw-Sing Su
30	0	Traceroute	x		RFC1393
31	0	Datagram Conversion Error		x	RFC1475
32	0	Mobile Host Redirect			David Johnson
33	0	IPv6 Where-Are-You	x		Bill Simpson
34	0	IPv6 I-Am-Here	x		Bill Simpson
35	0	Mobile Registration Request	x		Bill Simpson
36	0	Mobile Registration Reply	x		Bill Simpson
39	0	SKIP			Tom Markson
40	0	Photuris			RFC2521

附录D TCP选项

下面是所有的 TCP 选项列表:

Table D-1. TCP Options

Copy	Class	Number	Value	Name	Reference
0	0	0	0	EOOL - End of Options List	[RFC791, JBP]
0	0	1	1	NOP - No Operation	[RFC791, JBP]
1	0	2	130	SEC - Security	[RFC1108]
1	0	3	131	LSR - Loose Source Route	[RFC791, JBP]
0	2	4	68	TS - Time Stamp	[RFC791, JBP]
1	0	5	133	E-SEC - Extended Security	[RFC1108]
1	0	6	134	CIPSO - Commercial Security	[???]
0	0	7	7	RR - Record Route	[RFC791, JBP]
1	0	8	136	SID - Stream ID	[RFC791, JBP]
1	0	9	137	SSR - Strict Source Route	[RFC791, JBP]
0	0	10	10	ZSU - Experimental Measurement	[ZSu]
0	0	11	11	MTUP - MTU Probe	[RFC1191]*
0	0	12	12	MTUR - MTU Reply	[RFC1191]*
1	2	13	205	FINN - Experimental Flow Control	[Finn]
1	0	14	142	VISA - Experimental Access Control	[Estrin]
0	0	15	15	ENCODE - ???	[VerSteeg]
1	0	16	144	IMITD - IMI Traffic Descriptor	[Lee]
1	0	17	145	EIP - Extended Internet Protocol	[RFC1385]
0	2	18	82	TR - Traceroute	[RFC1393]
1	0	19	147	ADDEXT - Address Extension	[Ullmann IPv7]
1	0	20	148	RTRALT - Router Alert	[RFC2113]
1	0	21	149	SDB - Selective Directed Broadcast	[Graff]
1	0	22	150	NSAPA - NSAP Addresses	[Carpenter]
1	0	23	151	DPS - Dynamic Packet State	[Malis]
1	0	24	152	UMP - Upstream Multicast Pkt.	[Farinacci]

附录E. 其他资源和链接

这里有一些资源的链接，我从这些地方获得了不少信息，相信对你应该也很有帮助：

- ip-sysctl.txt -来自内核 2.4.14，一篇关于IP网络控制参数的短小精干的参考文章。
- RFC 768 - User Datagram Protocol - UDP协议的RFC
- RFC 791 - Internet Protocol - IP协议的RFC
- RFC 792 - Internet Control Message Protocol - 一篇很好的详细介绍ICMP协议的文章，作者是Ralph Walden
- RFC 793 - Transmission Control Protocol - TCP的权威文件，从1981年开始，它就成为TCP的规范了。只要你想学习TCP，就一定要读读这篇技术性很强的文章。作者：J. Postel
- RFC 1122 - Requirements for Internet Hosts - Communication Layers - This RFC defines the requirements of the software running on a Internet host, specifically the communication layers.
- RFC 1349 - Type of Service in the Internet Protocol Suite - 藐视了IP报文头的TOS字段变化。
- RFC 1812 - Requirements for IP Version 4 Routers - 路由的RFC
- RFC 2401 - Security Architecture for the Internet Protocol - IPSEC的RFC.
- RFC 2474 - Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers - In this document you will find out how the DiffServ works, and you will find much needed information about the TCP/IP protocol additions/changes needed for the DiffServ protocol to work.
- RFC 2638 - A Two-bit Differentiated Services Architecture for the Internet - This RFC describes a method of implementing two different differentiated service architecture into one. Both where described originally by D. Clark and van Jacobsen at the Munich IETH meeting 1997.
- RFC 2960 - Stream Control Transmission Protocol - This is a relatively new protocol developed by several large telecoms companies to complement UDP and TCP as a layer 3 protocol with higher reliability and resilience.
- RFC 3168 - The Addition of Explicit Congestion Notification (ECN) to IP - This RFC defines how ECN is to be used on a technical level and how it should be implemented in the TCP and IP protocols. Written by K. Ramakrishnan, S. Floyd and D. Black.
- RFC 3260 - New Terminology and Clarifications for Diffserv - This memo captures Diffserv working group agreements concerning

new and improved terminology, and provides minor technical clarifications.

- RFC 3286 - An Introduction to the Stream Control Transmission Protocol - RFC introducing the Stream Control Transmission Protocol, a relatively new layer 3 protocol in the TCP/IP stack. Developed by several large telecom companies.
- ip_dynaddr.txt - from the 2.4.14 kernel. A really short reference to the ip_dynaddr settings available via sysctl and the proc file system.
- iptables.8 - The iptables 1.3.1 man page. This is an HTMLized version of the man page which is an excellent reference when reading/writing iptables rule-sets. Always have it at hand.
- Ipsysctl tutorial - Another tutorial I have written about the IP System Control in Linux. A try to make a complete listing of all the IP variables that can be set on the fly in Linux.
- Policy Routing Using Linux - This is an excellent book that has now been opened up on the Internet regarding Policy routing in Linux. It is well written and most definitely worth buying. Written by Matthew G. Marsh.
- Security-Enhanced Linux - The official site of the Security-Enhanced Linux (SELinux) system developed as a proof of concept by the National Security Agency (NSA). SELinux is a fine grained Mandatory Access Control system, which lets you have a much higher control on who can do what and what processes has what privileges, et cetera.
- Firewall rules table - A small PDF document gracefully given to this project by Stuart Clark, which gives a reference form where you can write all of the information needed for your firewall, in a simple manner.
- http://l7-filter.sourceforge.net/ - The l7-filter project is basically a set of patches and files to make iptables and netfilter able to handle layer 7 filtering, mainly for QoS and traffic accounting. It works less reliably for filtering however, since it will allow the first couple of packets through before actually blocking traffic.
- http://www.netfilter.org/ - The official Netfilter and iptables site. It is a must for everyone wanting to set up iptables and Netfilter in linux.
- http://www.insecure.org/nmap/ - Nmap is one of the best, and most known, port scanners available. It is very useful when debugging your firewall scripts. Take a closer look at it.
- http://www.netfilter.org/documentation/index.html#FAQ - The official Netfilter *Frequently Asked Questions*. Also a good

place to start at when wondering what iptables and Netfilter is about.

- http://www.netfilter.org/unreliable-guides/packet-filtering-HOWTO/index.html - Rusty Russells Unreliable Guide to packet filtering. Excellent documentation about basic packet filtering with iptables written by one of the core developers of iptables and Netfilter.
- http://www.netfilter.org/unreliable-guides/NAT-HOWTO/index.html - Rusty Russells Unreliable Guide to Network Address Translation. Excellent documentation about Network Address Translation in iptables and Netfilter written by one of the core developers, Rusty Russell.
- http://www.netfilter.org/unreliable-guides/netfilter-hacking-HOWTO/index.html - Rusty Russells Unreliable Netfilter Hacking HOW-TO. One of the few documentations on how to write code in the Netfilter and iptables user-space and kernel space code-base. This was also written by Rusty Russell.
- http://www.linuxguruz.org/iptables/ - Excellent link-page with links to most of the pages on the Internet about iptables and Netfilter. Also maintains a list of iptables scripts for different purposes.
- Policy Routing using Linux - The best book I have ever read on Policy routing nad linux. This is an absolute must when it comes to routing in linux. Written by Matthew G. Marsh.
- Implementing Quality of Service Policies with DSCP - A link about the cisco implementation of DSCP. This shows some classes used in DSCP, and so on.
- IETF SIP Working Group - SIP is one of the "next big things" it seems. Basically it is the defacto standards for Internet telephony today. It is horribly complex as you can see from the amount of documentation on the working groups homepage, and should hopefully be able to cope with pretty much any needs of session initiation in the future. It is used mainly to setup peer to peer connections between known users, for example to connect to user@example.org and setup a phone connection to that user. This is the IETF Working group handling all SIP work.
- IETF TLS Working Group - TLS is a transport layer security model that is one of the most common host to server based security mechanisms. The current version is running is 1.1 and work is ongoing to get 1.2 out the door with support for newer and better cryptos as of this writing. This is a standardized way of sending and receiving public keys for servers and handling trusted certificate agents etc. For more information, read the RFC's on this page.

- IPSEC Howto - This is the official IPSEC howto for Linux 2.6 kernels. It describes how IPSEC works in the 2.6 kernels and up, however, it is not the place to find out exactly how the Linux 2.2 and 2.4 kernels worked when it comes to IPSEC. Go to the *FreeS/WAN* site for that information.
- FreeS/WAN - This is the official site for FreeS/WAN, an IPSEC implementation for the Linux 2.2 and 2.4 kernel series. This site contains documentation and all necessary downloads for the IPSEC implementation. This effort has been discontinued due to several reasons discussed on the page, but efforts will still be put into bugfixes, documentation and the forums. For an IPSEC implementation for Linux 2.6 kernels, please look at the *IPSEC Howto* site and the information there.
- http://www.islandsoft.net/veerapen.html -Excellent discussion on automatic hardening of iptables and how to make small changes that will make your computer automatically add hostile sites to a special ban list in iptables .
- /etc/protocols - An example protocols file taken from the Slackware distribution. This can be used to find out what protocol number different protocols have, such as the IP, ICMP or TCP protocols have.
- /etc/services - An example services file taken from the Slackware distribution. This is extremely good to get used to reading once in a while, specifically if you want to get a basic look at what protocols runs on different ports.
- Internet Assigned Numbers Authority - The IANA is the organisation that is responsible for fixing all numbers in the different protocols in an orderly fashion. If anyone has a specific addition to make to a protocol (for example, adding a new TCP option), they need to contact the IANA, which will assign the numbers requested. In other words, extremely important site to keep an eye on.
- RFC-editor.org - This is an excellent site for finding RFC documents in a fast and orderly way. Functions for searching RFC documents, and general information about the RFC community (I.e., errata, news, et cetera).
- Internet Engineering Task Force - This is one of the biggest groups when it comes to setting and maintaining Internet standards. They are the ones maintaining the RFC repository, and consist of a large group of companies and individuals that work together to ensure the interoperability of the Internet.
- Linux Advanced Routing and Traffic Control HOW-TO - This site hosts the Linux Advanced Routing and Traffic Control HOWTO. It

is one of the biggest and best documents regarding Linux advanced routing. Maintained by Bert Hubert.

- Paksecured Linux Kernel patches - A site containing all of the kernel patches written by Matthew G. Marsh. Among others, the FTOS patch is available here.
- ULOGD project page - The homepage of the ULOGD site.
- The Linux Documentation Project is a great site for documentation. Most big documents for Linux is available here, and if not in the TLDP, you will have to search the net very carefully. If there is anything you want to know more about, check this site out.
- Snort - this is an excellent open source "network intrusion detection system" (NIDS) which looks for signatures in the packets that it sees, and if it sees a signature of some kind of attack or break-in it can do different actions that can be defined (notifying the administrator, or take action, or simply logging it).
- Tripwire - tripwire is an excellent security tool which can be used to find out about host intrusions. It makes checksums of all the files specified in a configuration file, and then it tells the administrator about any files that has been tampered with in an illegit way every time it is run.
- Squid - This is one of the most known webproxies available on the market. It is open source, and free. It can do several of the filtering tasks that should be done before the traffic actually hits your webserver, as well as doing the standard webcaching functions for your networks.
- http://kalamazoolinux.org/presentations/20010417/conntrack.html - This presentation contains an excellent explanation of the conntrack modules and their work in Netfilter. If you are interested in more documentation on conntrack, this is a "must read".
- http://www.docum.org/ - Excellent information about the CBQ, tc and the ip commands in Linux. One of the few sites that has any information at all about these programs. Maintained by Stef Coene.
- http://lists.samba.org/mailman/listinfo/netfilter- The official Netfilter mailing-list. Extremely useful in case you have questions about something not covered in this document or any of the other links here.

And of course the iptables source, documentation and individuals who helped me.

附录F. 感谢

很多朋友在我写这篇文章时给了我热心的帮助，我要感谢他们:

- *Fabrice Marie*, 对我糟糕的语法和拼写做了大量的订正，还用make文件等工具把这篇指南转换成了DocBook。
- *Marc Boucher*, 在状态匹配代码的使用方面给了我很多帮助.
- *Frode E. Nyboe*, 大幅度改善了rc.firewall的规则，当我要重写这个规则集、把多个表的遍历（the multiple table traversing）引入同一份文件时，给了我很多灵感.
- *Chapman Brad*, *Alexander W. Janssen*, 开始时，我对包如何穿越nat和filter 表的理解是错误的，是他们使我了解到这一点的，而且他们还给了我正确的顺序.
- *Michiel Brandenburg*, *Myles Uyema*, 帮我解决了一些状态匹配代码，并让它正常问题.
- *Kent `Artech' Stahre*, 帮我绘制图形，还帮我查错
- *Anders 'DeZENT' Johansson*, 提示我有些古怪的 ISP 在 Internet 上使用保留的网址，至少对他来说遇到了这样的情况.
- *Jeremy `Spliffy' Smith*, 提示我有些内容容易使大家糊涂，还帮我进行了测试和查错.

还有很多人，我和他们进行过讨论，也请教过他们，这里不能一一提及了.

Appendix G. History

Version 1.2.2 (19 Nov 2006)
http://iptables-tutorial.frozentux.net
By Oskar Andreasson
Contributors: Jens Larsson and G. W. Haywood.

Version 1.2.1 (29 Sep 2006)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Ortwin Glueck, Mao, Marcos Roberto Greiner, Christian Font,
Tatiana, Andrius, Alexey Dushechkin, Tatsuya Nonogaki and Fred.

Version 1.2.0 (20 July 2005)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Corey Becker, Neil Perrins, Watz and Spanish translation team.

Version 1.1.19 (21 May 2003)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Peter van Kampen, Xavier Bartol, Jon Anderson, Thorsten Bremer
and Spanish Translation Team.

Version 1.1.18 (24 Apr 2003)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Stuart Clark, Robert P. J. Day, Mark Orenstein and Edmond Shwayri.

Version 1.1.17 (6 Apr 2003)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Geraldo Amaral Filho, Ondrej Suchy, Dino Conti, Robert P. J. Day,
Velev Dimo, Spencer Rouser, Daveonos, Amanda Hickman, Olle Jonsson and
Bengt Aspvall.

Version 1.1.16 (16 Dec 2002)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Clemens Schwaighower, Uwe Dippel and Dave Wreski.

Version 1.1.15 (13 Nov 2002)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Mark Sonarte, A. Lester Buck, Robert P. J. Day, Togan Muftuoglu,
Antony Stone, Matthew F. Barnes and Otto Matejka.

Version 1.1.14 (14 Oct 2002)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Carol Anne, Manuel Minzoni, Yves Soun, Miernik, Uwe Dippel,
Dave Klipec and Eddy L O Jansson.

Version 1.1.13 (22 Aug 2002)
http://iptables-tutorial.haringstad.com

By: Oskar Andreasson
Contributors: Tons of people reporting bad HTML version.

Version 1.1.12 (19 Aug 2002)
http://www.netfilter.org/tutorial/
By: Oskar Andreasson
Contributors: Peter Schubnell, Stephen J. Lawrence, Uwe Dippel, Bradley
Dilger, Vegard Engen, Clifford Kite, Alessandro Oliveira, Tony Earnshaw,
Harald Welte, Nick Andrew and Stepan Kasal.

Version 1.1.11 (27 May 2002)
http://www.netfilter.org/tutorial/
By: Oskar Andreasson
Contributors: Steve Hnizdur, Lonni Friedman, Jelle Kalf, Harald Welte, Valentina Barrios and Tony Earnshaw.

Version 1.1.10 (12 April 2002)
http://www.boingworld.com/workshops/linux/iptables-tutorial/
By: Oskar Andreasson
Contributors: Jelle Kalf, Theodore Alexandrov, Paul Corbett, Rodrigo Rubira Branco, Alistair Tonner, Matthew G. Marsh, Uwe Dippel, Evan Nemerson and Marcel J.E. Mol.

Version 1.1.9 (21 March 2002)
http://www.boingworld.com/workshops/linux/iptables-tutorial/
By: Oskar Andreasson
Contributors: Vince Herried, Togan Muftuoglu, Galen Johnson, Kelly Ashe, Janne
Johansson, Thomas Smets, Peter Horst, Mitch Landers, Neil Jolly, Jelle Kalf,
Jason Lam and Evan Nemerson.

Version 1.1.8 (5 March 2002)
http://www.boingworld.com/workshops/linux/iptables-tutorial/
By: Oskar Andreasson

Version 1.1.7 (4 February 2002)
http://www.boingworld.com/workshops/linux/iptables-tutorial/
By: Oskar Andreasson
Contributors: Parimi Ravi, Phil Schultz, Steven McClintoc, Bill Dossett,

Dave Wreski, Erik Sj□lund, Adam Mansbridge, Vasoo Veerapen, Aladdin and
Rusty Russell.

Version 1.1.6 (7 December 2001)
http://people.unix-fu.org/andreasson/
By: Oskar Andreasson
Contributors: Jim Ramsey, Phil Schultz, G□ran B□ge, Doug Monroe, Jasper
Aikema, Kurt Lieber, Chris Tallon, Chris Martin, Jonas Pasche, Jan Labanowski, Rodrigo R. Branco, Jacco van Koll and Dave Wreski.

Version 1.1.5 (14 November 2001)
http://people.unix-fu.org/andreasson/
By: Oskar Andreasson
Contributors: Fabrice Marie, Merijn Schering and Kurt Lieber.

Version 1.1.4 (6 November 2001)
http://people.unix-fu.org/andreasson
By: Oskar Andreasson
Contributors: Stig W. Jensen, Steve Hnizdur, Chris Pluta and Kurt Lieber.

Version 1.1.3 (9 October 2001)
http://people.unix-fu.org/andreasson
By: Oskar Andreasson
Contributors: Joni Chu, N.Emile Akabi-Davis and Jelle Kalf.

Version 1.1.2 (29 September 2001)
http://people.unix-fu.org/andreasson
By: Oskar Andreasson

Version 1.1.1 (26 September 2001)
http://people.unix-fu.org/andreasson
By: Oskar Andreasson
Contributors: Dave Richardson.

Version 1.1.0 (15 September 2001)
http://people.unix-fu.org/andreasson
By: Oskar Andreasson

Version 1.0.9 (9 September 2001)
http://people.unix-fu.org/andreasson
By: Oskar Andreasson

Version 1.0.8 (7 September 2001)
http://people.unix-fu.org/andreasson
By: Oskar Andreasson

Version 1.0.7 (23 August 2001)
http://people.unix-fu.org/andreasson
By: Oskar Andreasson
Contributors: Fabrice Marie.

Version 1.0.6
http://people.unix-fu.org/andreasson
By: Oskar Andreasson

Version 1.0.5
http://people.unix-fu.org/andreasson
By: Oskar Andreasson
Contributors: Fabrice Marie.

Appendix H. GNU Free Documentation License

Version 1.1, March 2000

---

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get

credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

---

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

---

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept

compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

---

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

---

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

---

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this

License in all other respects regarding verbatim copying of that document.

---

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

---

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

---

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is

void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

---

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-

Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix I. GNU General Public License

Version 2, June 1991

---

0. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

---

1. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or

with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of

this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   A. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   B. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   C. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial

distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

1. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
2. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
3. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of

the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

4. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

   If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

5. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

6. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.
7. NO WARRANTY

   BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

8. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

---

2. How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) year name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for
> details type `show w'. This is free software, and you are
> welcome to redistribute it under certain conditions; type
> `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest
> in the program `Gnomovision' (which makes passes at
> compilers) written by James Hacker. <signature of Ty
> Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix J. Example scripts code-base

Example rc.firewall script

```
#!/bin/sh
#
# rc.firewall - Initial SIMPLE IP Firewall script for Linux 2.4.x and
iptables
#
# Copyright (C) 2001  Oskar Andreasson <blueﬂuxATkoffeinDOTnet>
```

```
#
# This program is free software; you can redistribute it and/or
modify
# it under the terms of the GNU General Public License as published
by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59
Temple
# Place, Suite 330, Boston, MA  02111-1307  USA
#

##############################################################################
######
#
# 1. Configuration options.
#

#
# 1.1 Internet Configuration.
#

INET_IP="194.236.50.155"
INET_IFACE="eth0"
INET_BROADCAST="194.236.50.255"

#
# 1.1.1 DHCP
#

#
# 1.1.2 PPPoE
#

#
# 1.2 Local Area Network configuration.
#
```

```
# your LAN's IP range and localhost IP. /24 means to only use the
first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#

LAN_IP="192.168.0.2"
LAN_IP_RANGE="192.168.0.0/16"
LAN_IFACE="eth1"

#
# 1.3 DMZ Configuration.
#

#
# 1.4 Localhost Configuration.
#

LO_IFACE="lo"
LO_IP="127.0.0.1"

#
# 1.5 IPTables Configuration.
#

IPTABLES="/usr/sbin/iptables"

#
# 1.6 Other Configuration.
#

##############################################################################
######
#
# 2. Module loading.
#

#
# Needed to initially load modules
#

/sbin/depmod -a

#
# 2.1 Required modules
```

```
#

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state

#
# 2.2 Non-Required modules
#

#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ipt_MASQUERADE
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc

########################################################################
######
#
# 3. /proc set up.
#

#
# 3.1 Required proc configuration
#

echo "1" > /proc/sys/net/ipv4/ip_forward

#
# 3.2 Non-Required proc configuration
#

#echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
#echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr
```

```
#########################################################################
######
#
# 4. rules set up.
#

######
# 4.1 Filter table
#

#
# 4.1.1 Set policies
#

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# 4.1.2 Create userspecified chains
#

#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_tcp_packets

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N tcp_packets
$IPTABLES -N udp_packets
$IPTABLES -N icmp_packets

#
# 4.1.3 Create content in userspecified chains
#

#
# bad_tcp_packets chain
#
```

```
$IPTABLES -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j
LOG \
--log-prefix "New not syn:"
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j
DROP

#
# allowed chain
#

$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j
ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# TCP rules
#

$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 22 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 80 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 113 -j allowed

#
# UDP ports
#

#$IPTABLES -A udp_packets -p UDP -s 0/0 --destination-port 53 -j
ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --destination-port 123 -j
ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --destination-port 2074 -j
ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --destination-port 4000 -j
ACCEPT

#
# In Microsoft Networks you will be swamped by broadcasts. These
lines
# will prevent them from showing up in the logs.
```

```
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d $INET_BROADCAST \
#--destination-port 135:139 -j DROP

#
# If we get DHCP requests from the Outside of our network, our logs
will
# be swamped as well. This rule will block them from getting logged.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d 255.255.255.255 \
#--destination-port 67:68 -j DROP

#
# ICMP rules
#

$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# 4.1.4 INPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Rules for special networks not part of the Internet
#

$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -s $LAN_IP_RANGE -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LO_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LAN_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $INET_IP -j ACCEPT

#
# Special rule for DHCP requests from LAN, which are not caught
properly
# otherwise.
```

```
#

$IPTABLES -A INPUT -p UDP -i $LAN_IFACE --dport 67 --sport 68 -j
ACCEPT

#
# Rules for incoming packets from the internet.
#

$IPTABLES -A INPUT -p ALL -d $INET_IP -m state --state
ESTABLISHED,RELATED \
-j ACCEPT
$IPTABLES -A INPUT -p TCP -i $INET_IFACE -j tcp_packets
$IPTABLES -A INPUT -p UDP -i $INET_IFACE -j udp_packets
$IPTABLES -A INPUT -p ICMP -i $INET_IFACE -j icmp_packets

#
# If you have a Microsoft Network on the outside of your firewall,
you may
# also get flooded by Multicasts. We drop them so we do not get
flooded by
# logs
#

#$IPTABLES -A INPUT -i $INET_IFACE -d 224.0.0.0/8 -j DROP

#
# Log weird packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
# 4.1.5 FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets

#
```

```
# Accept the packets we actually want to forward
#

$IPTABLES -A FORWARD -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG
\
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# 4.1.6 OUTPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $INET_IP -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG
\
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

######
# 4.2 nat table
#
```

```
#
# 4.2.1 Set policies
#

#
# 4.2.2 Create user specified chains
#

#
# 4.2.3 Create content in user specified chains
#

#
# 4.2.4 PREROUTING chain
#

#
# 4.2.5 POSTROUTING chain
#

#
# Enable simple IP Forwarding and Network Address Translation
#

$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j SNAT --to-source
$INET_IP

#
# 4.2.6 OUTPUT chain
#

######
# 4.3 mangle table
#

#
# 4.3.1 Set policies
#

#
# 4.3.2 Create user specified chains
#

#
```

```
# 4.3.3 Create content in user specified chains
#

#
# 4.3.4 PREROUTING chain
#

#
# 4.3.5 INPUT chain
#

#
# 4.3.6 FORWARD chain
#

#
# 4.3.7 OUTPUT chain
#

#
# 4.3.8 POSTROUTING chain
#
```

Example rc.DMZ.firewall script

```
#!/bin/sh
#
# rc.DMZ.firewall - DMZ IP Firewall script for Linux 2.4.x and
iptables
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or
modify
# it under the terms of the GNU General Public License as published
by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59
Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

########################################################################
######
#
# 1. Configuration options.
#

#
# 1.1 Internet Configuration.
#

INET_IP="194.236.50.152"
HTTP_IP="194.236.50.153"
DNS_IP="194.236.50.154"
INET_IFACE="eth0"

#
# 1.1.1 DHCP
#

#
# 1.1.2 PPPoE
#

#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the
first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#

LAN_IP="192.168.0.1"
LAN_IFACE="eth1"

#
```

```
# 1.3 DMZ Configuration.
#

DMZ_HTTP_IP="192.168.1.2"
DMZ_DNS_IP="192.168.1.3"
DMZ_IP="192.168.1.1"
DMZ_IFACE="eth2"

#
# 1.4 Localhost Configuration.
#

LO_IFACE="lo"
LO_IP="127.0.0.1"

#
# 1.5 IPTables Configuration.
#

IPTABLES="/usr/sbin/iptables"

#
# 1.6 Other Configuration.
#

##############################################################################
######
#
# 2. Module loading.
#

#
# Needed to initially load modules
#
/sbin/depmod -a



#
# 2.1 Required modules
#

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
```

```
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state

#
# 2.2 Non-Required modules
#

#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ipt_MASQUERADE
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc

########################################################################
######
#
# 3. /proc set up.
#

#
# 3.1 Required proc configuration
#

echo "1" > /proc/sys/net/ipv4/ip_forward

#
# 3.2 Non-Required proc configuration
#

#echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
#echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr

########################################################################
######
#
# 4. rules set up.
#
```

```
######
# 4.1 Filter table
#

#
# 4.1.1 Set policies
#

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# 4.1.2 Create userspecified chains
#

#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_tcp_packets

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N icmp_packets

#
# 4.1.3 Create content in userspecified chains
#

#
# bad_tcp_packets chain
#

$IPTABLES -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j
LOG \
--log-prefix "New not syn:"
```

```
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j
DROP

#
# allowed chain
#

$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j
ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# ICMP rules
#

# Changed rules totally
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# 4.1.4 INPUT chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Packets from the Internet to this box
#

$IPTABLES -A INPUT -p ICMP -i $INET_IFACE -j icmp_packets

#
# Packets from LAN, DMZ or LOCALHOST
#

#
# From DMZ Interface to DMZ firewall IP
#
```

```
$IPTABLES -A INPUT -p ALL -i $DMZ_IFACE -d $DMZ_IP -j ACCEPT

#
# From LAN Interface to LAN firewall IP
#

$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -d $LAN_IP -j ACCEPT

#
# From Localhost interface to Localhost IP's
#

$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LO_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LAN_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $INET_IP -j ACCEPT

#
# Special rule for DHCP requests from LAN, which are not caught
properly
# otherwise.
#

$IPTABLES -A INPUT -p UDP -i $LAN_IFACE --dport 67 --sport 68 -j
ACCEPT

#
# All established and related packets incoming from the internet to
the
# firewall
#

$IPTABLES -A INPUT -p ALL -d $INET_IP -m state --state
ESTABLISHED,RELATED \
-j ACCEPT

#
# In Microsoft Networks you will be swamped by broadcasts. These
lines
# will prevent them from showing up in the logs.
#

#$IPTABLES -A INPUT -p UDP -i $INET_IFACE -d $INET_BROADCAST \
#--destination-port 135:139 -j DROP
```

```
#
# If we get DHCP requests from the Outside of our network, our logs
will
# be swamped as well. This rule will block them from getting logged.
#

#$IPTABLES -A INPUT -p UDP -i $INET_IFACE -d 255.255.255.255 \
#--destination-port 67:68 -j DROP

#
# If you have a Microsoft Network on the outside of your firewall,
you may
# also get flooded by Multicasts. We drop them so we do not get
flooded by
# logs
#

#$IPTABLES -A INPUT -i $INET_IFACE -d 224.0.0.0/8 -j DROP

#
# Log weird packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
# 4.1.5 FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets


#
# DMZ section
#
# General rules
#

$IPTABLES -A FORWARD -i $DMZ_IFACE -o $INET_IFACE -j ACCEPT
```

```
$IPTABLES -A FORWARD -i $INET_IFACE -o $DMZ_IFACE -m state \
--state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -i $LAN_IFACE -o $DMZ_IFACE -j ACCEPT
$IPTABLES -A FORWARD -i $DMZ_IFACE -o $LAN_IFACE -m state \
--state ESTABLISHED,RELATED -j ACCEPT

#
# HTTP server
#

$IPTABLES -A FORWARD -p TCP -i $INET_IFACE -o $DMZ_IFACE -d
$DMZ_HTTP_IP \
--dport 80 -j allowed
$IPTABLES -A FORWARD -p ICMP -i $INET_IFACE -o $DMZ_IFACE -d
$DMZ_HTTP_IP \
-j icmp_packets

#
# DNS server
#

$IPTABLES -A FORWARD -p TCP -i $INET_IFACE -o $DMZ_IFACE -d
$DMZ_DNS_IP \
--dport 53 -j allowed
$IPTABLES -A FORWARD -p UDP -i $INET_IFACE -o $DMZ_IFACE -d
$DMZ_DNS_IP \
--dport 53 -j ACCEPT
$IPTABLES -A FORWARD -p ICMP -i $INET_IFACE -o $DMZ_IFACE -d
$DMZ_DNS_IP \
-j icmp_packets

#
# LAN section
#

$IPTABLES -A FORWARD -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG
\
```

```
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# 4.1.6 OUTPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $INET_IP -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG
\
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

######
# 4.2 nat table
#

#
# 4.2.1 Set policies
#

#
# 4.2.2 Create user specified chains
#

#
# 4.2.3 Create content in user specified chains
#
```

```
#
# 4.2.4 PREROUTING chain
#

$IPTABLES -t nat -A PREROUTING -p TCP -i $INET_IFACE -d $HTTP_IP --
dport 80 \
-j DNAT --to-destination $DMZ_HTTP_IP
$IPTABLES -t nat -A PREROUTING -p TCP -i $INET_IFACE -d $DNS_IP --
dport 53 \
-j DNAT --to-destination $DMZ_DNS_IP
$IPTABLES -t nat -A PREROUTING -p UDP -i $INET_IFACE -d $DNS_IP --
dport 53 \
-j DNAT --to-destination $DMZ_DNS_IP

#
# 4.2.5 POSTROUTING chain
#

#
# Enable simple IP Forwarding and Network Address Translation
#

$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j SNAT --to-source
$INET_IP

#
# 4.2.6 OUTPUT chain
#

######
# 4.3 mangle table
#

#
# 4.3.1 Set policies
#

#
# 4.3.2 Create user specified chains
#

#
# 4.3.3 Create content in user specified chains
#
```

```
#
# 4.3.4 PREROUTING chain
#

#
# 4.3.5 INPUT chain
#

#
# 4.3.6 FORWARD chain
#

#
# 4.3.7 OUTPUT chain
#

#
# 4.3.8 POSTROUTING chain
#
```

Example rc.UTIN.firewall script

```
#!/bin/sh
#
# rc.UTIN.firewall - UTIN Firewall script for Linux 2.4.x and
iptables
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or
modify
# it under the terms of the GNU General Public License as published
by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
```

```
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59
Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

########################################################################
######
#
# 1. Configuration options.
#

#
# 1.1 Internet Configuration.
#

INET_IP="194.236.50.155"
INET_IFACE="eth0"
INET_BROADCAST="194.236.50.255"

#
# 1.1.1 DHCP
#

#
# 1.1.2 PPPoE
#

#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the
first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#

LAN_IP="192.168.0.2"
LAN_IP_RANGE="192.168.0.0/16"
LAN_IFACE="eth1"

#
# 1.3 DMZ Configuration.
#
```

```
#
# 1.4 Localhost Configuration.
#

LO_IFACE="lo"
LO_IP="127.0.0.1"

#
# 1.5 IPTables Configuration.
#

IPTABLES="/usr/sbin/iptables"

#
# 1.6 Other Configuration.
#

##############################################################################
######
#
# 2. Module loading.
#

#
# Needed to initially load modules
#

/sbin/depmod -a

#
# 2.1 Required modules
#

/sbin/modprobe ip_tables
/sbin/modprobe ip_conntrack
/sbin/modprobe iptable_filter
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_state

#
```

```
# 2.2 Non-Required modules
#

#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ipt_MASQUERADE
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc

########################################################################
######
#
# 3. /proc set up.
#

#
# 3.1 Required proc configuration
#

echo "1" > /proc/sys/net/ipv4/ip_forward

#
# 3.2 Non-Required proc configuration
#

#echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
#echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr

########################################################################
######
#
# 4. rules set up.
#

######
# 4.1 Filter table
#

#
# 4.1.1 Set policies
#
```

```
$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# 4.1.2 Create userspecified chains
#

#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_tcp_packets

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N tcp_packets
$IPTABLES -N udp_packets
$IPTABLES -N icmp_packets

#
# 4.1.3 Create content in userspecified chains
#

#
# bad_tcp_packets chain
#

$IPTABLES -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j
LOG \
--log-prefix "New not syn:"
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j
DROP

#
# allowed chain
#
```

```
$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j
ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# TCP rules
#

$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 22 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 80 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 113 -j allowed

#
# UDP ports
#

#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 53 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 123 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 2074 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 4000 -j ACCEPT

#
# In Microsoft Networks you will be swamped by broadcasts. These
lines
# will prevent them from showing up in the logs.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d $INET_BROADCAST \
#--destination-port 135:139 -j DROP

#
# If we get DHCP requests from the Outside of our network, our logs
will
# be swamped as well. This rule will block them from getting logged.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d 255.255.255.255 \
#--destination-port 67:68 -j DROP

#
# ICMP rules
#
```

```
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# 4.1.4 INPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Rules for special networks not part of the Internet
#

$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LO_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LAN_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $INET_IP -j ACCEPT

#
# Rules for incoming packets from anywhere.
#

$IPTABLES -A INPUT -p ALL -d $INET_IP -m state --state
ESTABLISHED,RELATED \
-j ACCEPT
$IPTABLES -A INPUT -p TCP -j tcp_packets
$IPTABLES -A INPUT -p UDP -j udp_packets
$IPTABLES -A INPUT -p ICMP -j icmp_packets

#
# If you have a Microsoft Network on the outside of your firewall,
you may
# also get flooded by Multicasts. We drop them so we do not get
flooded by
# logs
#

#$IPTABLES -A INPUT -i $INET_IFACE -d 224.0.0.0/8 -j DROP

#
```

```
# Log weird packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
# 4.1.5 FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets

#
# Accept the packets we actually want to forward
#

$IPTABLES -A FORWARD -p tcp --dport 21 -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -p tcp --dport 80 -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -p tcp --dport 110 -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG
\
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# 4.1.6 OUTPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
```

```
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $INET_IP -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG
\
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

######
# 4.2 nat table
#

#
# 4.2.1 Set policies
#

#
# 4.2.2 Create user specified chains
#

#
# 4.2.3 Create content in user specified chains
#

#
# 4.2.4 PREROUTING chain
#

#
# 4.2.5 POSTROUTING chain
#

#
# Enable simple IP Forwarding and Network Address Translation
#
```

```
$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j SNAT --to-source
$INET_IP

#
# 4.2.6 OUTPUT chain
#

######
# 4.3 mangle table
#

#
# 4.3.1 Set policies
#

#
# 4.3.2 Create user specified chains
#

#
# 4.3.3 Create content in user specified chains
#

#
# 4.3.4 PREROUTING chain
#

#
# 4.3.5 INPUT chain
#

#
# 4.3.6 FORWARD chain
#

#
# 4.3.7 OUTPUT chain
#

#
# 4.3.8 POSTROUTING chain
#
```

Example rc.DHCP.firewall script

```
#!/bin/sh
#
# rc.DHCP.firewall - DHCP IP Firewall script for Linux 2.4.x and
iptables
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or
modify
# it under the terms of the GNU General Public License as published
by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59
Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

########################################################################
######
#
# 1. Configuration options.
#

#
# 1.1 Internet Configuration.
#

INET_IFACE="eth0"

#
# 1.1.1 DHCP
#
```

```
#
# Information pertaining to DHCP over the Internet, if needed.
#
# Set DHCP variable to no if you don't get IP from DHCP. If you get
DHCP
# over the Internet set this variable to yes, and set up the proper
IP
# address for the DHCP server in the DHCP_SERVER variable.
#

DHCP="no"
DHCP_SERVER="195.22.90.65"

#
# 1.1.2 PPPoE
#

# Configuration options pertaining to PPPoE.
#
# If you have problem with your PPPoE connection, such as large mails
not
# getting through while small mail get through properly etc, you may
set
# this option to "yes" which may fix the problem. This option will
set a
# rule in the PREROUTING chain of the mangle table which will clamp
# (resize) all routed packets to PMTU (Path Maximum Transmit Unit).
#
# Note that it is better to set this up in the PPPoE package itself,
since
# the PPPoE configuration option will give less overhead.
#

PPPOE_PMTU="no"

#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the
first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#
```

```
LAN_IP="192.168.0.2"
LAN_IP_RANGE="192.168.0.0/16"
LAN_IFACE="eth1"

#
# 1.3 DMZ Configuration.
#

#
# 1.4 Localhost Configuration.
#

LO_IFACE="lo"
LO_IP="127.0.0.1"

#
# 1.5 IPTables Configuration.
#

IPTABLES="/usr/sbin/iptables"

#
# 1.6 Other Configuration.
#

################################################################################
######
#
# 2. Module loading.
#

#
# Needed to initially load modules
#

/sbin/depmod -a

#
# 2.1 Required modules
#

/sbin/modprobe ip_conntrack
/sbin/modprobe ip_tables
/sbin/modprobe iptable_filter
```

```
/sbin/modprobe iptable_mangle
/sbin/modprobe iptable_nat
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_limit
/sbin/modprobe ipt_MASQUERADE

#
# 2.2 Non-Required modules
#

#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc

########################################################################
######
#
# 3. /proc set up.
#

#
# 3.1 Required proc configuration
#

echo "1" > /proc/sys/net/ipv4/ip_forward

#
# 3.2 Non-Required proc configuration
#

#echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
#echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr

########################################################################
######
#
# 4. rules set up.
#

######
```

```
# 4.1 Filter table
#

#
# 4.1.1 Set policies
#

$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# 4.1.2 Create userspecified chains
#

#
# Create chain for bad tcp packets
#

$IPTABLES -N bad_tcp_packets

#
# Create separate chains for ICMP, TCP and UDP to traverse
#

$IPTABLES -N allowed
$IPTABLES -N tcp_packets
$IPTABLES -N udp_packets
$IPTABLES -N icmp_packets

#
# 4.1.3 Create content in userspecified chains
#

#
# bad_tcp_packets chain
#

$IPTABLES -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j
LOG \
--log-prefix "New not syn:"
```

```
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j
DROP

#
# allowed chain
#

$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j
ACCEPT
$IPTABLES -A allowed -p TCP -j DROP

#
# TCP rules
#

$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 22 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 80 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 113 -j allowed

#
# UDP ports
#

$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 53 -j ACCEPT
if [ $DHCP == "yes" ] ; then
 $IPTABLES -A udp_packets -p UDP -s $DHCP_SERVER --sport 67 \
 --dport 68 -j ACCEPT
fi

#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 53 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 123 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 2074 -j ACCEPT
#$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port 4000 -j ACCEPT

#
# In Microsoft Networks you will be swamped by broadcasts. These
lines
# will prevent them from showing up in the logs.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE \
#--destination-port 135:139 -j DROP
```

```
#
# If we get DHCP requests from the Outside of our network, our logs
will
# be swamped as well. This rule will block them from getting logged.
#

#$IPTABLES -A udp_packets -p UDP -i $INET_IFACE -d 255.255.255.255 \
#--destination-port 67:68 -j DROP

#
# ICMP rules
#

$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# 4.1.4 INPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Rules for special networks not part of the Internet
#

$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -s $LAN_IP_RANGE -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -j ACCEPT

#
# Special rule for DHCP requests from LAN, which are not caught
properly
# otherwise.
#

$IPTABLES -A INPUT -p UDP -i $LAN_IFACE --dport 67 --sport 68 -j
ACCEPT

#
```

```
# Rules for incoming packets from the internet.
#

$IPTABLES -A INPUT -p ALL -i $INET_IFACE -m state --state
ESTABLISHED,RELATED \
-j ACCEPT
$IPTABLES -A INPUT -p TCP -i $INET_IFACE -j tcp_packets
$IPTABLES -A INPUT -p UDP -i $INET_IFACE -j udp_packets
$IPTABLES -A INPUT -p ICMP -i $INET_IFACE -j icmp_packets

#
# If you have a Microsoft Network on the outside of your firewall,
you may
# also get flooded by Multicasts. We drop them so we do not get
flooded by
# logs
#

#$IPTABLES -A INPUT -i $INET_IFACE -d 224.0.0.0/8 -j DROP

#
# Log weird packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
# 4.1.5 FORWARD chain
#

#
# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets

#
# Accept the packets we actually want to forward
#

$IPTABLES -A FORWARD -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
#
# Log weird packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG
\
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# 4.1.6 OUTPUT chain
#

#
# Bad TCP packets we don't want.
#

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -o $INET_IFACE -j ACCEPT

#
# Log weird packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG
\
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

######
# 4.2 nat table
#

#
# 4.2.1 Set policies
#

#
# 4.2.2 Create user specified chains
```

```
#

#
# 4.2.3 Create content in user specified chains
#

#
# 4.2.4 PREROUTING chain
#

#
# 4.2.5 POSTROUTING chain
#

if [ $PPPOE_PMTU == "yes" ] ; then
 $IPTABLES -t nat -A POSTROUTING -p tcp --tcp-flags SYN,RST SYN \
 -j TCPMSS --clamp-mss-to-pmtu
fi
$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j MASQUERADE

#
# 4.2.6 OUTPUT chain
#

######
# 4.3 mangle table
#

#
# 4.3.1 Set policies
#

#
# 4.3.2 Create user specified chains
#

#
# 4.3.3 Create content in user specified chains
#

#
# 4.3.4 PREROUTING chain
#
```

```
#
# 4.3.5 INPUT chain
#

#
# 4.3.6 FORWARD chain
#

#
# 4.3.7 OUTPUT chain
#

#
# 4.3.8 POSTROUTING chain
#
```

Example rc.flush-iptables script

```
#!/bin/sh
#
# rc.flush-iptables - Resets iptables to default values.
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or
modify
# it under the terms of the GNU General Public License as published
by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59
Temple
# Place, Suite 330, Boston, MA  02111-1307  USA
```

```
#
# Configurations
#
IPTABLES="/usr/sbin/iptables"

#
# reset the default policies in the filter table.
#
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT
$IPTABLES -P OUTPUT ACCEPT

#
# reset the default policies in the nat table.
#
$IPTABLES -t nat -P PREROUTING ACCEPT
$IPTABLES -t nat -P POSTROUTING ACCEPT
$IPTABLES -t nat -P OUTPUT ACCEPT

#
# reset the default policies in the mangle table.
#
$IPTABLES -t mangle -P PREROUTING ACCEPT
$IPTABLES -t mangle -P POSTROUTING ACCEPT
$IPTABLES -t mangle -P INPUT ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT
$IPTABLES -t mangle -P FORWARD ACCEPT

#
# flush all the rules in the filter and nat tables.
#
$IPTABLES -F
$IPTABLES -t nat -F
$IPTABLES -t mangle -F
#
# erase all chains that's not default in filter and nat table.
#
$IPTABLES -X
$IPTABLES -t nat -X
$IPTABLES -t mangle -X
```

Example rc.test-iptables script

```
#!/bin/bash
#
# rc.test-iptables - test script for iptables chains and tables.
#
# Copyright (C) 2001  Oskar Andreasson <bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or
modify
# it under the terms of the GNU General Public License as published
by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59
Temple
# Place, Suite 330, Boston, MA  02111-1307  USA
#

#
# Filter table, all chains
#
iptables -t filter -A INPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="filter INPUT:"
iptables -t filter -A INPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="filter INPUT:"
iptables -t filter -A OUTPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="filter OUTPUT:"
iptables -t filter -A OUTPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="filter OUTPUT:"
iptables -t filter -A FORWARD -p icmp --icmp-type echo-request \
-j LOG --log-prefix="filter FORWARD:"
iptables -t filter -A FORWARD -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="filter FORWARD:"
```

```
#
# NAT table, all chains except OUTPUT which don't work.
#
iptables -t nat -A PREROUTING -p icmp --icmp-type echo-request \
-j LOG --log-prefix="nat PREROUTING:"
iptables -t nat -A PREROUTING -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="nat PREROUTING:"
iptables -t nat -A POSTROUTING -p icmp --icmp-type echo-request \
-j LOG --log-prefix="nat POSTROUTING:"
iptables -t nat -A POSTROUTING -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="nat POSTROUTING:"
iptables -t nat -A OUTPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="nat OUTPUT:"
iptables -t nat -A OUTPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="nat OUTPUT:"

#
# Mangle table, all chains
#
iptables -t mangle -A PREROUTING -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle PREROUTING:"
iptables -t mangle -A PREROUTING -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle PREROUTING:"
iptables -t mangle -I FORWARD 1 -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle FORWARD:"
iptables -t mangle -I FORWARD 1 -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle FORWARD:"
iptables -t mangle -I INPUT 1 -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle INPUT:"
iptables -t mangle -I INPUT 1 -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle INPUT:"
iptables -t mangle -A OUTPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle OUTPUT:"
iptables -t mangle -A OUTPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle OUTPUT:"
iptables -t mangle -I POSTROUTING 1 -p icmp --icmp-type echo-request
\
-j LOG --log-prefix="mangle POSTROUTING:"
iptables -t mangle -I POSTROUTING 1 -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle POSTROUTING:"
```

Index

Symbols

--hashlimit-htable-max, Hashlimit match
--hashlimit-htable-size, Hashlimit match
--hashlimit-mode, Hashlimit match
--hashlimit-name, Hashlimit match
--hashmode, CLUSTERIP target
--helper, Helper match
--hitcount, Recent match
--icmp-type, ICMP matches
--in-interface, Generic matches
--length, Length match
--limit, Limit match
--limit-burst, Limit match
--local-node, CLUSTERIP target
--log-ip-options, LOG target options
--log-level, LOG target options
--log-prefix, LOG target options
--log-tcp-options, LOG target options
--log-tcp-sequence, LOG target options
--mac-source, Mac match
--mark, Connmark match, Mark match
--mask, CONNMARK target
--match, Implicit matches
--mss, Tcpmss match
--name, Recent match
--new, CLUSTERIP target
--nodst, SAME target
--out-interface, Generic matches
--pid-owner, Owner match
--pkt-type, Packet type match
--pkt-type match, Packet type match
--port, Multiport match
--protocol, Generic matches
--queue-num, NFQUEUE target
--rcheck, Recent match
--rdest, Recent match
--realm, Realm match
--reject-with, REJECT target
--remove, Recent match
--restore, CONNSECMARK target
--restore-mark, CONNMARK target
--rsource, Recent match
--rttl, Recent match
--save, CONNSECMARK target
--save-mark, CONNMARK target

--seconds, Recent match
--selctx, SECMARK target
--set, Recent match
--set-class, CLASSIFY target
--set-dscp, DSCP target
--set-dscp-class, DSCP target
--set-mark, CONNMARK target, MARK target
--set-mss, TCPMSS target
--set-tos, TOS target
--sid-owner, Owner match
--source, Generic matches
--source-port, TCP matches, UDP matches, SCTP matches, Multiport match
--src-range, IP range match
--src-type, Addrtype match
--state, State match
--syn, TCP matches
--tcp-flags, TCP matches
--tcp-option, TCP matches
--to, NETMAP target, SAME target
--to-destination, DNAT target
--to-destination target, DNAT target
--to-ports, MASQUERADE target, REDIRECT target
--to-source, SNAT target
--tos, Tos match
--total-nodes, CLUSTERIP target
--ttl-dec, TTL target
--ttl-eq, Ttl match
--ttl-gt, Ttl match
--ttl-inc, TTL target
--ttl-lt, Ttl match
--ttl-set, TTL target
--uid-owner, Owner match
--ulog-cprange, ULOG target
--ulog-nlgroup, ULOG target
--ulog-prefix, ULOG target
--ulog-qthreshold, ULOG target
--update, Recent match
[ASSURED], TCP connections
[UNREPLIED], TCP connections

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X