



Linux

网络安全技术与实现 (第2版)

极其灵活的防火墙使用技巧
全面的带宽合并和管理知识
您必须知晓的透明式防火墙
常见网络攻击以及防御之道
防火墙硬件评估和性能优化

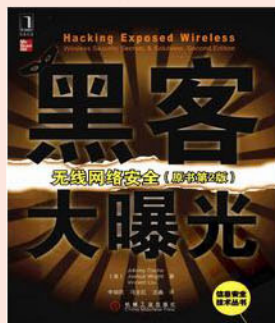


陈勇勋 著
黄强 审校

清华大学出版社



热读推荐



免费下载



免费下载



免费下载



免费下载

电子书共享请加QQ群: 208139770

最新分享动态 <http://e-book.blog.163.com>



免费下载



免费下载



免费下载



免费下载



免费下载



免费下载



免费下载



免费下载

Linux 网络安全技术与实现(第2版)

陈勇勋 著

黄 强 审校

清华大学出版社

北 京

Linux 網路安全技術與實現(第 2 版)

陈勇勋

精誠資訊股份有限公司-悦知文化, 2011.07

ISBN: 978-986-6072-14-7

本书为精诚资讯股份有限公司-悦知文化授权清华大学出版社于中国大陆(台港澳除外)地区之中文简体版本。本著作物之专有出版权为精诚资讯股份有限公司-悦知文化所有。该专有出版权受法律保护, 任何人不得侵害之。

本书封面贴有清华大学出版社防伪标签, 无标签者不得销售。

版权所有, 侵权必究。侵权举报电话: 010-62782989 13701121933

图书在版编目(CIP)数据

Linux 网络安全技术与实现(第 2 版)/陈勇勋 著; 黄强 审校. —北京: 清华大学出版社, 2012.3

ISBN 978-7-302-27886-3

I. L… II. ①陈… ②黄… III. Linux 操作系统—安全技术 IV. TP316.89

中国版本图书馆 CIP 数据核字(2012)第 008395 号

责任编辑: 王 军 韩宏志

封面设计: 康 博

责任校对: 蔡 娟

责任印制: 何 芊

出版发行: 清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址: 北京清华大学学研大厦 A 座 邮 编: 100084

社 总 机: 010-62770175 邮 购: 010-62786544

投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者: 清华大学印刷厂

装 订 者: 三河市新茂装订有限公司

经 销: 全国新华书店

开 本: 185mm×230mm 印 张: 32

字 数: 655 千字

版 次: 2012 年 3 月第 1 版

印 次: 2012 年 3 月第 1 次印刷

印 数: 1~4000

定 价: 68.00 元

产品编号: 044541-01

推荐序

现在开源软件日趋丰富，熟悉这个领域的技术人才也越来越多，这对企业，尤其是中小企业来说不啻是一大福音。

企业信息系统包含各种不同用途、不同规模的服务器以及各种应用程序，在规划系统时，规划人员必须谨慎考虑哪些位置上可以使用免费资源；在这方面，那些使用者直接接触到，但又必不可少服务器可以作为重点考虑对象。本书使用Linux实现网络安全就是一个很好的例子。信息安全极其重要，但如果考虑到成本，每个企业管理者都会犹豫一番，毕竟还没有遇到问题就要先投入一大笔钱，其紧迫性显然不如将钱花在直接与业绩挂钩的事情上。但如果安全成本较低，并且掌握这种技术的人才又容易找到，企业管理者将可以放心地、大刀阔斧地部署安全解决方案。

本书就是要提供这样的解决方案，不管是为自己的公司降低信息系统的成本，或是为别的公司建设新的系统，本书都有极高的参考价值。如果你已经是Linux相关的技术人员，本书会提高你的应用能力；如果你是Windows平台的技术人员，本书会增加你跨领域就业的资本。

作者陈勇勋常年研究Linux的相关技术，教学经验丰富，已经撰写了多本畅销著作。他一直以推广开源软件的应用为己任，在这个领域的技术研究上投入了很多心血，相信由他来编写本书，可以让读者快速掌握Linux安全技术知识并将所学技术得心应手地运用于实际工作中。

张智凯

Richard Chang

精诚资讯知识产品事业部处长

作者序

很多年前我就一直想撰写本书，只是时间上实在不允许，如今总算了结了许多年来的一桩心愿，我为此感到十分欣慰。当然这还要感谢屠立刚和唐任威两位老师，他们总是恰逢其时地为我提供理论指导，使我得以突破许多盲点来完成这一著作。

在接触因特网的这16年来，我亲眼目睹Linux系统不断成长和和完善，心中总有一种莫名的感动，因为背后的功臣竟是一群不求回报的、夜以继日默默付出的志愿者，而他们的成果却是足以抗衡拥有千亿资产的商业公司，甚至超越了他们的产品。由于缺少市场营销的推动，Linux系统的市场占有率始终不高，当然这样也就埋没了Linux许多令人赞赏的功能，如防火墙(Layer2、3、7)、流量控制(Traffic Control)、基于策略的路由(Advance Routing)、虚拟专用网络(Virtual Private Network)等，这也是激发我撰写本书的动力；我的下一个目标是完成《Linux系统虚拟化技术》一书，请大家拭目以待吧！

本书可分为安全、基于策略路由、流量控制及通信安全四大部分，主要内容如下

■ 安全

以Linux内置的Netfilter为主线，介绍Netfilter模块的用法及组合应用上的技巧，再逐步延伸出应用层防火墙(Layer 7 Firewall)及透明式防火墙(Transparent Firewall)的技术原理及应用，还讨论如何结合反向代理机制来确保企业网络的安全。

■ 基于策略的路由

基于策略的路由一直都是网络管理人员梦寐以求的利器。有了基于策略的路由之后，你可以将网络上所有的数据包玩于股掌，例如，所有http数据包经由第一条ADSL连接因特网，企业中一半的客户端使用第一条ADSL，另一半的客户端使用第二条ADSL，甚至还可以做到让企业对外的网络连接平均分摊到多条ADSL上等高难度的网络管理操作。其实市场上很多负载均衡设备就是使用Linux来搭建的，而且这个功能通常只存在于价格高昂的商用路由器之上。

■ 流量控制

讲述如何使用Linux内置的流量控制功能来管理网络上的流量；除了传统上以IP及端口的方式来限制流量之外，也将介绍如何使用应用层协议来控制流量，以便轻松地管理企业对带宽的使用。

■ 虚拟专用网络

由于企业海外分部的建立及无线上网的普及，使得网络通信安全主题受到人们的重视，VPN硬件设备也因此逐渐流行起来。这样的设备通常价格不菲，但我们只要巧妙使用Linux系统，就可以建立起适于企业使用的VPN系统。

只要你能随着本书的章节循序渐进地学习，一定可以很快将Linux系统应用于企业网络安全的管理上。

陈勇勋

目 录

第 1 章 防火墙的基本概念	1
1.1 TCP/IP的基本概念	2
1.1.1 应用层	2
1.1.2 传输层	3
1.1.3 网络层	4
1.1.4 链路层	4
1.2 数据包传输	4
1.3 TCP、UDP及Socket的关系	9
1.4 何谓防火墙	12
1.5 防火墙的判断依据	14
1.5.1 各层数据包包头内的信息	14
1.5.2 数据包所承载的数据内容	16
1.5.3 连接状态	16
1.6 防火墙的分类	17
1.6.1 数据包过滤防火墙	17
1.6.2 应用层防火墙	18
1.7 常见的防火墙结构	19
1.7.1 单机防火墙	19
1.7.2 网关式防火墙	20
1.7.3 透明防火墙	24
1.8 小结	24
第 2 章 Netfilter/iptables	25
2.1 何谓内核	26
2.2 何谓Netfilter	27
2.3 Netfilter与Linux的关系	27
2.4 Netfilter工作的位置	28
2.5 Netfilter的命令结构	30
2.6 Netfilter的filter机制	31

2.7 规则的匹配方式	35
2.8 Netfilter与iptables的关系	36
2.9 iptables工具的使用方法	38
2.9.1 iptables命令参数	38
2.9.2 iptables规则语法	48
2.9.3 学以致用: iptables的规则语法	56
2.10 使用iptables机制来构建简单的单机防火墙	57
2.10.1 如何测试防火墙规则正确与否	59
2.10.2 解决无法在防火墙主机上对外建立连接的问题	62
2.10.3 管理防火墙规则数据库的办法	68
2.11 使用filter机制来构建网关式防火墙	71
2.12 Netfilter的NAT机制	73
2.12.1 IP网段的划分	73
2.12.2 私有IP	74
2.12.3 NAT	74
2.12.4 数据包传输方向与SNAT及DNAT的关系	76
2.12.5 NAT的分类	79
2.12.6 NAT并非无所不能	86
2.13 Netfilter的Mangle机制	86
2.14 Netfilter的raw机制	89
2.15 小结	91
第3章 Netfilter的匹配方式及处理方法	93
3.1 匹配方式	94
3.1.1 内置的匹配方式	94
3.1.2 从模块扩展而来的匹配方式	98
3.2 处理方法	139
3.2.1 内置的处理方法	139
3.2.2 由模块扩展的处理方法	142
3.3 小结	153
第4章 Netfilter/Iptables的高级技巧	155
4.1 防火墙性能的最优化	156

4.1.1 调整防火墙规则顺序	156
4.1.2 巧妙使用multiport及iprange模块	158
4.1.3 巧妙使用用户定义的链	158
4.2 Netfilter连接处理能力与内存消耗	159
4.2.1 计算最大连接数	160
4.2.2 调整连接跟踪数	160
4.2.3 连接跟踪数量与内存消耗	164
4.3 使用raw 表	162
4.4 简单及复杂通信协议的处理	163
4.4.1 简单通信协议	163
4.4.2 复杂通信协议	164
4.4.3 ICMP数据包的处理原则	171
4.4.4 在DMZ上使用NAT将面临的问题及解决方案	172
4.4.5 常见的网络攻击手段及防御方法	175
4.5 小结	191
第5章 代理服务器的应用	193
5.1 何谓代理服务器	194
5.2 代理服务器支持的通信协议	195
5.3 代理服务器的分类	195
5.3.1 何谓缓存代理	195
5.3.2 何谓反向代理	196
5.4 代理服务器的硬件要求	197
5.5 安装Squid代理	198
5.6 使用Squid构建缓存代理	199
5.6.1 缓存代理的基本配置	199
5.6.2 缓存代理客户端的配置	204
5.6.3 缓存代理的高级配置	205
5.6.4 缓存代理连接访问控制	209
5.6.5 缓存对象的管理	210
5.6.6 Squid代理的工作日志	214
5.6.7 Squid代理的名称解析	216
5.7 透明代理	217

5.7.1	透明代理的工作原理	217
5.7.2	透明代理的配置	218
5.8	反向代理	219
5.8.1	Web 服务器的分类	219
5.8.2	构建反向代理	221
5.9	小结	226
第 6 章	使用Netfilter/Iptables保护企业网络	227
6.1	防火墙结构的选择	228
6.2	防火墙本机的安全	230
6.2.1	网络攻击	230
6.2.2	系统入侵	231
6.2.3	入站/出站的考虑事项	231
6.2.4	远程管理的安全考虑事项	232
6.3	防火墙的规则定义	232
6.3.1	企业内部与因特网	232
6.3.2	DMZ与因特网	234
6.3.3	企业内部与DMZ	238
6.4	入侵与防御的其他注意事项	238
6.4.1	更新系统软件	238
6.4.2	Syn Flooding攻击防御	238
6.4.3	IP欺骗防御	241
6.5	小结	242
第 7 章	Linux内核编译	243
7.1	为何需要重新编译内核	245
7.2	内核编译	246
7.2.1	安装软件开发环境	246
7.2.2	获取内核源代码	247
7.2.3	整合源代码	248
7.2.4	设置编译完成后的内核版本号	249
7.2.5	清理内核源代码以外的临时文件	249
7.2.6	设置内核编译参数	250

7.2.7	执行编译操作	252
7.2.8	安装模块及结构中心	253
7.2.9	修改开机管理程序	255
7.3	如何安装内核补丁	257
7.3.1	下载补丁文件及内核源代码	257
7.3.2	准备内核及补丁的源代码	258
7.3.3	运行内核补丁	259
7.3.4	设置内核编译参数	259
7.3.5	内核编译完毕后的检查	260
7.4	小结	260
第 8 章	应用层防火墙	261
8.1	如何为iptables安装补丁	263
8.2	Layer7模块识别应用层协议的原理	264
8.3	安装Layer7模块的模式	265
8.4	如何使用Layer7模块	267
8.5	Layer7模块使用示例说明	268
8.6	结合使用包过滤器与Layer7模块	271
8.7	小结	273
第 9 章	透明式防火墙	275
9.1	何谓桥接模式	278
9.2	何谓透明式防火墙	279
9.3	构建透明式防火墙	279
9.3.1	使用Linux构建网桥	280
9.3.2	Netfilter在Layer3及Layer2的工作逻辑	284
9.3.3	另一种透明式防火墙	290
9.3.4	配置代理ARP	290
9.4	小结	292
第 10 章	基于策略的路由及多路带宽合并	293
10.1	何谓基于策略的路由	294

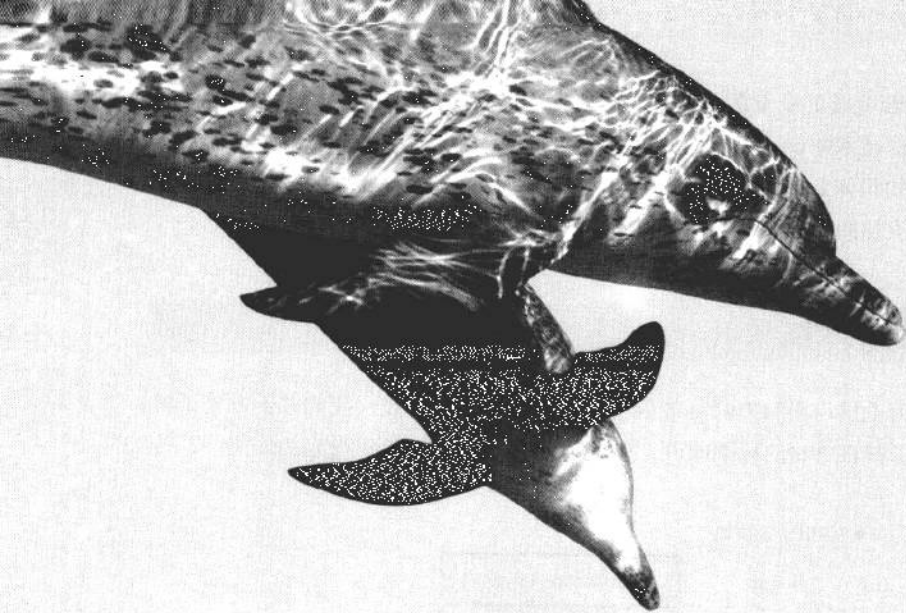
10.2	了解Linux的路由机制	296
10.3	路由策略数据库与路由表的管理	298
10.3.1	管理策略数据库	298
10.3.2	管理路由表	302
10.4	带宽合并	305
10.4.1	何谓带宽合并	306
10.4.2	企业内的带宽合并	307
10.5	小结	319
第 11 章	Linux的带宽管理	321
11.1	队列	322
11.1.1	不可分类的队列算法	323
11.1.2	可分类的队列算法	323
11.2	Linux带宽管理	324
11.3	过滤器	325
11.3.1	FW过滤器	326
11.3.2	U32过滤器	326
11.4	带宽管理部署示例	326
11.4.1	带宽划分	327
11.4.2	设置队列算法	327
11.4.3	设置队列规则	328
11.4.4	设置过滤器	329
11.4.5	测试	330
11.5	带宽借用	332
11.6	类别中的队列	334
11.7	Linux带宽管理的限制	335
11.8	网桥模式中的带宽管理	338
11.9	多接口的带宽管理	339
11.9.1	为内核及iptables安装补丁	340
11.9.2	多接口带宽管理	341
11.10	实际案例	343
11.11	小结	348

第 12 章 流量统计	349
12.1 安装及测试SNMP服务器	350
12.1.1 安装SNMP服务器	350
12.1.2 测试SNMP服务器	351
12.2 安装及设置MRTG	352
12.2.1 安装MRTG	352
12.2.2 设置MRTG	352
12.2.3 使用cfgmaker工具编写MRTG针对网卡的配置文件	353
12.3 另一种网络流量监测方式	357
12.3.1 结合使用Netfilter/Iptables和MRTG来监测网络流量	357
12.3.2 手动编写MRTG的配置文件	359
12.4 外部程序及MRTG配置文件的示例	360
12.5 小结	362
第 13 章 弱点扫描、入侵检测及主动防御系统	363
13.1 何谓弱点扫描	364
13.1.1 OpenVAS弱点扫描工具	364
13.1.2 OpenVAS弱点扫描工具的工作架构	365
13.1.3 下载及安装OpenVAS弱点扫描工具	365
13.1.4 进行弱点扫描	368
13.2 入侵检测系统	374
13.2.1 网络设备的限制	374
13.2.2 入侵检测系统的分类	375
13.2.3 入侵检测系统的部署	375
13.2.4 Snort入侵检测系统介绍	376
13.2.5 下载及安装Snort入侵检测系统	377
13.2.6 下载及安装Snort的规则数据库	378
13.2.7 配置Snort	381
13.2.8 Snort的启停	382
13.2.9 Snort的警告	382
13.3 主动防御系统	383
13.3.1 下载Guardian	384
13.3.2 安装Guardian	384

13.3.3	设置Guardian	385
13.3.4	Guardian的启停	386
13.4	小结	387
第 14 章	VPN基础篇	389
14.1	何谓VPN	390
14.1.1	VPN的原理	392
14.1.2	常见的VPN架构	393
14.1.3	VPN的安全问题	393
14.1.4	VPN机制的优缺点	393
14.2	数据加解密	394
14.2.1	何谓“明文”	394
14.2.2	何谓“密文”	395
14.3	数据加密类型	396
14.3.1	对称加密	396
14.3.2	非对称加密	397
14.4	哈希算法	398
14.4.1	常见的哈希算法	399
14.4.2	哈希算法的特性	399
14.5	基于IPSec的VPN	400
14.5.1	IPSec的工作模式	400
14.5.2	IPSec的组成要素	401
14.5.3	AH及ESP协议运行时需要设置的参数	409
14.5.4	安装IPSec参数的管理工具	411
14.5.5	配置传输模式IPSec VPN	411
14.6	Linux中的IPSec架构	420
14.6.1	IPSec机制的SPD	421
14.6.2	IPSec机制的SAD	422
14.7	小结	425
第 15 章	VPN实战篇	427
15.1	IKE	428
15.2	Preshared Keys验证模式下的传输模式VPN	433

15.2.1	数据库服务器的设置	434
15.2.2	客户端主机的设置	435
15.2.3	启动VPN	436
15.3	Preshared Keys验证模式下的隧道模式VPN	437
15.3.1	VPN 服务器(A)主机上的设置	438
15.3.2	VPN 服务器(B)主机上的设置	439
15.4	何谓数字证书	440
15.4.1	数字证书的必要性	440
15.4.2	证书管理中心	441
15.4.3	将Linux系统作为企业的CA	447
15.5	数字证书验证模式下的传输模式VPN	453
15.5.1	证书的生成及保存	453
15.5.2	客户端VPN主机的设置	454
15.6	数字证书验证模式下的隧道模式VPN	457
15.6.1	证书的生成及保存	457
15.6.2	设置VPN 服务器(A)	457
15.6.3	设置VPN 服务器(B)	458
15.6.4	启动IPSec	459
15.7	小结	459
第 16 章	VPN: L2TP Over IPSec	461
16.1	何谓PPP	462
16.2	何谓L2TP协议	462
16.2.1	L2TP协议的原理	463
16.2.2	L2TP协议的安全问题	465
16.2.3	L2TP协议安全问题的解决方案	465
16.2.4	Client to Site的L2TP VPN结构探讨	466
16.2.5	L2TP 客户端及服务器之间网段的选择	467
16.2.6	Proxy ARP的工作原理	467
16.3	构建L2TP VPN	470
16.3.1	配置L2TP服务器	470
16.3.2	配置PPP服务器	472
16.3.3	建立VPN的拨号帐户	472

16.3.4	证书的生成及保存	473
16.3.5	配置安全策略	473
16.3.6	IKE配置文件	474
16.3.7	启动L2TP服务器	475
16.4	配置L2TP客户端	475
16.4.1	生成L2TP客户端证书	475
16.4.2	将证书导入Windows XP/7系统前的准备工作	476
16.4.3	设置Windows XP系统上的L2TP客户端	476
16.4.4	设置Windows 7系统中的L2TP客户端	484
16.5	IPSec连接穿透NAT的问题	492
16.6	小结	494



LINUX

|第1章| 防火墙的基本概念

1



如果要构建一个固若金汤的企业级网络防火墙,需要具备熟练的操作系统使用能力,也需要完整掌握网络通信技术知识,而其中最为重要的莫过于TCP/IP的网络知识,但这却是大多数网络技术自学者知识体系中最薄弱的一环。因此本书开头将详细描述与防火墙相关的TCP/IP技术,并讨论防火墙的原理、种类、架构及其优缺点。

1.1 TCP/IP的基本概念

OSI标准将网络分为七层,而TCP/IP标准仅把网络分为四层。不过,我们并不需要了解OSI的七层架构,只需了解TCP/IP的标准即可。如图1-1所示,其自上而下分别为应用层、传输层、网络层及链路层。

● TCP/IP协议模型

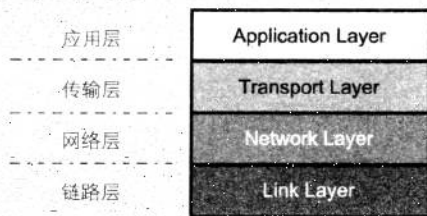


图1-1 TCP/IP网络模型

1.1.1 应用层

常用的应用层通信协议包括HTTP、HTTPS、SMTP、POP3及FTP等,这些协议主要用于定义客户端与服务器端的数据交换方法,如Outlook Express如何使用POP3协议来接收电子邮件。

示例1.1显示了POP3协议的工作过程。本示例将使用Telnet客户端工具模拟POP3客户端,流程如下。

步骤01: 使用Telnet客户端工具连接192.168.1.254主机的端口110。

步骤02: POP3客户端向POP3服务器端发送帐户。

步骤03: POP3客户端向POP3服务器端发送密码。

步骤04: POP3客户端向POP3服务器端发送list命令,要求POP3服务器端列出该POP3客户端在服务器端上的所有信件。

步骤05: POP3客户端向POP3服务器端发送retr 1命令,要求POP3服务器端发送信箱中的第1封电子邮件给POP3客户端。示例1.1中的虚线部分即为返回的电子邮件内容。

步骤06: POP3 客户端向POP3服务器端发送quit命令, 以便终止POP3协议的连接。

示例1.1 POP3通信协议说明

```
[root@mail etc]# telnet 192.168.1.254 110 ❶
Trying 192.168.1.254...
Connected to localhost.localdomain (192.168.1.254).
Escape character is '^'.
+OK dovecot ready.
user kevin ❷
+OK
pass 12345 ❸
+OK Logged in.
List ❹
+OK 1 messages:
retr 1 ❺
+OK 568 octets
Return-Path: <jacky@example.com>
X-Original-To: kevin
Delivered-To: kevin@example.com
Received: by mail.example.com (Postfix, from userid 500)
        id 7F7FE1FADE; Mon, 19 Dec 2005 15:04:25 +0800 (CST)
To: kevin@example.com
Subject: TEST-1
Message-Id: <20051219070425.7F7FE1FADE@mail.example.com>
Date: Mon, 19 Dec 2005 15:04:25 +0800 (CST)
From: jacky@example.com
X-IMAPbase: 1134975837
Status: O
X-UID: 1
Content-Length: 13
X-Keywords:
*
1234567890
.
quit ❻
+OK Logging out.
Connection closed by foreign host.
[root@mail etc]#
```

1.1.2 传输层

传输层用于定义数据传输方法, 传输层主要定义了两种通信协议, 分别是TCP协议和UDP协议。

- TCP协议: TCP协议在传输数据过程中会检查数据的完整性, 因此传输中的数据是不会丢失的。如传输一封电子邮件, 那么就应该选择TCP通信协议作为其传输数据的方法。

- UDP协议: 当选择UDP协议作为数据传输方法时, 其目的通常在于满足效率方面的要求, 而非数据正确性方面的要求。如网络广播电台, 因为在一连串的语音信号中, 若有数据丢失或者出现误码, 并不会造成接收端听众的困扰, 因此要求发送端重新发送正确的数据也不具有任何实质意义。

1.1.3 网络层

使用IP(Internet Protocol)地址来定位网络上的每一台计算机, 并采用路由(Routing)方法决定数据传输路径, 将数据传输到正确的目的端。

1.1.4 链路层

链路层又称为数据链路层(Data Link)或网络接入层(Network Interface), 也就是网络的基础设施, 它可能是以太网(Ethernet)、光纤(Fiber)、无线网络(Wireless)、帧中继网络(Frame Relay)或者点对点网络(PPP)等实体网络, 其最重要的任务是传输和接收实体网络层所传输的光电信号。

1.2 数据包传输

数据在传输过程中, 必须被分解成一个个的小碎片, 然后才能够被传输。这就如同我们在运送大批的货物, 因为每辆卡车所能运载的货物量是有限的, 如果货物无法在一辆卡车上全部装载完毕, 就必须使用多辆卡车来执行这项任务。

在网络世界里也是同样的道理, 因为不同的网络实体层技术, 其每次所能承载的数据量不同。例如, 光纤为4352个字节, 而以太网为1500个字节, 因此, 数据在传输过程中, 必须先被分解成一个个的小碎片才能被传输, 而数据传输过程是一层层地由上往下传送。

以图1-2为例, 当使用者在计算机上运行某一应用程序(如MSN)时, 该应用程序一定会先定义一种数据交换方法(应用层通信协议), 接着必须确定数据传输方式, 例如, 数据在传输过程中是不可丢失或者错误的, 那么就需要使用TCP作为数据传输的方法(传输层通信协议); 接着, 为了能将数据正确地传输到目的端, 我们使用网络上每台计算机唯一的识别码IP地址, 作为发送端和接收端的地址, 但由于IP地址是属于逻辑信息, 无法以光电信号呈现, 而实体层的寻址方式是用MAC地址来识别(假设实体层是以太网), 因此, 当数据发送到实体层时, 会在该数据中附加上发送端和接收端的MAC地址, 这样便可以将数据传输到正确的目的地。

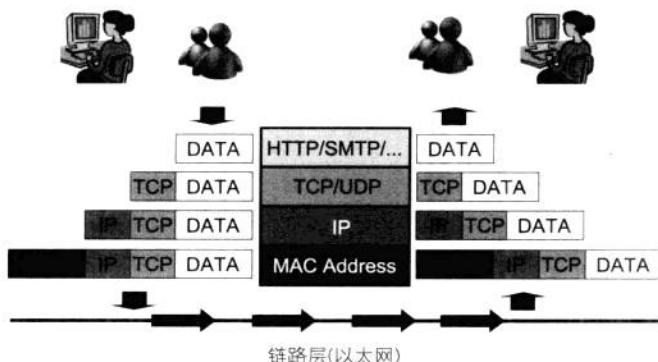


图1-2 数据包的传输

我们使用Fedora Core内置的Wireshark软件来截取网络上所传输的数据包，即可了解整个数据包传输的过程，如图1-3所示。以图1-3中编号6的数据包为例，中间的窗体部分就是数据包的结构，其结构分别如下：

- Frame6/Ethernet II：实体层。
- Internet Protocol：网络层。
- Transmission Control Protocol：传输层。
- Hypertext Transfer Protocol：应用层。

No.	Time	Source	Destination	Protocol	Info
5	2.990013	192.168.2.10	202.43.195.13	TCP	1857 > http [ACK]
6	2.990035	192.168.2.10	202.43.195.13	HTTP	GET / HTTP/1.1
7	3.011906	202.43.195.13	192.168.2.10	HTTP	HTTP/1.1 302 Found
8	3.022152	192.168.2.10	202.43.195.13	TCP	1858 > http [SYN]
# Frame 6 (276 bytes on wire, 276 bytes captured) # Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:11:22:33:44:55 # Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 202.43. # Transmission Control Protocol, Src Port: 1857 (1857), Dst Port: http (80), # Hypertext Transfer Protocol					
0000	00 11 22 33 44 55 00 0f	b0 93 b2 7d 08 00 45 00	..3DU... ..)..E.		
0010	01 06 7d 53 40 00 80 06	2c b3 c0 a8 02 0a ca 2b	..}S0... ..+.		
0020	c3 0d 07 41 00 50 f2 bb	65 5d be d3 1e 7a 50 18	..A.P.. e)...2P.		
0030	ff ff 5a 3a 00 00 47 45	54 20 2f 20 48 54 54 50	..2:...GE T / HTTP		
0040	2f 31 2e 31 0d 0a 41 63	63 65 70 74 3a 20 2a 2f	/1.1..Ac cept: */		
0050	2a 0d 0a 41 63 63 65 70	74 2d 4c 61 6e 67 75 61	*..Accep t-Langua		
0060	67 65 3a 20 7a 68 2d 74	77 0d 0a 41 63 63 65 70	ge: zh-t w..Accep		
0070	74 2d 45 6e 63 6f 64 69	6e 67 3a 20 67 7a 69 70	T-Encodi ng: gzip		
0080	2c 20 64 65 66 6c 61 74	65 0d 0a 55 73 65 72 2d	, deflat e..user-		

图1-3 Wireshark截取的数据包

接下来继续深入探讨每层的内容及用途。首先介绍应用层，当我们把应用层的结构展开后，即可看到整个应用层的数据内容，如图1-4所示。从数据内容中可以看到，客户端使用HTTP1.1协议中的GET方法①，去获取Web服务器根目录的HTML文档；接下来要选择一种数据传输方法，将这些数据正确地传输到目的地，因为HTTP的数据包是不允许有任何一个包丢失的，所以我们选用TCP协议来传输这个数据包，而这个数据包随即被加上一个TCP包

头，以表明这个数据包是要使用TCP协议来传输的。

```

Frame 6 (276 bytes on wire (276 bytes captured))
Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:11:22:33:44:55
Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 202.43.195.13 (202.43.195.13)
Transmission Control Protocol, Src Port: 1857 (1857), Dst Port: http (80), Seq: 1, Ack: 1, Len: 222
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Accept: */*\r\n
  Accept-Language: zh-tw\r\n
  Accept-Encoding: gzip, deflate\r\n
  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1; SV1; NET CLR 1.1.4322)\r\n
  Host: www.kimo.com.tw\r\n
  Connection: keep-alive\r\n
  \r\n

```

图1-4 数据包的内容

图1-5是TCP层的数据内容，我们称为TCP包头，从TCP包头中可以看到两个很重要的内容：“Source port: 1857(1857)”及“Destination port: http(80)”，这两个字段^②表示发送端计算机使用TCP端口1857连接到目的计算机的TCP端口80。

```

Frame 6 (276 bytes on wire (276 bytes captured))
Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:11:22:33:44:55
Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 202.43.195.13 (202.43.195.13)
Transmission Control Protocol, Src Port: 1857 (1857), Dst Port: http (80), Seq: 1, Ack: 1, Len: 222
  Source port: 1857 (1857)
  Destination port: http (80) ②
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 223 (relative sequence number)]
  Acknowledgement number: 1 (relative ack number)
  Header length: 20 bytes
  Flags: 0x0018 (PSH, ACK)
  Window size: 65535
  Checksum: 0x5a3a (correct)
Hypertext Transfer Protocol

```

图1-5 TCP包头的内容

在选择好数据传输方法后，接下来需要对数据包进行定位操作，而这个定位操作在IP层中完成，请参见图1-6。图1-6是IP包头的数据内容，其中最重要的是“Source: 192.168.2.10(192.168.2.10)”及“Destination: 202.43.195.13(202.43.195.13)”这两个字段^③，它们表示这个数据包是由192.168.2.10这台计算机所发送出来的，而接收端的计算机地址是202.43.195.13。有了这两个数据之后，就可以在网络上正确地传输数据包了。

```

Frame 6 (276 bytes on wire (276 bytes captured))
Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:11:22:33:44:55
Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 202.43.195.13 (202.43.195.13)
  version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 262
  Identification: 0x7d53 (32083)
  Flags: 0x04 (Don't Fragment)
  Fragment offset: 0
  Time to live: 128
  Protocol: TCP (0x06)
  Header checksum: 0x2cb3 (correct)
  Source: 192.168.2.10 (192.168.2.10)
  Destination: 202.43.195.13 (202.43.195.13) ③
Transmission Control Protocol, Src Port: 1857 (1857), Dst Port: http (80), Seq: 1, Ack: 1, Len: 222
Hypertext Transfer Protocol

```

图1-6 IP包头的内容

不过，这样就可以通过网络传输数据包了吗？当然是不行的。请考虑一个问题：IP是一个逻辑概念，而数据在网络上是以光电信号来传输的，因此，我们必须在数据包上标明“实体层”的地址(在以太网实体层的地址是以太网网卡地址——Media Access Control Address，

MAC), 请参见图1-7。在网络层处理完数据包后, 即将数据包送往实体层并加上MAC地址, 图中“Destination: 00:11:22:33:44:55”以及“Source: 00:0f:b0:93:b2:7d”即为MAC地址④, 它们分别代表该数据包是由哪一个网卡发送出来的, 以及要发送给哪一个网卡。有了这些完整的机制之后, 就可以在网络上正常传输数据包了。

No.	Time	Source	Destination	Protocol	Info
5	2.990013	192.168.2.10	202.43.195.13	TCP	1857 > http [ACK] Seq=1 Ack=1
6	2.990255	192.168.2.10	202.43.195.13	HTTP	GET / HTTP/1.1
7	3.011406	202.43.195.13	192.168.2.10	HTTP	HTTP/1.1 200 Found (text/html)

Frame 6 (276 bytes on wire, 276 bytes captured)					
Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:11:22:33:44:55					
Destination: 00:11:22:33:44:55 (00:11:22:33:44:55) ④					
Source: 00:0f:b0:93:b2:7d (00:0f:b0:93:b2:7d)					
Type: IP (0x0800)					
Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 202.43.195.13 (202.43.195.13)					
Transmission Control Protocol, Src Port: 1857 (1857), Dst Port: http (80), Seq: 1, Ack: 1, Len: 222					
Hypertext Transfer Protocol					

图1-7 以太网数据包包头的内容

在了解了以上各层包头的的数据内容后, 你是否对整个TCP/IP网络模型有了更进一步的了解呢? 或许你已经发现了一个不合理之处, 例如, 当我们运行ping命令去测试网络上一台计算机时, ping的明明是IP而不是MAC地址, 为什么本地计算机会知道对方计算机IP与MAC地址的对应关系呢? 如果不知道对方的MAC地址, 而实体网络又是以太网, 那么网络不就不能连通了吗? 没错! 在以太网上, 如果没有对方计算机的MAC地址, 确实是无法连通的, 可以使用什么机制来解决这个问题呢?

ARP通信协议

IP地址和MAC地址(MAC是一组烧录在网卡上的号码, 而且MAC地址理论上是不会重复的)都是网络上唯一的标识, 在Linux系统中, 可以使用ifconfig eth0命令来查看IP地址及MAC地址, 请参考示例1.2。

示例1.2 查看网卡卡号

```
[root@mail ~]# ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 00:14:85:35:50:EA
inet addr:192.168.1.254  Bcast: 192.168.1.255  Mask:255.255.255.0
inet6 addr: fe80::214:85ff:fe35:50ea/64 Scope:Link
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:2494776 errors:0 dropped:0 overruns:0 frame:0
TX packets:2657916 errors:0 dropped:0 overruns:0 carrier:0
   collisions:0 txqueuelen:1000
RX bytes:1302023628 (1.2 GiB)  TX bytes:1746790991 (1.6 GiB)
Interrupt:177 Base address:0x2000

[root@mail ~]#
```


或许你会认为既然有了MAC地址，为什么还需要IP地址呢？这是因为实体层网络技术众多，且每种技术都有其存在的价值。例如，在企业内我们会选用以太网作为实体层传输数据的介质，若是在两栋距离约一、两百米的建筑物之间，则可选用光纤作为传输介质，如果要将台北和高雄分公司的网络连接起来，那就会选用Frame Relay(帧中继)。MAC地址是以太网的地址形式，在其他网络实体层技术中都会有其特有的地址形式，所以请注意：“MAC地址只在单一以太网上有效，因此MAC地址是不会跨越路由器设备的”，也因此，当初TCP/IP网络模型并没有定义实体层技术一定要使用哪一种。

了解这些概念后，让我们来看看传输数据的两台主机是如何获取到对方网卡上的MAC地址的。首先启动两台Linux主机，并在其中一台主机打开Wireshark数据包截取软件，接着使用ping命令测试另一台主机，即可截获到如图1-8的内容。以第三个数据包为例，该数据包是由10.0.1.203主机发送出来的，目的端主机是10.0.1.200，Protocol标识为ICMP协议，而数据栏内显示此为一个ICMP的请求包，也就是10.0.1.203主机去ping 10.0.1.200这台主机。接着，第四个数据包就是10.0.1.200主机应答给10.0.1.203主机的ICMP数据包。我们因此可以推断，10.0.1.203这台主机共发送了四次ICMP请求包给10.0.1.200主机。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.1.203	Broadcast	ARP	who has 10.0.1.200? Tell 10.0.1.203
2	0.001271	10.0.1.200	10.0.1.203	ARP	10.0.1.200 is at 00:20:ed:6d:68:19
3	0.001282	10.0.1.203	10.0.1.200	ICMP	Echo (ping) request
4	0.002312	10.0.1.200	10.0.1.203	ICMP	Echo (ping) reply
5	0.999948	10.0.1.203	10.0.1.200	ICMP	Echo (ping) request
6	1.001087	10.0.1.200	10.0.1.203	ICMP	Echo (ping) reply
7	1.999930	10.0.1.203	10.0.1.200	ICMP	Echo (ping) request
8	2.002012	10.0.1.200	10.0.1.203	ICMP	Echo (ping) reply
9	2.999971	10.0.1.203	10.0.1.200	ICMP	Echo (ping) request
10	3.001100	10.0.1.200	10.0.1.203	ICMP	Echo (ping) reply

图1-8 Wireshark截取ARP数据包

你或许已经发现，为什么在ICMP数据包发送之前会出现两个ARP协议数据包呢？没错，这就是我们要讨论的关键，因为当两台主机第一次交换数据时，由于彼此都不知道对方的MAC地址是什么，只有发送端主机知道目的端主机的IP是多少，因此，发送端会在网络上发出一个ARP的请求数据包，请参考图1-9。从图1-9中可以看出，ARP的请求数据包是“广播”包(因为Dst MAC为ff:ff:ff:ff:ff:ff)，因此，整个网络上的主机都会收到该数据包。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.1.203	Broadcast	ARP	who has 10.0.1.200? Tell 10.0.1.203
2	0.001271	10.0.1.200	10.0.1.203	ARP	10.0.1.200 is at 00:20:ed:6d:68:19
* Frame 1 (42 bytes on wire, 42 bytes captured)					
* Ethernet II, Src: 00:14:a4:32:65:12, Dst: ff:ff:ff:ff:ff:ff					
* Address Resolution Protocol (request)					
Hardware type: Ethernet (0x0001)					
Protocol type: IP (0x0800)					
Hardware size: 6					
Protocol size: 4					
Opcode: request (0x0001)					
Sender MAC address: 00:14:a4:32:65:12 (10.0.1.203)					
Sender IP address: 10.0.1.203 (10.0.1.203)					
Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)					
Target IP address: 10.0.1.200 (10.0.1.200)					

图1-9 RP请求数据包的内容

接着来分析这个ARP请求数据包的内容。由于接收端的MAC Address是未知的，因此Target MAC Address内容为00:00:00:00:00:00，但接收端的IP是已知的，所以Target IP Address内容为10.0.1.200，而发送端的信息都是已知的，因此该数据包的Send MAC Address内容为00:14:a4:32:65:12，Send IP Address内容为10.0.1.203。该数据包是属于广播包，因此网络上的所有主机都会收到该数据包，但只有10.0.1.200这台主机会处理这个数据包，因为该数据包内容中标明接收端IP应该是10.0.1.200，如此就等于告诉10.0.1.200这台主机，发送端的IP为10.0.1.203，且MAC地址为00:14:a4:32:65:12。

在接收端成功取得发送端的MAC地址及IP地址对应的关系之后，接收端随即会给发送端返回一个ARP的应答数据包，请参见图1-10。我们从图1-10中看到该数据包的Dst不再是广播数据包了，因为该字段中清楚注明这个数据包是要给哪台主机的，而数据包的内容也清楚记录着接收端及发送端所有IP地址及MAC地址的对应状态，ARP数据包如此一来一往，即可交换两台主机的MAC地址信息。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.1.203	Broadcast	ARP	who has 10.0.1.200? Tell 10.0.1.203
2	0.001271	10.0.1.200	10.0.1.203	ARP	10.0.1.200 is at 00:20:ed:6d:68:19
# Frame 2 (60 bytes on-wire, 60 bytes captured)					
# Ethernet II, Src: 00:20:ed:6d:68:19, Dst: 00:14:a4:32:65:12					
# Address Resolution Protocol (reply)					
Hardware type: Ethernet (0x0001)					
Protocol type: IP (0x0800)					
Hardware size: 6					
Protocol size: 4					
Opcode: reply (0x0002)					
Sender MAC address: 00:20:ed:6d:68:19 (10.0.1.200)					
Sender IP address: 10.0.1.200 (10.0.1.200)					
Target MAC address: 00:14:a4:32:65:12 (10.0.1.203)					
Target IP address: 10.0.1.203 (10.0.1.203)					

图1-10 ARP应答数据包的内容

1.3 TCP、UDP及Socket的关系

了解数据在网络中的传输方式之后，接着来研究传输层中的另一个重要标记——端口(Port)。端口在传输层中是一个很重要的概念，我们之所以能够让一台主机同时运行多个服务，都要归功于端口这个概念。在开始讨论端口之前，我们来了解一下什么是套接字(Socket)。

Socket是指一个上面有很多“洞”的东西，比方说，计算机主板上CPU的插座、我们称之为Socket 478或Socket 939等，而Socket上面的这些洞，则称为端口。在OS的网络系统中会有两个Socket，分别是TCP Socket及UDP Socket，Socket上各有65 536个洞，由0开始算起，其范围为0~65 535，我们称其为Port 0~Port 65 535。

端口的作用是什么呢？以图1-11为例，图的左边是客户端，右边是服务器端，我们假设服务器端主机上运行了Web、SSH及DNS三项服务。在TCP/IP的网络规范中，当一个网

络应用程序运行起来时,都会占用一个端口,如服务器端的Web服务启动时,即会占用TCP Socket中的端口80,我们简称为TCP Port 80,而SSH服务是TCP Port 22、DNS服务则是UDP Port 53,因此,我们可以说TCP Port 80代表Web服务这一个程序,TCP Port 22代表SSH服务这个程序,而UDP Port 53代表DNS服务这个程序。

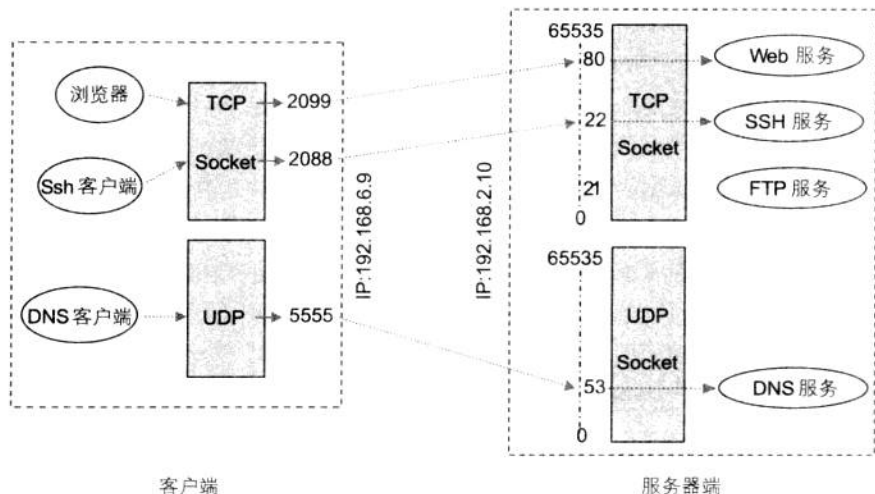


图1-11 TCP、UDP及端口的关系

另外在客户端部分,我们假设客户端启动了浏览器、SSH客户端及DNS客户端这三个网络应用程序。同样在TCP/IP的规范中,当客户端的网络应用程序启动时,也会占用某一个端口。如图1-11所示,我们假设浏览器使用TCP Port 2099、SSH客户端使用TCP Port 2088,DNS客户端则使用UDP Port 5555。因此,我们也可以说TCP Port 2099代表浏览器这一个应用程序,TCP Port 2088代表SSH 浏览器这个应用程序,而UDP Port 5555则代表着DNS客户端这个应用程序。

最后,让我们来看看端口存在的重要性,假设客户端的浏览器想向服务器端的Web服务传输数据,那么客户端会生成一个数据包并发送到服务器端,我们假设客户端的IP为10.0.1.219、服务器端的IP为202.43.195.52,该数据包的内容就如图1-12所示。首先,数据包内必然会记录包的“来源”及“目的”。如图中❶标记的部分就是数据包的来源与目的地址,但这两个数据只能表明来源端及目的端,并无法说明该数据包要发送到服务器端的什么地方,不过,TCP包头内的信息将会标明包的去处。在❷所标记的部分即标明了数据包的去处,因此,当服务器端收到这个数据包后,系统就会依据这些信息把这个数据包交给TCP Port 80来处理,而TCP Port 80所代表的就是Web服务这个程序。

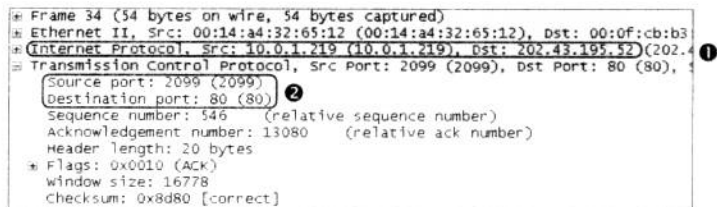


图1-12 客户端发送到服务器端的数据包内容

当服务器端的Web服务程序收到这个数据包后，必然会应答一个数据包给客户端，但是Web服务怎么会知道该把这个数据包回送给哪部主机呢？又该送给该主机的哪个应用程序呢？别忘了，当客户端发送数据包给服务器端时，该数据包内已经包含了客户端的IP地址①，以及该数据包是由客户端主机的哪个应用程序②发送出来的，因此，服务器端即可根据这些信息将数据包回送到正确位置。如图1-13即是服务器端回送给客户端的数据包，从图中标注③及标注④部分可以看到，服务器端把数据包回送给客户端的TCP Port 2099，通过这个例子，相信你已经了解Socket及端口的重要性。

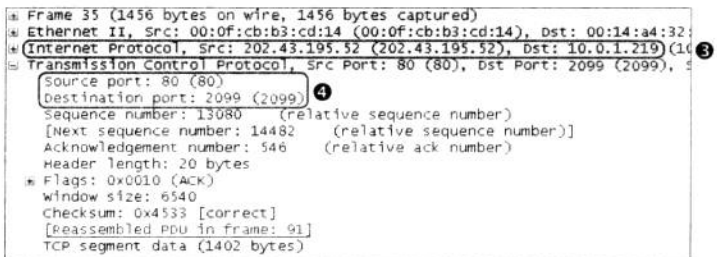


图1-13 服务器端应答给客户端的数据包内容

端口的分类

在了解端口的用途之后，我们必须了解这多达65 536个端口是如何被分配使用的。其实不管是TCP端口或是UDP端口的使用，在RFC 1700文件中都有完整的规范，我们可以到IANA的网站(<http://www.iana.org/assignments/port-numbers>)中获取端口的使用列表，或是从Linux系统中的/etc/services文件中取得这些信息。下面整理了这些端口：

- 公认的端口：0~1023
- 注册的端口：1024~49151
- 动态端口：49152~65535

1. 公认的端口

公认的端口一般都应用在“服务”上，例如，Web服务在规范中会使用TCP Port 80，

因为它是一个为人熟知的“规范”，因此当客户端要到Web服务器上访问Web服务时，客户端默认就会将数据包发送到Web服务器的TCP Port 80。也因为有规范的存在，所以当我们要去访问远程的某个服务时，就不必特别打电话去询问：“你们家的Web服务是使用哪个端口？”，换句话说，每个服务都会有一个固定的端口，而这些公认端口的范围是0~1023。

2. 注册的端口

由于在同一时间内，同一端口只能供一个程序使用，因此在程序设计师编写网络应用程序时，就得特别注意自己编写的程序所使用的端口是否已经有其他的应用程序在使用，但问题是程序设计师如何得知该端口是否已经被其他应用程序占用了呢？所幸IANA这个组织制定了一个规范，规定了需要用到固定端口的网络应用程序。为了能避免不必要的端口冲突，在设置端口之前，可先到IANA的网站上，选用一个没有其他应用程序在使用的端口，并可以将其注册下来，以告诉其他程序设计师，某个端口已经被你选用并已经注册。通过IANA这个机制，即可避免端口冲突问题，而这些可以被使用者注册的端口范围是1024~49151。

3. 动态端口

动态端口通常供临时使用。例如，几乎所有客户端应用程序都会使用动态端口，当客户端应用程序启动时，系统就会分配一个“动态端口”给该应用程序来使用，当该应用程序结束时，即会将其占用的端口归还给系统，而这些动态端口的范围是49152~65535。

请注意，以上所讲的是“规范”，但并非所有程序设计师都会按章办事。例如，Linux系统上的SSH客户端使用的端口通常都会在“注册端口”范围内，其实这并不会对系统造成太大问题，原则上，“同一个端口在同一时间内不得有多个应用程序同时使用”，只要把握这个原则，基本上就不会出问题。

1.4 何谓防火墙

没有防火墙的网络环境就如图1-14所示。



图1-14 没有防火墙的网络环境

当主机A发送数据包给主机B时，不管该数据包的内容是什么，主机B都要照单全收。例如，主机A对主机B发送服务请求数据包，或者主机A对主机B发送攻击性的数据包，主机B都需要将其收下。当然，如果主机B本身的服务系统在设计时考虑是很周全的，那么不管收到的是什么样的数据包，应该都不会对系统造成任何问题，但如果主机B的服务系统本身设计存在缺陷，那么这个攻击数据包可能使得整台计算机瞬间瘫痪。

在早期的Windows 98中，由于其NetBIOS机制在设计上有所不同，而形成了一个让黑客足以在瞬间瘫痪掉整台计算机的致命漏洞，当时有黑客把攻击这个弱点的步骤直接写成一个可执行程序。图1-15即为该程序运行时的界面，只要在界面中“Target hostname/IP address:”的位置，输入想要攻击对象的IP地址即可，你根本不需要掌握任何高深的网络知识，而收到这个数据包的Windows 98主机会立刻变成“蓝屏”，然后整台计算机的网络系统瞬间瘫痪掉。被攻击的主机只有重新开机，才能恢复原有的功能。



图1-15 WinNUKE

一个有防火墙的网络如图1-16所示。

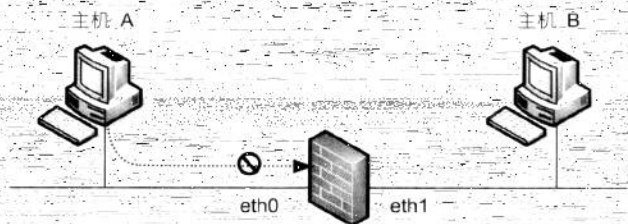


图1-16 有防火墙的网络环境

从某个角度看，在这个架构中的防火墙就像一个路由器，只是这个路由器(Router)可以选择哪些数据包可以将其转发到另一边，而哪些数据包不需要执行转发的操作，如此即可达到保护防火墙后面主机的目的。但问题在于路由器是如何判断哪些数据包是需要转发的，而哪些数据包是不需要或者不能将其转发的？关键就在于防火墙上规则(Rule)的定义了，由于尚未介绍防火墙规则语法，因此先以口语化的方式来描述防火墙规则的大致定义方式，下面列举两个简单例子。

示例1.3 在防火墙上指定：主机A“不得”访问主机B上的任何服务

如果数据包由eth0进入，而且数据包的来源端IP等于主机A的IP，目的端IP等于主机B的IP，就把该数据包丢弃掉。

示例1.4 在防火墙上指定：主机A“不得”访问主机B上的SSH服务

如果数据包由eth0进入，而且数据包的来源端IP等于主机A的IP，目的端IP等于主机B的IP，数据包的目的端口等于22，就把该数据包丢弃掉。

1.5 防火walls的判断依据

防火墙的任务简单描述就是“放行合法”或者“封锁不合法”的数据包，第1.4一节中大致介绍了合法与不合法的定义，接下来将解释哪些数据可以拿来作为“匹配”条件。事实上，可以拿来作为匹配条件的数据相当多，但由于有些数据较为复杂，因此并不建议一下子就去了解或牢记所有的匹配条件。本书将随着章节的难易程度，逐一介绍可用来匹配的条件；首先将条件划分为以下三大类。

1.5.1 各层数据包包头内的信息

在防火墙的匹配条件中，最基本且最简单的就是数据包各层包头里面的信息，这里将根据各层不同包头内的信息来说明。

1. 链路层

图1-17中是数据包中链路层包头的内容，在这些内容中最重要的莫过于MAC地址，在防火墙的过滤规则中，我们可以通过Source MAC来判断数据包是由哪部主机送出来的。

No.	Time	Source	Destination	Protocol	Info
5	0.990013	192.168.2.10	202.43.195.13	TCP	1857 > http [ACK] Seq=1 Ack=1
6	0.990015	192.168.2.10	202.43.195.13	HTTP	GET / HTTP/1.1
7	0.011066	202.43.195.13	192.168.2.10	HTTP	HTTP/1.1 302 Found (text/html)
Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:11:22:33:44:55					
Destination: 00:11:22:33:44:55 (00:11:22:33:44:55)					
Source: 00:0f:b0:93:b2:7d (00:0f:b0:93:b2:7d)					
Type: IP (0x0800)					
Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 202.43.195.13					
Transmission Control Protocol, Src Port: 1857 (1857), Dst Port: http (80) Seq: 1, Ack: 1, Len: 222					
Hypertext Transfer Protocol					

图1-17 链路层数据包包头内容



提示

我们可以将公司内所有登记有案的网卡的MAC地址，填写在企业对外的防火墙上，这样，非本公司的计算机将无法经由本公司的网络连接因特网。

2. 网络层

如图1-18即为数据包中IP包头的内容，这个包头中的很多数据可以拿来作为过滤条件。

如表1-1所示。

表1-1 IP包头中的内容解释

字段	说明
Header Length	IP包头的长度
Differentiated服务	差别服务判断, 执行QOS时可能会使用到, 一般正常值应为0
Total Length	数据包不包含链路层包头的整体长度
Flags	标明该数据包是否被分割, 或者设置数据包可不可以被分割
Time to Live	数据包的存活时间
Protocol	上层协议, 如果TCP为06、UDP为17、ICMP为01, 其他的上层协议请参阅Linux系统的/etc/protocol文件
Source	来源端主机的IP地址
Destination	目的端主机的IP地址

```

# Frame 6 (276 bytes on wire; 276 bytes captured)
# Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:11:22:33:44:55
# Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 202.43.195.13 (202.43.195.13)
  version: 4
  header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP: 0x00; Default; ECN: 0x00)
  Total Length: 262
  Identification: 0x7d53 (32083)
  Flags: 0x04 (Don't Fragment)
  Fragment offset: 0
  Time to live: 128
  Protocol: TCP (0x06)
  Header checksum: 0x2cb3 (correct)
  Source: 192.168.2.10 (192.168.2.10)
  Destination: 202.43.195.13 (202.43.195.13)
# Transmission Control Protocol, Src Port: 1857 (1857), Dst Port: http (80), Seq: 1, Ack: 1, Len: 222
# Hypertext Transfer Protocol
  
```

图1-18 IP层数据包包头内容



提示

如果我们不希望有因特网上的主机来ping公司的Web服务器, 我们可通过IP包头中的Protocol字段来判断送到Web服务器的数据包是否为ICMP包。

3. 网络传输层

图1-19即为传输层包头, 在此以TCP协议为例进行说明。更准确一点讲, 图1-19是一个TCP包头的内容, 在这个包头中也有相当多的信息可作为防火墙过滤的条件, 如表1-2所示。

```

# Frame 6 (276 bytes on wire; 276 bytes captured)
# Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:11:22:33:44:55
# Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 202.43.195.13 (202.43.195.13)
# Transmission Control Protocol, Src Port: 1857 (1857), Dst Port: http (80), Seq: 1, Ack: 1, Len: 222
  Source port: 1857 (1857)
  Destination port: http (80)
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 223 (relative sequence number)]
  Acknowledgement number: 1 (relative ack number)
  Header length: 20 bytes
  Flags: 0x0013 (PSH, ACK)
  Window size: 65535
  Checksum: 0x5a2a (correct)
# Hypertext Transfer Protocol
  
```

图1-19 TCP数据包包头内容

表1-2 TCP数据包包头的内容解释

字段	说明
Source Port	来源端应用程序所使用的端口号
Destination Port	目的端应用程序所使用的端口号
Header Length	TCP包头的整体长度
Flags	TCP包头内的连接控制标志，请务必参阅相关的TCP/IP书籍，这是TCP包头中很重要的信息



提示

如果我们只允许因特网上的主机访问Web服务器的TCP端口80，可将TCP包头的Destination Port作为防火墙上过滤条件。

1.5.2 数据包所承载的数据内容

图1-20是一个HTTP协议应用的数据包，从图中我们可以清楚看到，客户端想要访问的对象是www.kimo.com.tw这台主机，可以在防火墙上使用“Host: www.kimo.com.tw”这个字符串作为过滤条件。

```

* Frame 6 (276 bytes on wire, 276 bytes captured)
* Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:11:22:33:44:55
* Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 202.43.195.13 (202.43.195.13)
* Transmission Control Protocol, Src Port: 1857 (1857), Dst Port: http (80), Seq: 1, Ack: 1, Len: 222
* Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Accept: */*\r\n
  Accept-Language: zh-tw\r\n
  Accept-Encoding: gzip, deflate\r\n
  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; windows NT 5.1; SV1; .NET CLR 1.1.4322)\r\n
  Host: www.kimo.com.tw\r\n
  connection: keep-alive\r\n
  \r\n

```

图1-20 应用层数据包的内容



提示

如果我们不允许企业内的客户端计算机连接到因特网上的*.playboy.com任何主机，可以在防火墙上检查所有DNS的名称解析包中是否包含“playboy.com”这个字符串。如果有，就将这个数据包丢弃，如此就可以让*.playboy.com主机名称解析失败，从而限制客户端计算机连接到*.playboy.com的任何主机。

1.5.3 连接状态

以图1-21为例，我们假设希望达到的目的是：“不允许网络上任何连接进入到企业内部，但允许所有企业内的计算机可以连接到因特网”。这样的设定确实能够阻止来自因特网的网络攻击威胁，但当企业内部计算机连接上因特网的时候，由于“企业内部”→“因特网”是允许

的，因此，由“企业内部”发送到“因特网”的数据包是可以成功穿过防火墙的；但是当因特网主机应答企业内部的主机时，该数据包变成了由“因特网”进入“企业内部”的数据包，但因为我们在防火墙上的规则是“因特网不允许有数据包送入企业内部”，因此，该数据包一定会被丢弃掉，如此会造成企业内部的使用者也无法访问因特网的困境。

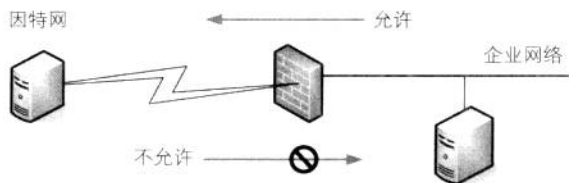


图1-21 连接状态判断

处理以上问题的方法其实很简单，一般防火墙都会提供所谓的“连接状态判断”机制，而连接状态判断是一个有点难度但又不会太复杂的操作。这里列举一个简单例子，详细的内容将在稍后章节中说明。假设企业内部计算机的IP是192.168.0.100，并且客户端计算机上运行的是IE应用程序，其所占用的端口是TCP Port 3365，而因特网上主机IP是192.168.6.2，并且其Web服务所使用的端口是TCP Port 80，这样，当客户端对Web服务发出服务请求时，该数据包的结构将是192.168.0.100:3365→192.168.6.2:80，也就是192.168.0.100使用TCP Port 3365连接到192.168.6.2的TCP Port 80。

试问，在以上情况下，如果有一个结构为192.168.6.2:80→192.168.0.100:3365的数据包出现在防火墙上时，你觉得该数据包是否应该放行，是否可以成功进入到企业内部呢？答案显然是肯定的，因为如果外送的数据包为192.168.0.100:3365→192.168.6.2:80，那么应答的数据包一定是192.168.6.2:80→192.168.0.100:3365，这就是连接状态判断的原理之一。

1.6 防火墙的分类

可以根据过滤技术将防火墙分为两类：其一是数据包过滤(Packet Level Filter)防火墙；其二是应用层(Application Level Filter)防火墙。这两种防火墙在应用上各有优缺点，适用的场景及范围各有不同，下面将说明这两种防火墙的差异及优缺点。

1.6.1 数据包过滤防火墙

以图1-22为例，我们假设这台计算机上共装有两块网卡，并让其扮演路由器的角色。因此，当一个数据包从左边接口进入后，即由路由表引导到右边接口送出，接着，我们在其上

运行数据包过滤防火墙的操作,而这个防火墙的位置位于路由表之后。从这个结构我们大概可以知道数据包过滤防火墙每一次执行检查的最小单位是“一个数据包”。

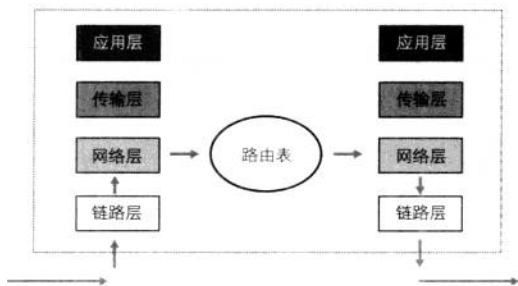


图1-22 数据包过滤防火墙

数据包过滤防火墙的优缺点如下:

- 优点: 由于数据包过滤防火墙的检查范围是一个数据包,因此,当一个数据包从左边接口送入后,在防火墙检查其无误之后,该数据包即可从右边接口送出,当然在数据包从右边接口送出之后,就再也与防火墙无关了,因此,数据包过滤防火墙对“内存”及“CPU性能”的要求较低。通常200人以下的企业大约只需要一台Pentium III 450及128M的内存就够了,由此可知,数据包过滤防火墙的成本较低。

另一个优点是,数据包过滤防火墙其本身以路由器的形式在网络上工作,因此,数据包过滤防火墙完全与“协议”无关,所以应用范围较广。

- 缺点: 因为数据包过滤防火墙的检查范围只有一个数据包,因此,数据包过滤防火墙无法对连接中的数据进行更精准的过滤操作。比如说,我们无法使用数据包过滤防火墙来检查一封电子邮件中是否带有计算机病毒。

1.6.2 应用层防火墙

数据包过滤防火墙是以路由方式将数据包转发到另一方,而应用层防火墙就不是这样了,应用层防火墙不需要包含路由机制。以图1-23为例,我们假设现在有一个SMTP协议的连接要穿过防火墙,当然这个连接上所有数据还是以一个个的数据包为单位的,当第一个数据包进入到防火墙主机后,该数据包随即被一层一层地往上传递,最后这个数据包被应用层防火墙上的应用程序过滤进程(Application Filter Process)收到;第二个数据包进入防火墙主机之后,也是如此。其实该连接中的所有数据包都会被应用程序过滤进程收到。也就是说,应用程序过滤进程将会收到该连接的所有数据包,因此,应用程序过滤进程就可以把这些数据

包重新还原成一封完整的电子邮件。既然是一封完整的电子邮件，应用层防火墙就能检查该电子邮件内容中的每一个字节，甚至是电子邮件所携带的附件，当检查无误后，应用程序过滤进程再将这封电子邮件发送到它原来的目的地。

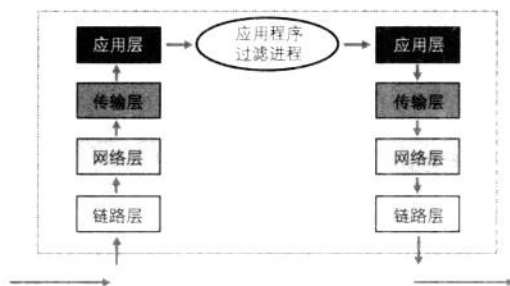


图1-23 应用层防火墙

应用层防火墙的优缺点如下：

- 优点：因为应用层防火墙能够检查任何一个连接中的任何字节，因此应用层防火墙能够进行比数据包过滤防火墙更精确的过滤操作。
- 缺点：应用层防火墙必须将所有数据包都保存下来，并将其还原成一条完整数据内容，因此应用层防火墙一定与“协议”有关。也就是说，无法处理应用程序过滤进程不支持的通信协议，因此，应用层防火墙的使用范围较小。

此外，由于应用层防火墙必须执行数据包的存储和还原操作，因此应用层防火墙需要更多内存，对CPU性能的要求也很高，导致成本较高。

1.7 常见的防火墙结构

常见的防火墙结构大概可分为单机防火墙、网关式防火墙以及透明防火墙三种，而不同的防火墙结构都有其特定的用途。接下来将介绍各种不同防火墙结构的特点及优缺点，以便在你的网络环境中，选择一种合适的防火墙结构。当然选择正确的防火墙结构也会直接影响整个网络环境的安全级别，这是一个不可轻视的重要主题。

1.7.1 单机防火墙

近年来由于宽带网络服务(ADSL)的盛行，出现了所谓的单机型、多机型及网络型服务，而这些服务的最大差异在于ISP提供给企业IP的数量。所谓单机型就是指客户只能有一个公网IP(Public IP)，而多机型可以有3个公网IP，网络型可以有8个或者16个公网IP，当然你

所选择的公网IP越多，价格也就越高。相对于以往的数据专线，ADSL所需的费用其实是低的，因此，很多中小型企业都改用ADSL作为企业的上网方式。

如果企业选择ADSL上网，其网络结构通常如图1-24所示。第一个公网IP通常会分配给网络地址转换器(Network Address Translation, NAT)来使用，而整个企业的内部网络就隐藏在NAT之后，另外两个公网IP可能就直接分配给企业对外的服务主机使用。

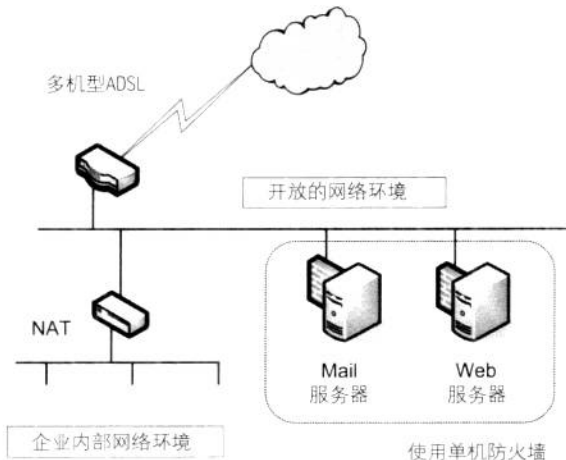


图1-24 单机防火墙

这个结构最大的好处是成本低廉，缺点则是企业对外的服务主机是直接挂在因特网上的，任何来自因特网的攻击行为，都由这两部主机自己承担。因此建议不要使用这样的网络结构，在网络环境必须如此时，其实也有其他方式来解决安全性的问题。我们可以使用“单机防火墙”来保护本机，所谓的单机防火墙通常是软件，而这个软件是工作在网卡的驱动程序上，凡是要进出本机的数据包，都会受到这个防火墙的监控，由此达到维护本机安全的目的。

1.7.2 网关式防火墙

顾名思义，网关式防火墙就是布置在“网关”位置的防火墙，网关式防火墙不同于单机防火墙，单机防火墙所能保护的范围只有本机而已，但网关式防火墙所能保护的范围则是整个网络，而网关式防火墙因其结构上的差异，又可分为以下三种不同的类型。

1. 网关式防火墙类型1

以图1-25为例，我们在上一个网络结构中发现了它的缺点，因此有人建议将企业对外的服务主机部署在企业的内部网络，或许你会觉得很奇怪，一般企业内的网络不都是私有IP吗？那么，对外提供服务的主机部署在私有IP的位置，因特网上的使用者能够访问得到吗？

其实这并不是什么大问题，通常在防火墙上都会伴随着一个机制，我们称其为NAT(Network Address Translation)，在NAT技术中有一项我们称为“一对一NAT”。

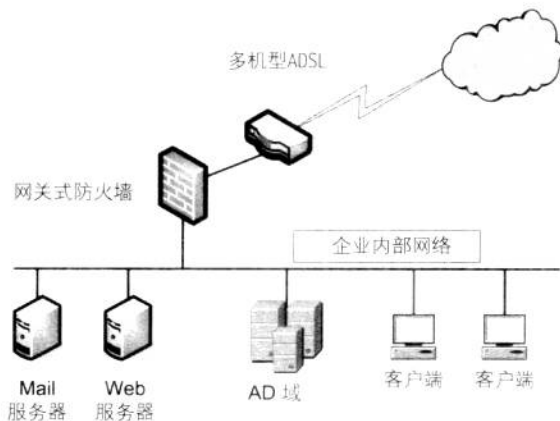


图1-25 网关式防火墙类型1

一对一NAT的工作原理如图1-26所示。我们先在防火墙对外的接口上设置3个ISP所分配的IP，而企业内部我们使用192.168.1.0/24这个私有IP网段，并且分别在Mail和Web主机上设置192.168.1.1及192.168.1.2两个私有IP，接着在防火墙上设置：“当有因特网上的使用者来访问A这个IP时，我们就将该数据包转发到192.168.1.1这个私有IP上”，然后再设置“当192.168.1.1主机送出数据包时，我们就将该数据包转发到IP A”，如此一进一出，即可实现一对一NAT的功能，这样就可以在防火墙上执行数据包过滤操作，达到保护Web和Mail主机的目的。

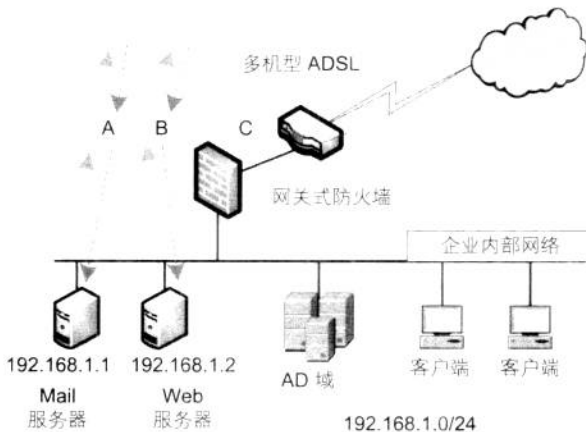


图1-26 一对一NAT的工作原理

这个架构看起来似乎不错,但也隐藏着致命的危险。以Web服务为例,因为Web主机存在的目的就是要让因特网上的使用者来访问,因此,我们在防火墙上一定会开放供因特网上所有使用者都可以访问的Web主机的端口80(端口80是Web服务),当Web服务隐含安全漏洞时,将可能变成黑客用来入侵网络的通道。多年前曾经造成很多企业网络瘫痪的Nimda病毒就是应用IIS服务的漏洞作为其传播途径。

“http://192.168.1.129/scripts/..xcl\xlc../winnt/system32/cmd.exe?/c+dir”是我从Apache Web服务的Access Log中取出的一段记录,其中的意思是客户端试图使用浏览器去运行/winnt/system32/目录下的cmd.exe执行文件,因为我使用的是Linux系统加上Apache Web服务,因此这个操作对我的Web服务当然不会起任何作用;但如果是IIS Web服务,并且IIS存在安全漏洞,这个客户端便达到了它的目的。

试想,如果今天客户端运行的是Web主机上FTP客户端的工具,那么客户端就有机会通过FTP协议将“后门程序”放进Web主机,接着再使用相同的手法,就可以在Web主机上启动这个后门程序,这样黑客就等于存在于企业网络的内部,当然防火墙就形同虚设了。由此可知,此结构是相当不明智的选择。

2. 网关式防火墙类型2

在了解了类型1的网关式防火墙的缺点,便有人将其改良成如图1-27的结构,在防火墙上再加一块网卡,因此,我们就可以多加一个网络区段,此区段我们称之为非军事区(Demilitarized Zone, DMZ),并将提供对外服务的主机部署在DMZ之中。我们想一下,如果这种结构存在类型1的安全漏洞,请问对于企业网络所造成的危害,是否比类型1低得多?因此建议无论如何,至少在防火墙主机上安装三块网卡,把因特网上的使用者可以直接访问到的主机部署在DMZ中。

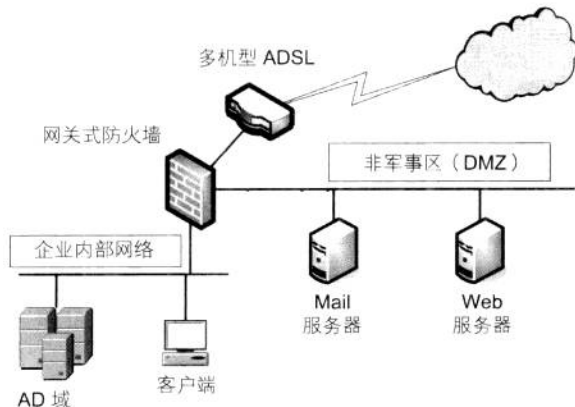


图1-27 网关式防火墙类型2

不过在此架构下,比较正规的做法是将真正提供服务的主机部署在企业内部网络中,如图1-28所示,而DMZ中仅部署反向代理服务器(反向代理)。接着,告诉因特网上的所有使用者,本公司Web服务器的IP地址是192.168.1.10。所谓反向代理服务器,其实就是一个中介程序,因此,当因特网上的使用者提出网页请求时,这个请求必定会落在反向代理服务器之上①,接着反向代理服务器再到企业内②将客户端所需要的网页取回③,最后反向代理服务器再把这些数据传回给客户端④,如此才能保证服务主机的安全性。不过这样将会造成企业内主机数量大增,因此绝大多数使用者还是会将真正的主机放在DMZ中。

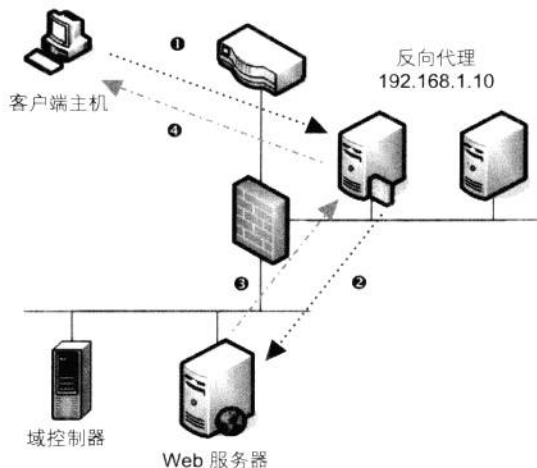


图1-28 反向代理工作原理

3. 网关式防火墙类型3

为了提升网关式防火墙的安全性,有人主张将它改良成如图1-29的架构,其实这个架构在原理上与类型2完全相同,其目的是把对外服务的主机放在一个独立的网段中,只不过这个结构使用了两台防火墙主机,因此其安全性也就相对提升。不过,在这个结构中选用防火墙时,请务必选用两家不同公司的产品,以便真正发挥出这种结构的作用。

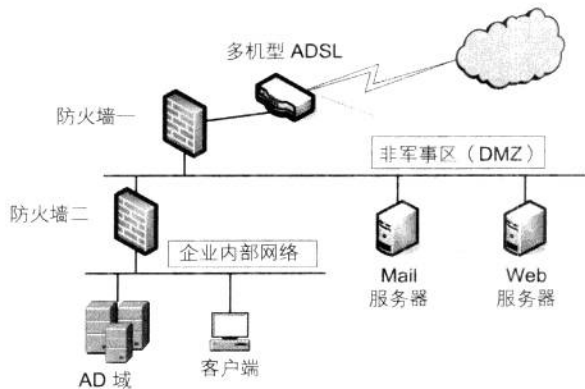


图1-29 网关式防火墙类型3

1.7.3 透明防火墙

在介绍了单机防火墙以及网关式防火墙之后，接下来讨论较新一代的防火墙，我们称其为透明防火墙(Transparent Firewall)。在网关式防火墙中，不管是哪一种类型，防火墙本身就是一个路由器，因此在部署防火墙时，我们必须仔细考虑路由问题。以图1-30为例，如果我们希望使用一台防火墙主机来保护原有的Mail及Web主机，那么Mail及Web的IP就一定要改变，如果网络环境再复杂一点，那么要改变的设置也会更复杂。

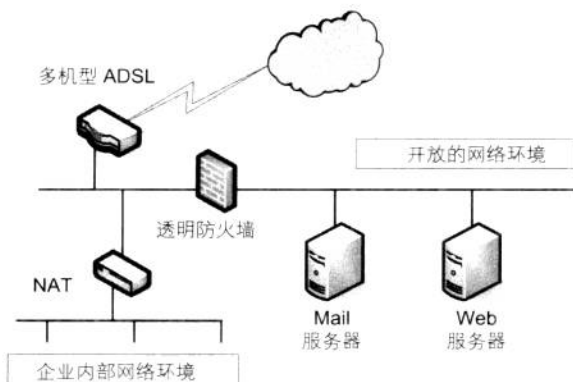
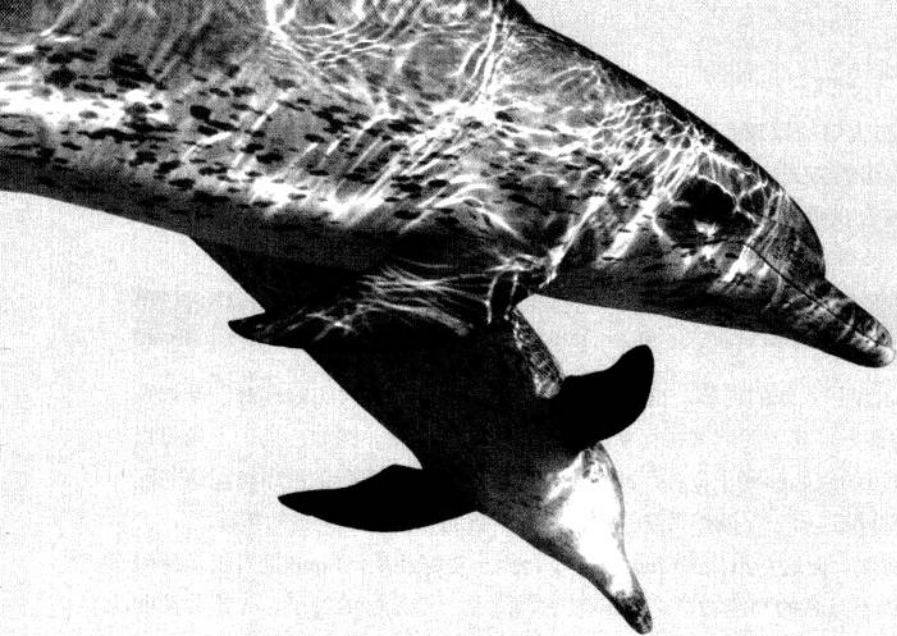


图1-30 透明防火墙类型

可以使用透明防火墙将一举解决所有这些问题。简单来说，透明防火墙就是一个网桥(Bridge)设备，并且在网桥设备上赋予了过滤器(Filter)功能。因为网桥是工作在OSI第二层的网络设备，因此不会有任何路由的问题，并且网桥上可以不需要设置任何的IP，如此就没有路由的问题。又因为防火墙主机本身无需设置IP，所以透明防火墙的部署能力相当强，隐蔽性相当高，即使黑客要攻击这个防火墙，也可能会因为没有目的端IP而无功而返。

1.8 小结

本章介绍数据在网络上传输的原理，其中包括以太网实体层的数据传输方式，同时也说明了TCP/IP的基本概念以及应用层协议的概念，虽然这些都是很简单的网络通信技术，但如果没有真正理解这些技术，那么对于后面章节的学习将会产生严重影响，因此请你务必真正了解本章的内容。



| 第2章 | Netfilter/iptables

第1章简要介绍了防火墙的原理及部署要领，本章将循序渐进地介绍Linux防火墙的核心部分，并采用深入浅出的方式完整描述Linux防火墙的四大功能，分别是：filter、nat、mangle及raw，最后再结合我所设计的实际示例进行讨论。

2.1 何谓内核

内核(Kernel)指的是操作系统的内核。内核的功能到底是什么呢？其实内核对于操作系统来说，是极为重要的部分，其主要用来执行系统资源的分配及调度。例如，当一个应用程序运行时所使用内存区段的起始和终止位置，或者在现在这个时刻哪个应用程序可以访问硬盘等，都由内核来负责分配；而一个操作系统稳定与否，内核的优劣将是关键的指标。

虽然Linux是自由软件，但这并不代表Linux上不了台面，正好相反，Linux拥有很多商用版本的UNIX操作系统尚不具备的高级功能。Linux防火墙就是一个很好的例子，无论其扩展性、稳定性还是安全性，绝对堪称业界的翘楚。Linux内置的防火墙因其发展年代的不同，会有所差异，如表2-1所示。

表2-1 内核版本与防火墙机制的对应表

内核版本	主要的防火墙机制
Kernel 2.0	ipfwadm
Kernel 2.2	ipchains
Kernel 2.4 / 2.6	Netfilter/iptables

在此要澄清一个概念，就是Linux这个名词的含义，很多人都曾说“我所使用的是Linux操作系统”，但这样的叙述并不是很正确，事实上，我们所使用的CentOS(或者Fedora Linux、Suse Linux、Red Hat Enterprise Linux等)都由两大部分共同组成。如图2-1所示，图的底部是内核，上部则是由应用程序所组合而成，只有这两个部分结合在一起，才会是一个完整的操作系统；而其中内核是由Linus Torvalds所带领的团队开发完成的(其官方网站是<http://www.kernel.org>)，而应用程序部分则由因特网上无私的程序设计师们所贡献的，而其中又以GNU这个计划贡献得最多，因此一个完整的操作系统应该包含“Linus所带领的团队”，再加上“GNU这个计划”的心血结晶而成，所以这个系统比较正确的名称应该是GNU/Linux。因此，千万别再说你所学的系统叫Linux，因为Linux所指的只是“内核”部分而已。



图2-1 Linux系统的组成

2.2 何谓Netfilter

由于Linux的高安全性、高稳定性以及高性能的特性，加上Linux的源代码完全开放，因而让Linux拥有极为广泛的使用群体，例如，航天工业、军事工业甚至IT产业都有Linux的身影。但不管Linux多么强大，也不可能完全满足每种产业的需要，所幸Linux的源代码是完全公开的，程序设计师可以依据自己的需要，在Linux内额外添加所需的功能，这样的例子比比皆是。Netfilter就是在这种模式下诞生的，也因为这样的特性，Linux才会如此快速地成长。

Netfilter可以说是Linux的第三代防火墙，在此之前，还有ipfwadm及ipchains两种防火墙。但由于Netfilter防火墙比ipfwadm及ipchains出色得多，因此Linux组织就将Netfilter作为其默认防火墙。虽然Netfilter是自由软件，但其功能、稳定性、安全性及可扩展性却丝毫不逊于商用版的防火墙，甚至Netfilter有许多高级功能是商用版防火墙所不及的。例如，Netfilter在网络层位置就可以直接检查单一数据包所承载的数据内容，而无需用到应用层防火墙。

2.3 Netfilter与Linux的关系

Netfilter与Linux是两个相互独立的组织，事实上，在Linux 2.4早期的版本中，并没有包含Netfilter的功能，是到后期的版本，Linux这个组织才把Netfilter的功能收录进来。下面通过图2-2来说明Netfilter、Linux、GNU/Linux发行组织以及使用者之间的一些关系。

从图2-2可以看到，Linux是由www.kernel.org这个组织所开发的，而Netfilter却是由www.netfilter.org组织为Linux所开发的一个新功能，但因为这个功能设计得非常完善，因此www.kernel.org这个组织每隔一段时间就会到www.netfilter.org下载最新版本的Netfilter源代码，并且将之加入到自己所开发的Linux系统中。但www.kernel.org上所存放的都只是Linux的程序源代码，而这些源代码只有经过编译，才能转换成CPU可以运行的二进制码。不过这个编译过程漫长而复杂，并不是人人都有能力去做，所幸有些GNU/Linux版本的发行组织，例如，RedHat、Suse、CentOS等组织会不定期地到www.kernel.org下载最新的Linux源代码回来，并将之制作成Linux的RPM软件。这样便可让最终用户通过RPM来管理程序，来安装最新的Linux版本，当然，最新的Netfilter也会一起被安装进来。

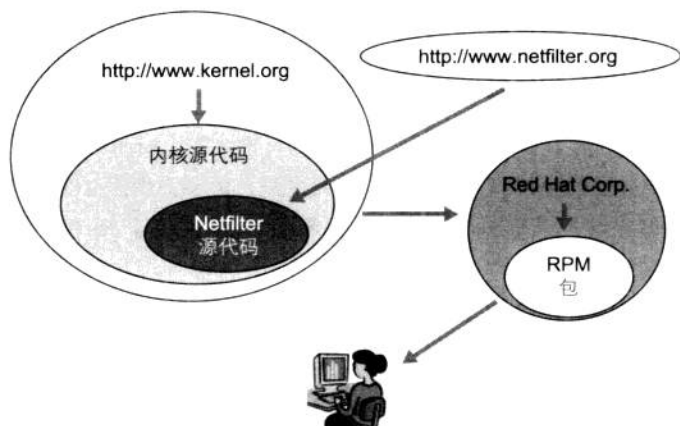


图2-2 Netfilter与内核的关系

Netfilter也是采用开源的开发模式，可以说是集因特网上各路高手之大成的产品；Netfilter的功能可以说是每天都在进步，但由于www.kernel.org不可能每天都更新其内置的Netfilter版本，再加上GNU/Linux版本的发行组织，也不可能每天制作最新的Linux RPM软件，因此最终用户也就不可能即时地享受到Netfilter最新的功能。不过，如果你有兴趣的话，不妨自己研究，或到www.kernel.org及www.netfilter.org下载最新的Linux及Netfilter程序源代码，再将其结合并重新编译Linux，就可以使用Netfilter的最新功能了。

2.4 Netfilter工作的位置

如图2-3所示，Netfilter是运行在Linux中的一个小功能，因为Linux是一个极度“模块”化的内核，所以Linux的绝大多数功能都以模块形式扩充出来，而这种模块化的设计最大的优点在于“弹性”，我们可以通过Linux的模块管理工具，随心所欲地将模块载入内存，或者将某一个目前不需要使用的模块从内存中移除，Netfilter也是以模块的形式存在于Linux中。因此我们可以理解，每当Linux多一个Netfilter的模块，就代表着Linux防火墙的功能多了一项，当然，有了更多模块，Linux防火墙的功能也就越多了。

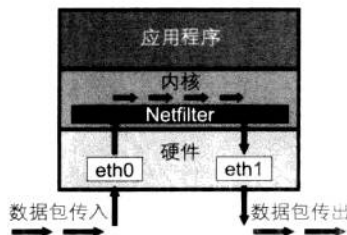


图2-3 Netfilter的工作位置

不过在CentOS 6.0(RedHat Enterprise Linux 6.0)开始有了点小改变，由于某些模块几乎一定会被用到，因此RedHat在编译内核时就直接把这些功能编译进内核里面，自然该功能就不

会有对应的模块出现。例如，在CentOS 5.x版本有一个名为xt_state.ko的模块已经不见了，但其功能还是存在的。接下来将介绍Netfilter模块存放的目录，以及这些目录之间的差异。

● /lib/modules/kernel_version/kernel/net/ipv4/netfilter/

arptable_filter.ko	ip_conntrack_tftp.ko	iptables_nat.ko
ipt_MASQUERADE.ko	arp_tables.ko	ip_nat_amanda.ko
iptables_raw.ko	ipt_NETMAP.ko	arpt_mangle.ko
ip_nat_ftp.ko	ip_tables.ko	ipt_owner.ko
ip_conntrack_amanda.ko	ip_nat_h323.ko	ipt_addrtype.ko
ipt_recent.ko	ip_conntrack_ftp.ko	ip_nat_irc.ko
ipt_ah.ko	ipt_REDIRECT.ko	ip_conntrack_h323.ko
ip_nat.ko	ipt_CLUSTERIP.ko	ipt_REJECT.ko
ip_conntrack_irc.ko	ip_nat_pptp.ko	ipt_dscp.ko
ipt_SAME.ko	ip_conntrack.ko	ip_nat_sip.ko
ipt_DSCP.ko	ipt_TCPMSS.ko	ip_conntrack_netbios_ns.ko
ip_nat_snmp_basic.ko	ipt_ecn.ko	ipt_tos.ko
ip_conntrack_netlink.ko	ip_nat_tftp.ko	ipt_ECN.ko
ipt_TOS.ko	ip_conntrack_pptp.ko	ip_queue.ko
ipt_hashlimit.ko	ipt_ttl.ko	ip_conntrack_proto_sctp.ko
iptables_filter.ko	ipt_iprange.ko	ipt_TTL.ko
ip_conntrack_sip.ko	iptables_mangle.ko	ipt_LOG.ko
ipt_ULOG.ko	等...	

● /lib/modules/kernel_version/kernel/net/ipv6/netfilter/

ip6_queue.ko	ip6table_raw.ko	ip6t_dst.ko
ip6t_hbh.ko	ip6t_ipv6header.ko	ip6t_REJECT.ko
ip6table_filter.ko	ip6_tables.ko	ip6t_eui64.ko
ip6t_hl.ko	ip6t_LOG.ko	ip6t_rt.ko
ip6table_mangle.ko	ip6t_ah.ko	ip6t_frag.ko
ip6t_HL.ko	ip6t_owner.ko	等...

/lib/modules/kernel_version/kernel/net/ipv4/netfilter/目录中存放的模块只能运行在IPV4的网络环境下；而/lib/modules/kernel_version/kernel/net/ipv6/netfilter/目录中存放的模块也只能运行在IPV6的网络环境下。以上两个目录的模块的一个共性是都与“协议”有关。因此，我们在使用这些模块时，必须特别留意哪个模块只能使用在哪个协议之下，这样的设计对于使用者来说，是非常不方便的，对程序设计师而言，要维护这么多模块其实也是挺辛苦的；在Linux 2.6.14之前的版本只有以上这两个目录。

● /lib/modules/kernel_version/kernel/net/netfilter/

nfnetlink.ko	xt_connbytes.ko	xt_esp.ko	xt_MARK.ko
xt_policy.ko	xt_statistic.ko	nfnetlink_log.ko	xt_connmark.ko
xt_helper.ko	xt_multiport.ko	xt_quota.ko	xt_string.ko
nfnetlink_queue.ko	xt_CONNMARK.ko	xt_length.ko	xt_NFQUEUE.ko
xt_realm.ko	xt_tcpmss.ko	xt_tables.ko	xt_sctp.ko

```

xt_limit.ko      xt_NOTRACK.ko      xt_tcpudp.ko      xt_CLASSIFY.ko
xt_conntrack.ko  xt_mac.ko          xt_physdev.ko     xt_SECMARK.ko
xt_comment.ko    xt_dccp.ko         xt_mark.ko        xt_pkttype.ko
xt_state.ko      xt_CONNSECMARK.ko  等

```

Linux从2.6.14版本开始,其内置的Netfilter模块在设计上有了重大改变,Netfilter组织希望模块与“协议”是无关的。也就是说,同一个模块可以同时应用于IPV4与IPV6的网络环境中。当然,这是最终的努力目标。就目前来说,已有部分模块可以达到这样的目标,而这些模块就存放在这个目录下。

2.5 Netfilter的命令结构

在了解Netfilter的结构之后,我们必须清楚,这些模块只是提供某些过滤匹配的功能而已,如果希望Netfilter能为我们做些事,就必须给予Netfilter执行“规则”,有了规则之后,Netfilter才会知道哪些数据包是可以被接受的,哪些数据包是必须丢弃的,又有哪些包是必须以特殊方式来处理的。而赋予Netfilter规则的方式,则是将规则填写到一块结构化内存中,这样Netfilter就会按照防火墙管理人员所给出的命令来运行。

图2-4即为Netfilter存放规则的内存块,从图中可以看到内存被分为四个表(Table),分别是filter、nat、mangle及raw表,这也是目前Netfilter机制所提供的四大功能。这里首先简要介绍这四个功能的用途,稍后的章节将给出完整介绍及实际应用示例。

表	链
filter	INPUT
	FORWARD
	OUTPUT
nat	PREROUTING
	POSTROUTING
	OUTPUT
mangle	PREROUTING
	INPUT
	FORWARD
raw	OUTPUT
	POSTROUTING
	PREROUTING

图2-4 Netfilter的命令结构

- **filter:** filter是Netfilter中最重要的机制,其任务是执行数据包的过滤操作,也就是起到防火墙的作用。
- **nat:** nat(Network Address Translation, NAT)也是防火墙上一个不可或缺的重要机制,比较通俗的方式来说,其功能就是IP分享器,只不过其所能执行的功能,比一般市场上的IP分享器功能强大得多。

- **mangle**: mangle是一个很特殊的机制，我们可以通过mangle机制来修改经过防火墙内数据包的内容。
- **raw**: 负责加快数据包穿过防火墙机制的速度，由此提高防火墙的性能。

在了解这四个表的用途之后，接着我们从图2-4中可以看到每个不同的表下，都有属于各自独立的几个不同的链，而这个链的空间就是我们存放“规则”的地方。当然，不同的链其功能及用途也是不一样的，接下来将根据防火墙的四大功能来分别介绍各个链的用途。

2.6 Netfilter的filter机制

filter是Netfilter中的防火墙机制，如果你想快速而且扎实地将防火墙功能学好，首先必须明确理解“数据包的分类”。以图2-5为例，假设计算机上安装有两块网卡，并在计算机上分别运行httpd及firefox两个程序。接下来将说明filter对于数据包的分类及其意义：

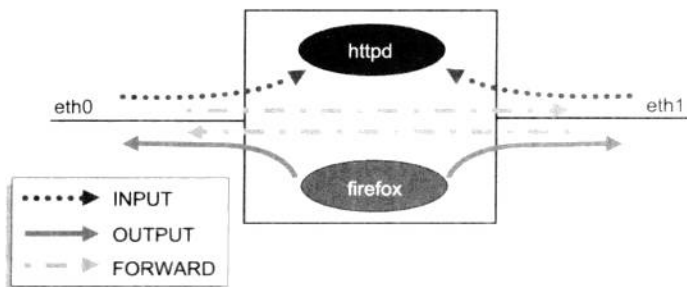


图2-5 Netfilter下的数据包分类

- **INPUT**类型：以图2-5为例，所谓INPUT类型是指“网络上其他主机发送给本机进程的数据包”，就是INPUT类型的数据包。例如，网络上其他使用者访问本机的httpd服务时，就会生成这种类型的数据包。
- **OUTPUT**类型：如果是“本机进程”所生成的数据包，即为OUTPUT类型的数据包。例如，使用者在本机启动firefox去访问网络上的其他主机时，就会生成这种类型的数据包。
- **FORWARD**类型：如果数据包对本机而言只是“路过”而已，那么这就是属于FORWARD类型的数据包。在什么情况下会有这种类型的数据包产生呢？试想，如果本机扮演的是路由器角色，是不是就会生成FORWARD类型的数据包呢？

在了解了filter对数据包分类之后,接下来以filter表的结构来讨论。图2-6就是filter表的内容,其中有三个链,其名称分别是INPUT、FORWARD及OUTPUT。

filter表		
INPUT链	FORWARD链	OUTPUT链
.....rule-1.....rule-1.....rule-1.....
.....rule-2.....rule-2.....rule-2.....
.....rule-3.....rule-3.....rule-3.....
.....rule-4.....rule-4.....rule-4.....
.....rule-5.....rule-5.....rule-5.....
.....rule-6.....	rule-6.....
	rule-7.....

图2-6 filter表结构

你或许早已发现了这三个链的名称竟与三种数据包类型的名称是一样的!难道这只是巧合?其实这是特意设计的,以下将分别说明这三个链的功能。

- INPUT链: 请以图2-5为例来思考一个问题,如果我们要保护本机的httpd这个进程,应该要注意哪种数据包呢?毫无疑问,是INPUT类型的数据包,因为只有INPUT类型的数据包才能对本机的进程造成损害,因此请将用来过滤INPUT类型数据包的“规则”填写在INPUT链之中。例如,我们可以在INPUT链中填写:“如果传入的数据包要到本机的TCP Port 80,而且这个数据包是由192.168.2.1主机送来的,就将该数据包丢弃”,如此即可达到保护httpd这个进程的目的。由以上的解释可以得知,INPUT链是用来存放过滤INPUT类型数据包的规则,也就是说,INPUT链是用于“保护”本机的机制。
- OUTPUT链: 同样以图2-5为例来思考一个问题,如果我们要限制使用者不得在本机上使用Firefox来浏览www.playboy.com网站,那么我们需要管制的应该是哪一种类型的数据包?毫无疑问的是OUTPUT类型的数据包,因此请将用来过滤OUTPUT类型数据包的“规则”填写在OUTPUT链之中。例如,我们可以在OUTPUT链中填写:“如果数据包是由本机的进程所生成的,而且数据包是要送往www.playboy.com网站的TCP Port 80,就将该数据包丢弃掉”,如此即可限制本机进程的网络访问行为。

由以上可以得知，OUTPUT链用来存放过滤OUTPUT类型数据包的规则，也就是说，OUTPUT链用于“限制”本机应用程序的网络访问。

- FORWARD链：以图2-5及图2-7为例来思考另一个问题，如果我们要以图2-7中的防火墙来保护Web服务器，应该管制的是哪类数据包？毫无疑问的应该是FORWARD类型的数据包，因此，请将用来过滤FORWARD类型数据包的“规则”填写在FORWARD链之中。例如，我们可以在FORWARD链中填写：“如果数据包是由192.168.0.10主机发送出来的，而且数据包要送往Web服务器的TCP Port 80，那就将该数据包丢弃掉”，如此可以达到以防火墙来保护Web服务器的目的。

由以上解释可以得知，FORWARD链是用来存放过滤FORWARD类型数据包的规则，也就是说，FORWARD链保护防火墙“后端”的主机。

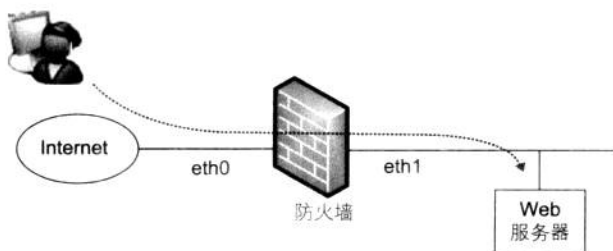


图2-7 FORWARD链的功能

前面希望以一种容易理解及记忆的方法来介绍相关内容。最终，我们还是得真正去了解Netfilter的完整结构。不过，由于Netfilter的完整结构有点复杂，因此将Netfilter的结构分成四个部分来解释，最后将四个部分结合成完整的Netfilter机制。

图2-8是filter的完整结构图，这个结构图看起来较为复杂，但实际上并没有想象中的难懂。首先，我们可以看到图中有两个路由表(Routing Table)，虽然看到的是两个路由表，但实际上是同一个，而这个路由表就是一般常用的路由表。在Linux系统下，我们可以使用route-rn命令来查看路由表的内容，而路由表的目的是决定一个数据包被传输的路径，我想这对一般使用者来说，绝对不是什么问题；另外，我们可以看到一个名为“本机进程”的东西，这是代表在本机上所运行的应用程序，再来可以看到数据包传入(Packet IN)及数据包传出(Packet OUT)的标识，这两个标识代表数据包进入及离开的位置，但请不要把这两个标识与任何一个网络接口联系在一起，因为数据包传入及数据包传出不一定会与特定的网络接口有关；最后，我们可以看到INPUT链、FORWARD链及OUTPUT链，这是filter表的三个链。

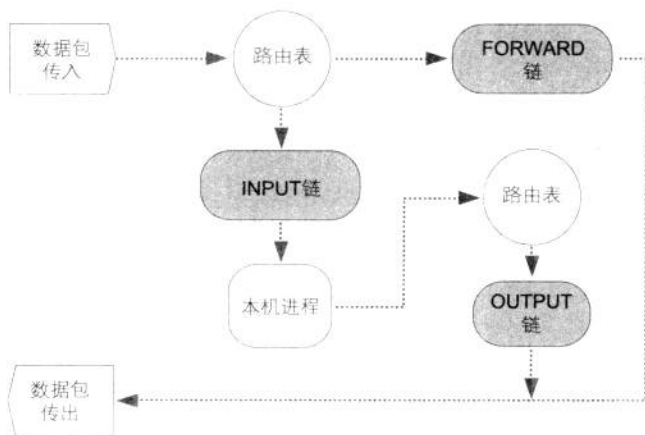


图2-8 filter机制的结构图

在了解图2-8的内容之后，接下来将分三种不同的状况来解释网络数据包在filter机制中的处理情形。

● 当网络数据包的目的端是本机进程时：

当一个数据包进入本机时，该数据包即由“数据包传入”所标识的位置进入，接着，数据包被送入路由表，并由路由表的内容来决定数据包传输的路径。本例中假设数据包是要送给本机的进程，因此，该数据包随即被送入INPUT链，此时若INPUT链内的规则不允许该数据包进入，那么，该数据包就会被丢弃，当然也就不可能送达本机进程的位置；反之，如果INPUT链内的规则允许该数据包进入，该数据包即被送入本机进程的位置。从这个例子我们可以看到，INPUT链机制是为保护本机进程而设计的。

● 当本机进程生成数据包往外发送时：

如果使用者在本机启动应用程序，这个应用程序就如图中的本机进程，这时若本机进程生成了一个数据包往外发送，首先该数据包会被送入路由表，并根据路由表的内容来决定数据包被传送的路径。接着，数据包随即被送入到OUTPUT链，此时若OUTPUT链内的规则不允许该数据包离开本机，该数据包就会立刻被丢弃掉；反之，该数据包即可由“数据包传出”所标记的位置送离本机。

由以上两点可以看出，如果数据包与“本机”有所关联，那么“数据包传入”及“数据包传出”应该是代表相同的一个接口，如eth0或eth1。

● 当网络数据包穿过本机时：

这个例子有个先决条件，就是防火墙是部署在网关位置，也就是网关式防火墙。所以，当一个数据包要穿过防火墙到另一边时，该数据包会先由“数据包传入”的位置进入，接着

数据包进入路由表，由于该数据包的目的端并不是本机，因此系统会由路由表的内容来决定，数据包是要由另一个接口送出的，并立即被送入FORWARD链，此时若FORWARD链内的规则不允许该数据包穿过，该数据包就会被丢弃掉；反之，则该数据包即由“数据包传出”的位置送离本机。

由以上的例子可以看到，如果数据包与“本机”是无关的，那么“数据包传入”及“数据包传出”将分别代表两个不同的接口。

2.7 规则的匹配方式

到目前为止，相信你对filter机制已经有了更深入的了解，接下来将介绍Netfilter机制中另一个极为重要的概念，就是数据包在每一个链之中被规则“匹配”的方式。因为在设置防火墙规则时，一定会有为数不少的规则，而这些规则将会存在于不同的链中，当然，每个链所包含的规则“数量”可能都不尽相同，不过不管是哪一个表的哪一个链，规则匹配的方法都是“first match”，即所谓的“优先”匹配。以图2-9中的INPUT链为例，当我们在防火墙上添加新规则时，这些规则是按照“先后顺序”一条一条被加入到INPUT链。因此，第一条被加进来的规则，就会存放在INPUT链内的第一条规则，及rule 1，最后被加进来的规则，当然就是INPUT链中的最后一条规则。

Filter 表		
INPUT链	FORWARD链	OUTPUT链
----- rule 1 -----	----- rule 1 -----	----- rule 1 -----
----- rule 2 -----	----- rule 2 -----	----- rule 2 -----
----- rule 3 -----	----- rule 3 -----	----- rule 3 -----
----- rule 4 -----	----- rule 4 -----	----- rule 4 -----
----- rule 5 -----	----- rule 5 -----	
----- rule 6 -----		
----- rule 7 -----		
默认策略	默认策略	默认策略

图2-9 规则的匹配方式

当一个数据包进入INPUT链之后，filter机制就会根据该数据包的特征，从INPUT链中的第一条规则逐一向下匹配。假设该数据包的特征在第一条规则就被匹配到，将由这条规则来决定是否放弃该数据包，如果该规则说：“将数据包丢弃掉”，那么数据包马上就会被丢弃掉，不管后面的规则内容是什么。相反，如果第一条规则说“该数据包可以进来”，那么该数据包随即就进入到“本机进程”的位置，当然，不管后面的规则内容是什么也都不再重

要，这就是“优先匹配”。

不过，还有另一种情况必须加以讨论，万一该数据包的特征是从INPUT链的第一条规则匹配到最后一条规则，却都没有匹配成功时，试问该数据包能不能进入到“本机进程”？这个关键在于每个链的最底端的默认策略(Default Policy)。请注意，不管链中有多少条规则，默认策略永远在每个链的最底端，而且每个链的默认策略各自独立。默认策略只会有一种状态，不是放行(ACCEPT)就是丢弃(DROP)，因此如果默认策略的状态是ACCEPT，那么该数据包就可以进入“本机进程”；如果是DROP，则该数据包就会在默认策略的位置上被丢掉，最后请注意，默认策略的默认状态是ACCEPT。

2.8 Netfilter与iptables的关系

我们常听说Linux防火墙叫做“iptables”，其实这样的称呼并不是很正确，什么是iptables呢？在前面我们提过Netfilter所需要的规则是存放在内存中的，但问题是防火墙管理人员该如何将规则存放到内存呢？因此，防火墙管理人员会需要一个规则编辑工具，通过这个工具来对内存中的规则执行添加、删除及修改等操作，这个工具就是iptables以及ip6tables，其中iptables是在IPV4网络环境中使用，而ip6tables是在IPV6网络环境中使用，因此，Linux防火墙比较正确的名称应该是Netfilter/iptables。

此外，对于iptables结构上的问题我们必须加以了解，前面曾提到，Netfilter的模块越多，防火墙的功能也就越多，而且我们可以通过升级内核的方式来达到一起升级Netfilter的目的。不过，防火墙的升级不是仅仅升级内核就可以解决的，也并非一味增加Netfilter的模块，就可以无限扩充Netfilter的功能。我们要知道，防火墙管理人员是通过iptables工具，将“规则”写入到Netfilter的规则数据库(就是前面所提到的链)中，而这些规则有特定的“语法”，例如：“iptables -t filter -A INPUT -p tcp -m state --state ESTABLISHED, RELATED -j ACCEPT”，因此，当我们将规则传给iptables工具时，iptables工具会先检查语法是否正确，如果不正确，则iptables工具会显示语法错误的警告信息；反之，iptables就会把这些规则写入到规则数据库中。问题是iptables工具如何得知一个“新模块”？其语法该如何下达呢？

事实上，不是只有Netfilter有模块，iptables工具也有模块，这些模块就存放在/lib/x Tables目录下，该目录包含了iptables及ip6tables两个工具的模块。表2-2即为iptables及ip6tables工具的所有模块，其中libxt开头的文件是iptables的模块，libip6t开头的文件则为ip6tables的模块；这些模块理应与Netfilter的模块是一一对应的，如/lib/modules/kernel_version/kernel/net/netfilter/目录中有一个模块称为xt_string.ko，在/lib/x Tables/目录中就会有一

个叫libxt_string.so的模块，也就是说，当我们下达与xt_string.ko相关的语法时，iptables工具会根据libxt_string.so模块的指示去检查语法是否正确，并将Netfilter的libxt_string.ko模块载入到系统内存中，iptables最后将规则写入到规则数据库中。

表2-2 iptables及ip6tables工具的模块列表

libipt_addrtype.so	libipt_ecn.so	libipt_MASQUERADE.so
libipt_REDIRECT.so	libipt_SET.so	libipt_ULOG.so
libipt_ah.so	libipt_ECN.so	libipt_MIRROR.so
libipt_REJECT.so	libipt_SNAT.so	libipt_unclean.so
libipt_CLUSTERIP.so	libipt_icmp.so	libipt_NETMAP.so
libipt_SAME.so	libipt_ttl.so	libipt_DNAT.so
libipt_LOG.so	libipt_realn.so	libipt_set.so
libipt_TTL.so		
libip6t_ah.so	libip6t_eui64.so	libip6t_hbh.so
libip6t_HL.so	libip6t_ipv6header.so	libip6t_mh.so
libip6t_rt.so	libip6t_dst.so	libip6t_frag.so
libip6t_hl.so	libip6t_icmp6.so	libip6t_LOG.so
libip6t_REJECT.so		
libxt_CLASSIFY.so	libxt_conntrack.so	libxt_length.so
libxt_NOTRACK.so	libxt_RATEEST.so	libxt_string.so
libxt_TPROXY.so	libxt_cluster.so	libxt_dccp.so
libxt_limit.so	libxt_osf.so	libxt_recent.so
libxt_tcpmss.so	libxt_TRACE.so	libxt_comment.so
libxt_dscp.so	libxt_mac.so	libxt_owner.so
libxt_sctp.so	libxt_TCPMSS.so	libxt_u32.so
libxt_connbytes.so	libxt_DSCP.so	libxt_mark.so
libxt_physdev.so	libxt_SECMARK.so	libxt_TCPOPTSTRIP.so
libxt_udp.so	libxt_connlimit.so	libxt_esp.so
libxt_MARK.so	libxt_pkttype.so	libxt_socket.so
libxt_tcp.so	libxt_connmark.so	libxt_hashlimit.so
libxt_multiport.so	libxt_policy.so	libxt_standard.so
libxt_time.so	libxt_CONNMARK.so	libxt_helper.so
libxt_NFLOG.so	libxt_quota.so	libxt_state.so
libxt_tos.so	libxt_iprange.so	libxt_NFQUEUE.so
libxt_rateest.so	libxt_statistic.so	libxt_TOS.so
libxt_CONNSECMARK.so		

从以上内容我们可以推断，如果iptables工具没有包含下达规则中所需的模块，那么iptables会说你所下达的语法不正确，又如果iptables包含了我们所下达语法的模块，但Netfilter没有对应的模块可用，iptables会告诉我们某个Netfilter的模块不存在，因此，如果升级内核或Netfilter，iptables的软件也应随之升级才行。

2.9 iptables工具的使用方法

由于iptables命令使用上比较复杂,导致很多有心学好Netfilter/iptables防火墙的人望而却步。其实iptables命令的用法并没有想象中的复杂,只要跟随我的笔触,你很快就会在不经意间把该记的东西通通都记了下来。因此,请千万别急着去参考任何有关iptables的其他资料,这只会增加学习上的困难。

iptables命令可以划分为两个部分,一个是“iptables命令参数”,另一个则是“规则语法”。

2.9.1 iptables命令参数

iptables的参数相当多,但这并不表示你需要记住所有参数,我们实际上用到的参数并不多,只需要记住几个特定参数即可,而且iptables的参数是有规律可循的。在了解这些参数的规则之后,就可以非常容易记住这些参数的用途,请参考图2-4及图2-10,并通过示例来逐一理解每个参数的用法。

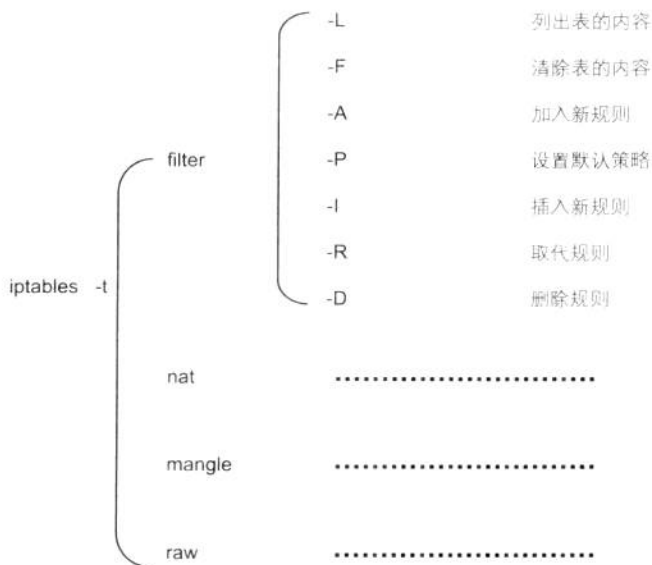


图2-10 规则的匹配方式

● iptables的命令结构

```
iptables -t TABLE - 操作方式 规则条件
```

语法	说明
-t Table	目前的Netfilter版本内置有filter、nat、mangle及raw四个表，即Netfilter的四大功能。-t参数的含义是选择我们所要操作的“功能”，如果这个字段没有输入，默认则是filter表
-操作方式	当我们选择好所要使用的表之后，接下来来决定如何操作这个表，在iptables工具中有很多不同的操作方式，在此仅列出必须了解的操作方式，如下： <ul style="list-style-type: none"> • -L：将所选择的表内容列出 • -A：在指定的链中添加新规则 • -F：将所选择的表内容清除掉 • -P：设置某个链的默认策略
规则条件	关于规则条件稍后将有完整的说明，在此先提供三个规则，请你先不要关心其意义，只要照样输入即可，毕竟目前我们的学习目的在于iptables的参数，而非语法。此处提供这三条规则的目的是让你可以试着使用iptables工具，规则如下： <ul style="list-style-type: none"> • -p tcp -j ACCEPT • -p udp -j ACCEPT • -p icmp -j ACCEPT

示例2.1 列出filter表的所有内容

● 语法

```
iptables -t filter -L
```

● 语法解释

语法	说明
-t filter	本次所要使用的功能是filter
-L	本次对filter表所要执行的操作是list，就是将filter表的所有内容列出来。不过，我们也可以另外加上链名，如此即可单独列出某一个链的内容，如示例2.2所示

● 执行结果(如图2-11所示)

```
[root@localhost ~]# iptables -t filter -L
Chain INPUT (policy ACCEPT) ⑤
 ① target prot opt source destination
  ACCEPT tcp -- anywhere anywhere
  ACCEPT udp -- anywhere anywhere ④
  ACCEPT icmp -- anywhere anywhere

Chain FORWARD (policy ACCEPT) ⑥
 ② target prot opt source destination

Chain OUTPUT (policy ACCEPT) ⑦
 ③ target prot opt source destination
[root@localhost ~]#
```

图2-11 列出filter表的所有内容

标注代码	说明
①	INPUT链的所有内容
②	FORWARD链的所有内容,但其内容是空的,也就是说,目前FORWARD链内没有任何规则存在
③	OUTPUT链的所有内容,但其内容是空的,也就是说,目前OUTPUT链内没有任何规则存在
④	为INPUT链内的所有规则,从图2-11中我们可以看到,INPUT链目前只有三条规则
⑤	INPUT链的默认策略,其当前状态为ACCEPT(允许)
⑥	FORWARD链的默认策略,其当前状态为ACCEPT(允许)
⑦	OUTPUT链的默认策略,其当前状态为ACCEPT(允许)

示例2.2 列出filter表中的INPUT链的内容

● 语法

```
iptables -t filter -L INPUT
```

● 语法解释

语法	说明
-t filter	本次所要使用的功能是filter
-L INPUT	本次对filter表所要执行的操作是list,就是将filter表中的INPUT链内容列出

● 执行结果(如图2-12所示)

```
[root@localhost /]# iptables -t filter -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
ACCEPT     tcp  --  anywhere               anywhere
ACCEPT     udp  --  anywhere               anywhere
ACCEPT     icmp --  anywhere               anywhere
```

图2-12 列出filter表的INPUT链内容

示例2.3 清除filter表中的所有内容

● 语法

```
iptables -t filter -F
```

● 语法解释

语法	说明
-t filter	本次所要使用的功能是filter
-F	本次操作是清除filter表的所有内容,此外,也可以如示例2.2那样设置所要清除的链,如此即可对特定的链执行清除操作

● 执行结果(如图2-13所示)

```
[root@localhost /]# iptables -t filter -F
[root@localhost /]# iptables -t filter -L
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
[root@localhost /]#
```

图2-13 将filter表所有内容清除后的结果

示例2.4 将规则添加到filter表的INPUT链中

● 语法

```
iptables -t filter -A INPUT -p icmp -j ACCEPT
```

● 语法解释

语法	说明
-t filter	本次所要使用的功能是filter
-A INPUT	将规则加入到INPUT链中
规则	-p icmp -j ACCEPT, 为本次添加到INPUT链的规则, 在此暂时不必关心规则的内容

● 执行结果(如图2-14所示)

```
[root@localhost /]# iptables -t filter -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
① ACCEPT    icmp -- anywhere           anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
[root@localhost /]#
```

图2-14 将规则添加到INPUT链后的结果

标注代码	说明
①	为本次所添加的规则

示例2.5 将FORWARD链的默认策略设置为DROP

● 语法

```
iptables -t filter -P FORWARD DROP
```


● 语法解释

语法	说明
-t filter	本次所要使用的功能是filter
-P FORWARD	本次所要设置的是FORWARD链的策略
DROP	将FORWARD链的默认策略设置为DROP

● 执行结果(如图2-15所示)

```
[root@localhost /]# iptables -t filter -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

① Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
[root@localhost /]#
```

图2-15 将FORWARD链的默认策略设置为DROP后的结果

标注代码	说明
①	FORWARD链的默认策略已经被设置为DROP

● 注意事项

iptables的-F参数不会影响到默认策略的状态，要改变默认策略的状态，一定要使用-P的参数来设置。

示例2.6 在INPUT链中插入新的规则

● 语法

```
iptables -t filter -I INPUT 2 -p tcp -j ACCEPT
```

● 语法解释

语法	说明
-t filter	本次要使用的功能是filter
-I INPUT	在INPUT链中插入新的规则
2	插入到第二条规则的位置
规则	-p tcp -j ACCEPT是本次要插入的规则内容

● 执行结果(如图2-16所示)

```

[root@localhost ~]#
[root@localhost ~]# iptables -t filter -I INPUT
Chain INPUT (policy ACCEPT)
target    prot opt source      destination
ACCEPT    tcp  --  anywhere    anywhere
ACCEPT    udp  --  anywhere    anywhere
[root@localhost ~]#
[root@localhost ~]# iptables -t filter -I INPUT --line-number
Chain INPUT (policy ACCEPT)
num target prot opt source      destination
1 -- ACCEPT tcp -- anywhere    anywhere
2 -- ACCEPT udp -- anywhere    anywhere
[root@localhost ~]#
[root@localhost ~]# iptables -t filter -I INPUT 2 -p icmp -j ACCEPT
[root@localhost ~]#
[root@localhost ~]# iptables -t filter -I INPUT --line-number
Chain INPUT (policy ACCEPT)
num target prot opt source      destination
1 -- ACCEPT tcp -- anywhere    anywhere
2 -- ACCEPT icmp -- anywhere    anywhere
3 -- ACCEPT udp -- anywhere    anywhere
[root@localhost ~]#

```

图2-16 在INPUT链中插入新的规则

标注代码	说明
❶	INPUT链的原有内容
❷	我们可以在-I参数后面加上--line-number参数,这样就会在规则前面自动加上行号
❸	将-p icmp-j ACCEPT插入到INPUT链的第二行
❹	验证步骤❸的操作是否成功
❺	在INPUT链的第二行位置果真添加了一条新的规则

示例2.7 取代INPUT链内已经存在的规则

● 语法

```
iptables -t filter -R INPUT 2 -p tcp -j ACCEPT
```

● 语法解释

语法	说明
-t filter	本次所要使用的功能是filter
-R INPUT	取代INPUT链已经存在的规则
2	取代原来的第二条规则
规则	-p tcp -j ACCEPT, 用本规则取代原来的规则

● 执行结果(如图2-17所示)

```
[root@localhost ~]# iptables -t filter -L INPUT --line-number
Chain INPUT (policy ACCEPT)
num target prot opt source destination
1 ACCEPT tcp -- anywhere anywhere
2 ACCEPT icmp -- anywhere anywhere
3 ACCEPT udp -- anywhere anywhere
[root@localhost ~]#
[root@localhost ~]# iptables -t filter -R INPUT 2 -p tcp -j ACCEPT
[root@localhost ~]#
[root@localhost ~]# iptables -t filter -L INPUT --line-number
Chain INPUT (policy ACCEPT)
num target prot opt source destination
1 ACCEPT tcp -- anywhere anywhere
2 ACCEPT tcp -- anywhere anywhere
3 ACCEPT udp -- anywhere anywhere
[root@localhost ~]#
[root@localhost ~]#
```

图2-17 取代INPUT链中原来的规则

标注代码	说明
❶	INPUT链的原来的第二条规则
❷	用-p tcp -j ACCEPT取代原来的第二条规则
❸	验证步骤❷的操作是否成功
❹	INPUT链原来的第二条规则已经被取代

示例2.8 删除INPUT链中已经存在的规则

● 语法

```
iptables -t filter -D INPUT 2
```

● 语法解释

语法	说明
-t filter	本次所要使用的功能是filter
-D INPUT	删除INPUT链已经存在的规则
2	删除原来的第二条规则

● 执行结果(如图2-18所示)

```
[root@localhost ~]# iptables -t filter -L INPUT --line-number
Chain INPUT (policy ACCEPT)
num target prot opt source destination
1 ACCEPT tcp -- anywhere anywhere
2 ACCEPT icmp -- anywhere anywhere
3 ACCEPT udp -- anywhere anywhere
[root@localhost ~]#
[root@localhost ~]# iptables -t filter -D INPUT 2
[root@localhost ~]#
[root@localhost ~]# iptables -t filter -L INPUT --line-number
Chain INPUT (policy ACCEPT)
num target prot opt source destination
1 ACCEPT tcp -- anywhere anywhere
2 ACCEPT udp -- anywhere anywhere
[root@localhost ~]#
```

图2-18 删除INPUT链中原有的规则

标注代码	说明
❶	INPUT链原来的第二条规则
❷	删除INPUT链的第二条规则
❸	验证步骤❷的操作是否成功
❹	INPUT链原来的第二条规则已经被删除

看完以上七个参数后，你是否在“不小心”的情况下就记住了呢？事实上，你所记住的比你以为记住的还要多，怎么说呢？你可以将刚刚所学的七个参数套用到其他表上。下面将演示如果在其他表中使用这些参数，当然啦！其他较为具体的内容会在稍后章节详细介绍，现在你只需了解如何使用这七个参数就可以了。

示范2.1 nat表的操作

● 操作过程(如图2-19所示)

```

[root@localhost ~]#
[root@localhost ~]# iptables -t nat -L ❶
Chain PREROUTING (policy ACCEPT)
target prot opt source destination
Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
[root@localhost ~]# iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.0/24 -j SNAT --to 192.168.5.178 ❸
[root@localhost ~]# iptables -t nat -L ❹
Chain PREROUTING (policy ACCEPT)
target prot opt source destination
Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
SNAT all -- 192.168.0.0/24 anywhere to:192.168.5.178 ❺
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
[root@localhost ~]# iptables -t nat -F ❻
[root@localhost ~]# iptables -t nat -F ❼
Chain PREROUTING (policy ACCEPT)
target prot opt source destination
Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
[root@localhost ~]#

```

图2-19 nat表的操作示范

● 流程解释

标注代码	字段解释
①	使用iptables工具将nat表的内容列出
②	nat表的内容, 并且我们可以看到nat表中的三个链, 也可以看到每个链的默认策略
③	将规则-o eth0 -s 192.168.0.0/24 -j SNAT --to 192.168.5.178添加到NAT表的POSTROUTING链中
④	使用iptables工具将nat表的内容列出, 以验证步骤③的操作是否成功
⑤	我们可以看到步骤③的规则已经正确地添加到POSTROUTING链之中
⑥	使用iptables工具将nat表中的内容清空
⑦	验证步骤⑥的操作是否成功
⑧	POSTROUTING链的内容已成功地被清空了

示范2.2 mangle表的操作

● 操作过程(如图2-20所示)

```

[root@localhost ~]# iptables -t mangle -F ①
Chain PREROUTING (policy ACCEPT)
target prot opt source destination

Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain FORWARD (policy ACCEPT) ②
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# iptables -t mangle -A INPUT -p tcp -j ACCEPT ③
[root@localhost ~]#
[root@localhost ~]# iptables -t mangle -F ④
Chain PREROUTING (policy ACCEPT)
target prot opt source destination

Chain INPUT (policy ACCEPT)
target prot opt source destination ⑤
ACCEPT tcp -- anywhere anywhere

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
[root@localhost ~]#
[root@localhost ~]# iptables -t mangle -F ⑥
[root@localhost ~]#
[root@localhost ~]# iptables -t mangle -F ⑦
Chain PREROUTING (policy ACCEPT)
target prot opt source destination

Chain INPUT (policy ACCEPT)
target prot opt source destination ⑧

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
[root@localhost ~]#

```

图2-20 mangle表的操作示例

● 流程解释

标注代码	说明
❶	使用iptables工具将mangle表的内容列出
❷	mangle表的内容, 并且我们可以看到mangle表中的五个链, 也可以看到每个链的默认策略
❸	将规则-p icmp -j ACCEPT添加到mangle表的INPUT链中
❹	使用iptables工具将mangle表的内容列出, 以验证步骤❶的操作是否成功
❺	我们可以看到步骤❸的规则已经正确添加到INPUT链之中
❻	使用iptables工具将mangle表中的内容清空
❼	验证步骤❹的操作是否成功
❽	INPUT链的内容已成功地被清空了

示范2.3 raw表的操作

● 操作过程(如图2-21所示)

```

[root@localhost ~]# iptables -t raw -F ❶
Chain PREROUTING (policy ACCEPT)
target     prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
[root@localhost ~]# iptables -t raw -A OUTPUT -p tcp -j NOTRACK ❸
[root@localhost ~]# iptables -t raw -F ❷
Chain PREROUTING (policy ACCEPT)
target     prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
NOTRACK    tcp  --  anywhere              anywhere  ❺
[root@localhost ~]# iptables -t raw -F ❻
[root@localhost ~]# iptables -t raw -F ❼
Chain PREROUTING (policy ACCEPT)
target     prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination  ❽
[root@localhost ~]#

```

图2-21 raw表的操作示例

● 流程解释

标注代码	说明
❶	使用iptables工具将raw表的内容列出
❷	raw表的内容, 并且我们可以看到raw表中的两个链, 也可以看到每个链的默认策略
❸	将规则-p tcp -j NOTRACK添加到raw表的OUTPUT链之中
❹	使用iptables工具将raw表的内容列出来, 以验证步骤❸的操作是否成功
❺	我们可以看到步骤❸的规则, 已经正确的加入到OUTPUT链之中
❻	使用iptables工具将raw表中的内容清空
❼	验证步骤❻的操作是否成功
❽	OUTPUT链的内容已成功地被清空了

从以上示例, 你是否已经找到了共性呢? 其实在使用iptables工具时, 只需设置所要使用的功能(Tables)、操作方法以及选择正确的链, 即可轻松地使用iptables工具。其余三个参数就不多做说明了, 你不妨自己动手操作看看。

2.9.2 iptables规则语法

在了解了iptables参数的用法之后, 接下来要进入较复杂的“语法”部分。不过, 实际上也没想象中的复杂, 只要能够了解语法间的关联性, 即可很容易理解及记忆。此处将iptables的语法分成两部分, 其一是“基本语法”, 其二是“高级语法”, 如下例所示。

● 基本语法

```
iptables -t filter -A INPUT -p icmp -j DROP
```

● 高级语法

```
iptables -t filter -A INPUT -m mac --mac-source 00:E0:18:00:7C:A4 -j DROP
```

如何区分基本语法及高级语法呢? 首先要了解一点, filter机制是由ipTable_filter.ko模块提供的功能, 而这个模块本身就已经提供了一些简单的匹配及过滤方式, 而所谓基本语法是指只使用ipTable_filter.ko模块自身所提供的功能; 那高级语法呢? 因为基本语法本身所能执行的匹配及过滤操作较简单, 因此如果我们要进行较复杂的过滤操作, 就必须调用其他模块的功能。从高级语法的示例来看, -m mac就是告知iptables工具, 我们要调用xt_mac.ko这个模块的功能, 由于是调用其他模块, 因此语法部分将会因模块而异, 且每个模块的语法都是不一样的, 这部分称为“高级语法”。

由于iptables的语法部分变化比较大, 因此经常成为想跨入Netfilter/iptables领域的人最大的障碍。其实你根本不用担心这个问题, 只要按照这里描述的步骤去做, 很快就可以学会,

现阶段我们先学会一部分基本语法，高级语法部分则安排在下一个单元中介绍。

下面将列举几个不同示例来引导你快速学习iptables基本语法部分。

示例2.9 将192.168.0.200进入本机的icmp协议包都丢弃

● 语法

```
iptables -A INPUT -p icmp -s 192.168.0.200 -j DROP
```

● 语法解释

● -A INPUT

保护的對象 - 因为本示例所使用的是INPUT链，故其保护的對象是本机。

● -p icmp

原文	-p, Protocol
目的	匹配某种特定协议的数据包，本示例是匹配icmp协议包
语法	-p icmp、-p udp、-p tcp、-p all等，稍后章节将更完整地介绍其他更高级的用法

● -s 192.168.0.200

原文	-s, Source
对应的参数	-d, Destination
目的	匹配数据包中“来源”或“目的”端的IP
语法	-s 192.168.1.10、-s 192.168.1.0/24、-s www.playboy.com、 -d 192.168.1.10、-d 192.168.1.0/24、-d www.playboy.com、 从以上这几个示例可以得知，IP位置的标识方法可以是单一IP或者标准的CIDR网段，至于FQDN的部分，事实上，当使用者按下Enter键时，iptables工具会将FQDN送到DNS去执行名称解析，最后添加的规则还以IP形式表示

● -j

原文	JUMP
目的	将符合以上两个条件的数据包以特定方式来处理

● 常见的处理方式

ACCEPT	允许通过
DROP	将数据包丢弃掉，此种处理方式将导致来源端误以为数据包丢失而不断重新发送数据包。这个动作将持续到连接超时为止
REJECT	将数据包丢弃掉，并回送一个Destination Unreachable的icmp数据包给发送端，发送端的应用程序在收到这个错误信息数据包后，会终止连接

示例2.10 不允许192.168.0.200主机通过本机的DNS服务来执行名称解析

● 语法

```
iptables -A INPUT -p udp -s 192.168.0.200 --dport 53 -j REJECT
```

● 语法解释

● -A INPUT

保护的對象 因为本示例所使用的是INPUT链，故其保护的對象是本机

● -p udp

目的 匹配udp协议的数据包

● -s 192.168.0.200

目的 匹配来源端IP为192.168.0.200的数据包

● --dport 53

原文 --dport、Destination Port

对应的参数 --sport、Source Port

目的

匹配TCP、UDP包头中的“来源端口”或“目的端口”，如此即可判断连接要访问的服务。例如-p udp --dport 53代表客户端要访问udp的53端口，而udp的53端口就是DNS服务

--dport 80、--sport 80，但请注意，当使用--dport或--sport参数时，一定要指明是tcp或udp协议，不能写成iptables -A INPUT -s 192.168.1.10 --dport 80 -j DROP，原因是我们没有设置所要匹配的协议。因此不管是tcp、udp或icmp协议都会符合，也就是说，这个示例至少符合以下三种情况：

语法

- iptables -A INPUT -p tcp -s 192.168.1.10 --dport 80 -j DROP
- iptables -A INPUT -p udp -s 192.168.1.10 --dport 80 -j DROP
- iptables -A INPUT -p icmp -s 192.168.1.10 --dport 80 -j DROP

第一种为匹配tcp协议及目的端口为80者，这并没有什么问题；第二种为匹配udp协议及目的端口为80者，这也没有什么问题；但第三种就不行了，因为第三种情况所要匹配的是icmp协议及目的端口为80者，问题是icmp协议哪来的端口80，所以这个语法是错误的。

最后请记住一点，只要规则语法中使用到“端口参数”，一定要加上-p tcp或-p udp参数

● -j REJECT

目的 将符合以上三项条件的数据包丢弃，并回送ICMP的错误信息数据包给来源端

示例2.11 允许192.168.0.200主机连接到本机的TELNET服务

● 语法

```
iptables -A INPUT -p tcp -s 192.168.0.200 --dport 23 -j ACCEPT
```

- 语法解释

- -A INPUT

保护的對象	因为本示例使用INPUT链，故其保护的對象是本机
-------	--------------------------

- -p tcp

目的	匹配tcp协议的数据包
----	-------------

- -s 192.168.0.200

目的	匹配来源端IP为192.168.0.200的数据包
----	---------------------------

- --dport 23

目的	匹配目的端Port为23的数据包，而TCP Port 23即为TELNET服务
----	---

- -j ACCEPT

目的	允许符合以上三个条件的数据包进入
----	------------------

示例2.12 允许192.168.1.0/24网段的主机向本机192.168.0.1提出任何服务请求

- 语法

```
iptables -A INPUT -p all -s 192.168.1.0/24 -d 192.168.0.1 -j ACCEPT
```

- 语法解释

- -A INPUT

保护的對象	因为本示例使用INPUT链，故其保护的對象是本机
-------	--------------------------

- -p all

目的	匹配任何协议的数据包
----	------------

- -s 192.168.1.0/24

目的	匹配来源端IP为192.168.1.0/24网段的数据包
----	------------------------------

- -d 192.168.0.1

目的	匹配目的端IP为192.168.0.1的数据包
----	-------------------------

- -j ACCEPT

目的	允许符合以上三个条件的数据包进入
----	------------------

示例2.13 只允许客户端主机从eth1这个接口访问本机的SSH服务

- 语法

```
iptables -A INPUT -p tcp -i eth1 --dport 22 -j ACCEPT
```

- 语法解释

- -A INPUT

保护的對象	因为本示例所使用的是INPUT链，故其保护的對象是本机
-------	-----------------------------

- -p tcp

目的	匹配tcp协议的数据包
----	-------------

- -i eth1

来源	-i、in-interface
对应的参数	-o、out-interface
目的	匹配数据包的进出接口
语法	-i eth1、-o eth2

- --dport 22

目的	匹配目的端口为22的数据包
----	---------------

- -j ACCEPT

目的	允许符合以上三个条件的数据包进入
----	------------------

示例2.14 不允许本机的应用程序从eth0接口发送数据包去访问edu.uuu.com.tw以外的网站

- 语法

```
iptables -A OUTPUT -o eth0 -p tcp -d ! edu.uuu.com.tw --dport 80 -j REJECT
```

- 语法解释

- -A OUTPUT

限制的对象	因为本示例所使用的是OUTPUT链，故其目的是限制本机对外访问
-------	---------------------------------

- -o eth0

目的 匹配数据包是否由eth0接口送出的

- -p tcp

目的 匹配tcp协议的数据包

- -d !edu.uuu.com.tw

目的

请先不要看“!”部分，所以我们看的是-d edu.uuu.com.tw，整个规则意思是“如果本机的应用程序要从eth0接口发送tcp协议的数据包到edu.uuu.com.tw的端口80，是不允许的”，不过当我们把“!”考虑进来时，结果就不一样了。“!”是反向的意思，因此，这个规则的意思将变成“如果本机的应用程序要从eth0接口发送tcp协议的数据包到edu.uuu.com.tw以外其他主机的端口80，是不允许的”。

- --dport 80

目的 匹配目的端口为80的数据包

- -j REJECT

目的 将符合以上四个条件的数据包丢弃，并回送ICMP的错误信息包给来源端

示例2.15 不允许本企业内部的主机访问企业以外的任何网站

- 语法

```
iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 80 -j DROP
```

- 补充说明

本示例使用网关式防护墙，因此所使用的链是FORWARD，另外假设这台防火墙主机的eth0接口连接到因特网，而eth1连接到企业内部。

- 语法解释

- -A FORWARD

保护的対象 因为本示例所使用的是FORWARD链，因此被保护或限制的对象是防护墙后面的主机

- -i eth1

目的 匹配数据包进入的接口进入

- -o eth0

目的 匹配数据包离开的接口，因此结合-i及-o两个参数即可匹配数据包的流向

- -p tcp

目的 匹配tcp协议的数据包

- --dport 80

目的	匹配目的端口为80的数据包
----	---------------

- -j REJECT

目的	将符合以上四个条件的数据包丢弃
----	-----------------

看完以上这几个示例之后，相信你已经了解iptables的基本语法，但为了更清楚地记住这些参数，此处将所使用的参数进行了整理，如下整理。

- 接口的匹配参数

参数名称	-i、-o
参数值	参数值会因为防火墙主机所使用的网络实体接口而异，以下为常见的网络接口名称： eth0：以太网的接口名称 ppp0：ppp接口的名称 lo：Local Loopback接口 fd0：光纤网络接口
使用示例	-i eth0：匹配从eth0接口送入的数据包 -o eth0：匹配从eth0接口离开的数据包
意义	匹配数据包进出的接口
补充	可搭配“!”来代表反向，例如“-i! eth0”，即代表匹配不是从eth0接口进入的数据包

- 上层协议(upper layer protocol)的匹配参数

参数名称	-p																					
	<p>这些参数会因匹配的上层协议而异，一般常见的参数如下：</p> <p>tcp：匹配的上层协议是为tcp协议</p> <p>udp：匹配的上层协议是为udp协议</p> <p>icmp：匹配的上层协议是为icmp协议</p> <p>all：匹配所有的上层协议</p> <p>其实，上层协议不止这三项，关于上层协议你可以参考/etc/protocols文件，下面摘选其部分内容：</p>																					
参数值	<table><tr><td>ip</td><td>0</td><td>IP</td></tr><tr><td>icmp</td><td>1</td><td>ICMP</td></tr><tr><td>igmp</td><td>2</td><td>IGMP</td></tr><tr><td>ggp</td><td>3</td><td>GGP</td></tr><tr><td>ipencap</td><td>4</td><td>IP-ENCAP</td></tr><tr><td>st</td><td>5</td><td>ST</td></tr><tr><td>tcp</td><td>6</td><td>TCP</td></tr></table> <p>其中第一列是上层协议的名称(给系统看的)，第二列是上层协议的代码(给系统看的)，第三列是上层协议的名称(给系统管理员看的)。因此在使用-p参数时，除了可以使用“tcp”来表示TCP协议之外，还可以使用“6”来表示TCP协议</p>	ip	0	IP	icmp	1	ICMP	igmp	2	IGMP	ggp	3	GGP	ipencap	4	IP-ENCAP	st	5	ST	tcp	6	TCP
ip	0	IP																				
icmp	1	ICMP																				
igmp	2	IGMP																				
ggp	3	GGP																				
ipencap	4	IP-ENCAP																				
st	5	ST																				
tcp	6	TCP																				

参数名称	-p
使用示例	-p tcp、-p 6：匹配tcp协议 -p icmp、-p 1：匹配icmp协议
意义	匹配上层通信协议
补充	可搭配“!”来代表反向，例如“-p ! icmp”表示匹配的不是icmp的数据包

● 匹配来源/目的的IP地址

参数名称	-s、-d
参数值	来源及目的IP地址匹配，其可以识别的IP地址格式如下： 192.168.0.1：匹配单一IP 172.10.0.0/16：匹配一个B类的网段 192.168.0.0/24：匹配一个C类的网段 192.168.0.0/28：也可以是任何标准CIDR的网段 www.playboy.com：可以是FQDN，但最后存放到链中的值还是IP
使用示例	-s 192.168.0.1：匹配从192.168.0.1主机发送来的数据包 -s 192.168.0.0/24：匹配从192.168.0.0/24网段所发送来的数据包 -d 192.168.0.10：匹配要发送到192.168.0.10主机的数据包
意义	匹配数据包的来源或目的IP地址
补充	可搭配“!”来代表反向，例如“-s ! 192.168.0.0/24”代表匹配来源端IP不是192.168.0.0/24网段的数据包

● 匹配来源/目的的端口

参数名称	--sport、--dport
参数值	匹配端口的用意在于匹配连接所需访问的服务。例如，我们可以使用--dport 80参数来匹配所有要访问Web服务的数据包，也可以使用--sport 80参数来匹配所有由Web服务应答给客户端的数据包
使用示例	--dport 80：匹配要访问Web服务的数据包 --sport 110：匹配由POP3服务应答给客户端的数据包
意义	匹配数据包的来源或目的端口
补充	可搭配“!”来代表反向，例如“--sport ! 80”代表匹配不是从Web服务发送出去的数据包

● 处理方式

参数名称	-j
参数值	较常见的处理方式有三种，其分别如下： ACCEPT：允许 DROP：将数据包丢弃 REJECT：将数据包丢弃，并回送给发送端一个ICMP数据包
使用示例	-j ACCEPT：允许 -j DROP：将数据包丢弃
意义	采用特定方式来处理符合条件的数据包

2.9.3 学以致用：iptables的规则语法

看完以上内容之后，你是否对iptables工具的使用方式以及规则语法有了初步认识呢？接下来以图2-22为例，列举几个例子试试看是否真能达到期望的进度。

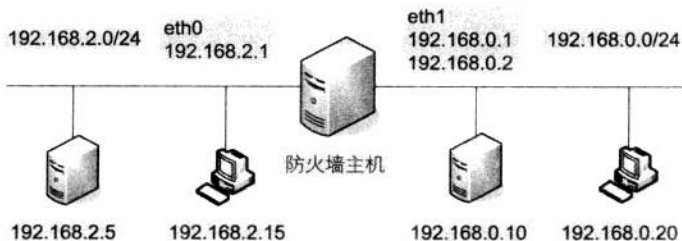


图2-22 iptables示例

题目

第1题	假设在防火墙主机上执行“iptables -A INPUT -p icmp -j DROP”命令，请问192.168.2.15及192.168.0.20哪一台主机可以Ping到防火墙主机？
第2题	假设在防火墙主机上执行“iptables -A INPUT -i eth0 -p icmp -d 192.168.0.2 -j DROP”命令，请问192.168.2.15及192.168.0.20哪一台主机可以Ping到防火墙主机的192.168.0.2这个IP？
第3题	假设在防火墙主机上有Web 服务正在运行中，其我们在防火墙主机上执行“iptables -A INPUT -i eth1 --dport 80 -s 192.168.0.0/24 -j REJECT”命令，请问在图2-22中有哪些主机可以访问到该Web 服务？
第4题	假设192.168.2.5主机为Web 服务器，且我们在防火墙主机上执行“iptables -A INPUT -i eth1 -p tcp -d 192.168.2.5 --dport 80 -j REJECT”命令，请问192.168.0.20及192.168.2.15哪一台主机可以访问到该Web 服务？
第5题	假设192.168.2.5及192.168.0.10主机都是Web 服务器，且我们在防火墙主机上执行“iptables -A FORWARD -i eth0 -o eth1 -p tcp --dport 80 -j REJECT”命令，请问： <ul style="list-style-type: none"> 192.168.0.20可以访问到哪一台Web 服务器？ 192.168.2.15可以访问到哪一台Web 服务器？

解答

第1题	都不行。因为在这个示例中的匹配条件只有一个“-p icmp”，因此不管是哪一台主机去ping防火墙主机，该数据包都会符合-p icmp的条件，如此数据包就会被丢弃。
第2题	只有192.168.0.20可以Ping到。因为在这个例子中有两个条件，分别是“-i eth0”和“-p icmp”，因此如果是192.168.2.15发送icmp包到192.168.0.2，该数据包就会符合以上两个条件，并遭到丢弃；但如果是192.168.0.20发送来的数据包，因为该包是从eth1进来的，因此并不会全部符合我们设定的条件，所以192.168.0.20就可以成功ping到防火墙主机192.168.0.2的IP。
第3题	都可以。因为这个语法是不正确的，前面特意提到过，如果在语法中有使用到--sport或--dport参数，那么就一定要指明是TCP还是UDP协议。

第4题

都可以。因为192.168.2.15与192.168.2.5是在同一个网络实体层上，所以无论如何，防火墙都不可能干涉到192.168.2.15对192.168.2.5的访问操作，而192.168.0.20为何也可以访问到192.168.2.5主机？因为这个示例的规则使用的是INPUT链，而INPUT链是用来保护防火墙本机用的，因此根本不可能干涉到192.168.0.20对192.168.2.5的访问操作，除非改用FORWARD链

第5题

- 192.168.0.20可以访问到两台Web Server，因为192.168.0.20与192.168.0.10是在防火墙的同一侧，因此防火墙无法干涉，而192.168.0.20也可以访问到192.168.2.5，因为规则条件指的是“-i eth0 -o eth1”，也就是说，从eth0进入，并从eth1送出的数据包才符合；192.168.0.20访问192.168.2.5主机时，该数据包并不符合这个条件
- 192.168.2.15可以访问到192.168.2.5，因为192.168.2.15与192.168.2.5是在防火墙的同一侧，因此防火墙干涉不到这个访问操作。但当192.168.2.15要访问192.168.0.10主机时，这个数据包正好符合条件中的-i eth0 -o eth1及--dport 80两个条件，然后数据包就会被丢弃了

如果以上示例你没有全部答对，则建议你从第2.9.1一节开始再复习一次；如果全部答对了，恭喜你，对iptables的基本语法应该已经掌握80%以上了，且对于接下来章节所提到的语法，也应该可以很快进入状态。

2.10 使用iptables机制来构建简单的单机防火墙

本节将以图2-23为例来构建一个简单的单机防火墙，图中192.168.0.1即为这个示例的主角，我们在192.168.0.1主机上分别启用SSH、TELNET、SMTP、WEB及POP3五项服务。此外，测试用的客户端主机分别为192.168.0.100及192.168.0.200这两台主机，为了确保服务器192.168.0.1的系统安全，我们使用Linux内置的防火墙来保护服务器本身的安全，为了使防火墙的规则更为安全可靠，请你在设置这些规则之前，

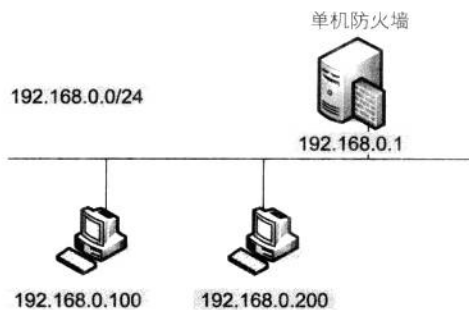


图2-23 单机防火墙示例

务必先将我们所需的需求列出来之后，再来实际搭建防火墙，本示例的需求如下：

- 网络上的任何主机都能正常访问192.168.0.1主机上的SSH及TELNET以外的服务。
- 网络上只有192.168.0.200这台主机可正常访问192.168.0.1主机上的所有服务。

制定出规则后，接着就是将规则转换成iptables看得懂的语法，而在转换成iptables语法时，请务必记住撰写防火墙规则的原则：“先拒绝所有连接，再逐一开放对外提供的服务”，因此将规则转成如表2-3所示的iptables语法。

表2-3 单机防火墙的规则列表

```

1. iptables -P INPUT DROP
2. iptables -A INPUT -p tcp -d 192.168.0.1 --dport 25 -j ACCEPT
3. iptables -A INPUT -p tcp -d 192.168.0.1 --dport 80 -j ACCEPT
4. iptables -A INPUT -p tcp -d 192.168.0.1 --dport 110 -j ACCEPT
5. iptables -A INPUT -p tcp -s 192.168.0.200 -d 192.168.0.1 --dport 22 -j ACCEPT
6. iptables -A INPUT -p tcp -s 192.168.0.200 -d 192.168.0.1 --dport 23 -j ACCEPT

```

当按照表2-3的内容运行iptables后，INPUT链的内容就会如图2-24所示。前面曾说过设置防火墙规则的原则：“先拒绝所有连接，再逐一开放对外提供的服务”，所以表2-3内容的第一行“iptables -P INPUT DROP”意即拒绝所有连接。试想，如果INPUT链中没有任何内容，而且默认策略又处于DROP状态，那么任何进入到INPUT链的数据包都会直接进入默认策略而遭到丢弃，所以防火墙的规则不可能只写这一行。

```

[root@localhost /]# iptables -L INPUT
Chain INPUT (policy DROP)
target     prot opt source                destination           tcp dpt:25
ACCEPT     tcp  --  0.0.0.0/0             192.168.0.1           tcp dpt:80
ACCEPT     tcp  --  0.0.0.0/0             192.168.0.1           tcp dpt:110
ACCEPT     tcp  --  192.168.0.200         192.168.0.1           tcp dpt:22
ACCEPT     tcp  --  192.168.0.200         192.168.0.1           tcp dpt:23
[root@localhost /]#

```

图2-24 单机防火墙INPUT链列表

表2-3的第2~4行规则中设置了三个条件，分别是“-p tcp”、“-d 192.168.0.1”及“--dport XX”。这三行规则刚好符合我们需求的第一项：“网络上的任何主机皆可正常访问192.168.0.1主机上的SSH及TELNET以外的服务”，我们来看看这三条规则是如何符合第一项需求的。

假设网络上有一台主机需要访问服务器上的POP3服务，接下来分析一下这个访问操作能否成功。当访问POP3的数据包进入INPUT链之后，将会由表2-3的第二行开始执行匹配(别忘了，前面所学的默认策略永远在每个链的最后一行才会被匹配)，第二行规则中有三项匹配条件，分别是“-p tcp”、“-d 192.168.0.1”、“--dport 25”。而进来的数据包将会符合“-p tcp”及“-d 192.168.0.1”两个条件，不过，第三个条件“--dport 25”则无法符合，因为该数据包要访问的是POP3服务，也就是该数据包中的目的端口应为110，故第三个条件是无法符合的。

既然无法符合，数据包必定将进行第三行规则的匹配，在第三行规则中一样有三个条件，其实与上一行规则差不多，只有第三个条件不一样，是“--dport 80”，因此这个访问POP3服务的数据包当然也符合完全符合第三行规则。接着数据包进入第四行规则进行匹配操作，而这个访问POP3服务的数据包，将会符合第四行规则中的三个条件。如此一来，这个数据包就会被第四行规则ACCEPT进来，当然该数据包就无需继续往下匹配(别忘了，前面所学的Netfilter匹配原则是优先匹配——First Match)，当然也就不会因为遇到默认策略而遭丢弃了。

接着将以上的文字以图形来解说。如图2-25所示，图中的防火墙就好比默认策略，如果这道墙上没有打开任何洞口，任何由客户端送来的数据包都会撞壁而死。因此，如果我们要让客户端可以正常访问某些服务，就必须在墙上“挖洞”，当我们在服务对应的位置挖开一个洞时，如端口110，这时客户端送来的POP3服务请求包就可以通过我们在防火墙上所挖开的洞口钻进来，然后送到POP3服务上。

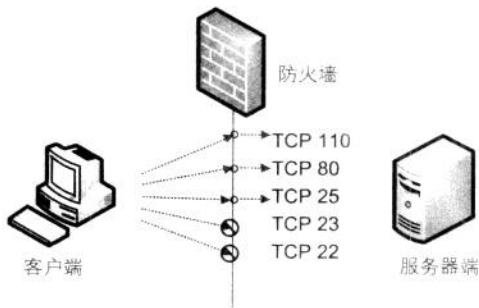


图2-25 单机防火墙机制

那如果网络上的主机试图访问服务器上的TELNET或SSH服务，是否会成功呢？其实这个访问操作并不一定会成功，因为在表2-3的第5条及第6条规则中，比前面的规则多了一个条件“-s 192.168.0.200”；因此，网络上除了192.168.0.200这台主机之外，任何主机都无法访问服务器上的TELNET及SSH服务，而这两条规则将满足我们的第二项需求：“网络上只有192.168.0.200这台主机可正常访问192.168.0.1主机上的所有服务。”

2.10.1 如何测试防火墙规则正确与否

设置完防火墙上的规则后，就可以开始测试单机防火墙的设定值是否符合自己的需要，而整个测试流程则分为“网络上任意主机对服务器的访问测试”及“服务器主机对网络上其他主机的访问测试”两个阶段。

1. 网络上任意主机对服务器的访问测试

这个测试过程并不难，我们只需在192.168.0.100及192.168.0.200这两台主机上，分别对服务器主机192.168.0.1提出各种不同的服务请求即可。如果请求有得到回应，那就表示防火墙上对该项服务是开放的；如果没有得到回应，就代表防火墙对该项服务是没有开放的，在此请注意以下几点：

- 当客户端访问一个防火墙没有开启的服务时，因为默认策略状态为DROP，因此客户端的应用程序将会有卡住的现象而不会立即断线，如果要等到断线，通常需要等一段时间，而这段时间约为3分钟左右。
- 在这个试验中，如果你在客户端主机，尝试去访问服务器上防火墙有开放的服务时，有可能在等很久之后，服务器才会给你应答，这个问题并不是每一台服务器都会如此，所以当你在客户端主机上连接服务器时，服务器如果没有立即应答，就请耐心等待。为何会有此类现象？解决办法是什么？稍后会有完整且详细的说明，现阶段请暂时把这个

现象视为正常。

如果网络状态机防火墙的参数设置都没有问题，那么这项测试应该可以很顺利地完
成，并且符合我们的需求。图2-26为笔者实际的测试流程，首先可以看到①②③④⑤五个服务的
测试，其分别指代表2-4显示的内容。

表2-4 标注与测试服务的对应表

标注	测试的服务项目
①	Port 25、SMTP测试服务
②	Port 80、HTTP测试服务
③	Port 110、POP3测试服务
④	Port 22、SSH测试服务
⑤	Port 23、TELNET测试服务

```
① { root@localhost tmp # telnet 192.168.0.1 25
    Trying 192.168.0.1
    Connected to 192.168.0.1 (192.168.0.1)
    Escape character is '^'
    220 localhost ESMTP Sendmail 8.13.8/8.13.8, Fri, 13 Apr 2007 16:21:51 +0800
    quit
    221 2.0.0 localhost localdomain closing connection
    Connection closed by foreign host
    [root@localhost tmp] #
    [root@localhost tmp] #

② { root@localhost tmp # wget http://192.168.0.1/index.html
    --001223-- http://192.168.0.1/index.html
    正在连接192.168.0.180... 连接上了。
    已发送HTTP请求，正在等待应答... 200 OK
    长度: 1701(1.7K)[text/html]
    Saving to 'index.html'

    100%[=====>] 1,701 --K/s in 0.01s

    001223 (167 KB/s) -- 已存储 'index.html' (1701/1701)
    [root@localhost tmp] #
    [root@localhost tmp] #

③ { root@localhost tmp # telnet 192.168.0.1 110
    Trying 192.168.0.1
    Connected to 192.168.0.1 (192.168.0.1)
    Escape character is '^'
    +OK Dovecot ready
    quit
    +OK Logging out
    Connection closed by foreign host
    [root@localhost tmp] #
    [root@localhost tmp] #

④ { root@localhost tmp # ssh 192.168.0.1 22
    ssh connect to host 192.168.0.1 port 22 Connection timed out
    [root@localhost tmp] #
    [root@localhost tmp] #

⑤ { root@localhost tmp # telnet 192.168.0.1 23
    Trying 192.168.0.1
    telnet connect to address 192.168.0.1 Connection timed out
    telnet Unable to connect to remote host: Connection timed out
    [root@localhost tmp] # }
```

图2-26 网络上任意主机对服务器的访问测试

● 第①阶段测试说明:

在SMTP的测试中,笔者使用Telnet客户端工具去连接192.168.0.1主机的端口25②,接下来需要一些时间来等待服务器的应答;当看到服务器有应答之后,就代表防火墙是允许这个连接操作的。接着输入quit命令,以终止与SMTP服务的连接②。

● 第②阶段测试说明:

在HTTP的测试中,笔者使用wget工具去下载服务器上的index.html文件③,接着,我们可以从标注④的地方清楚看到,index.html文件已经成功被下载回来了,由此也可以确定防火墙是允许这个连接操作的。

● 第③阶段测试说明:

在POP3的测试中,笔者使用Telnet客户端工具去连接192.168.0.1主机的端口110⑤,并且即刻就可以得到服务器的应答,由此可以证明防火墙是允许这个连接请求操作,接着可以输入quit命令来终止与POP3服务的连接⑥。

● 第④阶段测试说明:

在这个项目中,我们测试的是SSH服务,因此使用SSH客户端工具来连接192.168.0.1服务器的SSH服务⑦;接着,大约需要约三分钟时间等待结果,最后会看到⑦中的“ssh: connect to host-192.168.0.1 port 22: Connection timed out”错误信息,这可以证明防火墙不允许这个连接请求操作。

● 第⑤阶段测试说明:

在这个项目中,我们测试的是TELNET服务,因此使用Telnet客户端工具来连接192.168.0.1服务器的TELNET服务⑧,接着,约需三分钟的时间等待结果,最后就会看到⑧的“telnet: connect to address 192.168.0.1: Connection timed out; telnet: Unable to connect to remote host: Connection timed out”错误信息,这也可以证明防火墙是不允许这个连接操作的。

2. 服务器对网络上其他主机的访问测试

在测试完“网络上任意主机对服务器的访问测试”项目之后,接着要做的是“服务器对网络上其他主机的访问测试”。表2-3是我们在服务器上所设置的规则条件,请问在这个规则条件下,如果我们在服务器上使用SSH客户端去连接192.168.0.100主机上的SSH服务(该主机并没有设置任何的防火墙参数),这个SSH的连接请求操作是否可以成功?

当你看完表2-3之后,可能很快就能回答“可以”,因为服务器上并没有对OUTPUT链施加任何限制,乍看之下好像蛮有道理的,但可惜答案是“行不通”。我们以图2-27来解释“行不通”的理由。首先,我们假设客户端主机上的SSH服务已经正常启动,并且在TCP

Port 22位置正确运行,接着在服务器上启动SSH 客户端,我们假设SSH 客户端这次使用的是TCP Port 12345;然后,服务器上的SSH 客户端从端口 12345发出服务请求数据包给客户端主机的端口 22,而这个请求包一定可以成功送达客户端主机,因为服务器上的OUTPUT链并没有设置任何规则,但此时收到请求包的客户端主机,自然会回应数据包给这个SSH 客户端,不过,这个数据包将会从192.168.0.100的TCP Port 22回送到192.168.0.1的TCP Port 12345.

试问,服务器的INPUT链允许网络上的主机发送数据包到TCP Port 12345吗?如果不允许,那么这个应答包当然就无法正确地回送到服务器上,因此,我们从服务器上连接网络上其他主机的操作将无法成功。或许你会问,在INPUT链中将TCP Port 12345打开不就可以了!别忘了,我们在第1章曾经提过,客户端的应用程序所使用的端口是随机的(Random),所以我们不可能事先预知客户端的应用程序会使用哪个端口,也就不可能事先帮客户端把端口打开等着客户端来使用。该如何解决这个问题呢?请耐心地往下看。

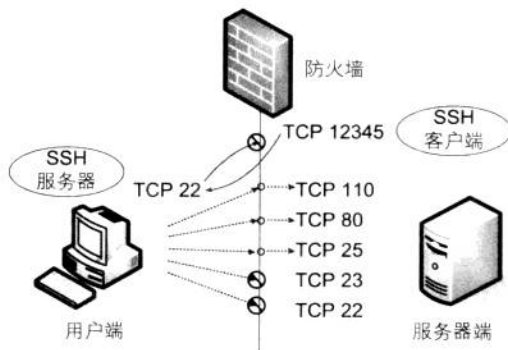


图2-27 服务器对网络上其他主机的访问操作

2.10.2 解决无法在防火墙主机上对外建立连接的问题

要解决无法在防火墙主机上对外建立连接的问题,其实比想象中简单多了。在ipchains防火墙的年代,要解决这个问题的做法是将1024以上的端口全部打开,因为我们可以确认一点,客户端的应用程序必定使用大于1024以上的端口,所以只要将1024以上的端口打开,这个问题就解决了,而且这种方法同样可以用在Netfilter/iptables防火墙上。但这种做法实在令人难以接受,因为部署防火墙的目的就是为了限制某些连接的建立,现在却将1024以上的端口全部打开,不就违反了防火墙的部署目的吗?

或许你无法接受以上做法,但在ipchains防火墙的年代,这是唯一的选择。到了Netfilter/iptables防火墙世界中,我们还有更简单、更安全的解决方案,那就是Netfilter/iptables的“连接跟踪”功能,这个功能由xt_state.ko模块所提供,这也是本章唯一介绍的模块。

在开始使用`xt_state.ko`模块之前,你必须先对`xt_state.ko`模块有所了解。首先,这个模块在`iptables`的使用叫做`state`,而其在`Netfilter`机制中是以`xt_state.ko`文件存在,在此我们称其为`state`模块。另一个很重要的概念你一定要确切了解,否则就无法发挥`Netfilter`机制的过滤器功能,那就是`state`模块定义的数据包连接状态。要知道在标准TCP/IP模型中,连接状态共分为十二种,但在`state`模块的描述中只有四种,分别是`ESTABLISHED`、`NEW`、`RELATED`及`INVALID`,切勿将这四种状态与TCP/IP的十二种状态混为一谈,因为这是两种完全不相干的定义。

例如,在TCP/IP标准的定义中,UDP及ICMP数据包是没有连接状态的,但在`state`模块的定义中,任何数据包都有连接状态。接下来首先简单介绍`state`模块定义的四种连接状态。

1. ESTABLISHED

在刚才的实验中,客户端对服务器提出服务请求时,服务器要等很久才会应答客户端,而且应用程序无法从防火墙主机上对外建立连接;下面以图2-28为例来解释如何处理这些问题。

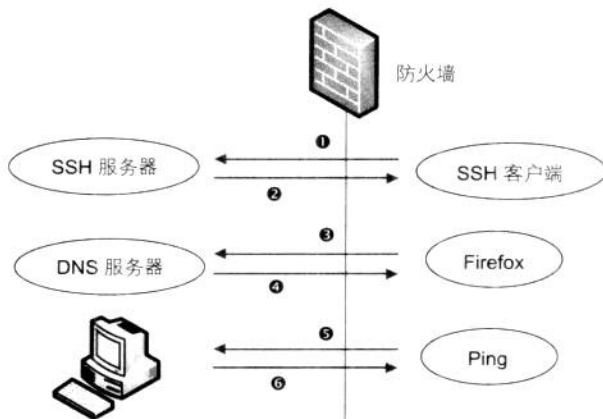


图2-28 ESTABLISHED状态解释

ESTABLISHED状态与TCP、UDP及ICMP协议的关系

以图2-28为例来说明`ESTABLISHED`状态与TCP、UDP及ICMP协议的关系,假设图的右边为一台设置为单机防火墙的主机,并且我们在该主机上分别进行以下三个实验。

● 与TCP包的关系

首先在防火墙主机上运行SSH 客户端,并对网络上的SSH服务提出服务请求,而这时送出的第一个数据包就是服务请求数据包,如果这个数据包能够“成功穿过防火墙”,那么,接下来不管是①②哪个方向的数据包,其状态都会是`ESTABLISHED`。

● 与UDP包的关系

假设我们在防火墙主机上使用Firefox应用程序来浏览网页，而浏览网页的操作则需要名称解析服务的帮助，才能顺利浏览到网页，因此Firefox会发送一个UDP数据包给DNS服务器，以请求名称解析服务；如果这个数据包能够“成功穿过防火墙”，那么，接下来不管是③④哪个方向上的数据包，其状态都会是ESTABLISHED。

● 与ICMP包的关系

假设我们在防火墙主机上使用Ping命令来检测网络上其他主机，如果Ping命令所发送的第一个ICMP包可以“成功穿过防火墙”，那么接下来不管是⑤⑥哪个方向上的数据包，其状态都会是ESTABLISHED。

由以上解释可以得知，只要数据包能够成功穿过防火墙，那么之后的所有数据包(包括反向的所有数据包)，其状态都会是ESTABLISHED。

为什么服务器要等很久才会应答客户端

这个问题的关键在于Linux主机上某些服务在接收到客户端的服务请求时，会将客户端的IP地址送到DNS进行反向名称解析。以图2-29为例，当SSH客户端对SSHD提出服务请求时①，SSHD即将SSH客户端所在主机的IP地址送至DNS服务器执行反向名称解析②，但这是有点像第2.10.2一节中所讨论的情况。这时数据包③一定能够成功发送给DNS服务器，因为防火墙的OUTPUT链并没有设置任何参数，而DNS服务器在收到这个数据包之后，当然也会回应④。不过，这个数据包将会因匹配到INPUT链的默认策略而遭到丢弃，这时，SSHD会因为等不到DNS服务器的应答，而不断重复名称解析的操作⑤，这个重复的操作要一直等到请求超时为止，此后SSHD才会应答给SSH客户端⑥，以上就是为什么服务器要等很久才会应答给客户端的原因。

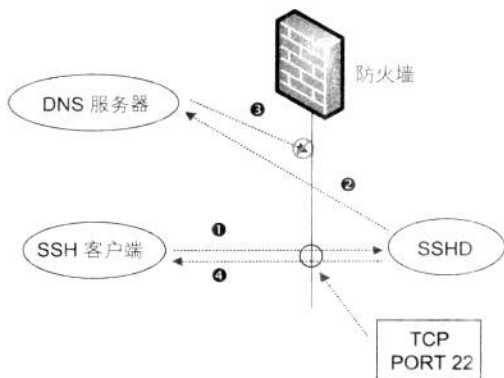


图2-29 ESTABLISHED状态应用示例

了解了这个原理之后，你是否已经想出解决办法呢？其实方法很简单，当SSHD发送数据包给DNS服务器时，标注为②的数据包已经成功穿过防火墙，那么，接下来SSHD与DNS服务器“在这条连接上”的所有数据包及所有反向的数据包不就是ESTABLISHED状态的数据包吗？因此，我们只要运用state模块去判断ESTABLISHED状态的数据包，并让ESTABLISHED状态的数据包可以成功返回，就可以解决问题了。

解决应用程序无法从防火墙主机上对外建立新连接的问题

这个问题其实与前一个问题是完全相同的，解决方法一样，只要运用state模块去判断ESTABLISHED状态的数据包，并让ESTABLISHED状态的数据包可以成功返回，问题就轻松解决了。

在了解了数据包ESTABLISHED状态的含义之后，你是否迫不及待想试试这个功能强大的模块呢？接下来就列举一个例子，使用state模块来匹配TCP协议下ESTABLISHED状态的数据包。

```
iptables -A INPUT -p tcp -m state --state ESTABLISHED -j ACCEPT
```

语法	说明
-p tcp	本次所匹配的是tcp协议数据包
-m state	本规则需要引用state模块
--state ESTABLISHED	给state模块的参数，告知所要匹配的是ESTABLISHED状态的数据包
-j ACCEPT	只有符合以上两个条件的数据包才能进入

要解决“单机防火墙示例”中所遇到的所有问题，我们只需要把本示例的规则加入INPUT链之中即可，至于要加在第几行，可是一门学问了。后面将进行完整详细的说明，现阶段就请你先加入到最后一行即可。

2. NEW

在了解ESTABLISHED这个重要的状态之后，接下来要解释的NEW也是一个非常重要的状态，因此请务必牢记。以图2-30为例来解释什么是NEW状态，首先我们要知道NEW与协议完全无关，其所指的是每一条连接中的第一个数据包。如图所示，客户端主机使用TCP Port 50 000，连接服务器端TCP Port 22，而这一条连接中的第一个数据包其状态就是NEW，其他又如客户端使用TCP Port 50 001，连接到服务器端的TCP Port 22；客户端的UDP Port 50 000，连接到服务器端的UDP Port 53，还有客户端对服务器端送出的第一个icmp包，以上这些例子的第一个数据包状态都是NEW。

或许你会觉得奇怪，为什么去匹配连接中的第一个数据包？其实这个操作在设置“高级防火墙规则”时，是很重要的一个状态，但因为在高级防火墙规则才会用到，所以现阶段并不打算介绍NEW状态的应用，第3.1.2一节的第8部分将会有完整且详细的说明。

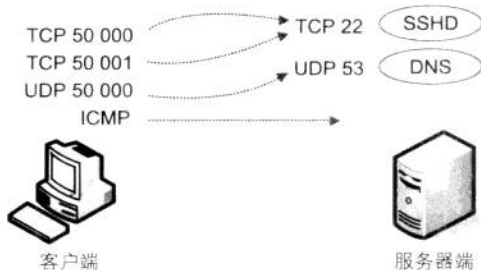


图2-30 NEW的状态解释

3. RELATED

另一个重要的状态是RELATED，这个状态在“高级防火墙规则”中也是很重要的，但现阶段不介绍RELATED状态的应用，在第3.1.2一节的第8部分将会有完整且详细的说明。不过，我们还是可以先大略了解什么是RELATED状态。

你应该有在Windows平台上使用tracert这个命令的经验吧！图2-31即为tracert命令执行的结果，而这个工具的目的就是让我们去检测两台主机之间，总共间隔了多少个路由器。但你是否曾经想过这个工具的工作原理？其实tracert工具的工作原理相当简单，首先我们得从IP包头中的TTL值谈起，这个TTL值是指数据包在网络上所能存活的时间，在早期是以秒为单位，不过，现在则改为“所能跨越的路由器数量”。

```
C:\>tracert 211.22.34.78
Tracing route to tp-s6-cl2r1.router.hinet.net [211.22.34.78]
over a maximum of 30 hops:
  1  <1 ms    <1 ms    <1 ms    10.0.1.254
  2  1 ms     <1 ms    <1 ms    10.10.15.1
  3  1 ms     <1 ms    <1 ms    10.10.12.27
  4  24 ms    24 ms    27 ms    61-219-23-254.HINET-IP.hinet.net [61.219.23.254]
  5  24 ms    23 ms    24 ms    tp-s2-c76r5.router.hinet.net [168.91.84.202]
  6  23 ms    37 ms    24 ms    tp-s2-cl2r1.router.hinet.net [211.22.34.78]
Trace complete.
```

图2-31 tracert命令执行的结果

下面以图2-32为例来看看tracert工具是如何查询图中最右边与最左边的主机，且总共间隔了多少个路由器？首先tracert工具会发送第一个数据包，这个数据包的目的端IP就是最左边主机的IP，并且可以将这个数据包的TTL值设定为1。接着，这个数据包就被送到第一个路由器①，而第一个路由器在收到这个数据包之后，即会将数据包内的TTL值减1，因此，这个数据包的TTL值变为0，这个值代表数据包生命周期已尽，所以第一个路由器即会丢弃这个数据包，并且回送一个ICMP Type 11(Time to live exceeded)的数据包给包的发送端主机，告知“你发送的数据包因生命周期已尽，故已遭到丢弃”，如此tracert工具就可从这个ICMP数据包得知第一个路由器的IP地址。

tracert接着会送出第二个数据包，不过，这个数据包的TTL值会特意设置为2，接着，第二个数据包会被送到第一个路由器①，第一个路由器收到这个数据包之后，会将其TTL值减1，这是数据包内的TTL值将会变为1，因为TTL值不为0，故其生命周期未尽，因此第一个路由器会将这个数据包转发给第二个路由器②。不过，当第二个路由器把数据包内的TTL减1之后，这个数据包内的TTL值即为0，代表这个数据包的生命周期已尽，因此第二个路由器即会丢弃这个数据包，并且回送一个ICMP Type 11(Time to live exceeded)的数据包给发送端主

机，告知“你所发送的数据包因生命周期已尽，故已遭到丢弃”。如此一来，tracert工具就可从这个ICMP数据包得知第二个路由器的IP地址。

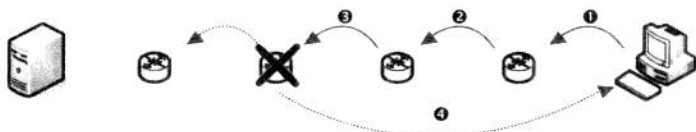


图2-32 RELATED的状态解释

紧接着，tracert会送出第三个数据包，不过，这个数据包的TTL值会特意设置为3，接着，这个数据包会被送到第一个路由器①，第一个路由器收到这个数据包之后，会将其TTL值减1，这是数据包内的TTL值将会变为2，因为TTL值不为0，故其生命周期未尽，因此第一个路由器会将这个数据包转发给第二个路由器②。接着，第二个路由器收到这个数据包之后，也会将数据包内的TTL值减1，这是数据包内的TTL值将会变为1，因为TTL值不为0，故其生命周期未尽，因此，第二个路由器会将数据包转发给第三个路由器③。接着，第三个路由器收到这个数据包之后再把TTL值减1，这时数据包内的TTL值将变为0，则数据包的生命周期已尽，因此，第三个路由器会丢弃这个数据包，并且回送一个ICMP Type 11(Time to live exceeded)的数据包给发送端主机④，告知“你所发送的数据包因生命周期已尽，故已遭到丢弃”。如此tracert工具就可以从这个ICMP包得知第三个路由器的IP地址是什么，最后tracert工具就可以通过收集数据来了解到整个路径上的路由器有几个，其IP地址是什么。

看完tracert的工作原理之后，我们回头来看哪种数据包的状态会是RELATED。以图2-32为例，标注③这个数据包的状态就是RELATED状态，RELATED状态的数据包其含义是指“被动产生的应答数据包，而且这个数据包不属于现在任何的连接”。例如，图2-32最右边的主机送出一个TCP协议的数据包，但因为这个数据包的生命周期已尽，而遭到第三个路由器丢弃。接着，第三个路由器会回送一个ICMP数据包给发送端，请注意！最右边的主机送出的是TCP包，这是一条“连接”，但第三个路由器所回送的是ICMP数据包，很明显，这绝对是另一条新的连接，而这个数据包之所以会产生，完全是因为第一条连接的存在才产生应答的数据包，且该数据包不属于任意现有的连接。

最后要澄清一个概念，在上例中以应答的“ICMP”数据包为例，但是RELATED状态的数据包与“协议”无关，“只要应答的数据包是因为本机先送出一个数据包而导致另一条连接的产生，那么这个新连接的所有数据包都属于RELATED状态的数据包”。

4. INVALID

INVALID虽然是一个容易了解的状态，但它同样是十分重要的。所谓INVALID状态是指

“状态不明的数据包”，也就是不属于ESTABLISHED、NEW及RELATED状态的数据包。凡是INVALID状态的数据包皆视为“恶意”的数据包，因此，请将所有INVALID状态的数据包丢弃掉；匹配INVALID状态数据包的方法如下例所示。

```
iptables -A INPUT -p all -m state --state INVALID -j DROP
```

再举个小例子，如果防火墙所需的规则如表2-4所示，请问我们应该将匹配INVALID状态数据包的规则插入到表2-5的第几条规则中？

表2-5 单机防火墙的规则列表

```
1.
2. iptables -P INPUT DROP
3. iptables -A INPUT -p tcp -d 192.168.0.1 --dport 25 -j ACCEPT
4. iptables -A INPUT -p tcp -d 192.168.0.1 --dport 80 -j ACCEPT
5. iptables -A INPUT -p tcp -d 192.168.0.1 --dport 110 -j ACCEPT
6. iptables -A INPUT -p tcp -s 192.168.0.200 -d 192.168.0.1 --dport 22 -j ACCEPT
7. iptables -A INPUT -p tcp -s 192.168.0.200 -d 192.168.0.1 --dport 23 -j ACCEPT
8.
```

最正确的答案是第一条，因为规则的匹配方式是由第一条开始往下逐一匹配的，如果我们将匹配INVALID状态数据包的规则放在第一的位置，那么数据包只要已进入INPUT Chain，就会先被匹配其是否为INVALID状态的数据包，如果是INVALID状态的数据包，那么马上就会将这个数据包丢弃掉，如此才能更周密地保护我们的服务器。

2.10.3 管理防火墙规则数据库的办法

到目前为止，相信你已具有足够的能力来构建一个简单的防火墙，但问题来了，该如何管理防火墙的“规则数据库”呢？因为当我们执行完iptables命令之后，规则即被存储在各个不同的链中，但每次计算机重新开机之后，这些原本在链中的规则将会全部消失，该不会每次重新开机之后，系统管理人员都要重新输入一次吧？因此，我们需要一个数据库来存储iptables的规则，以免在计算机重新开机之后，遗失掉所有的防火墙规则。

1. 使用iptables工具的规则数据库

其实这个问题，Netfilter/iptables的工程师早就帮我们设计好了，在修改完Netfilter的规则之后，可以使用“service iptables save”命令把所有Netfilter规则存储到“/etc/sysconfig/iptables”文件中。接着使用“chkconfig iptables on”命令设置系统，以便在重新开机之后，自动将“/etc/sysconfig/iptables”文件中的规则载入到Netfilter的各个链中，如此就不用担心防火墙规则消失的问题了。

这个功能看来相当贴心，但强烈建议你“绝对不要使用”！否则以后在管理这些规则，只能用“欲哭无泪”四个字来形容。试想，如果防火墙内有100条规则，而每条规则都有“-d 192.168.0.100”这个条件，如果有一天，192.168.0.100这台主机更改了IP，请问你该如何更改这些Netfilter数据库中的规则呢？全部删除再重新输入一次？还是使用iptables命令的“-R; replace”参数，一条条更新规则？其实这些都是非常低效的管理方式，有没有什么比较好的管理办法呢？

2. 使用Shell 脚本来管理规则数据库

在所有的“资料管理库”管理办法中，我认为最实用的办法就是使用Shell 脚本来记录防火墙的规则数据库，如果把之前示例的内容改写成Shell 脚本的样式，将如表2-6所示。

表2-6 使用Shell 脚本管理防火墙的规则

```
1. #/bin/bash
#=====< Set Variable >=====
2. IPT=/sbin/iptables
3. SERVER=192.168.0.1
4. PARTNER=192.168.0.200

#=====< Clear Original Rule >=====
5. iptables -t filter -F

#=====< Set INPUT RULE >=====
6. $IPT -A INPUT -p tcp -m state --state INVALID -j DROP
7. $IPT -A INPUT -p tcp -d $SERVER --dport 25 -j ACCEPT
8. $IPT -A INPUT -p tcp -d $SERVER --dport 80 -j ACCEPT
9. $IPT -A INPUT -p tcp -d $SERVER --dport 110 -j ACCEPT
10.$IPT -A INPUT -p tcp -s $PARTNER -d $SERVER --dport 22 -j ACCEPT
11.$IPT -A INPUT -p tcp -s $PARTNER -d $SERVER --dport 23 -j ACCEPT
12.$IPT -A INPUT -p tcp -m state --state ESTABLISHED,RELATED -j ACCEPT
```

把表2-6内的规则保存为firewall.sh文件，最后分别将文件的“Owner”及“Group”设置为root，并将权限设置为700。设置权限的目的在于确保除了root之外，任何人都无法使用这个文件，如此才能正确保护防火墙数据库的安全。

下面列出使用Shell 脚本来管理防火墙数据库的优缺点。

- 优点
 - 提升管理防火墙规则的简便性：

使用Shell 脚本的第一个好处就是不需要记忆太多iptables参数(例如-D、-R、-I之类)，即可快速方便地管理防火墙的规则。我们只需使用文本编辑器来修改Shell脚本的内容，然后再重新执行一次Shell 脚本即可。不过，当你在写防火墙的Shell 脚本时，别忘了加上表2-6的

第五行, 先把已经存在的旧规则清除掉, 然后再加上这次的新规则, 否则防火墙规则一定会“越加越多”。

- 可以在Shell脚本中使用变量:

稍早曾经假设过一个例子: “如果防火墙内有100条规则, 而每一条规则中都有‘-d 192.168.0.100’这个条件, 如果有一天, 192.168.0.100这台主机更改了IP, 请问, 该如何更改这些Netfilter数据库中的规则呢?” 其实, 如果是使用Shell脚本, 这个令人头痛的问题根本就不会存在, 怎么说呢? 在表2-6的Shell脚本中就可以得到答案了, 在其中的第2~4行设置了3个变量, 例如“SERVER=192.168.0.100”即为一个变量, 而用设置变量的优点在于, 如果Server的IP变更了, 我们只需要用文本编辑器将Shell脚本中的“SERVER=192.168.0.100”改为“SERVER=192.168.0.50”, 接着再重新执行一次Shell脚本, 防火墙中的规则就变更完毕了。

- 防火墙规则容易阅读:

由于iptables的规则语法与实际存在于各个链中的规则语法在写法上稍有差异, 因此, 如果链中的规则条件较多, 使用“iptables -t filter -L”命令来阅读防火墙规则实在不是一种很好的方式, 但如果是以Shell脚本的方式来记录防火墙规则, 因为Shell脚本所呈现出来的规则就是我们下发给iptables命令的语法, 所以阅读上就轻松多了, 特别是Shell脚本可以加“注释”, 如此将可以确保长时间后, 依然可以清楚了解当初设置防火墙规则的目的。

- 备份防火墙规则数据库极为方便:

如果以Shell脚本的方式来记录防火墙的规则数据库, 我们只需要将这个Shell脚本的文件备份起来即可, 如果哪天防火墙主机的硬盘损坏了, 只需要换个硬盘, 然后将系统安装起来, 最后再把Shell脚本重新复制回硬盘即可。

- 缺点

- 无法在开机后自动载入规则数据库:

使用Shell脚本来作为防火墙的规则数据库, 最麻烦的就是无法使用“chkconfig iptables on”方法来设置开机后自动载入规则数据库的内容。不过, 这也不是什么困难的问题, 因为Linux系统有一个类似DOS下的autoexec.bat, 这个文件为/etc/rc.d/rc.local, 如果我们的Shell脚本文件为/root/firewall.sh, 那么, 只需要在/etc/rc.d/rc.local文件中加入下列命令, 则系统在重新开机后, 就会自动执行firewall.sh, 当然其内的规则也就可以顺利地加入到各个不同的链中了。

```
/root/firewall.sh
```

2.11 使用filter机制来构建网关式防火墙

在前一节中，我们基本了解了单机防火墙构建的基础，接着以图2-33为例来构建一个简单的网关式防火墙。

假设10.0.1.100是因特网上的一台主机，且其上运行了SMTP、POP3及HTTP三项服务，由于我们尚未介绍nat机制，因此本示例先假设企业内所使用的IP是公网IP，其网段是192.168.0.0/24。在这个示例中，我们希望达到的环境需求如下：

- 192.168.0.200这台主机只能访问10.0.1.100主机的SMTP及POP3服务。
- 192.168.0.0/24网段上的其他主机只可以访问因特网上的DNS、SMTP、POP3、HTTP及HTTPS服务。
- 因特网上的主机不得访问企业内的任何主机。

有了明确地需求后，接下来就可以把它写成iptables的规则语法，如表2-7所示。这些规则语法较容易理解。因为与单机防火墙的设置原理差不多，在此仅简单介绍与表2-7有关的内容。

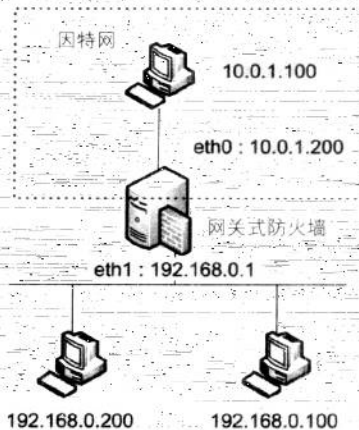


图2-33 网关式防火墙

表2-7 简单网关式防火墙的规则数据库

```
1. #/bin/bash
#-----< Set Variable-->-----
2. IPT=/sbin/iptables
3. MAIL_SRV=10.0.1.100
4. ACC_PC=192.168.0.200
#-----< Set Default Policy-->-----
5. $IPT -t filter -P INPUT DROP
6. $IPT -t filter -P FORWARD DROP
#-----< Clear Original Rule-->-----
7. $IPT -t filter -F
#-----< Set INPUT RULE-->-----
8. $IPT -A INPUT -p tcp -m state --state INVALID -j DROP
9. $IPT -A INPUT -p tcp -m state --state ESTABLISHED,RELATED -j ACCEPT
#-----< Set FORWARD RULE-->-----
10. $IPT -A FORWARD -i eth0 -o eth1 -m state --state INVALID -j DROP
11. $IPT -A FORWARD -i eth0 -o eth1 -m state --state ESTABLISHED,RELATED -j ACCEPT
12. $IPT -A FORWARD -i eth1 -o eth0 -p tcp -s $ACC_PC -d $MAIL_SRV --dport \
    25:110 -j ACCEPT
13. $IPT -A FORWARD -i eth1 -o eth0 -p all -s $ACC_PC -j DROP
14. $IPT -A FORWARD -i eth1 -o eth0 -p tcp --dport 25:110 -j ACCEPT
15. $IPT -A FORWARD -i eth1 -o eth0 -p tcp --dport 80:443 -j ACCEPT
16. $IPT -A FORWARD -i eth1 -o eth0 -p udp --dport 53 -j ACCEPT
```


- 第1行

设置Shell解释器的路径。

- 第2~4行

将未来可能会变动的信息或太长的命令简化成“变量”。

- 第5~6行

设置INPUT及FORWARD链的默认策略为DROP，其目的分别如下：

- 设置INPUT链的默认策略为DROP的目的：虽然本示例的主要目的是使用网关式防火墙来保护及限制企业内的主机，但请千万别忽略了防火墙自身的安全，否则

防火墙若被入侵了，整个企业的安全防护就形同虚设了。因此，我们拒绝了所有对防火墙的连接操作，但别忘了，因为我们有可能在防火墙主机上执行如软件更新等需要对外产生连接的操作，所以在此并没有将OUTPUT链的默认策略设置为DROP。

- 设置FORWARD链的默认策略为DROP的目的：因为本示例是以严格的方式来管制企业对外的连接，以及因特网对企业内的连接，因此，笔者将FORWARD链的默认策略设置为DROP，如此一来，企业内外就等于“断线”的状态。稍后，我们再从FORWARD链中逐一启用允许的对外连接通道即可。

- 第7行

清除原有的防火墙规则。

- 第8~9行

因为INPUT链的默认策略已设置为DROP，因此将导致对外建立连接时应答数据包无法正常穿过防火墙的问题，我们必须在INPUT链中设置ESTABLISHED及RELATED状态的数据包，才可以正常地返回。

- 第10行

将所有从因特网送到企业内部且状态是INVALID的数据包丢弃。

- 第11行

允许所有由企业内部对外建立连接时所产生的应答数据包正常返回企业内部。

- 第12~13行

设置192.168.0.200主机只能访问因特网上10.0.1.100这台主机的SMTP及POP3服务。

- 第14~16行

在192.168.0.0/24网段中的主机只能访问因特网上的SMTP、POP3、HTTP、HTTPS及

DNS服务。

相信你对以上的防火墙规则应该都可以轻松了解，以上示例中的单机防火墙及网关式防火墙都只是一个非常基础的防火墙，如果要构建企业级防火墙则绝对是不够的。不过，你无须担心，稍后章节将逐一介绍防火墙的高级功能，以便你能运用这些高级功能来构建企业级的防火墙。但你必须对这两个示例的内容及概念有全面的了解，这样才能顺利地学习后面章节。

2.12 Netfilter的NAT机制

NAT是Network Address Translation的缩写，意即“网络地址转换”。NAT机制可以应用在服务器端，也可以应用在客户端，当然其用意是不一样的。比方说，NAT应用在客户端的主要目的是隐藏客户端的IP，由此达到保护客户端主机免于遭受因特网的攻击行为，以及节省公用IP的使用量；而在服务器端的主要用途是保护服务器端主机在因特网上的安全。NAT种类繁多，有一对多、多对多、一对一和NATP等四种，而这四种NAT架构将在本章中逐一介绍，请拭目以待。

2.12.1 IP网段的划分

你一定很清楚在因特网上识别计算机主机的方式就是IP地址，而IP地址是由一组二进制的数字所构成，其长度是32位，但长度32位的二进制数据实在是让人太难记忆了，于是有人想到把这一长串的二进制数字，分割为4组8位长度的二进制数字，接着再将其转换成人们容易记忆的十进制数字，如192.168.1.10。然而，这些数字如果没有适当的规划，使用起来也会有很大的困难，因此，IANA将IP地址划分为三类，分别是A类、B类及C类，而其范围如表2-8所示。

表2-8 IP地址的分类

类别	范围	子网掩码
A	1.*.*.*~127.*.*.*	255.0.0.0
B	128.*.*.*~191.*.*.*	255.255.0.0
C	192.*.*.*~223.*.*.*	255.255.255.0

2.12.2 私有IP

因为IP是由32位的二进制数字所组成的，所以其能代表的主机数量是有限的，而这个上限是 $2^{32}=4\,294\,967\,296$ 台主机，这个数字看起来相当惊人，有42亿多台计算机！但现在地球上需要设置IP的设备(计算机、路由器、网络设备、PDA、无线网络电话等)很多，这42亿多个IP真的够用吗？未来不会再有增长吗？事实上，这42亿的数字是绝对不够用的。

当初IANA在划分因特网IP时，早就考虑到这个问题，因此IANA分别在A类、B类及C类网段中，各保留了一段私有IP区间，这些区段的划分请参考表2-9。何谓私有IP？私有IP是指这些IP只能在企业内部使用，而无法应用在因特网上。也就是说，因特网上的路由器如果看到这些私有IP，就会将之丢弃掉，这有点像企业内每个成员都会有一组分机号码，而这个分机号码在企业内部是可以彼此互通的，但你的朋友不可能用他家里的电话直接拨这组分机号码给你，两者是一样的原理。私有IP只能让企业内的计算机彼此互通，却无法让企业内的计算机与因特网上的计算机彼此互通。

表2-9 私有IP的分布区域

类别	范围	子网掩码
A	10.0.0.0~10.255.255.255	255.0.0.0
B	172.16.0.0~172.31.255.255	255.255.0.0
C	192.168.0.0~192.168.255.255	255.255.255.0

2.12.3 NAT

私有IP是无法在因特网上使用的，而如今普遍使用的宽带网络(ADSL)最多所能提供给用户的IP为16个，最少则为一个，万一企业内部有50台计算机要同时连接上因特网，该如何解决呢？

额外加一个路由器

或许你会想到额外加一个路由器来解决这个问题，但加一个路由器真的可以解决这个问题吗？以图2-34来说明。在图中，我们在192.168.0.0/24这个私有IP与10.0.1.0/24这个公网IP区间加了一个路由器，而这台ADSL只有3个公网IP，因此，我们把一个公网IP设置在路由器的接口上(假设10.0.1.200是公网IP)。另外，私有IP区段上的两台主机的默认网关则是指向路由器的另一个接口。

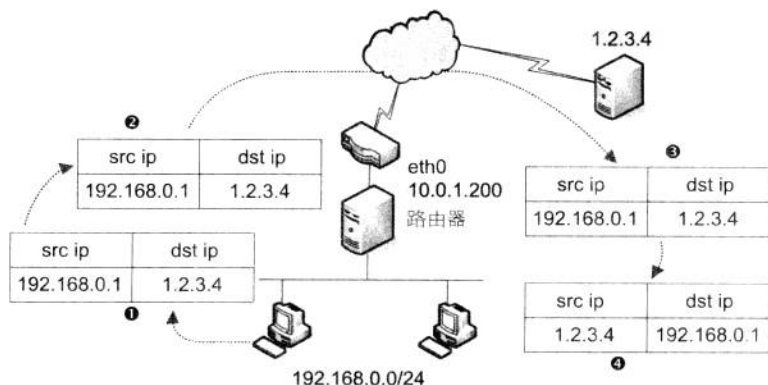


图2-34 NAT原理(一)

首先, 192.168.0.1 这台主机尝试访问因特网上的 1.2.3.4 主机, 因此, 192.168.0.1 主机会发送一个请求数据包到默认网关, 这个数据包内的“来源”及“目的”IP 分别如❶所示, 来源IP为192.168.0.1, 目的IP为1.2.3.4, 由于是路由器的缘故, 所以这个数据包的来源及目的IP并不会有任何的改变。接着, 路由器会将该数据包原封不动地发送给1.2.3.4 这台主机, 当1.2.3.4 主机收到这个数据包时, 该数据包内的来源及目的IP就如❷所示, 或许你会觉得奇怪, 私有IP不是不能在因特网上传输吗? 为什么1.2.3.4 还可以收到这个数据包呢?

为了提高因特网上的数据包的传输效率, 因特网上的路由器通常不检查数据包中的“来源端IP”, 而只会看目的端的IP, 所以192.168.0.1 主机所发送的数据包, 是可以被正常传输到1.2.3.4 主机上。而当1.2.3.4 主机接到这个数据包之后, 其会认定数据包的发送者为192.168.0.1, 因此, 1.2.3.4 主机即会应答192.168.0.1 的请求, 所以就会送了一个数据包给来源端, 这个数据包就如❸所示, 来源端IP为1.2.3.4, 目的端IP为192.168.0.1。但这下麻烦了, 因为数据包中的“目的端IP”为私有IP, 因此, 这个数据包是不可能被送回192.168.0.1 的, 因为目的端的IP为私有IP而遭到因特网上路由器的丢弃, 外加路由器的方式是不能解决当前问题的。

通用NAT

这个问题的正确解决办法是使用NAT, 只要通过NAT的机制, 就可以让成千上万台计算机同时通过一个公网IP来连上因特网。乍听之下好像很神奇, 其实原理很简单, 以图2-35为例来说明NAT机制。假设图中192.168.0.1 要通过NAT主机来访问因特网上的1.2.3.4 主机, 此外, 我们假设NAT主机上所设定的10.0.1.200 为公网IP, 而NAT后方所设置的是192.168.0.0/24 的私有IP网段。

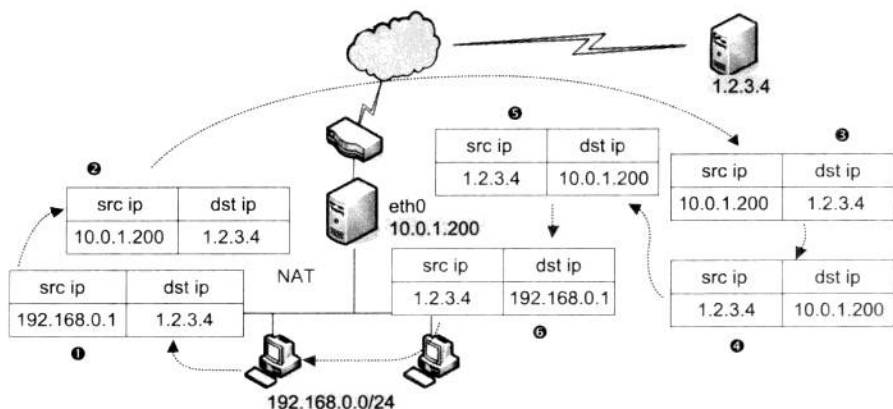


图2-35 NAT原理(二)

首先, 192.168.0.1 会发送数据包给 1.2.3.4 这台主机, 而这个数据包的内容就如 ❶ 所示, 其来源端 IP 为 192.168.0.1, 目的端 IP 为 1.2.3.4, 当然, 这个数据包一定会交给默认的网关来处理, 也就是交给 NAT 主机来处理, 接着 NAT 主机将这个数据包内的“来源端 IP”改成 NAT 主机上的公网 IP, 如 ❷ 所示, 如此来源及目的 IP 都会变成公网 IP, 同时, NAT 主机将这个数据包的信息记录下来。接着, 这个数据包当然可以被发送到 1.2.3.4 主机上 ❸, 而主机 1.2.3.4 当然也会应答给这个发送端, 这个应答的数据包就如同 ❹ 所示, 但这次应答的目的端主机为 10.0.1.200 这个公网 IP, 而不会是 192.168.0.1 这个私有 IP。如此一来, 这个数据包当然就可以被送回 NAT 主机 ❺, 当 NAT 主机收到这个数据包后, 就可以从之前记录下来的信息中找到当初 NAT 主机把 192.168.0.1 转成 10.0.1.200 的记录, 因此, NAT 主机就会把这个数据包内的目的端 IP 改成 192.168.0.1 ❻。

从以上流程可以看到, 当 192.168.0.1 主机发送数据包给 1.2.3.4 主机之后, 其所希望得到的是 1.2.3.4 应答给 192.168.0.1, 如果能够符合这个条件, 192.168.0.1 当然就可以正常访问 1.2.3.4 主机上的资源, 也就代表私有 IP 的主机真的可以通过 NAT 主机去访问因特网上的资源。

2.12.4 数据包传输方向与 SNAT 及 DNAT 的关系

在了解 NAT 的原理之后, 有两个很重要的名词请一定要了解且牢记, 那就是 SNAT 及 DNAT, 下面将进行简单说明。请注意图 2-35 中 ❶ 到 ❷ 的步骤, 谁被改变了? 是 Source IP, 对吧? 变更 Source IP 的机制称为 SNAT, 而在 ❺ 到 ❻ 的步骤中, Destination IP 被改变了, 因此变更 Destination IP 的机制就叫做 DNAT。

NAT 机制有很多种, 有一对多, 多对多, 一对一及 NAT, 基本上, 不管是哪一种

NAT，都是由SNAT及DNAT所共同搭配出来的，因此，如果你可以完全了解NAT的原理，那么在稍后学习这几种不同的NAT机制时，就容易理解和上手。

以图2-36为例来进一步解说SNAT及DNAT机制。图中这台主机共安装了两块网卡，假设有一个数据包由eth1接口送入，并从eth0接口离开，那么这个数据包将流经三个不同的机制，其先后顺序分别为PREROUTING、路由表(Routing Table)及POSTROUTING机制。此外，如果数据包是由eth0接口送入并由eth1接口离开，数据包同样会流经三个不同的机制，其先后顺序分别为PREROUTING、路由表及POSTROUTING机制。由此可以得出一个结论，PREROUTING、路由表及POSTROUTING并没有规定在特定的一侧，关键在于数据包的流向。

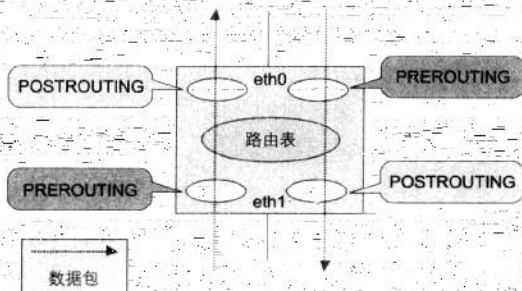


图2-36 数据包流向与SNAT及DNAT的关系

究竟PREROUTING和POSTROUTING是什么呢？从图2-37可以看到nat表内所有的链，以及一个名为PREROUTING及名为POSTROUTING的链，而图2-36中所谈到的PREROUTING及POSTROUTING就是指这两个链。此外，我们还可以看到一个名为OUTPUT链的机制，不过请你注意，这个OUTPUT链与filter机制中的OUTPUT链可是各自独立、毫无关系的两个机制。

```
[root@test ~]# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target prot opt source destination
Chain POSTROUTING (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination=
[root@test ~]#
```

图2-37 nat表的结构

前面的解说中希望以一种比较容易思考及记忆的方式来帮助你快速记忆。接下来以图2-38为例介绍NAT的完整结构，并解释每个链的用途。

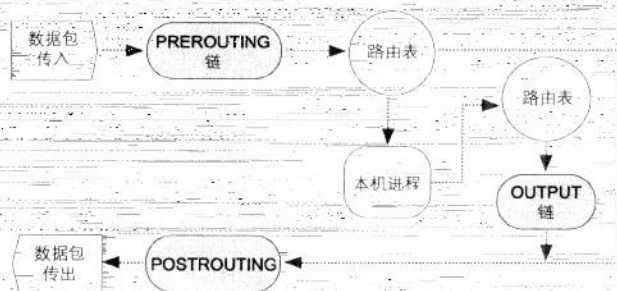


图2-38 NAT的结构

● PREROUTING

NAT有很多不同的种类，但不管是哪一种NAT，都是由SNAT及DNAT搭配组合而成的，因此，当我们下发“规则”要去修改数据包的“Destination IP”时，请将该规则放在PREROUTING链中，因为PREROUTING链的功能在于执行DNAT的任务。

此外，可以从图2-38看到，PREROUTING链的位置在整个NAT机制的最前面。因此在执行DNAT时，几乎是数据包一旦进入NAT机制，数据包内的“Destination IP”即被修改，而这个“顺序”问题在整个防火墙上是非常重要的。

● POSTROUTING

与PREROUTING链不同的是，POSTROUTING链的任务是修改数据包内的“来源端IP”，也就是说，POSTROUTING链的功能在于执行SNAT的任务。此外注意POSTROUTING链的位置，POSTROUTING链位于整个NAT机制的最末端，因此当我们执行SNAT操作时，Source IP是在整个NAT机制的最末端才会被修改的。

● OUTPUT

最后以图2-38为例来介绍NAT机制中的OUTPUT链，不过请你先看图2-38来回答下列问题：

(1) 如果网络上有一个数据包是要送给本机进程，请问在这个数据包送达进程之前，我们可以对该数据包执行DNAT的操作吗？

(2) 如果本机进程生成了一个数据包往外发送，请问在这个数据包离开本机之前，我们可以对这个数据包执行SNAT的操作吗？

(3) 如果本机进程生成了一个数据包往外发送，请问在这个数据包离开本机之前，我们可以对这个数据包执行DNAT的操作吗？

你心中是否已经有答案了呢？下列是笔者的分析：

(1) 从图2-38可以看到，如果一个数据包要送给本机进程，这个数据包进入系统之后，

就会先进入PREROUTING链，所以我们可以PREROUTING链内执行DNAT任务，接着数据包才会送到路由表进行路由判断，最后抵达进程。第1题的答案是“可以”。

(2) 当本机进程生成数据包并向外发送时，该数据包会先交给路由表来判别路由，接着数据包进入OUTPUT链，最后进入POSTROUTING链，接着送离本机。因此，如果我们要改变该数据包的Source IP，可以在POSTROUTING链中执行SNAT的操作。第2题的答案是“可以”。

(3) 当本机进程生成数据包并向外发送时，这个数据包会先交给路由表来判断路由，接着数据包进入OUTPUT链，最后进入POSTROUTING链，然后送离本机。因此，这个数据包绝不可能送到PREROUTING链内，即使你在PREROUTING链内写再多的规则，都不可能改变这个数据包的Destination IP。为了解决这个问题，NAT特别设计了一个名为OUTPUT链的机制，而这个链的功能就是执行DNAT的任务，其对象就是本机进程产生并要外送的这些数据包。因此，第3题的答案是“可以”，但DNAT的规则是要放在OUTPUT链之内。

2.12.5 NAT的分类

NAT会因其所要执行的任务而异，而在技术的应用上会有所差异，常见的NAT机制大概可分为“一对多NAT”、“多对多NAT”、“一对一NAT”及“NAPT(Network Address Port Translation)”四种。不过，你不用担心，只要对NAT的原理够了解，不管构建哪一种NAT都是很容易的，如果你对NAT的原理还不是很了解的话，建议你不妨再回顾一下前一节中介绍的内容，将NAT的原理及Netfilter的NAT机制好好地再研读一次。接下来将逐一介绍每种NAT的使用环境、构建方式及原理。

1. 一对多NAT

在这四种NAT机制中，使用率最高的就是一对多NAT。或许一对多NAT这个名词大家不太熟悉，但如果换个方式来形容它，或许你就豁然开朗。IP分享器有听过吧？其实IP分享器就是一种一对多的NAT机制，这种机制最主要有如下两项功能。

● 节省公网IP的使用量：

由于因特网发展速度惊人，网民数量飙升，因此，造成了因特网上的公网IP日益吃紧，即使企业申请的是T1数据专线，但要能拿到64个公网IP已经很困难了，更别说企业若申请ADSL上网，那能拿到的公网IP就更少了。为了使企业内所有人都可以同时上网，一对多NAT可说是一个很好的解决方案，而所谓的一对多NAT就是“让大家共用一个公网IP来上网”，如此，即使企业内有再多的使用者，也只需一个公网IP就够了，如此即可大量减少公网IP的消耗量。

● 隐藏企业内部主机的IP地址:

以图2-39为例,一般在规划企业网络时,企业内部的网段都会使用私有IP来当做企业内的网络地址,然后使用NAT的机制将私有IP转成公网IP再连上因特网,也因为企业内部是躲在NAT后方,因此,因特网使用者根本不知道原来某个公网IP后方藏着一群主机,即便知道了,因NAT后方通常都是使用私有IP,也提高了黑客攻击企业网络内部主机的难度。因此,一对多NAT机制对于企业网络在安全性的考虑上具有一定的帮助。

接下来以图2-39为例来构建一对多NAT的网络环境。首先在NAT主机上安装两块网卡,并在eth0接口上设置10.0.1.200这个公网IP,此外,在eth1接口上设置192.168.0.254这个私有IP,而企业内的主机都将是192.168.0.0/24网段上的主机,且其默认网关是192.168.0.254。

在设置NAT机制时,我们必须考虑数据包的进出。首先考虑企业内部发送数据包到因特网的情况,如果192.168.0.0/24网段的主机要访问因特网上的主机,其服务请求数据包内的Source IP必然是私有IP。如果让这个数据包直接送往因特网,这个访问操作必然会失败,因此,当数据包有企业内部送出时,我们必须通过NAT主机上的SNAT机制,将外送数据包内的Source IP改为NAT主机上的公网IP,如此才能让因特网上主机应答回来的数据包能顺利返回到NAT主机对外的公网IP上,这个SNAT的规则语法如下:

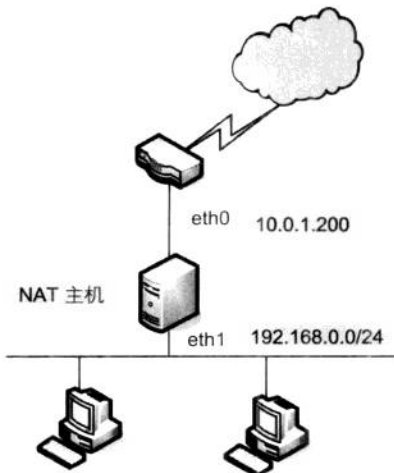


图2-39 一对多NAT

● 语法

```
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.0/24 \
-j SNAT --to 10.0.1.200
```

● 语法解释

● -t nat

选择表	选择所要使用的功能为Netfilter的NAT机制
-----	---------------------------

● -A POSTROUTING

选择功能	选择所要执行的操作为SNAT机制
------	------------------

- -o eth0

标明数据包的流向

前面章节曾提到，POSTROUTING及PREROUTING并没有固定在具体哪个方向，关键在于数据包的流向，因此在执行SNAT或DNAT操作时务必设置这项参数，否则iptables一定会告诉你“语法错误”。

- -s 192.168.0.0/24

设置私有IP网段

只有来自192.168.0.0/24网段的数据包才符合本规则

- -j SNAT --to 10.0.1.200

执行的操作

如果数据包符合以上所有条件，就将其Source IP的内容改为10.0.1.200，但请注意，这个IP必须为NAT主机对外的公网IP。

在将以上规则添加到POSTROUTING链后，192.168.0.0/24网段的主机就可以正常连接到因特网上了。你可能会觉得很奇怪，不是要考虑数据包的进出吗？而刚刚也才设置好数据包出去的方向，回来的方向都还没设置，为什么现在192.168.0.0/24网段的主机就可以正常连接上因特网？有这样的想法很正确的，不过，如果NAT机制每次都要写“双向”规则，那也未免太累了吧！所以Netfilter的程序员们早就帮我们考虑到这个问题，不管我们下达的是SNAT或DNAT的规则，NAT机制都会自动帮我们判别“另一个方向的应答数据包”，因此，我们只需要下达单一方向的规则即可。

看完以上内容，你是否觉得一对多NAT的设置真的很简单呢？最后再补充以下几点供你参考。

- 接口名称的问题：

因为本示例使用的是以太网，因此，在-o的参数后面加的是eth0；但如果你所使用的是拨号ADSL，那么接口名称将会是ppp0或pppX。-o参数后面的“接口名称”请务必设置为当时所使用的接口名称，如果你不确定接口名称，可以试着执行ifconfig命令，即可看到公网IP所使用的接口名称是什么。

- 如果公网IP不是固定的：

本示例所设置的公网IP是10.0.1.200，因此语法规则是“-j SNAT --to 10.0.1.200”。请注意，10.0.1.200在我们的Shell脚本中是被写死的，但如果你所使用的对外连接为拨号ADSL或Cable Modem的话，每次所拿到的公网IP都是不固定的，不就每次连上因特网之后，就要修改一次Shell脚本中的“-j SNAT --to Public IP”？这个问题倒不用担心，因为Netfilter程序员们早就帮我们想到，如果你对外的公网IP不固定，请将规则修改如下，如此一来，我们的规则就与公网IP无关了。

```
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.0/24 -j MASQUERADE
```


2. 多对多NAT

学习完一对多NAT机制之后，你会发现一对多NAT可能存在的一些问题，因为192.168.0.0/24这个私有IP网段的主机，都是转到10.0.1.200这个公网IP，然后再连上因特网的，因此，不管是企业内部哪一台主机对因特网上的主机进行访问，对因特网上的主机而言，都是10.0.1.200这台主机在进行访问，但这可能造成某些因特网上应用服务无法正常运行。例如，少数的网络游戏主机就无法接受来自同一个IP对其进行多次的连接操作，而最终可能导致相同一个NAT下的主机，只有一台主机可以玩某个网游，不过，这样的网络游戏已经越来越少了，游戏玩家们不必过于担心。

不过，对NAT机制而言，也有相对应的解决办法，只是看够不够完美罢了！多对多NAT或许可以减少这种问题发生的概率。什么是多对多NAT？要能够进行多对多NAT的先决条件是你必须拥有多个公网IP，而且这些公网IP必须是“连续”的，否则无法进行多对多NAT的操作。以图2-40为例，我们先将拥有的这五个公网IP设置在NAT主机的对外接口上，并运行以下命令：

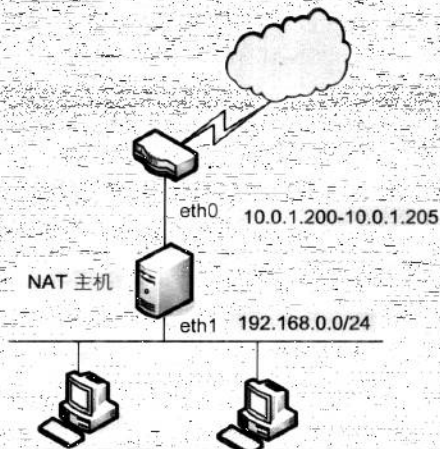


图2-40 多对多NAT

```
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.0/24 \
    -j SNAT --to 10.0.1.200-10.0.1.205
```

以上命令你一定会觉得很熟悉，几乎与一对多NAT的语法一样！唯一的差别只有最后所转换的公网IP部分有所改变而已。当以上的规则生效之后，企业内对外建立的第一条连接，将会带着第一个公网IP连上因特网，而第二条连接产生时，将会带着第二个公网IP连上因特网，当然第三条连接产生时，将会带着第三个公网IP连上因特网，如此不断地循环到了第六条连接时，将会轮回到第一个公网IP上，如此就可以降低相同公网IP连接到同一台因特网主机的概率。当然，公网IP的数量越多，发生冲突的概率就越低。

3. 一对一NAT

对NAT机制的主要目的是提供给对外服务主机一个更安全的运行环境。第1章中曾提过，把服务主机直接部署在因特网之上对服务主机本身的安全性而言是不好的。因此，为了给服务主机提供一个较安全的运行环境，我们可以将服务主机部署在企业内部。如图2-41所示，在NAT主机上安装三张网卡，并将服务主机部署在eth1接口的网段上，我们可以随意

设置一个私有IP的网段，例如将Web 服务器的IP设置为192.168.0.1，Mail 服务器的IP设置为192.168.0.2。

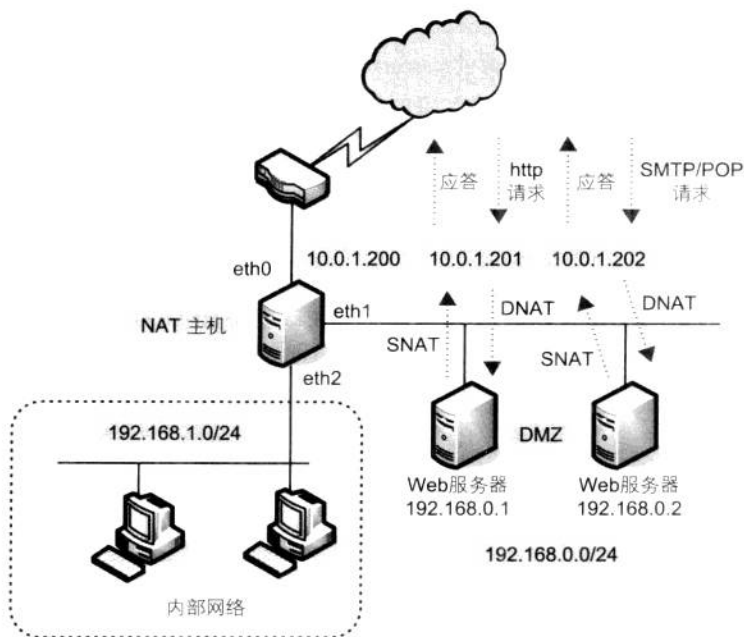


图2-41 一对一NAT

你或许会觉得奇怪，对外服务主机部署在私有IP的网段上，因特网使用者能够访问到其所需的服务吗？这个问题我们只需要通过一对一NAT的机制就可以轻松解决了。首先必须准备足够数量的公网IP，如图2-41所示，我们将10.0.1.200、10.0.1.201及10.0.1.202三个公网IP设置在eth0的接口上，并分配10.0.1.200给内部网络执行一对多NAT使用，10.0.1.201给Web 服务器使用，10.0.1.202给Mail 服务器使用；接着在NAT主机上设置：当有数据包要送给10.0.1.201这个IP时，就将数据包内的目的端IP改为192.168.0.1，如此这个数据包就会被转发给Web 服务器；但当Web 服务器应答给客户端时，我们就要特别处理这个数据包了，因为这个数据包是由Web 服务器应答的，而Web 服务器的IP是192.168.0.1，所以客户端收到的应答包将会是192.168.0.1主机所送出的，但是客户端期望收到的应该是由10.0.1.201主机所应答的数据包。因此，当数据包应答给客户端时，我们应当设法将数据包内的192.168.0.1改回10.0.1.201，如此才可以正常提供服务，这就是一对一NAT的原理。

接下来分析如何设置一对一NAT的运行环境，同样我们必须考虑到数据包的“进入”及“出去”两个部分，因此将其分开来讨论。

● 数据包的“进入”：

由于我们设置10.0.1.201为Web服务器的IP，必然将DNS内的WWW记录指向10.0.1.201这个IP。因此，因特网使用者在浏览网页时，所有网页的请求包势必都会送往10.0.1.201这个IP。接着，我们必须将这些数据包内的Destination IP改为192.168.0.1，这些数据包才能真正地送达到Web服务器上，以下是这个DNAT的规则语法。

```
iptables -t nat -A PREROUTING -i eth0 -d 10.0.1.201 \
-j DNAT --to 192.168.0.1
```

从这语法不难发现，这个语法与一对多NAT的语法相当类似，因为我们所运行的是DNAT，因此，必须将规则加入到PREROUTING链。另外需要注意的是，POSTROUTING链是以数据包“离开”的接口来标明数据包的流向，如-o eth0，但PREROUTING链则是以数据包“进入”的接口来标明数据包的流向，如-i eth0，务必要注意这一点。

此外，还必须特别注明，我们要处理的数据包是送给10.0.1.201这个IP的，因此将-d 10.0.1.201作为筛选条件之一。最后如果进入的数据包符合以上规则，就执行DNAT的任务，将数据包内的Destination IP改为192.168.0.1，如此就完成了一对一NAT机制。也就是说，如果现在有用因特网使用者来访问10.0.1.201这台主机，实际上，其所访问到的就是192.168.0.1这台主机。

● 数据包的“出去”：

或许你会问，前面不是才说过，不管我们执行的是SNAT或DNAT操作，NAT机制都会自动帮我们判别另一方向的应答数据包，因此，我们只需要下达单一方向的规则即可，为什么还需要考虑数据包出去的问题呢？没错！如果要让因特网使用者正常访问我们的Web服务器，只需要设置DNAT的部分即可，但是别忘了，我们会不会在Web服务器上访问因特网上的主机呢？如果不会，数据包“出去”的部分就大可不必理会；但如果会，就请你帮Web服务器设置一个出去的管道吧！这条SNAT的规则如下。设置好规则之后，每当你在Web服务器上访问因特网上的其他主机时，你的Web服务器就是以10.0.1.201这个公网IP在因特网上活动的。

```
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.1 \
-j SNAT --to 10.0.1.201
```

关于Mail服务器的部分，其原理与Web服务器是相同的，就留给你自己去思考了。

4. NAPT

在了解了一对一NAT之后，你或许会觉得设置很麻烦。笔者经常在教室里和同学说，有多个IP让你可以使用一对一NAT是幸福的，万一你工作的公司所使用的ADSL就只有一个公网IP，或者公网IP不够用时该怎么办？其实这个问题并没有想象中的那么难，我们只需要通过NAPT机制就可以轻松解决这个问题。什么是NAPT？NAPT(Network Address Port Translation)

以端口为单位来执行NAT任务。以图2-42为例，假设我们只有一个公网IP可以使用，因此，我们将这个宝贵的公网IP设置在eth0接口上，另外将对外服务的主机部署于192.168.0.0/24的网段上，分别使用192.168.0.1及192.168.0.2两个IP。接着来分析该如何设置这个NAPT。在这个示例中，我们只有两台对外服务的主机，分别是Web服务器及Mail服务器，下面首先进行归类。

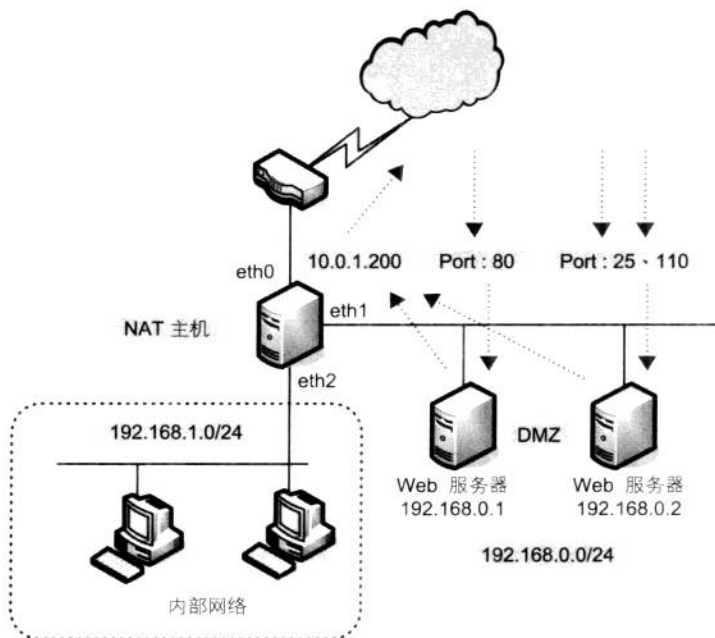


图2-42 NAPT

- Web 服务器：Web 服务器所需要使用到的端口只有TCP Port 80及TCP Port 443，因此，我们只需将访问10.0.1.200这个IP的TCP Port 80及TCP Port 443的数据包转发给192.168.0.1即可。
- Mail 服务器：Mail 服务器所需要使用到的端口只有TCP Port 25及TCP Port 110，因此，我们只需将访问10.0.1.200这个IP的TCP Port 25及TCP Port 110的数据包转发给192.168.0.2即可。

分析完以上事项之后，就可以着手编写这些规则了，规则如下。

● Web 服务器：

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 \
-j DNAT --to 192.168.0.1:80
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 443 \
-j DNAT --to 192.168.0.1:443
```

请注意, 因为以上规则中使用到--dport的参数, 因此, 我们必须额外添加-p tcp参数。当以上命令下达完毕后, 因特网上使用者就可以正常访问Web服务器了。

● Mail 服务器:

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 25 \
-j DNAT --to 192.168.0.2:25
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 110 \
-j DNAT --to 192.168.0.2:110
```

如Web服务器一样, 因为规则中使用--dport的参数, 因此需要加上-p tcp参数。当以上命令下达完毕后, 因特网使用者就可以正常访问Mail服务器了。

虽然以上命令就可以让因特网使用者正常访问Web及Mail的服务, 但还是要讨论一下有关数据包出去的部分, 如一对一NAT。如果你不需要在Web服务器或Mail服务器上访问其他因特网上的主机, 那么数据包出去的部分就不需要考虑; 但如果有这个需求, 就请你帮Web服务器及Mail服务器执行一个SNAT操作吧! 不过, 在此并不需要为192.168.0.0/24网段的每一台主机单独写一条SNAT规则, 因为在这个例子中不管你怎么写, 最后还是只能转到10.0.1.200这个公网IP上, 因此, 我们的规则只需要如下一行即可。

```
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.0/24 \
-j SNAT --to 10.0.1.200
```

2.12.6 NAT并非无所不能

看完以上的示例, 相信你已具备构建NAT环境的能力。但有一点必须先提醒你, NAT不是万能的, 并不是把服务于因特网上的主机搬到NAT后方, 它就安全了。事实上, 以一对一NAT为例, 当因特网上的使用者攻击一对一NAT的公网IP时, 这个攻击包是会被转发到NAT后方的服务主机上, 因此, 真正提供服务的主机并没有因为NAT的存在而得到任何安全性的提升。你一定会觉得很奇怪, 那为什么还要把服务主机放到NAT后方呢? 因为NAT要与Filter机制结合使用, 才能使服务主机得到真正的安全, 至于NAT与Filter机制该如何结合? 这部分将在2.14.1一节中给出非常清楚的解释, 请耐心等待, 因为还有很多基本知识需要你了解。

2.13 Netfilter的Mangle机制

Mangle是一个比较容易被人忽略的机制, 而且Mangle的使用机会并不是很多, 但如果需要时又不懂Mangle机制, 也是件很麻烦的事, 在此还是要说明一下Mangle机制。

当一个数据包“穿过”防火墙时, 我们可以通过Mangle的机制来修改数据包的内容, 至

于修改范围有多大？就得看Mangle机制里模块的支持程度。目前支持Mangle机制的模块并不多，所以我们的选择范围也不大，为什么需要修改路过防火墙数据包的内容呢？下面将用两个示例来说明Mangle目前能提供的功能及用途。

● 修改IP包头的TTL值：

由于每一种操作系统所生成的数据包的IP包头内TTL的默认值都不一样，因此我们只需要通过ping这个命令，就可检查出某一台主机所使用的操作系统是什么。为了增加黑客的入侵难度，可以修改所有由Linux主机所发送的数据包，将这些数据包内的TTL值改为128，让黑客误以为是Windows操作系统；也可以将所有由Windows操作系统所送出的数据包TTL值改为64，让黑客误以为是Linux系统。

● 修改IP包头的DSCP值或对特定的数据包设置特征：

在网络应用中，有时对某些特定通信协议会有特别的需求。例如，在网络电话的应用环境里，我们会希望“不管网络多么拥塞，VoIP的数据包都要能在不延迟的情况下发送出去”。因此就有了QOS(Quality of Service)机制，QOS机制可以让我们在有限的带宽中，有效分配不同的带宽给不同的协议来使用。如图2-43所示，总带宽是1.5MB/s，我们可以将其分为几部分：512KB/s给HTTP协议来使用、384KB/s给SMTP协议使用、256KB/s给Voice的数据包使用，最后剩下的带宽则留给未定义的协议。

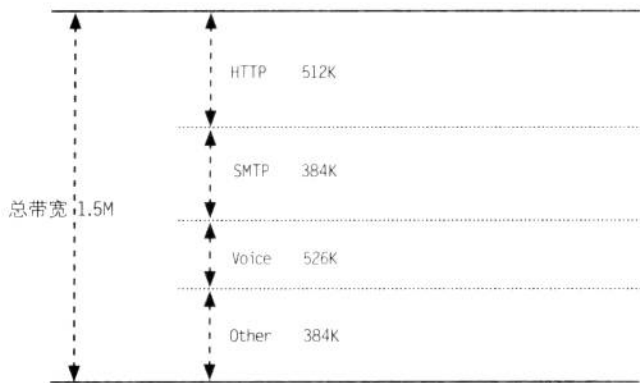


图2-43 QOS带宽分配图

QOS机制是由两个不同部分组成，其一为“数据包分类器”，其二为“带宽分配器”。如图2-44所示，当一个数据包进入“数据包分类器”之后，数据包即会被加以分类，被分类后的数据包接着进入“带宽分配器”，再由带宽分配器来决定各类数据包可以使用多少网络带宽，问题是如何分类数据包呢？我们至少可以运用Mangle机制中的两个模块来达到此目的，如下。

- 通过IP包内的DSCP值来分类:

我们可以通过Mangle机制来修改IP包内的DSCP值,例如,把DSCP值改为0000-01,接着在“带宽分配器”上设置,如果数据包内的DSCP值为0000-01的话,就给予64KB/s的带宽。

- 使用Mangle机制为数据包标示识别码:

Mangle机制可以为特定的数据包来标示不同的识别码,例如,如果数据包内的Source Port为80,就标示该数据包的识别码为80。接着在“带宽分配器”上设置,如果数据包的识别码为80,就给予512KB/s的带宽,如此也可以达到带宽分类的目的。



图2-44 QoS结构图

图2-45是Mangle机制的结构图,从图中不难发现,Mangle机制的链很多,但你必须要了解,不管是filter、nat、mangle或是稍后会提到的raw机制,它们的链都各自独立。因此,filter机制的INPUT链内容与Mangle机制的INPUT链内容绝对是不相同的,请务必认清这一点。而数据包在图2-45中的处理方式则与filter及NAT相同,因此如果想要改变本机进程所生成的数据包内的DSCP值,就必须将规则放置于OUTPUT链之中,或是POSTROUTING链之内都可以达到我们的目的,因为本机进程所产生的数据包,除了会流经OUTPUT链之外,也会经过POSTROUTING链,因此,这两个链都可供使用。

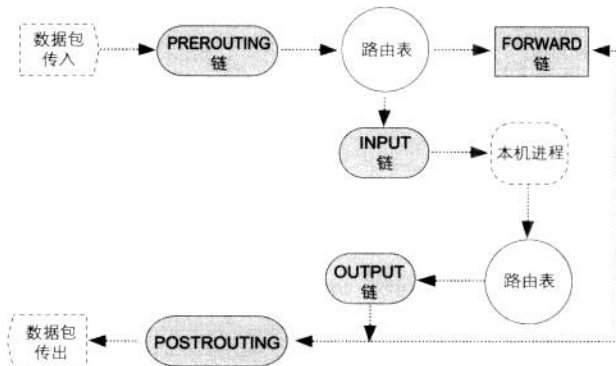


图2-45 Mangle的结构

最后以“修改数据包的DSCP值”为例进行讨论。我们希望在QoS机制中将SSH协议的数据包设置为较为优先传输的数据包,因此使用Mangle机制来修改数据包内的DSCP值。在尚未修改DSCP值之前,先抓取几个数据包来分析,图2-46即为数据包IP包头的内容,其中表示①的位置就是DSCP值,默认值为0000-00。

```

# Frame 1 (74 bytes on wire, 74 bytes captured)
# Ethernet II, Src: 00:0f:b0:93:b2:7d, Dst: 00:02:b3:0c:2
# Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10)
  version: 4
  Header length: 20 bytes
# Differentiated Services Field: 0x00 (DSCP 0x00: Default)
  0000 00.. = Differentiated Services Codepoint: Default
  .... 0.. = ECN-Capable Transport (ECT): 0
  .... 0.. = ECN-CE: 0
  Total Length: 60
  Identification: 0x2ee3 (12003)

```

图2-46 Linux系统默认的DSCP值

接着在本机使用Mangle机制修改本机进程所产生的数据包，并且设置所要修改的数据包为SSH协议的数据包，而修改后的DSCP值为43，其规则如下。

```
iptables -t mangle -A OUTPUT -p tcp --dport 22 -j DSCP --set-dscp 43
```

图2-47为使用Mangle机制修改后的数据包，并且我们从标示②的位置看到DSCP值已被修改为1010-11，而1010-11的十进制刚好就是43，以上即可证明Mangle机制真的可以修改数据包的内容。

```

# Frame 20 (74 bytes on wire, 74 bytes captured)
# Ethernet II, Src: 00:02:b3:0c:23:1b, Dst: 00:0f:b0:93:b2
# Internet Protocol, Src Addr: 192.168.2.11 (192.168.2.11)
  version: 4
  Header length: 20 bytes
# Differentiated Services Field: 0xac (DSCP 0x2b: Unknown)
  1010 11.. = Differentiated Services Codepoint: Unknown
  .... 0.. = ECN-Capable Transport (ECT): 0
  .... 0.. = ECN-CE: 0
  Total Length: 60
  Identification: 0x7423 (29731)

```

图2-47 修改后的DSCP值

2.14 Netfilter的raw机制

RAW是目前Netfilter发展历程中的最后一个表，不过raw表的功能会与一个被称为“连接跟踪”的机制有关。连接跟踪的机制在第3章中才会提到，因此我们将在连接跟踪机制介绍完毕之后，再介绍raw表的功能。

Netfilter的完整结构

图2-48为Netfilter的完整结构图，从图中可以清楚看到整个Netfilter的完整结构，这个结构共由四个不同的机制组合而成，分别是filter、nat、mangle及raw四个表，而每个机制所含有的链都不一样，图2-48可以帮助我们了解每个机制与机制之间的关系。此外，请务必熟记

链与链之间的先后顺序, 因为“顺序”在防火墙的规则设计上时非常重要的, 无论如何, 请你都把图2-48全部记住。

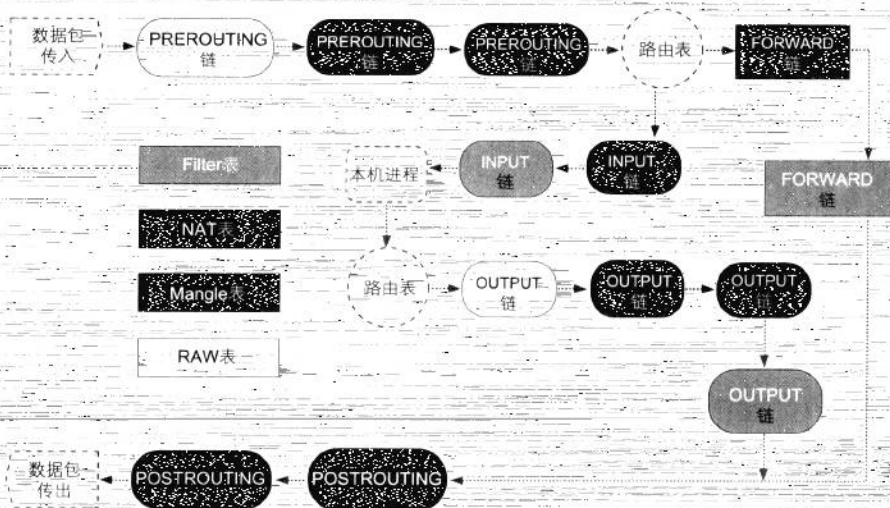


图2-48 Netfilter的完整机构

如果你还是无法了解这个结构顺序的重要性, 可以通过图2-49来了解“顺序”上的重要性。图2-49是一个一对一NAT的结构, 假设10.0.1.200、10.0.1.201及10.0.1.202都是公网IP, 并且将10.0.1.201对应到192.168.0.1, 也就是我们的Web服务器; 接着再假设有一黑客其IP为192.168.10.12, 该黑客目前正在攻击10.0.1.201这个IP, 我们稍早曾经说过“NAT并非万能的”, NAT无法保护我们对外公开的服务主机, 因为当有任何的攻击数据包送到10.0.1.201这个IP时, NAT也会把攻击数据包转发到真正的Web服务器上, 因此, 必须通过filter机制来保护我们的Web服务器, 问题是这个filter的语法该如何下达呢? 笔者将规则改写为如下形式。

```
iptables -t filter -A XXX -s 192.168.10.12 -d ww.xx.yy.zz -j DROP
```

请试着思考以下两个问题:

- 规则中XXX应该设置为哪个链?
- 规则中的ww.xx.yy.zz应该设置为10.0.1.201还是192.168.0.1呢?

要真正了解这两个问题, 就得从Netfilter的结构图来着手, 以图2-50为例, 当黑客发出攻击包时, 这个包的Destination IP为10.0.1.201。因此, 当数据包送达防火墙时①, 其Destination IP应该是10.0.1.201, 但当这个数据包经过NAT的PREROUTING链之后, 由于我们执行DNAT操作, 因此数据包到达②的位置时, 其Destination IP已被改变成192.168.0.1。当数据包进入路由表判断后, 数据包即会被转送到FORWARD链, 而由于数据包到达FORWARD

链之前，其Destination IP已被改变成192.168.0.1了，因此在FORWARD链中根本不可能看到10.0.1.201这个IP(DNAT机制在FORWARD链之前)。

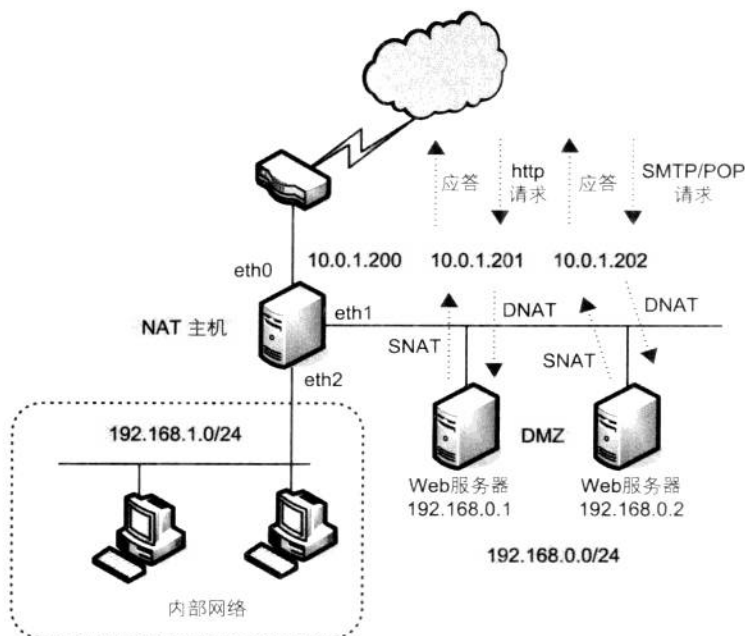


图2-49 一对一-NAT

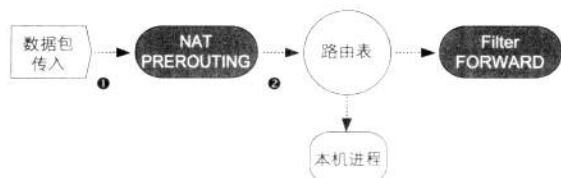
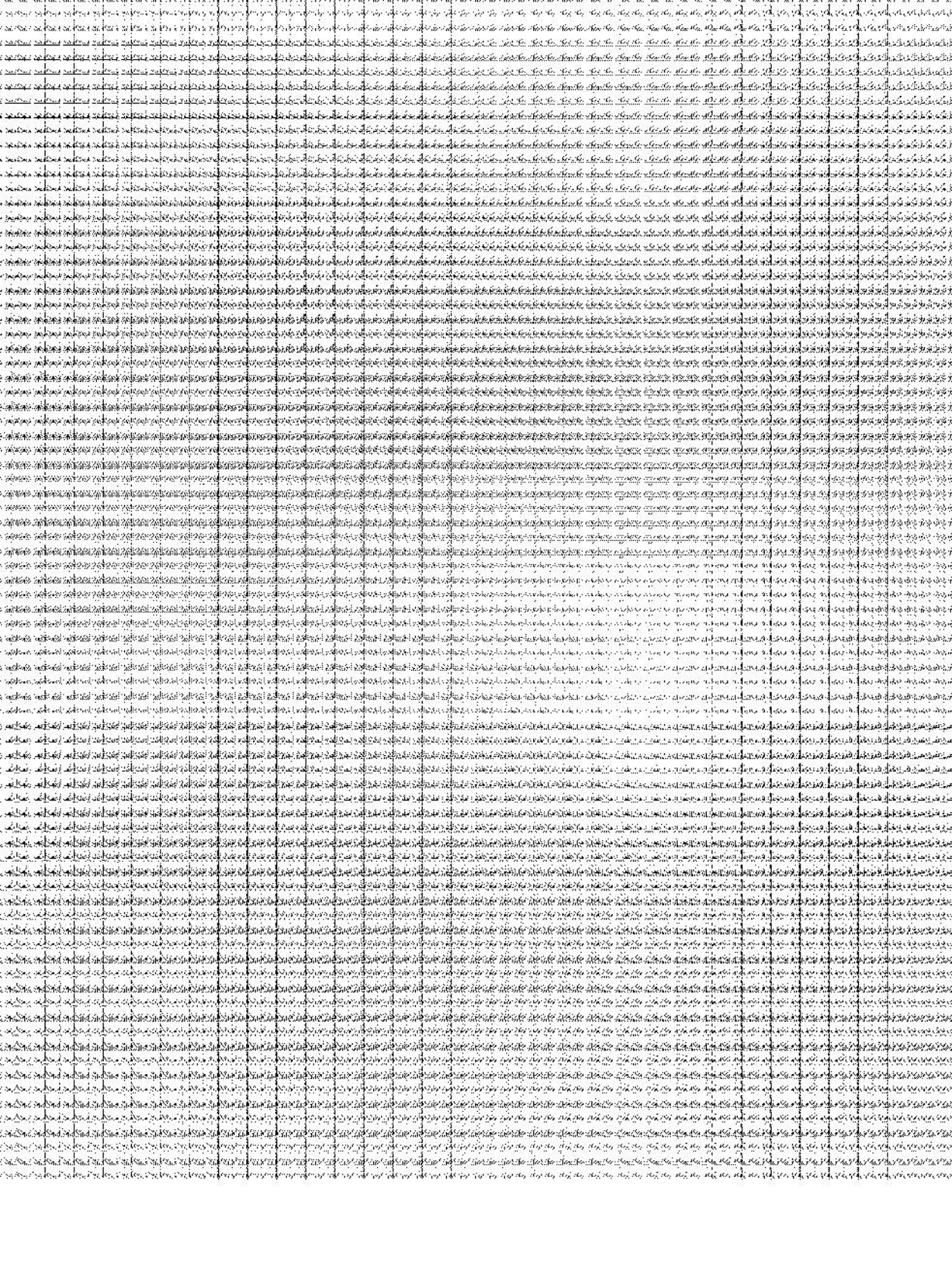


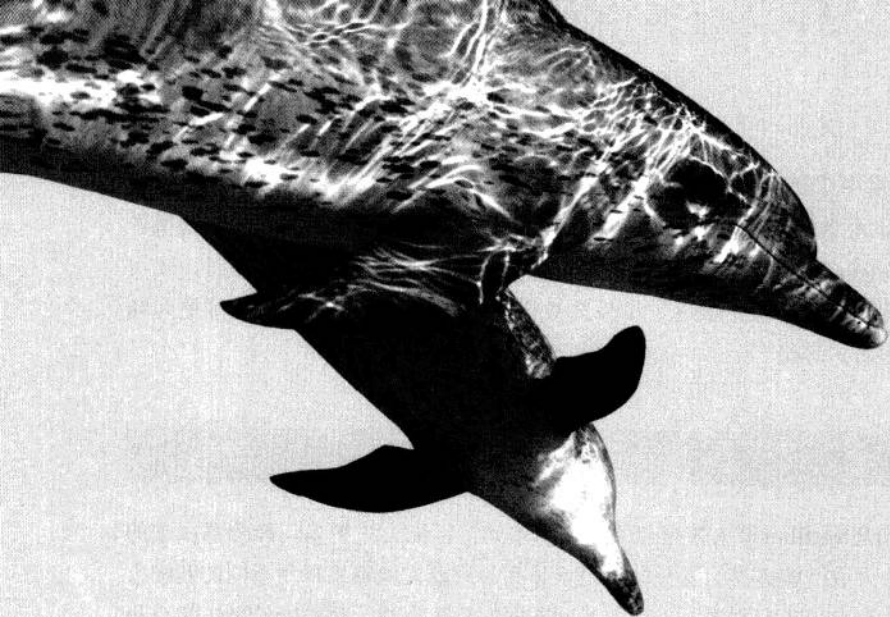
图2-50 简化的Netfilter结构

以上两个问题的答案分别为FORWARD链及192.168.0.1，你答对了吗？如果没有的话，请再次好好品味一下图2-48。

2.15 小结

本章介绍了Netfilter的结构、iptables工具的使用方法以及Netfilter的规则语法，虽然这些都是Linux防火墙最基础的部分，但也是Linux防火墙在许多运用中进行扩展的基础，因此无论如何这个章节中的每一个细节都请你明确地理解，否则将严重影响到后续章节的学习，也会直接导致所设计的防火墙系统不够可靠。





Linux

| 第3章 | Netfilter的匹配方式及处理方法

3

相信你已经从第2章中了解到Netfilter的强大功能, 不过, 到目前为止所谈的大多只是Netfilter最基本的功能。本章将介绍Netfilter的模块, 我们可以利用这些丰富的模块来提供各种不同的“匹配方式”(matches)及“处理方法”(target)。这样在编写防火墙规则时, 就可以运用这些多元化匹配方式及处理方法进一步筛选数据包, 好让防火墙的过滤规则更为精准, 因此提高防火墙的防护等级。

3.1 匹配方式

匹配方式(matches)是Netfilter筛选数据包的最基本单元, 匹配方式越多, 能分离出来的数据包种类就越细, 防火墙所能提供的安全等级也就越高。只要你能熟悉每种不同的匹配方式, 就能灵活地结合各种不同匹配方式, 设计出各种不同的防护方法, 建成固若金汤的企业安全堡垒。

3.1.1 内置的匹配方式

第2章中曾经提到过Netfilter的四大功能, 分别为filter、NAT、mangle及raw, 这些功能是由iptables_filter.ko、iptables_nat.ko、iptables_mangle.ko及iptables_raw.ko四个模块所提供, 而这些模块本身就内置了一些简单的匹配方式, 如接口(Interface)、IP地址(IP Address)及协议(Protocol)三种匹配方式。接下来将介绍这些内置的匹配方式。

1. 接口的匹配方式

在Netfilter内置的匹配方式中, 我们可以将数据包“进出”的“接口”作为匹配条件, 例如:

```
iptables -t filter -A FORWARD -i eth0 -o eth1 -j DROP
```

该示例运用了-i及-o两个参数来表明数据包的进出方向, 另外, “接口名称”的部分就要稍微留意了, 因为接口名称不一定全部是eth开头, 而是会与实体层的网络接口有关。表3-1笔者整理了常见的接口及名称。在编写防火墙规则时, 请务必留意接口的名称, 但如果还是无法分辨防火墙主机的接口名称, 不妨执行ifconfig命令, 就会看到你的主机接口名称。表3-1列出了常见的接口及名称。

表3-1 常见的接口及名称对应表

实体接口名称	逻辑接口名称
本机接口	lo(小写的LO)
Ethernet	eth0、eth1、...、ethN
Token Ring	tr0、tr1、...、trN
FDDI	fddi0、fddi1、...、fddiN
PPP	ppp0、ppp1、...pppN

2. Source/Destination Address的匹配

IP包头中有Source IP及Destination IP两项信息，其中Source IP代表数据包发送端主机的IP地址，Destination IP则代表数据包要发送往哪台主机，因此，我们可以运用这两个字段的信息，来限制是否要接受特定主机送来的数据包，或限制数据包是否可以送往特定主机，可以使用以下两个参数来执行限制操作，如下所示：

- -d：匹配目的端的IP，如-d 192.168.0.1。
- -s：匹配来源端的IP，如-s 192.168.0.1。

另外在“地址”的表示方法上，可以是单一IP，如192.168.0.1，或者任何服务标准CIDR网段的表示法皆可，如192.168.0.0/24。下面列举两个例子来说明。

示例3.1 不允许企业内的使用者访问http://www.playboy.com网站

```
iptables -A FORWARD -p tcp -i eth1 -o eth0 -d www.playboy.com -j DROP
```

记得在第2章中曾提到过，iptables会先将FQDN转成IP之后再保存到规则数据库中，因此，以上规则将可限制所有要连接到www.playboy.com网站的连接。

示例3.2 不允许因特网上的192.168.10.1主机访问公司的Web 服务器，假设192.168.10.1为公网IP

```
iptables -A FORWARD -i eth0 -o eth1 -p tcp -s 192.168.10.1 \
-d WEB_SRV --dport80 -j DROP
```

3. 协议的匹配方式

第2章曾经介绍过-p tcp、-p udp及-p icmp的匹配方式，不过Netfilter对于“协议”的匹配多达256种。可参考表3-2，其中列出了/etc/protocol文件中的前面一小部分，表中的第一个字段是给“管理员”看的，第二个字段则是给计算机看的。icmp的代码是“1”因此“-p icmp”与“-p 1”的意义相同。所以，以下两个示例的实际意义是相同的。

```
iptables -t filter -A INPUT -p icmp -j DROP
```

```
iptables -t filter -A INPUT -p 1 -j DROP
```

虽然协议的匹配可以多达256种, 不过目前实际有确定意义的协议并不多, 若你感兴趣, 可自行参考/etc/protocol这个文件。

表3-2 协议名称与编号对应表

ip	0	IP	# internet protocol, pseudo protocol number
icmp	1	ICMP	# internet control message protocol
igmp	2	IGMP	# internet group management protocol
ggp	3	GGP	# gateway-gateway protocol
ipencap	4	IP-ENCAP	# IP encapsulated in IP (officially "IP")
st	5	ST	# ST datagram mode
tcp	6	TCP	# transmission control protocol
cbt	7	CBT	# CBT, Tony Ballardie <A.Ballardie@cs.ucl.ac.uk>
egp	8	EGP	# exterior gateway protocol
igp	9	IGP	# any private interior gateway (Cisco: for IGRP)
bbn-rcc	10	BBN-RCC-MON	# BBN RCC Monitoring
nvp	11	NVP-II	# Network Voice Protocol
pup	12	PUP	# PARC universal packet protocol
argus	13	ARGUS	# ARGUS
emcon	14	EMCON	# EMCON
xnet	15	XNET	# Cross Net Debugger
chaos	16	CHAOS	# Chaos
udp	17	UDP	# user datagram protocol

● ICMP协议的高级匹配:

一般在防火墙的设置上是不接受来自因特网的icmp数据包, 过多的icmp数据包只会徒增服务器的负担, 因此将来自因特网上的icmp数据包丢弃掉, 其设置可能如下:

```
iptables -A INPUT -p icmp -j DROP
```

这样的设置虽然可以帮助我们拦截来自外界的ICMP数据包, 但也会导致服务器本身无法使用ping命令去检测服务器与因特网是否保持连接, 我们通过图3-1来解释造成这个问题的原因。ping命令的原理是对被检测的一方发出“icmp请求”包①, 并等待被检测一方应答“icmp应答”包②, 若没有接收到被检测者的应答, 则界面显示Request time out(以Windows系统为例); 若收到被检测者的应答, 则界面显示Reply from 192.168.1.254: bytes=32 time<1ms TTL=64(以Windows系统为例)。下面从图3-1中服务器的角度探讨以下两个问题。

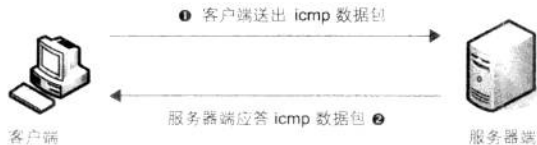


图3-1 Ping命令的操作分析

- 客户端使用ping命令检测服务器端：

客户端对服务器端送出icmp请求包，但这个icmp数据包会因符合“iptables -t filter -A INPUT -p icmp -j DROP”过滤条件而遭到丢弃。因此，客户端ping不到服务器端，当然这是我们所期望的。

- 服务器端使用ping命令检测客户端：

服务器端送出ping请求包给客户端，由于服务器端对icmp数据包的过滤条件为INPUT类型的数据包，但这个icmp请求包为OUTPUT类型，因此这个icmp数据包将可以正确送达给客户端主机。当客户端主机收到这个icmp请求包之后，接着应答服务器端主机一个icmp应答包。但很不幸的，这个应答包刚好符合服务器端对icmp包的过滤条件，因此这个icmp应答包将遭到丢弃的命运！服务器端就无法得到客户端的应答，因而造成服务器端ping不到客户端的问题。

不过，要解决以上问题其实相当容易。只要稍微了解icmp包的结构，就很容易解决这个问题，icmp并不像tcp或udp协议可以使用端口来区分数据包的应用差别。例如，端口3342送到端口80，我们可以很轻易地判断这是由客户端送给服务器端的数据包。但在icmp包的应用中，不管是客户端送给服务器端或是服务器端应答给客户端，都是使用icmp包来沟通，因此，我们无法单就协议来区分某个icmp包是客户端主动送给服务器端，又或是服务器端主动送给客户端。

接着来观察icmp包的内容，或许就可以轻易地解决这个问题了，图3-2是一个“icmp请求”包的内容。图3-3是“icmp应答”包的内容，从图中我们可以看到“请求”与“应答”数据包之间的差异。事实上，icmp有很多不同的应用，为了区别某个icmp包的应用，在icmp包包头内会有type及code两个字段，其内容都是数字，而每个不同数字的组合都代表某一种特定的应用。若你对所有icmp包的应用有兴趣，不妨参阅RFC 793和RFC 2463，如此即可查询到所有type及code的组合以及其应用。



图3-2 icmp请求包

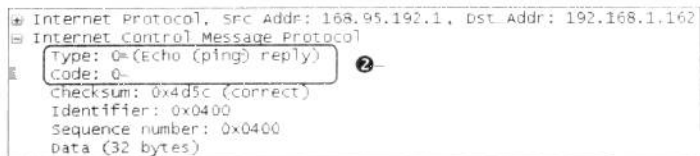


图3-3 icmp应答包

以图3-1为例,我们将icmp包分为“请求❶”及“应答❷”两种不同的数据包,而这两种icmp数据包的差异如下:

- icmp请求包: Type=8、Code=0,如图3-2所示。
- icmp应答包: Type=0、Code=0,如图3-3所示。

因此,如果希望别人ping不到我们,而我们能ping到别人,只需在防火墙上进行设置,将所有来自因特网且type为8类型的icmp包过滤掉即可。其规则语法如下:

```
iptables -A INPUT -p icmp --icmp-type 8 -j DROP
```

3.1.2 从模块扩展而来的匹配方式

看完内置匹配方式后,接下来介绍从模块扩展而来的匹配方式。由于这些模块数量众多,而且会随着Linux版本的更新而增加,因此这里不准备以表格形式将其列出。若你有兴趣,不妨自己到这里列出的路径去看看你所使用的Linux包含哪些模块可供使用。另外,先说明一下,我们并不需要特别去执行加载模块的操作,只要在netfilter规则中使用到模块功能,iptables这个工具便会自动将相应的模块载入,因此不需要去考虑这个问题。

- /lib/modules/kernel_version/kernel/net/netfilter
- /lib/modules/kernel_version/kernel/net/ipv4/netfilter
- /lib/modules/kernel_version/kernel/net/ipv6/netfilter

1. TCP/UDP协议的匹配方式

第3.1.1一节的第3部分介绍过ICMP协议的匹配方式,本节将介绍TCP及UDP协议的高级匹配方式。

- TCP协议的高级匹配:

图3-4为TCP包的包头内容,在包头之中有两项很重要的信息,如标注❶所示的“Source Port”及“Destination Port”,以及标注❷所示的TCP-Flags,而以上信息都可作为Netfilter的匹配条件。

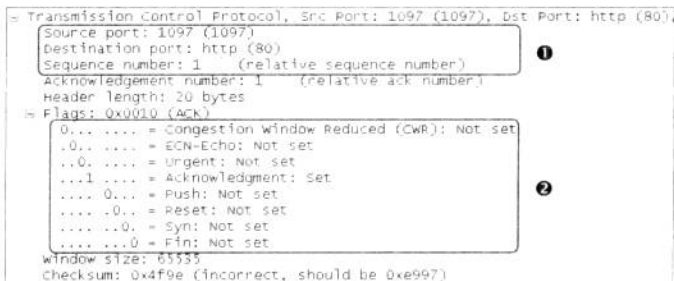


图3-4 TCP包头

● Source Port及Destination Port的匹配:

如图3-4所示,在TCP的包头之中有Source Port及Destination Port两个字段,其目的在于记录数据包发送端应用程序所使用的端口,以及这个数据包是要送往目的端主机的哪个端口,因此,我们可以利用这两个字段的数据来判断客户端主机是要访问服务器端主机的哪项服务,其语法分别是--dport及--sport,例如:

- --dport 22: 匹配Destination Port为22的数据包。
- --sport 80: 匹配Source Port为80的数据包。

以图3-5为例,如果我们要限制192.168.0.0/24整个网段的使用者不得使用因特网上的FTP服务,那么只需在防火墙上执行以下设置即可:

```
iptables -A FORWARD -i eth1 -o eth0 -p tcp -s 192.168.0.0/24 \
--dport 21 -j REJECT
```

因此,当192.168.0.0/24网段的使用者要访问因特网的FTP服务,其服务请求包内的Destination Port必定是21。我们可以使用语法“--dport 21”来匹配此类数据包,当符合这个条件的数据包经过防火墙主机时,就将其丢弃掉,如此即可达到不让192.168.0.0/24网段上的主机去访问因特网上的FTP服务的目的。



提示

--dport/--sport除了可以匹配单个端口外,也可以用来匹配一个范围的端口,例如--dport 20:50即为匹配端口20到端口50之间的范围。

● TCP-Flags的匹配:

如图3-4所示,在TCP的包头中有1个字节的空间用来存放tcp-flags的状态,而tcp-flags是由8位所构成(旧版本的规格为6位的),要了解其详细信息,请参阅RFC3168。在图3-4中,可以完整看到tcp-flags每一位的状态,这里仅简单介绍较重要的4个位,如表3-3所示,而TCP-Flags的目的是作为TCP连接控制之用。

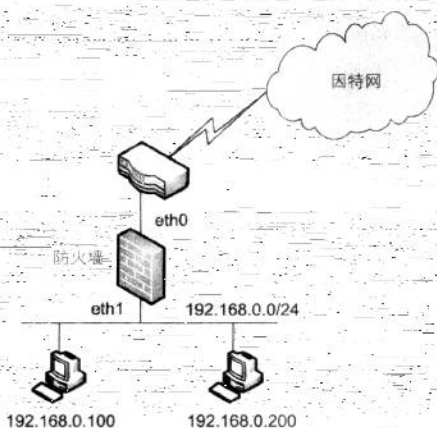


图3-5 网关式防火墙示例图

表3-3 TCP-Flags的位含义

字段	标志	说明
位1	Finish	连接终止信号
位2	Synchronize	连接请求信号
位3	Reset	立即终止连接
位5	Acknowledge	确认应答信号

当客户端与服务器端将TCP协议作为数据交换协议时, 在开始传输数据之前, 必须先客户端与服务器端之间建立好“TCP连接”。有了这条连接之后, 就可以通过它来传输数据。如图3-6即为TCP连接的建立过程, 客户端先送出一个带有syn标记的数据包给服务器端, 这个数据包就是连接请求包, 服务器端接着应答一个带有syn及ack标记的数据包给客户端, 表示服务器端已经接受客户端的连接请求操作, 接着客户端再回应一个带有ack标记的数据包给服务器端。在完成以上流程后, 客户端及服务器端的TCP连接即正式建立。

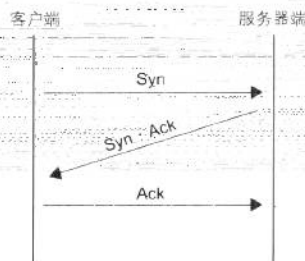


图3-6 TCP连接建立过程

我们可以从图3-7的实际案例中看到, 主机10.0.1.100使用端口1304向主机10.0.1.101的端口123送出一个带有syn标记的数据包, 接着主机10.0.1.101使用端口123向主机10.0.1.100的1304端口应答一个带有syn及ack标记的数据包, 最后主机10.0.1.100使用端口1304应答主机10.0.1.101的端口123一个带有ack标记的数据包。到此, 连接正式建立, 客户端及服务器端可以通过这条连接来传输数据了。以上连接建立过程我们称为“3 way handshake”, 并将之翻译为“三次握手”。

No.	Time	Source	Destination	Protocol	Info.
1	0.000000	10.0.1.100	10.0.1.101	TCP	1304 > 123 [SYN] Seq=0 Len=0 MSS=1460
2	0.000163	10.0.1.101	10.0.1.100	TCP	123 > 1304 [SYN, ACK] Seq=0 Ack=1 win=
3	0.000216	10.0.1.100	10.0.1.101	TCP	1304 > 123 [ACK] Seq=1 Ack=1 win=6553
4	2.765612	10.0.1.100	10.0.1.101	TCP	1304 > 123 [FIN, ACK] Seq=1 Ack=1 win=
5	2.767513	10.0.1.101	10.0.1.100	TCP	123 > 1304 [ACK] Seq=1 Ack=2 win=5840
6	2.768504	10.0.1.101	10.0.1.100	TCP	123 > 1304 [FIN, ACK] Seq=1 Ack=2 win=
7	2.768544	10.0.1.100	10.0.1.101	TCP	1304 > 123 [ACK] Seq=2 Ack=2 win=6553

图3-7 TCP连接建立过程

TCP连接的建立与终止都有固定的流程, 我们通过图3-8来讨论TCP连接终止的流程, 当客户端完成数据的传输之后, 客户端就会对服务器端送出一个“连接终止”信号, 而这个信号就是一个带有fin标记的数据包。当服务器端收到这个数据包之后, 会先应答客户端一个带有ack标记的数据包, 紧接着又送出一个带有fin标记的数据包给客户端, 在客户端收到这个

数据包之后，会回应服务器端一个带有ack标记的数据包。当以上四个数据包传送完毕之后，客户端及服务器端的TCP连接就正式结束了。

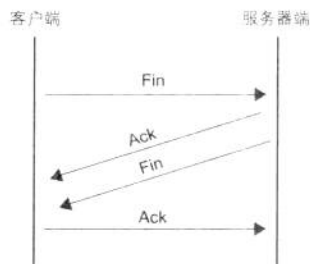


图 3-8 TCP连接终止过程

我们可以从图3-9的实例中看到TCP连接终止的过程，首先主机10.0.1.100对主机10.0.1.101送出一个带有fin标记的数据包，接着主机10.0.1.101应答一个带有ack标记的数据包给主机10.0.1.100，此后主机10.0.1.101送出一个带有fin标记的数据包给主机10.0.1.100，最后主机10.0.1.100回送一个带有ack标记的数据包给主机10.0.1.101，如此，这条TCP连接就正式结束。

No.	Time	Source	Destination	Protocol	Info.
1	0.000000	10.0.1.100	10.0.1.101	TCP	1304 > 123 [SYN] Seq=0 Len=0 MSS=1460
2	0.000163	10.0.1.101	10.0.1.100	TCP	123 > 1304 [SYN, ACK] Seq=0 Ack=1 win=
3	0.000216	10.0.1.100	10.0.1.101	TCP	1304 > 123 [ACK] Seq=1 Ack=1 win=6553
4	2.765612	10.0.1.100	10.0.1.101	TCP	1304 > 123 [FIN, ACK] Seq=1 Ack=1 win=
5	2.767513	10.0.1.101	10.0.1.100	TCP	123 > 1304 [ACK] Seq=1 Ack=2 win=5840
6	2.768504	10.0.1.101	10.0.1.100	TCP	123 > 1304 [FIN, ACK] Seq=1 Ack=2 win=
7	2.768544	10.0.1.100	10.0.1.101	TCP	1304 > 123 [ACK] Seq=2 Ack=2 win=6553

图 3-9 TCP连接终止过程

看完TCP连接的建立及终止过程，我们基本上已经了解TCP-Flags为TCP连接控制之用。在此请你仔细思考一个问题，在一个数据包上是否可能同时出现syn及fin标记呢？答案当然是“不可能”，因为收到这个数据包的主机根本无从判断，这个数据包的目的是服务请求还是终止连接。但如果系统真的收到这种数据包，系统又该如何处理呢？那就得看系统程序设计师的考虑是否周全，如果曾考虑到这样的问题，系统当然就安然无事；但如果没有考虑过这个问题，系统会如何？谁也不知道，也许就崩溃了。

为防止这类不正常的数据包，Netfilter当然有其解决之道，在Netfilter语法中提供了我们筛选TCP-Flags的方法，再此以一台邮件服务器上的单机防火墙为例进行说明：

```
#!/bin/bash

iptables -P INPUT DROP

iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 25 -j ACCEPT
iptables -A INPUT -p tcp --dport 110 -j ACCEPT
iptables -A INPUT -p all -m state --state ESTABLISHED,\
RELATED -j ACCEPT
```

不过，这个单机防火墙设计得并不理想，怎么说呢？如果有一客户端直接对本机的端口25送出一个带有fin标记的数据包，请问这个数据包是否会进入系统？很不幸的，答案是“会”，但以上这个操作不符合TCP协议的原则，因为TCP的连接一定是先有“建立”才会

有“终止”。因此,如果一台主机收到客户端送过来的第一个数据包,其内就带有fin标记,而系统会如何,谁也不知道(不过,事实上应该是不会有这么不严谨的系统)。为了防止发生此类行为,我们将防火墙规则改为:

```
iptables -A INPUT -p tcp --syn --dport 22 -m state --state NEW -j ACCEPT
```

我们知道,在建立连接的过程中,TCP的第一个数据包内只会包含syn标记,因此可以使用state模块来判断数据包是否为该连接的第一个数据包,并检查该数据包是否包含syn标记。若两项条件都符合才允许该数据包进入,相对于前者,这样的设置显然在规则的考虑上相对安全多了。

除了使用--syn来判断数据包内“只”包含syn标记外,我们还可以使用以下方式来判断所有标记的状态:

示例3.3 检查所有TCP-Flags

```
iptables -A INPUT -p tcp --tcp-flags ALL SYN,FIN -j DROP
```

这个示例的目的在于检查“所有”TCP-Flags,但只有syn及fin两个标记同时为1时,数据包才会被筛选出来,因此,我们可以使用这个语法来筛选掉同时包含syn及fin标记的数据包。

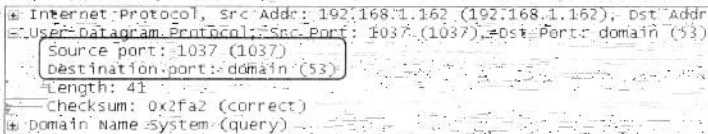
示例3.4 只检查syn和fin两个标记

```
iptables -A INPUT -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP
```

这个示例“只检查”TCP-Flags中syn及fin两个标记,而且这两个标记必须同时为1时,数据包才会被筛选出来。

● UDP协议的高级匹配:

如图3-10所示,相对于TCP包头而言,UDP包头就显得简单多了。对UDP协议的匹配只有下列两种情况:



```
Internet Protocol, Src Addr: 192.168.1.162 (192.168.1.162), Dst Addr: 192.168.1.162 (192.168.1.162)
User Datagram Protocol, Src Port: 1037 (1037), Dst Port: domain (53)
Source port: 1037 (1037)
Destination port: domain (53)
Length: 41
Checksum: 0x2fa2 (correct)
Domain Name System (query)
```

图3-10 UDP包头

- --sport: 匹配数据包来源端的端口。
- --dport: 匹配数据包目的端的端口。

再以图3-5为例,假设我们要让192.168.0.0/24网段的主机通过因特网上的DNS来进行名

称解析操作。可在防火墙上设置以下规则，让所有送到因特网且Destination Port为53的数据包正常穿过防火墙。

```
iptables -A FORWARD -i eth1 -o eth0 -p udp --dport 53 -j ACCEPT
```



提示

--dport/--sport除了可以匹配单个端口外，也可以用来匹配一个范围的端口，例如--dport 20:50即为匹配端口20到端口50之间的范围。

2. MAC地址的匹配

以太网包头中记录着数据包来源端的MAC地址，以及数据包目的端的MAC地址。可以从图3-11看到这两项信息，这两项信息在以太网的环境中是非常重要的，因为它记录着某个数据包是由那一块网卡送出，并且要送给哪一块网卡，而这个匹配功能是由xt_mac.ko模块所提供的功能。

```

# Frame 6 (106 bytes on wire, 106 bytes captured)
# Ethernet II, Src: 00:02:b3:0c:23:1b, Dst: 00:0f:b0:93:b2:7d
#   Destination: 00:0f:b0:93:b2:7d (192.168.2.10)
#   Source: 00:02:b3:0c:23:1b (192.168.2.11)
#   Type: IP (0x0800)
# Internet Protocol, Src Addr: 192.168.2.11 (192.168.2.11), Dst:
# Transmission Control Protocol, Src Port: 22 (22), Dst Port: 15
# SSH Protocol

```

图3-11 以太网包头

有了这个模块，我们就可以实现一些较高级的匹配功能，在此列举两个例子。

示例3.5 情形1

假设公司内部有一台MySQL数据库服务器，我们想只有特定的使用者才可以连接该服务器，但公司内的IP地址是动态分配的，所以在Netfilter规则中，我们无法使用来源IP作为过滤条件。此时，xt_mac.ko模块就派上用场了。我们可以将这些特定计算机上网卡的MAC地址作为过滤条件，并在MySQL服务器主机上设置以下规则，MySQL使用的端口是TCP端口3306。

```

iptables -A INPUT -p tcp --dport 3306 -m mac --mac-source \
00:02:B3:0C:23:1B -j ACCEPT
iptables -A INPUT -p tcp --dport 3306 -m mac --mac-source \
00:50:56:C0:00:01 -j ACCEPT

```

其中-m mac表示要用到xt_mac.ko模块，--mac -source 00:50:56:C0:00:01是要给xt_mac.ko模块的参数，而这个参数就是要匹配的MAC地址。因此当00:50:56:C0:00:01这块网卡要访问

TCP端口3306时，我们是允许的，如此即可达到“锁定”MAC地址的目的。

示例3.6 情形2

如果我们不希望有任何人带着笔记本电脑到公司，插上网线就可以连到公司的网络，就可以在“网关”或是“DHCP 服务器”中着手。不过，前提是网关及DHCP 服务器最好都是Linux系统，但也不一定就是如此。因为有一些高级防火墙功能在稍后的章节才会介绍到，具备了这些功能后，不管网关及DHCP 服务器是不是Linux系统都无关紧要。

接下来必须首先将公司内所有计算机的MAC地址收集起来，保存为如下的文件格式。假设路径及文件名为/root/mac_list.txt。

```
00:0F:B0:93:B2:7D      # 老总计算机
00:11:22:33:44:55      # Jacky_Chen 的计算机

cat /root/mac_list.txt | while read MAC
do
    MAC=$( echo $MAC | awk '{print $1}' )

    iptables -t filter -A FORWARD -i eth1 \
        -o eth0 -m mac --mac-source $MAC -j ACCEPT
done
```

接着，我们可以在网关防火墙的Shell脚本文件中添加以上这段Shell脚本。这样当防火墙的Shell脚本执行后，以上这段Shell脚本就会自动将/root/mac_list.txt内的MAC地址添加到网关防火墙规则中，并让这些网卡所发送出来的数据包可以正常通过防火墙，从而达到我们的目的。当然，如果将以上的Shell脚本改进一下，就可以应用在DHCP 服务器之上，如果不属于公司的网卡，就不要分配IP给这个客户端。因此，有了这些模块之后，如果再结合你的想象力，我们就可以让Linux防火墙提供更多不同的应用了。

3. Multiport的匹配

很多人在设置防火墙规则时，会因为服务器本身所要提供的服务较多，而编写如下的防火墙规则。但这样的规则设置效果不佳，因为规则太多会造成阅读困难，又容易出错，另外也会影响防火墙的性能(稍后将单独讨论有关性能的问题，在此就先不多做说明)。可以采用什么方法可以解决这些问题呢？multiport匹配方式正是解决这个问题的独门利器。

```
#!/bin/bash
iptables -P INPUT DROP

iptables -t filter -F

iptables -A INPUT -p all -m state --state INVALID -j DROP
iptables -A INPUT -p tcp --syn -m state --state NEW --dport 21 -j ACCEPT
```



```
iptables -A INPUT -p tcp --syn -m state --state NEW --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --syn -m state --state NEW --dport 23 -j ACCEPT
iptables -A INPUT -p tcp --syn -m state --state NEW --dport 25 -j ACCEPT
iptables -A INPUT -p tcp --syn -m state --state NEW --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --syn -m state --state NEW --dport 110 -j ACCEPT
iptables -A INPUT -p tcp --syn -m state --state NEW --dport 443 -j ACCEPT
iptables -A INPUT -p all -m state --state ESTABLISHED, RELATED -j ACCEPT
```

multiport功能由xt_multiport.ko模块提供。由于multiport匹配的目标是端口(Port)，因此，multiport必须与-p tcp或-p udp结合使用。下面是multiport提供的匹配参数：

- --dports：匹配目标为Destination Port，如--dports 21,22,25。
- --sports：匹配目标为Source Port，如--sports 80,443。
- --ports：匹配目标为Destination或Source Port，如--ports 22,80。

接着使用multiport语法来改写以上示例。改写后的内容如下，这样看起来是不是就舒服多了？如果可以巧妙地使用multiport功能，将有助于我们精简防火墙规则数量，而且在同一条规则中，最多可以同时匹配多达15个端口。

```
#!/bin/bash
iptables -F INPUT DROP

iptables -t filter -F

iptables -A INPUT -p all -m state --state INVALID -j DROP
iptables -A INPUT -p tcp --syn -m state --state NEW \
    -m multiport --dports 21,22,23,25,80,110,443 -j ACCEPT
iptables -A INPUT -p all -m state --state ESTABLISHED,\
    RELATED -j ACCEPT
```

4. 匹配数据包的MARK值

相信你在看到这个匹配方式之后，一定会很惊讶，什么是数据包的MARK值？当我初次看到这个功能时，也怀有相同的疑问。其实MARK与Netfilter的一个“处理方法”(target)的模块有关，这个处理方法模块称为xt_MARK.ko，下面首先简单说明MARK处理方法的相关用途。

以如下示例及图3-12为例，我们可以在mangle表中运用MARK处理方法，将所有经过PREROUTING链且Destination Port为80的数据包标记一个值，而这个值由管理员自行确定。本示例将符合条件的数据包标记为80。

```
iptables -t mangle -A PREROUTING -p tcp --dport 80 -j MARK --set-mark 80
```

接着，可以在filter表的FORWARD链中设置如下的规则。这个规则是说“如果进入FORWARD链的数据包，其MARK值为80的话，就将其丢弃”。或许你会觉得奇怪，为何不

直接把符合条件的数据包丢弃掉，而是如此大费周折？其实这只是Netfilter提供给我们的另一种选择方式罢了，至于是否要使用这个功能，就看你设置Netfilter规则的习惯了。

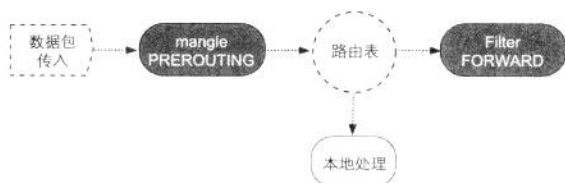


图3-12 简化的Netfilter结构

```
iptables -A FORWARD -p all -m mark --mark 80 -j DROP
```

5. Owner的匹配

Owner的匹配方式由ipt_owner.ko模块所提供，只适用于OUTPUT链与POSTROUTING链之中。因此，在使用时请特别注意。Owner可以提供以下两种不同的匹配方式，分别如下：

● --uid-owner userid | username:

我们可以使用--uid-owner来匹配数据包是由哪位“使用者”所产生的，由此用来控制特定的使用者只能访问哪些特定的服务。例如，使用者jacky在本机上只能使用浏览网页功能。因此，我们在本机上针对jacky这位使用者所产生的所有数据包，只需要放行DNS查询数据包以及HTTP请求数据包即可，规则如下：

```
iptables -A OUTPUT -p tcp -m owner --uid-owner jacky \
    --dport 80 -j ACCEPT
iptables -A OUTPUT -p udp -m owner --uid-owner jacky \
    --dport 53 -j ACCEPT
iptables -A OUTPUT -p all -m owner --uid-owner jacky -j DROP
```

● --gid-owner groupid | groupname:

我们可以使用--gid-owner来匹配数据包是由哪一个“使用者群组”所产生的，由此用来控制某一个部门的所有员工可以访问哪些特定的服务。例如，在本机上所有sales这个部门的成员只能使用浏览网页功能。因此，我们在本机上针对sales这个部门的成员，只需要放行DNS查询数据包以及HTTP请求数据包即可，规则如下：

```
iptables -A OUTPUT -p tcp -m owner --gid-owner sales \
    --dport 80 -j ACCEPT
iptables -A OUTPUT -p udp -m owner --gid-owner sales \
    --dport 53 -j ACCEPT
iptables -A OUTPUT -p all -m owner --gid-owner sales -j DROP
```

看完以上介绍之后，或许你会迫不及待地想试试这个模块，不过，这只能运用在“本

机”之上，不可能在网关上使用Owner模块来限制谁可以连上因特网，或谁不可以连上因特网。原因是我们所要匹配的这些特征并不会随着数据包送到网上，所以数据包只要离开本机就无法运用Owner模块来进行匹配。

最后再提醒一点，某些命令在执行时会改变执行者身份，例如，ping命令是一个suid的执行文件(-rwsr-xr-x 1 root root 35108 Jun 16 2004 /bin/ping)。因此，无论任何人执行ping命令，系统实际上都会以root身份来执行。在测试--uid-owner时，不要使用ping命令来测试，否则--uid-owner的匹配将会失效。

6. IP范围的匹配

假设要将某个网段内的“一段IP”封锁起来，如192.168.0.2到192.168.0.61，请问该如何设置这个规则？因为这个范围并不是一个标准的CIDR网段，因此并不能使用如“-s 192.168.0.0/28”的网段表示法来匹配这个范围，难道真要一行一行来写吗？其实也不需要如此大费周折，早期的做法是将192.168.0.0/24分割成多个小网段，再去匹配哪个小网段与要被封锁的网段较为接近，然后再补足遗漏的部分。例如，我们要封锁的网段是192.168.0.2到192.168.0.61，而这个网段的实际IP数量是59个。接着匹配表3-4的范围，其中最接近我们需求的网段数量是4，因为网段分为4段时，每一个网段所包含的IP数量为64个，与我们需要的59个最为接近。

表3-4 网段数量及IP数量表

切割的网段数量	每个网段拥有的IP数量
2	128
4	64
8	32
16	16
32	8

接着再从表3-5来匹配，哪一个网段范围最符合我们的需要？其中第一段的范围是192.168.0.0到192.168.0.63，与我们要封锁的192.168.0.2到192.168.0.61最为接近。因此可将规则写成：

```
iptables -A INPUT -p all -s 192.168.0.0/26 -j DROP
```

表3-5 网段所包含的范围

网段	范围	掩码
1	192.168.0.0~192.168.0.63	26
2	192.168.0.64~192.168.0.127	26
3	192.168.0.128~192.168.0.191	26
4	192.168.0.192~192.168.0.255	26

不过，这样的写法将会使192.168.0.1、192.168.0.2、192.168.0.62及192.168.0.63遭到错杀，并不完美。正确写法如下：

```
iptables -A INPUT -p all -s 192.168.0.1 -j ACCEPT
iptables -A INPUT -p all -s 192.168.0.2 -j ACCEPT
iptables -A INPUT -p all -s 192.168.0.61 -j ACCEPT
iptables -A INPUT -p all -s 192.168.0.62 -j ACCEPT
iptables -A INPUT -p all -s 192.168.0.0/26 -j DROP
```

通过网段的划分，我们可以将原本需要50行的规则缩短为5行，但计算过程还真的很麻烦；不过有了ipt_iprange.ko模块，往后遇到这种问题就必伤脑筋了。可将以上的示例改为：

```
iptables -A INPUT -m iprange --src-range 192.168.0.2-192.168.0.61 -j DROP
```

哇！真是太完美啦！一行搞定，有了iprange后，从此再也不需计算了。iprange模块提供了以下两个匹配参数：

- --src-range:

匹配“来源地址”范围，例如“iptables-A INPUT-m iprange--src-range 192.168.0.2-192.168.0.62 -j DROP”。

- --dst-range:

匹配“目的地址”范围，例如“iptables -A OUTPUT-m iprange--dstC-range 192.168.0.2-192.168.0.62 -j DROP”。

7. TTL值的匹配

如图3-13所示，在IP包头中有一个字段称为TTL，这个字段记录数据包的存活时间，早期是以时间单位来计算，不过现在的计算方式是以可跨越路由器的数量为单位。例如，当计算机主机送出数据包时，其数据包内IP包头的TTL字段默认就会被填上一个值，而这个默认值因操作系统而异。例如，Windows系统的默认值为128，Linux系统默认值为64。每当数据包跨越一个路由器，TTL值就会减1，最后当数据包内的TTL值为0时，数据包就会遭到丢弃。

```

Internet Protocol, Src Addr: 202.43.195.52 (202.43.195.52)
  Version: 4
  Header length: 20 bytes
  * Differentiated Services Field: 0x00 (DSCP 0x00: Default;
  Total Length: 40
  Identification: 0xef9a (61338)
  * Flags: 0x00
  Fragment offset: 0
  Time to live: 127
  Protocol: TCP (0x06)
  Header checksum: 0xfc22 (correct)
  Source: 202.43.195.52 (202.43.195.52)
  Destination: 192.168.2.10 (192.168.2.10)

```

图3-13 IP包头的数据内容

TTL值的匹配是由Netfilter的ipt_ttl.ko模块所提供的，其功能是为匹配数据包内的TTL值，而其匹配的参数共有三项，如下所示：

- -m ttl--ttl-eq 64：匹配TTL值“等于”64者。
- -m ttl--ttl-lt 64：匹配TTL值“小于”64者。
- -m ttl--ttl-gt 64：匹配TTL值“大于”64者。

例如，为将TTL值为64的数据包丢弃掉，可使用以下规则：

```
iptables -A INPUT -m ttl --ttl-eq 64 -j REJECT
```

8. 数据包的状态匹配

第2章曾介绍过状态(state)的匹配方式，这里将再次深入探讨state的匹配方式。首先，在TCP规范中，TCP的连接状态共分12种，分别为ESTABLISHED、SYN_SENT、SYN_RECV、FIN_WAIT1、FIN_WAIT2、TIME_WAIT、CLOSED、CLOSE_WAIT、LAST_ACK、LISTEN、CLOSING和UNKNOWN。但在Netfilter的state模块描述中，连接状态只有4种，分别是ESTABLISHED、RELATED、NEW及INVALID。因此，千万勿将TCP的状态与Netfilter的状态混为一谈。

由xt_state.ko模块提供state的匹配方式。接下来将在不同通信协议中解释ESTABLISHED、NEW、RELATED及INVALID四种连接状态：

● TCP协议：

state模块描述中的ESTABLISHED如图3-14所示。该示例以客户端对服务器端提出TCP的连接请求为例。首先客户端主机向服务器端发送一个带SYN标记的数据包，当这个数据包到达防火墙时，由于这个数据包是该条连接中的第一个数据包，因此，防火墙会表示该数据包为“NEW”的状态。接着这个数据包即送达服务器端主机，服务器端主机在收到带有SYN标记的数据包之后，随即应答客户端一个带有SYN及ACK标记的数据包。当这个应答数据包送达防火墙时，防火墙随即会检查这个数据包，如果该数据包是由内部主

机对外送出的请求包，用于应答回来的数据包，那么防火墙就会标记该数据包的状态为“ESTABLISHED”❶。建立ESTABLISHED状态后，在这一条连接中传输的任何一个数据包都会被防火墙认定为ESTABLISHED状态的数据包❷。看完以上解释之后，你或许心存以下两个疑问。

- 如标记❶所示，防火墙为何知道这个数据包是应答给内部主机的？
- 如标记❷所示，防火墙如何认定之后的这些数据包都是ESTABLISHED状态的

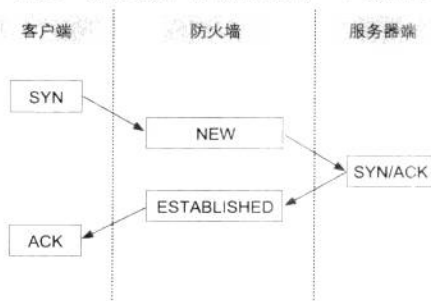


图3-14 state模块下的TCP连接状态解释

其实以上这两个问题的答案是一样的，关键在于`/proc/net/nf_conntrack`这个文件。`nf_conntrack`是Netfilter连接跟踪功能的数据库，这个数据库的内容大致如下所示：

```
tcp 6 247325 ESTABLISHED src=10.0.1.101 dst=192.168.10.1 sport=46892 dport=80 packets=1
bytes=40 [UNREPLIED] src=192.168.10.1 dst=10.0.1.101 sport=80 dport=46892 packets=0
bytes=0 mark=0 secmark=0 use=1
```

而连接跟踪功能是由`nf_conntrack.ko`模块提供的功能。也就是说，当我们在Netfilter的语法中使用`-m state`时，`iptables`工具除了会加载`xt_state.ko`模块之外，`nf_conntrack.ko`模块也会自动被载入(在CentOS 6的版本中，`nf_conntrack`及`state`模块已直接被编译到kernel里面了，因此在默认情况下你将不会看到这两个文件)。`nf_conntrack`模块随即在`/proc/net/`目录下建立`nf_conntrack`这个数据库，如表3-6所示。接下来分析“TCP三次握手”的过程中，每个数据包通过防火墙时，在连接跟踪数据库中的记录信息。

表3-6 SYN数据包送出时的连接跟踪信息

字段	内容
1	tcp
2	6
3	117
4	SYN_SENT
5	src=192.168.0.200 dst=10.0.1.102 sport=41501 dport=23
6	[UNREPLIED]
7	src=10.0.1.102 dst=192.168.0.200 sport=23 dport=41501
8	mark=0 secmark=0 use=1

当客户端发送的第一个SYN数据包达到防火墙时，`nf_conntrack`模块随即在连接跟踪数据库中建立如表3-6的信息，其中前7个字段是我们所需要了解的，随后介绍其含义。

- 字段1: 记录这个数据包是属于TCP协议的数据包。
- 字段2: TCP协议的代码为6, 这个信息可从/etc/protocols文件中找到对应关系。
- 字段4: 在TCP规范中, 当客户端发送带有SYN标记的数据包后, 这条TCP的连接状态即会设置为SYN_SENT, 而这个带有SYN标记的数据包送达防火墙时, 防火墙上的TCP连接状态也会设置为SYN_SENT, 而state模块的记录则是为NEW状态。接着等待服务器端应答带有SYN及ACK标记的数据包, 如果等到, 那么TCP的连接就会进入SYN_RECV状态; 如果没有等到服务器端的应答, 那么防火墙就必须依据第三个字段的信息来处理这条连接。
- 字段3: 如果客户端在超过120秒后都无法得到服务器端的应答, 那么该连接的状态在防火墙上就会被清除掉。而这个120秒是由/proc/sys/net/netfilter/nf_conntrack_tcp_timeout_syn_sent这个文件来决定的。
- 字段5: nf_conntrack模块会将该数据包的Source IP、Destination IP、Source Port及Destination Port记录下来, 从这个信息我们可以推断, 客户端的IP为192.168.0.200, 服务器端的IP为10.0.1.102, 而客户端使用端口41501来连接服务器端的端口23。
- 字段6: UNREPLIED代表防火墙尚未收到服务器端应答的数据包。
- 字段7: 由防火墙自动产生的, 其依据是第5个字段的反向; 也就是可以从第5个字段中推断出来。服务器端应答给客户端的数据包信息应为Source IP是10.0.1.102、Destination IP是192.168.0.200、而Source Port是23、Destination Port是41501。为何服务器端应答给客户端的数据包可以正常返回客户端? 关键就在这一条信息, 在看完以上的解释之后, 你是否了解ESTABLISHED是如何做到的呢?

当服务器端的应答数据包送达防火墙时, nf_conntrack模块随即“更新”连接跟踪数据库中的信息, 如表3-7所示。随后分析其中各个字段的含义。

表3-7 服务器端应答带有SYN及ACK标记的数据包

字段	内容
1	tcp
2	6
3	54
4	SYN_RECV
5	src=192.168.0.200 dst=10.0.1.102 sport=41501 dport=23
6	UNREPLIED
7	src=10.0.1.102 dst=192.168.0.200 sport=23 dport=41501
8	mark=0 secmark=0 use=1

- 字段1: 记录该数据包是属于TCP协议的数据包。
- 字段2: TCP协议的代码为6, 这个信息可从/etc/protocols文件中找到对应关系。
- 字段4: 在TCP规范中, 在服务器端收到带有SYN标记的数据包后, 随即会应答一个带有SYN及SCK标记的数据包, 而此时该连接即进入SYN_RECV状态, 而当这个数据包抵达防火墙时, 防火墙即会更新这条连接的状态为SYN_RECV, 同时这条TCP连接在Netfilter的描述下则为ESTABLISHED的状态; 如果没有等到服务器端的应答呢? 那么防火墙就必须根据第三个字段的信息来处理这条连接。
- 字段3: 如果服务器端应答客户端一个带有SYN及ACK标记的数据包, 却一直等不到客户端的进一步响应, 那么防火墙将在60秒后清除该连接记录, 而这个60秒是由/proc/sys/net/netfilter/nf_conntrack_tcp_timeout_syn_recv这个文件来确定的。
- 字段5: 此字段的记录并不会被改变, 还是记录着客户端及服务器端的连接参数。
- 字段6: 由于已经收到服务器端的应答, 因此, 这个字段内容已清空, 不再是UNREPLIED。
- 字段7: 同字段5。

当客户端应答“TCP三次握手”过程中的最后一个数据包时, nf_conntrack模块随即在连接跟踪数据库中“更新”连接信息, 如表3-8所示。随后分析各个字段的含义。

表3-8 客户端应答带有ACK标记的数据包

字段	内容
1	tcp
2	6
3	431956
4	ESTABLISHED
5	src=192.168.0.200 dst=10.0.1.102 sport=41501 dport=23
6	src=10.0.1.102 dst=192.168.0.200 sport=23 dport=41501
7	[ASSURED]
8	mark=0 secmark=0 use=1

- 字段1: 记录该数据包是属于TCP协议的数据包。
- 字段2: TCP协议的代码为6, 这个信息可从/etc/protocols文件中找到对应关系。
- 字段4: 在TCP规范中, 当“TCP三次握手”完成后, 连接状态即进入

ESTABLISHED，而这个状态是“TCP的连接状态”，而非state所描述的状态。另外，state模块的描述在前一个步骤中已是ESTABLISHED状态了。

- 字段3：在TCP状态的ESTABLISHED建立后，客户端及服务器端即可通过这条连接来传输数据。但如果这条连接建立起来之后，却一直没有传输任何数据包，那么防火墙将会在432 000秒之后，清除该条连接的记录。这时客户端及服务器端的连接将被迫中断，而这个432 000秒是由`/proc/sys/net/netfilter/nf_conntrack_tcp_timeout_established`这个文件来决定的。但只要该连接的生命周期未尽，且只要有数据包在传输，那么这个值就自动恢复到432 000秒，而继续延续该连接的生命周期。
- 字段5：此字段的记录并不会被改变，还是记录着客户端及服务器端的连接参数。
- 字段6：同字段5。
- 字段7：在连接建立后，这个字段会被设置为“ASSURED”，表示连接已确认。
- UDP协议：

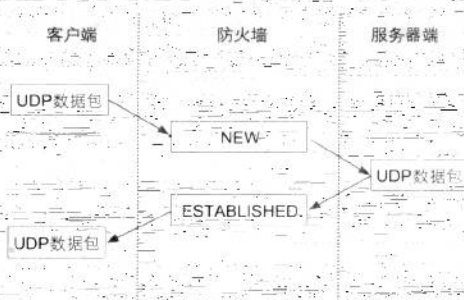


图3-15 state模块下的UDP连接状态解释

在图3-15中可以看到，其实UDP与TCP在state模块的描述下几乎是相同的，下面分析nf_conntrack是如何跟踪UDP数据包的。

表3-9是以DNS的名称解析为例进行说明，首先客户端对服务器端送出一个服务请求包，当这个数据包送达防火墙时，该连接在state模块描述下为NEW状态，因此，nf_conntrack模块随即将该连接信息添加到连接跟踪数据库中，这些信息如表3-9所示。随后分析各个字段的含义。

表3-9 客户端送出DNS的服务请求数据包

字段	内容
1	udp
2	17
3	22
4	src=10.0.1.103 dst=168.95.192.1 sport=32881 dport=53
5	[UNREPLIED]
6	src=168.95.192.1 dst=10.0.1.103 sport=53 dport=32881
7	mark=0 secmark=0 use=1

- 字段1: 记录该数据包是属于UDP协议的数据包。
- 字段2: UDP协议的代码为17, 这个信息可从/etc/protocols文件中找到对应关系。
- 字段3: 当服务器端收到客户端的服务请求数据包之后, 随即应答客户端一个应答数据包, 当这个应答数据包送达防火墙时, 这条连接随即进入ESTABLISHED状态; 但如果在30秒之后依然无法等到服务器端的回应, 那么防火墙将会清除掉这条连接的记录, 而这个30秒是由/proc/sys/net/netfilter/nf_conntrack_udp_timeout这个文件来决定的。
- 字段4: 记录客户端及服务器端的Source IP、Destination IP、Source Port及Destination Port。
- 字段5: 在这个字段记录UNREPLIED, 表示目前尚未收到服务器端的应答。
- 字段6: 由防火墙自动生成, 其依据是第4个字段的反向, 也就是说, 我们从第4个字段中可以推断出服务器端应答给客户端的数据包信息应该是什么, 而这个字段的信息用于判断UDP数据包是否为ESTABLISHED状态的关键指标。

当服务器端回应给客户端的数据包送达防火墙时, nf_conntrack模块随即更新连接跟踪数据库中的内容, 而该连接在state模块描述下则进入ESTABLISHED的状态, 其他各字段的含义分别如表3-10所示。

表3-10 客户端收到DNS的应答数据包

字段	内 容
1	udp
2	17
3	166
4	src=10.0.1.103 dst=168.95.192.1 sport=32881 dport=53
5	src=168.95.192.1 dst=10.0.1.103 sport=53 dport=32881
6	[ASSURED]
7	mark=0 secmark=0 use=1

- 字段1: 记录该数据包是属于UDP协议的数据包。
- 字段2: UDP协议的代码为17, 这个信息可从/etc/protocols文件中找到对应关系。
- 字段3: 当UDP的连接进入ESTABLISHED之后, 如果一直没有进行数据的传输, 那么防火墙即会在180秒后清除该连接的信息, 而180秒是由/proc/sys/net/netfilter/nf_conntrack_udp_timeout_stream这个字段来决定的, 但如果在这180秒尚未到达而有数据传输的操作发生, 这个180秒的值就会重新计算。
- 字段4: 记录客户端及服务器端的Source IP、Destination IP、Source Port及Destination Port。

- 字段5: 由防火墙自动生成, 其依据是第4个字段的反向, 也就是说, 我们从第4个字段中可以推断出服务器端应答给客户端的数据包信息应该是什么。
- 字段6: 在这个字段中标记着ASSURED, 表示这个连接已确认。
- ICMP协议:

从图3-16中我们可以看到, 其实ICMP与TCP在state模块的描述下看起来也几乎是一样的, 但在nf_conntrack模块的跟踪下是完全不相同的, 因为ICMP并不像TCP有“会话(Session)”的概念, 而是以单一数据包为单位, 而且ICMP又不像UDP协议一样会有一段较长的超时时间, 所以我们要在连接跟踪数据库中看到ICMP的跟踪记录就比较困难了。另外

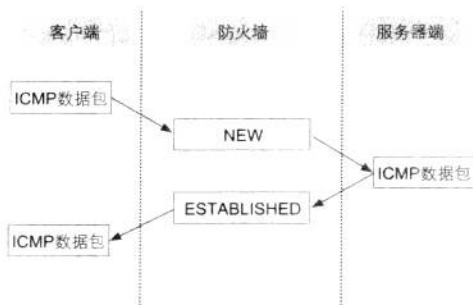


图3-16 state模块下的ICMP连接状态解释

nf_conntrack是以单一数据包为单位来跟踪ICMP数据包, 每当一个ICMP数据包经过防火墙时, nf_conntrack就会在数据库中建立一条跟踪记录; 但在ICMP数据包返回时, nf_conntrack就会从数据库中清除之前的记录, 因此我们必须在ICMP数据包尚未返回的那一瞬间打开nf_conntrack的数据库, 才有机会看到这条ICMP数据包的跟踪记录。如果你真的想看到ICMP的跟踪记录, 可以设法在单位时间内生成大量ICMP数据包(ping -f xx.xx.xx.xx), 这样在nf_conntrack数据库中看到ICMP的记录的机会就大多了, 下列我们来看看nf_conntrack如何跟踪ICMP数据包。

在表3-11中我们可以看到, 客户端送出ICMP并且在这个数据包送达防火墙时, 该连接在state模块描述下为NEW状态, nf_conntrack模块随即在连接跟踪数据库中添加这条连接的信息, 这些字段的含义如下:

表3-11 客户端送出ICMP服务请求包

字段	内容
1	icmp
2	1
3	29
4	src=192.168.0.200 dst=168.95.192.1 type=8 code=0
5	[UNREPLIED]
6	src=168.95.192.1 dst=192.168.0.200 type=0 code=0
7	mark=0 secmark=0 use=1

- 字段1: 记录该数据包是属于ICMP协议的数据包。
- 字段2: ICMP协议的代码为1, 这个信息可从/etc/protocols文件中找到对应关系。
- 字段3: 当客户端送出ICMP请求包之后, 接着会等待服务器端的应答。如果服务器端回送来的数据包到达防火墙, 该条连接在state模块的描述下已经进入ESTABLISHED状态; 但如果没有接到服务器端的应答, 那么防火墙将在30秒之后清除该条连接的记录, 而这个30秒是由/proc/sys/net/netfilter/ip_conntrack_icmp_timeout这个文件来决定的。
- 字段4: 记录客户端及服务器端的IP及ICMP包的类型。
- 字段5: 这个字段记录着“UNREPLIED”的信息, 表示目前尚未收到服务器端的应答。
- 字段6: 由防火墙自动生成, 其依据是第4个字段的反向, 也就是从第4个字段中可以推断出来服务器端应答给客户端数据包信息为何, 而这个字段的数据即是判断ICMP数据包是否为ESTABLISHED状态的关键指标。

看完以上的介绍, 相信你对NEW及ESTABLISHED的状态应该有了更深的了解, 但RELATED及INVALID状态呢? RELATED状态所包含的就比较广泛了, 其含义是客户端对服务器端产生连接A, 但因为连接A的产生而造成了“某一个IP”(当然这个IP一定是与连接A有关的IP, 如连接A所经过的路由器)主动对客户端产生连接B, 而这个连接B的数据包就属于RELATED状态。这里以ICMP Net Unreachable及图3-17为例来解释RELATED状态。

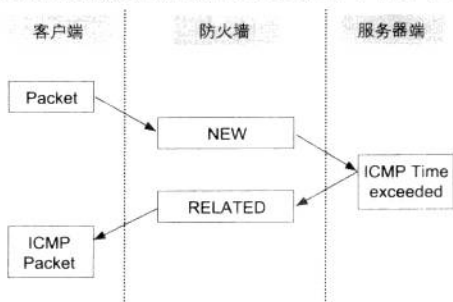


图3-17 state模块下的RELATED连接状态解释

假设客户端对外发送一个数据包(不需要假设这是什么协议的数据包), 当数据包达到防火墙时, 其状态为NEW, 但这个数据包在因特网上传输时TTL已归零, 因此被因特网上的路由器丢弃掉。此时, 丢弃这个数据包的路由器就会对数据包的发送者发送一条ICMP time exceeded错误控制信息, 说明因为TTL已超时, 数据包已被丢弃。

从以上例子可以清楚看到, 从客户端到服务器端一共会有两条连接, 其一是客户端到服务器端的连接, 而这条连接对防火墙而言, 状态为NEW; 其二是从因特网上的路由器到客户端, 而这一条连接对防火墙而言, 就是RELATED状态。不过你必须了解, RELATED状态的数据包未必就是ICMP数据包, 例如FTP协议工作在“主动模式”时, 当客户端对服务器端的TCP Port 21提出服务请求时, 服务器端会主动使用TCP Port 21对客户端建立另一条TCP连接, 而这条TCP连接也是RELATED状态。

最后就是INVALID状态，INVALID状态较容易理解，只要数据包不属于NEW、ESTABLISHED或RELATED，就属于INVALID。而INVALID状态的数据包是属于不应该存在的数据包，例如，数据包在传输的过程因传输错误，而导致数据包状态无法被识别时，该数据包就是INVALID状态；对防火墙的设置而言，只要是INVALID状态的数据包，我们就一律通通丢弃掉。

9. AH及ESP协议的SPI值匹配

IPSec的加密通信中包含了两个协议，分别是AH(Authentication header，认证头)及ESP(Encapsulating Security Payload，封装安全负载)协议。也就是说，IPSec由AH及ESP组合而成。其中AH负责进行数据包的“完整性验证”，例如数据包在网络上传输时有没有遭到篡改，ESP则负责进行数据包的“加密”操作。

不管是AH还是ESP协议，在工作过程中都需要进行“对称式加密”运算，而对称式加密运算一定会需要一把“密钥”，这个密钥是要拿来加密数据用的。假设有三台主机使用IPSec来加密传输中的数据，这三台主机分别为A、B、C，我们再假设主机A与主机B使用Key-A_B来作为加密密钥，主机B与主机C使用Key-B_C来作为加密密钥，因此，主机A的IPSec数据库中会有一把Key-A_B，主机C的IPSec数据库中会有一把Key-B_C，主机B的IPSec数据库中会有一把Key-A_B及Key-B_C，而IPSec为了加快系统查找密钥的速度，会在IPSec的数据库中为每一把密钥加上一个“索引”值，我们称其为SPI值。

接着当主机A发送一个数据包给主机B时，主机B如何知道该拿数据库中的哪一把密钥来解密这个数据包呢？以下我们就来看看IPSec中的AH包头，或许就可以找到答案了。从图3-18可以看到，不管是AH包头还是ESP包头，其内都会含有一个字段叫SPI，而这个SPI值就是告知接收端主机，要使用IPSec数据库中的哪一把密钥来加密这个数据包。如果以上解释你仍然无法理解，也没有关系，我们将在第14章中全面透彻地介绍IPSec，到时再回头来看这些内容，相信你就可以轻松地了解。

No.	Time	Source	Destination	Protocol	Info
6	2.198380	192.168.2.13	192.168.2.12	ESP	ESP (SPI=0x00000201)
7	3.196499	192.168.2.12	192.168.2.13	ESP	ESP (SPI=0x00000201)
8	3.157753	192.168.2.13	192.168.2.12	ESP	ESP (SPI=0x00000201)
9	4.005283	192.168.2.13	192.168.2.12	ESP	ESP (SPI=0x00000201)
Frame 7 (146 bytes on wire (116 bytes captured) on interface 0: Ethernet II, Src: 00:0c:29:ba:25:86, Dst: 00:0c:29:ba:25:86)					
Internet Protocol Version 4, Src: 192.168.2.12, Dst: 192.168.2.13					
Authentication Header					
Next Header: ESP (0x32)					
Length: 24					
SPI: 0x00000200					
Sequence: 2					
ICV					
Encapsulating Security Payload					
SPI: 0x00000201					
Sequence: 2					
Data (80 bytes)					

图3-18 IPSec的包头内容

SPI值的匹配是由`ipt_ah.ko`及`xt_esp.ko`模块所提供的, 我们可以使用这两个模块来匹配AH或ESP包头内的SPI值, 例如, AH包头内的SPI值为300, ESP包头内的SPI值为200, 符合这些条件的数据包可以通过防火墙, 其规则语法如下。

```
iptables -A FORWARD -p ah -m ah --ahspi 300 -j ACCEPT
iptables -A FORWARD -p esp -m esp --espspi 200 -j ACCEPT
```

10. pkttype匹配

在TCP/IP网络环境中, 数据包的传输方式有三种, 如下所示:

- **Unicast:** 数据包发送的对象是特定的, 如主机A传输给主机B即为unicast类型。
- **Broadcast:** 数据包传送的对象为广播地址, 如192.168.0.255。
- **Multicast:** 通常应用于网络的“音频”或“视频”广播, 而Multicast数据包的特点是, 其Source IP一定介于224.0.0.0/4之间。

Pkttype的匹配方式由`xt_pkttype.ko`模块提供, 我们可以使用这个模块来筛选以上三种传输方式的数据包, 其参数分别为multicast、unicast、broadcast。例如, 如果要过滤掉Ping Broadcast数据包, 可使用如下语法。

```
iptables -A FORWARD -i eth0 -p icmp -m pkttype --pkt-type broadcast -j DROP
```

所谓ping broadcast是指ping网段的广播地址, 如ping -b 192.168.0.255, 而这个操作将会使得同一个网段上的所有主机都收到一个ICMP请求包。当然, 所有收到这个ICMP请求包的主机, 也会回应发送一个ICMP应答包, 这样发送端就可以使用一个ICMP数据包来产生253个ICMP数据包(假设该网段上有254台主机)。如果发送端主机连续发送大量ICMP请求包, 当然网段上的ICMP应答包就会更多, 如此即可达到瘫痪网络的目的, 甚至我们可以使用ping broadcast来达到攻击特定主机的目的。例如, 发送端主机在送出ICMP请求包时, 可以伪造来源端IP为要攻击对象的IP, 这样使得网段上的所有ICMP应答包都会回送到被攻击的主机上, 因此, 该主机就会不断收到大量的ICMP应答包, 这样就可以达到瘫痪特定主机的目的。不过, 你大可不必太在意, 因为所有Windows系统都不会去关心ping broadcast的数据包; 不过稍早的Linux系统是会对ping broadcast做出回应的, 因此, 建议你稍微了解一下ping broadcast。

11. length(MTU值)匹配

Length用来匹配数据包的“长度”, 因此我们必须先了解数据包长度的定义。首先要了解每种不同的协议, 其数据包的结构可能都不相同。这里以较为常见的ICMP数据包为例进行说明, 图3-19即为ICMP数据包的结构图, 我们在讨论包长度时经常会听到两个名词,

分别是MTU(Maximum Transmission Unit)及MSS(Maximum Segment Size), 接着让我们来看看MTU及MSS。

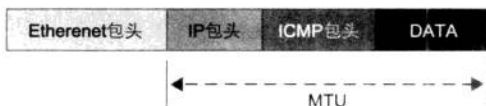


图3-19 数据包长度计算

- MTU: MTU是Maximum Transmission

Unit的缩写, 其所指的是实体网络层每一次所能传输数据大小的上限。在图3-19中, $MTU = (IP\text{包头} + ICMP\text{包头} + DATA)$ 。

- MSS: MSS是Maximum Segment Size的缩写, $MSS = (ICMP\text{包头} + DATA)$ 。

在了解MTU之后, 接下来就可以使用length来匹配特定长度的数据包。以图3-20为例, 这是一个由Windows XP系统所发送出来的ICMP数据包, 从❶标记中我们可以看到IP包头的长度, 另外❷标记则是ICMP包头的数据内容, 不过因为ICMP包头内并没有字段直接标明包头长度, 因此我们需要自己计算一下。首先Type及Code各需要一个字节的空间, Checksum、Identifier及Sequence则各需两个字节, ICMP包头的长度为8个字节, 最后则是ICMP所承载的数据内容❸共有64个字节, 因此 $MTU = (20 + 8 + 64) = 92$ 个字节, 如果我们希望正常的Windows系统可以ping到你的主机, 那么就可以使用如下的语法。

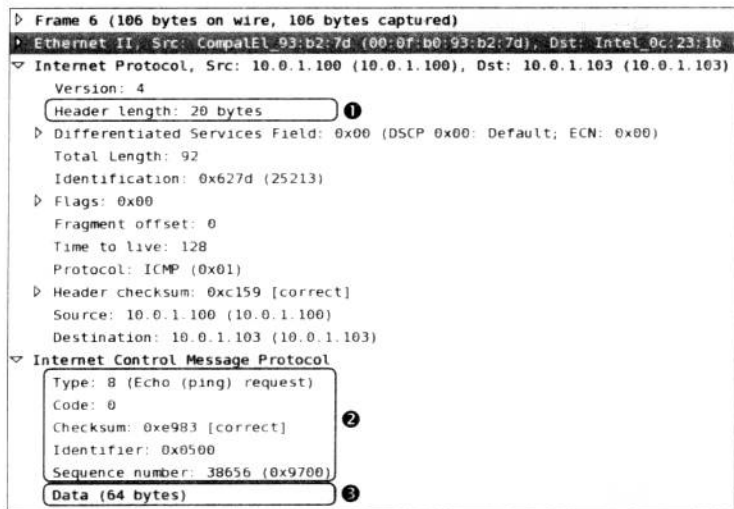


图3-20 ICMP数据包

```
iptables -A INPUT -p icmp --icmp-type 8 -m length --length 92 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 8 -j DROP
```

其中length是由xt_length.ko模块所提供的功能, 其匹配的参数共有以下四种。

- [!] --length 100 匹配MTU值刚好为100个字节的数据包。

- [!] --length 50 匹配MTU值刚好为50个字节的数据包。
- [!] --length :100 匹配MTU值小于100个字节的数据包。
- [!] --length 50:100 匹配MTU值介于50~100个字节的数据包。

或许你会觉得奇怪,为何需要匹配ICMP包的大小?其实这是有必要的,因为ICMP本身就是一个很容易被拿来当做攻击用的协议。例如,我们可以在Linux系统下使用ping -f -s 16384 xx.xx.xx.xx命令来生成大量“超大型”ICMP数据包来撑爆对方的外网带宽,进而达到瘫痪对方网络的目的。其中“-s 16386”是指生成的ICMP数据包,所承载的数据大小为16K字节;或者我们可以在Linux系统下使用ping -f -s 0 xx.xx.xx.xx命令,来生成大量ICMP请求包,进而消耗被攻击主机系统的CPU资源,如此即可达到DOS的攻击目的。

从以上两种攻击手法来看,如果要撑爆对方的外网带宽,那么数据包一定要“足够大”才能达到目的;如果要消耗被攻击主机的CPU资源,那么ICMP请求包的数量一定要“足够多”,而为了足够多,ICMP包一定得足够小,如此才能在有限带宽下,送入更多ICMP数据包,因此,我们只需要允许正常值的ICMP数据包通过即可。不过,ICMP包的DOS攻击并没有想象中那么容易达到,因为攻击者必须拥有足够多的主机及足够大的带宽,才有可能获得成功。

12. limit特定数据包重复率的匹配

在前面的例子中,我们都是把进入防火墙的ICMP包给过滤掉,但其实笔者并不是很建议完全这么做,因为我们把所有的ICMP Type 8的数据包过滤掉的目的是为了避免主机受到大量ICMP包的攻击,但相对地,如果要远程检测自己的服务器,就无法使用ping方式来检测了。如果我们可以开放适量的ICMP包进入,不但不会影响到系统安全,同时也可以远程使用ping命令来检测服务器是否还在网络上正常工作,这样的做法是不是更加完善呢?

要完成上述的机制其实并不难,我们可以通过limit匹配方式来达到目的,而limit是由xtables模块所提供的功能。比方说,我们希望每“分钟”允许进入的ICMP包数量是10个,但如果在1分钟内进来了超过10个以上的ICMP包,那么我们就限制每分钟只能进来6个ICMP包,接着就可以将以上规则写成iptables语法,如下。

```
iptables -A INPUT -p icmp --icmp-type 8 -m limit --limit 6/m \
--limit-burst 10 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 8 -j DROP
```

首先,我们可以看到第一行的6/m,其中m代表minutes也就是以一分钟为单位,再加上“--limit-burst 10”则代表每分钟进入10个数据包,后面的“-j ACCEPT”则代表“允许”的意思。第一行的意思是说“如果每分钟进入的ICMP包数量小于等于10个,是被允许的”,那如果进入的ICMP数据包数量大于10个以上,就限制每分钟只能进来6个,其中第二行的

“--limit 6/m”即为每分钟只能进来6个数据包，其余未经允许的数据包则会进入第二行而遭到丢弃。当设置好以上规则后，就可以从另一台Linux主机发送ICMP数据包来进行测试，得到的结果如下。

```
[root@localhost]# ping 10.0.1.103
PING 10.0.1.103 (10.0.1.103): 56(84) bytes of data:
64 bytes from 10.0.1.103: icmp_seq=1 ttl=64 time=1.99 ms
64 bytes from 10.0.1.103: icmp_seq=2 ttl=64 time=0.175 ms
64 bytes from 10.0.1.103: icmp_seq=3 ttl=64 time=0.154 ms
64 bytes from 10.0.1.103: icmp_seq=4 ttl=64 time=0.246 ms
64 bytes from 10.0.1.103: icmp_seq=5 ttl=64 time=0.590 ms
64 bytes from 10.0.1.103: icmp_seq=6 ttl=64 time=0.099 ms
64 bytes from 10.0.1.103: icmp_seq=7 ttl=64 time=0.168 ms
64 bytes from 10.0.1.103: icmp_seq=8 ttl=64 time=0.136 ms
64 bytes from 10.0.1.103: icmp_seq=9 ttl=64 time=0.175 ms
64 bytes from 10.0.1.103: icmp_seq=10 ttl=64 time=0.135 ms
64 bytes from 10.0.1.103: icmp_seq=11 ttl=64 time=0.576 ms
64 bytes from 10.0.1.103: icmp_seq=21 ttl=64 time=0.513 ms
64 bytes from 10.0.1.103: icmp_seq=31 ttl=64 time=0.203 ms
64 bytes from 10.0.1.103: icmp_seq=41 ttl=64 time=0.519 ms
64 bytes from 10.0.1.103: icmp_seq=51 ttl=64 time=0.230 ms
64 bytes from 10.0.1.103: icmp_seq=61 ttl=64 time=0.188 ms
64 bytes from 10.0.1.103: icmp_seq=71 ttl=64 time=0.155 ms
64 bytes from 10.0.1.103: icmp_seq=81 ttl=64 time=0.593 ms
64 bytes from 10.0.1.103: icmp_seq=91 ttl=64 time=0.441 ms
```

其中的icmp_seq=1即代表发送出来的第几个ICMP数据包，可以看到从icmp_seq=1到icmp_seq=10，刚好为我们所允许的10个数据包，但是当icmp_seq=11进入时，就已经超过我们规则的临界点了，这时limit限制就会开启，接着会看到icmp_seq=11到icmp_seq=21之间，总共掉了10个数据包，而ping命令又刚好每一秒钟送出一个ICMP包，因此，我们可以推断“每10秒只会允许1个ICMP进入”，如此，一分钟之内就只能进来6个ICMP数据包刚好等于“--limit 6/m”的要求。

如果发送端停止发送ICMP包，limit何时会解除6/m的限制呢？计算方法是把6/m的6和--limit-burst 10相乘，即会得到答案60，也就是说，60秒之后limit就会完全解除这个限制。

最后要注意的是如果没写--limit-burst，那么默认值就是--limit-burst 5，除--limit 6/m(minutes)之外，还有s(second)、h(hour)及d(day)可以使用。

13. recent特定数据包重复率匹配

在看完limit模块，你可能还沉醉于它强大的功能之中，不过，当了解recent的匹配方式后，你可能就不会觉得limit有何特别之处了，因为recent提供了一个“先取证”、“后处理”的特殊机制，这可以设计出多样化的匹配方式。例如，同一个用户在某一段时间

内,对服务器只能发起多少次的SSH服务请求,又如recent可以有效防止网络上各种端口扫描(port scan)攻击,总之, recent所能做的事情非常多,但也有赖于你发挥想象力!

recent的匹配方式是由xt_recent.ko模块所提供的功能,另外,这个模块包含的参数非常多,如表3-12所示,这些参数都很重要,因此,请务必了解每个参数的用途。

表3-12 recent模块的参数列表

参数名称	说明
--name	设置跟踪数据库的文件名
[!]=set	将符合条件的来源端数据添加到数据库中,但如果来源端数据已经存在,则更新数据库中的记录信息
[!]=rcheck	只进行数据库中信息的匹配,并不会对已存在的数据做任何变更操作
[!]=update	如果来源端的数据已存在,则将其更新;若不存在,则不做任何处理
[!]=remove	如果来源端数据已经存在,则将其删除,若不存在,则不做任何处理
[!]=seconds second	当事件发生时,只会匹配数据库中前“几秒”内的记录,--seconds必须与--rcheck或--update参数共用
[!]=hitcount hits	匹配重复发生的次数,必须与--rcheck或--update共用

接下来以ICMP的流量控制为例进行说明。假设我们希望每分钟只能进来6个ICMP数据包,如果超过这个数量就将之丢弃掉,将规则编写如下。

```
#!/bin/bash
iptables -F
iptables -A INPUT -p icmp --icmp-type 8 -m recent --name icmp_db \
--rcheck --second 60 --hitcount 6 -j DROP
iptables -A INPUT -p icmp --icmp-type 8 -m recent --set --name icmp_db
```

当以上规则生效后,我们就可以在其他主机上使用ping命令来测试这个防火墙规则,以下就是测试结果。由于ping命令刚好每一秒送出一个icmp的请求包,因此可以从icmp_seq这个值来观察“时间”的间隔。看到前6秒有连续6个icmp请求包送入防火墙主机内,接下来的icmp请求包似乎没有办法送进防火墙主机,所以icmp包的发送端主机并没有收到防火墙主机所应答回来的数据包,但是到了第61秒时,又有连续6个icmp请求包可以正常地送入防火墙主机。由以上的实验可以证明,这个防火墙的规则是符合我们需求的。

```
[root@localhost ~]# ping 10.0.1.101
PING 10.0.1.101 (10.0.1.101) 56(84) bytes of data:
64 bytes from 10.0.1.101: icmp_seq=1 ttl=64 time=0.101 ms
64 bytes from 10.0.1.101: icmp_seq=2 ttl=64 time=0.135 ms
64 bytes from 10.0.1.101: icmp_seq=3 ttl=64 time=0.537 ms
```



```

64 bytes from 10.0.1.101: icmp_seq=4 ttl=64 time=0.146 ms
64 bytes from 10.0.1.101: icmp_seq=5 ttl=64 time=0.117 ms
64 bytes from 10.0.1.101: icmp_seq=6 ttl=64 time=0.139 ms
64 bytes from 10.0.1.101: icmp_seq=61 ttl=64 time=0.159 ms
64 bytes from 10.0.1.101: icmp_seq=62 ttl=64 time=0.138 ms
64 bytes from 10.0.1.101: icmp_seq=63 ttl=64 time=0.120 ms
64 bytes from 10.0.1.101: icmp_seq=64 ttl=64 time=0.137 ms
64 bytes from 10.0.1.101: icmp_seq=65 ttl=64 time=0.131 ms
64 bytes from 10.0.1.101: icmp_seq=66 ttl=64 time=0.123 ms
64 bytes from 10.0.1.101: icmp_seq=121 ttl=64 time=0.120 ms
64 bytes from 10.0.1.101: icmp_seq=122 ttl=64 time=0.136 ms
64 bytes from 10.0.1.101: icmp_seq=123 ttl=64 time=0.174 ms

```

接下来分析规则的语法和含义，首先请先看第二条规则，这条规则是说：“如果进来的数据包是icmp type 8，就使用recent模块来处理这个数据包，并将符合规则的数据包相关信息记录到名为icmp_db的数据库之中”，而这个数据库会存放在/proc/net/xt_recent目录下，并以使用者所设置的文件名来存储，因此，这时我们应该可以看到/proc/net/xt_recent/icmp_db这个文件。

```

1. iptables -A INPUT -p icmp --icmp-type 8 -m recent --name icmp_db \
    --rcheck --second 60 --hitcount 6 -j DROP
2. iptables -A INPUT -p icmp --icmp-type 8 -m recent --set --name icmp_db

```

在了解第二条规则的含义之后，我们接着说明这组防火墙规则是如何工作的。首先，当第一个icmp请求包进来时，先交给第一条规则来进行匹配，而第一条规则的含义是说：“当进来的数据包是icmp type 8时，就使用recent模块来处理这个数据包，而我们以icmp_db这个数据库作为匹配的依据，并且以目前这个时间点向前搜索60秒内的数据来做匹配，如果在这个60秒之内已有超过6个以上的符合记录，那就将目前符合规则的这个icmp包丢弃掉”；由于目前这个icmp包是第一个服务规则的数据包，在icmp_db数据库中根本不会有符合这条规则的任何记录，所以这个数据包会送入到第二条规则来进行匹配；这时候，第二条规则会将这个数据包的相关信息记录到icmp_db数据库中，因此这个数据包的相关信息就被记录在第1秒的位置。另外需要注意，虽然这个数据包符合了两条规则，但这个数据包依然会送到下一条规则来匹配，也就是说第二行规则只会做记录操作，并不会去“处理”这个数据包。

当第二个icmp包进来时，同样先进行第一条规则的匹配，而此时时间点是在第2秒的位置上，接着recent模块从icmp_db数据库内向前搜索60秒内的信息出来匹配，不过，只能匹配到前一秒有符合规则的相关记录数据，所以也没有符合第一条规则60秒内有6次的符合记录；于是数据包又送入到第二条规则来进行匹配，第二条规则会将第二个数据包的信息记录到数据库中，所以目前数据库内应该有两条记录，分别为第1秒时的记录和第2秒时的记录。

这里假设每一秒刚好进来一个icmp包,因此,如果连续6秒内都有icmp包进来,当然这6个icmp包的相关信息都会被记录到数据库之中。前1-6秒都会有记录被存放到数据库之中,这时若在第7秒有第7个icmp包进来,此时,第一条规则则会从第7秒的位置向前搜索60秒内的数据以匹配,不幸的是,这第7个icmp包已经超出了我们允许的上限,也就是超出了--hitcount 6的上限而遭到丢弃,所以从第7秒~第60秒这段时间内,不管这个来源端送多少个icmp包进来,都会在第一规则中被丢弃掉。不过,请特别注意,第7秒~第60秒内的icmp包已经被“丢弃掉”,因此,在这段时间之内就不会有icmp包进入到第二条规则,所以我们的数据库在第7秒~第60秒之间,都不会有新的数据被记录进来。

为此,时间到了第61秒时有一个icmp包进来,当然还得交给第一条规则来匹配。因此,第一条规则同样会向前搜索60秒内的数据出来匹配,不过,这时已经是第61秒了,所以向前搜索时只会搜索到5个符合的记录(第2秒~第6秒的记录);当然这第61个进来的数据包就不会符合第一条规则,接着就被送到第二条规则来匹配,这时候,数据库在第61秒时又有新的记录被记载下来了,接着第62秒~第66秒进来的数据包也会被记录下来,当第67秒进来的数据包到达第一条规则时,第一条规则再向前搜索60秒,这时应该是在第67秒,所以向前搜索的范围应该是第6秒~66秒之间,所以数据库内的记录又到达每分钟6个icmp包的上限,因此,第67个包又会被丢弃掉,如此周而复始就会出现刚刚测试的结果。

经过以上一长串的介绍之后,相信你应该可以轻松掌握recent中的5个参数了。如果发送端不停发送ICMP包,我们是否可以不断地更新最后一个进来的ICMP包信息?如果以上诉求是可行的,那么,对规则的测试结果就不会像之前一样,每隔60秒就可以进来6个icmp包,而是只要icmp包的发送端不停止icmp包的发送,这些icmp包就永远别想再进入到防火墙之中。因为我们的数据库会不断地更新,以上这个诉求是可行的,不过,就得使用到另一个参数~update,这里将前一个示例的规则更改如下。

```
1. iptables -A INPUT -p icmp --icmp-type 8 -m recent --name icmp_db \
    --update --second 60 --hitcount 6 -j DROP
2. iptables -A INPUT -p icmp --icmp-type 8 -m recent --set --name icmp_db
```

接着来看看recent数据库中的一些信息,以下即为recent数据库的内容,其实这个数据库的内容我们并不需要去管理它,由recent模块自行维护即可,而这些数据会在每一次重新执行防火墙的规则数据库(Shell Script)时自动归零。另外,也可以看到这一行记录中的第一个字段,这个字段会记录数据包的来源端IP。recent是以独立的客户端来执行其匹配操作,而limit则是以进来的数据包总量来进行限制,因此recent模块的匹配方式要比limit模块更灵活。

```
[root@localhost TEST]# cat /proc/net/xt_recent/icmp_db
src=10.0.1.102 ttl: 64 last_seen: 419239815 oldest_pkt: 1 419239815, 411210101,
411211088, 411212109, 411213090, 411214116, 411269144, 411270171, 411271131, 411272170,
411273138, 411274164, 411329153, 411330178, 411331175, 419234807, 419235792, 419236811,
419237789, 419238821
```

最后一点请务必注意，在recent的数据库之中，默认最多只能记录100条视图入侵者的IP。在数据库已满的情况下，新的数据将会取代旧的数据，但如果我们所希望记录的IP超过100笔以上时，该如何处理呢？这个问题很容易解决，我们只要改为手动的方式来载入recent模块，并结合recent所提供的参数，就可以轻松解决这个问题，方法如下。

```
[root@fw ~]# modprobe xt_recent ip_list_tot=1024
```

其中ip_list_tot=1024即为设置所要记录的IP上限为1024条，另外，还有一个参数也必须了解，这个参数名为ip_pkt_list_tot。前面的例子曾提到：“如果在1分钟之内有超过10个以上的icmp包进入，我们就……”，其中“10个以上的icmp包”即代表recent模块会记录这10个icmp包的信息，也就是说，recent模块的数据库必须要有足够的“空间”来存放这10个数据包的信息，但不幸的是，这些空间的默认最大值为20，因此，如果在每一个IP上记录其进来的次数要超过20次以上，我们就可以使用ip_pkt_list_tot参数，其使用方法如下。

```
[root@fw ~]# modprobe xt_recent ip_list_tot=1024 ip_pkt_list_tot=50
```

14. IP包头内TOS值的匹配

在IP包头中有一个字段叫做TOS(type of service)，如图3-21所示。TOS位在早期是用来作为QOS系统中的数据包分类依据，不过较新的RFC文件(RFC 2474)已经将TOS改为DSCP(Differentiated Service)，其用途也是作为QOS的数据包分类用。第2.13一节中曾经讲到过这一点，如果你忘记了的话，请回头再复习一次。

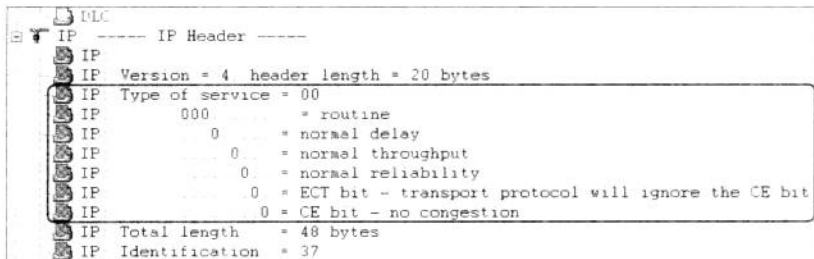


图3-21 TOS

tos的匹配方式是由ipt_tos.ko模块所提供的功能，但由于这个字段已不再定义为TOS

了,因此这里就不再介绍这个模块的使用方式。如果你需要匹配这个字段的数据,可以直接将这个字段当做DSCP进行匹配即可。

15. 使用String模块匹配数据包内所承载的数据内容

第1章中曾提到过,如果服务本身存在安全漏洞,黑客就可能运用服务本身的漏洞来达到入侵系统的目的。以图3-22的IIS Web 服务器为例,黑客可以运用IIS本身的漏洞以URL方式来访问IIS 服务,帮我们启动服务器硬盘上的某一个执行文件,如此即可能达到入侵的目的。例如,黑客可以运用此方式在我们的Web 服务器上执行FTP 客户端,并通过FTP 客户端到特定的FTP 服务器上下载如Virus.exe的病毒程序回来,接着以相同方式运行下载回来的Virus.exe执行文件,如此达到入侵的目的。然而这样的入侵方式在一般的“包过滤防火墙”上无法解决,因为包过滤防火墙的匹配条件是“数据包的包头内容”,而非“数据包内所承载的数据内容”,因此,在商用版防火墙之中一定要使用“应用层防火墙”才有办法解决这一类的入侵行为,一旦启用了应用层防火墙,因为应用防火墙的检查速度较慢,势必会造成Web 客户端到Web 服务器之间的连接反应速度变慢。不过,可以在Netfilter/Iptables上得到圆满解决这个问题。

string是由xt_string.ko模块所提供的匹配功能,string匹配方式可在“网络层”的位置直接匹配数据包所承载的数据内容,而无需将数据包送到“应用层”的位置才能进行匹配的操作,因此string的匹配方式会比“应用层防火墙”更高效,并且比较而言不太占用系统的存储空间。但string的匹配范围仅局限于单一数据包之内,也就是说,如果要匹配的“特征”是分散在两个数据包之内,此时,string的匹配方式就会完全失效。不过,几乎所有的服务请求操作都可可在一个数据包之内完成,因此,使用者就不必太过于担心,下面分析string到底是如何工作的。

再以图3-32为例,我们假设黑客与IIS Web 服务器之间有一台网关式防火墙,因此,可在网关式防火墙上使用如下语法来解决这个令人头痛的问题。

```
iptables -A FORWARD -i eth0 -o eth1 -p tcp -d $WEB_SERVER --dport 80 \
-m string --algo bm --string "system32" -j DROP
```

语法的意思是说:“如果数据包是要送往Web 服务器的端口80,那我们就使用string模块来进行匹配的操作,而匹配的操作是用bm(Boyer-Moore)算法进行处理的;如果这个数据包内包含‘system32’字符串的话,就将这个数据包丢弃掉”。一旦这个规则生效,只要客

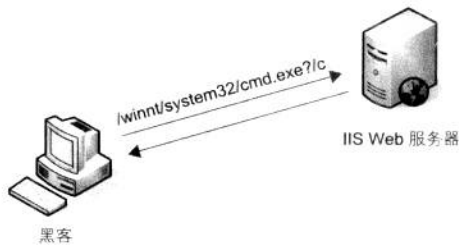


图3-22 入侵Web服务器

户端对Web 服务器送入的请求数据包内包含“system32”字符串，数据包就会被丢弃掉。不过，有一点要特别提醒你，别忘了，在Linux系统上字符是区分大小写的。

有了这个功能之后，即使Web 服务器存在这样的安全漏洞，所有针对这个漏洞而来的入侵行为都会在防火墙上遭到拦截，虽然Netfilter/Iptables可以帮我们抵挡这类攻击行为，但为求慎重，还是建议定时更新你的系统。

最后将string模块所提供的匹配条件整理到下表。

参数名称	功能说明
--algo	字符串匹配算法的选择，string模块提供了两种不同的算法，分别是bm(Boyer-Moore)及kmp(Knuth-Pratt-Morris)，其中bm算法平均速度会比kmp快，不过，你也不必太在意，因为我们匹配的对象不会太大，因此用哪种都差不多
--from、--to	设置匹配字符串的范围，我们可以使用--from来设置匹配的起点，并以--to来设置匹配的终点，其单位为字节，如--from 10意为从第10个字节开始匹配，如果没有设置这个参数，那么匹配的范围将是整个数据包
[!]-string	匹配条件，如--string "system32" 是指要匹配的字符串为system32
[!]-hex-string	匹配条件，但不同于--string参数的地方在于--hex-string是以16进制的方式进行匹配，特别适用于非ascii字符串的匹配，其匹配条件的表示方法为--hex-string " 2e2f303132333435 "，请注意实际匹配条件为2e2f303132333435，但其左右要使用 " " 符号

16. 使用connlimit模块限制连接的最大数量

connlimit机制由xt_connlimit.ko模块提供，用于以/proc/net/nf_conntrack文件中的数据为依据，来限制一个IP或一个网段同时对目标主机或服务所能建立的最大连接数。或许有恶意的使用者通过telnet或wget之类的网站复制软件来复制你的网站，这类软件通常会极尽可能地对你的服务器建立庞大的连接数以加快其复制网站的目的，这样的行为将会大量消耗服务器的资源及外网带宽，使得想要浏览网页的正常使用者无法或难以浏览网页，在此就可以运用connlimit来进行连接数量的限制。

不过在进行连接限制时得先了解一件事：在浏览受保护的网页时，浏览器与网页服务器之间所需要的最大连接数量是多少？如此才能不至于把连接数量设置得太低而干扰到正常的网页浏览行为。但问题是该如何得知网页所需要的最大连接数呢？这里推荐你使用Fiddle(<http://www.fiddler2.com>)这套工具软件来测试你网页所需的连接数。另外，我们还得考虑到NAT的使用问题，因为现在大多数企业都是通过NAT来连接因特网，因此如果某企业内有10个用户在浏览你的网站，NAT对外的IP对你的网站将会产生比正常值多10倍的连接数量，这点你必须谨慎考虑，建议你可以结合第2章及第3.2.2一节第2部分的内容来跟踪连接数量的设定值是否需要调整。

下面以网站为例来说明connlimit模块的用法。首先使用Fiddler软件测得网站上所有网页连接的最大值为10, 另外假设在同一个NAT下同时会连上来浏览网页的人数为3, 如此估算一下连接所需的数量大约为30, 因此, 可得到以下规则。

```
iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn -d $Web_Server --dport 80 \
--m connlimit --connlimit-above 30 --connlimit-mask 32 -j DROP
```

其中connlimit的参数整理如下表。

参数名称	功能说明
[!]--connlimit-above	指定最大连接数量
--connlimit-mask	此参数为子网络掩码, 用于匹配范围, 例如 --connlimit-mask 8代表一个A类子网 --connlimit-mask 16代表一个B类子网 --connlimit-mask 24代表一个C类子网 --connlimit-mask 25代表1/2个C类子网 --connlimit-mask 32代表单一一个IP

17. 使用connbytes模块限制每个连接中所能传输的数据量

connbytes机制是由xt_connbytes.ko模块所提供, 目的是用来限制单一连接中所能传输的数据量上限, 由此限制使用者在一条连接上长时间的大量数据传输, 例如下载超大文件、观看网上视频等。

如图3-23所示, 我们以connbytes模块为例, 说明如何限制使用者以HTTP协议下载20M以上的数据。

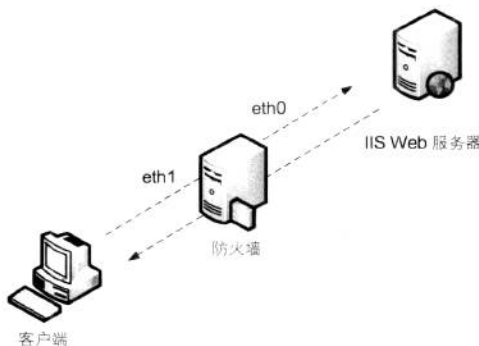


图3-23 connbytes示例

```
iptables -A FORWARD -p tcp -d $WEB_SERVER --dport 80 -m connbytes \
--connbytes-dir reply \
--connbytes-mode bytes \
--connbytes 20971520: -j DROP
```


以上规则使用connbytes模块来进行流量统计，其中--connbytes mode bytes是指以传输的数据量来进行统计，因为我们要统计使用者下载的数据量，因此以--connbytes-dir reply来指定只统计服务器返回给客户端的数据，而无需统计客户端送给服务器端的数据，最后以--connbytes 20971520:来指定大于20MB($1024*1024*20=20971520$)以上时将数据包丢弃掉。下表整理了connbytes模块的其他参数。

参数名称	功能说明
[!] <code>--connbytes-dir</code>	<p>original: 来源方向</p> <p>reply: 应答方向</p> <p>both: 双向</p> <p>以图3-23为例，如果我们要统计客户端发送了几个数据包给服务器端，就可以使用--connbytes-dir original；如果要统计服务器端回送了多少个数据包给客户端，可以使用--connbytes-dir reply；如果要统计客户端与服务器端之间共传输了多少数据，可以使用--connbytes-dir both</p>
<code>--connbytes-mode</code>	<p>packets: 以数据包的数量来计算</p> <p>bytes: 以传输的数据量来计算</p> <p>要以哪种单位进行统计</p>
<code>--connbytes</code>	<p>10: 匹配10个以上的单位量</p> <p>:50 匹配50个以下的单位量</p> <p>10:50 匹配10个~50个之间的单位量</p>

不过，在实际应用中我们不太可能采取以上策略，因为使用者在断网后很可能以为是网络不稳定而进行第二次、第三次甚至更多次的重试操作，如此不但没有限制到使用量，反而浪费掉更多的流量。因此，最好结合QOS进行连接控制。例如，当连接中传输的数据量大于20M时，即对该条连接进行带宽的限制，如此才会是比较好的解决方法。

18. 使用quota模块限制数据传输量的上限

相信有网络管理经验的读者们一定都有相同的困扰，如果企业内有使用者一天到晚在因特网上大量上传下载数据，往往会严重影响其他使用者的网络使用流畅度，此时我们就可以使用quota模块来限制每个使用者每小时或每天所能够传输的数据总量。

```
iptables -A FORWARD -i eth0 -o eth1 -p tcp --sport 80 \
-m quota --quota 524288000 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -p tcp --sport 80 -j DROP
```

在上面的例子中，我们使用quota模块来限制每一个IP使用http通信协议只能下载500MB的数据量。其中--quota 524288000即为限制条件，而数字单位则是字节，再有quota的匹配方式为

“总量累计小于条件时成立”。因此，上述例子中需要两行规则才能完成这一功能。

另外需要注意，被限制的IP一旦累积到设置的量之后，这笔记录将会永久保存直到防火墙上规则被清除为止。因此，如果我们要限制使用者每天所能下载的数据总量，请记得在系统调度中设置每天刷新(refresh)防火墙规则一次，方法是在/etc/crontab中加入如下一行。

```
01 1 * * * root /root/firewall.sh
```

其中第一个字段是指调度执行时间为每天凌晨一点整，第二个字段是指以root身份来执行这项任务，而第三个字段则指要执行的内容是/root/firewall.sh这个Shell脚本。

19. 使用time模块来设置规则的生效时间

time机制由xt_time.ko模块所提供，用来设置规则的生效时间。例如，我们可以使用time模块来限制公司内服务器群(Server farm)可被访问的时间是周一~周五的早上九点~晚上九点，以降低服务器群遭受攻击的概率，规则语法如下。

```
iptables -A FORWARD -o eth1 -d $SRV_FARM -m time \
--weekdays Mon,Tue,Wed,Thu,Fri \
--timestart 09:00 --timestop 21:00 -j ACCEPT
iptables -A FORWARD -o eth1 -d $SRV_FARM -j DROP
```

其中--weekdays是指定一个星期中的哪几天，--timestart及--timestop为起始及结束的时间，而时间格式为24小时制。下表整理了与time模块有关的其他参数。

参数名称	功能说明
--datestart --datestop	包含日期的时间范围表示法 格式: YYYY[-MM[-DD[Thh[:mm[:ss]]]]] 例如2010年9月1号表示为2010-09-01T00:00:00
--timestart --timestop	不包含日期的时间范围表示法 格式: hh:mm[:ss] 例如下午两点整表示为14:00
[!]-monthdays	设置在一个月之中的特定日期 格式: day[,day]... 例如是一个月之中的1、9、19、29及31五天，就表示为1,9,19,29,31，如果被匹配的月份中没有31号，则time模块会自动将其忽略
[!]-weekdays	设置在一个星期之中的特定几天 格式: day[,day]... 例如一个星期中的一、三、五表示为Mon,Thu,Fri，请注意必须使用time模块的特定字符串来表示，分别是Mon、Tue、Wed、Thu、Fri、Sat和Sun

20. 使用connmark模块来匹配mark值

connmark机制是由xt_connmark.ko模块所提供的功能必须与CONNMARK“处理方法”结合使用，你可以先参阅第3.2.2一节的第7部分，connmark与CONNMARK的组合就是有点像mark与MARK的组合，相关信息请参阅第3.1.2一节的第4部分。这两种组合之间的差异如下。

- “mark匹配方式”与“MARK处理方法”的组合：

MARK处理方法所设置的mark值的有效范围仅局限于一个数据包。因此，若要对连接单一方向的所有数据包设定mark值，我们必须单独为每个数据包来设置mark值；要为连接上双向的所有数据包设置mark值，就必须借助CONNMARK的功能才有办法实现。

- “connmark匹配方式”与“CONNMARK处理方式”的组合：

CONNMARK处理方法就是为了弥补MARK的不足而被设计出来的，CONNMARK是对一整条连接来设置mark值的。也就是说，只要连接中的某一个数据包被标记了mark，那么，其后该连接双向的所有数据包都会自动设置这个mark值。

虽然MARK与CONNMARK所标记的值都叫mark值，但其中还是有差异的，MARK所标记的值我们称之为nfmark，而CONNMARK则称为ctmark。另外有一点你得特别留意，mark匹配方式只识别nfmark，而connmark则可识别nfmark及ctmark。

connmark匹配方式的使用语法如下：

```
iptables -A INPUT -m connmark --mark 1 -j DROP
```

21. 使用conntrack模块匹配数据包的状态

conntrack模块可视为state模块的加强版，且使用方法与state模块如出一辙，但其功能更强大，语法使用也更灵活，因此conntrack的参数比state模块更多。为便于你理解conntrack每一个参数的含义，我们以图3-24来辅助说明。



图3-24 conntrack参数说明

参数名称	功能说明
[!] <code>--ctstate</code>	匹配数据包的状态, 状态列表分别为NEW、ESTABLISHED、RELATED、INVALID、DNAT及SNAT。其中前四种状态与state模块的状态完全相同, DNAT及SNAT则为新的匹配状态, DNAT状态是用来匹配某一条连接是否经过DNAT的处理, 而SNAT则是匹配某一条连接是否经过SNAT的处理。请注意这两种状态匹配仅适用于执行NAT的主机上
[!] <code>--ctproto</code>	用于匹配OSI七层中第四层的通信协议, 其功能与用法就如同iptables中的-p参数, 如-p tcp、-p udp、-p 47等
[!] <code>--ctorigsrc</code>	匹配连接发起方向的来源端IP
[!] <code>--ctorigdst</code>	匹配连接发起方向的目的端IP
[!] <code>--ctreplsrc</code>	匹配数据包应答方向的来源端IP
[!] <code>--ctrepldst</code>	匹配数据包应答方向的目的端IP
[!] <code>--ctorigsrcport</code>	匹配连接发起方向的来源端口
[!] <code>--ctorigdstport</code>	匹配连接发起方向的目的端口
[!] <code>--ctreplsrcport</code>	匹配数据包应答方向的来源端口
[!] <code>--ctrepldstport</code>	匹配数据包应答方向的目的端口
[!] <code>--ctexpire</code>	连接在Netfilter Conntrack数据库(/proc/net/nf_conntrack)的存活时间, 使用方法如下: 匹配特定的存活时间--ctexpire time 匹配特定区间的存活时间--ctexpire time:time
--ctdir	设置要匹配哪个方向的数据包, 使用方法如下: 只匹配连接发起方向的数据包--ctdir ORIGINAL 只匹配数据包应答方向的数据包--ctdir REPLY 若没有设置这个参数, 默认会匹配双向的所有数据包

在此举例说明conntrack与state的用法差异。在此之前如果我们要限制只有192.168.1.0/24网段的主机可以访问FTP服务, 我们可能会使用以下防火墙规则。

```
#!/bin/bash
iptables -F
modprobe nf_conntrack_ftp
iptables -A INPUT -p tcp -m state --state \
    ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp -s 192.168.1.0/24 --dport 21 \
    -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --dport 21 -j DROP
```

在改用conntrack模块之后, 由于conntrack模块中有包含iptables或其他模块的功能, 我们只要使用conntrack模块即可完成所需的规则。

```
#!/bin/bash
iptables -F
```

```
modprobe nf_conntrack_ftp
```

```
iptables -A INPUT -m conntrack --ctstate \
    ESTABLISHED,RELATED -j ACCEPT
```

```
iptables -A INPUT -m conntrack --ctproto tcp --ctorigsrc 192.168.1.0/24 \
    --ctorigdstport 21 --ctstate NEW -j ACCEPT
```

```
iptables -A INPUT -p tcp --dport 21 -j DROP
```

22. 使用statistic模块进行比率匹配

第3.1.2一节的第13部分使用recent模块来防范可能的icmp攻击。不过，要达到相同甚至更好的防范效果其实有更好的选择。statistic机制是由xt_statistic.ko模块所提供的功能，其功能是在必要的时候以随机或者规律的方式丢弃部分数据包。下面列举两个示例来说明statistic模块的用法。

示例3.7 随机丢弃50%送到本机的icmp包

如下规则中--mode random与--probability 0.5参数指以随机方式丢弃50%的数据包。

```
iptables -A INPUT -p icmp -m statistic --mode random --probability 0.5 -j DROP
```

示例3.8 以规律的方式在每10个icmp包中丢弃1个icmp包

如下规则中--mode nth与--every 10参数指按一定规律在每10个icmp包中丢弃掉1个icmp包。

```
iptables -A INPUT -p icmp -m statistic --mode random --probability 0.5 -j DROP
```

statistic模块的完整参数如下表。

参数名称	功能说明
--mode	设置statistic模块的工作方式，分别如下： random：以随机方式丢弃数据包 nth：按一定规律丢弃数据包
--probability	此参数需结合random模式使用，例如--probability 0.4 即代表丢弃40%的数据，其中的数值为0~1
--every	此参数需结合nth模式使用，例如--every 10代表在每10个数据包中要丢弃1个数据包
--packet	此参数需要在nth模式与--every参数结合使用，例如--every 10 --packet 5，因为statistic会在我们下达完iptables命令之后马上执行计数操作，而--packet 5则指在忽略前5个数据包之后才开始计数

23. 使用hashlimit模块进行重复率匹配

hashlimit的功能与limit模块的功能雷同但功能更强大。从某些方面来说, 可视为recent的初级版, 而其主要的功能为限制特定行为的重复率。下面的示例说明了hashlimit模块的用法。

```
iptables -A INPUT -p icmp -m hashlimit \
    --hashlimit-name ICMP \
    --hashlimit-burst 5 \
    --hashlimit-upto 6/minute \
    --hashlimit-mode srcip \
    --hashlimithtable-size 8192 \
    --hashlimithtable-expire 60000 \
    --hashlimithtable-gcinterval 5000 \
    -j ACCEPT

iptables -A INPUT -p icmp -j REJECT
```

这里自编的一个故事来解释hashlimit模块工作的方法。在公园里有一个卖棉花糖的老伯伯, 他平均每分钟可以制作出6支棉花糖, 但因为手推车比较小, 因此, 车上只有摆放5支棉花糖的空间, 在空间填满了之后就暂停棉花糖的生产工作。这位老伯伯制定了一个棉花糖购买规则, 凡是要购买棉花糖的人请排成一列, 并请购买者一直往前走不可停留, 当你到老伯伯面前时, 如果有制作好的棉花糖, 将可以买到棉花糖并欢心欢喜地离开(ACCEPT); 万一走到老伯伯面前时刚好没有棉花糖, 那你也不准停留, 只能失望地离开(匹配下一条防火墙规则而被DROP掉); 但如果排队的人都走光了, 老伯伯还是会继续制作他的棉花糖直到把车上的空间再度放满时, 以便再次应付下一波突如其来的大量顾客。

由以上的故事可以得知, 5支棉花糖就是老伯伯的缓冲区, 可以用来应付突如其来的较大流量, 而老伯伯每分钟可生产6支棉花糖, 这6支棉花糖就是平均的放行速度, 就如同上述示例中的--hashlimit-burst 5及--hashlimit-upto 6/minutes。--hashlimit-name参数用来设置本规则存放记录的数据库名称, 这个数据库将会存在于/proc/net/ipt_hashlimit/目录下; --hashlimit-burst用来设置缓冲区Quota的量, 就相当于老伯伯的车上最多只能存放5支棉花糖; --hashlimit-upto参数用来控制icmp数据包进入的“平均”速度, 例如--hashlimit-upto 6/minute为每分钟6个icmp包。接下来, --hashlimit-mode则是指我们要以来源端的IP作为统计及上限的依据; --hashlimithtable-size则是用来设置ICMP数据库所能存放数据的上限, 但请注意实际所能存放的数据为--hashlimithtable-size所设置的8倍, 也就是说, 在这个例子中实际所能存放的数据为8192*8=65536笔; --hashlimithtable-expire用来设置数据库中某一条数据过期的时间, 而其时间的计算方式为“从最后一个符合本规则的数据包, 进入系统的时

间开始累加”，其时间单位为1毫秒；不过，在设置expire时间时要特别注意一件事，就是expire的时间不要低于我们所限定的单位时间，以这个示例来说，其时间单位为minute，因此，expire的时间笔者设为 $1*60*1000=60000$ ；--hashlimithtable-gcinterval则是定义在数据expire之后的多少时间内将之清除，其单位时间也为1毫秒。最后凡没有违反本规则的数据包都放行，其余的数据包则交由下面的防火墙规则来进行处理。

下表列出了hashlimit模块的所有参数。

参数名称	功能说明
--hashlimit-upto	在单位时间内符合条件的数据包数量超过几个以上，其单位时间可为N/second, N/minute, N/hour及N/day
--hashlimit-above	在单位时间内符合条件的数据包数量低于几个以下，其单位时间可为N/second, N/minute, N/hour及N/day
--hashlimit-burst	设置缓冲区大小
--hashlimit-mode	设置匹配依据，依据可为srcip、srcport、dstip及dstport
--hashlimit-srcmask	若没有设置此参数，hashlimit将会以来源端的IP作为判断及限制依据；但如果加入此参数，hashlimit将会以网段方式来作为判断及限制的依据，例如我们设置hashlimit-srcmask参数为24(255.255.255.0)时，来自192.168.1.10及192.168.1.130两个IP的行为将会被归类到相同一条记录内，因此数据库只会生成一笔记录
--hashlimit-dstmask	如同--hashlimit-srcmask参数，差别仅在判断对象是目的端IP的网段
--hashlimit-name	用来设置hashlimit数据库的名称
--hashlimithtable-size	用来设置一个数据库内最多可记录多少条数据，不过请注意，我们必须把--hashlimithtable-size所设置的值乘8才是数据库记录的最大值。例如所需要的最大值为10000时，--hashlimithtable-size=(10000/8)=1250，只要直接设置--hashlimithtable-size=1250即可
--hashlimithtable-expire	用来设置数据库中某一条数据过期的时间，而其时间的计算方式为“从最后一个符合本规则的数据包进入系统的时间开始累加”，当累加后的时间大于或等于--hashlimithtable-expire的时间时，这条数据就算过期，其时间的单位是毫秒
--hashlimithtable-gcinterval	设置数据库中的数据过期之后的多久，自动将这条数据从数据库中删除

24. 多功能匹配模块u32

在此之前，我们介绍了很多netfilter的匹配方法，例如-p、-s、-d、ttl、length等，其实在netfilter下有个新增的扩展模块称作u32。u32可以视为一个多功能匹配模块，允许我们检查数据包内各包头的任意数据，其功能几乎可以涵盖一半的其他匹配方法，因此u32模块的使用方法也就比其他模块要复杂很多。下面列举几个示例来说明u32的用法。

示例3.9 使用u32来匹配数据包是否为TCP包

如何判断一个数据包所使用的传输协议？其关键就在于IP包头中的Protocol字段内的值，我们可以在/etc/protocol这个文件中找到传输协议所对应的ID，例如TCP为6，UDP为7，ICMP为1等。因此，若要判断数据包内所有的通信协议是否为TCP，我们只要找出IP包头内Protocol字段内的值来匹配是否为6即可，u32的语法如下。

```
iptables -A INPUT -m u32 --u32 '6&0xFF=6' -j DROP
```

这语法看来有点复杂，但事实上并非如此，我们可以从图3-25 IP包头结构来开始看，图中最上面的数字列是用来表示位，也就是说，图中的每一行都有4个字节。在u32模块的描述下，第一列第一行的起始字节称为第零个字节，而u32模块每次固定抓取4个字节来进行匹配。因此，如果要取Protocol字段的内容来匹配，就得从第6个字节开始取得数据，但是我们所需要数据其实就只有第4个字节而已，所以接下来必须把多余的内容清除掉，清除的方法是把抓取到的4个字节与0x000000FF进行与运算(AND)，如此就可以只得到Protocol字段的值了。在本示例中的6&0xFF所代表的即是从第4个字节开始抓取数，然后再与0xFF进行与运算，最后的值如果等于6即条件成立。



图3-25 IP包头的结构

示例3.10 使用u32来匹配数据包是否为IPV4协议的数据包

在这个例子中，我们一样从图3-25 IP包头结构来开始看。判断的依据是使用IP包头的最前面4位进行匹配。从第零个字节开始抓取数据，不过，这次我们需要的数据并不是在最后一个字节，也不是完整的第一个字节，而是第零个字节中的前4个位，因此必须把抓取到的数据与0xF0000000进行与运算，然后再把前4个位右移28位，如此就可以轻松地进行匹配操作。

语法中的>>符号即是右移符号，而其后的28就是移位距离；另外，我们也可以使用<<符号来代表左移。

```
iptables -A INPUT -m u32 --u32 '0&0xF0000000>>28=4' -j DROP
```

示例3.11 使用u32来匹配数据包内的来源端IP为10.10.15.29

本例使用IP包头中的Source Address字段来进行判断，因此从第12个字节开始抓取数据。因为Source Address字段刚好是4个字节，所以不需要进行与的运算操作；接着把得到的值直接与10.10.15.29进行匹配，不过IP的部分必须转成16进制。

```
iptables -A INPUT -m u32 --u32 '12=0x0A0A0F1D' -j DROP
```

示例3.12 使用u32来匹配数据包内的来源端IP为10.10.15.29、通信协议为icmp以及TTL为128

下例应该无需再解释了，比较需要注意的是&&符号代表and的意思，如果有多个条件要匹配时就可以使用&&符号；匹配的值也可以是一个范围，例如，5&0xFF=0:64是指TTL值介于0~64之间的值。

```
iptables -A INPUT -m u32 --u32 '12=0x0A0A0F1D && \
6&0xFF=1 && 5&0xFF= '128 -j DROP
```

在分析了几个示例后，相信聪明的读者们应该都已了解如何使用u32模块来匹配IP包头的内容了。但如果我们要匹配的内容是IP包头以外的数据(例如TCP包头)，那可就有小麻烦了，因为IP包头的长度并非固定不变。以下示例将说明如何使用u32模块来匹配IP包头以外的数据。

示例3.13 使用u32来匹配数据包内TCP包头的目的端端口为80

首先使用6&0xFF=6来确认这是一个TCP协议的数据包，接着我们要匹配的目标TCP包头的第2、3个字节(这里也是从第零个字节算起)。u32无法识别“包头”这种东西，u32模块下的所有匹配都由数据包的第零个字节开始算起，因此我们要匹配的目标应该是：

```
IP包头的长度 + 0&0xFFFF = 80
```

但“IP包头的长度”又该如何取得呢？答案是：

```
IHL(IP包头第0个字节的低4位) * 4 = IP包头的长度(bytes)
```

接着以0&0xF000000>>24来取得IP包头第0个字节的低4位，假设为00000101₍₂₎，接着通过00000101₍₂₎*4得到结果；但*4又要如何计算呢？其实这很容易的，先把4转换成二进制的数字即100₍₂₎，因为有两个0，我们只要把00000101₍₂₎左移两位即可，所以最简单的方式是把0&0xF000000>>24改写成0&0xF000000>>22即可。

结合以上的过程，我们可以写成：

```
0&0x0F000000>>22@0&0xFFFF = 80
```

其中@前面的参数是告诉u32模块在偏移几个字节之后，才是真正要匹配的数据起点，而0&0xFFFF=80则与前面匹配IP包头内容的意义相同，总结最后所得到的匹配方法如下：

```
iptables -A INPUT -m u32 --u32 '6&0xFF=6 && \
0&0x0F000000>>22@0&0xFFFF=80' -j DROP
```

示例3.14 使用u32来匹配ICMP type 8的数据包

如下规则所示，其中6&0xFF=1确认哪些为icmp数据包，而0&0x0F000000>>22@0&0xFFFF000000>>24=8则是匹配icmp包头内的第0个字节是否等于8，若两者相符则条件成立。图3-26、3-27和3-28分别列出了TCP、UDP和ICMP包头的结构。

```
iptables -A INPUT -m u32 --u32 '6&0xFF=1 && \
0&0x0F000000>>22@0&0xFFFF000000>>24=8' -j DROP
```

01234567				01234567				01234567				01234567			
Source Port								Destination Port							
Sequence Number															
Acknowledgment Number															
Offset		Reserved		Flags				Window							
Checksum								Urgent Pointer							
Options(optional)															

图3-26 TCP包头的结构

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

图3-27 UDP包头的结构

01234567	01234567	01234567	01234567
Type	Code	Checksum	
Other message specific information			

图3-28 ICMP包头的结构

3.2 处理方法

在介绍完“匹配方式”之后，接下来看当一个数据包符合我们的匹配原则之后，可以用什么办法来处理这个数据包。“处理方法(target)”与“匹配方式”同样可以分为内置及模块扩展的处理方法，分别说明如下。

3.2.1 内置的处理方法

前面曾提过，Netfilter是由iptables_filter.ko、iptables_nat.ko、iptables_mangle.ko及iptables_raw.ko四个模块所组成。这些模块本身除了内置了一些简单的匹配方式之外，也内置了一些简单的处理方法，如ACCEPT、DROP等，接下来分析这些内置的处理方法。

1. ACCEPT及DROP的处理方法

在Netfilter的数据包处理方法中，ACCEPT和DROP可说是最基本的方法，分别为“允许”及“不允许”数据包进入，接下来通过两个例子来说明ACCEPT及DROP的使用方法。

示例3.15 如果数据包是由192.168.1.0/24网段送来的，就“允许”这个数据包进入

```
iptables -A INPUT -p all -s 192.168.1.0/24 -j ACCEPT
```

示例3.16 如果数据包是由192.168.1.0/24网段送来的，就将其丢弃

```
iptables -A INPUT -p all -s 192.168.6.0/24 -j DROP
```

2. QUEUE的处理方法

QUEUE的功能是将符合条件的数据包转发给User Space的应用程序来处理，但目前能支持这种机制的应用程序相当罕见，这里根据图3-29来说明QUEUE模块是如何工作的。当一个数据包由eth0接口进入之后，接着由Netfilter来匹配这个数据包的特征，如果这个数据包不符合Netfilter的条件，那么，这个数据包可能就由eth1接口送离本机，但如果这个数据包符合用QUEUE方法来处理的条件，这个数据包就会被送入到User Space应用程序来处理，待User Space的应用程序处理完之后，再依次由eth1接口送离本机。

或许你会觉得这有什么用？其实这样的一个机制可以帮助我们做很多事，如User Space这个应用程序可以是个“杀毒软件”或“垃圾邮件过滤软件”。我们可以在Netfilter上指

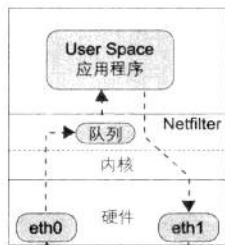


图3-29 Queue模块的运作原理

定,如果数据包要送往Mail服务器,那就将数据包转发到User Space上进行“病毒过滤”及“垃圾邮件的检测操作”,这就是一种应用场景。如果你想试试这个功能的话,有一套叫做shaperd的软件支持这样的机制,其功能是来执行简单的带宽管理操作,但如果真的需要一个完善的带宽管理机制,shaperd是无法满足企业需要的。关于带宽管理的内容,我们在第11章再来讨论。

接着以下面的示例来说明如何将符合条件的数据包转发给User Space应用程序。

```
iptables -A FORWARD -p tcp -d $MAIL_SERVER --dport 25 -j QUEUE
```

3. RETURN的处理方法

要讨论RETURN的处理方法,就得先讨论什么是用户定义的链(User-Define Chain),因为RETURN的处理方法是运用在用户定义的链之中的,因此,我们先来认识什么是用户定义的链。

何谓用户定义的链?

这里以filter机制为例进行说明。在filter机制中共有三个链,分别是INPUT、FORWARD及OUTPUT,不过,在这三个基本的链以外,我们还可以任意扩充新的链,而这些扩充出来的链就称之为用户定义的链。接下来分析用户定义的链的添加、删除及修改操作方式。

● 添加用户定义的链

添加用户定义的链的方法如图3-30所示,我们可以使用iptables -N WEB_SRV的命令添加一个用户定义的链,其中WEB_SRV就是我们这次要添加的链。

```
[root@localhost ~]# iptables -N WEB_SRV
[root@localhost ~]#
[root@localhost ~]# iptables -L
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

Chain WEB_SRV (0 references)
target prot opt source destination
[root@localhost ~]#
```

图3-30 添加用户定义的链

● 修改用户定义的链

有时候,我们可能因为某些规则上的改变,而需要调整防火墙的规则,当然这时用户定义的链的名称就有可能需要改变。可以如图3-31所示,使用iptables -E WEB_SRV MAIL_SRV命令来改变链名,其中WEB_SRV为原来名称,MAIL_SRV则为新名称。另外,我们在改变链的同时, iptables命令会帮我们整个filter表中与WEB_SRV这个用户定义的链相关联的信

息“一起改变”，因此，在改变用户定义的链名称时，我们无须担心其他关联性问题。

```
[root@localhost ~]# iptables -E WEB_SRV MAIL_SRV
[root@localhost ~]#
[root@localhost ~]# iptables -L
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

Chain MAIL_SRV (0 references)
target     prot opt source               destination
[root@localhost ~]#
```

图3-31 更改用户定义的链名称

● 删除用户定义的链

用户定义的链的删除方法如图3-32，我们可以使用`iptables -X MAIL_SRV`命令来删除设置的用户定义的链。但如果用户定义的链内有“规则”存在时，删除操作就会失效，在删除用户定义的链之前，请先清除用户定义的链内的规则。

```
[root@localhost ~]# iptables -X MAIL_SRV
[root@localhost ~]# iptables -L
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
[root@localhost ~]#
```

图3-32 删除用户定义的链

用户定义的链的工作方式

在使用用户定义的链之前，请你必须先要有个概念，就是“数据包经过Netfilter结构时，默认不会进入任何的用户定义的链之中”，如图3-33中的服务器上下达以下命令：

```
iptables -N ICMP
iptables -A ICMP -p icmp -j DROP
```

在这个命令下达完毕之后，客户端还是一样可以正常地ping到服务器主机。因为默认数据包并不会进入任何的用户定义的链，由客户端送进来的icmp包当然就不会被丢弃掉。我们必须让用户定义的链与INPUT链产生关联的关系，才能使INPUT类型的数据包可以被传输到

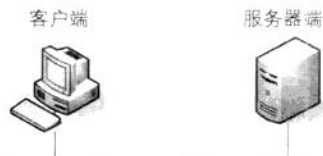


图3-33 用户定义的链的工作方式(一)

ICMP链之中，因此，使用以下命令让ICMP链与INPUT链关联在一起：

```
iptables -A INPUT -p icmp -j ICMP
```

这个命令是说，如果进来的数据包是属于icmp协议的数据包，我们就将之导入到ICMP链中。当以上命令下达完毕，客户端就无法ping到服务器了。再以图3-34为例来说明数据包在链里面经过的路径，首先数据包进入INPUT链内①，而该数据包刚好符合INPUT链的第二条规则，接着数据包被转发到ICMP链之中②，数据包在ICMP链中的匹配方式与一般链的匹配方式是一样的③。如果数据包不符合ICMP链的内容，则数据包会在ICMP链的最后一行匹配完毕之后，被重新送往INPUT链之内④，并且接着匹配INPUT链中的其他规则，以上就是用户定义的链的工作方式。

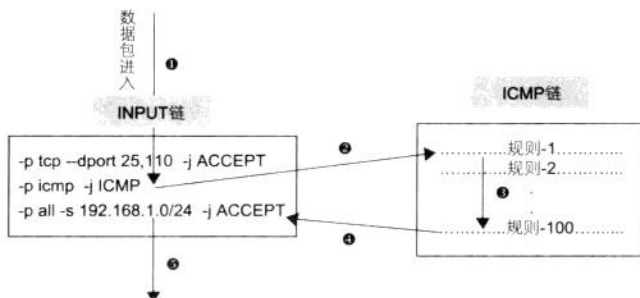


图3-34 用户定义的链的工作方式(二)

在了解用户定义的链的工作方式之后，接下来，我们来看看RETURN的处理方式，RETURN是用在用户定义的链之中的，其目的是让符合规则的数据包提前返回其原来的链。接着前面的例子，现在的状态客户端应该还是无法ping到服务器，接着试试在ICMP链之中，加入以下的规则看看会有什么效果：

```
iptables -I ICMP 1 -p all -j RETURN
```

以上规则会被插入到ICMP链的第一行，因此，任何被送入到ICMP链的数据包都会在这一行规则中，重新被送回INPUT链之中，如此客户端又可以ping到服务器了。由以上的实验，我们应该可以清楚了解到RETURN的功能。

3.2.2 由模块扩展的处理方法

看完内置的处理方式之后，接下来要介绍的是通过模块扩展出来处理方法，而这些模块所保存的位置如下，若你有兴趣不妨自己参考看看。

- /lib/modules/kernel_version/kernel/net/netfilter
- /lib/modules/kernel_version/kernel/net/ipv4/netfilter
- /lib/modules/kernel_version/kernel/net/ipv6/netfilter

1. REJECT的处理方式

REJECT是由ipt_REJECT.ko模块所提供的功能，REJECT与DROP有点相似。两者的差异在于DROP仅会将数据包丢弃掉，这将使得发送端误以为在网络上传输时丢失了，因此发送端将会重复地发送数据包直到超时为止；但REJECT就不同了，虽然REJECT也会丢弃掉发送端所送过来的数据包，不过REJECT会回送一个icmp包给发送端，由此告诉网络或者服务发生问题，当发送端收到这个icmp包之后，就会终止服务请求的操作。

接下来实际采用DROP方式来处理数据包，并且结合Wireshark软件实际截取网络上传输的数据包，如此我们就可以更清楚确认DROP的处理方法。在图3-35中可以看到192.168.1.189主机不断地重复发送发送请求包(SYN)，不过，这些数据包都被防火墙被丢弃掉了，192.168.1.189主机等不到192.168.1.194主机的应答(SYN,ACK)，而不断重复发送服务请求包(SYN)，一直等到超时为止，客户端(192.168.1.189)才会放弃这个服务请求的动作。

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.189	Broadcast	ARP	who has 192.168.1.194
2	0.000229	192.168.1.194	192.168.1.189	ARP	192.168.1.194 is at 0
3	0.000276	192.168.1.189	192.168.1.194	TCP	32772 > smtp [SYN] Seq
4	2.741059	192.168.1.189	192.168.1.194	TCP	32772 > smtp [SYN] Seq
5	8.158087	192.168.1.189	192.168.1.194	TCP	32772 > smtp [SYN] Seq
6	19.158541	192.168.1.189	192.168.1.194	TCP	32772 > smtp [SYN] Seq
7	41.141121	192.168.1.189	192.168.1.194	TCP	32772 > smtp [SYN] Seq
8	45.766348	192.168.1.189	192.168.1.194	ARP	who has 192.168.1.194
9	45.766653	192.168.1.194	192.168.1.189	ARP	192.168.1.194 is at 0
10	84.206891	192.168.1.189	192.168.1.194	TCP	32772 > smtp [SYN] Seq
11	88.790014	192.168.1.189	192.168.1.194	ARP	who has 192.168.1.194
12	88.790305	192.168.1.194	192.168.1.189	ARP	192.168.1.194 is at 0

图3-35 DROP对数据包的影响

不过，从图3-36我们可以清楚看到，当客户端(192.168.1.189)对服务器端(192.168.1.194)发送服务请求包(SYN)时，这个服务请求包是被REJECT给丢弃掉的，接着REJECT就会对客户端回送一个icmp数据包；我们可以从图中清楚看到，这个icmp数据包的错误通知为Destination unreachable，当客户端收到这个数据包之后，就会立即中断服务请求的操作。

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.189	Broadcast	ARP	who has 192.168.1.194
2	0.000223	192.168.1.194	192.168.1.189	ARP	192.168.1.194 is at 00
3	0.000262	192.168.1.189	192.168.1.194	TCP	32771 > smtp [SYN] Seq
4	0.000669	192.168.1.194	192.168.1.189	ICMP	Destination unreachable

图3-36 REJECT对数据包的影响

看完以上两者的差异之后，往后当你使用DROP及REJECT时，请把握以下原则：

● DROP:

凡是对付来自因特网的数据包一律使用DROP来处理, 因为我们无需浪费系统资源来回复因特网上的攻击者, 而DROP还可以拖延攻击者的攻击操作, 因为攻击者可能需要等待一段超时的时间, 不过, 也并非一定是如此, 还是得看攻击的方式而定。

● REJECT:

以网关式防火墙而言, 如果要限制企业内的用户连接到因特网上, 最好是使用REJECT的处理方法, 因为REJECT将可以让企业内的使用者快速地得知这个连接是不允许的, 而不需要去等待一段超时的时间。

另外, 关于REJECT所应答的icmp数据包, 是可以由使用者自己来定义其错误信息的, 方法如以下命令所示:

```
iptables -A INPUT -p tcp --dport 25 -j REJECT \
--reject-with icmp-net-unreachable
```

我们只需使用--reject-with <icmp-error-messages>即可达到伪装的目的, 可以使用的错误信息分别如下, 至于这些错误信息所代表的含义为何? 请你自己参阅RFC-792。

- icmp-net-unreachable
- icmp-host-unreachable
- icmp-port-unreachable
- icmp-proto-unreachable
- icmp-net-prohibited
- icmp-host-prohibited

2. LOG的处理方法

LOG的处理方法是由ipt_LOG.ko模块所提供的功能。在开始介绍LOG的处理方法之前, 请要先了解一件事, Netfilter默认并不会生成任何日志, 如果需要日志记录, 就得使用LOG这个模块了。

举个例子来说明LOG的用法。假设我们要记录所有曾经对本机有提出SSH服务请求的客户端IP, 可以使用以下命令来完成日志的记录操作。

```
iptables -A INPUT -p tcp --dport 22 -j LOG
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

不过, 以上这个命令看起来或许会觉得怪怪的, 如果根据first match原则来看, 当数据包符合第一条规则时不就应该结束INPUT链的匹配操作吗? 那么, 第二条规则是做什么用的? 其

实是这样的，LOG是一个比较特殊的处理方法，因为LOG只会记录数据包的信息，而不会真正处理这个数据包，因此，这个数据包继续匹配INPUT链中的其他规则，而LOG所记录下来的日志则会存放在/var/log/messages这个文件内，只不过在这里产生了几问题，如下：

- 日志量的增长速度惊人：

如果真的用以上规则来记录日志，那么日志量增长的速度将会非常迅速，因为LOG会记录每一个ssh协议数据包的信息。可以将第一条规则改写如下，因为我们只需要记录每一条连接的第一个数据包即可。

```
iptables -A INPUT -p tcp --syn --dport 22 -j LOG
```

- 如何从/var/log/messages文件中分离出Netfilter的日志：

由于/var/log/messages文件会记录整个系统的大多数日志，因此，我们要从messages这个文件中筛选出Netfilter的日志。但这并不是一件简单的工作，所幸LOG模块提供了一组相当好用的参数--log-level。使用这个参数之后，我们只需要经过简单的设置，就可以将Netfilter的日志独立保存在特定文件中，方法如下：

- Netfilter的规则语法：

为了将Netfilter的日志独立存放在特定的文件中，在撰写Netfilter规则时，必须额外加上特定的参数才行。因此将第一条规则改写如下：

```
iptables -A INPUT -p tcp --syn --dport 22 -j LOG --log-level alert
```

其中--log-level alert的意思是说，我们要将Netfilter所生成的日志设置为alert级别的日志。

- rsyslog服务的设置

请在rsyslog的配置文件/etc/rsyslog.conf内加上下一行，这一行的意思是说，如果日志是由kernel所生成的，而且某日志级别是alert级的日志，就将该日志保存在/var/log/netfilter这个文件中。更改完毕之后，别忘了重新启动rsyslog服务，以后关于SSH协议的日志就会被保存到/var/log/netfilter这个文件中了。

```
kern.=alert /var/log/netfilter
```

- 如何从Netfilter的日志之中分离出特定的日志：

虽然可以将Netfilter的日志独立保存在特定的文件中，但这样还不够完美，因为在/var/log/netfilter这个文件中可能会存放着大量Netfilter的日志。如果我们只需要特定的日志，又该如何处理呢？例如，我们只需要与SSH协议有关的日志时，该如何从成千上万行的日志之中，选出我们所需要的特定日志呢？其实这很容易做到，只需要结合使用--log-prefix参数即可，其用法如下：

```
iptables -A INPUT -p tcp --syn --dport 22 -j LOG --log-level alert \
--log-prefix " SSH-REQUEST "
```

以后当有日志被记录起来后, Netfilter会自动在这一条日志上加注“SSH_REQUEST”字符串。因此, 可以使用grep命令来筛选所需的日志, 使用方法如下:

```
grep ' SSH-REQUEST ' /var/log/netfilter
```

● LOG的其他参数:

除了--log-level及--log-prefix两个参数之后, 还有--log-tcp-sequence、--log-tcp-options、--log-ip-options和--log-uid四个参数, 这些参数的用途分别如下:

参数名称	功能说明
--log-tcp-sequence	记录TCP数据包的序号
--log-tcp-options	记录TCP包头Options字段的信息
--log-ip-options	记录IP包头Options字段的信息
--log-uid	记录数据包是由本机的哪一位用户所生成的

● 如何解读Netfilter所生成的日志:

如下所示即为Netfilter所生成的日志, 不过, 这些内容会因为参数使用的多少而有所差异。这里通过表3-13来解释下列日志中每一个字段代表的含义。

```
Jun  8 10:48:49 mail:  SSH-REQUIRE IN=eth0 OUT=  MAC=00:02:b3:0c:23:1b:00:0f:b0:93:b2
:7d:08:00 SRC=10.0.1.100 DST=10.0.1.101 LEN=48  TOS=0x00 PREC=0x00 TTL=128 ID=27846 DF
PROTO=TCP SPT=1366 DPT=22 WINDOW=65535 RES=0x00 SYN URG=0
```

表3-13 Netfilter Log字段解释

字段名称	说明
Jun 8 10:48:49	日志的生成时间
mail	生成这个日志的主机名称
SSH-REQUIRE	--log-prefix参数所设置的字符串
IN=eth0 OUT=	设置数据包的“进入”及“离开”接口
MAC	目的及来源端的MAC地址
SRC、DST	数据包的来源及目的端地址
LEN	IP包+承载数据的总长度(MTU)
TOS	IP包头内的Type Of Service值
PREC	Type Of Service字段内的Precedence位值
TTL	数据包内的TTL值
ID	IP包头内的Identification值
DF	Do not fragment值有被设置

字段名称	说明
PROTO	上层协议是什么
SPT、DPT	来源端及目的端的端口号
WINDOW	IP包头内的Window Size 值
RES	TCP-Flags中的ECN bits的值
SYN	TCP-Flags中SYN标记的值
URGP	TCP-Flags中URGENT标记的值

除表3-13所示之外，其实还有更多日志字段，只是在以上这个示例中刚好没有这个值可以记录，表3-14中列出了所有Netfilter日志字段。

表3-14 Netfilter完整日志信息

字段名称	用途说明
IN、OUT	数据包进出的接口名称
MAC	目的及来源端的MAC地址
Ethernet Frame Types	MAC字段的末两位，在以太网上标明Ethernet Frame上面所承载的是什么协议的数据包
SRC、DST	数据包的来源及目的端地址
LEN	IP包+承载数据的总长度(MTU)
TOS	IP包头内的Type Of Service的值
PREC	Type Of Service字段内的Precedence位值
TTL	数据包内的TTL值
ID	IP包头内的Identification值
DF、MF、FRAG	分别代表IP包头内以下三个字段 Do not fragment More fragments follow Fragment offset
OPT(0727-A200)	IP包头内的Options字段信息，只有加上--log-ip-options参数才会有这项信息
PROTO	上层协议是什么
SPT、DPT	来源端及目的端的端口号
SEQ、ACK	SEQ及ACK的序号，只有加上--log-tcp-sequence参数才会有这项信息
WINDOW	IP包头内的Window Size 值
RES	TCP-Flags中ECN bits的值
SYN、FIN、ACK、RST、URG、PSH	TCP-Flags的值
URGP	TCP-Flags中URGENT标记的值
OPT(020405-300)	TCP包头中Options字段信息，只有加上--log-tcp-options参数才会有这项信息

3. ULOG的处理方法

ULOG的处理方法是由ipt_ULOG.ko模块所提供的功能,且ULOG与LOG都是拿来记录Netfilter日志的,差别仅在于LOG是将日志交给系统的syslogd处理,而ULOG则是将日志交给特定的User Space机制来处理。目前发展比较完整的要算ulogd这个机制了。若你感兴趣,可以到<http://www.netfilter.org/projects/ulogd/downloads.html>下载最新版本。图3-37即为ulogd的工作原理,在Netfilter生成日志之后会将日志交给User Space程序来处理,而这个User Space程序就是ulogd;等到ulogd收到日志后,ulogd就会将收到的日志写入到数据库之中,这个数据库通常会结合使用MySQL或Postgresql。

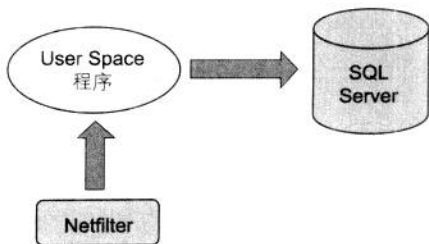


图3-37 ULOGD的工作原理

最后,可以使用一些支持SQL Server的日志分析工具来分析数据库中的日志,其中较为流行的日志分析工具为nulog;nulog是用php程序编写的,至于这个工具是否符合你的需求就不一定了,因此你有必要了解一下GNU/Linux下的编程语言,这样至少可以按照自己的需求来修改需要分析的日志。

至于ULOG的处理方式该如何运用,这牵扯到PHP以及MySQL的设置及组合,因此这里不打算介绍ULOG的使用方式。若有兴趣,不妨自己参考Netfilter的官方网站,其网址为<http://www.netfilter.org/projects/ulogd/index.html>。

4.TOS的处理方法

TOS的处理方法是由ipt_TOS.ko模块所提供的功能,其目的是修改路过mangle机制的数据包,TOS所能改变的对象为IP包头内的TOS(Type Of Service)值。前面的章节中曾提到过,TOS字段已在新版的RFC文件中被改成了DSCP(Differentiated Services Code Point),现在会用到TOS机制的机会几乎等于零,况且TOS的处理方式完全可以用DSCP的处理方式来取代,因此这里并不多想花时间介绍TOS的使用方法。

5. DSCP的处理方法

DSCP的处理方法是由ipt_DSCP.ko模块所提供的功能,其目的是修改路过mangle机制的数据包,而DSCP所能改变的对象为IP包头内的DSCP值。或许你会觉得奇怪,为何需要修改数据包IP包头内的DSCP值呢?其实确实有必要这么做;例如,在执行QOS(Quality of Service)的任务时,IP包头内的DSCP值的修改就是一个很关键的操作,下面让我们来看QOS的工作原理。

在设计QOS机制之前，我们必须明确划分出企业对于各项服务的需求程度，然后适当分配每项服务所需的带宽。以图3-38为例，我们把企业的对外带宽分成四块，分别是HTTP、SMTP、Voice及Other四项应用，并且希望HTTP协议的应用最多不得超过512K bps、SMTP的上限是384K bps、Voice的上限是256K bps，其他以外的应用总和不得超过384K bps。图3-39显示了QOS结构图。

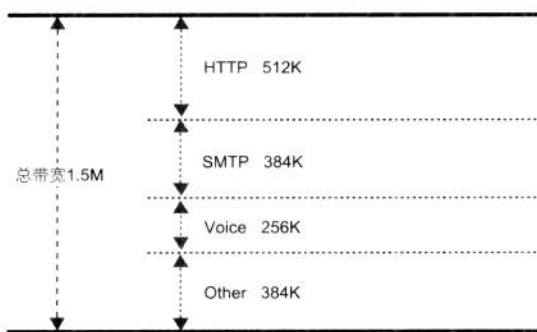


图3-38 QoS带宽分配图



图3-39 QOS的结构图

有了以上明确的规划之后，我们假设QOS机制要部署在网关位置上。因此，当有数据包进入到网关时，这个数据包会先被送到“数据包分类器”，再由数据包分类器来为不同的数据包进行分类的操作。而“分类”的依据有很多种方法，例如可以依据数据包IP包头内的DSCP值来分类，由于IP包头内的DSCP值默认设为0000-00共6个位，因此，总共可以有 $2^6=64$ 中变化。也就是说，如果用DSCP值来对数据包分类，最多可将数据包分成64种，执行方法如下：

```
iptables -t mangle -A FORWARD -p tcp --sport 80 -j DSCP --set-dscp 1
```

上面命令的意思是指，如果数据包的来源端口为80，就把这个数据包内的DSCP值改为1；接着数据包会进入带宽分配器，我们就可以在带宽分配器中，根据数据包的DSCP值来给予数据包不同的带宽，如数据包的DSCP值为1，就给予512K bps的带宽。以上就是QOS的工作原理。

6. MARK的处理方法

前面的章节介绍过DSCP的使用方法，也讨论了DSCP在QOS中所扮演的角色，不过，由于DSCP只有6个bits，因此，运用DSCP最多只能将数据包分成64种，如果是在中小型企业执行QOS机制，这64种分类应该是足够的；但若是在大型企业或是较为复杂的网络环境中执行QOS，这64种分类显然是不够用的，因此，势必要寻找其他更为多样化的数据包分类机制。

MARK应该就是我们要寻找的数据包分类方式，MARK是由xt_MARK.ko模块所提供的功能。MARK的处理方式可以让我们在特定的数据包上标记一个“记号”，而这个记号是由数字所构成的。但请特别注意，“记号”并没有真正的写入到数据包里面，也就是说，MARK的操作并不会修改数据包的内容，而是Linux内核使用一块内存来记录数据包与MARK值的对应关系，因此，当数据包离开本机之后，MARK值也就随之消失了。

由于Linux系统本身就内置了数种不同的“数据包调度”算法，如果再结合Netfilter的DSCP或是MARK模块，就可以让Linux变成一个企业级的QoS管理系统。此处仅讨论“数据包分类”的部分，至于“带宽分配”的任务则留到第11章来讨论。下面为MARK的使用方法，其意是将SMTP的数据包的MARK值标记为25。

```
iptables -t mangle -A FORWARD -p tcp --sport 25 -j MARK --set-mark 25
```

7. CONNMARK的处理方法

CONNMARK与MARK的功能类似，都是用来对数据包设置mark值；CONNMARK所设置的mark我们称为ctmark，MARK所设置的mark值我们称为nfmark，而两者之间的差异在于mark值的有效范围；nfmark的有效范围局限在单一一个数据包，ctmark则为一个完整的连接。我们通过以下两个示例来说明。

```
iptables -t mangle -A INPUT -p tcp --dport 80 -j MARK --set-mark 1
iptables -t filter -A INPUT -m mark --mark 1 -j DROP
```

```
iptables -t mangle -A INPUT -p tcp --dport 80 -j MARK --set-mark 1
iptables -t filter -A OUTPUT -m mark --mark 1 -j DROP
```

在第一个示例中，进入本机的数据包将会被丢弃掉，因为数据包进入本机后会先进入mangle表的INPUT链，然后才会进入filter表的INPUT链，如此数据包在mangle表中会被标记nfmark为1，接着该数据包进入filter表之后，我们使用mark的匹配方式来匹配数据包，只要数据包上的nfmark为1者，就会被丢弃掉。

在第二个示例中将不会有任何数据包被丢弃掉，原因在于虽然进入本机的数据包在mangle表中被标记nfmark为1，但这个数据包一旦被本机的进程接收之后，就相当于这个数据包已经消失了，如此nfmark值自然也就跟着消失了，而本机应答给来源端主机的数据包则为另一个新生成的数据包，其上当然就不会有任何mark值存在。

在以下两个示例中，数据包都被丢弃掉，因为CONNMARK的ctmark值有效范围为一完整的连接，因此不管是送入或送出的数据包，其上都会包含这个ctmark值。

```
iptables -t mangle -A INPUT -p tcp --dport 80 -j CONNMARK --set-mark 1
iptables -t filter -A INPUT -m connmark --mark 1 -j DROP
```

```
iptables -t mangle -A INPUT -p tcp --dport 80 -j CONNMARK --set-mark 1
iptables -t filter -A OUTPUT -m connmark --mark 1 -j DROP
```

CONNMARK除了用来设置ctmark之外，CONNMARK还有一个特异功能，就是用来复制nfmark。此处通过以下示例来说明，在数据包进入本机之后，我们以MARK将数据包nfmark标记为1，接着再以CONNMARK将nfmark存储起来，最后再把nfmark值复制到这条连接上，这样通过CONNMARK的帮助，使得nfmark也可以作用在连接的所有数据包之上。或许你会觉得很奇怪，为何不都使用CONNMARK就好了，何必大费周折复制来复制去的？这是因为Linux系统在某些情况下不支持ctmark，至少到目前为止是这样的，或许在未来的Linux版本中会完全支持ctmark，如此就可以不再使用nfmark了。

```
1. iptables -t mangle -A INPUT -p tcp --dport 80 -j MARK --set-mark 1
2. iptables -t mangle -A INPUT -j CONNMARK --save-mark
3. iptables -t mangle -A OUTPUT -j CONNMARK --restore-mark
4. iptables -t filter -A INPUT -m mark --mark 1 -j DROP
```

8. TTL的处理方法

TTL模块的功能在于修改“路过”防火墙上数据包内的TTL值，例如，一个专门感染Windows系统的木马程序，该如何找到网络上其他的Windows主机来进行感染呢？其实这个问题很容易解决，我们只需要观察对方主机送来数据包内的TTL值即可，如图3-40所示。

```
[root@localhost ~]# ping 168.95.192.1
PING 168.95.192.1 (168.95.192.1) 56(84) bytes of data:
64 bytes from 168.95.192.1: icmp_seq=1 ttl=247 time=31.6 ms
64 bytes from 168.95.192.1: icmp_seq=2 ttl=247 time=38.4 ms
64 bytes from 168.95.192.1: icmp_seq=3 ttl=247 time=30.6 ms
```

图3-40 ping命令的应用

图中我们可以看到ttl=247的字样，也就是说，主机168.95.192.1把数据包送到我们这边时数据包内的TTL值为247，不过这247绝对不会是该数据包内TTL的初始值，因为数据包在网络上传输时是通过路由器一个一个地往外传送，而数据包每跨越一个路由器时，其内的TTL值就会被减1。这个距离我们可以用tracert 168.95.192.1命令来查询，在笔者的主机测得的结果是8个路由器，因此，把247+8=255应该就是数据包内TTL的初始值。

为什么要讨论数据包内TTL的初始值呢？这是因为每种操作系统对数据包的初始值定义不同，例如Linux为64、Solaris为255而Windows则为128。因此，如果要判断远程某一个IP

上所运行的操作系统是什么,最简单的方式就是ping它一下再由TTL值来判断即可,如此凡是受到Netfilter/Iptables所保护的系统,就可以运用Netfilter的TTL模块把路过防火墙的数据包内的TTL改掉。例如,把Windows主机送出的数据包内的TTL值改为64、Linux主机送出数据包内的TTL值改为128,由此达到欺骗入侵者的效果。TTL模块的用法如下:

```
iptables -t mangle -A POSTROUTING -s $WEB_SRV -j TTL --ttl-set 64
iptables -t mangle -A POSTROUTING -s $MAIL_SRV -j TTL --ttl-set 128
```

以上示例假设Web服务器是Windows系统,而Mail服务器是Linux系统,因为我们通过TTL模块把Web服务器送出数据包内的TTL值改为64,Mail服务器送出数据包内的TTL值改为128,TTL模块的其他参数如下表所示。

参数名称	功能说明
--ttl-set	把数据包内的TTL值设置为特定值
--ttl-dec	把数据包内既有的TTL值减掉特定的值
--ttl-inc	把数据包内既有的TTL值加上特定的值

9. REDIRECT的处理方法

REDIRECT的处理方法是由ipt_REDIRECT.ko模块所提供,而REDIRECT是一种特殊的DNAT机制,通常会与代理服务器(代理服务器)结合使用,使其成为透明代理(Transparent Proxy)的结构。因为代理服务器的机制尚未介绍,因此将在讨论代理服务器章节中再来讨论REDIRECT的用法。

10. MASQUERADE的处理方法

MASQUERADE的处理方法是由ipt_MASQUERADE.ko模块所提供的功能,MASQUERADE是一种特殊的SNAT机制。第2.12.5一节的第1部分介绍过MASQUERADE的用途,若你已忘记,请再复习一下。

11. NETMAP的处理方法

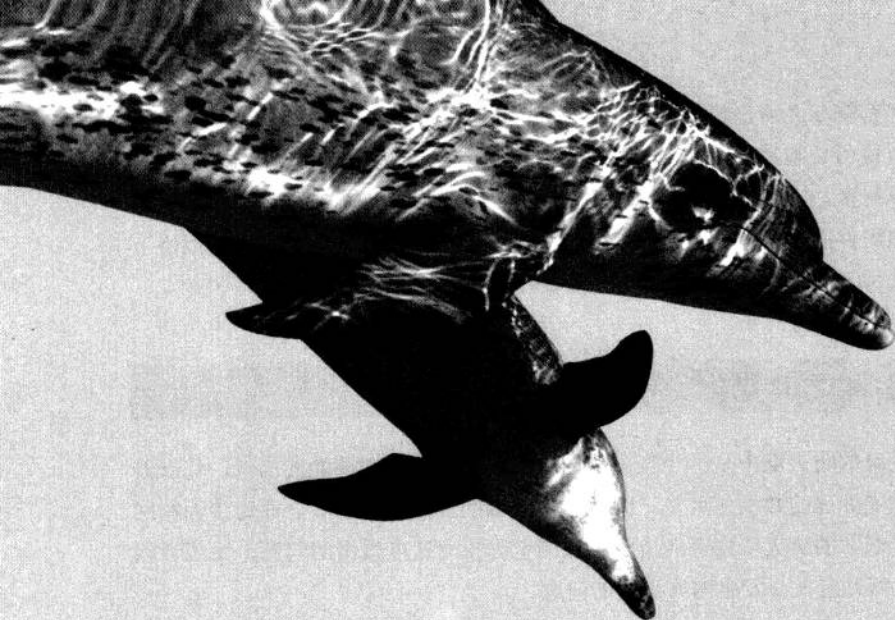
我们知道,一对一NAT是由一个SNAT及一个DNAT组合而成。如果需要将整个C类网络的公网IP映射到一个私有IP的网段,请问共需要写多少条规则?答案笔者并不想去计算,因为有了NETMAP这个处理办法之后,无论要映射的是C类、B类或是A类的IP网段,通通只需要两条规则就可以搞定了。如我们要将一个10.0.0.0/24的网段映射到192.168.1.0/24的网段,只需要下列两行规则即可。


```
iptables -t nat -A PREROUTING -i eth0 -d 10.0.0.0/24 \  
                                                -j NETMAP --to 192.168.1.0/24  
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.1.0/24 \  
                                                -j NETMAP --to 10.0.0.0/24
```

3.3 小结

Linux防火墙“功能”的数量及灵活性取决于Netfilter模块的数量，相信在你读完本章之后，一定更加能体会笔者所说的这句话。本章介绍了很多关于Netfilter模块的使用方法，同时也列举多种不同的使用环境，不过并不要求你将每个模块的参数都背下来，因为那是毫无意义的，请在阅读完本章节之后将其当做工具书来使用，需要时拿出来查询一下即可。





Linux

| 第4章 | Netfilter/Iptables的高级技巧

4

经过前面三章的辛勤耕耘之后,相信你已经掌握了整个Netfilter/Iptables结构及语法,但这都只是“使用”防火墙所必须具备的基本能力,而要构建一个企业级的网络防火墙所需要的知识可能比想象得多。本章将介绍一些较为深入且实用的防火墙处理技巧,如防火墙性能的最优化、防火墙硬件需求的评估、防火墙对于各种不同通信协议的处理技巧、网络上各种常见的攻击方式以及应对的策略等,了解这些后才能真正将Linux防火墙应用于网络。

4.1 防火墙性能的最优化

防火墙性能在企业级的防火墙中是一个很重要的主题,因为防火墙性能的高低,代表着防火墙在单位时间内所能处理的数据包数量,因此,防火墙的性能将关系着企业连接因特网速度的快慢,一个性能低下的防火墙将可能造成企业浏览因特网速度缓慢的问题,又或者因特网使用者在浏览企业网站时,会发生网络缓慢的问题。

“如何在相同的硬件平台上,让Linux防火墙的性能发挥到极致”,这就是本章所要讨论的重点。以往在课堂上讨论防火墙性能的问题时,大多数同学都会通过“更换更高级的硬件”来提升防火墙的性能,这也许是一种解决办法,但相对增加了企业的成本。其实我们只要稍微了解一些规则的编写技巧,就可以轻易地在相同的硬件平台上,将Linux防火墙的性能发挥到极致,并在不需要增加硬件成本的情况下,让Linux防火墙执行得更有效率。

不过,这些技巧其实都是出自于同一个原则:“尽量减少不必要的规则匹配”,因为每当数据包进入防火墙之后,就会在特定的链中逐一匹配每条规则,规则数量越多、数据包在防火墙内停留的时间就会越长,当然防火墙在单位时间内所能处理的数据包数量就减少,防火墙性能也就显得越低,因此,减少规则的匹配次数将是提升防火墙性能关键,但如何能减少不必要的规则匹配呢?请分析以下几种方式。

4.1.1 调整防火墙规则顺序

这里以Mail服务器上的单机防火墙为例,假设有一位使用者正准备使用POP3协议来接收电子邮件,我们再假设把这封电子邮件完整收下来,服务器端总共需要给客户端传输100个数据包,请问当信件全部传输完毕之后,这些数据包在防火墙上总共需要经过多少次的匹配操作?

```
1. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 22 -j ACCEPT
2. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 23 -j ACCEPT
3. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 25 -j ACCEPT
4. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 80 -j ACCEPT
5. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 110 -j ACCEPT
6. iptables -A INPUT -p all -m state --state ESTABLISHED,RELATED -j ACCEPT
```

粗略估算最少需要599次匹配，你可能觉得惊讶，为何需要如此多的匹配操作？因为TCP连接是一来一往的，也就是说，当服务器端送出一个数据包给客户端，客户端在收到数据包之后会应答一个确认包给服务器端，由此告诉服务器端：“你所发送的过来的数据包我收到了，请送下一个数据包”。服务器端发送几个数据包给客户端，客户端就会应答几个数据包给服务器端。了解这个原理后，我们再看一次示例中的规则，当客户端发送第1个数据包时，这个数据包即会在INPUT链中由上而下逐条匹配，直到第5条规则才会停止匹配操作，因此，第一个数据包需要经过5次匹配操作。不过当第1个数据包进入防火墙之后，其后的所有数据包都会属于ESTABLISHED状态的数据包，因此，第2个到第100个数据包都会需要6次匹配操作，当100个数据包全部传送完毕，防火墙共产生 $1*5+99*6=599$ 次匹配操作。

看完上一段的解释，不知你是否想到一个问题，如果我们将第6条规则移到第1条规则的位置，那么，匹配操作会变成几次呢？答案是105次！一下就省去了494次无效匹配操作，由此可证明，规则顺序对于防火墙性能有莫大的影响。

或许你会问：“那将原来的第5条规则移动到第2条规则的位置上，是不是又可以省去4次无效匹配操作？”这样的想法没有错，但不要忘了第2条规则下方还有其他4条规则，如果把原来的第5条规则移动到前方，那还不是会有其他规则变成第5条规则？因此，在编写规则时请把握一条原则，越热门(匹配命中次数越高者)的规则越靠前，如果每月平均HTTP的请求次数为100 000次，SMTP的服务请求次数为50 000次，POP3的服务请求次数为10 000次，SSH的服务请求次数为100次，TELNET的服务请求次数为10次，那就应该将上述示例改写成如下形式：

```
1. iptables -A INPUT -p all -m state --state ESTABLISHED,RELATED -j ACCEPT
2. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 80 -j ACCEPT
3. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 25 -j ACCEPT
4. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 110 -j ACCEPT
5. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 22 -j ACCEPT
6. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 23 -j ACCEPT
```

但在大多数情况下，我们很难衡量到底哪个服务比较热门，但你也不需要担心，其实Netfilter早就为我们设计了一个统计功能，如果真的无从判断规则的先后顺序，就请随便排列这些规则，等到防火墙执行一段时间之后再执行iptables -L -n -v命令，即可以看到每一条规则被匹配的次数，如图4-1所示，最后再根据这些数据重新调整顺序即可。

Chain INPUT (policy DROP 206K packets, 65M bytes)									
pkts	bytes	target	prot	opt	in	out	source	destination	
7962	876K	DROP	all	--	*	*	0.0.0.0/0	0.0.0.0/0	state INVALID
2170K	1803M	ACCEPT	all	--	*	*	0.0.0.0/0	0.0.0.0/0	state RELATED,ESTABLISHED
29840	1701K	ACCEPT	tcp	--	*	*	0.0.0.0/0	0.0.0.0/0	tcp flags:0x16/0x02 state
7010	456K	ACCEPT	udp	--	*	*	0.0.0.0/0	0.0.0.0/0	state NEW multiport dport
0	0	ACCEPT	tcp	--	*	*	192.168.2.0/24	0.0.0.0/0	tcp dpt:21
0	0	ACCEPT	tcp	--	*	*	192.168.3.0/24	0.0.0.0/0	tcp dpt:21

图4-1 Netfilter的统计信息

4.1.2 巧妙使用multiport及iprange模块

multiport及iprange是之前曾介绍过的模块，如果可以巧妙使用一下这两个模块，也可以达到精简防火墙规则的目的。

1. 巧妙使用multiport模块

运用multiport模块改写以上示例，如下，这样以后不管哪个服务器所需的匹配次数都会是一样的。

```
1. iptables -A INPUT -p all -m state --state ESTABLISHED,RELATED -j ACCEPT
2. iptables -A INPUT -p tcp --syn -m state --state NEW \
    -m multiport --dports 22,23,25,80,110 -j ACCEPT
```

2. 巧妙使用iprange模块

如果在防护墙的规则中需要开放一段IP区间，如192.168.1.12到192.168.1.16四个IP，以往我们可能需要四条规则来处理，有了iprange模块后，就可以将四条规则精简成一条规则，如下所示：

```
iptables -A INPUT -m iprange --src-range 192.168.1.12-192.168.1.16 -j ACCEPT
```

4.1.3 巧妙使用用户定义的链

以图4-2为例，假设防火墙后方共有三台主机，并在防火墙上分别为这三台主机各设置100条规则，也就是说在防火墙上总共会有300条规则，如果我们没有特别去注意规则的编写方式，那么，FORWARD链中将有高达300条规则。如果我们要匹配最后一条规则，数据包就必须先经过299次的无效匹配后，在第300次的匹配中才会被匹配出来，因此，这将会是一个非常低效的防火墙。

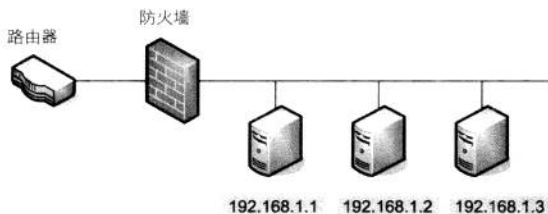


图4-2 用户定义的链示例

为了有效减少“无效匹配”次数，我们还有一个法宝可以使用，那就是在第3.2.1一节中所介绍过的用户定义的链。以图4-3为例，在多达300条规则的防火墙上使用用户定义的链是一个很好的主意，不过，在使用用户定义的链之前，我们必须先想好如何分类这些规则才能

达到最精简的效果。比如，可以根据“上层协议”来区分，也可以根据“应用层协议”来区分，也可以根据IP来区分，总之要用什么方式来区分并没有一定的标准，也没有永远正确的答案，只要能够让整个防火墙的无效匹配次数降到最低，那就是我们所需要的。

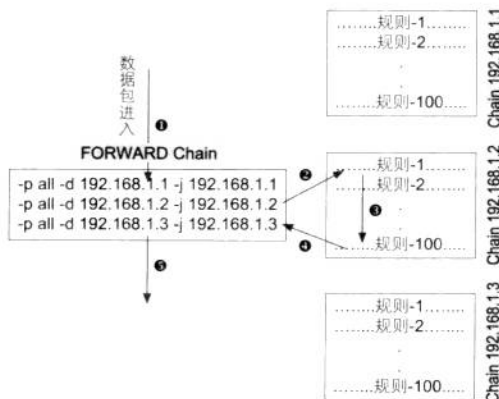


图4-3 巧妙使用用户定义的链

假设我们以IP为划分的依据，可以建立三个用户定义的链，分别为192.168.1.1、192.168.1.2、192.168.1.3三个链，当然，这些链的名称要如何设置由你来决定，不见得使用IP作为链名。最后再将300条规则分别放入到这三个链中，这样以后不管要匹配哪个链的最后一条规则，都可以把无效匹配次数降到102次左右；当然，如果我再将用户定义的链细分为192.168.1.1_ICMP、192.168.1.1_TCP、192.168.1.1_UDP、……，那么可以进一步减少无效匹配次数。到底该如何区分？还是那句老话：“没有一定的标准，也没有永远正确的答案，只要能够让防火墙整体的无效匹配次数降到最低，那就是我们所需要的。”

4.2 Netfilter连接处理能力与内存消耗

第3.1.2一节中介绍过nf_conntrack模块以及nf_conntrack模块的数据库/proc/net/nf_conntrack，nf_conntrack模块除了在filter机制下会用到外，在NAT机制下也是必须的，因为不管是filter还是NAT机制都会使用到连接跟踪的技术。不知你是否还记得nf_conntrack模块数据库的内容？这里再次将nf_conntrack模块的数据库内容列出：

```
tcp 6 426421 ESTABLISHED src=10.0.1.101 dst=10.0.1.102 sport=2502
dport=22 packets=291 bytes=21611 src=10.0.1.102 dst=10.0.1.101 sport=22
dport=2502 packets=264 bytes=33572 [ASSURED] mark=0 secmark=0 use=1
```

第3章已介绍过以上这些内容,在此不再赘述,不过,先问你一个问题,以上这些数据保存在何处?如果你十分熟悉GNU/Linux系统,这应该不是什么难事,所有/proc目录下的文件都存放在内存中,因此,filter或NAT机制在跟踪连接时所产生的跟踪信息就存放在内存中。换句话说,Netfilter每跟踪一条连接,就必然消耗一块内存。

4.2.1 计算最大连接数

/proc/sys/net/netfilter/nf_conntrack_max文件限制着nf_conntrack模块所能跟踪的最大连接数量,我们可以使用cat命令来查看文件中的默认值,而默认值会随着硬件的内存大小而改变。官方网站上提供一个简单的计算方式,即 $\text{CONNTRACK_MAX} = \text{RAMSIZE}(\text{in bytes}) / 16384 / (x/32)$,其中x是操作系统的地址位数,如果操作系统是32位的,x请填写32,若是64位的,x请填写64,因此,如果你的操作系统为32位并拥有512M的内存,nf_conntrack_max中的默认值为 $512 * 1024 * 1024 / 16384 / 1 = 32768$ 。

由上可知,防火墙的最大连接跟踪数量是有限制的,因此,企业所需要的最大连接数量若超过防火墙的最大连接跟踪数量,超出的部分无法被防火墙所接受,也就无法通过防火墙连接到因特网上,此时我们在/var/log/messages文件中可看到错误消息“nf_conntrack: table full, dropping packet”,不过就一般企业而言,在正常情况下,连接数量32 768应该是足够用的。

4.2.2 调整连接跟踪数

如果默认连接跟踪数量不足时该如何处理呢?我们先来复习一下第2.4.1节所介绍的内容,Linux系统核心本身大约为4MB左右(以RHEL6,Kernel Version 2.6.32.x为例),在这有限的空间大小中势必无法提供太多功能,而Linux系统的解决方法是允许我们帮其额外编写一段代码,然后再把这段代码加入到Linux中执行,由此来扩展Linux本身的功能。而这段代码我们可以令其以单一文件的形式来存在,必要时可以使用modprobe工具将其载入及移除,这种形式我们称之为“动态模块”,另一种方式则是把这段小程序直接结合到Linux本身,如此将Linux载入系统时,这段小程序也就随之被载入计算机系统中,这种形式我们称为“静态模块”。

接下来看一下,如果默认连接跟踪数量不足时该如何解决?首先要弄清在Linux系统中,nf_conntrack功能是以动态模块或是以静态模块的形式存在,我们可以使用下面两种方式来看。

● 查看Kernel Source配置文件

```
[root@localhost ~]# grep -l 'CONFIG_NF_CONNTRACK=' /boot/config-kernel-version
CONFIG_NF_CONNTRACK=y
[root@localhost ~]#
```

我们可以使用grep工具去搜索当初编译这个内核时所使用的配置文件，如此就可以了解nf_conntrack是以何种形式存在，如果搜索到字符串为CONFIG_NF_CONNTRACK=y，就是静态模块；如果CONFIG_NF_CONNTRACK=m，就是动态模块。

● 查看内核模块列表

在硬盘中，每个不同版本的内核都有自己专属的模块清单列表，而这份清单在/lib/modules/Kernel_Version/modules.dep文件中，可以使用grep工具去查找nf_conntrack.ko字符串，如果找到，nf_conntrack就是以动态模块存在，找不到则证明是静态模块。

在查看完nf_conntrack的形式后，再计算要传输的参数值，这个值的计算方式如下：假设需要跟踪的连接数量为262 144，参数值=262144/8=32768，计算出参数值后，可以把这个值传输给nf_conntrack模块，如果nf_conntrack模块为动态模块，请先手动执行以下命令；如果nf_conntrack模块已经载入系统，则请你先将之移除或重新启动系统：

```
modprobe nf_conntrack hashsize=32768
```

如果准备长期使用这个值，建议将上述命令直接加入到firewall.sh文件的前半部分(在尚未使用state或ctstate模块之前)，如此在执行firewall.sh文件时，连接跟踪数量就会自动设为262 144。

如果nf_conntrack模块是以静态模块的形式存在，那么，请将nf_conntrack hashsize=32768参数加入到/boot/grub/grub.conf的核心参数区，如下：

```
default=0
timeout=10
splashimage=(hd0,0)/grub/splash.xpm.gz
title Red Hat Enterprise Linux 6 (Kernel-Version)
    root (hd0,0)
    kernel /vmlinuz- Kernel-Version ro root=/dev/mapper/VolGroup-lv_root
    rd_LVM_LV=VolGroup/lv_root rd_NO_LUKS rd_NO_MD rd_NO_DM LANG=zh.UTF-8
    KEYBOARDTYPE=pc KEYTABLE=us nf_conntrack.hashsize=32768
    initrd /initramfs-2.6.32-19.el6.x86_64.img
```

最后再重新启动系统，开机后，最大连接跟踪数量就会自动设置为262 144。

4.2.3 连接跟踪数量与内存消耗

从前面的介绍可以了解到，连接跟踪的数据是存放在内存中的，因此，跟踪的连接数量越多，内存需求量就越大。假设我们所需要的连接跟踪数量为262144，到底会消耗多少内存空间呢？如果只是单纯的连接跟踪，每跟踪一条连接会消耗276字节的空间，但如果跟踪的连接是复杂连接时，可能会需要额外加载其他模块(例如nf_conntrack_ftp)，此时所需的内存空间就会

大一点。另外需要注意的是不同版本的nf_conntrack模块所需的内存空间未必相同,因此,按照平均每条连接所需内存空间为512字节的空间来计算 $262144 \times 512 / 1024 / 1024 = 128\text{M}$ 字节,从这个结果来看其实nf_conntrack所需的内存空间并不算太大。



提示

什么是复杂的连线呢?你可以先行阅读第4.4一节。

4.3 使用raw表

第2.5一节中曾提过raw表,不过截至目前为止,还未介绍过关于raw表的相关知识,本节将完整介绍raw表。我们在上一节了解到,只要系统有载入nf_conntrack模块,任何穿过防火墙的连接都会被记录在/proc/net/nf_conntrack文件中,但有时并不需要记录每条连接的信息,以图4-4为例,请问企业内部的客户端在访问DMZ中的Mail服务器时,防火墙一定需要跟踪这些连接吗?其实未必如此,这里将防火墙规则的片段编写为:

```
iptables -A FORWARD -i eth1 -o eth2 -m state --state NEW -j DROP
iptables -A FORWARD -i eth2 -o eth1 -m state --state NEW -j ACCEPT
```

但问题来了,因为nf_conntrack模块默认会自动跟踪所有连接,那我们该如何让nf_conntrack模块不去跟踪某条连接呢?这时候就得靠raw表了。例如,我们不想让nf_conntrack模块去跟踪-i eth2 -o eth1这个方向的SMTP协议时,可在raw表加入以下规则,即可不被nf_conntrack跟踪,语法如下:

```
iptables -t raw -A PREROUTING -i eth2 -o eth1 -p tcp --dport 25 -j NOTRACK
```

不过,你可别忘了编写另一个方向的规则,语法如下:

```
iptables -t raw -A PREROUTING -i eth1 -o eth2 -p tcp --sport 25 -j NOTRACK
```

完成以上两个规则后,即使有-i eth2 -o eth1的SMTP协议产生,nf_conntrack模块也不会去跟踪这些连接,因此,在/proc/net/nf_conntrack数据库中自然也就不会有任何的连接记录存在了。

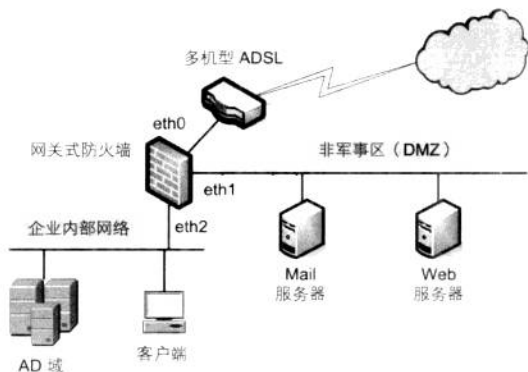


图4-4 网关式防火墙

在了解raw表的功能后，你是否发现了raw表所带来的好处？其实raw Table的好处在于“加速”，以及增加可跟踪的连接数量，因为被raw表所定义的连接不会被跟踪，也就不会算作连接跟踪数量。此外，raw表所定义的连接会直接跳过nat Table及nf_conntrack模块的处理，因此，可以加快数据包进出防火墙的速度，但因为raw Table所定义的连接会跳过nat表，所以任何被raw表定义的连接都无法被NAT机制所处理，因此，使用raw Table时请特别注意这个问题。

最后要提醒的一点是，raw表只有两个链，分别为PREROUTING和OUTPUT，其用途如下。

● PREROUTING链

如果是网关式防火墙，PREROUTING链可以用来处理防火墙两侧网络之间所建立的连接(因为这些数据包都会经过PREROUTING链)，另外，PREROUTING链也可以处理任何主动连接到防火墙本机的连接。

● OUTPUT链

例如nat Table的OUTPUT链，因为PREROUTING链只能处理防火墙两侧网络间相互的连接以及任何主动连接到防火墙的连接，但PREROUTING却无法处理任何由防火墙本机所对外建立的连接，因此，raw表与nat表同样特别设计了一个OUTPUT链，用来处理本机对外建立的连接。

4.4 简单及复杂通信协议的处理

不同的应用需求会衍生出不同的通信协议，而不同的通信协议具有不同的特性。不过，就Netfilter/Iptables来看，所有通信协议只会被分为两种，其一是“简单的通信协议”，其二是“复杂的通信协议”，而这两者之间的分类依据又是什么呢？此外，这两类通信协议在防火墙规则的设计上，我们又应该特别注意什么呢？本节将进行完整的说明。

4.4.1 简单通信协议

何谓简单通信协议？如果客户端访问服务器端时只使用“一条连接”的协议，我们就称为简单通信协议。在图4-5 HTTP协议的应用中，客户端使用端口50000连接服务器端的端口80，当连接建立后，不管是客户端要传输数据给服务器端，或者服务器端传输数据给客户端，都是使用这一条连接来传输数据，因此，HTTP协议就是属于简单通信协议。还有哪些通信协议属于简单通信协议呢？其实在常用的通信协议大多属于简单通信协议，如SSH、TELNET、SMTP、POP3、IMAP及HTTPS等。

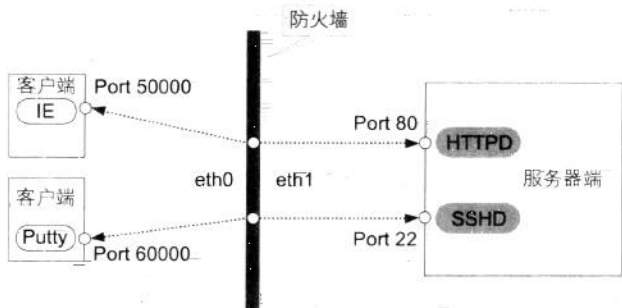


图4-5 HTTP通信协议

1. 简单通信协议穿过防火墙的处理原则

防火墙处理简单通信协议是很容易的，如图4-5所示，假设在客户端与服务器端之间有一台网关式防火墙，我们只需在防火墙上设置如下规则，即可让客户端正常访问服务器端的服务，也就是说，简单通信协议只需要考虑到连接的“来源端口”及“目的端口”，并且确认连接的双向皆可正常通过防火墙即可，因此，几乎所有防火墙在处理简单通信协议时都不会遇到问题。

```
iptables -F FORWARD DROP
iptables -A FORWARD -i eth0 -o eth1 -p tcp -d $MAIL --dport 25 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -p tcp -d $MAIL --dport 110 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -p tcp -d $WEB --dport 80 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p tcp -s $MAIL --sport 25 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p tcp -s $MAIL --sport 110 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p tcp -s $WEB --sport 80 -j ACCEPT
```

2. 简单通信协议穿过NAT的处理原则

由于简单通信协议只使用一条连接，而NAT只是单纯地转发来源及目的端的IP，因此，简单通信协议在一对多、多对多、一对一及NAPT的环境中都可以正常使用，并没有什么需要特别注意之处。

4.4.2 复杂通信协议

了解简单通信协议后，接下来介绍复杂通信协议，复杂通信协议的应用较少，如FTP、PPTP、H.323及SIP等。什么是复杂通信协议呢？简单来讲，就是客户端与服务器端之间，需要多条连接才能完成应用的协议，就属于复杂通信协议。

这里以较常用的FTP协议为例来说明FTP协议的工作原理，首先要说明的是，FTP通信协议有两种工作模式，即被动模式和主动模式，以下针对这两种不同工作模式来说明。

● 被动模式

以图4-6为例，我们假设客户端的IP为192.168.2.10，服务器端的IP为192.168.2.11，首先客户端使用端口1955来连接服务器端的端口21，接着客户端对服务器端送出一个含有PASV命令的数据包给服务器端，服务器端在收到这个数据包之后就会知道，客户端要求以被动(Passive)模式来传输数据，接着服务器端会发送一个含有PORT 192,168,2,11,114,134字符串的数据包给客户端。

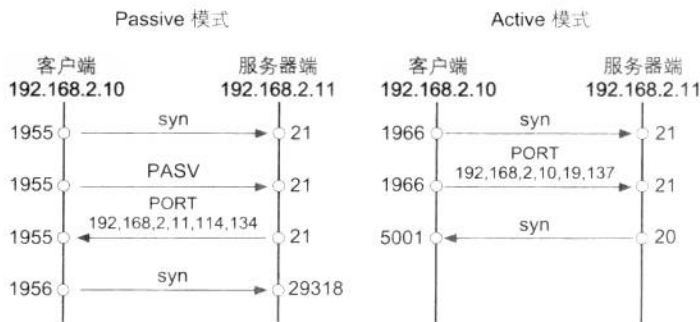


图4-6 FTP通信协议

客户端在收到这个数据包后，就可以从PORT 192,168,2,11,114,134字符串中了解到，服务器端192.168.2.11(即字符串中的前四段)在其上动态打开了一个端口29318(即字符串中后面两段 $114*256+134=29318$)，接着，客户端会使用另一个端口1956去连接服务器端的端口29318。

从以上流程可以发现，FTP协议在被动模式中建立了两条连接，分别是“端口1955-端口21”以及“端口1956-端口29318”，其中端口1955、端口1956及端口29318都以随机方式决定。另外，“端口1955-端口21连接”是客户端传输“命令”给服务器端所用的通道，而“端口1956-端口29318”连接则是服务器端传输“数据”给客户所用的通道。

● 主动模式

以图4-6为例，我们假设客户端的IP为192.168.2.10，服务器端的IP为192.168.2.11，首先客户端使用端口1966连接服务器端的端口21，接着客户端会发送一个含有PORT 192,168,2,10,19,137字符的数据包给服务器端。

服务器端在收到这个数据包之后，就可以从PORT 192, 168,2,10,19,137字符串中了解到，客户端192.168.2.10(即字符串中的前四段)在其上动态启用一个端口5001(即字符串中的后面两段 $19*256+137=5001$)，接着服务器端会固定使用端口20去连接客户端的端口5001。

从以上流程可以发现，FTP协议在主动模式中也会建立两条连接，分别是“端口1966-端口21”以及“端口20-端口5001”，其中端口1966及5001按随机方式决定。另外，“端口1966-端口21”连接时客户端发送“命令”给服务器端所使用的通道，而“端口20-端口

5001”连接则是服务器端传输“数据”给客户端用的通道。

经过以上的解释,相信你对FTP协议应该会有更深层的了解。接下来讨论filter及NAT机制在处理复杂通信协议时遇到的问题以及解决办法。

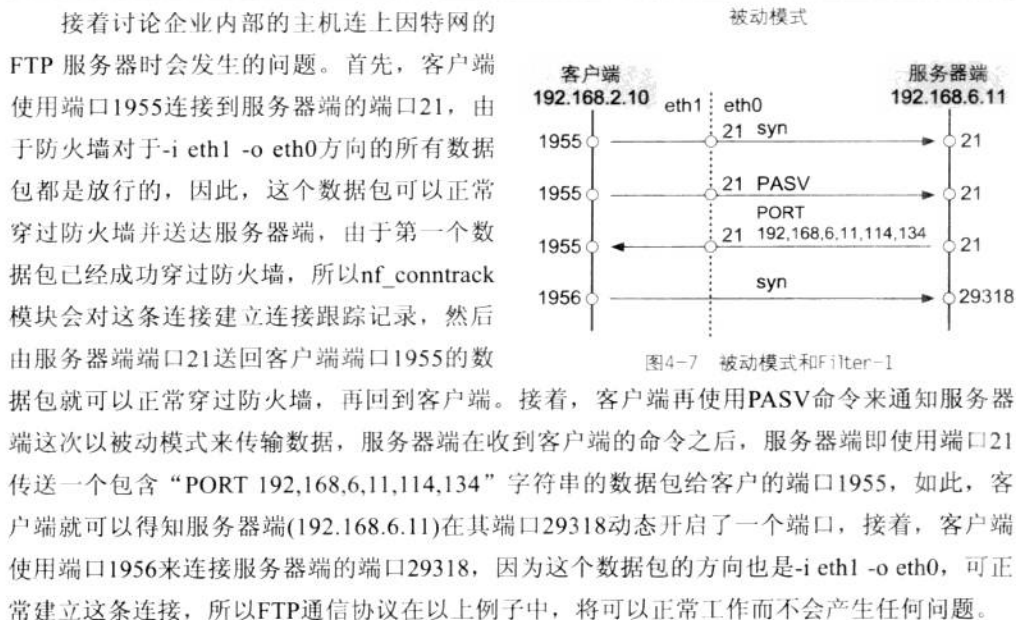
1. filter机制处理复杂通信协议时会遇到的问题及解决办法

由于复杂通信协议众多,而且每一种协议的处理方式都不同,这里使用较常见的FTP协议来说明Netfilter处理FTP通信协议的方式,只要你学会FTP复杂通信协议的处理方式,对其他复杂通信协议的处理大概就不会有什么问题了,因为netfilter机制在处理复杂通信协议时“语法规则”基本相同。这里使用FTP协议中的被动模式来说明当防火墙放在客户端以及放在服务器端时会遇到的问题、造成问题的原因以及解决方法。

问题发生的原因

以图4-7为例,假设客户端为192.168.2.10以及服务器端为192.168.6.11,且客户端通过防火墙连接到因特网。防火墙的规则如下:

```
iptables -P FORWARD DROP
iptables -A FORWARD -i eth0 -o eth1 -p all -m state \
    --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p all -j ACCEPT
```



以图4-8为例来分析FTP协议的被动模式工作在防火墙环境下可能遇到的问题。图4-8与图4-7最大的差异在于防火墙的部署位置，在图4-8中，我们假设FTP服务器安放在防火墙后，也就是说，因特网上的使用者必须先穿过防火墙才可以访问到FTP服务，在此假设客户端的IP为192.168.2.10，服务器端的IP为192.168.6.11。防火墙的规则设置为：

```
iptables -P FORWARD DROP
iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn -d 192.168.6.11 \
    -m state --state NEW --dport 21 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -m state \
    --state ESTABLISHED -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p all -j ACCEPT
```

接着分析以上这个例子将会发生什么问题。当客户端192.168.2.10使用端口1955送出第一个数据包给服务器端192.168.6.11的端口21时，由于防火墙允许因特网上的使用者访问服务器端口21，因此，客户端送来的第一个数据包将可以正常送达服务器端，接着客户端会再使用端口21送出一个包含PASV命令的数据包给服务器端，由此告诉服务器端这一次使用被动模式来传输数据。接着服务器端会使用端口21送出一个包含“PORT 192,168,6,11,114,134”字符串的数据包给客户端，如此客户端就可以知道服务器端192.168.6.11在其上动态开启端口29318，客户端就会使用端口1956去访问服务器端的端口29318。但问题在于，我们在防火墙上并没有开放因特网上使用者可以访问服务器端的端口29318，况且也不可能在防火墙上开启端口29318，因为端口29318是服务器端以随机的方式决定的，因此，在上例中，客户端的访问操作将会失败。

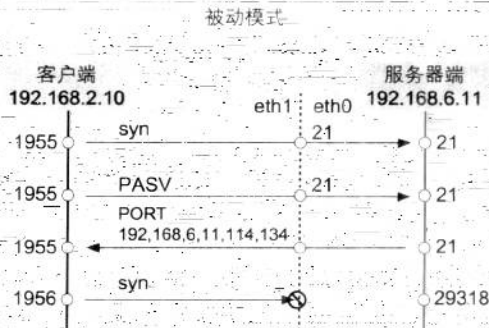


图4-8 被动模式和filter-2

在此首先要解释何谓“访问失败”？从以上例子可以清楚地看到，FTP协议中的两条连接的建立并没有完全失败，至少客户端访问服务器端口21的连接是成功的。前面我们曾解释过FTP协议的工作原理，其中客户端连接服务器端口21的连接为“命令”通道，另一条连接则是“数据”通道，因此，在以上的示例中，当客户端执行登录操作(Login)时，所有操作都可以正常完成，并不会产生任何的异常，不过在登录完成之后，如果客户端执行ls命令，由于FTP服务器传输数据给客户端是通过“数据”通道来传送的，但因为“数据”通道无法建立，所以此时FTP客户端就会出现如下的错误信息：

```
227 .Entering Passive Mode (192.168.6.11,114,134).
```

以上两个例子都是以被动模式为例加以说明，除了被动模式穿过防火墙会有问题之外，主动模式(Active Mode)穿过防火墙同样也会出现问题，至于问题产生的原因是什么？请你自己通过以上两个例子来推导，毕竟有思考才会有进步。

解决方案

通过阅读以上内容可以了解到，只要让FTP的“数据通道”可以正常穿过防火墙，那么所有问题都可以迎刃而解，但问题是该如何让防火墙“认识”FTP的数据通道呢？要解决这个问题并不难，请思考一个问题，图4-8中客户端使用端口1956连接服务器端端口29318这条连接，看起来像不像是RELATED状态的数据包呢？没错，其实FTP的数据通道就是state模块描述的RELATED状态，因此，只要在防火墙上允许RELATED状态的数据包正常通过，即可解决以上问题，只不过state模块并没有我们想象中的那样聪明。

第3.1.2一节中曾介绍过连接状态判断，而连接状态判断的幕后功臣就是nf_conntrack模块，不过请你务必要了解，nf_conntrack模块并非真的具有判断所有连接状态的能力，因此，单纯使用nf_conntrack模块无法将FTP协议的“数据通道”归类为RELATED状态，为此，Netfilter组织特别为各种不同复杂通信协议编写出不同的判断模块，如表4-1所示。

表4-1 Netfilter处理复杂通信协议的模块

模块名称	处理协议
nf_conntrack_amanda.ko	Advanced Maryland Automatic Network Disk Archiver(Amanda)备份服务协议
nf_conntrack_ftp.ko	FTP服务协议
nf_conntrack_h323.ko	视频服务协议
nf_conntrack_irc.ko	网络聊天室应用协议
nf_conntrack_netbios.ko	NETBIOS名称协议
nf_conntrack_pptp.ko	点对点的隧道协议(Point to Point Tunneling Protocol)
nf_conntrack_proto_gre.ko	通用路由封装协议(Generic Routing Encapsulation)
nf_conntrack_proto_sctp.ko	流控制传输协议(Stream Control Transmission Protocol)
nf_conntrack_sip.ko	会话初始协议(Session Initiation Protocol)是网络应用层的控制协议，通常用于VoIP环境
nf_conntrack_proto_dccp.ko	数据报拥塞控制协议(Datagram Congestion Control Protocol)是一种新的网络拥塞控制协议
nf_conntrack_proto_udplite.ko	udp lite是一种专门用来传输多媒体数据的协议，这种协议可以在传输质量不佳的网络上，依然保持良好的多媒体数据传输(例如无线网络的环境)
nf_conntrack_proto_sane.ko	SANE为一种访问远程扫描仪的通信协议
nf_conntrack_tftp.ko	普通文件传输协议(Trivial File Transfer Protocol)

从表4-1可以看到很多不同的模块，其中文件名的末端如“_ftp”就是表示这个模块是专门用来处理这种协议的，因此，对于nf_conntrack模块无法将FTP协议的数据通道归类为RELATED状态的问题，我们只要借助nf_conntrack_ftp模块就可以弥补nf_conntrack模块的不足，从而解决FTP协议穿过防火墙的问题。看到这里你一定会感到疑惑，万一我们所要处理的复杂通信协议并没有对应的模块，该如何处理呢？答案很简单，就是“没有模块就无法支持，而唯一能做的就是等待”，不过，如果你有能力执行kernel patch及编译Kernel的操作，或许就可以自己到Netfilter的官方网站或是因特网上寻找所需的模块，再将找到的“程序源代码”加入到内核中，也许就可以解决问题了。

由于nf_conntrack系列的模块众多，这里无法将每个模块的工作原理都逐一解释清楚，在此仅介绍nf_conntrack_ftp模块跟踪FTP协议的原理，至于其他模块的工作原理怎么样？如有兴趣请自行研究，不过，如果仅就防火墙规则的设置来看，原理其实也不是那么重要，只要我们能够了解这些模块的使用方法即可。

以图4-9为例来说明如何让FTP协议可以正常穿过Netfilter防火墙。可将上一个例子的防火墙规则按如下方式改写，如此即可让FTP客户端正常访问FTP服务器端，而这两份规则的差异就只有第一行及第四行规则，首先第一行是将nf_conntrack_ftp模块载入系统，以弥补nf_conntrack模块的不足之处，而第四行规则可让标记为RELATED状态的数据包正常穿过防火墙。

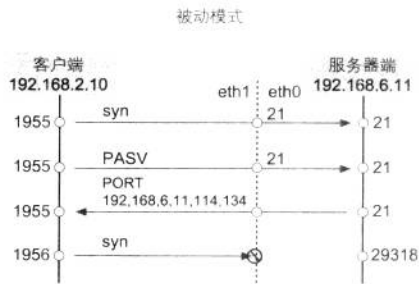


图4-9 被动模式和Filter-3

```
1. modprobe nf_conntrack_ftp
2. iptables -P FORWARD DROP
3. iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn -d 192.168.6.11 \
   --dport 21 -m state --state NEW -j ACCEPT
4. iptables -A FORWARD -i eth0 -o eth1 -p all -m state \
   --state ESTABLISHED,RELATED -j ACCEPT
5. iptables -A FORWARD -i eth1 -o eth0 -p all -j ACCEPT
```

接下来讨论nf_conntrack_ftp模块处理FTP协议的方式。nf_conntrack_ftp模块处理FTP协议的关键之处在于图4-9中的第三个步骤，nf_conntrack_ftp模块默认会监控所有与端口21相关的TCP数据包(源端口或目的地为21的TCP数据包)，并且从数据包中取出“192.168.6.11,114,134”字符串，如此nf_conntrack_ftp模块即可得知稍后客户端(192.168.2.10)会建立一条新的连接到FTP服务器端(192.168.6.11)的端口29318，nf_conntrack_ftp模块随即将192.168.2.10:1956 192.168.6.11:29318的连接归类为RELATED状态，最后再结合state模块将所有RELATED状态的

数据包放行，如此FTP协议的数据通道即可正常穿过防火墙了。

有一点特别注意，因为`nf_conntrack_ftp`模块默认仅监控与端口21有关的TCP数据包，如果FTP服务器端不是使用标准端口，`nf_conntrack_ftp`模块就无用武之地，因此，如果FTP服务器端未使用标准端口，在载入`nf_conntrack_ftp`模块时，就得加上额外参数，如“`modprobe nf_conntrack_ftp ports=21,2121,3131`”，其中的21,2121,3131就是告诉`nf_conntrack_ftp`模块凡与21,2121,3131端口有关的数据包都必须加以监控，如此即使FTP服务器不使用标准端口，`nf_conntrack_ftp`依然可以帮助FTP协议正常穿过防火墙，最后需要注意的是，参数`ports`最多只可以携带8个不同的端口号。

2. 复杂通信协议穿过NAT的问题及解决方案

复杂通信协议除了穿过防火墙会有问题外，复杂通信协议穿过NAT同样会有问题，而且问题可能比在防火墙上还要严重，但你也不需要太过于担心，因为Netfilter早有对应的方案了，下面就来讨论复杂通信协议穿过NAT时，问题发生的原因及解决方法。

发生问题的原因

以图4-10 FTP协议的主动模式穿过NAT为例来解释发生问题的原因。首先，假设图中NAT的右边即10.0.1.0/24网段是公网IP，NAT的左边即192.168.2.0/24网段是私有IP，另外，客户端的IP为192.168.2.10，而服务器端的IP为10.0.1.12。

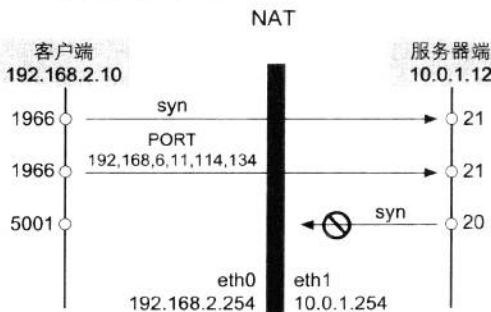


图4-10 FTP协议穿过NAT的问题

首先客户端发送连接请求数据包给服务器端，接着，客户端再发送一个包含“`PORT 192,168,2,10,19,137`”字符串的数据包给服务器端，由此告诉服务器端，稍后客户端的数据通道会在端口5001(19*256+137)的地址上，服务器端在收到这个数据包之后，随即针对192.168.2.10的端口5001建立数据通道，不过此时服务器端连接的对象会是一个私有IP，因此，FTP协议的数据通道建立将失败，而此时客户端将会看到以下错误信息：

```
500 Illegal PORT command.
```


解决方案

在了解FTP协议穿过NAT问题发生的原因之后，相信你大概可以猜到解决问题的办法，只要让服务器端(10.0.1.12)的端口20可以成功建立连接到客户端(192.168.2.10)的端口5001，即可让FTP客户端成功穿过NAT访问FTP服务器端。为此，Netfilter特别为FTP协议开发了一个nf_nat_ftp模块，以弥补原NAT机制功能上的不足。

那么，nf_nat_ftp是如何帮助FTP协议穿过NAT的呢？其中关键就在于“PORT 192.168.2.10,19,137”这个字符串，nf_nat_ftp模块的主要任务就是监控所有与端口21相关的数据包，并将数据包内所包含“PORT 192.168.2.10,19,137”字符串中的客户端地址改成NAT主机上的公网IP地址，因此FTP服务器端会误以为NAT主机就是FTP客户端，所以FTP服务器端即对NAT主机提出数据通道建立请求，当数据包送达NAT主机之后，NAT主机即将数据包转发给FTP客户端，如此FTP客户端即可正常穿过NAT来访问FTP服务器端。此外，与nf_conntrack_ftp模块有着相同的问题，如果FTP服务器端不是使用标准的端口21，nf_nat_ftp模块将无法发挥作用。

在kernel<=2.6.10的版本中，nf_nat_ftp模块名为ip_nat_ftp，ip_nat_ftp模块允许带ports的参数，如“modprobe ip_nat_ftp ports=21,2121”，不过，在kernel>2.6.10的版本中ip_nat_ftp的名称已改为nf_nat_ftp，而且nf_nat_ftp不再提供ports的参数可供使用，讲到这里或许你会觉得，如果我的FTP服务器端不是使用标准的端口21，那该怎么办？其实不必担心这个问题，只要在载入nf_nat_ftp模块之前，先把nf_conntrack_ftp模块载入并设置好ports的参数，接着再载入nf_nat_ftp模块即可，因为nf_nat_ftp模块会自动套用nf_conntrack_ftp模块的参数，如下：

```
modprobe nf_conntrack_ftp ports=21,2121
modprobe nf_nat_ftp
```

4.4.3 ICMP数据包的处理原则

ICMP(Internet Control Message Protocol)的用途是传输控制信号或传输某种信息，例如，我们可以运用ICMP type 11的特性来查找网络上两台主机间共跨越了多少个路由器；又如我们可以使用ICMP type 8的特性，检测因特网上两台主机间的网络是否正常，总之，ICMP协议能做的事非常多，若你有兴趣不妨自己阅读RFC文档。

虽然ICMP数据包种类繁多，但并不会增加在防火墙上的设置难度，因为在实际应用中，我们很少需要去接收其他主机“主动”送来的ICMP数据包。也就是说，在设置防火墙时可以直接把NEW状态的ICMP数据包丢弃掉，而不会影响到网络服务的正常运行。不过，如果因为我们因特网发送数据包而“引发”返回的ICMP的数据包，就必须让ICMP数据包能够正常穿过防火墙，然后回到我们的主机上。如果在主机上执行ping 168.95.192.1，而168.95.192.1主机应答ICMP数据包，就应该让其可以正常返回，也就是说，应该在防火墙上

开放ESTABLISHED状态的ICMP数据包可以正常返回。又如, 当我们对因特网上的主机送出数据包时—因为某种原因造成数据包传输时的错误, 由因特网上的设备(如路由器)送回的ICMP包应该让其可以正常返回。简单来说, 就是RELATED状态的ICMP数据包。

综上所述, 在防火墙上处理ICMP数据包时只需把握以下两条原则:

- 放行所有由因特网送来的ESTABLISHED及RELATED状态的ICMP数据包。
- 丢弃所有由因特网送来的其他状态的ICMP数据包。

4.4.4 在DMZ上使用NAT将面临的问题及解决方案

第2章已经介绍过Netfilter-NAT的机制, 相信你对NAT机制应该已经有所了解, 下面以图4-11为例来说明在DMZ上使用NAT所面临的问题及解决方案。首先来分析以下一对一NAT规则, 相信你会同意这样的设置规则, 但请先仔细思考这些规则是否真的符合我们的需要?

```
# 10.0.1.201 对192.168.0.1
iptables -t nat -A PREROUTING -i eth0 -d 10.0.1.201 -j DNAT --to 192.168.0.1
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.1 -j SNAT --to 10.0.1.201

# 10.0.1.202 对192.168.0.2
iptables -t nat -A PREROUTING -i eth0 -d 10.0.1.202 -j DNAT --to 192.168.0.2
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.2 -j SNAT --to 10.0.1.202

# 192.168.1.0/24 对 internet
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.1.0/24 -j SNAT --to 10.0.1.200
```

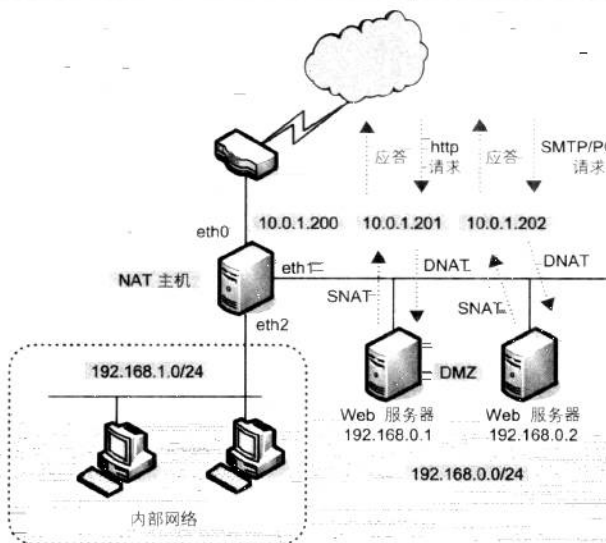


图4-11 一对一NAT

我们从四个方面展开讨论，首先是因特网对DMZ内主机的访问操作；第二是DMZ内主机对因特网的访问操作；第三是内部网络对因特网的访问操作；第四是内部网络对DMZ内主机的访问操作。以上规则可以符合前三种状况的要求(在此先不考虑复杂通信协议的问题)，但如果是第四种情况就会产生超乎我们预期的状况，下面将说明这个问题发生的原因及解决办法。

以内部网络主机192.168.1.1访问DMZ网络内的Web服务器为例来说明发生问题的原因：

(1) 因为Web服务器是要提供服务给因特网的使用者来访问的，因此，假想公司(example.com)的DNS服务器上一定存在“www IN A 10.0.1.201”的记录。

(2) 当192.168.1.1上的使用者在浏览器的网址列输入www.example.com时，浏览器即通过DNS服务器解析得到www.example.com主机的IP，而192.168.1.1主机所得到的应答应该就是10.0.1.201。

(3) 在192.168.1.1得到www.example.com所对应的IP后，192.168.1.1随即发送服务请求包给10.0.1.201，当这个数据包送达图4-12中❶的位置时，其内的SRC_IP应为192.168.1.1，而DST_IP应为10.0.1.201。

(4) 不过，当数据包进入PREROUTING链之后，虽然PREROUTING链内有我们所设置的规则(iptables -t nat -A PREROUTING -i eth0 -d 10.0.1.201 -j DNAT --to 192.168.0.1)，但规则中所指的接口为“-i eth0”，因此，这个请求包根本就不会符合这条规则，所以这个数据包到达❷的位置时，其内的SRC_IP依然是192.168.1.1，而DST_IP也依旧是10.0.1.201。

(5) 接着，数据包被送入路由表做路由判断，因为10.0.1.201这个IP被设置在eth0接口上，因此，这个数据包将被送入❸的位置，而被防火墙主机所接收，如果此时防火墙主机上的Web服务器处于启动状态，那么，192.168.1.1主机所看到的网页将是防火墙主机上Web服务器的网页，而非预期中Web服务器(192.168.0.1)的网页。

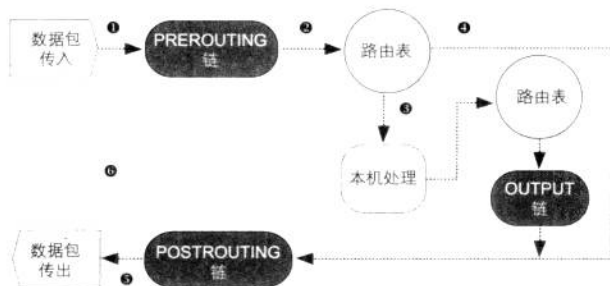


图4-12 NAT的结构

通过学习以上内容，相信你已经清楚了解问题发生的原因，该如何解决呢？下面提出两

种较常见的解决方案。

● 将example.com这个域名分为“内”、“外”DNS:

前面的例子假设内部网段主机解析www.example.com所得到的IP为10.0.1.201, 所以需要
使用DNAT的机制将目的端IP变更为192.168.0.1, 我们在内部网络也布置一台DNS 服务器,
但在这台DNS 服务器上特意将www记录的IP指向192.168.0.1。接着, 再将所有192.168.1.0/24
网段的名称解析操作交给内部的DNS 服务器来执行, 这样内部网段主机解析www.example.
com时所得到的IP将是192.168.0.1, 因此, 内部网段的主机就可以在不需执行NAT的情况
下, 直接访问DMZ网段内的服务。

● 从NAT机制本身着手:

因为从内部网络送到10.0.1.201这个IP的数据包不符合DNAT-NAT的规则时才会产生这
个问题。因此, 要解决以上问题并非难事, 只需多加一条192.168.1.0/24网段到10.0.1.201的
DNAT规则即可, 如:

```
iptables -t nat -A PREROUTING -i eth2 -d 10.0.1.201 -j DNAT --to 192.168.0.1
```

或许你会联想到是否该为192.168.0.1到192.168.1.0/24网段设置SNAT的规则, 其实这倒
不必, 在此只需要加入DNAT规则即可, 接着让我们来分析一下流程, 就可得知为何不需要
设置SNAT规则, 推导如下:

(1) 当192.168.1.1对10.0.1.201送出服务请求包, 在数据包达到❶的位置时, 数据包内的
SRC_IP为192.168.1.1, 且DST_IP为10.0.1.201。

(2) 数据包进入PREROUTING链, 因为在PREROUTING链内有我们刚刚加入的规则
(iptables -t a -A PREROUTING -i eth2 -d 10.0.1.201 -j DNAT --to 192.168.0.1), 因此, 数据包内
DST_IP会被DNAT机制改为192.168.0.1。

(3) 接着, 数据包进入路由表执行路由判断, 因为DST_IP已经被更改为192.168.0.1, 因
此, 数据包随即进入❷的位置, 最后数据包经由eth1接口送出。

(4) 当Web 服务器收到这个数据包时, 其内的SRC_IP为192.168.1.1, 而DST_IP为
192.168.0.1, 因此, Web 服务器即将数据包回送到192.168.1.1。

(5) 接着, Web 服务器回送数据包给来源端, 而这个数据包内的SRC_IP为192.168.0.1,
而DST_IP为192.168.1.1。

(6) 因为这个数据包内的SRC_IP及DST_IP都是防火墙附近网段的IP, 因此, 这个数据包
可以通过防火墙本身的路由机制, 将数据包送回到来源端主机, 如此, 我们便可以在只设置
DNAT而不设置SNAT的情况下, 让内部网络的主机可以正常访问DMZ网段内的服务。

4.4.5 常见的网络攻击手段及防御方法

如果将网络世界比作是现实生活，那么你一定觉得现实生活的治安实在是太好了，因为在网络世界，无时无刻都有受到黑客攻击的可能性，如果你有已部署在因特网上一段时间的Linux主机，不妨查看一下/var/log/secure[1234]这些文件内是包含如下信息，这些信息就是你的主机遭到攻击时所留下的记录。

```
Sep 3 19:03:54 mail sshd[31950]: Failed password for root from 70.102.115.234 port 53639 ssh2
Sep 3 19:03:58 mail sshd[31953]: Failed password for root from 70.102.115.234 port 54951 ssh2
Sep 3 19:04:03 mail sshd[31955]: Failed password for root from 70.102.115.234 port 56036 ssh2
Sep 3 19:04:07 mail sshd[31958]: Failed password for root from 70.102.115.234 port 57296 ssh2
Sep 3 19:04:12 mail sshd[31961]: Failed password for root from 70.102.115.234 port 58146 ssh2
Sep 3 19:04:17 mail sshd[31963]: Failed password for root from 70.102.115.234 port 59464 ssh2
```

从以上这些信息可以得知，70.102.115.234这台主机试图以ssh连接的方式，来“猜测”当前Linux系统上root帐户的密码，当然，这个猜测动作绝对不是黑客以手动的方式在尝试，而是以程序来执行“猜密码”的操作，因此，如果你系统上密码的组成没有严格的限制(比方说，密码至少需要8个字符，一定要有大小写字母、符号及数字等共同组合)，那么，你的密码可能很快就会被猜到，当然系统也就宣告沦陷了。

由于网络攻击手段众多，而且有些可用防火墙来防范，有些则不行。比方说，上述例子中的“密码攻击”手法，我们就可以轻易地依靠防火墙来抵挡，但某些属于服务本身的安全漏洞，防火墙就不一定有效了。不过，有防火墙总比没有好，你务必要有一个概念，就算有防火墙系统，也非绝对安全，所以绝对有必要定期更新系统上的软件。接下来就针对一些常见的网络攻击手段来分析其攻击的行为及防御方法。

1. PortScan攻击

PortScan(端口扫描)是一种极常见的攻击行为，其目的是在探测特定主机上有哪些服务是开启的，如下即为nmap PortScan工具执行后所得到的结果。接着，再从这些信息来分析可能有机会入侵的通道在哪里，因此，PortScan的攻击通常都只是真正入侵之前的准备，而真正的攻击行动可能随之展开，因此，如果能有效防御PortScan攻击，将有助于提高系统的安全性。

```
[root@mail /]# nmap 59.120.????.???
Starting Nmap 4.03 ( http://www.insecure.org/nmap/ ) at 2007-09-11 14:27 CST
Interesting ports on 59-120-???-???-HINET-IP.hinet.net (59.120.???-???):
(The 1665 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
21/tcp open ftp
80/tcp open http
```

```
135/tcp open msrpc
139/tcp open netbios-ssn
443/tcp open https
MAC Address: 00:91:1A:718:06:F1 (Unisphere Solutions)
Nmap finished: 1 IP address (1 host up) scanned in 6.272 seconds
```

接下来将讨论PortScan攻击的原理，以及如何使用Netfilter来防御PortScan攻击。



提示

在讨论PortScan的攻击原理之前，请回顾一下TCP连接建立及终止的过程，如果你不记得，不妨复习一下第3.1.2一节中的内容。

PortScan的原理

其实PortScan技术并没有想象得那样难懂，PortScan只是运用TCP/IP某些特性来达到扫描的目的，比方说，我们可以对某主机的端口80送出一个带有SYN标记的数据包，如果这台主机的端口80是打开的，该主机就会应答一个带有SYN及ACK标记的数据包，我们可以从“TCP三次握手”去验证这件事；相反，如果这台主机的端口80没有打开，这台主机就会应答一个带有RST及ACK标记的数据包，因此，我们可以使用TCP的这个特性来检测某一台主机提供哪些服务。以上这种方式称为Syn Scan，当然Port Scan的方式还有很多种，如FIN Scan、Null Scan等。

使用Netfilter来防御PortScan的攻击

虽然PortScan攻击的手段相当多，但我们并不需要了解每种PortScan的攻击原理，因为这并不影响我们检测及防御PortScan攻击，不过，我们必须要了解PortScan攻击的“特性”，PortScan会探测出系统有哪些端口是打开的，因此，被攻击的主机的几乎每个端口都会收到一个探测数据包，在了解这个特性之后，要侦测及防御PortScan攻击就容易多了。

假设我们有一台主机，其上正运行SSH、SMTP、HTTP及POP3这四项服务，我们应该如何运用PortScan的特性来防止PortScan的攻击呢？首先分析这台主机，如下所示：

- (1) 客户端访问主机端口22、25、80及110都属于正常行为。
- (2) 客户端访问端口22、25、80及110以外的端口都属于不正常行为。
- (3) PortScan想要侦测出我们主机有哪些端口是打开的，因此，PortScan几乎会对我们主机的每个端口送出探测数据包，而这些探测数据包在这一点及第二点的定义中绝大多数都属于不正常行为。
- (4) 当然客户端有可能“不小心”打错IP，而产生不正常的行为，但这种情况的概率很低，可以忽略不计。

整理出以上四条规则之后，我们就可以运用recent模块来防止PortScan攻击，下面分析作者所编写的PortScan攻击防范规则：

(1) 请先看第4行规则，这条规则是说，只要有任何数据包符合本规则时，recent模块就会把这个客户端的IP及数据包的信息记录在port_scan这个数据库中。

(2) 接着再看第3行规则，这条规则是说，如果传入的数据包是TCP协议的数据包而且数据包是要送到端口22、25、80或110的，就符合这条规则，如果数据包符合这条规则，那么，数据包就不会被送到第4条规则，当然就不会被记录在port_scan的数据库中。

(3) 因为PortScan的特性会对主机的每个端口送出探测包，因此，这些大量的探测包除了有可能匹配到端口22、25、80或110之外，其他大多数的探测包应该都会匹配到第4行规则而被记录下来。

(4) 第2条规则为匹配port_scan数据库的内容，如果这个数据包的“特征”在数据库中已有记录，我们就更新数据库中的信息。另外，如果从目前这个时间点往前推1800秒，且在port_scan数据库中有超过10次以上的记录，就把这个数据包丢掉。

这个示例中以半小时作为单位，如果在半小时之内客户端访问端口22、25、80或110以外的端口超过10个以上，我们就判定为PortScan攻击。

```
1. iptables -A INPUT -p all -m state --state \
    ESTABLISHED,RELATED -j ACCEPT
2. iptables -A INPUT -p all -m state --state NEW -m recent --name port_scan \
    --update --seconds 1800 --hitcount 10 -j DROP
3. iptables -A INPUT -p tcp --syn -m state --state NEW \
    -m multiport --dports 22,25,80,110 -j ACCEPT
4. iptables -A INPUT -p all -m recent --name port_scan --set
```

2. 密码攻击

“密码攻击”是指黑客运用主机上需要经过帐户及密码验证后才能访问的服务，来达到密码攻击的目的。这里以POP3服务为例来解释密码攻击的原理。首先可以使用telnet 客户端工具来连接特定的POP3 服务器❶，并使用user命令来传输帐户给POP3 服务器❷，接着，POP3 服务器应答+OK代表服务器已经收到命令❸，客户端再使用pass命令来传输密码给POP3 服务器❹。接着，POP3 服务器就会执行帐户及密码的验证操作，此时，如果帐户或密码不正确，POP3 服务器就会“固定”应答“-ERR Authentication failed”字符串❺，以告诉客户端验证操作没有成功，不过，客户端可以再次使用user及pass命令来进行帐户或密码的验证操作，如果客户端发送的帐户和密码是正确的，POP3 服务器就会“固定”应答“+OK Logged in.”的字符串❻，以通知客户端：你已通过帐户及密码的验证操作。

在了解POP3协议的特性之后，相信你已具备执行密码攻击的能力，我们只需通过telnet

客户端工具, 再加上极大的耐心及毅力相信总有一天可以把对方的密码给“猜”出来, 不过真要这么做, 一定会被累死, 因此, 就有黑客将这些纯手工的步骤写成工具程序, 执行者只需告诉工具所要攻击的IP, 该工具就会自动将对方的密码猜出来, 根本就无需耗费任何人力资源。

```
[root@localhost ~]# telnet 10.0.1.101 110 ❶
+OK Dovecot ready.
user kevin ❷
+OK ❸
pass 11111 ❹
-ERR Authentication failed. ❺
user kevin
+OK
pass 12345
+OK Logged in. ❻
quit
+OK Logging out.
```

● 未加密通信协议下的密码攻击防御(以POP3通信协议为例)

接着, 我们来分析有何方法可以防止POP3协议的密码攻击。因为POP3属于未加密的通信协议, 因此, 我们可以使用string模块来匹配POP3协议中所传送的数据内容, 或许你马上就联想到使用recent这个模块来计算特定时间内客户端总共发送几次“pass”这个字符串, 并将这作为我们防御密码攻击的依据。其实, 以上的思考逻辑完全正确, 只不过某些通信协议为了加强其验证的安全性, 会故意将客户端送给服务器端的数据拆解成一个个数据包来传送, 如图4-13所示(可由wireshark工具所截取的数据包来证明), 即使我们使用string模块再加上recent模块, 也无法算出客户端总共猜错多少次密码。

3	0.000807	192.168.5.200	192.168.5.179	TCP	1881 > 110 [ACK] Seq=1 Ack=1 win=65535 [TCP C
4	0.001796	192.168.5.179	192.168.5.200	POP	Response: +OK Dovecot ready.
5	0.200828	192.168.5.200	192.168.5.179	TCP	1881 > 110 [ACK] Seq=1 Ack=21 win=65515 [TCP C
6	1.117789	192.168.5.200	192.168.5.179	POP	Request: u
7	1.118262	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=2 win=5840 Len=0
8	1.246843	192.168.5.200	192.168.5.179	POP	Request: s
9	1.247269	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=3 win=5840 Len=0
10	1.384930	192.168.5.200	192.168.5.179	POP	Request: e
11	1.385279	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=4 win=5840 Len=0
12	1.580495	192.168.5.200	192.168.5.179	POP	Request: r
13	1.581092	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=5 win=5840 Len=0
14	2.131539	192.168.5.200	192.168.5.179	POP	Request:
15	2.131927	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=6 win=5840 Len=0
16	2.320779	192.168.5.200	192.168.5.179	POP	Request: d
17	2.521152	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=7 win=5840 Len=0
18	2.668165	192.168.5.200	192.168.5.179	POP	Request: a
19	2.668962	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=8 win=5840 Len=0
20	2.797433	192.168.5.200	192.168.5.179	POP	Request: v
21	2.797970	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=9 win=5840 Len=0
22	2.935691	192.168.5.200	192.168.5.179	POP	Request: f
23	2.936180	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=10 win=5840 Len=0
24	3.074214	192.168.5.200	192.168.5.179	POP	Request: g
25	3.074788	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=11 win=5840 Len=0
26	3.317632	192.168.5.200	192.168.5.179	POP	Request:
27	3.318203	192.168.5.179	192.168.5.200	TCP	110 > 1881 [ACK] Seq=21 Ack=13 win=5840 Len=0
28	3.318399	192.168.5.179	192.168.5.200	POP	Response: +OK

图4-13 POP3协议的验证过程

不过你也不必太过沮丧, 虽然我们无法将“pass”字符串作为匹配目标, 但可以将匹配

条件换成“-ERR Authentication failed.”这个字符串，因为只要客户端验证失败，服务器端就会送出“-ERR Authentication failed.”这个字符串给客户端，因此，我们可以使用这个字符串作为匹配条件。

为防御POP3协议的密码攻击，以下示例所设置的条件是：如果在10分钟内，客户端验证失败次数超过6次以上，就认定为密码攻击，规则如下：

```
1. iptables -A OUTPUT -p tcp --sport 110 -m string --algo bm \
    --string "-ERR Authentication failed:" \
    -m recent --name pop3 \
    --update --seconds 600 --hitcount 6 -j REJECT
2. iptables -A OUTPUT -p tcp --sport 110 -m string --algo bm \
    --string "-ERR Authentication failed:" \
    -m recent --name pop3 --set
```

从以上的示例可以了解到，未加密通信协议的密码攻击防御并没有固定的规则可循，因此，你必须学会wireshark这类软件的使用，并善用这类软件来帮助分析想要监控的协议；在规则第1行中，为何用的TARGET会是REJECT而不是DROP呢？关于这个问题，就留给你自己去思考了（可参阅第3.2.2一节的第2部分的内容）。

● 加密通信协议下的密码攻击防御(以SSH通信协议为例)

在未加密的通信协议中，可以使用string模块来检测“验证失败”的信息字符串，但在加密通信协议的处理上，可就没那么幸运了！因为通信过程中的所有数据包都已经加密，因此，我们根本不可能使用string模块来检测任何“验证失败”的信息字符串，所以只好寻找其他办法。

下面以最常见的SSH协议为例，来说明如何防御SSH协议的密码攻击。首先要了解SSH的验证特性，才有办法进一步找到防御方法。以图4-14为例，CentOS 6.0及RHEL 6内置的SSH服务器默认允许用户不小心输错6次密码，如果第7次还是输错密码的话，SSH服务器就会自动终止与SSH客户端的连接。



图4-14 SSH协议的验证操作

在了解SSH协议的验证特性之后，你是否想出正确的防御方式呢？我所想到的方法是，因为CentOS 6.0及RHEL 6系统默认的SSH连接允许用户不小心输错6次密码，如果在第7次还是不正确，SSH 服务器就会终止与SSH的连接，如果是系统遭受到密码攻击的话，那么攻击者势必会在此对SSH 服务器建立新的连接，然后继续进行猜测密码的操作。总结以上这些特性可以得到一个结论，每当客户端对我们的SSH 服务器送出一个带有SYN标记的数据包之后，客户端就可以猜7次密码，因此，要防御SSH协议的密码攻击，必须从下列两点着手：

- 改变SSH 服务器所能接受的最大密码错误次数：

SSH 服务器所能接受的最大密码错误次数由MaxAuthTries参数定义，因此，只需修改SSH 服务器的配置文件/etc/ssh/sshd_config，并将MaxAuthTries参数值设置为所要指定的次数，然后重新启动SSH 服务器即可。

- 将送至SSH 服务器的SYN数据包数量作为匹配依据：

由于SSH协议的数据包都是经过加密处理的，所以string模块是帮不上忙的，我们可以改用TCP连接三次握手的第一个数据包，来作为防御密码攻击的匹配目标。

综上所述，可以使用以下规则来防御SSH协议的密码攻击。首先假设，如果同一个客户端在10分钟内验证失败次数超过12次，就认定系统遭到密码攻击，因此，我们先将SSH 服务器所能接受的最大密码错误次数设定为2，这样客户端在输入第3次错误的密码之后，SSH 服务器就会终止与SSH 客户端的连接。我们所设置的条件是：如果客户端在10分钟之内，验证失败次数超过12次，就认定系统遭受到密码攻击。因此，所要匹配的SYN数量为4次以上，且时间范围是在10分钟之内。依据以上条件编写的规则如下：

```
iptables -A INPUT -p tcp --syn --dport 22 -m recent --name ssh \
--update --seconds 600 --hitcount 4 -j DROP
iptables -A INPUT -p tcp --syn --dport 22 -m recent --name ssh --set
```

3. 何谓DOS及DDOS网络攻击

所谓DOS(Denial of Service)攻击是指严重消耗被攻击者的系统或网络资源，由此达到干扰需要访问该服务的正常使用者。例如，大量消耗被攻击者的CPU资源，如此该服务器就无法快速应答需要访问该服务的使用者，或者设法让被攻击者的对外带宽耗尽，这样需要访问该服务器的使用者就可能因为连接速度过慢，而导致连接超时。总之，DOS攻击的目的是让被攻击者的服务器无法提供正常服务，如此DOS攻击就算达到了目的。

可以使用很多不同的方法来达到DOS攻击的目的。例如，在Linux系统中使用ping -f 192.168.0.1命令，这时ping命令就会向192.168.0.1主机疯狂地持续发送icmp数据包，此时

192.168.0.1主机也会随之疯狂地回复icmp数据包,如此一来,就可以占用192.168.0.1系统CPU的一些资源;但如果使用ping命令发动DOS攻击,而且在只有一台攻击主机的情况下,想要达到瘫痪192.168.0.1主机的概率并不高(如图4-15所示),因为系统应答icmp所占用的CPU资源并不多,因此,黑客组织通常会在网络上到处入侵他人的主机,并在这些被入侵的主机上植入木马程序,当有攻击目标时,黑客就可以运用这些木马程序对同一个目标主机发动攻击(如图4-16所示),如此攻击的来源就不再只是一台,而是一个群体,因此,在这种攻击模式下,要瘫痪掉某台主机的概率就大大增加了,而这种方式就是所谓的DDOS(Distributed Denial of Service, 分布式DOS攻击)。



图4-15 DOS攻击

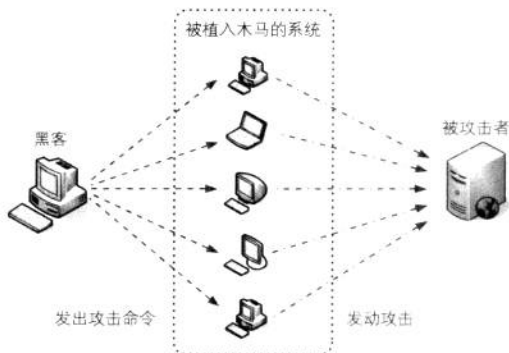


图4-16 DDOS攻击

● Syn Flooding攻击

Syn Flooding攻击也是一种DOS攻击,而且Syn Flooding是公认的最容易实施的网络攻击,同时也是最有效率且最难防御的攻击行为,因为Syn Flooding是利用TCP协议本身设计上的瑕疵来达成攻击的目的,因此,攻击者只需要少

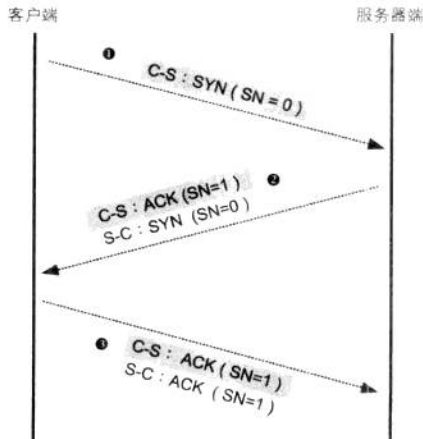


图4-17 TCP三次握手

● TCP三次握手的流程:

要了解Syn Flooding的攻击原理以及防御方式,就必须对TCP连接的三次握手机制有所理解。这里以图4-17为例来说明建立TCP连接的过程:

(1) 首先客户端送出连接请求包给服务器端①,

而这个数据包内将会包含一个SYN标记及一组序号, 序号的用途为记录客户端到目前为止总共传输了多少字节的数据给服务器端。不过, 序号的初始值为0, 当客户端送出这个数据包之后, 随即进入SYN_SENT状态(如图4-18), 并等待服务器端的应答。

```

root@localhost:~# netstat -na
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1:631          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1:25          0.0.0.0:*               LISTEN
tcp        0      0 10.0.1.101:56465       10.0.1.102:22          SYN_SENT
tcp        0      0 0.0.0.0:110           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22            0.0.0.0:*               LISTEN

```

图4-18 TCP三次握手: SYN_SENT

(2) 服务器端在收到这个数据包之后, 便会应答客户端一个确认数据包②, 而这个数据包内将带有SYN及ACK两个标记, 下面分别描述这两个标记。

• ACK标记:

ACK标记代表服务器端应答给客户端的确认数据包, 而且这个标记会伴随着一个ACK编号, 在这个步骤中的ACK编号固定为上一个步骤的序号+1, 在之后的步骤中, ACK编号将会固定等于前一个步骤的序号值。

• SYN标记:

第一个步骤提到过, 序号的用途是在于记录客户端到目前为止, 总共传输了多少字节的数据给服务器端。不过, 除了客户端到服务器端的这个方向会有序号值之外, 其实服务器端到客户端的方向也会有一个序号, 而这个步骤中的SYN标记则代表着服务器端传输给客户端的第一个数据包, 且这个SYN标记也会伴随一组序号。在这个例子, 笔者假设序号的初始值为0。

在服务器端送出这个数据包之后, 随即进入SYN_RECV状态(如图4-19), 并等待客户端的应答, 在此同时, 服务器端会把这个客户端的相关信息记录在一块我们称为“TCP队列”的内存空间中。

```

root@localhost:~# netstat -na
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:32769          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1:25          0.0.0.0:*               LISTEN
tcp        0      0 10.0.1.102:22          10.0.1.101:56465       SYN_RECV
tcp        0      0 0.0.0.0:1:631          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:1:25          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22            0.0.0.0:*               LISTEN

```

图4-19 TCP三次握手: SYN_RECV

(3) 客户端在收到服务器端所应答的数据包之后③, 接着应答确认数据包给服务器端,

而这个数据包将带有ACK标记：一个序号及一个ACK编号。接下来分别讨论这两个ACK标记：

● 客户端到服务的方向(C→S)的ACK标记

在上个步骤中所送回来的ACK编号为1，而这个步骤由于客户端并不会发送任何的数据给服务器端，因此，这个数据包内的序号等于1。

● 服务的到客户端方向(S→C)的ACK标记

上个步骤中，SYN的序号等于0，而在这个步骤，S→C方向的ACK编号固定为前一个步骤的序号+1，之后步骤中ACK编号将会等于前一个步骤的序号值。

在客户端发送出这个数据包之后，其连接状态随即进入ESTABLISHED状态，以图4-20为例，服务器端在收到这个数据包之后，随即匹配TCP队列中的连接信息，如果信息存在(即完成①、②两个步骤)，那么三次握手即告完成，TCP连接也就正式建立了，同时服务器端会将TCP队列中属于这条连接的缓存信息清除，以腾出队列空间给新的连接来使用。

```
root@localhost:~# netstat -na
Active Internet connections (servers and established)
```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	*:::0.0.0.0:::2208	*:::0.0.0.0:::	LISTEN
tcp	0	0	*:::0.0.0.0:::13631	*:::0.0.0.0:::	LISTEN
tcp	0	0	*:::0.0.0.0:::135	*:::0.0.0.0:::	LISTEN
tcp	0	0	*:::0.0.0.0:::10.0.1.101:56465	*:::10.0.1.10:22	ESTABLISHED
tcp	0	0	*:::0.0.0.0:::1110	*:::0.0.0.0:::	LISTEN
tcp	0	0	*:::0.0.0.0:::82	*:::0.0.0.0:::	LISTEN

图4-20 TCP三次握手，ESTABLISHED

最后利用图4-21来展示TCP三次握手的过程，其中前三个数据包即为三次握手中所传输的三个数据包，并且我们可以从info字段中看出三次握手的完整过程。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.37	202.43.195.13	TCP	1367 > http: [SYN] Seq=0 Len=0 MSS=1460
2	0.546166	202.43.195.13	192.168.0.37	TCP	http > 1367: [SYN, ACK] Seq=0 Ack=1 win=0
3	0.546234	192.168.0.37	202.43.195.13	TCP	1367 > http: [ACK] Seq=1 Ack=1 win=17040
4	0.551180	192.168.0.37	202.43.195.13	HTTP	GET / HTTP/1.1
5	1.332397	202.43.195.13	192.168.0.37	HTTP	HTTP/1.1 302 Found (text/html)
6	1.333854	202.43.195.13	192.168.0.37	TCP	http > 1367: [FIN, ACK] Seq=326 Ack=286
7	1.333922	192.168.0.37	202.43.195.13	TCP	1367 > http: [ACK] Seq=286 Ack=327 win=0

图4-21 三次握手建立过程

● TCP三次握手的致命危险：

在了解TCP三次握手的过程之后，你是否已经发现TCP三次握手的缺点呢？以图4-17为例，其实问题就发生在三次握手的第三个步骤，我们知道在第二个步骤中，服务器端会把连接信息记录在TCP队列中，而这一条信息会一直存放在TCP队列中，直到服务器端收到客户端的确认数据包之后，才会从队列中清除掉。

但问题来了，如果客户端故意不回送确认包③给服务器端，服务器端在经过一段时间后，会重新再发送一次第二个步骤的数据包②给客户端。如果经过一段时间之后仍然无法得

到客户端的应答时,服务器端依然会再次重发第二个步骤中的数据包②给客户端,而这样的重试动作将会持续5次之多,如果最后还是无法得到客户端的应答,服务器端就会从TCP队列中清除该条连接的缓存信息。不过,这5次的重试时间将会长达3分15秒之久(以单纯读秒方式计算得到的值)。

如果以上情况是在正常的网络环境中偶尔发生几次,倒也不至于对系统造成什么影响,万一客户端不断地对服务器端发送大量的SYN数据包(三次握手的第一个数据包),却又故意不回送确认包(三次握手的第三个数据包)给服务器端时,服务器端的TCP队列很可能就被这些无法正常建立连接的信息给占满,因而导致其他正常需要访问服务的链接无法被排入TCP队列,当然TCP连接也就无法正常地建立起来,如此即达到Syn Flooding攻击的目的。图4-22为一台主机遭受到Syn Flooding攻击时的网络连接状态信息,从图中我们可以清楚看到10.0.1.1及10.0.1.101之间有大量SYN_RECV状态的连接,而这就是主机遭受Syn Flooding攻击时表现出来的特征。

tcp	0	0	10.0.1.101:80	10.0.1.1:1706	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1981	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1276	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1361	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1453	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1867	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1984	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1392	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1728	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1611	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1406	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1258	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1444	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1132	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1803	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1918	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1677	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1559	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1520	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1665	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1490	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1979	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1185	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1800	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1563	SYN_RECV
tcp	0	0	10.0.1.101:80	10.0.1.1:1215	SYN_RECV

图4-22 DOS攻击

● Syn Flooding攻击的可怕之处

其实单纯的Syn Flooding攻击是很容易防御的。以图4-22为例,我们可以从图中轻易确定攻击者的IP地址是10.0.1.1,因此,只需通过recent模块来判断SYN的“量”,即可轻易消除Syn Flooding的攻击威胁,但是攻击者绝不可能就这样放过我们的,因此,黑客实际在执行Syn Flooding攻击时,都会通过不断改变“来源端IP”,让我们无法将“Syn的量”当作防

御条件。更糟糕的是，我们根本无法根据一个“SYN数据包”来区分攻击者和正常的服务请求者。

因此，黑客如果采用Syn Flooding攻击手段，辅以伪造来源端地址，并使用DDOS这种大规模的攻击方式(如图4-23所示)，应该不会有哪个网站可以承受得了，连美国Yahoo! 如此庞大的网站也只能被迫关闭休息。

tcp	0	0	10.0.1.101:80	161.41.122.118:1085	SYN_RECV
tcp	0	0	10.0.1.101:80	100.226.254.175:1035	SYN_RECV
tcp	0	0	10.0.1.101:80	160.151.37.184:1593	SYN_RECV
tcp	0	0	10.0.1.101:80	101.120.8.90:1811	SYN_RECV
tcp	0	0	10.0.1.101:80	111.42.58.8:1592	SYN_RECV
tcp	0	0	10.0.1.101:80	62.18.117.38:1847	SYN_RECV
tcp	0	0	10.0.1.101:80	219.102.144.137:1067	SYN_RECV
tcp	0	0	10.0.1.101:80	209.220.34.223:1285	SYN_RECV
tcp	0	0	10.0.1.101:80	141.176.226.253:1153	SYN_RECV
tcp	0	0	10.0.1.101:80	91.228.46.197:1123	SYN_RECV
tcp	0	0	10.0.1.101:80	132.176.243.149:1359	SYN_RECV
tcp	0	0	10.0.1.101:80	95.45.17.44:1932	SYN_RECV
tcp	0	0	10.0.1.101:80	91.189.232.88:1381	SYN_RECV
tcp	0	0	10.0.1.101:80	54.41.120.207:1631	SYN_RECV
tcp	0	0	10.0.1.101:80	214.71.194.61:1385	SYN_RECV
tcp	0	0	10.0.1.101:80	141.182.40.213:1758	SYN_RECV
tcp	0	0	10.0.1.101:80	161.138.31.171:1936	SYN_RECV
tcp	0	0	10.0.1.101:80	203.37.9.238:1672	SYN_RECV
tcp	0	0	10.0.1.101:80	107.24.133.179:1861	SYN_RECV
tcp	0	0	10.0.1.101:80	61.88.199.104:1930	SYN_RECV
tcp	0	0	10.0.1.101:80	214.155.173.23:1818	SYN_RECV
tcp	0	0	10.0.1.101:80	149.174.192.205:1799	SYN_RECV
tcp	0	0	10.0.1.101:80	121.231.41.223:1582	SYN_RECV
tcp	0	0	10.0.1.101:80	164.228.52.70:1176	SYN_RECV
tcp	0	0	10.0.1.101:80	148.150.172.63:1896	SYN_RECV
tcp	0	0	10.0.1.101:80	131.68.139.86:1340	SYN_RECV

图4-23 DDOS攻击

● Syn Flooding的攻击防御

很不幸的，Syn Flooding攻击到目前为止，并没有任何有效的防御方法，因为问题根本在于：我们无法从三次握手的第一个数据包来判断其是否为攻击行为。因此，至今我们所能做的，就是“尽量降低”Syn Flooding攻击所带来的伤害，要完全阻止Syn Flooding的攻击是不可能做到的，下面列出几种可以降低伤害程度的方法。

● 单一主机上的防御：

回想一下Syn Flooding攻击的原理，其中被攻击的系统之所以会瘫痪，是因为“TCP队列被占满了”，如果我们可以想办法提高TCP队列被占满的难度，就可以确保当系统遭受到Syn Flooding攻击时，系统不会那么容易崩溃，当然正常访问系统服务的使用者也就不会被系统拒之门外。

我们可以从两方面入手来实施以上想法，其一是调整系统内核中关于TCP连接的参数，其二是启动tcp_syncookies机制来对抗Syn Flooding攻击，做法分别如下：

● 调整Linux系统内核参数

以图4-24为例,前面曾经提到过,如果服务器端等不到客户端的应答数据包①,服务器端总共会重发5次确认数据包给客户端②,而这段时间将长达3分15秒之久,因此,如果我们可以缩短这种无谓的TCP队列占用时间,那么攻击者只有在相同的时间内送出更多的SYN数据包,才有机会占满TCP队列,而另一个方法则是扩大TCP队列空间。

● net.ipv4.tcp_synack_retries:

这个参数的用途在于设置系统重发数据包次数,以CentOS 6.0以及RHEL 6为例,其默认值为5,但这里的建议值为3,因为重送次数越多,当Syn Flooding攻击发生时,所占用TCP队列的时间就会越长,经过实际测试,如果tcp_synack_retries参数值设置为3,TCP队列被占用的时间将缩短为不到50秒的时间,因此,改变这个参数将有助于加速系统清除TCP队列中不正常的信息。我们可以在/etc/sysctl.conf内加入“net.ipv4.tcp_synack_retries=3”,或是在防火墙的Shell脚本文件中加入“echo 3 >/proc/sys/net/ipv4/tcp_synack_retries”来改变这个参数值。

● net.ipv4.tcp_max_syn_backlog:

这个参数的用途在于定义TCP队列的长度,以CentOS 6.0及RHEL 6为例,其默认值为1024,建议的值为2048,如果我们同时调整以上两个参数,将有助于提高系统对Syn Flooding攻击的防御能力。

我们可以在/etc/sysctl.conf内加入“net.ipv4.tcp_max_syn_backlog=2048”,或是在防火墙的Shell脚本文件中加入“echo 2048 >/proc/sys/net/ipv4/tcp_max_syn_backlog”,即可改变这个参数的值。

● 启动tcp_syncookies机制

以上两种方式是属于比较消极的防御方式,其实在Linux内核中还内置了一个称为tcp_syncookies的机制,tcp_syncookies运用特有的算法来提供较小规模的Syn Flooding攻击防御,但如果攻击的能量很大,tcp_syncookies还是起不到完全防御的效果。

CentOS 6.0或RHEL 6版本默认不启用tcp_syncookies机制,如果要启用tcp_syncookies机制,可在/etc/sysctl.conf文件内加入net.ipv4.tcp_syncookies=1这个参数,在重新启动系统之后,就会自动启用tcp_syncookies机制;或是在防火墙的Shell脚本文件中加入“echo 1 >/proc/sys/net/

客户端

服务器端

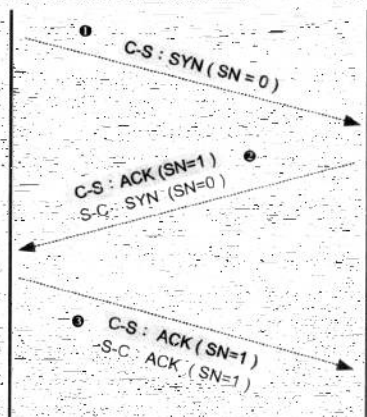


图4-24 TCP三次握手

ipv4/tcp_syncookies”，在执行Shell脚本之后，tcp_syncookies的机制就会立即启用。

● 网关上的防御：

在“单一主机上的防御”中所提到的方法都只适用于单机之上，因此，在实施Syn Flooding攻击防御时，我们必须在每台主机上逐一设置，防御的设置较为麻烦，但如果我们可以将Syn Flooding的攻击防御移至“网关”上，就可以在“网关”位置保护所有主机，不过网关式Syn Flooding攻击防御也只能降低遭受攻击时的伤害程度，无法完全抵挡Syn Flooding攻击行为，如图4-25所示。

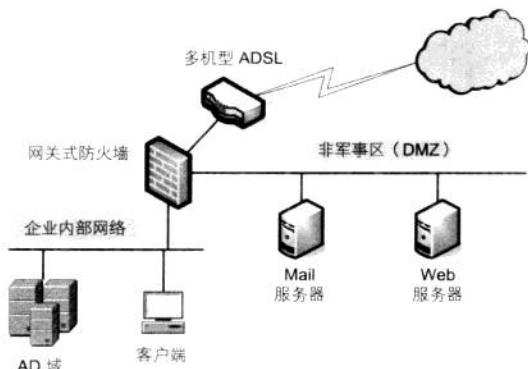


图4-25 网关式防火墙

目前，网关上的Syn Flooding防御方式有很多种，下列列举三种方法来说明其原理：

● 缩短后端服务器端上TCP队列被占用的时间：

Syn Flooding攻击的原理就是占满服务器端的TCP队列，如果我们可以把TCP队列被占用的时间缩短，就可以在某种程度上使得服务器端的TCP队列“比较不容易”被占满，因此，当服务器端遭受Syn Flooding攻击时，可以在一定程度上防止服务器端的TCP队列被占满。

以图4-26为例来说明如何在网关上缩短服务器端TCP队列被占用的时间。首先，客户端送出一个服务请求包给服务器端❶，接着服务器端应答客户端一个确认数据包❷，此时，防火墙开始计数等待客户端回应数据包的时间，如果在一段时间之内无法等到客户端的应答（防火墙所等待的时间会远小于服务器端TCP队列的超时时间），此时防火墙就会伪造客户端的IP向服务器端送出一个包含Reset标记的数据包❸，服务器端在收到这个数据包之后，就会立即中断与客户端的连接，当然，TCP队列中缓存的信息也就随之被清除掉。

以上方法确实可以缓解Syn Flooding攻击时所带来的压力，不过，当Syn的量再大一点时，服务器端依然有TCP队列被占满的可能；如果在服务器端上同时启动tcp_syncookies，将更能缓解Syn Flooding的攻击威胁。

• 使用Syn网关:

除了上述做法之外,我们还可以运用“操作系统本身所能接受的正常连接数量远大于TCP队列的数量”这个特性,来达到缓解Syn Flooding攻击时所带来的压力,以图4-27为例来解释这个原理。

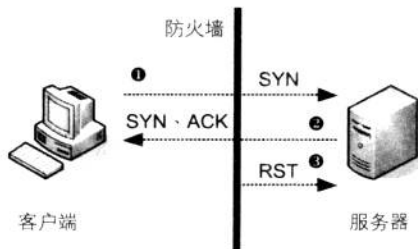


图4-26 Syn网关(一)

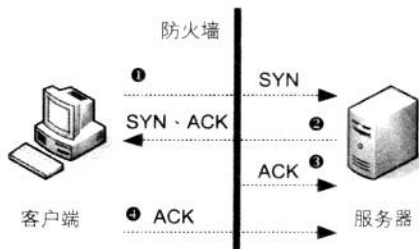


图4-27 Syn网关(二)

首先客户端发送一个服务请求包给服务器端①,服务器端在收到这个数据包之后,随即应答客户端一个确认数据包②,接着,防火墙会伪造客户端的IP,发送一个包含ACK标记的数据包给服务器端③,服务器端在接到这个数据包之后,就会结束TCP三次握手的流程,并且完成这条TCP连接的建立。当然TCP队列中的缓存信息也就跟着被清除了,如果这个SYN数据包属于攻击数据包,它就无法占用任何TCP队列。如果防火墙在一段时间之内都无法得到客户端的应答,防火墙就会伪造客户端的IP对服务器端送出包含Reset标记的数据包,强迫服务器端终止刚刚建立好的那条连接;反之,如果防火墙收到客户端应答回来的数据包④,那么防火墙将丢掉这个数据包。

以上这个方法和第一个方法存在同样的问题,也就是在SYN数据包的量很大的情况下,这个机制还是无法确保服务器端的TCP队列不会被占满。

• 使用反向代理机制:

防御Syn Flooding攻击还有一招,我们可以使用反向代理机制来担当“替死鬼”的角色,以便“完全”阻止Syn Flooding的攻击数据包。这听起来似乎非常不可思议,不过,这个机制还是同样只能有限度地防御Syn Flooding的攻击,当发生大规模的SYN攻击时,我们的“替死鬼”依然会阵亡的。

下面分析反向代理的工作原理。以图4-28为例,首先要准备一台主机,并在主机上安装反向代理这个“应用程序”,接着我们令反向代理在TCP Port 80上工作,并且在DNS服务器内将www的记录指向反向代理主机的IP,因此,任何来自因特网上的服务请求都会送到反向代理上①。反向代理可从这个请求的URL中看出客户端所要求的网页是哪一个,接着,反向代理连接到后端真正的Web服务器上②,提出网页访问请求,并把刚才客户端所要求的那个网页取回

③，在反向代理接到这个网页之后，即将这个网页传输给客户端④，如此客户端即可取得其所需的网页信息。

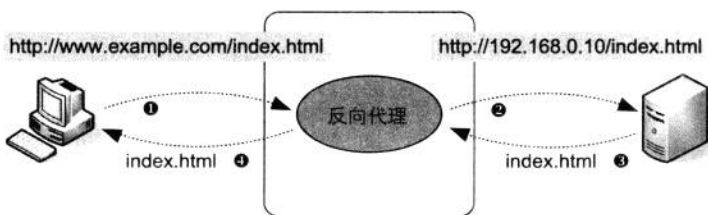


图4-28 反向代理(一)

由上述流程可以看出，如果黑客真的发动Syn Flooding的攻击行动，那么所有攻击数据包将回落在反向代理上，因此，反向代理的TCP队列一样还是会被占满的。不过别忘了，我们可以在反向代理主机上实行“单一主机的Syn Flooding防御”，如此即可把攻击所带来的伤害转嫁到反向代理上，而Linux系统本身对于Syn Flooding的攻击承受能力比起Windows要好得许多，而且反向代理本身对于安全的管控能力也很强，因此，我们还可以使用反向代理来防御其他对于IIS的攻击行为，例如对URL的攻击防御特别有效，至于反向代理的应用，我们将在第5章中加以说明，这里暂不做深入的探讨。

此外，为了加强反向代理机制的防御能力，实际应用中，我们通常会规划网络结构，并在企业的DNS服务器上，把WWW记录分别指向三个不同的反向代理(即实施DNS循环)，如图4-29所示。这个结构的好处在于我们可以无限增加反向代理的数量，企业也可以通过这样的结构，大量扩充企业网站的对外带宽，而且在维护网页时，我们只需要针对Web服务器即可，因此，并不会影响网页的管理。而且在这个结构下，如果企业网站真的遭受Syn Flooding攻击，这些攻击包会自动被分摊在各个不同的反向代理主机上，因此，黑客如果想要瘫痪企业网站，其所要发动的攻击能量就必须再提高，否则攻击行动难有效果。

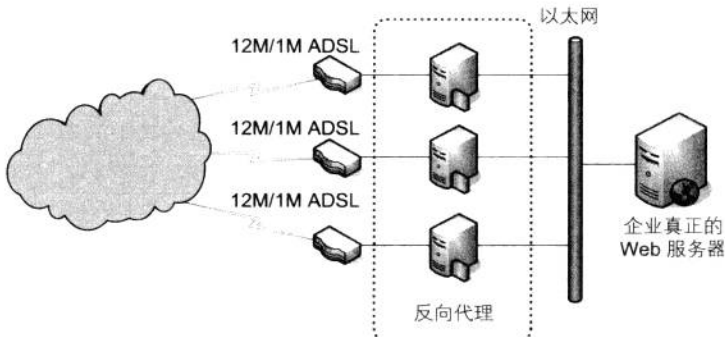


图4-29 反向代理(二)

4. URL攻击防御

URL攻击是黑客常用的手法之一,同时也是病毒及木马扩散的重要途径,而这样的攻击利用IIS本身设计上的漏洞达到入侵目的,因此,单纯封锁端口无法杜绝这样的攻击,不过,我们可以通过Netfilter的string模块来检查数据包内所承载的数据内容,以杜绝这样的攻击手段。

以图4-30为例,URL中的“/c/winnt/system32/cmd.exe”即为黑客通过URL来命令IIS执行C:\winnt\system32目录下的cmd.exe这个可执行程序,在了解URL入侵的手段之后,我们就可以运用Netfilter的string模块来匹配任何送给Web服务器TCP端口80的数据包,看看数据包内是否包含winnt或system32的字符串,如果有的话,就将数据包丢弃掉,如此即可防御URL的攻击行为。不过,需要注意以下两点,其一是在我们网站上不得存在任何有winnt或system32字符串的URL;其二是使用string模块时,千万不要匹配/winnt/system32/这样的字符串,因为“/”这个符号可能会被编码成“%5C”(即URL Encoder可以对所有非字母符号进行编码),如此string模块的匹配操作就会失效。



图4-30 内容过滤器

5. 管理病毒感染时的连接消耗

由于网络技术迅速发展,使得每台计算机都有连上因特网的机会,因此,计算机病毒的形态也就随之改变,以往只有在计算机上运行被感染的文件,计算机才会被病毒感染。但现在只要把计算机连上因特网,即使什么都不做,也有可能莫名其妙地被病毒感染,而且在感染病毒后,你的计算机就会开始搜索因特网上的其他计算机并进行感染操作,这样的感染途径可能会对企业的防火墙造成极大伤害,如果防火墙管理人没有采取适当的防范措施,防火墙就可能因此瘫痪。

或许你会觉得奇怪,这与防火墙有什么关系?别忘了,我们之前曾提过,每当客户端对因特网建立一条新的连接,防火墙就必须为这条连接建立一份连接跟踪记录,而防火墙能同时跟踪的连接数量有限。如果全部被病毒感染的连接给占满了,其他正常想要连接到因特网的使用者将无法建立连接,也就无法访问因特网上的服务。另外,除了计算机病毒感染所造成的问题之外,P2P软件的使用也可能造成同样的问题,因为P2P的工作方式与计算机病毒的感染行为其实是很类似的,就是在短时间内对因特网“疯狂”地建立新的连接。

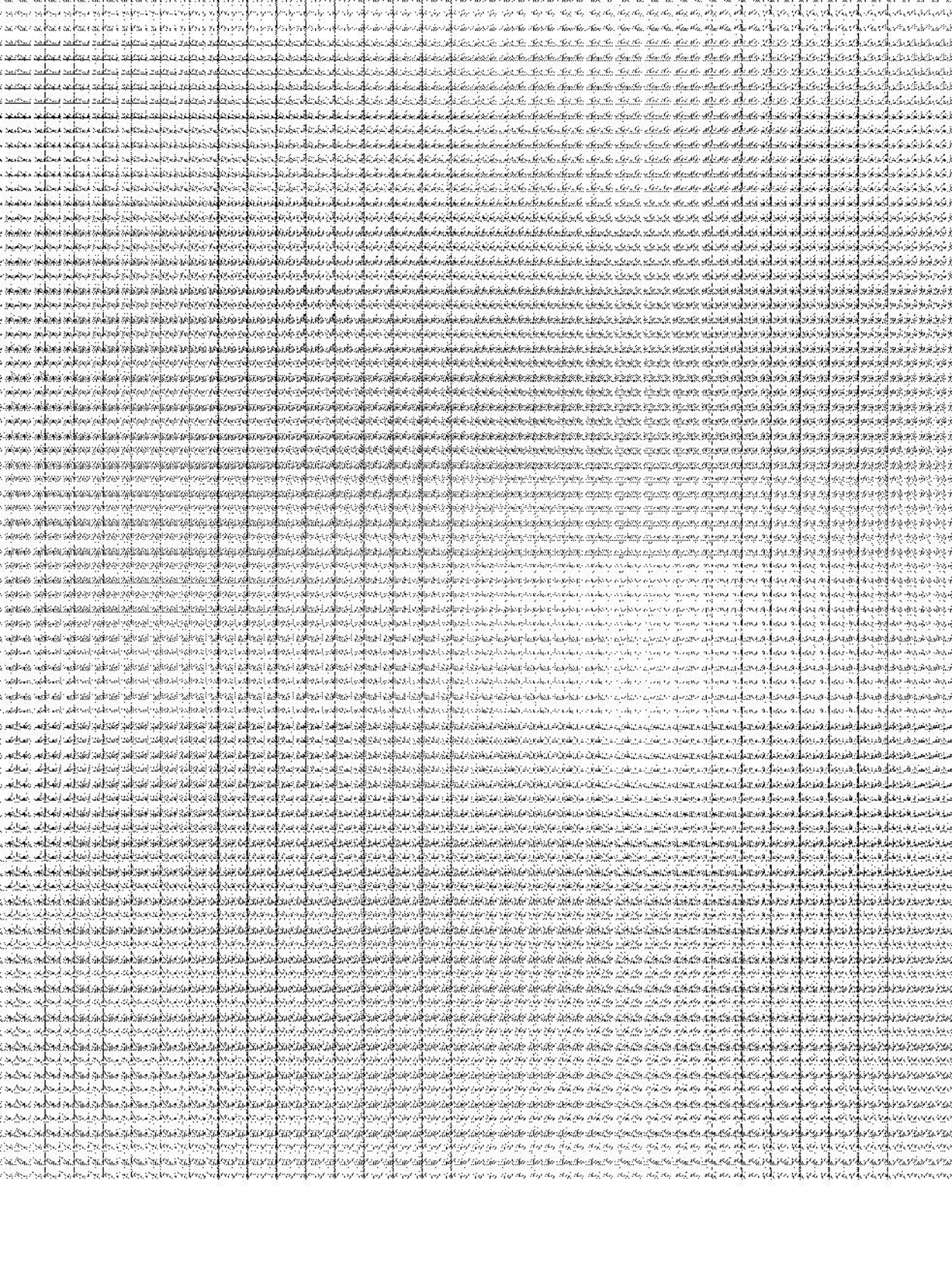
为了避免以上问题影响到防火墙的正常功能,我们可以在防火墙上做适当限制,我的想

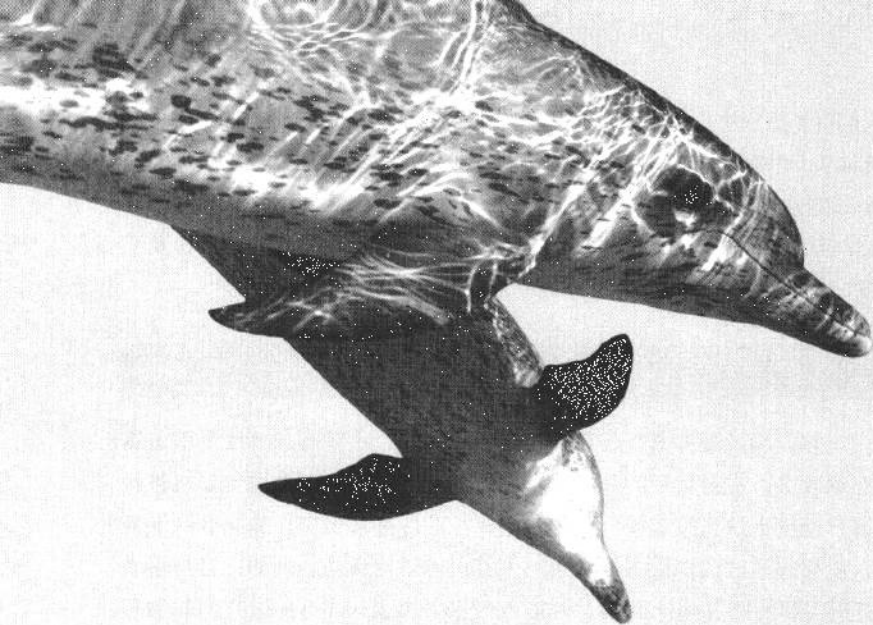
法是“如果同一个客户端在60秒钟之内，对因特网建立超过120条新的连接，那我们就认定这个客户端已中毒，或者这个客户端正在使用P2P软件”，因此就封锁这个客户端不允许其连上因特网，下列是我编写的规则：

```
iptables -A FORWARD -i eth1 -o eth0 -p all -m state --state NEW -m recent \  
--name virus --update --seconds 60 --hitcount 120 -j REJECT  
iptables -A FORWARD -i eth1 -o eth0 -p all -m state --state NEW -m recent \  
--name virus --set
```

4.5 小结

在网络速度日益加快的时代，应更重视防火墙的性能，在撰写本文时，Hinet光世代的带宽已达到100M/10M bps了，如果防火墙的规则设计不当，将会出现网络连接缓慢的问题，另外则是复杂通信协议跨越防火墙的问题，如果在防火墙上没有针对复杂通信协议进行很好地处理，那么就很可能导致必需的应用被防火墙给阻挡下来，从而无法正常使用相应的网络服务，因此你必须认真面对这个问题。不过还要再次强调，即使你是购买商用防火墙，一样需要面对相同的问题，最后则是关于网络上常见的攻击手段及防御之道，文中所要传达给你的信息是“灵活”运用你手上的积木(Netfilter的模块)，去拼出你所想要的防御方法，从而使网络变得更加安全，这才是Netfilter/Iptables设计的精髓所在。

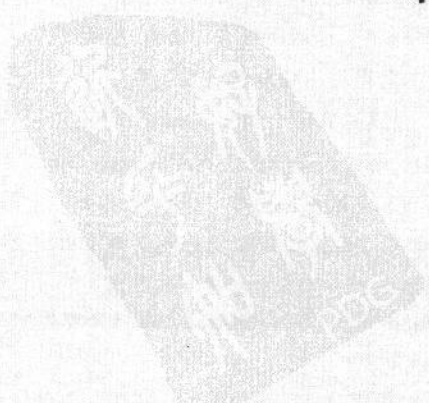




Linux

| 第5章 | 代理服务器的应用

5



一提到网络安全,人们首先想到的几乎都是“防火墙”,代理(Proxy)这个名词在网络安全领域里,似乎已渐渐被淡忘了;但你可能不知道,在早期有不少企业都使用代理来维护其企业网络的安全,而现在由于防火墙技术不断进步,且价格不再像以前那么高昂,因此,防火墙几乎已成为企业网络的安全中心。但事实上,在现在的企业网络之中,代理还是有其存在的价值及意义,当然,这些优点绝非防火墙所具有的,本章将介绍代理在企业中所扮演的重要角色。

5.1 何谓代理服务器

在没有代理的环境中,客户端及服务器端是直接连接的,因此,不管客户端还是服务器端都可能遭受到对方的恶意攻击,在有代理的环境下(如图5-1所示),客户端及服务器端被代理隔离在两侧,因此,客户端就无法直接连接到服务器端,在这样的环境中,如果客户端需要访问服务器端的资源,就必须通过代理服务器的帮忙才有可能访问成功。例如:客户端通过eth0告诉代理服务器其所需要的网页是什么①,这时代理服务器由eth1向真正的Web服务器提出客户端刚才所提出的请求②,接着服务器端应答代理所提出的请求③,当代理服务器收到这些数据后,再将这些数据交给客户端④,如此即完成客户端的一个服务请求操作。这个例子引起我们对一个问题的思考,如果代理服务器不支持http通信协议,那么客户端能否看到网页?答案当然是不行的。例如:我们在两个只懂中文的人之间,找了一个美国人来当传话筒,结果会怎么样?因此,使用代理之前请记住,只有代理服务器能够支持的通信协议才能在代理服务器中正常工作。

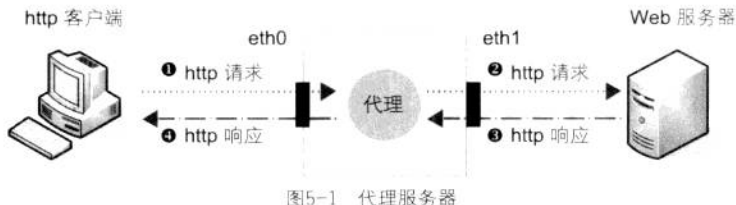


图5-1 代理服务器

代理服务器的另一个重要特点是,如果要使Linux主机担当路由器角色,就一定要启用ip_forward的功能,否则IP数据包就无法从一个接口转发到另一个接口,但是启用ip_forward功能后,我们需要面对安全方面的问题,因为黑客可以利用Linux路由器将攻击数据包转发到真正的服务器端上。但请仔细分析图5-1的环境,我们是否需要启用代理服务器上的ip_forward功能?其实在代理服务器的环境并不需要启用ip_forward机制,因为代理是一个守护进程(Daemon Process),而这个守护进程同时工作在eth0及eth1两张网卡上,当客户端从eth0接口送入请求时,代理这个守护进程再由eth1接口对Web服务器提出服务请求,因此,只要代理服务器(Proxy Daemon Process)本身不存在安全漏洞,那么部署在代理服务器后方的客户端或服务器端

基本上都会拥有一个很安全的运行环境。

5.2 代理服务器支持的通信协议

以本章介绍的Squid代理为例, 其所能支持的通信协议就只有HTTP、HTTPS、FTP、GOPHER及WAIS这五种, 因为时代的变迁, GOPHER及WAIS目前已经很少有人使用, 较为常用的现在只剩下HTTP、HTTPS及FTP这三种通信协议了, 因此, 千万要记住刚才的特别提示: “只有代理服务器能够提供支持的通信协议才能在代理服务器中正常工作”, 所以在规划代理服务器时, 一定要将通信协议考虑进来。

5.3 代理服务器的分类

代理服务器的主要功能就是“代理”, 我们可以通过代理来达到“屏蔽”目的, 如此即可为客户端或服务器端提供一个安全的服务运行环境, 另外, 代理服务器在应用上因为保护“对象”的不同, 基本上可分为缓存代理(Cache Proxy)及反向代理两种, 下面来分析什么是缓存代理及反向代理(Reverse Proxy)。

5.3.1 何谓缓存代理

以图5-2为例来说明缓存代理存在的必要性。我们假设企业内有10台Windows XP的客户端, 如果这10台客户端都要连接到因特网来更新软件, 那么, 相同的一份更新数据将会重复下载10次, 如此不但浪费企业的外网带宽, 又很低效, 但这样的问题在使用代理服务器的环境下将不会再发生。

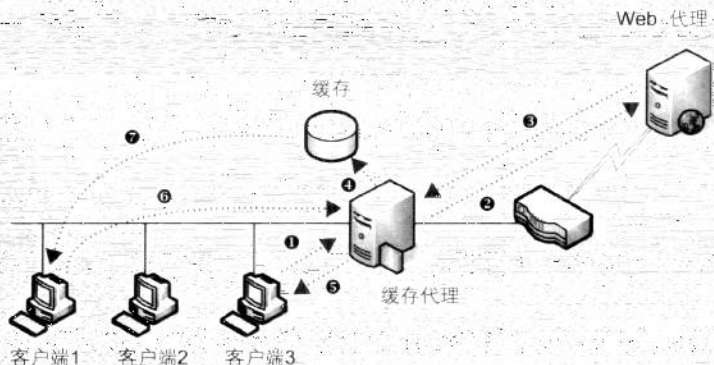


图5-2 缓存代理(-)

首先,我们假设代理服务器的缓存中空无一物,当客户端3对缓存代理提出服务请求时①,由于缓存代理的缓存中内什么都没有,因此,代理服务器会对Web服务器提出刚才客户端3所提出的请求②,接着Web服务器把代理服务器所需的网页内容传输给代理服务器③,在代理服务器收到这个网页数据后,随即将网页数据保存一份到代理服务器上的缓存空间内④,并将取得的网页数据传输一份给客户端3⑤,如此客户端3即可得到其所需的网页内容;假设这时客户端1也对代理服务器提出与客户端3相同的请求内容⑥,由于代理服务器刚才已经把网页内容保存一份到缓存中,代理服务器就会从缓存内将客户端1所需要的数据直接传给客户端1⑦,因此,代理服务器就无须再到Web服务器上重新取得网页数据,这样不但可以使客户端更快地浏览网页,也可以有效节省企业的外网带宽。

但在图5-2的架构中,由于代理服务器部署在网关位置上,而代理服务器本身所支持的通信协议并不多,因此,在这个结构下用户除了HTTP、HTTPS及FTP之外,就无法再使用因特网上的其他服务了。或许你会问,真会有人把代理服务器放在网关位置上吗?其实在早期防火墙机制尚不完善的年代里,代理服务器大概就是如此这般地应用,不过,近年来因为防火墙机制日趋完善,其价格不断下降,因此,代理服务器的应用架构大多已改变成如图5-3所示的形式。也就是说,使用防火墙来保护企业内部网络,而代理就只拿来作为缓存使用。

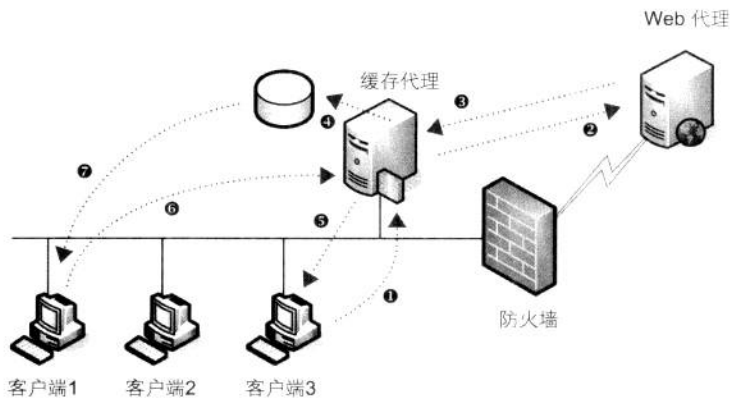


图5-3 缓存代理(二)

5.3.2 何谓反向代理

代理服务器除了可以用来保护客户端之外(如缓存代理),其实也可以用来保护服务器端,这个机制称为反向代理。下面以图5-4为例来说明反向代理的工作原理,首先把企业真正的Web服务器部署在企业的内部网络中,并且在DMZ的区段中部署反向代理,在DNS服务器中将WWW记录指向反向代理所在的IP,如此一来,任何要浏览这家企业网站的服务请求都会送

给反向代理①。接着，反向代理会帮客户端到真正的Web服务器上②取回其所需要的网页内容③，最后反向代理再把取得的网页数据传输给客户端④，如此客户端即可得到其所需的网页内容。此外，反向代理也可以拥有缓存能力，客户端所需的网页在反向代理的缓存内已经存在时，反向代理就会直接从缓存中供应给客户端，而无须再对Web服务器提出服务请求，所以反向代理也有降低Web服务器负载的功能。

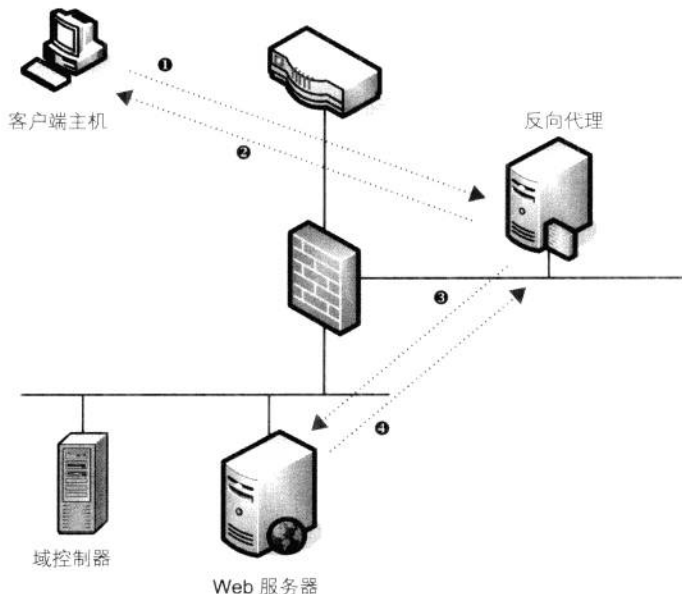


图5-4 反向代理

从以上流程可以发现，客户端并不会知道自己所访问的是一台反向代理，而且就算客户端知道自己是通过反向代理的代理才能访问到Web服务器，客户端还是无法得知真正Web服务器所在的地址，因此，黑客并无法直接攻击到企业真正的Web服务器，如果你所在的企业使用MS IIS作为Web服务器，建议在IIS之前部署一台反向代理，如此可极大地提升IIS Web服务器的可靠性。

5.4 代理服务器的硬件要求

相对于Netfilter而言，代理服务器的硬件可就马虎不得了，因为代理服务器必须处理客户端所下载回来的网页数据，而且代理服务器还可能被用来过滤客户端所访问的URL，再加上代理服务器的网络流量通常非常大，所以硬件要求会苛刻一点，下面分析代理服务器的各

项硬件要求。

● 处理器(CPU)

虽然代理服务器需要执行一些运算,但代理服务器性能的好坏与CPU并没有太大关系,但也无须特意去寻找太旧的CPU,以一个拥有500人的企业为例,如果只是执行代理任务,那么使用P4 2.0就足以满足要求了。

● 内存(Memory)

由于代理服务器的缓存管理机制是将所有缓存放置于硬盘之中,而Squid代理的设计者考虑到硬盘访问速度不够快,因此,将硬盘中“点击率”较高的缓存复制一份到内存中,如此当客户端需要缓存数据时,即可从内存中快速地提供给客户端。内存的大小严重影响代理服务器性能,至于内存该装多少?并没有标准答案,通常的建议是“越大越好”。

● 硬盘(Hard Disk)

在前面我们了解了Squid代理缓存的处理方式,虽然缓存会额外被复制一份到内存中,但如果客户端数量庞大或是内存空间不够大时,那么当客户端需要这些缓存数据时,仍需要到硬盘中存取,可见硬盘的速度也是影响代理服务器性能的指标之一。

从硬盘本身的性能来讲,SCSI介质的硬盘绝对是首选,如果是要应用在学校(大专院校),建议将SCSI组合成RAID 5或RAID 10的机制;如果只是运用于一般中小企业,如500人的企业,那么使用单个SCSI硬盘就足够了。

我曾服务于某一个拥有200多个客户端的公司,当时在代理服务器上所安装的只是一个20G的IDE硬盘,感觉用起来一点也不慢;如果有预算上的考虑,不妨先试着用IDE或SATA的硬盘跑跑看,如果真的不符合需求,再选择SCSI的硬盘也不迟。

● 网卡

因为代理服务器的网络流量很大,因此,选用一张稳定性高且性能好的网卡绝对是必要的,虽然市场上可以选用的网卡品牌众多,但建议选用3COM的网卡,如果没有3COM,至少也要选用Intel的网卡,至于其他品牌的网卡产品,就不太建议使用。

5.5 安装Squid代理

在了解了Proxy的基本原理、架构及硬件要求之后,接下来就可以把Squid代理安装起来,Squid代理的套件几乎内置在所有品牌的Linux系统之中。比如,我比较喜欢的RedHat Enterprise Linux或是CentOS中都内置了Squid代理,因此,只需要通过yum或rpm软件管理系

统，就可以轻松安装Squid代理，其命令如下：

```
yum install squid-Version.i386.rpm
或
rpm -ivh squid-Version.i386.rpm
```

安装完毕后，可以通过修改/etc/squid/squid.conf配置文件，按要求配置Squid代理。

5.6 使用Squid构建缓存代理

本节将使用Squid来构建缓存代理，以图5-5为例来说明缓存代理的部署方式。首先假设代理服务器的外连接口是eth0，其IP设置为10.0.1.200，此外，代理服务器对内的接口为eth1，IP设置为192.168.0.1，而企业内部的网段假设为192.168.0.0/24。由于Squid代理的参数众多，因此将其配置参数分为两阶段来讨论，分别为“缓存代理的基本配置”及“缓存代理的高级配置”。

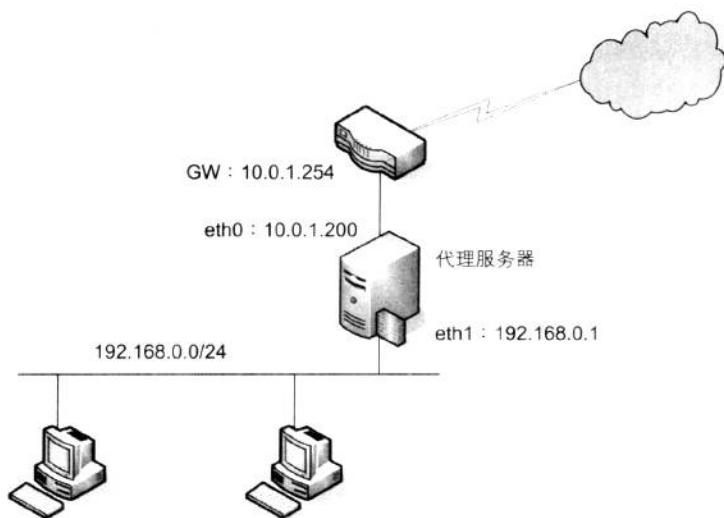


图5-5 缓存代理示例

5.6.1 缓存代理的基本配置

如果只是要让缓存代理运行起来，其实并不需要什么复杂的配置，只需将服务的端口号、缓存空间以及简单的连接访问控制配置好就可以了，关于这些参数的配置分别如下：

- 设置Squid代理的服务器端口号：

Squid代理默认所使用的端口号为TCP Port 3128，如果你不习惯使用这个端口号，那么可以使用“http_port 3128”参数来更改Squid代理的服务器端口号，例如：

```
http_port 8080
```

- 设置缓存磁盘的大小：

cache_dir为Squid存储缓存的参数定义，不过，缓存磁盘参数在设置上较为复杂，下面讨论Squid缓存磁盘的结构及参数的设置。

- 缓存磁盘的结构

Squid代理的缓存存储结构如图5-6所示，在/Cache/Root的multiport下是L1的目录，在L1的目录下可以看见其他目录，称为L2的目录，最后在L2目录下又看见许多文件，这就是Squid代理所存储的缓存。

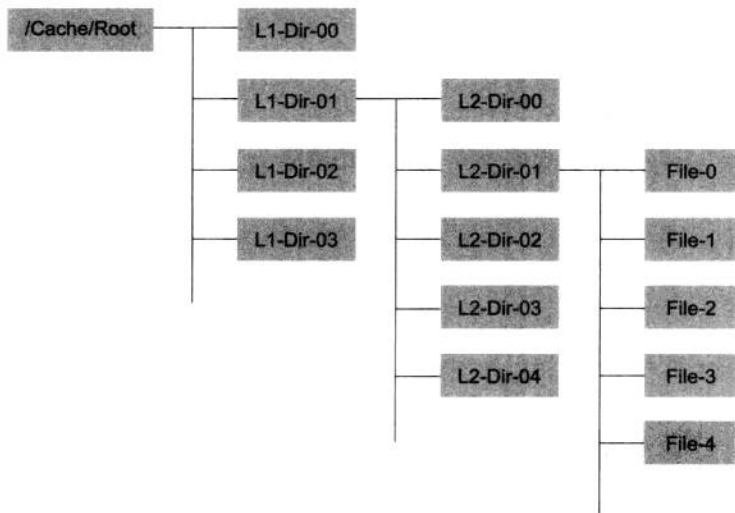


图5-6 缓存代理结构

- 计算cache_dir参数

cache_dir的参数共分5个字段，其含义分别如下：

```
cache_dir ufs /var/spool/squid 100 16 256
```

- ufs：

设置squid存储缓存的方式，可分为ufs、aufs、diskd等，单从性能分析，ufs会是比较好的选择，因为与其他方式相比，ufs在各种不同环境下的性能及稳定性都较好。

- /var/spool/squid:

缓存文件所存储的路径，squid的默认值是/var/spool/squid，不过，这并不是一个很好的建议值，如果可以的话，尽量让缓存存储于独立的分区(Partition)之上，以便提高Squid代理读取缓存的速度。

此外，根据Squid作者的建议，单个缓存存储空间最好不要超过3G的空间，若所需的缓存空间大于3G，可在squid.conf的配置文件中同时加入多行cache_dir参数，然后指向于不同的分区空间。

- 100: 设置单一缓存空间的存储上限，其单位为MB。
- 16 256: 16、256为L1及L2的目录数量，这两个数值只要随意设置就可以让Squid代理运行起来，但如果要让Squid代理的性能更好一些，那么这两个参数最好是经过计算而得到，其计算公式如下：

$$L1 * L2 = \text{缓存总容量 (KB)} / \text{第二层目录下文件总数} / \text{每个文件的大小 (KB)}$$

假设缓存总容量为9GB，因为9GB已经超过Squid组织的建议值，因此，我们将之分割为3个3GB的分区，接着，由于Squid组织建议“第二层目录下文件总数”最好不要超过256个，因此，我们就设置为256，另外，假设缓存中文件的平均大小为20K，这20K是一个经验值，你也可以假设为30K、40K都可以，这里之所以设置为20K，是因为一般网页的html文件、图片等其平均值约为20K。有了这些数值之后，就可以套用公式进行计算：

```
L1 * L2 = cache 总容量 (KB) / 第二层目录下文件总数 / 每个文件的大小 (KB)
L1 * L2 = 3000000KB / 256 / 20KB
L1 * L2 = 585.9375
16 * 37 = 592 略大于 586
```

从以上计算公式可以发现，其中L1*L2=585，而L1与L2并没有标准答案，不过根据Squid的建议是“尽量让这两个数值”较为接近，因此，这里选定了16*37=592略大于586，最后将cache_dir的参数设置如下：

```
cache_dir ufs /cache-1 3000 16 37
cache_dir ufs /cache-2 3000 16 37
cache_dir ufs /cache-3 3000 16 37
```

- 设置高速缓存的大小：

就如在前面所提过的：“Squid代理会将所有缓存数据保存于缓存磁盘中，但是会将使用率较高的缓存复制到内存中”，因此设置一个适当的高速缓存空间，将有助于提升Squid代理的性能。高速缓存的大小可以通过“cache_mem”参数来设置，不过这个参数默认情况下在squid.conf文件中并不存在，我们只要手动加上即可，至于值的大小通常会设置为实体内存的1/2大小，但如果代理服务器主机是专门用来执行缓存代理的任务，可以试着再将值调高到实体内存的2/3，而cache_mem的默认值为256MB。

```
cache_mem 2048 MB
```

● 连接访问控制：

为了避免恶意的使用者擅自通过代理服务器到因特网上从事非法行为，Squid代理服务器在默认情况下，只允许localhost及localnet的主机通过代理服务器连接至因特网，不过这里的localhost及localnet指的是Squid代理配置文件中的变量，而非主机名称的localhost。请特别注意，Squid代理的配置文件中默认localhost及localnet两个变量名称的含义如表5-1所示。

表5-1 变量名称与变量内容对照表

变量名称	IP范围(来源端IP)
localhost	127.0.0.1/32 ::1/128
localnet	10.0.0.0/8 172.16.0.0/12 192.168.0.0/16 fc00::/7 fe80::/10

接着以图5-7为例来说明Squid的连接管理机制，这里分成两种情况，如果浏览器与Squid代理位于同一台主机上，可以将浏览器内的代理服务器指向本机(127.0.0.1)，这种情况下，浏览器可以正常通过Squid代理连接因特网，另外如果浏览器与Squid代理分属于不同主机(如图中的客户端A及客户端B)，而且客户端的IP位于localnet所指的范围内，那么客户端可以正常地通过Squid代理连接因特网，否则你将在浏览器上看到“Access Deny”错误信息。

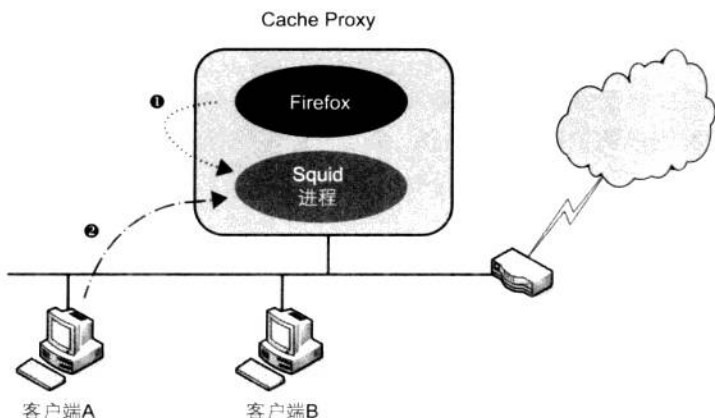


图5-7 Squid代理的连接访问控制

可以从squid.conf的配置文件中看到这些默认值，如下是squid.conf文件内的部分默认值，其中第5行~第8行定义localhost变量的值，第13行~第17行为定义localnet变量的值，第57行及第58行是设置localhost及localnet所指定的主机可以通过Squid代理连接因特网，但如果客户端的IP不在localhost及localnet范围内，就会因符合第61行的条件而被Squid代理所拒绝，其中all变量代表所有IP的意思。

```
5 acl localhost src 127.0.0.1/32
6 acl localhost src ::1/128
7 acl to_localhost dst 127.0.0.0/8 0.0.0.0/32
8 acl to_localhost dst ::1/128
13 acl localnet src 10.0.0.0/8
14 acl localnet src 172.16.0.0/12
15 acl localnet src 192.168.0.0/16
16 acl localnet src fc00::/7
17 acl localnet src fe80::/10
57 http_access allow localnet
58 http_access allow localhost
61 http_access deny all
```

以上内容看来并不好理解，不过这里还是要说明一下Squid.conf内容的匹配原则为“自上而下匹配，若条件符合时则成立不再向下匹配”。例如，客户端192.168.122.10主机要通过Squid代理连接到因特网时，Squid代理会将其IP拿到配置文件中进行匹配，在第57行中因为192.168.122.10是属于localnet 192.168.0.0/16区段的IP，因此这个访问操作在第57行就被允许而终止匹配的操作，如果客户端的IP不属于localhost及localnet区段的IP，这个匹配操作将会

继续向下匹配而符合第61行的规则，最后被Squid代理所拒绝。

了解以上内容后再来思考一个问题，如果IP网段为168.95.0.0/24，我们该如何设置才能让客户端主机正常通过代理服务器连接上因特网呢？首先插入第18行来声明来自168.95.0.0/24网段的主机为internal，接着再在第59行放行来自internal的主机。

```
16 acl localnet src fc00::7  
17 acl localnet src fe80::110  
18 acl internal src 168.95.0.0/24  
57 http_access allow localnet  
58 http_access allow localhost  
61 http_access deny all
```

从RHEL 6.0及CentOS 6.0开始，Squid代理的默认值已经能够满足绝大多数应用的需要，因此需要修改的内容自然就变得很少，以下为这个示例中所修改过的参数。

```
64 http_port 8080  
71 cache_dir ufs /var/spool/squid 3000 16 37
```

在完成所有设置后，就可以启动Squid缓存代理服务器，Squid代理的控制方式如下：

```
service squid {start | stop | restart | reload | status | ...}
```

接着可以使用命令“service squid start”启动Squid缓存代理服务器，并在启动完成后到cache_dir参数的定义路径下，看是否创建了大量EI目录。

5.6.2 缓存代理客户端的配置

在配置完缓存代理服务器的后，接着来分析客户端浏览器上的配置，由于浏览器种类繁多，此处仅以较常用的Internet Explorer为例进行说明，在IE里，我们可以选择“工具”|“Internet选项”，打开如图5-8所示的界面，并选择“连接”|“局域网设置”进而打开“局域网设置”界面，接着，在局域网设置界面，选中“为LAN使用代理服务器”，最后将代理服务器的IP及其端口号填入即可。

完成以上设置后，IE就可以通过Squid缓存代理来连接因特网了，或许你会问：“我怎么知道我的IE是不是通过代理服务器来连上因特网的”，这个问题很容易解决，如果在代理服务器停止之后，还能够正常浏览到网页，那么可以肯定，“你的IE设置一定不正确”。

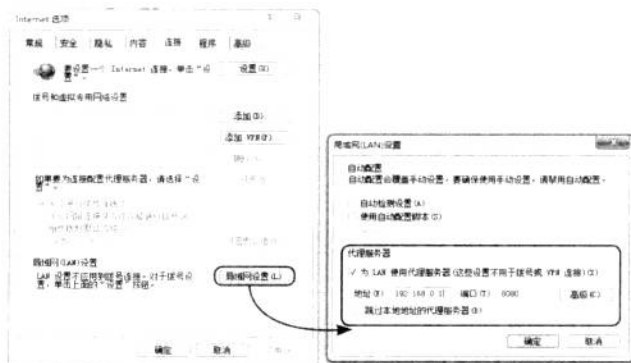


图5-8 缓存代理客户端的配置

5.6.3 缓存代理的高级配置

前面的配置只是让Squid代理能够运行而已，如果要让代理服务的工作更符合需求，绝对有必要熟悉以下参数。

1. 缓存磁盘的空间管理

不管缓存磁盘的空间有多大，也总有用完的时候，如果缓存代理的缓存空间用尽，缓存代理就会将较旧的缓存清除掉，并将新的缓存保存下来，如此反复的操作将会降低缓存代理的性能，我们可以通过以下两个参数来解决这个问题：

```
cache_swap_high 95
cache_swap_low 80
```

从图5-9中可以了解cache_swap_high及cache_swap_low两个参数的关系，只要缓存代理服务上的缓存大小累积到cache_swap_high的大小时，Squid就会开始大量删除旧缓存，直到缓存大小等于cache_swap_low时，Squid才会停止删除旧的缓存，如此一来，就可以避免缓存空间用尽后所造成的性能低下问题。

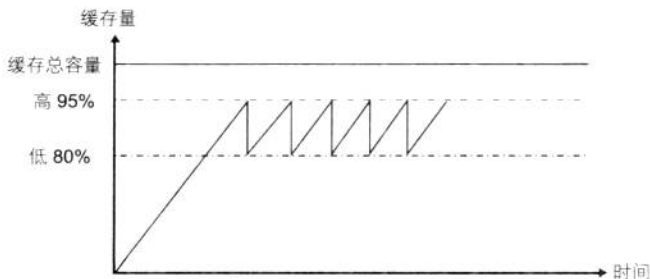


图5-9 缓存磁盘的空间管理

2. 缓存数据大小上限

默认情况下, Squid代理只会缓存小于4MB的文件, 但在目前因特网的应用中, 随便一个文件都会超过4MB。因此, 这个默认值并非十分恰当, 这里给出的建议值是40M, 这会是一个较好的选择, 特别是企业内有大量的Windows系统在进行系统更新时, 就可以明显体验到缓存代理所带来的好处, 参数使用如下:

```
maximum_object_size 40960 KB
```

3. 访问控制列表的定义

在Squid的连接控制中, 我们可将使用者的访问方式分为几种不同的类型, 并针对不同类型的访问行为加以限制, 比如, 我们以前使用过的例子:

```
acl uuu src 192.168.0.0/24  
http_access allow-uuu
```

其中acl就是用于“分类”的定义, 而uuu就是用于某一种分类的“变量名称”, src则为分类的“方式”, 其所指的是Source IP的意思, 最后的192.168.0.0/24则是条件, 因此, 这一行的意思是: “我们要定义一个变量, 名称为uuu, 而uuu这个变量值的内容来自于192.168.0.0/24这个网段的主机”, 定义好名称之后, 就可以结合http_access的命令, 来针对我们所定义的变量名进行“允许”或“拒绝”的操作, 接下来介绍Squid代理分类的相关定义方式。

● src: 根据“来源”端IP进行分类

src是指Source IP, 我们可以使用src对不同的来源端IP进行分类, 例如①把来自192.168.0.0/24这个网段的主机归类为internal的类别, 又或者②将192.168.2.10这个IP归类于bad_client的类别。

```
acl internal src 192.168.0.0/24 ①  
acl bad_client 192.168.2.10/255.255.255.255 ②
```

● dst: 根据“目的”端IP进行分类

dst指Destination IP, 我们可以使用dst对不同目的端IP进行分类。例如, ①可将192.168.0.0/24这个网段主机提出服务请求的客户端归类为crack类别, 又或者②可将216.163.137.3这台主机提出服务请求的客户端归类为playboy类别。

```
acl crack dst 192.168.0.0/24 ①  
acl playboy dst 216.163.137.3/255.255.255.255 ②
```


- **srcdomain**: 根据“来源”端的域名(Domain Name)进行分类

srcdomain是Source Domain的意思, 我们可以根据来源端的域名进行分类。例如, ❶将来自**example.com**这个域下的所有主机归类为**internal**的类别, 但使用**srcdomain**时请特别注意, 如果来源端主机所在的IP并没有设置DNS的反向名称解析, **srcdomain**将无法执行分类操作。

```
acl internal srcdomain .example.com ❶
```

- **dstdomain**: 根据“目的”端的域名进行分类

dstdomain是Destination Domain的意思, 我们可以针对使用者所要访问的域进行分类。例如, ❶可以将要去访问**playboy.com**这个域中主机的客户端归类为**playboy**的类别。

```
acl playboy dstdomain .playboy.com ❶
```

- **srcdom_regex**: 使用“正则表达式”匹配来源端域名

我们可以使用“正则表达式”对来源端的域名进行分类。例如, ❶可以使用`*\example\com`的条件, 把所有来自于**example.com**域中的主机归类为**internal**类别, 其中`-i`的参数是指不区分大小写的意思。

```
acl internal srcdom_regex -i *\example\com ❶
```

- **dstdom_regex**: 使用“正则表达式”匹配目的端域名

也可以使用“正则表达式”对目的端域名进行分类。例如, ❶可以使用`*\playboy\com`的条件, 把所有连接到**playboy.com**域中主机的客户端归类为**playboy**的类别。

```
acl playboy dstdom_regex -i *\playboy\com ❶
```

- **urlpath_regex**: 使用“正则表达式”匹配URL不含域名的部分

可以使用“正则表达式”对URL不含域名的部分(即**path**的部分)进行分类操作。例如, ❶可以使用`*\gif$`的条件分类出所有要访问**gif**图片的请求操作, 并将之归类为**photo_gif**的类别。

```
acl photo_gif urlpath_regex -i *\gif$ ❶
```

- **url_regex**: 使用正则表达式匹配完整的URL

可以使用正则表达式来对完整的URL进行分类。例如, ❶可以使用`*\pchome\com\`

gx\./.*.jpg条件, 将URL中含有jpg文件的归类为pchome_photo_jpg类别。

```
acl pchome_photo_jpg url_regex -i .*\.pchome\.com\.gx\./.*.jpg$ ❶
```

- **time:** 根据“时间”来匹配访问的合法性

我们可以对访问“时间”进行分类。例如, ❶可将在12:00~13:00这段时间内进行网页访问的操作归类为none_work类别。

```
acl none_work time 12:00-13:00 ❶
```

另外, 也可以包含“星期”的匹配, 其中星期部分的表示方式分别如下。

表示式	星期
S	Sunday(星期天)
M	Monday(星期一)
T	Tuesday(星期二)
W	Wednesday(星期三)
H	Thursday(星期四)
F	Friday(星期五)
A	Saturday(星期六)

例如, ❷其含义是星期一~星期五早上的9点~12点, 以及下午的13点~18点。

```
acl on_work time MTWTF 9:00-12:00 MTWTF 13:00-18:00 ❷
```

- **maxconn:** 根据“连接数”来匹配访问的合法性

可以使用maxconn❶来限制相同一个客户端同时最多可以对因特网拥有多少条连接。

```
acl max_conn maxconn 10 ❶
```

- **port:** 根据“端口号”来匹配访问的合法性

我们可以使用端口来限制一个客户端只能通过Squid代理访问哪些端口。如❶定义web_port变量为Port 80。

```
acl web_port port 80 ❶
```

- 设置“不需要”缓存的数据

某些URL可能会包含一些动态数据, 例如: 股票的即时行情数据, 而这些即时性的网

页数据如果真被缓存起来,可能造成客户端一直看到旧的网页数据,为了解决这个问题,我们可以通过缓存命令进行筛选,例如:URL中不含域名的部分包含cgi-bin及?字符串或字符时,就不缓存这个网页。

```
acl QUERY urlpath_regex cgi-bin \?
cache deny QUERY
```

5.6.4 缓存代理连接访问控制

第1章曾经介绍过,防火墙可以分为“数据包过滤防火墙”及“应用层防火墙”,其实Squid代理就是一种简单的应用层防火墙,而其所能过滤的范围仅局限于使用者所送出的URL部分,至于网页内容的过滤操作Squid代理就无法完成了。下面列举几个实例来说明Squid代理的连接访问控制。

示例5.1 不允许客户端连接到playboy网站的任何一个网页

```
acl playboy dstdomain .playboy.com      # 根据dstdomain进行分类
http_access deny playboy                # 凡是符合playboy者全部不允许
acl internal src 192.168.0.0/24          # 假设企业内部网络为192.168.0.0/24
http_access allow internal               # 允许企业内部通过代理连上因特网
http_access allow localhost              # 允许符合localhost者连上因特网
http_access deny all                     # 不允许任何主机连上因特网
```

示例5.2 不允许客户端连接到无名小站看视频

```
acl wretch_video urlpath_regex -i \/video\/*func=.*vid=[0-9]*$
# 根据urlpath_regex进行匹配
http_access deny wretch_video            # 凡是符合 wretch_video者全部不允许
acl internal src 192.168.0.0/24          # 假设企业内部网络为192.168.0.0/24
http_access allow internal               # 允许企业内部通过代理连上因特网
http_access allow localhost              # 允许符合localhost者连上因特网
```

示例5.3 不允许客户端连接到pchome的“相册”及“贴图”区

```
acl pchome_photo url_regex -i ^http://photo\.pchome\.com
^http://ejoyimg\.pchome\.com            # 根据url_regex进行匹配
http_access deny pchome_photo            # 凡是符合pchome_photo者全部不允许
acl internal src 192.168.0.0/24          # 假设企业内部网络为192.168.0.0/24
http_access allow internal               # 允许企业内部通过代理连上因特网
http_access allow localhost              # 允许符合localhost者连上因特网
```

示例5.4 不允许客户端连接到因特网下载.exe及.vbs文件

```
acl exec_file url_regex -i .*\\.exe .*\\.vbs      # 根据url_regex进行匹配
http_access deny exec_file               # 凡是符合exec_file者全部不允许
```

```
acl internal src 192.168.0.0/24 # 假设企业内部网络为192.168.0.0/24
http_access allow internal # 允许企业内部通过代理连上因特网
http_access allow localhost # 允许符合localhost者连上因特网
http_access deny all # 不允许任何主机连上因特网
```

示例5.5 一个客户端通知最多只能到因特网上下载“3个”mp3文件

```
acl mp3_file url_regex -i .*\/.*\.mp3 # 根据url_regex进行匹配
acl conn_limit maxconn 3 # 根据maxconn限制最多3条连接
http_access deny conn_limit mp3_file # 凡是连接超过3条且为下载mp3者全部不允许
acl internal src 192.168.0.0/24 # 假设企业内部网络为192.168.0.0/24
http_access allow internal # 允许企业内部通过代理连上因特网
http_access allow localhost # 允许符合localhost者连上因特网
http_access deny all # 不允许任何主机连上因特网
```

看完以上示例后，你是否对Squid代理的连接控制有了更深入的了解呢？不过，以上设置方式实在不好管理，比如，前面列举的例子“acl pchome_photo url_regex -i ^http://photo\pchome\.com ^http://ejeokeimg\pchome\.com”包含两个条件，如果所需的条件不止两项，该如何去编写这个配置文件？还好Squid提供了一种方便的条件管理方式，这样可以把示例5.4的内容改成如下设置：

```
acl exec_file url_regex -i "/etc/squid/exec_file.txt" # 根据url_regex进行匹配
http_access deny exec_file # 凡是符合 playboy 者全部不允许
acl internal src 192.168.0.0/24 # 假设企业内部网络为192.168.0.0/24
http_access allow internal # 允许企业内部通过代理连上因特网
http_access allow localhost # 允许符合localhost者连上因特网
http_access deny all # 不允许任何主机连上因特网
```

可以把条件部分作为单独文件来管理，将来如果要加入或删除某些条件，就不再需要面对squid.conf这个庞大的文件，而这个存放条件的文件格式内容如下所示：

```
.*\/.*\.exe
.*\/.*\.vbs
```

简单来说，就是在一行中存放一个条件就可以了，此外，在修改完条件后，别忘了重新加载或重启Squid代理。

5.6.5 缓存对象的管理

缓存代理确实可以通过缓存方式，来达到节省外网带宽的目的，但也造成了缓存可能早已过期的问题。例如，客户端于AM 9:30浏览6214这支股票，假设此时的股价是40元，当然这个网页的数据就被缓存代理缓存下来了，接着到了PM 1:20时，假设客户端又连上来看这

只股票时，这时所看到的股票当然就是40元了，因为客户端所看到的就是AM 9:30时被缓存下来的旧网页。

既然提到了缓存对象，就得先从浏览器本身的缓存开始来谈起。以图5-10为例来说明浏览器如何管理其本身的缓存对象。我们假设浏览器目前并没有缓存任何对象，当浏览器第一次对Web 服务器提出网页请求后①，Web 服务器会把浏览器所要求的网页传输给浏览器②，并且通知这个网页(html或是图片等)最后一次被修改的时间(就是文件的Modify Time，在Linux系统上可通过stat命令来查看每个文件的Modify Time)，此时，浏览器会将这个对象及Modify Time的信息缓存下来，稍后如果浏览器又去浏览同一个网页，浏览器会发现本机上已经有缓存了，接着，浏览器就会对Web 服务器提出网页请求①，并告诉Web 服务器其自身已经有这个网页的缓存以及这个缓存文件的Modify Time，此时Web 服务器会先比较浏览器缓存内文件的最后修改时间是否等于Web 服务器上这些网页文件的最后修改时间，如果Web 服务器发现其上的文件在浏览器缓存之后有更改过网页的内容，Web 服务器将会把目前较新的网页传输给浏览器②；如果Web 服务器观察网页内容并没有修改过，那么Web 服务器将会传输“HTTP/1.1 304 Not Modified”的信息给浏览器②，由此告诉浏览器网页内容并没有被修改过，如此浏览器就会使用本身的缓存来呈现网页。

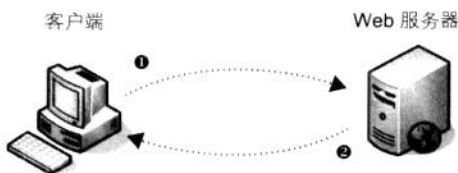


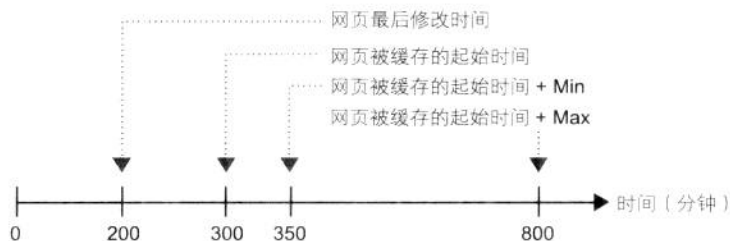
图5-10 浏览器、缓存及Web 服务器之间的关系

在了解浏览器本身缓存对象的管理方式之后，接着以图5-11为例来说明Squid代理管理缓存对象的方式。在图中我们应该分成两部分来看，第一个部分是浏览器与代理服务器，其实这一部分比较容易理解，只要把代理服务器视为Web服务器就可以了，其工作方式与图5-10中的内容完全一样，而第二部分则是此处要讨论的重点，首先点出其中的关键。我们假设代理服务器已经存在客户端现在要浏览的网页，因此浏览器要访问这个已经被代理缓存过的网页时①，代理服务器会发现这个网页其本身已经有一份缓存了，接着代理服务器会检查这一份缓存是否已过期，如果已经过期，代理服务器会重新向Web 服务器请求获得一份新的网页②；如果没有过期，代理服务器会直接从缓存中把网页传送给浏览器③，而问题就在于：“代理服务器是如何知道缓存是否已经过期了”？

后，我们可以开始讨论第一条规则的含义，从规则中可以看到，如果网页有设置生命周期，若网页的生命周期未到，则缓存是有效的；如果网页的生命周期已尽，则缓存是无效的。也就是说，如果缓存到一个设置“网页生命周期”的网页，由于会优先匹配，规则2、3、4、5对这个网页根本是没有意义的。

● 规则二：

所谓“网页被缓存的时间”是指网页从被缓存下来那个时间开始算起，直到目前时刻之间的时间差。以图5-13为例，我们假设网页被缓存下来的时间在第300分的位置，而在第二次再去浏览这个网页的时间在第810分的位置，那么我们可以发现810分大于800分(300分+Max)，在这次的网页请求中，Squid代理上的缓存已过期，所以Squid代理会到Web服务器上重新取得新的网页，而这个操作会与浏览器和Web服务器之间有着相同的处理方式，也就是说，代理服务器会将缓存的最后修改时间送给Web服务器，如果缓存文件的最后修改时间与Web服务器上网页文件的最后修改时间一样的话，Web服务器会传输“HTTP/1.1 304 Not Modified”信息给代理服务器，此时，代理服务器将重新设置网页被缓存的时间，否则Web服务器将会传输较新的网页数据给代理服务器。



PS：假设我们的设定值为 Min = 50 · Max=500

图5-13 refresh_pattern的参数含义

● 规则三：

所谓“文件最后修改时间因子”是指“(网页被缓存下来的时间)-(Web服务器上该网页文件最后的修改时间)*percent”。以图5-13为例，我们假设Web服务器上网页的最后修改时间在200分钟的位置，而该网页被缓存的时间是在第300分钟的位置，refresh_pattern percent参数为40%，因此，(300分钟-200分钟)*40%=40分钟，所以这个网页的有效期限是网页被缓存的时间+40分钟=340分钟。浏览器如果在第300~340分钟之间再次访问这个网页，其缓存会是有效的；但是在第340分钟之后的访问，其缓存则是无效的。

● 规则四：

如果网页被缓存的时间小于MIN所设置的时间，则为有效缓存。

● 规则五：

如果没有符合以上四项规定的缓存，则全部判断为无效的缓存。

从以上规则看，规则四及规则五根本就不可能用到，因为所有的情况一定在规则四之前就被决定出来，而这些规则在严格的测试下，证明了上述看法。至于Squid设置min这个参数的用意为何？我实在感到不解，如果有哪位高人了解其中的原因，还请指点一二。因此，最后的结论是：“min不管设置为何值，都不会影响到缓存的存活时间”。

接下来列举几个例子，相信你在看过后，对进一步了解refresh_pattern参数：

```
refresh_pattern -i ^http://.*\.*.mp3$ 0 100% 10080 ❶  
refresh_pattern -i ^http://.*\.*.cgi-bin.* 0 10% 60 ❷  
refresh_pattern -i ^http://.*\./download\.windowsupdate\.com/. * 0 100% 10080 ❸  
refresh_pattern -i ^http://.*\.*.jpg 0 50% 1440 ❹  
refresh_pattern -i ^http://.*\.*.gif 0 50% 1440 ❺  
refresh_pattern -i ^http://.* 0 50% 60 ❻
```

❶ 指定如果客户端下载mp3文件，那么我们就缓存一个星期，并且将percent设置为100%，如此尽量让mp3的文件可以被缓存到一星期，若超过一星期，就视为无效缓存；我在这个例子中的想法是，当一个新专辑推出之后，会通过网络下载mp3的人应该会在一个星期内下载完毕，因此设置的是一星期的时间。

❷ 如果URL中包含cgi-bin，应该是动态网页，为了避免缓存到一些即时数据，通常像是jsp、php、asp之类的网页都应该避免去缓存。因此，在这个例子中，将有效时间定得很短。

❸ 如果企业内有大量Windows的系统要执行Windows升级，最好对这些升级的文件保留长一点的有效时间，我的看法是，所有Windows的系统都会在一星期内更新完毕，因此，设定的有效时间是一星期。

❹、❺ 是针对图片来设置其有效时间，这里所设置的时间为一天。

❻ 如果所下载的文件没有被归类在❶❷❸❹❺，就设置它的有效时间为1小时。

5.6.6 Squid代理的工作日志

Unix like 操作系统中的所有服务在工作过程中都会生成一些日志文件，而这些日志是系统管理人员了解服务及工作状况的重要依据，当然，Squid代理也不例外，会生成一些日志文件，因此，系统管理人员如果要进行排错，这些日志文件将是最好的参考依据，Squid代理的日志文件分别如下：

1. /var/log/messages工作状态日志

/var/log/messages是整个Linux系统上日志的最大集中地，在Linux系统上至少75%以上的

日志都会存放在这个文件内，squid的日志也保存于此处，如果Squid代理的运行遇到问题，第一个应该想到的是去查看这个文件中的内容，例如，在重新启动Squid代理时，如果看到以下信息，就可以去参考/var/log/messages的文件内容。

```
[root@proxy-squid]# service squid restart
正在停止 squid: [失败]
正在启动 squid: [失败]
```

由于系统75%以上的日志都存放于此，因此，要从/var/log/messages文件中挑选出Squid所生成的日志，需要略施小计，我们可以使用Linux系统下的grep命令来完成这个任务，使用方法如下：

```
[root@proxy-squid]# grep 'squid' /var/log/messages | tail -10
```

其中grep 'squid' /var/log/messages是将Squid所生成的日志挑选出来，而tail -10则是只挑选出Squid最后生成的10条日志，下面即为从/var/log/messages筛选出来的Squid记录，可以从最后一条记录清楚地看到squid.conf的第2535行有设置上的错误。

```
Nov 29 17:40:13 squid squid[31979]: Squid Parent: child process 31982 exited with status 0
Nov 29 17:40:14 squid squid[32226]: Squid Parent: child process 32228 started
Nov 29 17:51:31 squid squid[32226]: Squid Parent: child process 32228 exited with status 0
Nov 29 17:51:33 squid squid[32272]: Squid Parent: child process 32274 started
Nov 29 17:57:38 squid squid[32272]: Squid Parent: child process 32274 exited with status 0
Nov 29 17:57:40 squid squid[32310]: Squid Parent: child process 32312 started
Nov 29 17:58:48 squid squid[32310]: Squid Parent: child process 32312 exited with status 0
Nov 29 17:58:50 squid squid[32354]: Squid Parent: child process 32356 started
Nov 30 09:54:39 squid squid[32354]: Squid Parent: child process 32356 exited with status 0
Nov 30 17:18:08 localhost-squid: Bungled squid.conf line 2535: http access deny pchime.photo
```

2. 客户端的浏览记录

Squid代理会将所有客户端浏览过的URL记录在/var/log/squid/access.log文件中，因此，企业如果有需要跟踪员工曾经去过哪些网站，那么access.log的记录文件将会有我们所需要的数据，其内容格式如下：

```
1142804485.796 363-192.168.0.100 TCP_MISS/200 2990 GET
http://www.google.com/images/hp2.gif DIRECT/66.102.7.147 image/gif
```

从这些记录中可以看到，客户端192.168.0.100通过Squid代理对www.google.com的images-hp2.gif图片提出访问请求，接着，Squid代理再对66.102.7.147这台Google的Web服务器提出访问请求，不过，以上这条记录的格式我并不喜欢，因为其第一个字段1142804485.796实际上记录的是时间，然后这个格式是需要经过计算之后才能阅读，因此，建议将Squid的记录

格式改为使用者自定义的格式来存储,这样便于阅读。其参数及设置方法如下:

```
logformat custom_log %>a %t%l %ru %h HTTP/%rv %mt
access_log /var/log/squid/access.log custom_log
```

其中logformat是设置记录格式的命令,接着,后面的“%>a %t%l %ru %h HTTP/%rv %mt”则是我们希望记录下来的数据,而custom_log参数就是所定义的格式。



提示

关于「%>a %t%l %ru %h HTTP/%rv %mt」所代表的意义,你可自行参阅/usr/share/doc/squid-Version/squid.conf.documented文档中第2096开头介绍的有关logformat的内容。

access_log命令则定义了记录所要存储的文件名及路径,最后的customer_log则是logformat所定义的记录格式,当以上内容都设置完毕后,别忘了重新启动Squid代理,接着就可以看到如下的记录格式:

```
192.168.0.100 30/Nov/2007:15:06:06 +0800 http://photo.pchome.com/pngfix.js
http://photo.pchome.com/aron1128/02/ HTTP/1.0 text/html
```

下面是我定义的记录格式,其内容分别如下:

- 192.168.0.100: 客户端的IP地址。
- 30/Nov/2007:15:06:06+0800: 客户端提出网页访问请求的时间。
- http://photo.pchome.com/pngfix.js: 客户端对缓存代理提出的网页访问请求内容。
- http://photo.pchome.com/aron1128/02/: 缓存代理对真实Web服务器所提出的网页访问请求内容。
- HTTP/1.0: 本次网页访问请求所使用的HTTP通信协议版本。
- text/html: 本次所需网页对象(可能是html、gif、jpg、doc、pdf等不同的文件)的mime type类型。

3. /var/log/squid/cache_store_log

这个日志文件的内容为缓存新增及删除的记录,而这些记录在一般实际的管理应用中并没有太大用处,而且其数量十分庞大,因此建议关闭这个日志功能,设置参数如下:

```
cache_store_log none
```

5.6.7 Squid代理的名称解析

在讨论这个问题之前,请先思考一个问题,如果你的浏览器是通过缓存代理连接到因特网,而我们只讨论浏览网页这件事,那么,请问客户端的计算机是否需要设置DNS?

答案是“不需要”，因为我们把浏览器的代理服务器地址设置好之后，每当使用者在网址栏中输入想要浏览的URL之后，浏览器就会把使用者所输入的URL转发给代理服务器，接着代理服务器就会将URL中的主机部分送到DNS服务器去执行名称解析的操作，所以名称解析操作是由代理服务器来执行的，因此，代理服务器将会执行大量的名称解析操作。为了加快执行代理服务器的名称解析操作，建议你在代理服务器本机上部署一个Cache Only Name Server，接着再把代理服务器本机上的DNS 服务器设为指向本机，这样对于名称解析速度的提升将有很大的帮助。

5.7 透明代理

在了解了许多关于缓存代理的技术内容之后，相信你已经具备构建及规划缓存代理的能力。现在请思考一个问题，如果你所服务的企业拥有200个客户端，要将这些客户端的代理地址指向你所设置好的代理服务器，该如何设置呢？另外客户端会不会自行删除代理的设置参数呢？如果客户端真的把代理的设置参数删除了，那我们在代理上所设置的连接控制不就全部无效了！幸好以上问题都有很好的解决方案，我们只需要通过透明(Transparent)机制，就可以轻松解决以上所有问题。

5.7.1 透明代理的工作原理

这里以图5-14为例来解释透明代理(Transparent Proxy)的工作原理。我们假设客户端192.168.0.100想通过代理服务器来访问Web 服务器10.0.0.16。另外，代理服务器对企业内部的IP地址是192.168.0.1，对因特网的IP地址是10.0.1.200。

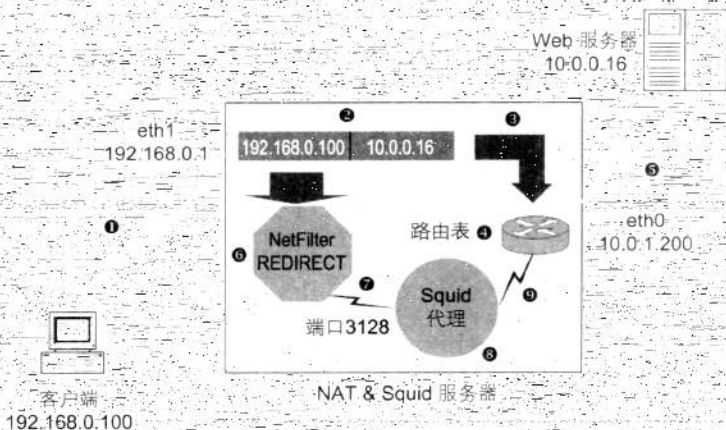


图5-14 透明代理的工作原理

假设企业是通过NAT连上因特网，这个过程是如何进行的呢？首先客户端通过DNS的查询得到Web服务器的IP地址，然后再对Web服务器提出网页请求①，因此，这个请求包②内的Source IP等于192.168.0.100，其Destination IP等于10.0.1.16，接着，这个数据包就会通过NAT机制③再加上路由表的判断④，将其发送给Web服务器⑤，如此客户端即可正常浏览因特网；但如果我们在NAT主机上启动缓存代理，然后再加上NAT机制，整个系统就变成透明代理了。

透明代理的工作原理是这样的，客户端先通过DNS的查询，然后再对Web服务器提出网页请求①，因此，这个数据包内②的Source IP等于192.168.0.100且Destination IP等于10.0.1.16。接着，可以通过NAT中的REDIRECT机制，将这个数据包内的Destination IP及Destination Port改为127.0.0.1及3128⑥，这样这个数据包就会被转发到“本机”的“端口3128”⑦，因为NAT主机上有启动缓存代理，而代理所服务的端口正是3128，所以这个数据包就等于是转发给本机的缓存代理⑧，最后由缓存代理机制连接到Web服务器，将客户端所请求的网页下载回来。

在了解透明代理的工作原理之后，再回头分析前面所担心的问题，因为这里使用NAT的REDIRECT机制，而把所有客户端送往因特网且目的端的端口是80的数据包转向到到本机的端口3128，所以根本不需要在客户端的主机上做任何的设置，以上的困扰自然就不会存在，且客户端不知道自己是通过代理服务器来浏览因特网上的网站。

5.7.2 透明代理的配置

配置透明代理总共需要两个部分，其一是Squid代理本身，其二是NAT的机制，接着来讨论如何配置透明代理。

1. 配置Squid代理

其实Squid代理的配置方式相当简单，我们只需要把以前的缓存代理配置稍微调整一下，就可让Squid代理执行透明代理任务，唯一需要改变的部分就是第64行的transparent参数，其余部分并不需要修改。

```
64 http_port 3128 transparent
```

2. 配置NAT

第2章和第3章中已经较为完整地介绍了NAT机制，唯一一种尚未介绍的DNAT机制就是REDIRECT。REDIRECT机制允许我们把TCP或UDP的数据包转向到本机(127.0.0.1)的任何一个端口。因此，我们可以通过REDIRECT机制，针对所有来自192.168.0.0/24网段，且要送

往因特网的TCP Port 80的数据包，将其中的Destination IP改为127.0.0.1，Destination Port改为3128。但千万别忘了，要设置一条“一对多的NAT”使企业内部可以连上因特网。

```
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 \
    -j REDIRECT --to-port 3128
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.0/24 \
    -j SNAT --to 10.0.1.200
```

5.8 反向代理

前面所提到的缓存代理或透明代理都是与客户端较为相关的应用，接下来将介绍用来保护Web服务器的反向代理机制，不过在开始讨论反向代理机制之前，你必须先了解一下Web服务器的服务类型。

5.8.1 Web服务器的分类

由于时代不断进步，因特网技术不断走向成熟，网页服务器的技术也跟着不断更新，目前，在一台Web服务器上同时提供多个网站已是司空见惯的事，但早期的Web服务器只能提供单一网站的内容，如果一个企业同时拥有多个网站，就得准备多台Web服务器，为了解决这个问题，Web服务器机制也就随之演进，下面就来分析Web服务器的处理方式。

1. 基于IP的虚拟主机

在那个年代，因特网还没有如此的兴盛，因此，IP地址的数量没有够不够的问题，当时的想法是在Web服务器上同时设置多个IP地址，再把这些IP对应到特定的网站上，而这就是所谓的基于IP的虚拟主机。下面通过图5-15来解释基于IP的虚拟主机的工作原理。

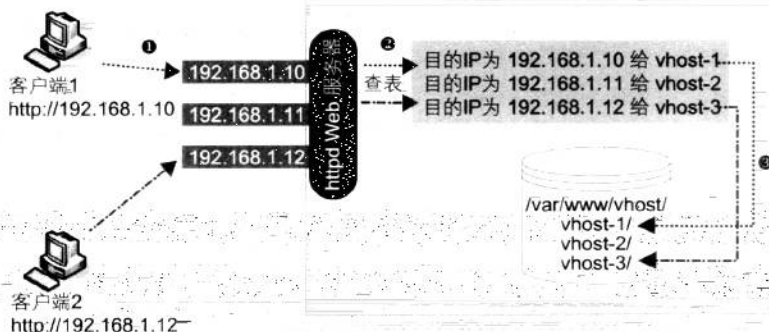


图5-15 基于IP的虚拟主机的工作原理

首先在Web服务器上设置192.168.1.10、192.168.1.11及192.168.1.12三个IP,并将三个网站的内容分别放在vhost-1/、vhost-2/及vhost-3/三个目录之中,接着告诉Web服务器:“如果客户端是访问192.168.1.10这个IP地址,那就给客户端vhost-1/这个目录下的文件”,这样当客户端在浏览器的网址栏中输入http://192.168.1.10时,客户端的浏览器就会对192.168.1.10这个IP地址提出服务请求❶,接着,Web服务器将客户端所访问的IP地址拿到对应表中匹配❷,由此确定应该为客户端提供哪个目录下的网页内容。

2. 基于名称的虚拟主机

虽然基于IP的虚拟主机机制可以在一台Web服务器上同时提供多个网站,但这个技术在后来的应用中,却暴露出重大问题,由于因特网应用的日益兴盛,使得IP地址的数量不够使用,因此,Web服务器的技术又有进一步的改进,为此基于名称的虚拟主机的技术就应运而生。在基于名称的虚拟主机的技术中,理论上,一个IP可以同时供N个网站来使用,如此IP的使用量就可以节省很多,下面以图5-16为例来解释基于名称的虚拟主机的工作原理。

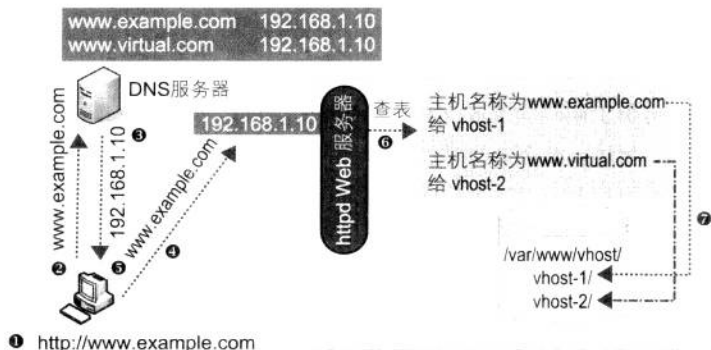


图5-16 基于名称的虚拟主机工作原理

首先在Web服务器上设置一个IP 192.168.1.10,并将两个网站的内容分别放在vhost-1/及vhost-2/两个目录中,并假设这两个网站的网址分别是www.example.com及www.virtual.com。接着,分别在DNS服务器上设置这两个网址所对应的IP都是192.168.1.10,最后还要告诉Web服务器:“如果客户端主机所送来的主机名称是www.example.com,就给客户端提供vhost-1/目录下的文件”,如此当客户端在浏览器的网址栏中输入http://www.example.com❶时,由于www.example.com这个字符串对浏览器而言是没有意义的,浏览器会将www.example.com送至DNS服务器上进行名称解析操作❷,接着DNS服务器回送www.example.com所对应的IP为192.168.1.10❸,在浏览器得知Web服务器的IP之后,就直接对Web服务器提出网页请求❹,此时,浏览器会执行一个非常重要的操作,就是将客户端在浏览器的网址栏中所输入的主机名称送给Web服务器❺,Web服务器在收到客户端所送上来的主机名称之后,就将主机名称送

到对应表中进行匹配⑥，如此Web服务器即可得知，应该给客户端提供哪个目录下的网页。

虽然基于名称的虚拟主机解决了浪费IP的问题，但引发了另一个问题：使用者在浏览网页时一定要输入网站的主机名称，如果使用者输入IP，Web服务器将无法判断应该给客户端提供哪个网站的内容，不过，这应该不是太大的问题，因为大部分人是不会使用IP来浏览网站的。

5.8.2 构建反向代理

在学习了反向代理的基本功能之后，我们将学习如何构建基于IP及基于名称的反向代理，最后讨论如何使用反向代理来保护IIS Web服务器。

1. 构建基于IP的虚拟主机

以图5-17为例来说明基于IP的反向代理的设置方式。在DMZ中有一台反向代理，其IP地址是192.168.0.10(我们假设其为公网IP)，而企业真实的Web服务器则部署在192.168.1.0/24的网段上，且其IP地址是192.168.1.100。

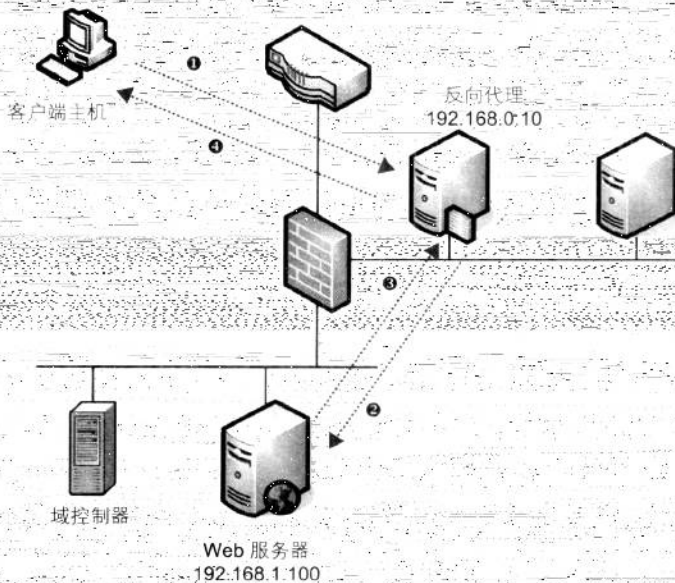


图5-17 基于IP的反向代理

接下来开始配置反向代理，不过在开始修改/etc/squid/squid.conf文件之前，务必确认这个文件的内容是最原始的，未经修改，如不确定也没关系，我们可以在/etc/squid的目录看到一个名为squid.conf.default的文件，在此只要将squid.conf.default复制为squid.conf即可，因为squid.conf.default就是Squid的原始配置文件。以下是本示例需要配置的参数。

```

31 acl web_server dst 192.168.1.100
32 acl web_port port 80
33 http_access allow web_server web_port
64 http_port 80 vport defaultsite=192.168.1.100
65 cache_peer 192.168.1.100 parent 80 0 no-query originserver

```

- 第31行及32行：分别定义客户端通过反向代理所能访问的“主机”及“端口号”。
- 第33行：限制客户端通过反向代理所能访问的范围。
- 第64行：设置Squid所服务的端口号是80，因为反向代理扮演着Web 服务器替身的角色，因此，客户端将反向代理当成Web 服务器来访问，所以一定要把Squid的服务器端口设置为80，而192.168.1.100则设置真实Web 服务器所在的IP地址。
- 第65行：告诉反向代理在客户端提出服务请求时，反向代理应该转向何处取得客户端所需的网页数据。

2. 构建基于名称的虚拟主机

以图5-18为例来说明基于名称的反向代理的工作流程及设置方法。图中的DMZ部分有一台反向代理，其IP是192.168.0.10，而这台反向代理所要代理的网站共有三个，分别是www.v1.com、www.v2.com及www.v3.com，其中www.v1.com及www.v2.com两个网站放在192.168.1.20的Web 服务器，www.v3.com放在192.168.1.30的Web 服务器。接着在DNS 服务器上分别将三个网站的IP地址都指向反向代理，也就是192.168.0.10这个IP。

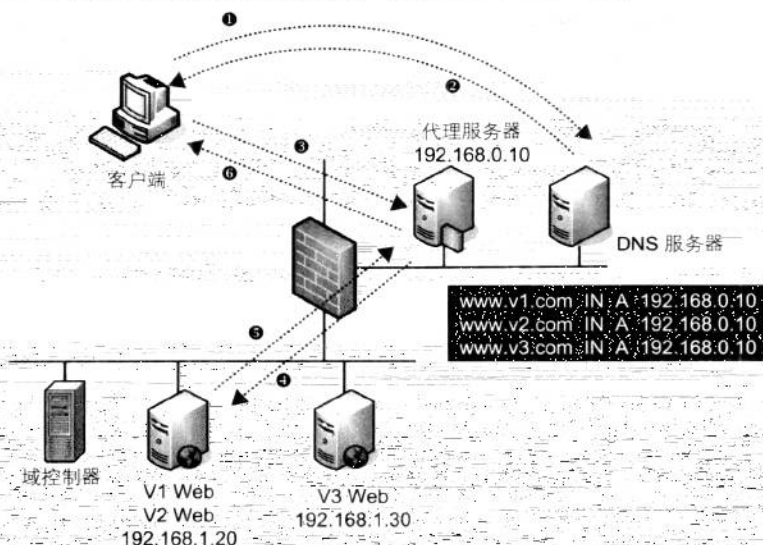


图5-18 基于名称的反向代理

因此，当客户端在浏览器的网址栏中输入www.v2.com时，由于这个字符串对浏览器而言没有意义的，浏览器将www.v2.com送到DNS 服务器进行名称解析❶，接着DNS 服务器应答给客户端192.168.0.10这个IP地址❷，客户端在得知www.v2.com所对应的IP之后，就会对反向代理提出网页请求，并且将使用者在网址栏中所输入的“主机名称”送给反向代理❸，反向代理在收到这些数据之后，就变成Web 客户端的角色，对真实的Web 服务器提出网页请求，并且将客户端刚刚所送来的主机名称送给真实的Web 服务器❹，Web 服务器接着将Web 客户端需要的网页送给反向代理❺，反向代理在收到这些网页数据之后，就将网页数据传送给真正的客户端，如此即完成一次基于名称的反向代理的工作周期。

下面分析基于名称的反向代理的设置方式，当然在开始设置之前，需将/etc/squid/squid.conf恢复为/etc/squid/squid.conf.default的默认值，如下是基于名称的反向代理的设置示例：

```
32 acl domain_v1 dstdomain www.v1.com
33 acl domain_v2 dstdomain www.v2.com
34 acl domain_v3 dstdomain www.v3.com
35 acl web_port port 80
42 http_access allow domain_v1 web_port
43 http_access allow domain_v2 web_port
44 http_access allow domain_v3 web_port
72 http_port 80 vhost
89 cache_peer 192.168.1.20 parent 80 0 no-query originserver name=web_server1
90 cache_peer 192.168.1.30 parent 80 0 no-query originserver name=web_server2
92 cache_peer_domain web_server1 www.v1.com www.v2.com
93 cache_peer_domain web_server2 www.v3.com
```

- 第32行~35行：分别定义客户端通过反向代理能够访问的“主机”及“端口号”。
- 第42行~44行：限制客户端通过反向代理所能访问的范围。
- 第72行：设置Squid所服务的端口号为80。
- 第89行~90行：设置真实Web 服务器所在的IP地址列表，并定义其代表名称。
- 第92行~93行：告诉反向代理，如果客户端送来的“主机名称”为www.v1.com或者www.v2.com，请到web_server1帮客户端取得其所要访问的网页数据，如果客户端送来的主机名称是www.v3.com，则到web_server2取得客户端所要访问的网页数据。

3. 如何在反向代理上运行https代理

在正常情况下，如果客户端与Web 服务器之间需要传输一些较机密的数据，会使用https 协议来加密Web 服务器与客户端之间的通信数据，这时需要在Web 服务器上安装服务器证书

(关于证书的相关技术,你可以先行参阅第15.4一节),以便让Web 服务器可以提供https加密服务,但是在反向代理的架构中并不是这样来保密的,下面以图5-19为例来说明这种结构。

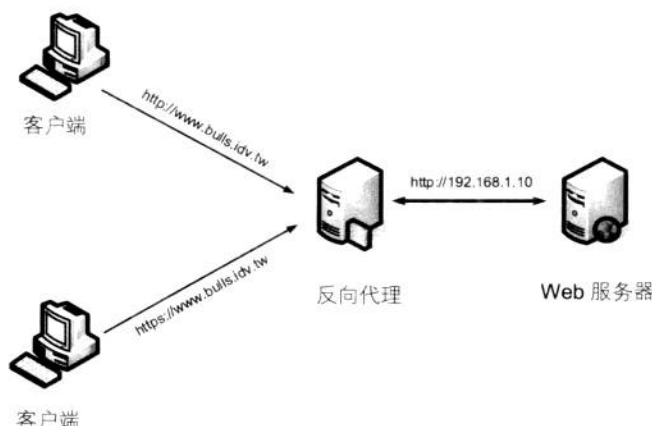


图5-19 结合使用反向代理与SSL

在图5-19中, Web服务器实际上并没有安装服务器证书,当然也就无法运行https的服务,反向代理与Web 服务器之间所使用的协议为http协议,而客户端与反向代理若要运行https服务,https服务则由反向代理本身提供,因此,必须在反向代理上安装服务器证书,如此客户端与反向代理之间就可以进行https加密服务。接着分析如何让Squid 反向代理提供https服务,在执行如下设置之后,请将/etc/squid/squid.conf恢复到/etc/squid/squid.conf.default的默认值。如下为设置示例:

```
32 acl web_server dst 192.168.1.10
33 acl web_port port 80
34 http_access allow web_server web_port
68 http_port 80 defaultsite=192.168.1.10 vhost
69 https_port 443 cert=/etc/squid/server.crt key=/etc/squid/server.key
   defaultsite=192.168.1.10 vhost
71 cache_peer 192.168.1.10 parent 80 0 no-query originserver
```

- 第32行及33行: 分别定义客户端通过反向代理所能访问的“主机”及“端口号”。
- 第34行: 限制客户端通过反向代理所能访问的范围。
- 第68行: 指定Squid所服务的端口号为80, 以及后端真实Web 服务器的IP为192.168.1.10。
- 第69行: 指定Squid下的https服务要工作于端口443, cert及key分别设置服务器证书与私钥的位置, 最后指定后端真实Web 服务器的IP为192.168.1.10。

- 第71行：告诉反向代理在客户端提出服务请求时，反向代理应该转向何处取得客户端所需的网页数据。

4. 使用反向代理及Netfilter来保护IIS Web 服务器

在讨论这个问题之前，必须基本了解HTTP通信协议，下面以一个HTTP访问操作为例来说明HTTP协议的交互方式。

这里列举一个虚拟的例子，我们可以在Linux系统中，使用telnet命令来连接Web 服务器的端口80，Web 服务器应答后，接着输入“GET /index.html HTTP/1.1”命令(这里假设index.html是根目录下的文件)，其中GET即为HTTP协议的命令，其含义是对Web 服务器提出请求，而请求的对象为index.html文件(/index.html)，并使用HTTP1.1的通信协议来传输，接着再输入一个基于名称的网站，所以Web 客户端必须送出一个字符串来告诉Web 服务器所要浏览的网站。

以上就是Web 客户端到Web 服务器访问一个网页的流程，这里舍弃了浏览器的图形界面，而改为最原始的命令行，以便你更好地了解HTTP协议的工作过程。不过，上例中仅介绍GET这个HTTP协议的命令，事实上，在HTTP协议下的命令还有很多，如下。

命令	功能说明
OPTIONS	在本列表中的所有命令都是HTTP协议的命令，但Web 服务器是可以控制其所要接受的命令范围。因此，Web 客户端可以通过OPTIONS命令询问Web 服务器可接受的命令范围
GET	Web 客户端发出请求的方法
HEAD	Web 客户端可以使用HEAD命令来检测Web 服务器上是否存在某个资源，如某个文件
POST	Web 客户端发出请求的另一种方法
PUT	Web 客户端可以通过PUT命令，将某个文件送到Web 服务器上
DELETE	Web 客户端可以通过DELETE命令，将服务器上的某个文件删除
TRACE	Web 客户端可以通过TRACE命令，来跟踪Web 客户端与Web 服务器之间的数据传输过程(例如，是否通过Web代理)
CONNECT	Web 客户端可以通过CONNECT命令来传输某些连接控制命令，例如Keep-Alive、Close等
DEBUG	DEBUG并不是HTTP的命令，而是MS Visual Studio的故障排除命令

要提高IIS的安全性，可从以下两处入手：

- 从HTTP协议的命令入手：

在HTTP协议的命令列表中，当前因特网上还用到的命令大概只剩GET及POST两个，因此，我们在设置Web 服务器时，应该停止使用GET及POST之外的命令，如此可防范一

些由于Web 服务器本身设计上的缺陷导致的安全问题；但如果从防火墙的角度看，可通过Netfilter的String模块来执行过滤操作，把GET及POST以外的命令全部过滤掉。

在此要特别说明一点，HTTP的命令一定使用“大写”字母，下面编写了这些Netfilter的过滤规则。

```
iptables -A INPUT -p tcp --dport 80 -m string --algo bm \
--string "OPTIONS" -j DROP
iptables -A INPUT -p tcp --dport 80 -m string --algo bm \
--string "HEAD" -j DROP
iptables -A INPUT -p tcp --dport 80 -m string --algo bm \
--string "PUT" -j DROP
iptables -A INPUT -p tcp --dport 80 -m string --algo bm \
--string "DELETE" -j DROP
iptables -A INPUT -p tcp --dport 80 -m string --algo bm \
--string "TRACE" -j DROP
iptables -A INPUT -p tcp --dport 80 -m string --algo bm \
--string "DEBUG" -j DROP
```

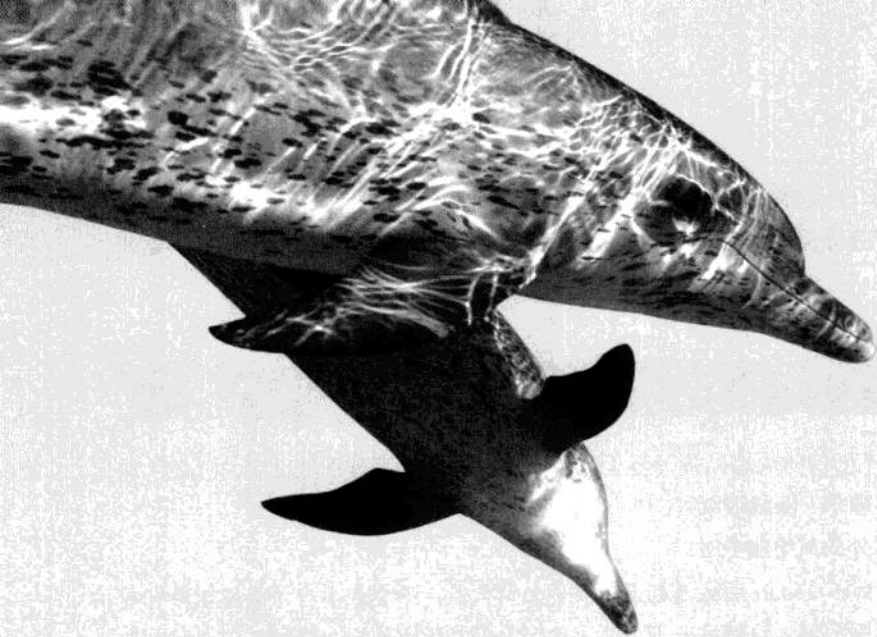
● 从URL的内容入手：

前面的章节曾经讨论过IIS安全漏洞的问题，例如，Web 客户端发出“/c/winnt/system32/cmd.exe/?c+dir”请求时，其目的是利用IIS的安全漏洞来入侵Web 服务器，在了解这个URL的含义之后，我们就可以使用反向代理来保护IIS Web 服务器。例如：可以使用url_regex的参数来匹配/winnt/system32这类字符串，并将这类请求拒绝，如此对于提升IIS的安全性是会有帮助的。下面列出几种常用来入侵IIS的URL字符串，如你感兴趣，不妨自己参考看看。

```
.*winnt.*system32.*
.*system32.*
.*\.exe
.*\.dll
```

5.9 小结

不管网络的连接速度再怎么快，HTTP 代理服务器都有其存在的价值。大容量且会被重复下载的数据依然会不断地在网络上出现(例如Microsoft Windows 升级的更新文件)，因此不管你的网络有多大的外网带宽，布置一台代理服务器对你的网络而言绝对是件好事；另外反向代理的角色对于企业的网站安全而言绝对有正面效果，如果贵公司所从事的行业较为机密，就更应该注意网站的安全性。



| 第6章 | 使用Netfilter/Iptables
保护企业网络

阅读了前面几章后, 相信你已经掌握了足够的技巧来使用Netfilter/Iptables, 接下来的章节将介绍一些防火墙规则方面的技巧及应该注意的事项, 以便使构建出来的防火墙更加贴近实际需要。

6.1 防火墙结构的选择

第1章中曾介绍过不同的防火墙结构, 至于其优缺点, 在此就不再重复说明。总之, 在选择防火墙结构时, 请遵守下面两项原则:

- 企业没有提供对外的网络服务时:

如果企业的DNS、Web及Mail 服务器都是由其他公司代管, 而企业本身可能根本没有实际的DNS、Web或Mail 服务器, 这种情况下, 防火墙可能仅是用来作为“保护”或“限制”企业网络对外的访问操作, 此类情况下可以选择如图6-1所示的架构。

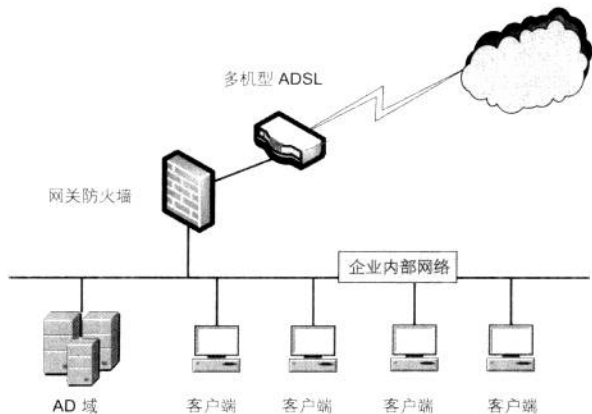


图6-1 网关式防火墙类型1

- 企业有提供对外的网络服务时:

如果企业本身布置了对外的服务主机, 那么就该选择如图6-2所示的架构, 当然, 如果需要更严格的安全环境, 可以选择如图6-3所示的架构, 不过, 第1章中曾经提到过, 在此架构中, 绝对不要使用两个相同的防火墙系统。

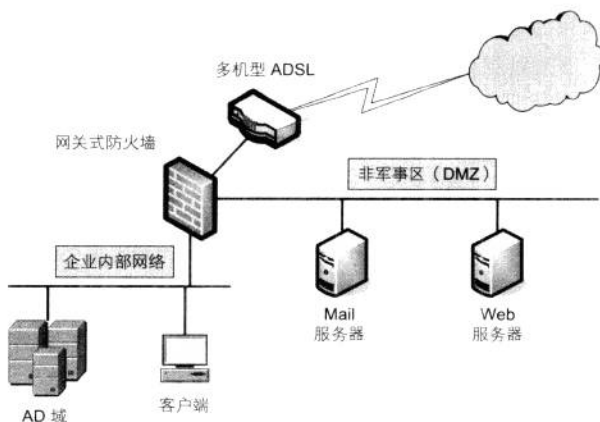


图6-2 网关式防火墙类型2

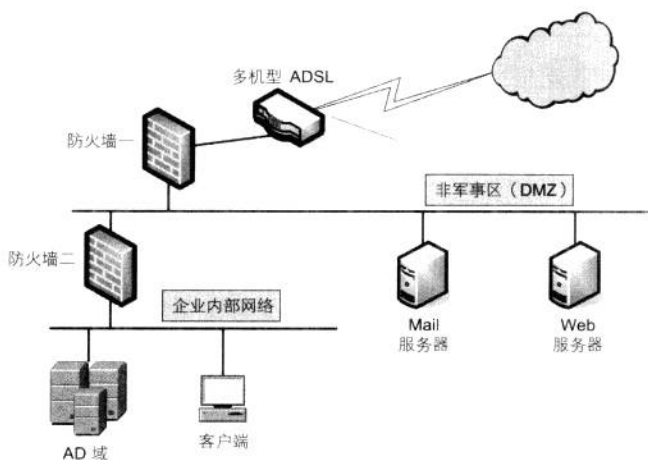


图6-3 网关式防火墙类型3

不管你选择的是图6-1、图6-2或是图6-3的架构，最后还要再次提醒你，千万不要选择图1-25中所示的架构，至于原因为何，不妨回顾一下第1.7.2一节第1部分的内容。

接下来以一般企业常会用到的服务为例，来说明这些服务器在存在防火墙的架构下该如何部署。如图6-4所示，首先选用DMZ式的防火墙架构，并将所有对外服务的主机部署于DMZ之上，但这里比较特别的是Web服务器部署的位置，我们选择将Web服务器及Web服务器用到的数据库服务器部署在企业网络内，并且使用DMZ内的反向代理将因特网上使用者

的请求转到企业内真实的Web 服务器，如此一来，才可以真正保护Web服务器及数据库服务器，最后再使用防火墙来限制企业内使用者访问因特网的行为。

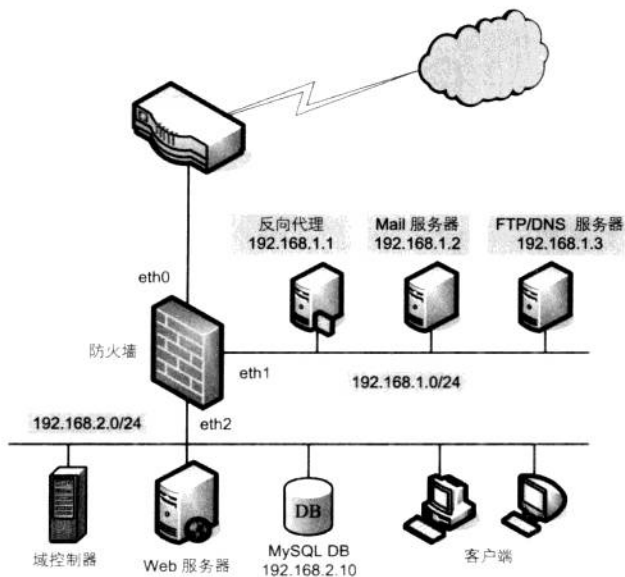


图6-4 防火墙部署示例

6.2 防火墙本机的安全

以图6-4为例，防火墙通常是企业网络的安全中心，而且会部署在企业内网的关键位置上，试想，如果防火墙本身被黑客入侵了，那么，黑客就可以从防火墙主机直接攻击DMZ或是企业内部的主机；或者当防火墙本机受到攻击而宕机时，也将导致整个企业网络的瘫痪；此外，还要考虑防火墙远程管理的问题，因为远程管理是通过“网络连接”来进行的，因此其中必然存在某些安全性问题。本节将进一步讨论防火墙的本机安全。

6.2.1 网络攻击

如果攻击者的目的只是要瘫痪某个企业，那么，攻击者极可能采取DOS或DDOS攻击方式。例如，使用ping of death通过大量ICMP数据包来占满企业的外网带宽。当然，如果攻击者拥有足够的外网带宽，要瘫痪一个小公司的网络是轻而易举的事。不过，如果企业真正遭

到这样的攻击，唯一能做的大概就是求助于ISP了，因为攻击者是以消耗带宽的方式进行攻击，即使企业安装了防火墙也无济于事。

6.2.2 系统入侵

谈到这里，有个很重要的概念你一定要牢记，千万不要为了节省经费而将对外的服务安装在防火墙中。例如，笔者常会看到一些小公司将Web、DNS、MAIL及FTP等服务部署在防火墙主机上，虽然这可以省去几台服务器的硬件费用，但防火墙的安全性也因此大打折扣。我们必须了解，如果一台主机(不管其安装什么操作系统)直接放在因特网之上，在其不开启任何网络服务的情况下，这台主机无论如何是不可能遭到入侵的。而主机之所以会遭到入侵一定要具备两个条件，其一是这台主机一定提供某些服务，例如Microsoft的网络邻居、Web服务、DNS服务、FTP服务等；其二是这些服务本身就存在安全漏洞，例如，前面提到IIS安全漏洞引发的URL攻击：“http://w.x.y.z/MSADC/root.exe?/c+dir”。

主机在具备这两个条件的情况下，黑客才有可能通过主机上服务的漏洞来进入系统，一旦黑客进入防火墙主机之后，就有可能执行防火防主机上的任何命令，例如，黑客只要在防火墙主机上执行iptables -F，防火墙的功能就会完全丧失，此外，黑客也极可能以防火墙为跳板，对内部网络或DMZ内的主机进行攻击。因此，无论如何你一定要记住，千万不可以在防火墙主机上部署对外的网络服务。

6.2.3 入站/出站的考虑事项

由于防火墙主机的安全极为重要，对于防火墙主机的进出连接必须严格地加以管控，以下分为两个部分进行讨论。

● 入站：

前面提过，绝对不能能在防火墙上部署任何对外的网络服务，因此，任何“主动”连接到防火墙本机的连接都应该拒绝。

● 出站：

我们有可能因为某些特殊功能的需求，而在防火墙上安装从因特网下载的软件(例如，进出电子邮件的监控)，如果软件是从官方网站上所获取，通常不会有什么存在“木马程序”的问题。但由于官方网站所提供的软件通常为“程序源代码”，而程序源代码是必须经过编译才能使用，而大多数的Linux防火墙管理人员通常不具备这样的能力，因此防火墙管理人员通常会在网络上寻找他人已经制作好的RPM软件，但毕竟这些RPM软件是经过人为“加工”的，里面是否放入木马程序，就不得而知了。

为了避免不小心安装了包含木马程序的软件，以免木马程序将系统的重要数据向外泄露，应该严格管理从防火墙主机主动对外建立连接，例如：

- ICMP协议：

我们通常会使用ping 168.95.192.1来检测对外网络是否中断，但你可能不知道，ICMP数据包实际上也是可以用来传输数据的，因此，对从防火墙往外发送的ICMP包一定要严加管理。例如，可以限制由防火墙本机所送出的ICMP包只能送到某几个特定的IP。

- UDP协议：

UDP协议最常被应用的场景就是DNS名称解析，但UDP协议能用来传输数据，因此，我习惯上会限制由防火墙本机所送出的数据包只能送往DNS服务器的UDP 53端口，至于其他在防火墙主机上不会用到的UDP协议则一律封锁掉。

- TCP协议：

在防火墙本机上经常会用到各种TCP服务，例如，需要执行软件的更新操作等，因此，我习惯上会限制由防火墙主机主动对外建立的TCP连接。例如，只能连到某几个特定的IP、CentOS软件的更新网站等。

6.2.4 远程管理的安全考虑事项

防火墙的管理如果能够完全在防火墙主机上执行，那当然是最安全的管理方式，但如果为了便于管理在有限度的情况下开放远程管理，也不是什么坏事。我的管理方式是通常只允许企业内部的主机使用SSH连接到防火墙主机上进行管理，习惯上还会以锁MAC地址的方式来确保只有我自己的计算机才能通过SSH连接进入防火墙主机。

6.3 防火墙的规则定义

构建防火墙的目的在于提升企业网络的安全，然而，要构建一个安全的防火墙系统，除了防火墙自身的质量之外，一个有计划且全盘考虑而规划出来的防火墙系统才能真正维护企业网络的安全。因此，了解防火墙的特性以及正确的思考逻辑将有助于设计出更符合企业需求的防火墙系统。至于Linux防火墙特性的部分，相信你应该都已了解，接下来将讨论如何正确编写出防火墙规则。将分三部分分别说明。

6.3.1 企业内部与因特网

在考虑企业内部与因特网之间的规划时，请从下面几个方面加以考虑：

● NAT的规划:

在规划NAT时,首先要选择好一个私有IP的区段,如果目前企业已有分部(分公司),最好系统地安排这些分部的IP网段,至于该如何划分并没有统一的标准,只要便于管理和记忆就好。

或许你感到疑惑,为什么要这样划分IP网段呢?如果将所有分公司的内部网段都设置为192.168.0.0/24不是很好记忆吗?之所以这样建议是因为分部与分部之间,会有很多的机会将其网络连接起来。而这个虚拟个人网络你可以暂且将之想象为两个私有IP网络之间在逻辑上以一个路由器连接起来,因此,如果路由器两侧的网段相同,路由器就无法正常提供路由的功能,所以为了长远考虑企业的网络规划,每一个分部内部网络的私有IP网段请尽量不要设置为一样,否则,未来可能遇到不可预知的麻烦。

在选择好IP区间之后,接下来选择NAT的类型,如果没有特殊原因,“一对多NAT”可能是最佳的选择,因为我们可以通过“一对多NAT”来减少公网IP的浪费,又可以通过“一对多NAT”的特性达到保护企业内部主机的目的。

● 防火墙的考虑事项:因特网与企业内部网络的事项

防火墙可用来保护企业服务网络,但别忘了,防火墙也可以用来限制企业内使用者所能访问的因特网资源。例如,我们可以限制企业内部的使用者不得连接到www.playboy.com的网站、不能使用FTP、不能使用MSN等,又或许我们可以考虑是否该规划一个代理服务器等。关于这个问题,这里将分为三个方面来讨论,如下说明。

● 由因特网到企业内网的考虑事项:

关于这个方向的连接管理,应该没有什么太多的考虑因素,我们只需要允许RELATED及ESTABLISHED两种状态的数据包可以正常送回企业内部即可,其余的数据包则一律丢弃。

● 由企业内网到因特网的考虑事项:

关于这个方向的连接管理,应该就企业本身的管理策略来考虑,有些企业可能会完全允许使用者连接到任何地方,而有些企业可能连网页的浏览都会限制。因此,如果你希望当个人缘好的防火墙管理员,或许可以选择不施加限制;如果你希望当个没有人情压力的防火墙管理员,可以选择按照公司的政策来执行;如果你希望当个尽忠职守的防火墙管理员,那么就应该严格管理由企业内“主动”向外建立的连接。

● 代理服务器的规划及考虑事项:

部署代理服务器可以有效缓解企业的外网“下载”带宽,而且代理服务器具有URL过滤机制,这是Netfilter所不及的地方,可见代理服务器的部署还是有其必要性的。

我通常使用Netfilter加上代理,并采取严格方式,也就是开放通常使用的协议,如http、

https及其他会使用到的服务,其余协议则一律不开放。这样不但可以通过Netfilter及代理进行深度“过滤”操作,还可以获得缓存代理所带来的网络“下载加速”效果。另外,可以结合使用Netfilter的recent模块,以防因为病毒感染而耗尽防火墙所能承受最大连接数量所产生的问题。

6.3.2 DMZ与因特网

由于这部分牵扯到企业对外服务器的安全性,因此必须格外谨慎去考虑,这里将其分成两个方面来讨论,如下说明:

● NAT的需求:

在以往防火墙规划中,通常将ISP所给的IP网段划分成两部分,例如,划分成两个各占一半的C类网段,然后将一半C类网段放在防火墙外面,另外一半C类网段放在DMZ上,但根据目前因特网的使用情况来看,不可能再采用以前的规划方式,因为公网IP数量已不够使用。因此,就目前的状况来看,DMZ在规划上大多还是使用私有IP,然后再以“一对NAT”或是“NAPT”的方式,将DMZ上的主机对应到公网IP。

至于应该选择“一对NAT”还是“NAPT”?就得看我们实际所拥有的公网IP数量,以及实际需要的服务器数量来衡量,如果实际需要的服务器数量比所拥有的公网IP数量还多,唯一选择就是NAPT;如果公网IP的数量 \geq 所需服务器数量,建议尽量使用“一对NAT”,因为这在管理上较不容易因疏忽而导致错误。

● 防火墙的考虑事项:

当我们规则好NAT之后,就等于把DMZ内的主机部署在因特网上,因此,一定要设置合适的防火墙规则来保护DMZ内的主机。关于防火墙的规则设置,以图6-4为例,可以将其分成两部分来讨论。

● 因特网对DMZ内主机的访问:

图6-4中在DMZ上有反向代理、Mail服务器及FTP/DNS服务器三部分,下面是我所思考的结果。首先根据服务类型分别建立HTTP_SRV、MAIL_SRV、FTP_SRV及DNS_SRV四个用户定义的链^⑤,并且把FORWARD的Default Policy设置为DROP^⑥,为了以后方便地改变服务器IP,又分别为三台服务器设置了IP变量,如Web_IP、MAIL_IP及FTP_IPD。因为在DMZ上有FTP服务器,而且FTP服务器是以NAT的方式对应到因特网,所以必须将filter及NAT处理FTP协议专用的模块加载到系统^⑦。

在^⑤的规则中,我们把任何由因特网送到DMZ中,且处于INVALID状态的数据包丢弃掉,而^⑥的规则则是允许ESTABLISHED及RELATED状态的数据包可以正常地在因特网及

DMZ之间穿梭；**HIOK**规则是将每条连接中的第一个数据包分别送到各项服务专属的用户定义的链，再按照不同协议的特性来加以限制。

LMNO规则为对各个不同协议的限制，其中**L**是对http及https的限制，**M**是对smtp及pop3的限制，**N**是对ftp的限制，**O**是对DNS的限制；我认为http及https应该不会有客户端在60秒之内，连续对Web服务器提出30次以上的连接请求操作，因此，限定60秒之内有30次以下的服务请求操作是合理的。不过，这60秒30次的频率也未必符合你的需要，因此，请自己统计一下当浏览Web服务器时，每浏览一个网页大概会需要建立几条连接，你可以试着使用Netfilter的LOG Target去记录，因为写法不同时，浏览一个网页所会建立的连接数量就会不一样；在Mail服务器部分，这里所设置的频率是60秒5次、FTP是60秒2次，因为在其他的DNS服务器上会保留缓存，因此将DNS服务器设置为60秒1次。

```
iptables -t filter -F A
iptables -t filter -N HTTP_SRV B
iptables -t filter -N MAIL_SRV
iptables -t filter -N FTP_SRV
iptables -t filter -N DNS_SRV
iptables -t filter -P FORWARD DROP C
WEB_IP=192.168.1.1 D
MAIL_IP=192.168.1.2
FTP_IP=192.168.1.3
modprobe ip_nat_ftp E
modprobe ip_conntrack_ftp
iptables -t filter -A FORWARD -i eth0 -o eth1 -m state --state \
INVALID -j DROP F
iptables -t filter -A FORWARD -m state --state \
ESTABLISHED,RELATED -j ACCEPT G
#=== Jump to HTTP_SRV Chain H
iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn -d $WEB_IP -m state \
--state NEW -m multiport --dports 80,443 -j HTTP_SRV
#=== Jump to MAIL_SRV Chain I
iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn -d $MAIL_IP -m state \
--state NEW -m multiport --dports 25,110 -j MAIL_SRV
#=== Jump to FTP_SRV Chain J
iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn -d $FTP_IP -m state \
--state NEW -m multiport --dports 21 -j FTP_SRV
#=== Jump to DNS_SRV Chain K
iptables -A FORWARD -i eth0 -o eth1 -p udp -d $FTP_IP -m state \
--state NEW -m multiport --dports 53 -j DNS_SRV
#=====<< Rule For Web Server >>===== L
iptables -A HTTP_SRV -m recent --name web_srv \
--update --seconds 60 --hitcount 30 -j DROP
iptables -A HTTP_SRV -m recent --name web_srv
```



```

iptables -A HTTP_SRV -j ACCEPT
#=====<< Rule For Mail Server >>=====
iptables -A MAIL_SRV -m recent --name mail_srv \
--update --seconds 60 --hitcount 5 -j DROP
iptables -A MAIL_SRV -m recent --name mail_srv \
iptables -A MAIL_SRV -j ACCEPT
#=====<< Rule For FTP Server >>=====
iptables -A FTP_SRV -m recent --name ftp_srv \
--update --seconds 60 --hitcount 2 -j DROP
iptables -A FTP_SRV -m recent --name ftp_srv \
iptables -A FTP_SRV -j ACCEPT
#=====<< Rule For DNS Server >>=====
iptables -A FTP_SRV -m recent --name dns_srv \
--update --seconds 60 --hitcount 1 -j DROP
iptables -A FTP_SRV -m recent --name dns_srv \
iptables -A FTP_SRV -j ACCEPT

```

• DMZ内主机对因特网的访问

对于这个方向的服务请求操作，基本上，还是建议采用“较严格”的方式来进行管理，我的想法如下：

规则一：

限制由DMZ对因特网的ICMP数据包只能传送到168.95.192.1这台主机，而这里这样限制的理由是：大多数使用者都会使用ping命令来检测网路是否中断，但如果我们完全不限ICMP的数据包，当DMZ主机被植入木马程序时，木马程序就很有可能通过ICMP数据包将系统重要的数据转发到黑客的系统上，因此，这里限制从DMZ送出的ICMP数据包只能送往168.95.192.1这台主机上。

规则二：

对于WEB_Server主机对外发送数据包的部分，这里并没有使用NEW状态，这与第一条规则不同，因为ICMP数据包是由DMZ内主机“主动”对外引发的连接，所以使用NEW的状态来匹配，而WEB_Server一定是有客户端对WEB_Server提出服务请求后，WEB_Server才会被动地回应客户端，所以应该使用ESTABLISHED的状态来进行匹配。此外，为了防止WEB_Server上被植入木马，而导致主机上重要的数据被泄露，这里仅允许WEB_Server使用TCP端口80及443来应答客户端。

规则三：

对于MAIL_Server的管理就有点不一样。首先我们讨论POP3服务的部分，在此依然假设POP3服务仅能被被动地应答客户端。

规则四：

这条规则也用来限制MAIL_Server，但这部分的写法比较特殊，为什么是匹配Destination端口呢？以图6-5为例来解释SMTP服务的工作流程。假设A公司的客户端要发一封电子邮件给B公司的使用者，首先客户端主机连接到A公司的邮件服务器的TCP端口25，并把邮件传输给SMTP服务，SMTP服务会先将这封电子邮件存储在/var/spool/mqueue/目录下②，接着SMTP服务会派生出另外一个SMTP守护进程(SMTP Daemon)③，而此时这个SMTP守护进程将扮演着SMTP客户端的角色，并且使用系统分配的一个TCP端口为其传输端口，将这封电子邮件发送给B公司的邮件服务器④⑤。

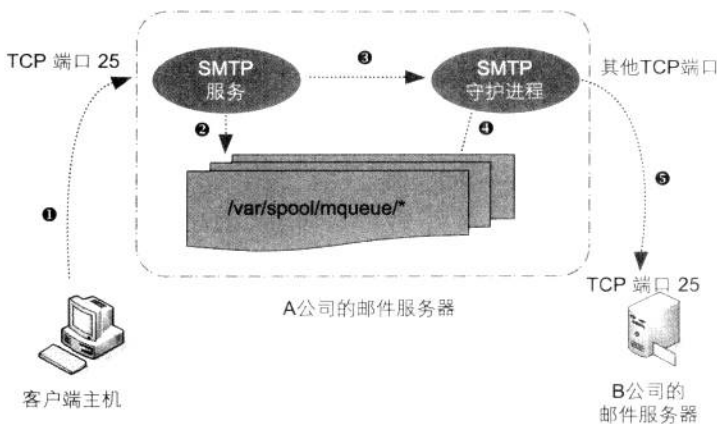


图6-5 SMTP Server工作流程

从以上流程可以看到，负责将电子邮件发送出去的端口并非TCP端口25，而是我们在第1章中所提到的动态端口(Dynamic Ports)，其范围会在端口49152~端口65535之间，因此，并无法得知SMTP守护进程会使用哪个端口将电子邮件发送出去，所以我的做法是：“限制从MAIL_Server送出去的数据包只能送到其他主机的TCP 25端口”。

规则五：

这条规则用来限制FTP服务，除了filter模块及NAT的模块需要加载外，其他部分基本与Web服务一样。

规则六：

这条规则是在限制DNS名称解析服务，因此，除了协议部分外，其他考虑基本与TCP服务类似。

```

1. iptables -A FORWARD -i eth1 -o eth0 -p icmp -d 168.95.192.1 -m state \
   --state NEW -j ACCEPT
2. iptables -A FORWARD -i eth1 -o eth0 -p tcp -s $HTTP_SRV -m multiport \
   --sports 80,443 -m state --state ESTABLISHED -j ACCEPT
3. iptables -A FORWARD -i eth1 -o eth0 -p tcp -s $MAIL_SRV -m state \
   --state ESTABLISHED --sports 110 -j ACCEPT
4. iptables -A FORWARD -i eth1 -o eth0 -p tcp -s $MAIL_SRV \
   --dports 25 -j ACCEPT
5. iptables -A FORWARD -i eth1 -o eth0 -p tcp -s $FTP_SRV -m state \
   --state ESTABLISHED,RELATED --sports 21 -j ACCEPT
6. iptables -A FORWARD -i eth1 -o eth0 -p udp -s $FTP_SRV -m state \
   --state ESTABLISHED --sports 53 -j ACCEPT

```

最后对于DMZ内的主机对因特网的http服务请求操作,我们可以通过缓存代理的管理,来限制DMZ内主机可以通过http协议访问因特网上的某些主机,如Windows更新服务器等。

6.3.3 企业内部与DMZ

在一般的环境下,企业内部与DMZ之间没有太大的问题需要考虑,一般的做法是允许企业内部全面访问DMZ,而不允许DMZ内的主机“主动”对企业内部网络进行访问操作,但在图6-4的示例中,因为使用到反向代理的架构,所以必须开放反向代理可以正常访问企业内部的Web服务器,但只要适度开放即可。

6.4 入侵与防御的其他注意事项

虽然我们已经将设计防火墙时该考虑的问题都考虑了,但防火墙并非一定能够防御系统入侵及弱点攻击,这个问题在第4.4.5一节中已解释过,如果忘记了不妨回头复习一下。接下来将要讨论实际的防御技巧。

6.4.1 更新系统软件

如果黑客攻击的是系统所存在的弱点,因而导致系统被入侵或是宕机,这就不是防火墙有办法可以防御的,因此,系统软件的及时更新是绝对必要的,这不会因为你使用的是Linux系统或是Windows系统而有所不同。

6.4.2 Syn Flooding攻击防御

对于某些安全问题,我们可以通过系统软件的更新来解决,但某些问题无法通过系统更

新来解决，例如，第4.4.5一节中提到的Syn Flooding攻击。关于这个问题，虽然在前面防火墙的规则中，已经使用recent模块对各项服务进行Syn Flooding攻击的防御操作，但这样的防御机制并无法防御Syn Flooding加上DDOS攻击模式，为了达成更有效地防御Syn Flooding攻击，我们可以从下面两个方面加以改进：

- 调整核心参数：

第4.4.5一节中曾提到三个核心参数，分别是net.ipv4.tcp_synack_retries、net.ipv4.tcp_max_syn_backlog及net.ipv4.tcp_syncookies，如果你已经忘记这三个参数的用途，那么，请阅读前面的章节再复习一下。

如果服务器使用Linux操作系统，就可以设置这些参数，如此即可提升防火墙后端服务器对Syn Flooding的防御能力。

- 我的独家秘方：

调整核心参数虽然可以提升Syn Flooding的防御能力，但如果Syn数据包的数量真的很大，以上办法所能得到的效果也是有限的，此时，我的做法是丢弃TCP连接中的第一个Syn数据包。因为TCP协议的特性会确保传输中的数据不会丢失，所以如果我们故意将第一个Syn数据包丢弃，那么，在一段时间后，发送端一定会再送一次Syn数据包，而这段时间约3~5秒。

了解这个特性后，我们再来看看DDOS+Syn Flooding的攻击特性，当此类攻击发生时，攻击者会不断地伪造来源端IP，并对我们的服务器送出含有Syn标记的数据包。因此，如果是DDOS攻击包，当我们将某个IP所送来的第一个Syn包丢弃时，它应该就不会重送第二个Syn数据包进来。

在结合以上两个特性之后，将防火墙规则设计如下(以http服务为例)：

```
1. #!/bin/bash
2. iptables -t filter -F
3. modprobe ipt_recent ip_list_tot=16384
4. iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn --dport 80 -m recent \
  --name SYN_FLOOD --update --second 120 --hitcount 1 --j ACCEPT
5. iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn --dport 80 -m recent \
  --name SYN_FLOOD --set
6. iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn --dport 80 -j DROP
```

从以上规则中，假设防御主机的eth0连接到因特网，而eth1连接到Web服务器。让我们先来看看规则四，在这条规则中，会匹配合送到Web服务器的Syn数据包；如果以目前这个时间点往前推120秒，某个IP在SYN_FLOOD数据库中没有被记录时，那么，这个数据包将不会符合这条规则，因此，这个数据包会被送到规则五进行匹配。接着，这个数据包的发送端IP会被记录在SYN_FLOOD数据库中，但别忘了，recent模块的--set参数仅会执行记录操作，

并不会对数据包进行任何的处理操作, 因此, 这个数据包接着会被送入到规则六中做匹配, 而规则六就将这个Syn数据包给丢弃掉。不过, 这个Syn数据包的发送端数据已经被记录在SYN_FLOOD的数据库中, 因此, 当这个发送端重送第2个Syn数据包进来时, 其将会符合规则四而被系统接受。

最后提醒你, recent模块默认只会存储100条数据, 而Syn Flooding攻击发生时, 会需要记录更庞大的客户端信息, 因此, 在规则三中把recent模块的记录数调整到16384, 当然, 如果你的外网带宽很大时, 笔者会建议再加大这个值。

以上这个方法确实可以更有效地防御Syn Flooding的攻击, 不过, 却也造成了其他更为头疼的问题。因为在以上规则中, 不管Syn Flooding的攻击是否发生, 任何http连接中的第一个数据包都会被丢弃掉, 因此, 客户端在浏览网页时, 一定会因为数据包重发而多等一段时间, 虽然这个时间只有3~5秒, 却让人感到相当不舒服, 特别是像我这种追求完美的技术人员, 所以想了又想, 最后把防火墙的规则修改如下:

```
1. #!/bin/bash
2. modprobe ipt_recent ip_list_tot=16384
3. iptables -N SYN_FLOODING
4. iptables -t filter -F
5. iptables -A FORWARD
6. #=====
7. iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn --dport 80 -m limit \
    --limit 1/m --limit-burst 300 -j ACCEPT
8. iptables -A FORWARD -i eth0 -o eth1 -p tcp --syn --dport \
    80 -j SYN_FLOODING
9. #=====
10. iptables -A SYN_FLOODING -i eth0 -o eth1 -p tcp --syn --dport 80 \
    -m recent --name SYN_FLOOD --update --second 120 \
    --hitcount 1 -j ACCEPT
11. iptables -A SYN_FLOODING -i eth0 -o eth1 -p tcp --syn \
    --dport 80 -m recent --name SYN --set
12. iptables -A SYN_FLOODING -i eth0 -o eth1 -p tcp --syn -j DROP
```

在以上规则中, 将Syn Flooding的防御规则放到SYN_FLOODING的用户定义的链中, 并在规则七中使用limit模块来判断系统是否遭受到Syn Flooding攻击, 判断方法如下:

- 如果在60秒内“没有超过”300个Syn数据包进来, 即表示系统并没有遭受Syn Flooding的攻击。
- 如果在60秒内“超过”300个Syn数据包进来, 表示系统正遭受Syn Flooding的攻击, 此时进来的syn数据包并不符合规则七, 而符合规则八, 所以这些Syn数据包将会被送

到SYN_FLOODING的用户定义的链进行匹配。

根据以上规则，如果系统没有遭受Syn Flooding攻击，所有的Syn数据包将会符合规则七并送到Web 服务器，但如果系统遭受Syn Flooding攻击，除了limit模块1/m参数部分每60秒可以进入一个Syn数据包(这是limit模块语法的限制，否则0/m会更好)，以及Syn Flooding攻击刚开始发生的前300个Syn数据包之外，其余每个发送端所送来的第一个Syn数据包都会被recent模块丢弃掉。我们可以发现以上规则在正常情况下，并不会延长任何的TCP连接时间，而在Syn Flooding攻击发生时，又能及时帮我们过滤掉Syn Flooding攻击。以上是我独创的用来防御Syn Flooding攻击的方法，如果你有兴趣，不妨将其加入到你的防火墙之中，就算笔者为你的防火墙做一份贡献。

6.4.3 IP欺骗防御

在设置防火墙规则时，经常忽略掉IP欺骗攻击问题，什么是IP欺骗呢？以图6-6为例来说明。首先请教你一个问题，如果图中防火墙的规则为“iptables -A FORWARD -i eth0 -p all -s 192.168.1.10 -j DROP”，请问黑客在192.168.1.10主机上，能不能将攻击包发送到192.168.2.50这台主机上？

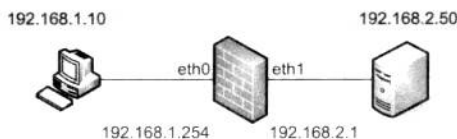


图6-6 IP欺骗

以上这个问题是肯定的，因为黑客只要伪造来源端IP，就可轻易骗过防火墙，将攻击包注入192.168.2.50这台主机，因此，在设计防火墙规则时，请留意以下几种特殊的来源端IP：

● Localhost IP:

在IP欺骗的攻击中，127.0.0.1这个IP也是常被拿来伪装的对象，可在防火墙规则中添加以下规则，以便过滤掉这类IP欺骗攻击。

```
iptables -A INPUT -i ! lo -s 127.0.0.0/8 -j DROP
```

● Private IP:

在因特网IP的划分上，将IP划分为A类、B类、C类三个区段，而在每个层级的类中都保留了一段私有IP，如果我们从因特网上收到从私有IP网段发送来的IP，都可视为不正常的来源数据包，因此，我们可以在防火墙规则中加入以下规则，来过滤掉这类IP欺骗攻击。

```
iptables -A INPUT -p all -s 10.0.0.0/8 -j DROP
iptables -A INPUT -p all -s 172.16.0.0/12 -j DROP
iptables -A INPUT -p all -s 192.168.0.0/16 -j DROP
```

● MULTICAST IP

MULTICAST 通常用于音频或视频信号的传输,而MULTICAST数据包所使用的IP网段为224.0.0.0/4,且数据包传输协议一定是UDP,如果有数据包来自于224.0.0.0/4网段,而其传输协议为UDP以外的协议,即可将其丢弃掉。我们可以在防火墙规则中加入以下规则,来过滤掉这类IP欺骗攻击。

```
iptables -A INPUT -p ! udp -s 224.0.0.0/4 -j DROP
```

在IP欺骗防御中,除了可以过滤掉前面所提到的这些IP之外,另一种可能出现的行为也需要注意,例如,防火墙主机上有连接A、B、C三个网段,我们可以在A网段上伪造B网段的IP来攻击C网段的主机,因此,早期的Linux防火墙的设置中都会添加以下规则:

```
iptables -A FORWARD -i eth0 -s ! 192.168.0.0/24 -j DROP
```

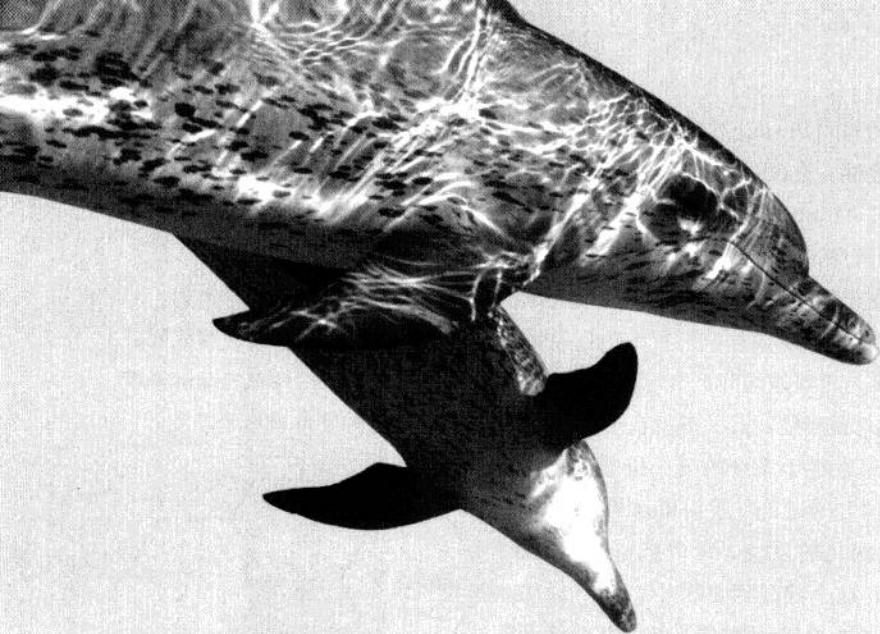
也就是说,对于从eth0接口送进来的数据包,如果其来源端IP不是来自于192.168.0.0/24网段,就将该数据包丢弃掉,由此过滤这类IP欺骗手段。不过,现在Linux内核提供特定的参数来自动过滤这类不合理的来源端IP,其参数及修改方式如下:

```
for i in /proc/sys/net/ipv4/conf/*; do
    echo 1 > $i
done
```

如此一来, Linux内核就会自动过滤掉这类数据包了。

6.5 小结

本章在一个假想实例的引导下,整合了前面所有章节所学的内容,并借助这个实例来描述设计一个防火墙所需要考虑的众多要素。当然啦,要设计出一个良好的防火墙系统,必须结合实际环境来考虑,这只是一个入门示例,往后的修行还得靠你自己。



Linux

| 第7章 | Linux内核编译

7



Linux内核堪称整个GNU/Linux系统的灵魂，我们以图7-1为例来说明Linux内核的重要性。在GNU/Linux系统中，我们粗略地把系统分为“应用程序”和“Linux内核”两个部分，其中应用程序所指的是一般运行的应用程序，例如：Firefox、Gnome、Mplayer等；



图7-1 GNU/Linux结构

而Linux内核的功能则在于管理及分配硬件资源等，也就是我们一般统称的kernel，例如，一个应用程序运行时要占用哪一块内存、运行中的程序是否具有访问硬盘中某个文件的权限、Mplayer在目前这个时刻是否可以访问声卡等操作。由此我们可以推断，一个GNU/Linux系统是否支持某个特定的硬件或功能，其关键就在于Linux内核是否支持这种硬件或这个功能。

那么Linux内核到底存在于系统的哪个位置呢？这个问题得从Linux的结构开始讨论，我们可以从图7-2看出Linux是一个高度模块化的系统，可以把整个Linux的中心称为结构中心，这个部分通常很小，在一般的GNU/Linux系统中，大约为2MB(我曾经把这个部分缩小到只有几百KB的大小而已)，因为在这有限的空间内并无法容纳太多代码，自然也就限制了Linux所能提供的功能，但因Linux是一个模块化系统，所以可以通过外加模块的方式不断扩展Linux的功能。



图7-2 Linux的结构

而模块又是什么呢？其实模块指的是具有某项功能的程序代码而已，有了这段代码，就可以让Linux去执行某个特定的任务。

在Linux系统中，模块存在的方式有两种，其一是这段代码以单一文件方式存在；其二是把这段代码结合到结构中心去。前者我们称为动态模块，后者称为静态模块。这两种形式的模块各有优缺点，如下：

● 动态模块：

在动态模块的结构下，由于模块代码都已经被单独放在单一的文件中，因此，无论Linux下有多少模块，结构中心的大小都不至于有太大变化。而且我们可以根据任务需要将模块加载到内存，或将暂时用不到的模块从内存中移除，以实现内存使用的最优化。动态模块的文件存放在/lib/modules/kernel_version/*.ko。

● 静态模块：

在静态模块的结构下，由于模块的代码已经结合到结构中心，因此，静态模块数量越多，结构中心就会越大，但最要命的是所有静态模块会与结构中心一起加载到内存。例如，在结构中心含有100个静态模块，而此时此刻我们只需要使用到其中的一个模块，但实际所

消耗的内存空间却是100个模块所占用的空间，因此，在一般应用中，除非这个模块在大多数情况下都会用到，否则，会优先考虑动态模块形式。

在一般GNU/Linux系统中，通常会把Kernel(结构中心)以vmlinuz-kernel_version格式的文件名保存在/boot目录下，而动态模块的部分则保存在/lib/modules/kernel_version目录下，并使用*.ko作为模块的文件名。

7.1 为何需要重新编译内核

重新编译内核的可能性并不是很大，通常是因为在内核中缺少了某个需要的功能时，才会自己编译内核。以图7-3为例，在开始进行内核编译之前，我们会需要内核的代码，然后通过一个称为menuconfig的接口来进行内核功能的筛选，由此来指定编译出来的内核会包含哪些功能，并指定这些被选定的功能是要以静态或动态模块的形式存在。

现在的Linux已经是一个功能相当完整且强大的系统，因其内置了非常多的模块，但并非所有功能都是大家所需要的，所以一些GNU/Linux厂商在编译内核时，就不选取某些比较冷门的功能。例如，图7-3的第二个选项并没有被选取，所以编译出来的内核自然就无法提供相应的功能；另外，第一个选项为<*>表示这个功能将会被编译成“静态模块”；第三个选项为<M>，表示这个功能将会被编译成“动态模块”。如果我们所需要的功能已经内置在内核的源代码中，只是厂商没有将其编译进内核，那么，只需将该功能选项选取进来，然后重新编译内核即可。

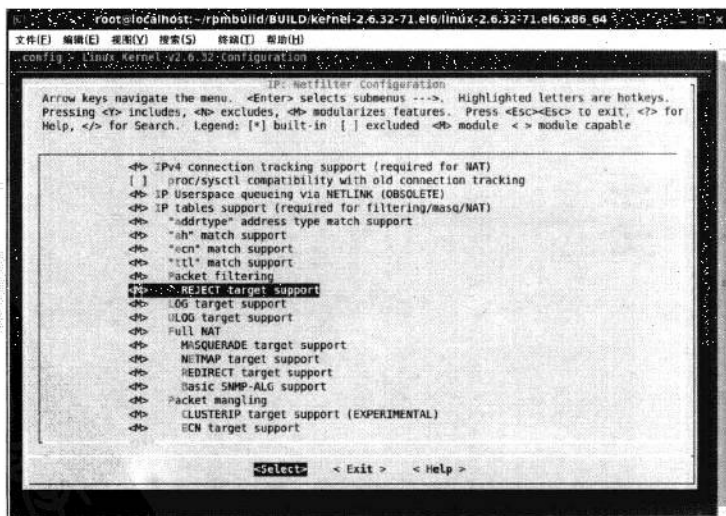


图7-3 核心功能选项

但如果我们所需要的是内核所不支持的功能,也就是说,在内核源代码中根本不包含所需要的功能时,这时只能到网络上寻找是否有高手额外为这个功能或硬件编写了代码。例如,笔者手上有一个三频电视棒,内核并不支持这个硬件,这时便可以试着到硬件厂商的网站寻找是否有Linux的驱动程序可供下载,如果运气够好,便可以下载到厂商提供的驱动程序,再按照说明文件将其安装即可;但如果厂商没有提供Linux的驱动程序,也不用太难过,因为在网络上仍有可能找到高手为这个硬件所编写的免费程序,这类的程序我们统称为补丁(Patch),只需要按照补丁内附的说明文件,把这部分源代码合并到内核的源代码中,然后重新编译内核即可。把外在的程序代码合并到内核的代码中,我们称之为内核补丁(Kernel Patch)。

7.2 内核编译

GNU/Linux是否可以读写NTFS文件系统呢?这个问题在大多数的使用者看来是否定的,实际上,Linux支持NTFS文件系统已经有相当长的一段时间,只是因为大多数GNU/Linux厂商并没有把支持NTFS的功能开放,而让使用者误以为Linux没有支持NTFS文件系统。接下来,就以“启用Linux NTFS Support”为例来解释内核编译的过程。

7.2.1 安装软件开发环境

在进行内核编译之前,我们必须先准备好“开发工具”,所谓的开发工具就是在编译过程中需要的工具及函数库等。例如gcc、glibc-*、rpm-build等可以在安装GNU/Linux时直接安装,或在系统安装完成之后再安装都可以。如图7-4为例,就是在安装GNU/Linux系统的过程中直接安装开发工具的方法,我们只需要在“开发”项目中选中“开发工具”即可。

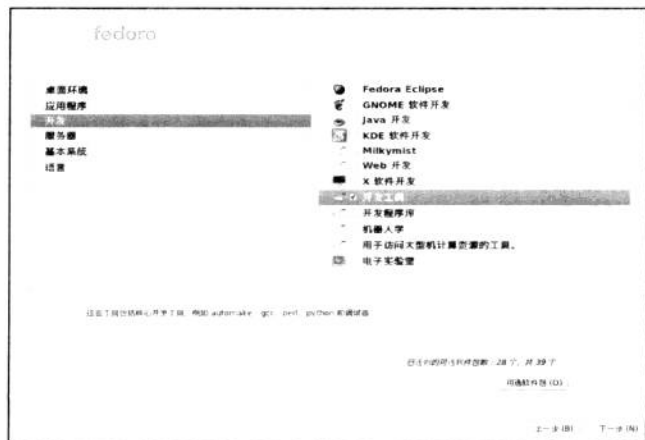


图7-4 软件管理系统

若是在GNU/Linux系统安装完成之后才安装开发工具，就会比较麻烦一点了。如果你使用的GNU/Linux为CentOS或Fedora，可以在系统连接到因特网的情况下，通过yum命令连接到因特网直接下载软件进行安装，命令执行如下所示：

```
yum -y groupinstall 'Development tools'
```

如果你使用的是RedHat Enterprise Linux，那么，只有购买RedHat网上的软件更新服务，才可以通过yum来安装软件，不然，就得自己架设安装服务器。

7.2.2 获取内核源代码

可以从两个途径取得Linux源代码，其一是到Linux的官方网站<http://www.kernel.org>下载；其二是到GNU/Linux的厂商网站获取。两处所得到的内核代码各有其优点，如下：

- Linux官方网站：

<http://www.kernel.org>是Linux源代码的发源地，因此，如果你想获取“最新版本”的Linux源代码，这里就是你的最佳选择。

- GNU/Linux厂商网站：

所有厂商开发的GNU/Linux系统的内核都下载自<http://www.kernel.org>网站，但官方网站所关心的是内核的大架构及未来发展方向，小细节并非他们所关心的重点。内核所包含的驱动程序数量就是一个明显的例子，例如将RedHat 8.0所使用的内核与官方网站相同版本的内核来比较，会发现前者可以支持U盘，但后者却不可以。

之所以存在这样的问题，完全是因为驱动程序数量不是官方关心的问题。GNU/Linux的厂商力求使其系统更符合我们的需求，因此会进行一些加工处理。例如，他们会在网络上到处寻找社区为内核所额外编写的补丁，再将这些程序源代码加入到从官方网站下载回来的内核源代码中，由此增加内核的功能或者增加一些驱动程序。所以GNU/Linux厂商所提供的内核源代码，通常比官方的内核源代码包含更多功能，不过，这样的差异在Linux 防火墙的范畴内，就不是那么令人关心的问题，因为在Linux防火墙上只要网卡能正常工作，并能正常运行netfilter即可，其他多余的模块并不一定需要。

在了解了以上差异后，你的选择会是什么呢？如果是官方网站，请自己到<http://www.kernel.org>网站下载linux-2.6.*.tar.gz文件；若选择GNU/Linux厂商所提供的内核代码，则请到相应厂商的网站去下载，如RedHat Enterprise Linux可以到<http://ftp.redhat.com/redhat/linux/enterprise/>网址下载；CentOS则可到<http://ftp.isu.edu.tw/pub/Linux/CentOS/>网址下载。

7.2.3 整合源代码

如果你是从<http://www.kernel.org>网站下载的源代码，那么所获得的源代码应该是linux-2.6.*.tar.gz或linux-2.6.*.tar.bz2的格式，我们只要直接将其解压缩到合适的路径下即可。建议的路径是/root/rpmbuild/BUILD。

如果你采用GNU/Linux厂商提供的内核源代码，通常会RPM的软件包，如kernel-2.6.*.src.rpm之类的文件名。这个RPM文件内除了包含官方网站所提供的linux-2.6.*.tar.gz内核源代码之外，还会包含很多由GNU/Linux厂商收集的大量补丁文件，我们只有通过以下步骤把这些补丁文件整合到内核源代码之中，才能进行内核的编译操作：

步骤01：使用rpm工具安装kernel-2.6.*.src.rpm软件包

使用以下命令安装软件包，在安装过程中会出现一些警告信息，请直接忽略不用关心，安装完毕后，内核源代码会保存在/root/rpmbuild/SOURCES路径下，在这个目录中我们可以看到一个名为linux-2.6.*.tar.bz2的文件，这就是厂商从<http://www.kernel.org>下载回来的内核源代码，而其他的*.patch的文件则是厂商在因特网上收集到的补丁文件。

```
rpm -ivh kernel-2.6.32-71.el6.src.rpm
```

步骤02：将所有补丁文件整合到内核源代码中

在进行整合之前，请先确认你操作系统的平台版本，可以使用uname -m命令来查询。确认后就可以开始整合操作，请将工作路径切换到/root/rpmbuild/SPECS目录下，接着执行以下命令来进行整合，其中x86_64就是我使用uname -m命令查询得到的。

```
rpmbuild -bp --target=x86_64 kernel.spec
```

如果执行过程中出现以下信息，则表示在你的系统中缺少了某些重要组件，只需使用yum命令将这些缺少的组件安装起来即可。

```
构建目标平台: x86_64
构建目标 x86_64
错误: 依赖性构建失败:
    elfutils-libelf-devel 是 kernel-2.6.32-71.el6.x86_64 所需要的
    zlib-devel 是 kernel-2.6.32-71.el6.x86_64 所需要的
    binutils-devel 是 kernel-2.6.32-71.el6.x86_64 所需要的
    hmaccalc 是 kernel-2.6.32-71.el6.x86_64 所需要的
[root@localhost SPECS]#
[root@localhost SPECS]# yum -y install elfutils-libelf-devel zlib-devel binutils-devel
hmaccalc
```

整合后的内核源代码会存放在/root/rpmbuild/BUILD/kernel-2.6.32-71.el6/linux-2.6.32-71.el6.

x86_64目录下, 其中2.6.32-71是我所使用内核源代码的版本号, x86_64则是我所使用的操作系统平台版本。若你使用的版本与此不同, 请自己替换成符合你工作环境中的路径位置。

7.2.4 设置编译完成后的内核版本号

可以使用`uname -r`命令来查询操作系统所使用的内核版本号, 方法如下:

```
[root@localhost linux-2.6.32-71.el6.x86_64]# uname -r
2.6.32-71.el6.x86_64
```

其中2.6.32-71.el6.x86_64就是内核版本号, 也就是说, 目前在我的计算机里已安装了一个2.6.32-71.el6.x86_64内核, 因此, 就不能再安装第二个相同版本号的内核, 那么, 请问编译好的内核版本号会是什么呢? 其实这个版本编号应该是在编译之前手动设置好的, 版本号记录在`/root/rpmbuild/BUILD/kernel-2.6.32-71.el6/Linux-2.6.32-71.el6.x86_64/Makefile`文件中, 默认值如下所示:

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 32
EXTRAVERSION =
```

如果我们希望编译好的内核版本编号是2.6.32-80, 请这些默认值, 如下所示:

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 32
EXTRAVERSION = -80
```

7.2.5 清理内核源代码以外的临时文件

内核源代码是可以不断重复使用的, 但前一次编译所遗留下来的临时文件, 有可能会導致这次编译的失败, 因此, 在每次进行内核编译之前, 建议你清理一下这些临时文件, 只需使用以下命令即可:

```
[root@localhost linux-2.6.32-71.el6.x86_64]# cd /root/rpmbuild/BUILD/kernel-2.6.32-71.el6/linux-2.6.32-71.el6.x86_64
[root@localhost linux-2.6.32-71.el6.x86_64]#
[root@localhost linux-2.6.32-71.el6.x86_64]# make mrproper
```

不过有一点你要特别注意, 这个操作会与下一个步骤所生成的配置文件一并清理掉, 因此在执行这个命令之前建议你先将其进行备份。

7.2.6 设置内核编译参数

稍早曾提过：Linux中包含很多功能和驱动程序，但并非所有功能我们都需要；另外，某些功能需要编译成静态模块，某些则编译成动态模块。

以上的决定权完全掌控在一个称为.config的文件中，这里从.config截取一小段与Netfilter有关的内容，其中等号左边的内容是内核的功能选项，右边若是y，则表示这个选项要编译成静态模块；若是m，则表示要编译成动态模块；如果你不需要这个选项，将它从清单中删除即可。

```
#
# IP: Netfilter Configuration
#
CONFIG_NF_DEFRAG_IPV4=y
CONFIG_NF_CONNTRACK_IPV4=y
# CONFIG_NF_CONNTRACK_PROC_COMPAT is not set
CONFIG_IP_NF_QUEUE=m
CONFIG_IP_NF_IPTABLES=y
CONFIG_IP_NF_MATCH_ADDRTYPE=m
CONFIG_IP_NF_MATCH_AH=m
CONFIG_IP_NF_MATCH_ECN=m
CONFIG_IP_NF_MATCH_TTL=m
```

在此提出一个问题，请问我们可不可以自己编译出一个与目前使用中的内核完全相同、但又多加一项NTFS Support的内核呢？其实这个问题很容易解决，以RedHat Enterprise Linux、Fedora Linux及CentOS为例，在这三种GNU/Linux系统中，厂商都会将其当时在编译内核时所使用的.config放在/boot目录下，例如，我目前计算机所使用的内核版本号为2.6.32-71.el6.x86_64，那么，在/boot目录下可以找到一个名为config-2.6.32-71.el6.x86_64的文件。也就是说，我目前使用中的内核就是通过这个配置文件定义出来的，因此，我们只需要将这个配置文件复制到/root/rpmbuild/BUILD/kernel-2.6.32-71.el6/linux-2.6.32-71.el6.x86_64目录下，并取名为.config，然后将NTFS Support的功能选项加进来即可。

接着执行以下操作，以便将/boot/config-2.6.32-71.el6.x86_64文件复制到/root/rpmbuild/BUILD/kernel-2.6.32-71.el6/linux-2.6.32-71.el6.x86_64目录下，并取名为.config。

```
[root@localhost linux-2.6.32-71.el6.x86_64]# cd /root/rpmbuild/BUILD/kernel-2.6.32-71.el6/linux-2.6.32-71.el6.x86_64
[root@localhost linux-2.6.32-71.el6.x86_64]#
[root@localhost linux-2.6.32-71.el6.x86_64]# cp /boot/config-2.6.32-71.el6.x86_64 .config
cp: 是否重写 '.config' ? y
[root@localhost linux-2.6.32-71.el6.x86_64]#
```

将配置文件复制进来之后，就可以开始来修改.config的内容了，不过，我们可不是使用vi来修改内容，因为难度实在是太高了，绝非一般人有能力处理的。修改.config时请使用内核源代码所附带的工具，并按照以下方式启动编辑工具。

```
[root@localhost linux-2.6.32-71.el6.x86_64]# cd /root/rpmbuild/BUILD/kernel-2.6.32-71.el6/linux-2.6.32-71.el6.x86_64
[root@localhost linux-2.6.32-71.el6.x86_64]# make menuconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/basic/docproc
HOSTCC scripts/basic/hash
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/kxgettext.o
*** Unable to find the ncurses libraries or the
*** required header files.
*** 'make menuconfig' requires the ncurses libraries.
***
*** Install ncurses (ncurses-devel) and try again.
***
make[1]: *** [scripts/kconfig/docchecklxdialog] Error 1
make: *** [menuconfig] Error 2
```

如果在执行过程中出现了以上错误信息，则代表在你的系统中缺少了重要组件，请使用以下命令安装所需组件，如可成功运行，将可以看到如图7-5所示的菜单：

```
yum install ncurses*
```

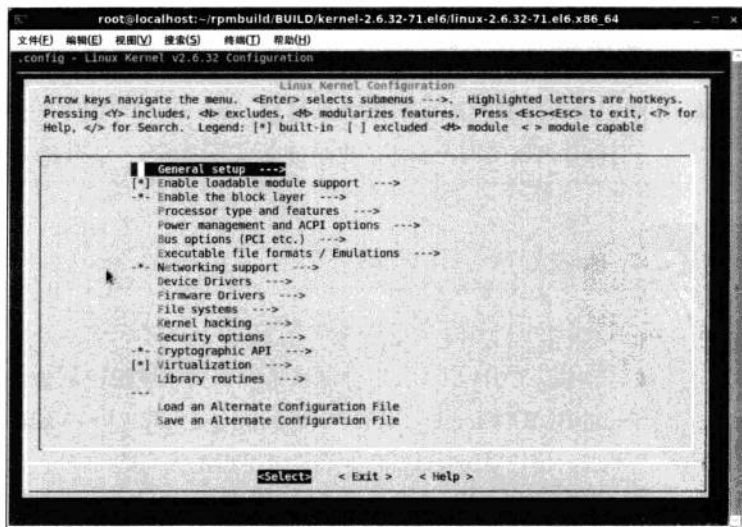


图7-5 menuconfig

等到menuconfig工具运行起来之后,就可以通过menuconfig工具来修改.config的内容,步骤如下:

步骤01: 加载原有.config文件的内容

请选择Load an Alternate Configuration File选项,并加载.config文件的内容,如此,.config文件内的设置值将会完全在menuconfig工具中呈现出来。

步骤02: 启用NTFS Support

请选择File Systems|DOS/FAT/NT Filesystems|NTFS file system support,我们可以看到config-2.6.32-71.el6.x86_64配置文件内默认情况下没有启用NTFS Support,请在这个选项上按“空格键”将其设置为M(动态模块),接着在NTFS write support选项上按“空格键”即可。

步骤03: 回到第一层菜单

请按左右方向键,将最下面的选项调整到Exit然后按Enter键,此时光标应该会往上跳一层,接着请再次选择Exit并按下Enter键,此时应该会回到第一层菜单。

步骤04: 保存设置及退出menuconfig工具

如果在操作过程中想先保存目前更改过的内容,可以选择Save an Alternate Configuration File选项;如果已经完成设置且想要保存并退出menuconfig工具,那么可以直接选择最底下的Exit,并按Enter键,此时menuconfig工具会询问我们是否要保存,只要选择Yes,就会将所有设置的内容保存在.config文件中。

7.2.7 执行编译操作

正如前面所提到的,Linux应该分成两个部分来讨论,其一是内核的“结构中心”,其二是内核的“模块”。而执行内核的编译操作,也是分成以下两部分来工作的。



提示

在此要提醒你一件事,请不要在图形界面中执行编译操作,因为这样会延长内核编译的时间。

● 编译结构中心:

为了加快编译速度,请按下Ctrl+Alt+F1组合键将系统切换到虚拟控制台,接着再将工作路径切换到/root/rpmbuild/BUILD/kernel-2.6.32-71.el6/linux-2.6.32-71.el6.x86_64,然后执行make bzImage命令,不过,请注意bzImage除了第三个字母是大写的I之外,其余的都是小写,千万别看错了。

此外,如果在整个编译过程中产生任何错误,编译操作都将会自动中断,并且会显示出

错误信息，通常这种情况都是我们将.config内不该去掉的东西给去掉了，请重新再复制并设置一次.config文件。当然，这也有可能是其他问题造成的，如果不会编写C及内核程序，遇到这种问题是没有办法处理的。但若我们没有修改内核源代码(如额外加上了什么补丁之类的)，在一般情况下是不会轻易发生问题的，若真的遇到了，就请再更换一个版本的内核源代码试试。

● 编译模块：

模块的编译是可以与结构中心编译同时进行的，请按下Ctrl+Alt+F2键将系统切换到虚拟控制台，接着再将工作路径切换到/root/rpmbuild/BUILD/kernel-2.6.32-71.el6/linux-2.6.32-71.el6.x86_64，最后执行make modules命令开始编译模块。整个编译过程所需要注意的事项与前一个步骤类似，只不过在此如果出现错误信息，我们可以很容易看出是哪个模块有问题，只需重新打开menuconfig工具，然后把无法编译的模块给删除掉，然后再执行make modules即可继续编译。

等到编译完毕内核后，我们就可以看到如图7-6所示的界面。模块编译完成后，将可以看到如图7-7所示的界面。

```
CC      arch/x86/boot/video-vga.o
CC      arch/x86/boot/video-vesa.o
CC      arch/x86/boot/video-bios.o
LD      arch/x86/boot/setup.efi
OBJCOPY arch/x86/boot/setup.bin
OBJCOPY arch/x86/boot/vmlinux.bin
HOSTCC  arch/x86/boot/tools/build
BUILD   arch/x86/boot/bzImage
Root device is (253, 0)
Setup is 135000 bytes (padded to 13824 bytes).
System is 3744 kB
CRC 36a21100
Kernel: arch/x86/boot/bzImage is ready (#1)
[root@localhost linux-2.6.32-71.x86_64]#
[root@localhost linux-2.6.32-71.x86_64]#
```

图7-6 内核编译

```
IHEX    firmware/ti_3410.fw
IHEX    firmware/ti_5852.fw
IHEX    firmware/mts_cdma.fw
IHEX    firmware/mts_gsm.fw
IHEX    firmware/mts_edge.fw
H16T0FW firmware/edgeport/boot.fw
H16T0FW firmware/edgeport/boot2.fw
H16T0FW firmware/edgeport/down.fw
H16T0FW firmware/edgeport/down2.fw
IHEX    firmware/edgeport/down3.bin
IHEX2FW firmware/whiteheat_loader.fw
IHEX2FW firmware/whiteheat.fw
IHEX2FW firmware/keyspan_pda/keyspan_pda.fw
IHEX2FW firmware/keyspan_pda/xircom_pgs.fw
[root@localhost linux-2.6.32-71.x86_64]#
```

图7-7 模块编译

7.2.8 安装模块及结构中心

在完成内核及模块的编译后，接下来把编译好的东西复制到其应该被放置的路径之下，

下面分成两部分进行讨论:

● 安装模块:

模块的安装较为简单,我们只需要在内核源代码的目录下执行make modules_install命令即可,如此编译好的模块将会被复制到lib/modules/kernel-version的目录下。

```
[root@localhost ~]# cd /root/rpmbuild/BUILD/Kernel-2.6.32-71.el6/linux-2.6.32-71.el6.x86_64
[root@localhost linux-2.6.32-71.el6.x86_64]#
[root@localhost linux-2.6.32-71.el6.x86_64]# make modules_install
```

● 安装结构中心:

整个安装过程如下例所示,为了方便执行,我习惯将工作路径切换到内核源代码的目录下①,接着,把编译好的结构中心复制到/boot目录下并取名为vmlinuz-kernel-version②,步骤③则是复制内核中各种功能的内存对应位置,至于这是什么东西对我们来说就不重要了,但请记得一定要做这个操作就是了。步骤④则是将工作路径切换到/boot目录下,以便制作initial ram disk,步骤⑤则是制作initial ram disk,在执行这个操作之前,请确认你的模块已经安装完成,否则,制作initial ram-disk的操作将会失败。这个文件内容是系统在启动过程中,用来初始化系统的一些重要文件,至于内容是什么?那并不是我们所关心的,但如果你对Linux Embedded System感兴趣的话,最好要百分之百了解这个文件的内容及用途,我在Linux Embedded System的课堂上,可是要求每位同学必须能手工编写出initial ram disk,因为如果不了解这个文件的内容,是没有能力编写这个文件的,就等于没有上过Linux Embedded System课程。

```
[root@localhost ~]#
[root@localhost ~]# cd /root/rpmbuild/BUILD/Kernel-2.6.32-71.el6/linux-2.6.32-71.el6.x86_64 ①
[root@localhost linux-2.6.32-71.el6.x86_64]#
[root@localhost linux-2.6.32-71.el6.x86_64]# cp arch/x86_64/boot/bzImage /Boot/vmlinuz-2.6.32-80 ②
[root@localhost linux-2.6.32-71.el6.x86_64]#
[root@localhost linux-2.6.32-71.el6.x86_64]# cp System.map /boot/System.map-2.6.32-80.img ③
[root@localhost linux-2.6.32-71.el6.x86_64]#
[root@localhost linux-2.6.32-71.el6.x86_64]# cd /boot ④
[root@localhost linux-2.6.32-71.el6.x86_64]#
[root@localhost linux-2.6.32-71.el6.x86_64]# dracut -H -f /boot/initramfs-2.6.32-80.img 2.6.32-80 ⑤
[root@localhost linux-2.6.32-71.el6.x86_64]#
```


7.2.9 修改开机管理程序

准备好所有文件后，最后的工作就是把编译好的内核加入到开机选择菜单中。以 RedHat、Fedora 及 CentOS 为例，我们所使用的开机管理程序都是 GRUB，GRUB 开机管理程序的配置文件是 `/boot/grub/grub.conf`，以下显示的是修改前与修改后的内容：

```
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu

title Red Hat Enterprise Linux (2.6.32-44.1.el6.x86_64)
    root (hd0,0)
    kernel /vmlinuz-2.6.32-44.1.el6.x86_64 ro root=/dev/mapper/VolGroup-lv_root rd
    LVM_LV=VolGroup/lv_root rd_LVM_LV=VolGroup/lv_swap rd_NO_LUKS rd_NO_MD rd_NO_DM LANG=zh.
    UTF-8 KEYBOARDTYPE=pc KEYTABLE=us crashkernel=auto rhgb=quiet
    initrd /initramfs-2.6.32-44.1.el6.x86_64.img

1. default=0
2. timeout=30
3. splashimage=(hd0,0)/grub/splash.xpm.gz
4. #hiddenmenu

5. title Red Hat Enterprise Linux (2.6.32-80)
6.     root (hd0,0)
7.     kernel /vmlinuz-2.6.32-80 ro root=/dev/mapper/VolGroup-lv_root rd_LVM_LV=VolGroup/
    lv_root rd_LVM_LV=VolGroup/lv_swap rd_NO_LUKS rd_NO_MD rd_NO_DM LANG=zh.UTF-8
    KEYBOARDTYPE=pc KEYTABLE=us crashkernel=auto rhgb=quiet
8.     initrd /initramfs-2.6.32-80.img

9. title Red Hat Enterprise Linux (2.6.32-44.1.el6.x86_64)
10.     root (hd0,0)
11.     kernel /vmlinuz-2.6.32-44.1.el6.x86_64 ro root=/dev/mapper/
    VolGroup-lv_root rd_LVM_LV=VolGroup/lv_root rd_LVM_LV=VolGroup/lv_
    swap rd_NO_LUKS rd_NO_MD rd_NO_DM LANG=zh.UTF-8 KEYBOARDTYPE=pc
    KEYTABLE=us crashkernel=auto rhgb=quiet
12.     initrd /initramfs-2.6.32-44.1.el6.x86_64.img
```

其中需要修改的部分如下：

- 第2行：`timeout=30`

把选择菜单等待使用者选择的时间调整为30秒，如此可以为使用者提供更长的时间来选
择要使用哪个内核系统来开机。

- 第4行：`hiddenmenu`

在开机时，默认情况下开机选择菜单是隐藏的。可以将这行删除或注释掉，以后在开机

时, 选择菜单就会直接显示在界面上。

- 第5行: title Red Hat Enterprise Linux(2.6.32-80)

设置新内核在开机选择菜单上的提示信息。

- 第6行: root(hd0,0)

设置kernel及initial ram disk的相对路径位置。

- 第7行: kernel / vmlinuz.....

设置kernel的文件名及其他提供kernel或系统用的参数。

- 第8行: initramfs /initramfs.....

指定initial ram disk的文件名称。

设置完GRUB开机管理程序后, 就可以重新开机了, 这时应该可以看到如图7-8的界面。

从图中可以看到系统安装了两个不同版本的内核, 此时, 可以使用“↑”、“↓”键来选择此次要用哪个内核开机, 选择后直接按“Enter”键, 计算机便会开机进入该系统。



图7-8 GRUB开机管理程序

在系统开机完成后, 就可以用命令uname -r来查询目前执行的内核版本编号是否为2.6.32-80, 也就是我们自己编译的新内核, 另外, 在/lib/modules/2.6.32-80/kernel/fs目录下, 可以看到一个名为ntfs的目录, 这个目录中应该有一个名为ntfs.ko的文件, 这就是我们之前选择NTFS Support功能所编译进来支持NTFS文件系统的模块, 如果以上两项检查都符合要求, 恭喜你, 已经具备编译内核的能力了。

7.3 如何安装内核补丁

接下来将以Application Layer Filter补丁为例，介绍整个内核补丁的安装过程，至于什么是Application Layer Filter呢？这个问题留到下一章再来说明。

7.3.1 下载补丁文件及内核源代码

在安装内核补丁之前请先做好心理准备，给内核安装补丁这个操作有可能会失败而导致内核无法编译出来，原因通常都是内核源代码版本与补丁文件所支持的内核版本不同所致，因此，如果你遇到安装内核补丁后导致编译错误，请将内核源代码删除，并寻找其他内核版本与补丁文件版本的组合，或许就可以成功编译好内核。以下版本组合是我亲自测试过的，如果你没有安装内核补丁的经验，建议先尝试使用表7-1中所示的版本组合：

表7-1 l7filter与内核版本兼容性测试列表

内核版本	layer7-v2.22.tar.gz	l7filter-2.6.35-support.tar.bz2
kernel-2.6.32-44.2.el6.src.rpm	○	X
kernel-2.6.32-71.el6.src.rpm	X	X
linux-2.6.32.27.tar.bz2	○	X
linux-2.6.35.10.tar.bz2	X	○

其中kernel-2.6.32-44.2.el6.src.rpm是Red Hat Enterprise 6 BETA版所使用的内核，kernel-2.6.32-71.el6.src.rpm是Red Hat Enterprise 6正式版所使用的内核，linux-2.6.32.27.tar.bz2及linux-2.6.35.10.tar.bz2则是官方网站所下载的内核，另外layer7-v2.22.tar.gz及l7filter-2.6.35-support.tar.bz2则是Application Layer Filter所提供不同版本的补丁文件。综合以上测试结果，我选择了linux-2.6.35.10.tar.bz2及l7filter-2.6.35-support.tar.bz2的组合作为本书内容的示例。

请到<http://www.kernel.org/pub/linux/kernel/v2.6/>下载Kernel程序源代码，Application Layer Filter原来的官方网站是<http://l7-filter.sourceforge.net>，但现在已经转到<http://l7-filter.clearfoundation.com>，因此，请到http://l7-filter.clearfoundation.com/tracker/my_view_page.php下载l7filter-2.6.35-support.tar.bz2文件。图7-9是Application Layer Filter补丁文件的下载页面。



图7-9 Layer 7下载页面

7.3.2 准备内核及补丁的源代码

为了避免不必要的错误，我们先把编译所需的内核源代码及Application Layer Filter的程序源代码准备好，执行以下步骤：

步骤01：内核源代码

请将下载回来的内核源代码linux-2.6.35.10.tar.bz2解压到/root/rpmbuild/BUILD目录下，并执行make mrproper，以清理掉前次编译时留下的临时文件，如下：

```
[root@localhost /]# cd /root/rpmbuild/BUILD
[root@localhost BUILD]# tar -jxvf /root/linux-2.6.35.10.tar.bz2

.....省略.....

[root@localhost BUILD]# cd linux-2.6.35.10/
[root@localhost linux-2.6.35.10]#
[root@localhost linux-2.6.35.10]# make mrproper
```

步骤02：补丁源代码

请将刚才下载回来的补丁文件解压到/root目录下，便可以看到7filter-2.6.35+.changes.patch及l7filter-2.6.35+.complete.patch两个文件。

```
[root@localhost ~]# cd /root/
[root@localhost ~]# tar -jxvf l7filter-2.6.35-support.tar.bz2
l7filter-2.6.35+.changes.patch
l7filter-2.6.35+.complete.patch
[root@localhost ~]#
[root@localhost ~]# ls -l
```

总计 208

```
-rw-----. 1 root root 2349 2011-01-03 13:48 anaconda-ks.cfg
-rw-r--r--. 1 root root 63171 2011-01-03 13:48 install.log
-rw-r--r--. 1 root root 13998 2011-01-03 13:45 install.log.syslog
-rw-r--r--. 1 root root 1676 2010-09-09 21:44 l7filter-2.6.35+.changes.patch
-rw-r--r--. 1 root root 60627 2010-09-09 21:46 l7filter-2.6.35+.complete.patch
-rw-----. 1 root root 16999 2011-01-07 16:53 l7filter-2.6.35-support.tar.bz2
drwxr-xr-x. 9 root root 4096 2011-01-07 15:35 rpmbuild
```

7.3.3 运行内核补丁

接着上一节的步骤，请将工作路径切换到/root/rpmbuild/BUILD/linux-2.6.35.10目录下，并且将/boot/config-2.6.32-71.el6.x86_64复制成.config，接着根据Layer7官方网站的文件<http://l7-filter.sourceforge.net/HOWTO-kernel#Kernel>来运行内核补丁，在此请执行patch -p1 < /root/l7filter-2.6.35+.complete.patch命令，设置完毕后，便可看到总共打了哪些补丁文件到内核源代码中，此外，也别忘了设置内核的版本号(我设置的编号是2.6.35-20)。

```
[root@localhost /]# cd /root/rpmbuild/BUILD/linux-2.6.35.10/
[root@localhost linux-2.6.35.10]# cp /boot/config-2.6.32-71.el6.x86_64 .config
[root@localhost linux-2.6.35.10]# patch -p1 < /root/l7filter-2.6.35+.complete.patch
patching file include/linux/netfilter/xt_layer7.h
patching file include/net/netfilter/nf_conntrack.h
patching file net/netfilter/Kconfig
patching file net/netfilter/Makefile
patching file net/netfilter/nf_conntrack_core.c
patching file net/netfilter/nf_conntrack_standalone.c
patching file net/netfilter/regexp/regexp.c
patching file net/netfilter/regexp/regexp.h
patching file net/netfilter/regexp/regmagic.h
patching file net/netfilter/regexp/regsub.c
patching file net/netfilter/xt_layer7.c
[root@localhost linux-2.6.35.10]#
```

7.3.4 设置内核编译参数

在运行内核补丁之后，就可以使用make menuconfig菜单工具来设置.config文件的内容了，而Layer7所打的补丁模块可以在Networking support | Networking options | Network packet filtering framework(Netfilter) | Core Netfilter Configuration下找到一个名为“layer7” match support(NEW)的模块，请将其选中，其他操作则与第7.2一节中的介绍完全一样，请自行参考。

7.3.5 内核编译完毕后的检查

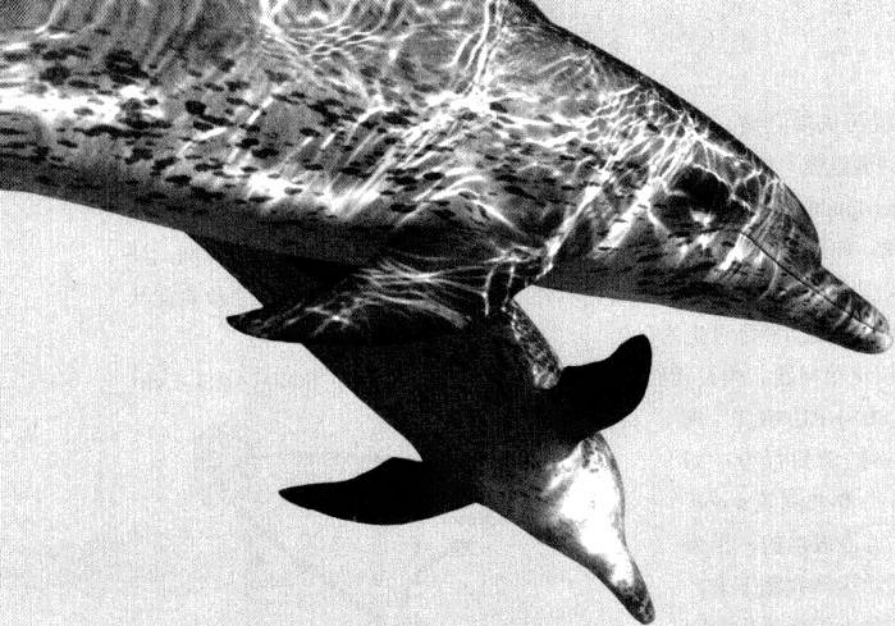
内核编译好且安装完毕后，如图7-10所示，我们应该可以在/lib/modules/2.6.35-20/kernel/net/netfilter目录下看到一个名为xt_layer7.ko的模块，当然，这也代表我们给内核打补丁成功了。

```
[root@localhost ~]#  
[root@localhost ~]# cd /lib/modules/2.6.35-20/kernel/net/netfilter  
[root@localhost netfilter]#  
[root@localhost netfilter]# ls  
ipvs          nf_tproxy_core.ko  xt_layer7.ko      xt_RATEEST.ko  
nf_conntrack_amanda.ko  xt_CLASSIFY.ko    xt_LED.ko         xt_real.ko  
nf_conntrack_ftp.ko     xt_cluster.ko     xt_length.ko      xt_recent.ko  
nf_conntrack_h323.ko    xt_connec.ko      xt_limit.ko       xt_sctp.ko  
nf_conntrack_irc.ko     xt_connbytes.ko   xt_mac.ko         xt_SECMARK.ko  
nf_conntrack_netbios_ns.ko  xt_connlimit.ko  xt_mark.ko        xt_socket.ko  
nf_conntrack_netlink.ko  xt_connmark.ko   xt_MARK.ko        xt_statistic.ko  
nf_conntrack_pptp.ko    xt_CONNSECMARK.ko  xt_multiport.ko  xt_string.ko  
nf_conntrack_proto_dccp.ko  xt_CONNSECMARK.ko  xt_NFLOG.ko      xt_tcpmss.ko  
nf_conntrack_proto_gre.ko  xt_dccp.ko       xt_NFQUEUE.ko    xt_TCPMSS.ko  
nf_conntrack_proto_sctp.ko  xt_dccp.ko       xt_NOTRACK.ko    xt_TCPOPTSTRIP.ko  
nf_conntrack_proto_udplite.ko  xt_DSCP.ko      xt_osf.ko         xt_time.ko  
nf_conntrack_sane.ko     xt_esp.ko        xt_owar.ko        xt_TPROXY.ko  
nf_conntrack_sip.ko      xt_hashlimit.ko  xt_physdev.ko     xt_TRACE.ko  
nf_conntrack_tftp.ko     xt_helper.ko     xt_pkttype.ko     xt_n32.ko  
nfnetlink.ko            xt_hl.ko         xt_policy.ko  
nfnetlink_log.ko        xt_HL.ko         xt_quota.ko  
nfnetlink_queue.ko      xt_iprange.ko    xt_rateest.ko  
[root@localhost netfilter]#
```

图7-10 xt_layer7

7.4 小结

本章旨在介绍Linux的组成结构，以及如何为Linux添加其他源代码，进而扩展Linux原本不提供的某些功能，最后则是将Linux的源代码编译成计算机能够直接执行的二进制码，我一直认为这项技术是一个称职的Linux技术人员所应该具备的能力，因为唯有这样的能力，才能真正把Linux的“自由”发挥出来，而不是像商用版系统一样，人家给你什么，你就只是用什么，完全没有任何自主能力。



Linux

| 第8章 | 应用层防火墙

8

在开始讨论应用层防火墙之前,先请教你一个问题,直到目前为止,你对Netfilter/Iptables的功能是否感到满意呢?如果你问我这个问题,我只能说不是很满意!为什么呢?因为到目前为止,Netfilter/Iptables只能处理OSI 7层以下的协议,因此,在应用上就会有无法企及之处。举例来说,如果要使用Netfilter/Iptables来封锁企业内部使用者浏览因特网,请问你会如何处理呢?我想大多数读者都会考虑封锁对外的TCP 端口80,但这样的手段真能杜绝浏览因特网的行为吗?答案当然是否定的。

我们以图8-1来说明这个问题。假设我们在防火墙上使用“iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 80 -j REJECT”规则来封锁企业对因特网的访问行为,如果使用者想要通过路径②来浏览www.facebook.com,那是不可能成功的,因为这个浏览操作一定会被防火墙拦截下来,但如果我们将浏览器的代理服务器指向因特网上不以端口80作为服务端口的代理服务器,那么,使用者的浏览器将会连接到代理服务器上,再由代理服务器将目标网页转发回来,如此便可绕过防火墙的拦截机制,例如路径③。

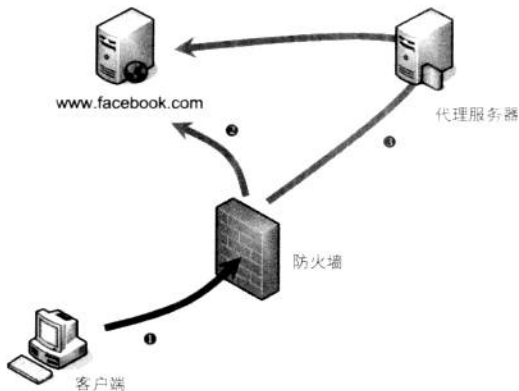


图8-1 包过滤的盲点

那么,如何才能完全拦截使用者浏览网页的行为呢?答案很简单!使用应用层防火墙就可以有效拦截浏览网页的行为,因为在网络上并没有强制应用层协议一定要使用哪个端口,我们可以通过不使用端口80的代理来帮我们躲避防火墙的检查,但是应用层防火墙通过检查所传输的数据内容来判断网络使用行为,因此,与端口并没有绝对的关联。所以不管你的http通信协议使用什么端口,在应用层防火墙的检查下根本无处遁形。不过,虽然应用层防火墙功能很强大,但也并非没有弱点,例如,传输的数据内容如果是经过加密处理的,应用层防火墙就完全使不上力,关于这点可要特别注意!

Netfilter/Iptables功能虽然强大,但只能处理OSI 7层以下的通信协议,还好网络上的志愿者朋友为Netfilter/Iptables编写了一个补丁,这个补丁使得Netfilter/Iptables得以提升为应用层防火墙,至于该如何为内核加上这个补丁?我已经在第7.3一节中实际带领读者们操作过了,因此,这里不再赘述。

不过,要让Netfilter/Iptables可以支持应用层防火墙,并不是给内核打完补丁就万事大吉了,别忘了第2.8一节中曾经说过,不是只有Netfilter有模块,iptables工具也是有模块的,iptables工具就是通过这些模块来了解Netfilter模块的控制语法该如何下达,这些模块默认安

装在/lib64/xtables目录下，但如果你的操作系统是32位的Linux系统，那么，模块的安装路径将会是/lib/xtables目录，因为Netfilter内的应用层防火墙不是内置的，而是我们事后安装补丁加上去的。那么，iptables工具是否也应该额外安装一个补丁呢？答案当然是肯定的。

8.1 如何为iptables安装补丁

第7.3一节中曾经讲到过从http://l7-filter.clearfoundation.com/tracker/my_view_page.php下载Layer7的补丁文件，不过，这个补丁文件中并没有包含iptables的补丁，iptables的补丁文件可以到<http://sourceforge.net/projects/l7-filter/>网站下载，在撰写本书时，最新版本的补丁文件名称是netfilter-layer7-v2.22.tar.gz，请将其下载回来，并解压到/root/netfilter-layer7-v2.22路径下。

在该路径下，我们可以看到一个名为kernel-2.6.25-2.6.28-layer7-2.22.patch的文件，这是一个内核的补丁文件。经我测试，这个补丁文件无法与Red Hat Enterprise 6.0的内核兼容，因此，请不要使用这个补丁文件。此外，我们还可以看到一个/root/netfilter-layer7-v2.22/iptables-1.4.3forward-for-kernel-2.6.20forward的目录，这个目录下所保存的正是iptables的补丁文件，其中有libxt_layer7.c及libxt_layer7.man两个文件，前者是补丁文件，后者是man page。

有了iptables的补丁文件之后，还需要iptables的程序源代码，可以到Netfilter/Iptables的官方网站<http://www.netfilter.org/projects/iptables/files>下载iptables的源代码。在撰写本书时，iptables的最新版本是iptables-1.4.9.1，因此，本书示例使用iptables-1.4.9.1.tar.bz2这个补丁文件，在取得iptables的源代码及iptables补丁文件后，请将它解压缩到/root目录下，路径分别是/root/iptables-1.4.9.1及/root/netfilter-layer7-v2.22/iptables-1.4.3forward-for-kernel-2.6.20forward。

接下来将/root/netfilter-layer7-v2.22/iptables-1.4.3forward-for-kernel-2.6.20forward目录下的libxt_layer7.c及libxt_layer7.man两个文件复制到/root/iptables-1.4.9.1/extensions目录下，并在/root/iptables-1.4.9.1目录下运行./configure--with-ksource=/root/rpmbuild/BUILD/kernel-2.6.35.10命令，以设置iptables的编译参数，接着执行make及make install命令来编译及安装iptables工具，你可以参考以下安装流程。

```
[root@localhost /]# cd /root/iptables-1.4.9.1/extensions/
[root@localhost extensions]#
[root@localhost extensions]# cp /root/netfilter-layer7-v2.22/iptables-1.4.3forward-for-kernel-2.6.20forward/* .
[root@localhost extensions]# cd ..
[root@localhost iptables-1.4.9.1]# ./configure --with-ksource=/root/rpmbuild/BUILD/kernel-2.6.35.10/
```

```
[root@localhost iptables-1.4.9.1]# make
[root@localhost iptables-1.4.9.1]# make install
```

安装好的iptables工具会在/usr/local/sbin目录下，文件名为iptables，而iptables的模块则会放在/usr/local/libexec/xtables目录中，与RedHat、Fedora及CentOS所提供的iptables组件路径不同，但不必担心会使用到旧的iptables，因为在RedHat、Fedora及CentOS系统中，root的PATH默认如下：

```
[root@localhost xtables]# echo $PATH
/usr/lib64/qt-3.3/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

所以root在执行iptables命令时，系统会优先执行我们所编译的iptables工具。

8.2 Layer7模块识别应用层协议的原理

由于Layer7模块是应用层防火墙，因此Layer7模块通过检查数据包内所承载的数据内容来判断数据包属于哪种应用层协议的数据包，而Layer7模块匹配数据包内容的方法是使用正则表达式来进行匹配的，这些用来匹配应用层协议的数据我们称为模式(Pattern)，模式以文件形式保存，并保存在/etc/l7-protocols目录下。

等到安装好Layer7的模式之后，/etc/l7-protocols目录下就会存在一堆文件，而每个文件都描述着某个应用层的协议，而且只会有一种，也就是说，文件的数量与被定义的应用层协议是1:1的，这里以/etc/l7-protocols/pop3.pat文件为例来说明，至于模式该如何安装，下一节再来说明。pop3.pat文件内容如下：

```
pop3
^(\\+ok |-err )
```

为什么pop3.pat文件内容会是这样的呢？我们可以使用第1章介绍过的wireshark数据包分析工具来截取并检查POP3连接中的网络数据包内容即可了解。首先使用wireshark工具截取网络上传输中的POP3协议数据包，接着使用wireshark中的Follow TCP Stream功能，将一整条POP3连接中的数据包重新组合成原始的数据内容。

从图8-2可以看到Follow TCP Stream界面中的信息，就是POP3连接中所传输的数据内容，从这些数据可以看到，客户端与服务器端在三次握手之后，服务器端就会传输+OK Dovecot ready...字符串给客户端。在POP3协议的规范中如果连接正常，那么服务器端第一个应答给客户端的字符串一定以+OK开头，如果连接不正常，服务器端应答给客户端的第一个

字符串会以-ERR开头，因此，可以根据以上两个特征来判断连接是否是POP3协议的应用。



图8-2 POP3协议

阅读完以上内容之后，或许你在思考Layer7模块与第3.1.2一节第15部分所介绍的String模块有何不同之处。其实这两个模块的差异真的很大，下面从几个方面加以说明：

- 匹配范围：

String模块的匹配范围仅限于单一数据包内所承载的数据内容，而Layer7模块则是把经过防火墙的数据包复制一份下来，并且把属于同一条连接的数据包内容重新组合成分割之前的数据内容，最后再进行匹配操作，因此，Layer7模块所能匹配的精度比String模块要更高。此外，Layer7模块具有匹配长度限制，其默认长度是2KB，也就是说，Layer7默认只会跟踪每条连接的前2KB的内容。

- 匹配条件

String模块是以字符串为基础的匹配方法，而Layer7则是以正则表达式为基础来进行匹配，因此，Layer7模块的匹配精度较高，而且匹配方式更灵活。

从以上叙述来看，Layer7模块的易用性及适用性绝对远远超过String模块，但可惜的是，Layer7模块至今尚未成为内核内置的Netfilter模块。

8.3 安装Layer7模块的模式

可以到<http://sourceforge.net/projects/l7-filter/files/Protocol%20definitions/>下载最新的Layer7模块的模式定义文件，我下载的版本是l7-protocols-2009-05-28.tar.gz，下载回来后，将其解

压缩到/root目录下,然后在/etc/目录下建立Layer7模块的模式存放目录,目录名请设置为l7-protocols。接着将模式的定义文件复制到/etc/l7-protocols目录中即可。请参考以下操作流程:

```
[root@localhost ~]# cd /root/
[root@localhost ~]# tar -zxvf l7-protocols-2009-05-28.tar.gz
[root@localhost ~]#
[root@localhost ~]# mkdir /etc/l7-protocols
[root@localhost ~]#
[root@localhost ~]# cp l7-protocols-2009-05-28/protocols/* /etc/l7-protocols/
[root@localhost ~]#
```

2009-05-28这个版本的模式数据库总共支持多达F14种应用层协议的识别能力,我们可以到<http://l7-filter.sourceforge.net/protocols>查看模式数据库所能识别的一系列应用层协议,例如图8-3中列出了一些协议。其中第四个字段是该项应用层协议的识别程度,而识别程度是通过颜色来标示的,如图8-4即为识别程度的说明。建议你在使用某种应用层协议识别之前先参考这项数据,这样或许可以少花费很多测试时间,因为要用来匹配的模式本身可能就有问题。

Protocols					
The pattern <i>name</i> is what you must use when issuing l7-filter commands. The names below link to the pattern files. Select column headings to sort.					
bad third line in replaytv-ivs.pat					
wiki	name	speed	quality	group	notes
	100bao	○ ○	■	group	100bao - a Chinese P2P protocol/program - http://www.100bao.com
no	aim	● ●	■	group	AIM - AOL instant messenger (OSCAR and TOC)
no	aimwebcontent	○ ○	■	group	AIM web content - ads/news content downloaded by AOL Instant Messenger
no	applejuice	○ ○	■	group	Apple Juice - P2P filesharing - http://www.applejuice.net
no	ares	○ ○	■	group	Ares - P2P filesharing - http://aresgalaxy.sf.net
no	armagetron	● ●	■	group	Armagetron Advanced - open source Tron/snake based multiplayer game
no	battlefield1942	○ ○	■	group	Battlefield 1942 - An EA game
no	battlefield2	● ●	■	group	Battlefield 2 - An EA game
no	battlefield2142	○ ○	■	group	Battlefield 2142 - An EA game
no	bgp	○ ○	■	group	BGP - Border Gateway Protocol - RFC 1771
no	buff	○ ○	■	group	Buff - new mail notification
no	bittorrent	● ●	■	group	Bittorrent - P2P filesharing - publishing tool - http://www.bittorrent.com

图8-3 支持协议列表

Quality

The "quality" gives a rough idea of how well the pattern works. This is a conglomerate of what variety of situations the pattern has been tested and (4) what fraction of ide

- Great: Works
- Good: Works as far as we know
- Ok: Probably works
- Marginal: Might work, might not
- Poor: Probably doesn't work

图8-4 协议识别程度

8.4 如何使用Layer7模块

Layer7模块的使用方法与一般的模块并没有太大差别，如果真的要寻找不同之处，大概是Layer7模块拥有额外的配置文件模式，此外就没有其他特殊之处了，Layer7模块的使用语法如下：

```
iptables -t filter -m layer7 --l7proto http -j REJECT
```

其中的--l7proto-http要求Layer7模块使用/etc/l7-protocols/http.pat文件的内容作为匹配条件。另外，使用Layer7模块时有一个小地方需要予以特别注意，请不要将与方向有关的条件与Layer7模块放在一起使用，因为这会导致Layer7无法完全获得一条连接中前2KB的完整数据内容，而导致Layer7模块无法判断应用层协议的问题。例如：

规则一：

```
iptables -t filter -A FORWARD -i eth0 -m layer7 --l7proto http -j REJECT
```

规则二：

```
iptables -t filter -A FORWARD -o eth0 -m layer7 --l7proto http -j REJECT
```

规则三：

```
iptables -t filter -A FORWARD -m layer7 --l7proto http -j REJECT
```

以上三条规则中，规则一及规则二都包含与方向性有关的条件，如-i与-o，因此这两条规则将无法正确判断http通讯协议，只有规则三可以正确过滤掉http通信协议。

由以上说明可以推测，如果要把Layer7模块运用于网关式防火墙大概不会有太大问题，如果要将Layer7模块运用于单机防火墙那就可能会有问题了，因为在单机防火墙上没有任何一个链可以同时接触到INPUT及OUTPUT数据包，如此Layer7在单机防火墙上将无法发挥其功效，而且这一部分的相关信息在Layer7的文件中并没有特别说明。一开始我也觉得奇怪，如下示例：

规则一：

```
iptables -A INPUT -m layer7 --l7proto pop3 -j REJECT
```

规则二：

```
iptables -A OUTPUT -m layer7 --l7proto pop3 -j REJECT
```

一开始，我无论是单独使用规则一还是规则二，都无法成功地拦截本机往外的POP3访问操作，后来我突然想到，如果同时加上两个规则会怎么样呢？结果竟如预期的一样成功。

了。后来又进一步发现，只要在OUTPUT链中随便放入一条包含Layer7的规则即可。例如，先执行如下规则一，在这样的规则下，在本机上不管是对外访问POP3还是HTTP服务，都可以正常使用，不会受到任何来自Layer7模块的拦截，但如果再加上如下规则二，神奇的事情发生了！不管POP3还是HTTP协议都可以正常拦截下来：

规则一：

```
iptables -A INPUT -m layer7 --l7proto pop3 -j REJECT
iptables -A INPUT -m layer7 --l7proto http -j REJECT
```

规则二：

```
iptables -A OUTPUT -m layer7 --l7proto telnet -j REJECT
```

为什么会这样？Layer7模块的说明文件并没有对此进行说明，根据我的推测：“只要Layer7模块能够接触一条连接两个方向的所有数据包，Layer7模块就可以将数据包整合到缓冲区中，进而得到足以用来被Layer7模块识别的数据”。因此，这里在OUTPUT链中又加入了一条与POP3及HTTP完全无关的规则，这也足以使INPUT链中的两条规则发挥作用。

8.5 Layer7模块使用示例说明

虽然Layer7模块的语法与一般模块没有太大区别，但在实际应用中却有极大差异，先以下列示例来说明：

示例8.1 构建一个网关防火墙，并设置只有DNS及HTTP通信协议可通过

在非应用层防火墙之中遇到这样的问题时，我们可能会这样做：

```
#!/bin/bash

iptables -F

iptables -t filter -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t filter -A FORWARD -i eth1 -o eth0 -p tcp --dport 80 -j ACCEPT
iptables -t filter -A FORWARD -i eth1 -o eth0 -p udp --dport 53 -j ACCEPT
iptables -t filter -A FORWARD -j DROP
```

如果使用应用层防火墙机制，我们又该如何改写规则呢？通常第一次接触Layer7的使用者会将规则改写为：

```
1. #!/bin/bash
2. iptables -F
3. iptables -t filter -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
4. iptables -t filter -A FORWARD -m layer7 --l7proto http -j ACCEPT
5. iptables -t filter -A FORWARD -m layer7 --l7proto dns -j ACCEPT
6. iptables -t filter -A FORWARD -j DROP
```

不过以上规则并不尽如人意，因为我们现在使用的是Layer7模块，而Layer7模块将一条连接前2KB的数据偷偷复制一份保存在内存中，然后再使用正则表达式去匹配内存中的内容，由此判断连接内容所使用的通信协议。因此，Layer7要判断一条连接所使用的通信协议，通常需要收集够多的数据包，才足以拼凑出应用层协议的特征。当第一个HTTP协议的数据包送出时，会先匹配到第4条规则，此时，Layer7模块会将这个数据包的内容复制到内存中，接着，这个数据包会匹配到第6条规则而被丢弃，因此，Web服务器根本就无法收到客户端的服务请求数据包。所以在此规则下，HTTP协议将无法正确穿过防火墙。

相对于HTTP协议，DNS协议就幸运多了。Layer7模块可根据第一个数据包内所承载的数据内容判断出DNS通信协议，因此，当第一个DNS查询数据包送入防火墙时，将会符合第5条规则而被防火墙放行，接着，该条连接将进入ESTABLISHED连接状态，在以上规则中，DNS协议将可以正常工作，而HTTP协议则无法正常工作。

了解了Layer7模块的用法特性后，我们该如何使用Layer7模块呢？这取决于你采取的防火墙管理策略。

● 放行所有数据包，再封锁要禁用的应用

如果采取这种较宽松的管理策略，那么Layer7模块在使用上大概不会有太大问题。例如，公司内不允许使用者使用skype、MSN、FTP、EDONKEY及POP3，只需以下规则即可满足我们的需要：

```
#!/bin/bash

iptables -F

iptables -t filter -A FORWARD -m layer7 --l7proto ftp -j DROP
iptables -t filter -A FORWARD -m layer7 --l7proto edonkey -j DROP
iptables -t filter -A FORWARD -m layer7 --l7proto pop3 -j DROP
iptables -t filter -A FORWARD -m layer7 --l7proto skypeout -j DROP
iptables -t filter -A FORWARD -m layer7 --l7proto skypetoskype -j DROP
iptables -t filter -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

iptables -t filter -A FORWARD -i eth0 -o eth1 -j DROP
```

不过，如果要考虑到防火墙性能的话，这样的规则还不够周密，为了降低应用层防火墙所造成的性能负担，可以试着将规则改写为：

```
1. #!/bin/bash
2. iptables -F
```

```
3. iptables -N LAYER7
```

```
#=====<< FORWARD Chain >>=====
```

```
4. iptables -A FORWARD -m connbytes \
    --connbytes-dir both \
    --connbytes-mode bytes \
    --connbytes 0:2500 -j LAYER7
```

```
5. iptables -A FORWARD -m connmark --mark 1 -j DROP
```

```
6. iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
7. iptables -A FORWARD -i eth0 -o eth1 -j DROP
```

```
#=====<< Layer7 Chain >>=====
```

```
8. iptables -A LAYER7 -m layer7 --l7proto msnmessenger \
    -j CONNMARK --set-mark 1
```

```
9. iptables -A LAYER7 -m layer7 --l7proto skypeout -j CONNMARK --set-mark 1
```

```
10. iptables -A LAYER7 -m layer7 --l7proto skypetoskype \
    -j CONNMARK --set-mark 1
```

```
11. iptables -A LAYER7 -m layer7 --l7proto ftp -j CONNMARK --set-mark 1
```

```
12. iptables -A LAYER7 -m layer7 --l7proto edonkey -j CONNMARK --set-mark 1
```

```
13. iptables -A LAYER7 -m layer7 --l7proto pop3 -j CONNMARK --set-mark 1
```

这中间最大的差别在于我运用了用户定义的链机制，并在第4条规则运用connbytes模块，将每条连接的前2.5KB数据导入到用户定义的链之中进行Layer7的匹配操作。如果符合规则，我们使用CONNMARK模块在该条连接上将ctmark值设置为1，而当这条连接所传输的数据超过2.5KB之后，该连接的数据包就不会再进入用户定义的链之中，如此即可降低无效的规则匹配次数，进而减少额外的性能负担，最后被设置ctmark值为1的数据包将会在第5条规则被丢弃掉。

在这个示例中，虽然不被允许的连接在被封锁之前，会有一小部分的数据被传输出去(约2.5KB左右)，但这是可被接受的，因为这个示例的管理策略本来就是较为宽松的策略。

● 封锁所有数据包，再放行被允许的应用

如果要采取较为严格的管理策略，那么，在使用Layer7模块时将产生极大困扰。例如，公司内只允许使用者使用HTTP、HTTPS、MSN及DNS，防火墙的规则便不能写成如下形式：

```
#!/bin/bash
iptables -F
iptables -A FORWARD -m state --state ESTABLISHED -j ACCEPT
iptables -A FORWARD -m layer7 --l7proto http -j ACCEPT
iptables -A FORWARD -m layer7 --l7proto ssl -j ACCEPT
iptables -A FORWARD -m layer7 --l7proto msnmessenger -j ACCEPT
iptables -A FORWARD -j DROP
```

前面讨论过不能按这种方式编写规则的原因，那么，应该如何修改呢？

```
1. #!/bin/bash

2. iptables -F
3. iptables -N LAYER7

#=====<< FORWARD Chain >>=====
4. iptables -A FORWARD -m connbytes \
    --connbytes-dir both \
    --connbytes-mode bytes \
    --connbytes 0:2500 -j LAYER7

5. iptables -A FORWARD -m state --state ESTABLISHED,RELATED \
    -m connmark --mark 1 -j ACCEPT

6. iptables -A FORWARD -j DROP

#=====<< Layer7 Chain >>=====
7. iptables -A LAYER7 -m layer7 --l7proto http -j CONNMARK --set-mark 1
8. iptables -A LAYER7 -m layer7 --l7proto ssl -j CONNMARK --set-mark 1
9. iptables -A LAYER7 -m layer7 --l7proto msnmessenger \
    -j CONNMARK --set-mark 1
10. iptables -A LAYER7 -m layer7 --l7proto dns -j CONNMARK --set-mark 1
11. iptables -A LAYER7 -j ACCEPT
```

这个示例其实和前面两个例子相差不大，只是反向使用而已。我们使用第4条规则将每一条连接的前2.5KB数据导入到用户定义的链，并使用Layer7模块来判断连接所使用的通信协议，如果协议是HTTP、HTTPS、MSN及DNS，就将该连接的ctmark值设置为1，接着，再使用第5条规则来判断数据包的ctmark值，为1者就放行，否则，就根据第6条规则将数据包丢弃。

以上示例虽然可以达到我们的要求，但结果还是不尽完美，因为不被允许使用的连接在被封锁之前，仍会有少部分的数据会被发送出去，这对较为严格的管理策略而言自然是不能被接受的，例如，使用者依然可以使用FTP通信协议来连接因特网，一样可以浏览FTP服务器上的目录及文件，也可以正常下载文件，只要该文件不超过2.5KB，就不会被封锁。

8.6 结合使用包过滤器与Layer7模块

如果单独使用Layer7模块，在较为严格的管理策略下，即使不被允许使用的通信协议，其每一条连接前面的部分数据依然可以穿过防火墙。为了更好地管理防火墙，可以将包过滤器(Packet Filter)与Layer7模块结合在一起使用。下面改写前一个例子“公司内只允许使用者使用HTTP、HTTPS、MSN及DNS”，如下：

```

1. #!/bin/bash

2. iptables -F

3. iptables -N LAYER7

#=====<< FORWARD Chain >>=====
4. iptables -A FORWARD -m connbytes \
    --connbytes-dir both \
    --connbytes-mode bytes \
    --connbytes 0:2500 -j LAYER7

5. iptables -A FORWARD -m connmark --mark 1 \
    -m state --state ESTABLISHED,RELATED -j ACCEPT

6. iptables -A FORWARD -i eth1 -o eth0 -p tcp --syn \
    -m multiport --dports 80,443,1863 \
    -m state --state NEW -j ACCEPT

7. iptables -A FORWARD -i eth1 -o eth0 -p udp --dport 53 -j ACCEPT
8. iptables -A FORWARD -j DROP

#=====<< Layer7 Chain >>=====
9. iptables -A LAYER7 -m layer7 --l7proto http -j CONNMARK --set-mark 1
10. iptables -A LAYER7 -m layer7 --l7proto ssl -j CONNMARK --set-mark 1
11. iptables -A LAYER7 -m layer7 --l7proto msnmessenger \
    -j CONNMARK --set-mark 1
12. iptables -A LAYER7 -m layer7 --l7proto dns -j CONNMARK --set-mark 1
13. iptables -A LAYER7 -m state --state ESTABLISHED,RELATED -j ACCEPT

```

这里以HTTP通信协议为例来解释这个规则的内容。当第一个HTTP协议的数据包送入时，这个数据包会先因为符合第3条规则而被送入LAYER7用户定义的链，但这个数据包并不符合LAYER7链中的任何规则，因此，这个数据包将返回第5条规则继续执行匹配操作。接着，这个数据包会因符合第6条规则的条件而被防火墙放行，因此，这条连接即进入ESTABLISHED状态；在第二个数据包送入时，这个数据包一样会被先送入到LAYER7链，但这时Layer7模块可能尚未收集到足够多的数据来判断该连接是否是HTTP协议的连接，因此，这个数据包会符合第13条规则。接着第三个数据包送入时，这个数据包一样会被送入LAYER7链，这时我们假设Layer7模块已经收集足够多的数据包来判断连接是HTTP协议的连接，那么，这个数据包将符合第9条规则，因此，这个连接的所有数据包都将被标记为ctmark等于1。

接下来，在该连接所送入的数据包数量(其内所承载的数据长度总和)未到达第4条规则的上限之前，所有数据包的处理步骤都与上面的步骤相同；当送入的数据包数量(其内所承载的数据长度总和)超过第4条规则所设置的上限时，其后该连接的所有数据包将因为符合第5条规则而被防火墙放行。也就是说，如果某条连接没有被标记ctmark为1，数据包将会因为

符合第8条规则而被丢弃。

```
#!/bin/bash

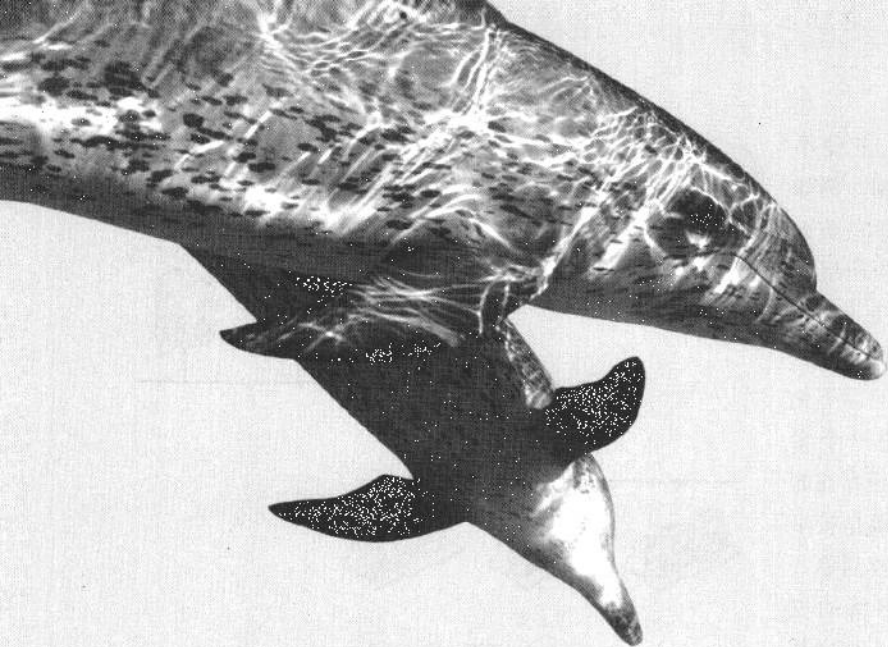
iptables -F

iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p tcp --syn \
    -m multiport --dports 80,443,1863 \
    -m state --state NEW -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p udp --dport 53 -j ACCEPT
iptables -A FORWARD -j DROP
```

或许你会问，为何需要这样去设置？我们先来想想上述这个例子有何缺点？在正常的使用中，通常应该没有什么太大问题，但如果因特网上有一台SMTP服务器使用的不是标准的端口25而是端口80，那么，企业内的使用者将可以通过防火墙上所开启的端口80来访问这台SMTP服务器。但如果我们通过Layer7模块来判断应用层的协议，将可以避免以上问题。特别是有为数不少的木马程序会使用防火墙所开启的端口80来传输数据，当这些木马程序不使用HTTP协议来传输数据，就会被防火墙拦下。当然，如果木马程序使用HTTP协议来传输数据，我们也只能跟这些被窃取的数据说再见了。

8.7 小结

本章主要介绍如何将Netfilter/Iptables提升为应用层防火墙，以弥补原有Netfilter/Iptables的不足之处，以使得Netfilter/Iptables防火墙得以媲美商用防火墙。本章的内容十分重要，请务必真正理解其中的内容。



Linux

| 第9章 | 透明式防火墙

9



前面已经介绍过多种不同的企业网络防御机制，例如Netfilter/Iptables中的单机防火墙、网关式防火墙、应用层防火墙，以及Squid代理的HTTP代理、反向代理等，虽然这些防御机制都有优点，但也不免存在缺点。在此假想一个案例：图9-1中有一个已经存在的网络环境，其中防火墙是较早期的产品，所以功能较简单，也不足以保障企业网络的安全，但防火墙上却有不少极其复杂的规则，请问你该如何提升现有网络的安全性？

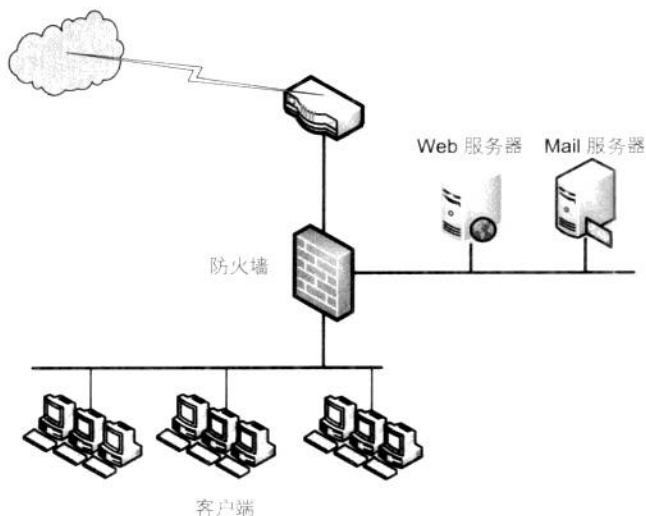


图9-1 现有的网络环境（一）

面对这样的问题，大多数人都会通过更换新的防火墙产品来加以解决，但这将是一项浩大的工程，因为仅是规则的迁移就令人头疼不已；另外的策略则是将规则改为图9-2结构，也就是在原有防火墙之前串接另一个防火墙FW，但我们必须改变原有网络的路由结构。

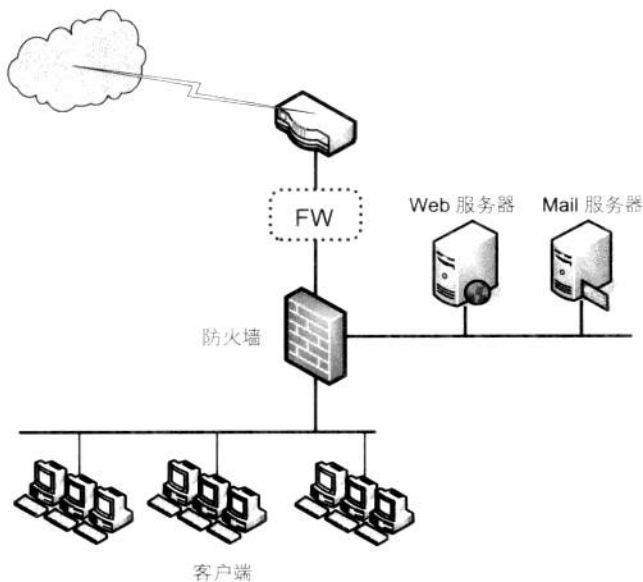


图9-2 增强后的网络环境（一）

再列举一个例子，假设有一家公司的网络结构如图9-3所示，从图中我们可以看到，这家公司内部的网络实体层并没有采取任何隔离措施，出于安全方面的考虑，老板要求将会计部门的计算机与其他部门的计算机隔离开，如果你是这家公司的IT人员，你会如何完成这项任务呢？

通常都会将会计部单独分成一个VLAN，然后再使用防火墙把这个VLAN连接到公司的网络上，如图9-4所示。通过这个防火墙，我们就可以对进出会计部门的数据包进行过滤操作，表面上看起来似乎很容易，但如果稍微思考一下细节，就会知道这是多么麻烦的一件事！原因如下：

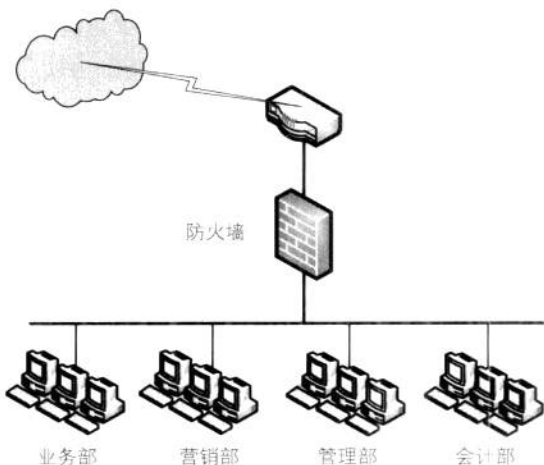


图9-3 现有的网络环境 (二)

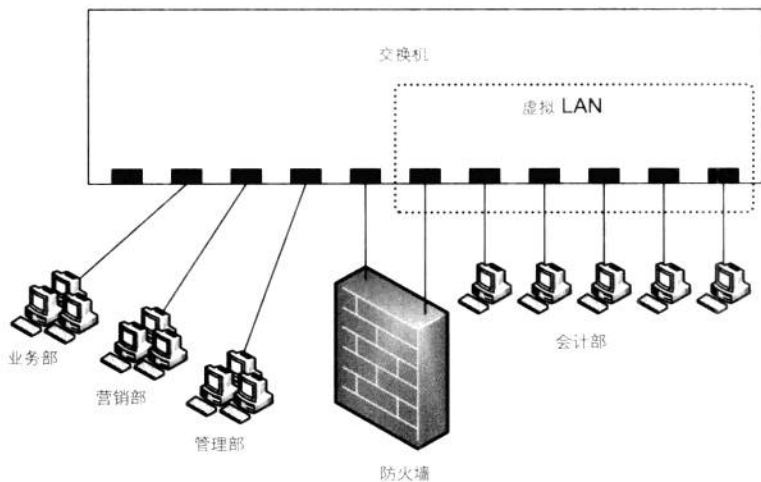


图9-4 使用VLAN隔离一组指定的主机

- 由于一般的防火墙都采用路由模式(Routing Mode)，因此，防火墙上的两个网络接口必须是两个不同的IP网络，公司内的路由必须重新设置。
- 会计部门所有计算机的网络配置必须全部更改，因此，需要为会计部门额外部署一台DHCP服务器。

- 原来一些与IP有关的设置都需要重新定义，如原来防火墙上的规则、数据库访问的ACL以及应用程序访问的ACL等。由以上例子可发现，这将是一项浩大的工程。

以上解决办法并没有什么太大问题，主要问题在于防火墙是在路由模式下工作，因而导致了上述不便，不过，以上两个示例的问题其实很简单，如图9-2及图9-5所示，只要让防火墙工作在透明模式(Transparent Mode)，那么所有问题都会迎刃而解。

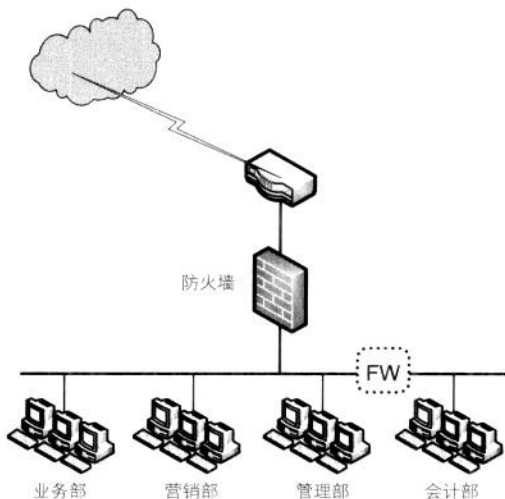


图9-5 改进后的网络环境(二)

9.1 何谓桥接模式

我想，大家应该都很熟悉路由模式是什么，但是桥接模式又是什么呢？刚入门的IT人员可能对这种模式不够熟悉，因为在现在网络实体层的部署上已经很少使用到网桥(Bridge)这种设备了。网桥是部署在OSI 7层中的第二层的网络设备，因此，网桥可以识别到MAC地址。网桥设具有以下三个特点：

- 转发广播数据包：以图9-6为例，如果计算机A发出一个以太网广播，当网桥收到广播包之后，就会将数据包“复制”成两份，一份送往计算机B所在的实体网段，另外一份送往计算机C、D所在的实体网段。也就是说，网桥不会隔绝以太网的广播数据包。
- 隔离相同实体网段的单播：以图9-6为例，如果计算机D传输数据给计算机C，这时网桥会发现数据包的发送端与接收端位于同一个实体网段，因此，网桥不会去处理这个数据包，所以计算机A、B也就看不见这个数据包。

- 不同实体网段间的单播，网桥只会将数据包转发到相关的实体网段上：以图9-6为例，如果计算机A发送数据给计算机B，在网桥收到这个数据包后，网桥会判断出这个数据包接收者所在的实体网段是什么，因为网桥只会把数据包复制

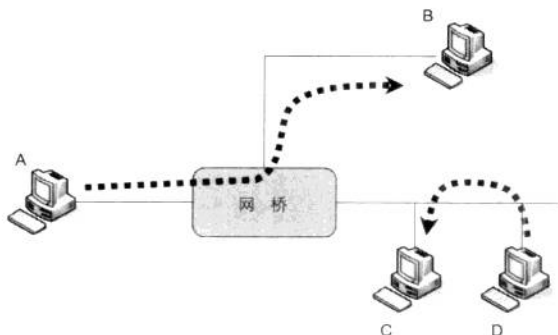


图9-6 网桥的特性

一份，然后送往计算机B所在的实体网段，所以计算机C、D无法看到这个数据包。

由于网桥属于第二层的网络设备，因此，网桥设备并不需要像路由器一样需要在网络接口上设置IP。不过如果要远程管理网桥，就必须在网桥接口上设置IP。

9.2 何谓透明式防火墙

透明式防火墙(Transparent Firewall)又称为桥接式防火墙(Bridge Firewall)，简单来说，在网桥设备上加入防火墙功能后，就成为透明式防火墙。如果有这样的防火墙设备，那在防火墙的部署上将会有什么改变呢？我想，只能用“自由”这个词来形容这样的变化吧！下面列出透明式防火墙的优点：

- 部署能力强：回顾图9-2及图9-5，如果图中额外加入的防火墙FW是一个透明式防火墙，那还需要改动原来的网络环境吗？答案是：完全不需要改变原来的网络环境。因为透明式防火墙属于第二层网络设备，不会存在任何路由上的问题，因此，透明式防火墙可以部署在任何你希望有防火墙的地方。
- 隐蔽性好：由于透明式防火墙可以不用设置IP，因此，黑客很难发现透明式防火墙的存在。
- 安全性高：由于透明式防火墙可以不设置IP，因而也增加黑客直接攻击透明式防火墙的难度，因为黑客没有直接攻击的目标。

9.3 构建透明式防火墙

了解了透明式防火墙之后，接下来就可以实际尝试来构建透明式防火墙。我们将其分为网桥及防火墙两部分来讨论：

9.3.1 使用Linux构建网桥

Linux在版本2.4.18中引入了网桥机制,所以不必执行内核补丁的操作,但仍需一个User Space管理工具。请使用以下命令来安装网桥的管理工具。

```
yum install bridge-utils
```

我们试着以图9-7为基础来构建这个网络环境。在图中,客户端A与客户端B两台计算机的网络配置是属于相同网段的主机,而其中有一台Linux网桥的计算机将左右两侧的实体网络隔开,但这样一来,客户端A与客户端B将无法进行网络

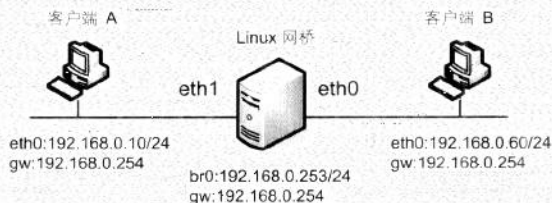


图9-7 Linux网桥

通信,但我们可以使用以下网络配置及命令,将Linux网桥计算机变成一台网桥设备,这样客户端A与客户端B就可以进行网络通信了。

网络配置

● /etc/sysconfig/network-scripts/ifcfg-eth0

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
HWADDR=XX:XX:XX:XX:XX:XX
```

● /etc/sysconfig/network-scripts/ifcfg-eth1

```
DEVICE=eth1
ONBOOT=yes
BOOTPROTO=static
HWADDR=XX:XX:XX:XX:XX:XX
```

命令

```
1. #!/bin/bash
2. echo 1 > /proc/sys/net/ipv4/ip_forward
3. brctl addbr br0
4. brctl addif br0 eth0
5. brctl addif br0 eth1
6. ifconfig br0 up
```

我们可以将命令部分编写成一个Shell脚本,这个脚本的内容如下:

- 第2行: 启用ip_forward功能,以便启用Linux数据包转发功能。例如,从eth0接口进入

的数据包可以转发到eth1接口输出，或者从eth1进入，从eth0送出。

- 第3行：使用brctl命令来新增一个虚拟的网桥接口，其中br0即为虚拟接口的名称，名称部分可以随意设置。
- 第4、5行：设置将eth0及eth1合并到br0接口上，如此一来，eth0及eth1就是br0这个网桥设备的两个端口。也可以将多张网卡合并成一个网桥设备。
- 第6行：启动新增的br0接口。

完成以上流程后，使用ifconfig命令可看到如图9-8的配置，其中br0、eth0及eth1都没有设置IP，这是因为网桥设置的网络接口本来就不需要设置IP。接下来，神奇的事情发生了，客户端A与客户端B已经可以正常通信了。

至于网桥上网络配置的部分，如果不嫌麻烦的话，基于安全方面的考虑，建议不要在网桥上设置IP，因为我们将Linux完全当成防火墙来使用。如果要在网桥上设置IP，只需将启动网桥的脚本稍微修改一下即可，如下：

```
[root@localhost ~]# ifconfig
br0      Link encap:Ethernet  HWaddr 00:60:6E:00:F1:7D
         inet6 addr: fe80::260:6eff:fe00:f17d/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:1496 errors:0 dropped:0 overruns:0 frame:0
         TX packets:111 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:0
         RX bytes:122190 (119.3 KiB)  TX bytes:23973 (23.4 KiB)

eth0     Link encap:Ethernet  HWaddr 48:5B:39:A7:AD:6B
         inet6 addr: fe80::4a5b:39ff:fea7:ad6b/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:2763 errors:0 dropped:0 overruns:0 frame:0
         TX packets:1034 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:409051 (399.4 KiB)  TX bytes:113124 (110.4 KiB)
         Interrupt:37 Base address:0xe000

eth1     Link encap:Ethernet  HWaddr 00:60:6E:00:F1:7D
         inet6 addr: fe80::260:6eff:fe00:f17d/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:1549 errors:0 dropped:0 overruns:0 frame:0
         TX packets:1539 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:89899 (87.7 KiB)  TX bytes:275367 (268.9 KiB)
```

图9-8 网桥下的网络接口

```
7. #!/bin/bash
8. echo 1 > /proc/sys/net/ipv4/ip_forward
9. brctl addbr br0
10. brctl addif br0-eth0
11. brctl addif br0-eth1
12. ifconfig br0 192.168.0.253 netmask 255.255.255.0 up
13. ip route add default via 192.168.0.254
```

其中第12行是br0接口的IP设置，而第13行则是网桥设备上设置的默认网关。

1. Linux 网桥的管理

可以使用brctl命令来管理网桥，下面列出几个常用命令：

- 查看系统上的网桥接口：

```
[root@localhost ~]# brctl show
bridge name      bridge id        STP enabled  interfaces
br0              8000.00606e00f17d  no           eth0
                                                         eth1
```

从输出信息可以看到，目前系统上有一个br0的网桥接口，如果有多个接口，也会一并显示出来。而br0接口上的STP(Spanning tree protocol)目前并未启动，至于什么是STP？稍后再来讨论。在最后可以看到，br0接口是由eth0及eth1组成的。

● 新增网桥接口：

```
[root@localhost ~]# brctl show
bridge name      bridge id                STP enabled        interfaces
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# brctl addbr br0
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# brctl show
bridge name      bridge id                STP enabled        interfaces
br0              8000.0000000000000       no
```

可以使用brctl addbr命令来增加网桥接口，至于接口的名称则由使用者自行确定，从以上输出信息可以看到，br0接口当前不包含任何网卡。

● 将网卡添加到网桥接口：

```
[root@localhost ~]# brctl show
bridge name      bridge id                STP enabled        interfaces
br0              8000.0000000000000       no
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# brctl addif br0 eth0
[root@localhost ~]#
[root@localhost ~]# brctl show
bridge name      bridge id                STP enabled        interfaces
br0              8000.485b39a7ad6b        no                  eth0
[root@localhost ~]#
[root@localhost ~]# brctl addif br0 eth1
[root@localhost ~]#
[root@localhost ~]# brctl show
bridge name      bridge id                STP enabled        interfaces
br0              8000.00606e00f17d        no                  eth0
                                                                eth1
```

将网卡加入到网桥接口的命令是brctl addif。例如命令brctl addif br0 eth0可将eth0接口添加到br0接口中，且同一个网桥接口上可以同时包含两个以上的网卡。

● 从网桥接口上删除网卡

```
[root@localhost ~]# brctl show
bridge name      bridge id                STP enabled        interfaces
br0              8000.00606e00f17d        no                  eth0
                                                                eth1
```

```
[root@localhost ~]#
[root@localhost ~]# brctl delif br0 eth0
[root@localhost ~]#
[root@localhost ~]# brctl show
bridge name      bridge id                STP enabled    interfaces
br0               8000.00606e00f17d        no             eth1
```

可以使用命令brctl delif br0 eth0将网卡从网桥接口删除。意指从br0接口删除eth0。

● 删除已经存在的网桥接口

```
[root@localhost ~]# brctl show
bridge name      bridge id                STP enabled    interfaces
br0               8000.00606e00f17d        no             eth1
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# brctl delbr br0
[root@localhost ~]#
[root@localhost ~]# brctl show
bridge name      bridge id                STP enabled    interfaces
[root@localhost ~]#
```

可以使用brctl delbr br0删除已经存在的网桥接口，意即要删除br0接口。另外，在删除网桥接口时，并不需要事先删除该网桥接口下的网卡。

2. 使用网桥设备时的注意事项

每当在网络实体上使用网桥设备时，都必须注意广播风暴(Broadcast Storm)问题，那什么是广播风暴呢？我们以图9-9为例来解释广播风暴的成因。图中网桥设备上有三个网络实体层连接，如果能确保这三个网络实体层之间都不会有实体层上的直接连接，那就不会产生广播风暴的问题；但如果我们不小心把其中某两个实体网络连接在一起，就会立即产生广播风暴。

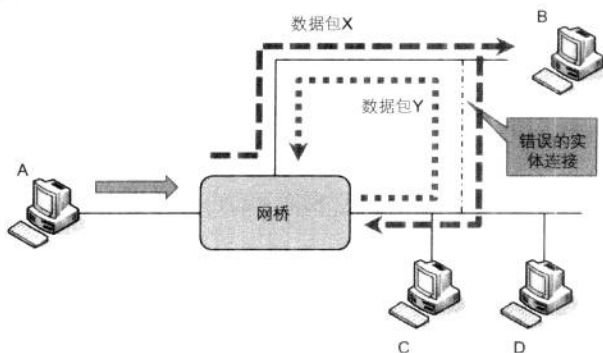


图9-9 广播风暴的产生

例如在图中，我们故意加入一条“错误的实体连接”，如图中的计算机A送出广播数据包，这个数据包进入网桥之后随即会被复制成两个数据包(将之命名为X及Y)，并且分别从另外两个端口送出。数据包X除了计算机B收到之外，这个数据包又被从网桥的另外一个端口送入，数据包送入之后，随即又被复制成两份，分别从网桥的另外两个端口送出。除了数据包X会产生这样的问题之外，数据包Y也会产生相同的问题，如此周而复始，网络上就会出现大量广播数据包，因而称之为广播风暴，广播风暴最后会使得整个网络瘫痪，甚至因为流量太大而

导致网桥本身或其他网络设备宕机。

那么可以采用什么方法来预防广播风暴的产生呢？当然是有的，网桥设备通常都会支持STP(Spanning Tree Protocol)，而STP就是专门为了解决广播风暴而设计的，且Linux网桥也支持这样的协议，只是默认没有启用而已。启用Linux网桥STP的方法如下：

```
[root@localhost ~]#
[root@localhost ~]# brctl show
bridge name      bridge id      STP enabled    interfaces
br0               8000.00606e00f17d  no             eth0
                  8000.00606e00f17d  no             eth1

[root@localhost ~]#
[root@localhost ~]# brctl stp br0 on
[root@localhost ~]#
[root@localhost ~]# brctl show
bridge name      bridge id      STP enabled    interfaces
br0               8000.00606e00f17d  yes            eth0
                  8000.00606e00f17d  yes            eth1

[root@localhost ~]#
[root@localhost ~]# brctl stp br0 off
[root@localhost ~]#
[root@localhost ~]# brctl show
bridge name      bridge id      STP enabled    interfaces
br0               8000.00606e00f17d  no             eth0
                  8000.00606e00f17d  no             eth1
```

其中brctl stp br0 on即是启用br0接口的STP，我们可以从brctl show br0的输出信息看到这个结果。如果要关闭STP，可以使用brctl stp br0 off命令。

9.3.2 Netfilter在Layer3及Layer2的工作逻辑

在此之前，我们都知道Netfilter工作于OSI 7层的第3层，但其实Netfilter也可在OSI 7层的第2层工作，只是这项特殊功能没有内置在早期的Kernel 2.4版本中，如果要使用这样的特殊功能，就要自己动手打补丁；还好这项功能现在已经内置在Linux之中，我们只需要确认所使用的内核是否有启用这个功能即可，检查方法如下：

```
[root@localhost /]# grep -i 'config_netfilter_advanced=y' /boot/config-$(uname -r)
CONFIG_NETFILTER_ADVANCED=y
```

如果你所使用的是Red Hat的Linux系统，那么这项功能默认是启用的，但如果因特殊需要而重新编译内核，请在执行make menuconfig之后，在选择菜单中的Networking support | Networking options | Network packet filtering framework(Netfilter)下，找到一个名为Advanced netfilter configuration的功能，将其选取并重新编译内核即可。

图9-10就是Netfilter在第2层及第3层的完整结构图。为便于印刷，本书将图旋转了90°，你可以将其放正来看。图中下半部分背景颜色较深的范围是第2层，上半部分是第3层，此外，我们从左到右把图划分为五部分，分别是PREROUTING、INPUT、FORWARD、

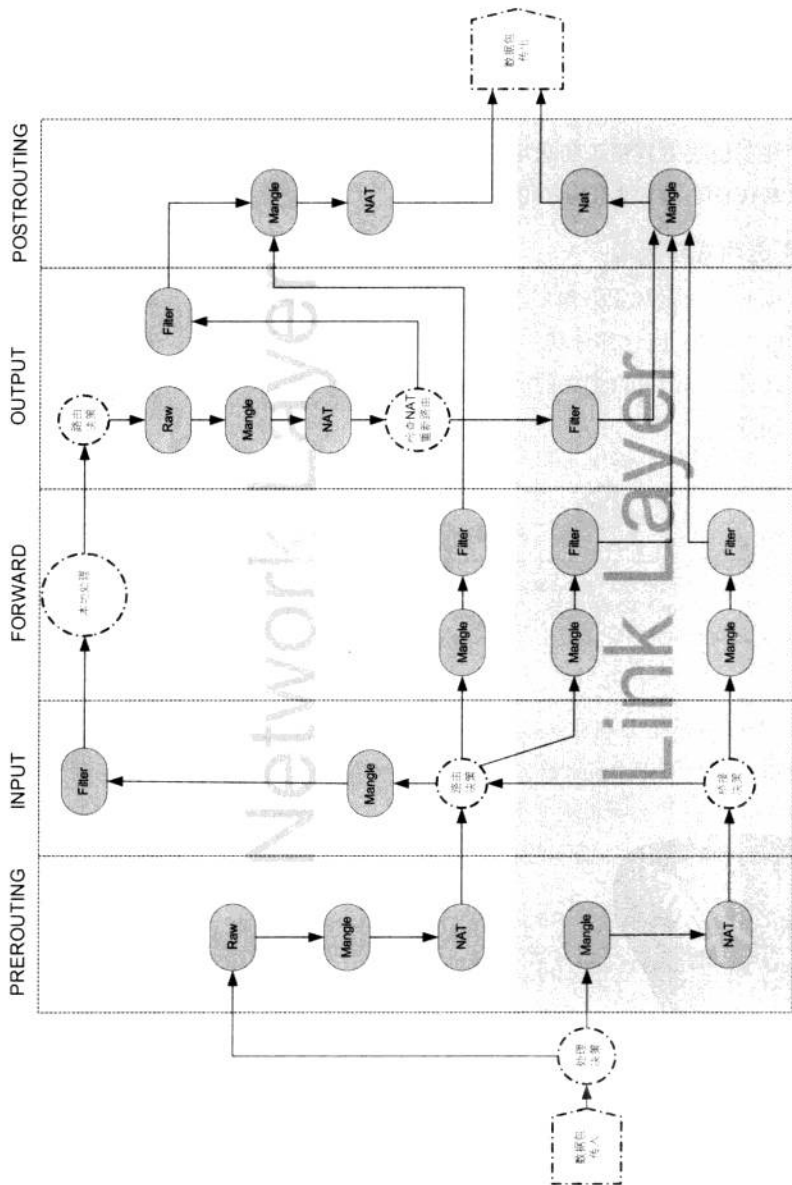


图9-10 Netfilter L2/L3的完整结构图

另外,对于链的部分则没有额外增加。以FORWARD链为例,虽然在第3层及第2层内部有一个Filter表的FORWARD链,但这两个链(两块内存空间)其实是同一个。例如,执行以下命令:

```
iptables -t filter -A FORWARD -p icmp -j DROP
```

这条Netfilter的规则将会同时作用于第2层级第3层的环境中,也就是说,如果我们使一台装有Linux系统的计算机同时支持路由模式及桥接模式,不管数据包以路由模式的方式穿过防火墙,还以桥接方式穿过防火墙,都会受到这条规则的限制,因此,在第2层及第3层所看到的FORWARD链其实都有同一份内容。

1. 使用透明式防火墙

了解了Netfilter在第2层及第3层的工作逻辑之后,接下来开始讨论在透明式防火墙执行Netfilter规则的方式。但这部分其实没什么好讨论的,因为除了内置匹配方法中的-i及-o语法会无法使用之外,我们可以把之前所学的所有Netfilter规则都套用进来。以图9-11为例来进一步说明这个问题:

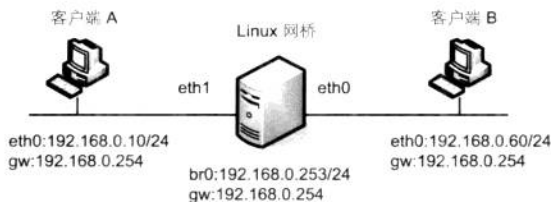


图9-11 透明式防火墙示例

如果在Linux网桥上分别测试以下两条规则,可以发现即使规则A运行在Linux网桥上,我们依然可以在客户端B上ping通客户端A,但如果把规则换成B,在客户端B上就无法ping到客户端A了,这是怎么回事呢?其实问题就出在目前Netfilter是工作在第2层位置,因此,Netfilter只能看到网桥接口,而无法识别到以太网的接口。

```
规则A: iptables -A FORWARD -i eth0 -o eth1 -p icmp -j DROP
```

```
规则B: iptables -A FORWARD -p icmp -j DROP
```

如果想要在第2层的位置上识别到以太网接口,可以通过一个叫physdev的模块来解决这个问题。如下规则是指:凡由eth0接口进入的icmp数据包,全部丢弃掉。

```
iptables -A FORWARD -m physdev --physdev-in eth0 -p icmp -j DROP
```

physdev模块的其他参数

● --physdev-in/--physdev-out

匹配数据包进出网桥的实体接口。例如，使用以下规则来匹配数据包进出网桥的方向。

```
iptables -A FORWARD -m physdev --physdev-in eth0 --physdev-out -p icmp -j DROP
```

● --physdev-is-in/--physdev-is-out

匹配数据包进出网桥的接口。例如，计算机主机上有多个网桥时，可使用以下规则来匹配数据包是由哪个网桥送入或送出。

```
例一: iptables -A FORWARD -m physdev --physdev-is-in -i br0 -p icmp -j DROP
```

```
例二: iptables -A FORWARD -m physdev --physdev-is-out -o br0 -p icmp -j DROP
```

了解了第2层模式中接口的匹配方式之后，这里通过如下示例来帮助你更清楚了解透明式防火墙的应用，至于规则部分就不再多加解释了。图9-12假设多机型ADSL实际拥有的公网IP是192.168.0.10、192.168.0.11及192.168.0.12，NAT主机所占用的IP是192.168.0.10，Web服务器的IP是192.168.0.11，Mail服务器的IP是192.168.0.12。

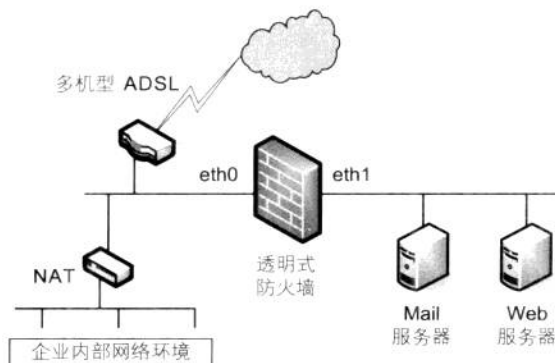


图9-12 透明式防火墙示例(一)

```
#!/bin/bash

WEB=192.168.0.11
MAIL=192.168.0.12
INT="-m physdev --physdev-in"
NAT="e0:cb:4e:eb:23:f3"

iptables -t filter -F

iptables -A FORWARD $INT eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
iptables -A FORWARD $INT eth0 -m mac --mac-source $NAT -j ACCEPT

iptables -A FORWARD $INT eth0 -p tcp --syn -d $MAIL \
                                -m multiport -dports 25,110,143 \
                                -m state --state NEW -j ACCEPT

iptables -A FORWARD $INT eth0 -p tcp --syn -d $WEB \
                                -m multiport --dports 80,443 \
                                -m state --state NEW -j ACCEPT

iptables -A FORWARD $INT eth0 -j DROP
```

2. 透明式防火墙的另一种应用

到目前为止, 我相信你应该已经体会到透明式防火墙所带来的种种优点了, 接下来再介绍一种较特殊的用法。如图9-13所示, 我们假设某企业对外提供服务的主机所使用的通信协议是Linux NAT所不支持的复杂通信协议, 且该企业只有8个固定的公网IP可供使用。在这样的条件下, 请问该如何处理呢? 当然, 我们可以选择图9-12的结构, 但这样的结构需要额外多一台主机来担任NAT的角色; 或许你会联想到图9-13的结构, 因为其拥有8个公网IP。如果以划分为两个子网的方式来处理, 那么这两个子网将各拥有4个公网IP, 但这4个IP中的第一个IP会被用作网段的地址, 最后一个IP则固定为广播地址使用。此外, 还需要一个IP给网关使用。因此, 实际可用的IP将只会剩下一个, 不管怎么样都不太划算了。

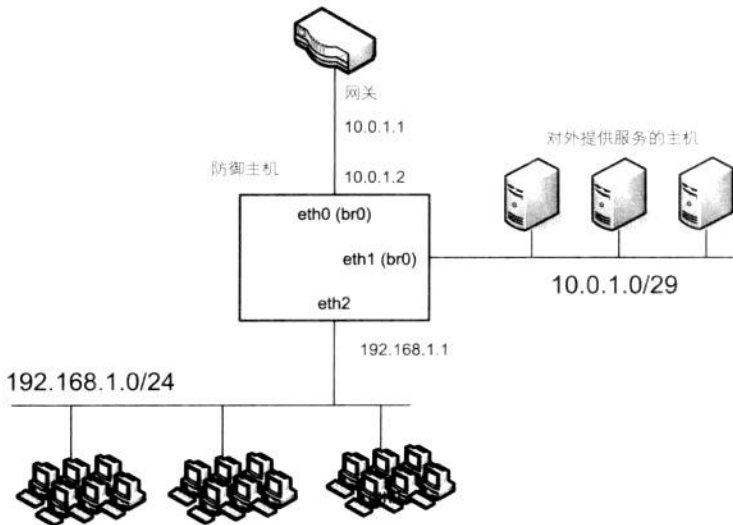


图9-13 透明式防火墙示例 (二)

如果在实际中遇到这样的问题，那么这个示例将可以轻松解决你的问题，不过，在开始之前先要提出一个问题：DMZ内的主机可以正常与企业内部主机通信吗？我想大多数人都会回答“是”！但实际上是行不通的，为什么呢？

```
10.0.1.0/29 dev br0 proto kernel scope link src 10.0.1.1
192.168.1.0/24 dev eth2 proto kernel scope link src 192.168.1.1
169.254.0.0/16 dev eth0 scope link metric 1002
169.254.0.0/16 dev eth2 scope link metric 1003
169.254.0.0/16 dev eth1 scope link metric 1004
default via 10.0.1.254 dev br0
```

我们先来看防御主机上的路由表，从路由表可以发现，凡是要送往192.168.1.0/24网段的数据包都由eth2接口送出，而要送往10.0.1.0/29网段的数据包则由br0接口送出，如此清楚的路由设置怎么会无法通信呢？这其中的关键在于br0是网桥接口，所以对于由br0接口送入的数据包，Linux系统只会判断该数据包是否是给本机的，如果是，则将这个数据包送往上一层(OSI 7层的第三层“网络层”)；如果不是，将数据包由br0接口的其他端口送出。

由此可知，如果DMZ内的主机要与192.168.1.1通信，因为192.168.1.1是网桥自己的IP，所以这个通信会是有效的，但如果DMZ内的主机要与192.168.1.3通信就会比较麻烦了，因为192.168.1.3对于DMZ内的主机来说，这是其他网段的IP。因此，这个数据包将会被送往默认网关，也就是10.0.1.1，且这个IP并非网桥自己的IP，所以这个数据包将会由网桥通过eth0接口送往默认网关，如此一来，这个数据包将会由路由器转发到因特网，这就是10.0.1.0/29网段主机无法与192.168.1.0/24网段主机互通的理由。

要如何才能使10.0.1.0/29网段的主机与192.168.1.0/24网段的主机互通呢？其实这个问题很容易解决，我们只需告诉10.0.1.0/29网段的主机去往192.168.1.0/24网段该如何通行即可。方法有两种：

- 在10.0.1.0/29网段上的每一台主机都设置静态路由，设置通往192.168.1.0/24网段的网关是10.0.1.2。
- 在路由器上设置静态路由，凡是要送往192.168.1.0/24网段的数据包，都转送给10.0.1.2来处理，如此一来，这个网段的路由就可以正常工作了。

最后别忘了给192.168.1.0/24网段的主机设置一对多NAT，让企业内部主机可以正常访问因特网。以下规则供你参考：

```
iptables -t nat -A POSTROUTING -o br0 -s 192.168.1.0/24 ! \
-d 10.0.1.0/29 -j SNAT --to 10.0.1.2
```

9.3.3 另一种透明式防火墙

要达到透明式防火墙的目的，除了可以使用网桥之外，还有其他技术手段可以实现，那就是代理ARP(Proxy ARP)。代理ARP是桥接在两个实体网段之间的设备，而这两个实体网段上使用同一个IP网段，其特点是代理ARP可以正常地路由两个实体网段之间的数据包。请注意，这里使用“路由”来形容这个操作，也就是说，代理ARP工作于OSI 7层的第三层，而网桥是工作于OSI 7层的第二层。

9.3.4 配置代理ARP

我们以图9-14为例来构建这个代理ARP主机。首先假设网关在图的右边，另外在代理ARP主机上有eth0和eth1两块网卡，且这两块网卡上所设置的IP都是相同的(也可以将其设置为不同的IP，不过，这只是额外浪费掉一个IP而已，在代理ARP的功能上完全没有加分效果)。

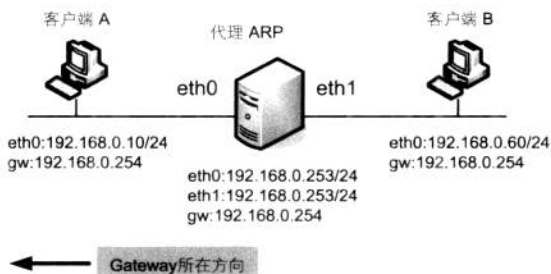


图9-14 代理ARP

如图所示，我们的任务是要通过代理ARP主机，让客户端 B主机与客户端 A主机所在的实体网段可以正常通信。首先，将Linux主机的IP、子网掩码、网关及DNS参数设置好，接着使用以下脚本来启用代理的机制，脚本中的第3及第4行用来启用内核中的代理ARP机制。我们只需要对“用来连接两个实体网络的网卡”进行设置即可，并不需要每一个网卡都设置，最后在第5行告诉代理ARP主机192.168.0.60所连接的实体网卡是eth1。

```
1. #!/bin/bash
2. echo 1 > /proc/sys/net/ipv4/ip_forward
3. echo 1 > /proc/sys/net/ipv4/conf/eth0/proxy_arp
4. echo 1 > /proc/sys/net/ipv4/conf/eth1/proxy_arp
5. ip route add 192.168.0.60 dev eth0
```


或许你会问，万一我们在代理ARP后端有大量计算机时，该如何设置呢？难道要每个IP都单独去设置吗？这个问题非常好，一下就指出了代理ARP机制的致命伤，除非你愿意将代理ARP后端的计算机都设置为固定IP，否则难以在企业内部实施代理ARP机制，那么，代理ARP到底有何作用呢？

以图9-13结构为例，万一我们无法在网关中设置静态路由，这个架构不就无法工作了吗？此时我们可以通过代理ARP的机制来轻松解决这个问题。以图9-15为例来设置这个代理ARP，首先将网络配置好，并使用以下脚本来完成代理ARP的设置，设置完毕后，DMZ内的主机及企业内的主机应该就能正常地访问因特网了，不过，有个小地方要先提醒你，DMZ内主机的默认网关是10.0.1.1，而非10.0.1.2。最后，不要忘记添加防火墙的规则。

```
6. #!/bin/bash
7. echo 1 > /proc/sys/net/ipv4/ip_forward
8. echo 1 > /proc/sys/net/ipv4/conf/eth0/proxy_arp
9. echo 1 > /proc/sys/net/ipv4/conf/eth1/proxy_arp
10. ip route add 10.0.1.3 dev eth0
11. ip route add 10.0.1.4 dev eth0
12. ip route add 10.0.1.5 dev eth0
13. iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.0/24 \
    -j SNAT --to 10.0.1.2
```

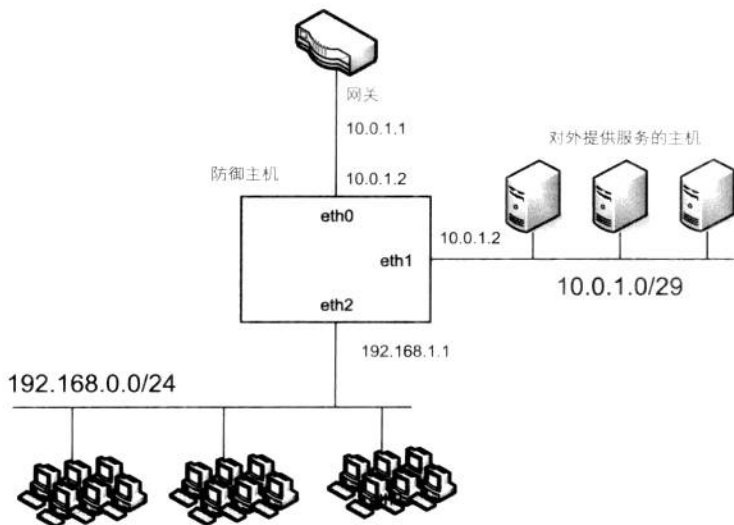
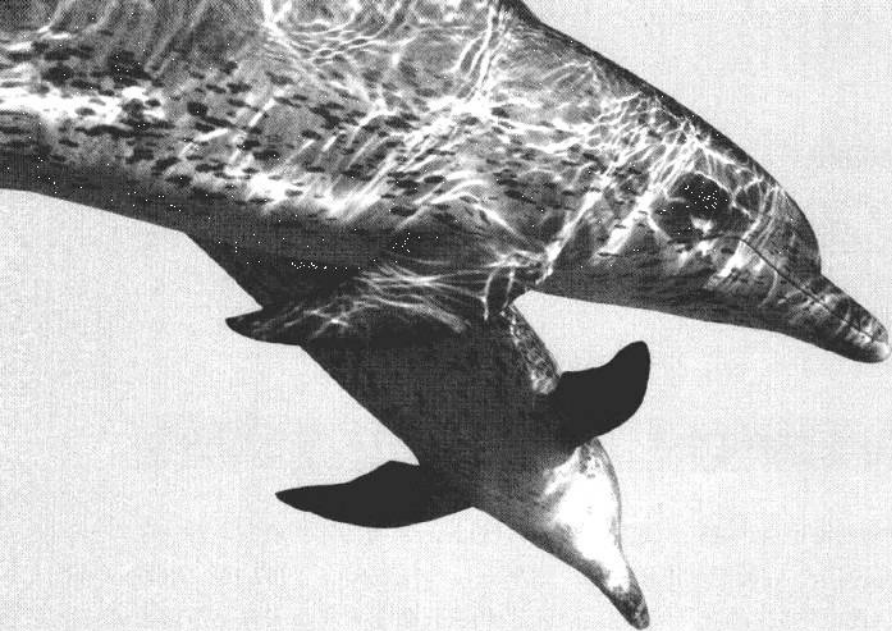


图9-15 代理ARP应用探索

9.4 小结

阅读完这个章节之后，你是否受到极大震撼呢？一个完全不用付费的防火墙系统，却可以提供如此强大的部署能力以及如此强大的安全机制，实在令人叹为观止。因此建议你无论如何都要深入理解本章的内容。



Linux

| 第10章 | 基于策略的路由及多路带宽合并

10

Linux的防火墙功能表现优异,此外Linux还有另一项鲜为人知的特殊功能,那就是基于策略的路由功能(Linux Advance Routing),这项功能通常只会存在于非常高级的路由设备上,但通过开源组织志愿者们们的努力,我们可以免费使用到这种强大的网络路由机制。在讨论基于策略的路由时,有必要讨论一下多路带宽合并,因此,本章除讨论基于策略的路由之外,还会增加多路带宽合并的相关话题。

10.1 何谓基于策略的路由

在开始讨论基于策略的路由前,我们先来熟悉一下传统Linux系统的路由机制。首先在实验用的主机上安装三块网卡,并设定其IP分别是192.168.0.1、192.168.1.1、192.168.2.1(假设这些IP都是公网IP),在网络服务重新启动后,可以使用route命令来观察主机上的路由表,如图10-1所示。路由表的作用是指导主机该如何向外发送数据包,如图10-2所示。当本机有数据包要送往168.95.1.1时,这个数据包必然标记来源端IP为192.168.0.2及目的端IP为168.95.1.1(在IP包头内),接着系统就会以数据包内的目的端IP 168.95.1.1作为路由匹配的依据,首先将168.95.1.1匹配路由表的第①条规则,结果发现168.95.1.1并不包含在192.168.0.0/24的网段内,接着只能向下匹配第②条规则,结果发现168.95.1.1还是不包含在192.168.1.0/24的网段内,接着向下匹配第③条规则,但结果还是一样,最后匹配第④条规则,不过,第④条规则的标记比较特殊,其destination标记为0.0.0.0,且网关标示为0.0.0.0,这0.0.0.0/0.0.0.0的含义是任何目的端都符合,其实最后一条规则就是在网络配置中所设置的“默认网关”,因此,可以看到第④条规则的网关值是192.168.0.254,正是我们所设置的默认网关地址。

```
[root@localhost ~]# route -n
```

Kernel IP routing table

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface	
192.168.0.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0	①
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1	②
192.168.2.0	0.0.0.0	255.255.255.0	U	0	0	0	eth2	③
0.0.0.0	192.168.0.254	0.0.0.0	UG	0	0	0	eth0	④

图10-1 何谓基于策略的路由

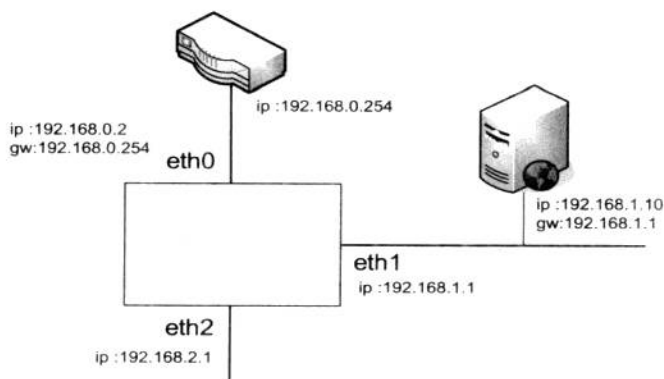


图10-2 Linux路由解释

再列举一个例子，如果本机要将数据包发送到192.168.1.10，又该如何发送数据包呢？如果数据包是由本机送给192.168.1.10，那么系统就会以192.168.1.10为依据与路由表从上到下逐条进行匹配。首先匹配第①条规则发现，192.168.1.10并不包含在192.168.0.0/24的网段内，接着匹配第②条规则发现，192.168.1.10正是属于192.168.1.0/24这个网段内的IP，系统就会将这个数据包从eth1接口(由规则中最后一个字段所设置)送出，从而完成一个数据包的转发操作。

由以上的处理流程可以了解到，Linux判断数据包转发的依据是目的端IP，因此，Linux系统无法提供一些较高级的路由机制，以图10-3为例，Linux系统无法提供：

- Group A的计算机经由HiNet线路上网。
- Group B的计算机经由Seednet线路上网。
- 所有POP3及FTP通信协议经由Seednet线路上网。
- 所有HTTP及SMTP协议经由HiNet线路上网。

以上仅列举几个简单例子，只要是以目的端IP为路由依据以外的规则，传统的Linux路由机制是无法做到的，其实，大多数的一般路由器也都无法提供此类功能，而这种可以随着需求而任意设置路由路径的方式，我们称为基于策略的路由。

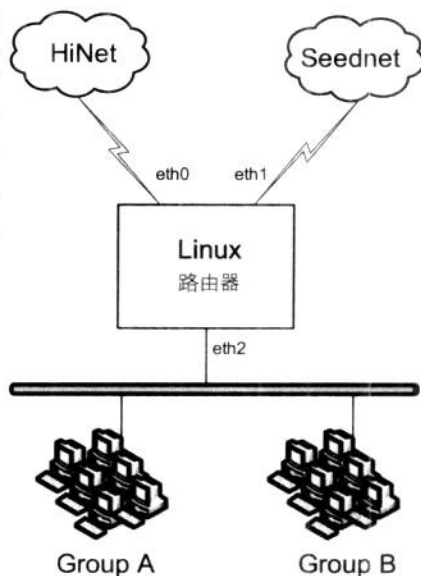


图10-3 何谓基于策略的路由

10.2 了解Linux的路由机制

Linux历经无数岁月的洗礼后，在Linux 2.2版本正式加入了Advanced Routing机制，使得Linux系统的路由机制超越了大多数的商用路由器，因此，也使得Linux系统的路由表变得更加复杂。图10-4显示了当今Linux系统的路由表，从图中可以看到，Linux系统可同时存在多个路由表，一般的操作系统都只有一个路由表，但Linux系统却可多达256个，且每个路由表都各自独立，互不相关。那么，Linux系统在传输数据包时，会根据哪个路由表来传输呢？这个答案就是“路由策略数据库”(Routing Policy Database, RPDB)，我们可以在RPDB内填写一些路由规则来确定“哪类数据包”应该“根据哪个路由表”来传输，如此，系统管理人员就会有更大的自主性来决定数据包的传输路径。

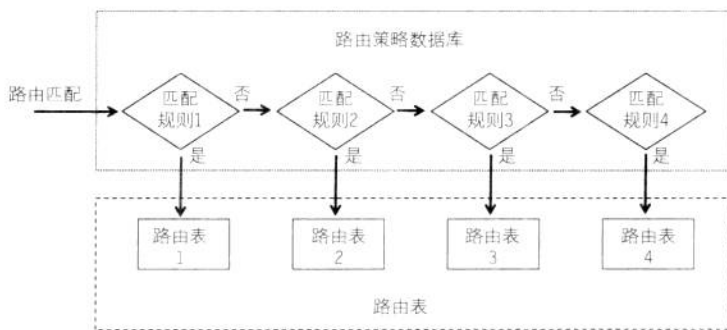


图10-4 Linux路由表

路由策略数据库与路由表

如图10-5所示，可以使用`ip rule show`命令将RPDB的内容显示出来，可将这些信息大概分为三个字段，其中字段①为Priority，简单来说，就是Rule被匹配的优先顺序，数字越小，代表优先级别越高，也就是越早被匹配；字段②是Rule的规则，也就是匹配数据包的依据，在此可以是Source IP、Destination IP、Type of Service、fwmark及dev等；字段③则是符合条件的数据包要通过哪个路由表送出。

```
[root@localhost/]# ip rule show
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
```

图10-5 规则表

在Linux系统中，路由表是根据ID来区分的，而ID的范围是0~255。由于ID对于管理员而言实在是不好记忆，因此基于策略路由的管理工具——`iproute`提供了一个ID与名称的对应表：`/etc/iproute2/rt_tables`，方便我们使用名称来记忆特定的路由表。下面使用图10-6及图

10-7来说明rt_table对应表的用法，如图10-6，先使用ip命令加入一条规则❶，这条规则的意思是说，如果来源端IP位于192.168.2.0/24网段，那么就使用ID编号为10的路由表加以处理，接着使用ip rule show命令将RPDB的内容显示出来，我们可以从❷的位置看到多了一条规则，且这条规则使用ID来代表路由表。

```
[root@localhost /]# ip rule show
0:      from all      lookup local
32766:  from all      lookup main
32767:  from all      lookup default
[root@localhost /]#
[root@localhost /]#
[root@localhost /]# ip rule add from 192.168.2.0/24 table 10 ❶
[root@localhost /]#
[root@localhost /]#
[root@localhost /]# ip rule show
0:      from all      lookup local
32765:  from 192.168.2.0/24 lookup 10 ❷
32766:  from all      lookup main
32767:  from all      lookup default
[root@localhost /]#
```

图10-6 路由表的命名(一)

接着以图10-7为例，在rt_table文件中加入一行如❸所示，保存文件后，再使用ip rule show命令来查看RPDB，这时就可以发现原来使用ID来代表路由表的地方，已经改为我们在rt_table之中所设置的名字了。

```
[root@localhost /]# cat /etc/iproute2/rt_tables
#
# reserved values
#
255    local
254    main
253    default
0       unspec
10      For_Rule_TEST ❸
#
# local
#
#1     inr.ruhep
[root@localhost /]#
[root@localhost /]# ip rule show
0:      from all      lookup local
32765:  from 192.168.2.0/24 lookup For_Rule_TEST ❹
32766:  from all      lookup main
32767:  from all      lookup default
```

图10-7 路由表的命名(二)

10.3 路由策略数据库与路由表的管理

一早期在管理Linux系统的网络时，常使用ifconfig及route之类的命令，不过如果你准备开始使用Linux强大的基于策略的路由机制，那么，就请不要使用这类工具了，因为这类工具根本无法用于功能强大的基于策略的路由机制，取而代之的工具是iproute。iproute这个软件在RedHat系列的Linux系统中是默认安装的，因此，你通常可以找到这个工具。如果真因为某些原因找不到这个软件，只要在使用Fedora或CentOS Linux时，在联网的情况下，用yum install iproute命令即可顺利安装；或者也可以使用ip -V命令来检查iproute软件是否已经安装，再次请注意，-V参数为大写的英语字母：

```
[root@localhost ~]# ip -V
ip utility, iproute2-ss091226
```

10.3.1 管理策略数据库

在Linux下，基于策略路由的策略数据库是由ip命令来管理的，下面讨论“管理”的几个方面：

1. 查看策略数据库

要查看策略数据库的内容，可以使用ip rule show命令，或者可以使用ip rule ls。如下是命令执行后所得到的输出结果，在这些数据中，可以看到系统的三条默认规则，而这三条规则默认分别对应于local、mail及default三个路由表。

```
[root@localhost ~]# ip rule show
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
```

2. 添加规则

在添加规则时，必须先确定好“条件”、“优先级”及“路由表ID”，此后才可以执行添加规则的操作。关于这三个参数的含义，我们根据图10-5再说明一次：

条件

条件是用来决定哪类数据包可以符合这项规则，而可用来匹配的字段为Source IP、Destination IP、Type of Service、fwmark及dev等，这些字段的使用方式如下：

● Source IP:

根据来源端IP来决定数据包参考哪个路由表发送出去。以下两个示例分别指出,如果数据包的来源端IP是192.168.1.10,就参考路由表10;如果来源端IP为192.168.2.0/24网段的IP,就参考路由表20。

```
ip rule add from 192.168.1.10 table 10
ip rule add from 192.168.2.0/24 table 20
```

● Destination IP:

根据目的端IP来决定数据包参考哪个路由表发送出去。以下两个示例分别指出,如果数据包的目的地IP是168.95.1.1,就参考路由表10;如果目的端IP是168.95.0.0/24网段的IP,就参考路由表20。

```
ip rule add to 168.95.1.1 table 10
ip rule add to 168.95.0.0/24 table 20
```

● fwmark:

fwmark指第3章所提到过的nfmark,如果你已忘了nfmark的含义,不妨参考第2.2.6节。将fwmark作为匹配条件时,必须搭配Netfilter一起使用,这看起来很麻烦,却是最灵活的匹配条件。如图10-8所示,某公司对外有二条ADSL,我们希望所有HTTP协议经由第一条ADSL,SMTP及POP3经由第二条ADSL,其余流量则经由第三条ADSL。可以使用如下的命令组合来达到这样的目的:

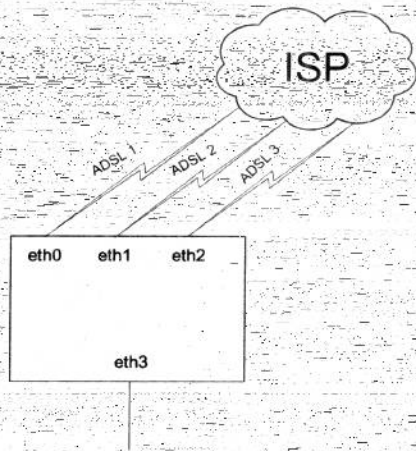


图10-8 fwmark示例

```
iptables -t mangle -A FORWARD -i eth3 -p tcp --dport 80 -j MARK --set-mark 1
iptables -t mangle -A FORWARD -i eth3 -p tcp --dport 25 -j MARK --set-mark 2
iptables -t mangle -A FORWARD -i eth3 -p tcp --dport 110 -j MARK --set-mark 2
iptables -t mangle -A FORWARD -i eth3 -j MARK --set-mark 3

ip rule add fwmark 1 table 1
ip rule add fwmark 2 table 2
ip rule add fwmark 3 table 3
```

首先使用Netfilter的mangle机制针对特定的数据包设置MARK值,在此将HTTP数据包的MARK值设置为1,SMTP及POP3数据包的MARK值设置为2,其余数据包则设置MARK值

为3。接着，再根据fwmark条件来判断数据包的MARK值，如果MARK值为1，则参考路由表1将数据包送出；MARK值为2时，则参考路由表2将数据包送出；最后，MARK值为3的数据包则参考路由表3送出。

以上示例只是一个概念而已，如果真要完整体现出这个示例的所有功能，还需要注意许多细节，稍后将使用详细的示例讲解这部分内容，在此只要首先了解fwmark与Netfilter结合使用的概念即可。

● dev:

最后，还可以使用数据包输入的接口来作为判断依据，如图10-9所示，我们希望凡是由eth2接口送入的数据包都由eth0接口转发出去，由eth3接口送入的数据包都由eth1接口转发出去。以下命令组合将能满足我们的要求：

```
ip rule add dev eth2 table 1
ip rule add dev eth3 table 3
```

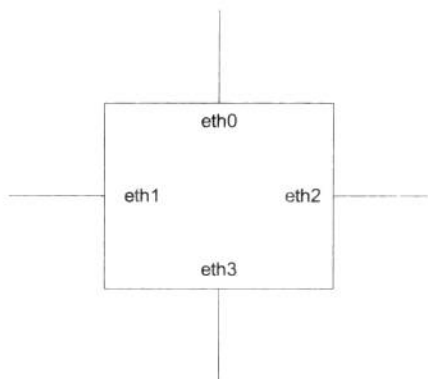


图10-9 接口示例

3. 优先级

前面介绍了规则中“条件”的使用方式，接下来要讨论的是优先级。优先级用数字来表示，其范围可由0~4亿多，堪称天文数字，我们实际上不可能在一台PC上设置如此庞大的路由机制。

```
[root@localhost ~]# ip rule show
0:      from all          lookup local
32766:  from all          lookup main
32767:  from all          lookup default
[root@localhost ~]#
[root@localhost ~]# ip rule add from 192.168.1.0/24 table 1
[root@localhost ~]# ip rule add from 192.168.2.0/24 table 2
[root@localhost ~]#
[root@localhost ~]# ip rule show
0:      from all          lookup local
32764:  from 192.168.2.0/24 lookup 2
32765:  from 192.168.1.0/24 lookup 1
32766:  from all          lookup main
32767:  from all          lookup default
```

如以上示例，我们执行ip rule show命令所显示内容的第一个字段就是优先级，数字越小，代表优先级越高，也代表这条规则可以排得越靠前，如此数据包在进行条件匹配时，就会越早匹配到这条规则，从输出的数据中，默认优先级0、32766及32767已被占用，因

此，在添加规则时，如果没有特别设置优先级，那么，优先级默认会从32766开始递减，如32765、32764……，如果我们特别设置优先级，可以在ip rule add命令的最后加上prio-XXX参数。如下例所示：

```
[root@localhost ~]# ip rule show
0:      from all          lookup local
32766:  from all          lookup main
32767:  from all          lookup default
[root@localhost ~]#
[root@localhost ~]# ip rule add from 192.168.1.0/24 table 1 prio 10
[root@localhost ~]# ip rule add from 192.168.2.0/24 table 2 prio 20
[root@localhost ~]#
[root@localhost ~]# ip rule show
0:      from all          lookup local
10:     from 192.168.1.0/24 lookup 1
20:     from 192.168.2.0/24 lookup 2
32766:  from all          lookup main
32767:  from all          lookup default
```

路由表ID

在Linux的基于策略的路由中，路由表用ID来表示，但如有必要，还可以用ID与名称对照表将ID转换成名称，关于这部分请参考第10.2.1节。

4. 删除规则

ip命令提供的删除规则的方式十分灵活，例如，要删除下列第2条规则，可以分别使用“优先级”、“条件”及“路由表”当中任何一个唯一的值来设置所需删除的规则，如下：

- ip rule del prio 10
- ip rule del from 192.168.1.0/24
- ip rule del table 1
- ip rule del from 192.168.1.0/24 table 1 prio 10

```
[root@localhost ~]# ip rule show
0:      from all          lookup local
10:     from 192.168.1.0/24 lookup 1
20:     from 192.168.2.0/24 lookup 2
32766:  from all          lookup main
32767:  from all          lookup default
[root@localhost ~]#
```


10.3.2 管理路由表

在讨论“管理路由表”之前，有必要先对路由表有所了解。以图10-10为例，先使用传统的route -n命令来查看系统上的路由表，我们可以得到如图10-11的信息，从图中可以看到两条169.254.0.0/16的路由信息，但问题在于主机上并未设置与该网段有关的IP，为什么会多出这两条路由信息呢？其实这与APIPA协议有关，简单来说，在局域网内，当DHCP服务器无法正常工作时，APIPA协议可以使得局域网内的主机依然能正常工作，如果不需要这个机制，只需在/etc/sysconfig/network文件中加入“NOZEROCONF=no”这一行，然后重新启动网络即可关闭APIPA的机制，此时，系统的路由表如图10-12所示。

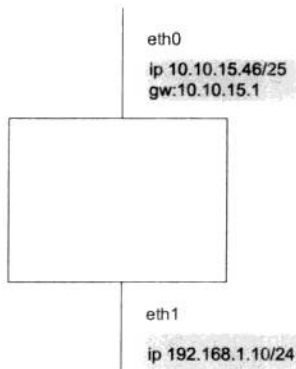


图10-10 Linux路由示例

```
[root@localhost /]# route -n
```

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.10.15.0	0.0.0.0	255.255.255.128	U	0	0	0	eth0
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
169.254.0.0	0.0.0.0	255.255.0.0	U	1002	0	0	eth0
169.254.0.0	0.0.0.0	255.255.0.0	U	1003	0	0	eth1
0.0.0.0	10.10.15.1	0.0.0.0	UG	0	0	0	eth0

```
[root@localhost /]#
```

图10-11 传统路由表(一)

```
[root@localhost /]# route -n
```

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.10.15.0	0.0.0.0	255.255.255.128	U	0	0	0	eth0
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
0.0.0.0	10.10.15.1	0.0.0.0	UG	0	0	0	eth0

```
[root@localhost /]#
```

图10-12 传统路由表(二)

由于route -n命令已经完全不适合在基于策略的路由使用，因此，route命令仅能操作一个特定的路由表，但在基于策略的路由中，会同时存在多个路由表，请放弃这个路由管理工具，取而代之的依然是ip命令。接下来将讨论如何使用ip命令来管理路由表。

1. 查看路由表内容

在查看路由表之前，首先使用ip rule show命令来查看目前使用了哪些路由表，接着，再使用ip route show [table id | name]命令来查看路由表的内容。例如，可以使用ip route show table main来查看路由表main的内容，如果省略路由表名称(如ip route show)，会默认地查看路由表main的内容。

```
[root@localhost /]# ip rule show
```

0:	from all	lookup local
32766:	from all	lookup main


```

32767: from all lookup default
[root@localhost ~]#
[root@localhost ~]# ip route show table main
10.10.15.0/25 dev eth0 proto kernel scope link src 10.10.15.46
192.168.1.0/24 dev eth1 proto kernel scope link src 192.168.1.10
default via 10.10.15.1 dev eth0
[root@localhost ~]#

```

在默认情况下，系统有三个路由表，这三个路由表的功能如下：

- **local**：路由表local包含本机路由及广播信息。例如，在本机上执行ssh 127.0.0.1时，就会参考这份路由表的内容，在正常情况下，只要配置好网卡的网络设置，就会自动生成local路由表的内容，我们应该也不必修改其内容。
- **main**：使用传统命令route -n所看到的路由表就是main的内容。Linux系统在默认情况下使用这份路由表的内容来传输数据包，因此，其内容极为重要，在正常情况下，只要配置好网卡的网络设置，就会自动生成main路由表的内容。
- **default**：最后是default路由表，这个路由表在默认情况下内容为空；除非有特别的要求，否则保持其内容为空即可。

在此使用路由表main的内容进行解释，以下是图10-10路由表main的内容，因为在主机上有eth0及eth1两块网卡，且为其设置的IP分别是10.10.15.46/25及192.168.1.10/24，因此，路由表内的第①行即是告诉系统，如果有数据包要送到10.10.15.0/25这个网段，就直接将数据包由eth0接口送出即可，而本机临近这个网段的IP是10.10.15.46，第②行则是设置到192.168.1.0/24的路由，其含义与第①行完全相同；以上这两行是只要将计算机网卡上的IP设置好，并在网络服务重启之后，默认就会生成的路由，无需特别的设置。最后一行③则指：如果数据包不是送往10.10.15.0/25及192.168.1.0/24网段，那么数据包将统一转发给10.10.15.1主机去处理，而10.10.15.1就是我们在网络配置中所设置的“默认网关”。

```

[root@localhost ~]# ip route show table main
10.10.15.0/25 dev eth0 proto kernel scope link src 10.10.15.46 ①
192.168.1.0/24 dev eth1 proto kernel scope link src 192.168.1.10 ②
default via 10.10.15.1 dev eth0 ③
[root@localhost ~]#

```

2. 添加路由

添加路由在此还是一样采用ip命令而不是route命令，下例首先使用ip route show①命令显示路由表main的内容，接着再使用ip route add命令将所需的路由添加到路由表main中②，最后再次使用ip route show命令将路由表main的内容打印出来，此时就可以在路由表main之中看到刚才添加的路由了。

```

[root@localhost /]# ip route show table main ❶
10.10.15.0/25 dev eth0 proto kernel scope link src 10.10.15.46
192.168.1.0/24 dev eth1 proto kernel scope link src 192.168.1.10
default via 10.10.15.1 dev eth0
[root@localhost /]#
[root@localhost /]# ip route add 192.168.2.0/24 via 10.10.15.50 table main ❷
[root@localhost /]#
[root@localhost /]# ip route show table main ❸
10.10.15.0/25 dev eth0 proto kernel scope link src 10.10.15.46
192.168.2.0/24 via 10.10.15.50 dev eth0
192.168.1.0/24 dev eth1 proto kernel scope link src 192.168.1.10
default via 10.10.15.1 dev eth0
[root@localhost /]#

```

如果要添加的路由并未出现在现有的路由表中, 又该如何处理呢? 在此请先有一个概念, 单纯添加路由表并无意义, 因为新增出来的路由表, 系统默认是不会去使用的, 如果要添加路由到main以外的路由表, 只有先添加“规则”才能确定新的路由表名称(Table ID), 有了新的路由表之后, 才会把路由添加到新的路由表中。

我们使用下列示例来说明这个过程。首先使用ip rule show❶来查询RPDB的当前状态, 可以看到目前只有三条默认规则, 接着, 再使用ip rule add命令来添加一条规则❷, 此时系统内就多了一个有用的路由表, 其路由表ID为10, 我们可以立即使用ip route show命令来查看这个新的路由表❸, 其内容默认为空, 接着可以在这个新路由表中添加路由, 在此使用ip route add命令来添加路由, 我们决定凡是来自于192.168.2.0/24网段的数据包, 都从eth1接口将数据包送离本机, 因此, 必须完整编写eth1接口的路由。首先将临近eth1接口的路由填入❹, 告诉系统本机与192.168.1.0/24网段的通信都通过eth1接口来处理, 接着填入这个路由表的默认路由❺, 最后使用ip route show命令显示路由表10的内容。

```

[root@localhost ~]# ip rule show ❶
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
[root@localhost ~]#
[root@localhost ~]# ip rule add from 192.168.2.0/24 table 10 ❷
[root@localhost ~]#
[root@localhost ~]# ip route show table 10 ❸
[root@localhost ~]#
[root@localhost ~]# ip route add 192.168.1.0/24 dev eth1 table 10 ❹
[root@localhost ~]# ip route add default via 192.168.1.254 table 10 ❺
[root@localhost ~]#
[root@localhost ~]# ip route show table 10 ❻
192.168.1.0/24 dev eth1 scope link
default via 192.168.1.254 dev eth1
[root@localhost ~]#

```

3. 删除路由

可以使用ip命令来方便地删除路由，我们使用以下示例来说明如何删除路由。首先将路由表10的内容显示出来❶，可以看到路由表10中当前有两条路由，接着使用ip route del命令删除默认路由❷，在此别忘了指定我们所要删除的是路由表10，否则默认会删除路由表main的默认路由，接着再使用ip route show 命令查看路由表10❸，此时路由表10的默认路由已经不存在了，再次使用ip route del命令删除192.168.122.0/24的路由❹，最后可以看到路由表10中已经没有任何路由了❺。

```
[root@localhost ~]# ip route show table 10           ❶
192.168.1.0/24 dev virbr0 scope link
default via 192.168.1.254 dev eth1
[root@localhost ~]#
[root@localhost ~]# ip route del default table 10    ❷
[root@localhost ~]#
[root@localhost ~]# ip route show table 10           ❸
192.168.1.0/24 dev virbr0 scope link
[root@localhost ~]#
[root@localhost ~]# ip route del 192.168.1.0/24 table 10 ❹
[root@localhost ~]#
[root@localhost ~]# ip route show table 10           ❺
[root@localhost ~]#
```

10.4 带宽合并

带宽合并这个主题似乎从有因特网开始就一直存在着，最初的名称是Multi-Link PPP，Multi-Link PPP是什么呢？记得当时我的计算机上网使用的是28800 bps的调制解调器，速度特别慢，因此，当时就有人提出Multi-Link PPP这种技术，也就是把多条通过调制解调器所建立起来的PPP连接叠加起来，组成一个带宽较大的逻辑连接。只是当年我还是个穷书生，玩不起如此高档的网络架构，事隔15年的今日，虽然上网的带宽已经很大了，但是带宽合并这个话题依然是所有网络工程师最喜欢讨论的话题之一。在可预见的未来不管上网的带宽变得多大，这个话题也会无止境地讨论下去，因为“上网的带宽有限，人的欲望无穷”！

要如何合并多条ADSL或光纤的带宽呢？很多人干脆买一台所谓的“负载均衡器”来执行带宽合并的操作，只是你可能不知道，其实这些负载均衡器里面所安装的就是Linux系统。此外，如果钱花得不够多，这个负载均衡器可能三天两头就会宕机一次，那时可就要劳驾你重启了。因此，如果你想购买负载均衡器的话，最好先掂量一下自己的口袋！

10.4.1 何谓带宽合并

所谓带宽合并是指把多条等速或不等速的外网带宽，合并成一个逻辑上更大的带宽来使用，如图10-13将三条12M/1M的ADSL合并起来，如此内部的使用者就可以得到36M/3M的外网带宽，这就是所谓的带宽合并。而用来执行这个任务的设备，厂商称之为负载均衡器，不过，这其中可是有不可告人的秘密，因为这些设备所宣称的“疗效”大部分都是有争议的，我个人认为那根本是骗人的，此话怎讲？除非我们通过ISP来执行带宽合并任务，否则这些负载均衡器所称的“疗效”在实际中根本体现不出来。

如图10-14所示，如果通过ISP来执行这项任务，ISP通常使用支持带宽合并功能的路由器来执行这项任务，这与负载均衡器的做法有何不同呢？关键之处在于流量分配上的不同，如图10-14的做法通常是把一条连接上的数据包均分到多条不同的实体连接上，如此每条实体链路所用的带宽趋于平均。但负载均衡器因为没有得到ISP的支持，因此，他们的做法会是根据会话(Session)来处理，例如，企业内部对因特网所产生的第一条连接，负载均衡器会将之导入到第一条实体线路，第二条连接会导入到第二条实体线路，第三条连接则会导入到第三条实体线路，那第四条连接呢？当然就是导入到第一条实体线路了，因此，你所得到的单一连接带宽上限永远不可能达到36M/3M的带宽，但ISP的做法是可行的。

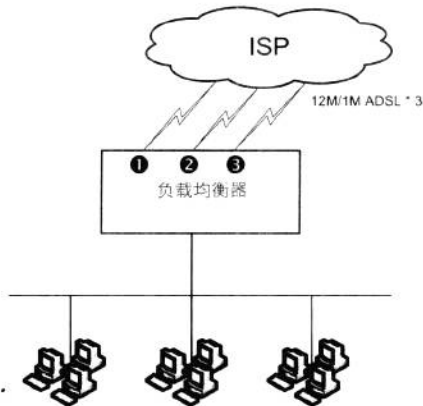


图10-13 带宽合并

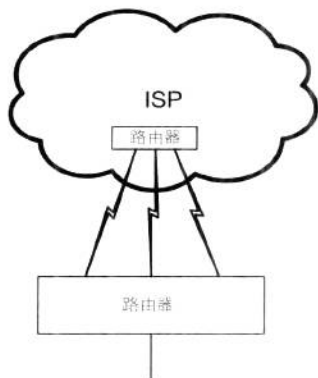


图10-14 传统的带宽合并

那么，负载均衡器还有什么用处吗？当然是有的，ISP做法的限制是所有连接必须都归同一家ISP所有，如果ISP出了问题，你们家就不能上网了；而且ISP的做法比较“昂贵”，如果换成负载均衡器则可以同时使用多家不同ISP的线路，有些比较好的负载均衡器在发现某条实体线路断线时，还会自动排除使用这条实体线路。另外，负载均衡器比较“便宜”，因此，负载均衡器一样有其存在的价值。而本章要介绍“一台比商场买得到的负载均衡器更聪明的负载均衡器”。

10.4.2 企业内的带宽合并

在真正着手合并带宽之前，我们先来回顾一下一对多NAT的概念。如图10-15所示，当192.168.0.0/24网段的主机要连上因特网时，NAT的POSTROUTING机制会将数据包内来源端IP换成10.10.15.46，然后再交给路由表判断数据包该送往何处。在此请注意，这个数据包最后的来源端IP是10.10.15.46。



图10-15 NAT概念

接下来以图10-16为例来示范如何进行带宽合并。首先看到10.10.15.46及192.168.1.10都是可以正常连上因特网的IP，这两个IP分别设置在实验主机的eth0及eth1两张网卡上，其网关是10.10.15.1及192.168.1.254，而内部网络则连接到eth2接口（IP是192.168.122.1），由于192.168.122.0/24是私有IP网段，因此内部网络主机如果要连上因特网，就必须在负载均衡器上完成“一对多NAT”任务。我们使用如下命令完成合并操作。

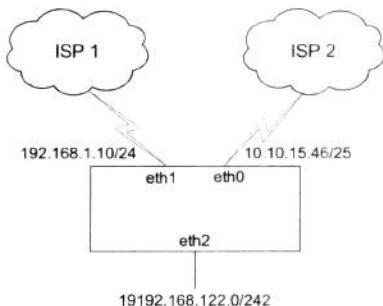


图10-16 带宽合并示例

```
[root@localhost ~]# ip rule show
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
[root@localhost ~]#
[root@localhost ~]# ip rule add from 10.10.15.46 table 10
[root@localhost ~]# ip rule add from 192.168.1.10 table 20
[root@localhost ~]#
[root@localhost ~]# ip route add 10.10.15.0/25 dev eth0 table 10
[root@localhost ~]# ip route add default via 10.10.15.1 table 10
[root@localhost ~]#
[root@localhost ~]# ip route add 192.168.1.0/24 dev eth1 table 20
[root@localhost ~]# ip route add default via 192.168.1.254 table 20
[root@localhost ~]#
[root@localhost ~]# ip route replace default \
nexthop via 10.10.15.1 dev eth0 weight 1 \
nexthop via 192.168.1.254 dev eth1 weight 1
[root@localhost ~]#
[root@localhost ~]# iptables -t nat -A POSTROUTING \
-s 192.168.122.0/24 -j MASQUERADE
[root@localhost ~]#
```

首先使用第⑨行的命令来执行“一对多NAT”任务，但问题来了，在此之前，我们在执行一对多NAT时都只有一个对外的网关，因此，经过NAT转换后的IP一定就是临近网

关的IP,但现在主机对外有两个网关,那么,通过NAT转换后的IP会是192.168.1.10还是10.10.15.46呢?关键就在于主机所设置的网关在哪里。例如,把网关设置为192.168.1.254时,通过NAT转换后的IP就是192.168.1.10;如果网关设置为10.10.15.1时,通过NAT转换后的IP就是10.10.15.46。但问题来了,如果主机的网关是固定的,那就只能使用单一一条对外的实体线路,当然也就不可能使用带宽合并的功能。

为此,我们只有让主机的网关不断地改变,才能使得对外的连接平均分摊在两条示例线路上,关于这个目标,我们可以使用第③行来完成,其中`replace default`参数是指添加或取代目前的网关地址,此外,还可以使用`nexthop via 10.10.15.1 dev eth0 weight 1`来添加一个网关地址,如果同时有多个网关(有多条对外的实体线路),则可以同时使用多组这样的参数来添加网关,因为这个示例中有两条实体线路,因此,我们添加了两个网关地址(分别是10.10.15.1及192.168.1.254),最后`weight 1`是该网关地址的使用率,在这个示例中,两条实体线路带宽相同,因此将使用率设置为1:1。

有了以上的设置之后,我们已经可以让由内对外的连接经过NAT,平均地转换到两条不同实体线路的IP上。例如,第一条连接会转换到第一条实体线路的IP,第二条连接会转换到第二条实体线路的IP,第三条连接会重新转换到第一条实体线路的IP,如此不断地循环。

接下来让属于A实体线路的IP经由A线路出去,属于B实体线路的IP经由B线路出去,关于这个问题,可以使用以前介绍过的“基于策略的路由”机制来轻松解决。我们使用第②行命令来指定,数据包的来源端IP若为10.10.15.46,则使用路由表10来决定数据包该如何传输;用第③行命令来设置数据包的来源端IP若是192.168.1.10,则使用路由表20来决定数据包该如何传输。那么,为什么是10.10.15.46及192.168.1.10这两个IP呢?这是因为NAT所转换的IP就是这两个IP。

有了路由规则后,当然就必须完整定义路由表,我们使用第⑤行命令来设置本路由表的默认网关地址是10.10.15.1;使用第④行命令来告诉系统,如果数据包是要送往10.10.15.0/25网段的数据包,则直接有eth0接口送出即可;再使用第⑥及第⑦行命令来设置eth1接口的路由。

通过以上的命令设置之后,本机及内部网络192.168.122.0/24对因特网的访问连接,就可以平均分摊在这两条实体线路上。以下是分别在本机及内部网络主机上的路由测试结果,而路由的目标分别为220.128.3.106及60.119.17.221两个IP。首先,图10-17及图10-18是在本机上测试的结果,从图10-16中可以清楚地看到测试的连接是从192.168.1.254连接到因特网,而图10-18则是从10.10.15.1连接因特网。另外,图10-18中的XX.XX.XX.XX及YY.YY.YY.YY是一家真实公司连接因特网的IP地址,基于隐私考虑,对其用马赛克进行处理。


```
[root@localhost ~]# traceroute -n 60.199.17.221
traceroute to 60.199.17.221 (60.199.17.221), 30 hops max, 60 byte packets
 1 192.168.1.254 0.633 ms 0.713 ms 0.962 ms
 2 10.8.53.222 3709.033 ms 3728.141 ms 3728.123 ms
 3 10.24.253.54 3728.962 ms 3748.956 ms 3758.055 ms
 4 111.71.246.254 3768.039 ms 3768.890 ms 3788.009 ms
 5 203.75.135.82 3787.986 ms 3797.824 ms 3807.932 ms
 6 220.128.8.198 3808.017 ms 220.128.9.234 3837.421 ms 220.128.9.198 3827.379 ms
 7 220.128.2.58 139.110 ms 121.872 ms
 8 220.128.4.177 119.840 ms 138.928 ms 220.128.3.245 128.898 ms
 9 210.242.214.153 129.792 ms 119.877 ms 148.922 ms
10 60.199.17.173 119.753 ms 60.199.17.165 140.710 ms 60.199.17.169 129.918 ms
11 60.199.17.221 120.893 ms 139.681 ms 129.797 ms
[root@localhost ~]#
```

图10-17 本机路由测试(一)

```
[root@localhost ~]# traceroute -n 220.128.3.106
traceroute to 220.128.3.106 (220.128.3.106), 30 hops max, 60 byte packets
 1 10.10.15.1 0.383 ms 0.405 ms 0.465 ms
 2 10.10.12.27 0.662 ms 0.715 ms 0.769 ms
 3 XX.XX.XX.XX 30.924 ms 31.549 ms 32.022 ms
 4 YY.YY.YY.YY 32.269 ms 33.207 ms 34.164 ms
 5 220.128.1.158 36.397 ms 220.128.1.234 37.361 ms *
[root@localhost ~]# [root@localhost ~]#
```

图10-18 本机路由测试(二)

图10-19及图10-20是在内部网络主机上执行的路由测试，我们可以从两个图中看到内部网络主机的网关都是192.168.122.1，不过，接下来的第二条路由则分别是10.10.15.1及192.168.1.254，如此即可证明内部网络上的主机在连接因特网时，其会话可以平均分摊在两条实体线路上。

```
[root@localhost ~]# traceroute -n 60.199.17.221
traceroute to 60.199.17.221 (60.199.17.221), 30 hops max, 40 byte packets
 1 192.168.122.1 0.855 ms 0.739 ms 0.715 ms
 2 10.10.15.1 0.812 ms 0.884 ms 0.871 ms
 3 10.10.12.27 1.191 ms 1.182 ms 1.238 ms
 4 XX.XX.XX.XX 120.968 ms 122.018 ms 123.276 ms
 5 YY.YY.YY.YY 122.895 ms 124.191 ms 125.874 ms
 6 220.128.3.142 127.110 ms 220.128.2.230 128.027 ms 220.128.3.142 128.964 ms
 7 220.128.3.245 130.699 ms 131.399 ms 132.471 ms
 8 210.242.214.153 134.069 ms 135.018 ms 136.014 ms
 9 60.199.17.181 137.287 ms 60.199.17.185 98.832 ms 60.199.17.181 99.437 ms
10 60.199.17.221 156.924 ms 157.935 ms 157.354 ms
[root@localhost ~]#
```

图10-19 内部网络路由测试(一)

```
[root@localhost ~]# traceroute -n 220.128.3.106
traceroute to 220.128.3.106 (220.128.3.106), 30 hops max, 40 byte packets
 1 192.168.122.1 0.822 ms 0.745 ms 0.728 ms
 2 192.168.1.254 1.245 ms 1.455 ms 1.450 ms
 3 10.8.53.222 112.670 ms 152.578 ms 192.600 ms
 4 10.24.253.54 232.541 ms 312.494 ms 392.446 ms
 5 111.71.246.254 431.946 ms 471.424 ms 510.994 ms
 6 203.75.135.82 630.692 ms 630.645 ms 670.486 ms
 7 220.128.3.106 609.565 ms * *
[root@localhost ~]#
```

图10-20 内部网络路由测试(二)

1. 如何使用拨号式实体线路来合并带宽

如果只是要让企业内的客户端连上因特网，那么，使用拨号式的宽带会比较经济的方案，图10-21是作者使用3.75G USB无线网卡拨号之后所产生的环境，在拨号完成后会得到一个PPP0的虚拟接口，这时如果使用ifconfig命令来查看系统，会看到如图10-22的情况，其中需要注意的是PPP0接口的参数，如inet addr: 111.82.149.58 P-t-P:10.64.64.64 Mask: 255.255.255.255。其中111.82.149.58就是这次拨号所得到的IP，P-t-P所指的IP 10.64.64.64则是PPP0的另一个端点，最后比较奇怪的部分是Mask，毫无疑问，PPP类型的子网掩码都是255.255.255.255。

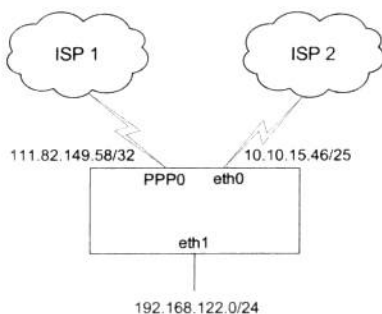


图10-21 拨号与固接混合带宽合并示例

```
[root@localhost ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr E0:CB:4E:EB:23:F3
          inet addr:10.10.15.35  Bcast:10.10.15.127  Mask:255.255.255.128
          inet6 addr: fe80::e2cb:4eff:feeb:23f3/64  Scope:Link
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:280533 errors:0 dropped:0 overruns:0 frame:0
          TX packets:224293 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
```

```

RX bytes:41294929 (39.3 MiB) TX bytes:267726211 (255.3 MiB)
Interrupt:50 Base address:0x6000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:135 errors:0 dropped:0 overruns:0 frame:0
          TX packets:135 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:12816 (12.5 KiB) TX bytes:12816 (12.5 KiB)

ppp0      Link encap:Point-to-Point Protocol
          inet addr: 111.82.149.58 P-t-P:10.64.64.64 Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
          RX packets:6 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:102 (102.0 b) TX bytes:138 (138.0 b)

[root@localhost ~]#

```

图10-22 拨号与固接混合带宽合并示例

为了便于解释，在此先说明PPP网络的特性。所谓PPP(Point-to-Point Protocol)是指用来连接计算机的这一条实体线路上只会有两台计算机，一台连接于线路的这一段，另一台连接到线路的另一端，当传输数据时并不需要专门设置数据是要送给谁，因为这条线路上只有两台计算机，这一方送出去的数据一定是给另一方，而另一方送出来的数据也一定是给这一方，有了这个概念之后，就可以合并带宽了。

在拨号完毕之后，可以使用下面的脚本将“固接”与“拨号”的带宽合并起来，这个脚本与前面合并两路固接式带宽的方法很类似，最主要的差异大概是第❶点，由于PPP0是属于PPP类型的网络连接，所以在写路由表时，不需要像步骤❷一样考虑相同网段上数据包传输的路由问题，我们只需要设置路由表20的默认网关是PPP0接口的IP即可。如果你不熟悉PPP网络，可能很奇怪，回顾刚刚讨论过的内容，在PPP网络上这一方送出去的数据就一定是给另一方的，因此，如果要将数据通过PPP网络送出去，只要把数据从PPP链路的这一端发送出去即可。

```

#!/bin/bash

ip rule add from 10.10.15.35 table 10
ip rule add from 111.82.149.58 table 20

ip route add 10.10.15.0/25 dev eth0 table 10

```

❶

```
ip route add default via 10.10.15.1 dev eth0 table 10

ip route add default via 111.82.149.58 dev ppp0 table 20 ②

ip route replace default \
    nexthop via 10.10.15.1 dev eth0 weight 1 \
    nexthop via 111.82.149.58 dev ppp0 weight 1

iptables -t nat -A POSTROUTING -o eth0 -s 192.168.122.0/24 \
    -j SNAT --to 10.10.15.35
iptables -t nat -A POSTROUTING -o ppp0 -s 192.168.122.0/24 \
    -j SNAT --to 111.82.149.58
```

不过，由于PPP的拨号连接(PPPOE, 3G)很容易断线，如果要使用拨号的对外带宽，可能需要自己编写一些Shell脚本来判断线路是否已经中断。如果中断了，就得自动重新拨号，并且获取新的IP，然后自动更新路由规则及路由表的内容，实际工作中确实会比较麻烦。

2. 带宽合并后的问题及解决办法

至此，我们已经把带宽合并起来了，使用上大概没有什么太大的问题，可能会有读者提出这样的质疑：“在负载均衡器后面的客户端，如果以FTP协议的被动模式，将文件上传到因特网上的FTP服务器，有没有可能造成命令通道经由实体线路A，而DATA通道经由实体线路B的情况呢？”如果你怀有相同的疑问，那就表示你对TCP/IP及FTP协议有足够的认识，不过，这个问题倒不必担心。

Linux将数据包发送出去之前，都会为每一个数据包选择发送路径，而这个操作将造成数据包传输上的延迟，当然，数据包的传输效率也就大打折扣，为此，Linux系统中保存了一份路由缓存(Routing Cache)，只要本机曾将数据包传输到远程的某一个IP，系统就会将前往这个IP的路由记录保存到路由缓存之中，以后如果本机要再次将数据包传输到这个IP，系统就会根据路由缓存中的信息将数据包发送出去，以节省系统判断路由路径的时间，从而达到加快发送数据包的目的。我们可以使用ip route show cache命令来查看路由缓存内的信息，以下示例就是本机前往168.95.1.1时所经由的网关，地址是10.10.15.1。

```
[root@localhost ~]# ip route show cache
local 10.10.15.35 from 10.10.15.26 dev lo src 10.10.15.35
  cache <local,src-direct> iif eth0
168.95.192.1 via 10.10.15.1 dev eth0 src 10.10.15.35
  cache mtu 1500 advmss 1460 hoplimit 64
168.95.1.1 dev ppp0 src 111.70.219.194
  cache mtu 1500 advmss 1460 hoplimit 64
local 10.10.15.35 from 168.95.192.1 dev lo src 10.10.15.35
```

```
cache <local> iif eth0
168.95.192.1 from 10.10.15.35 via 10.10.15.1 dev eth0
cache mtu 1500 advmss 1460 hoplimit 64
10.10.15.26 from 10.10.15.35 tos lowdelay dev eth0
cache mtu 1500 advmss 1460 hoplimit 64
```

因为存在路由缓存机制，因此，FTP在被动模式下传输文件到因特网时，绝对不会有命令通道经由实体线路A，而DATA通道经由实体线路B的现象发生。

虽然以上问题不会存在，但仍要告诉你一个小秘密，上面这些操作在Kernel 2.4的系统上可以正常执行，但如果你所使用的系统是Kernel 2.6，那就没有那么幸运了！我曾遇到一个非常奇怪的现象，在Kernel 2.6系统上进行带宽合并时，所有客户端主机都可以正常连上因特网；但在客户端主机上使用SSH协议连接到因特网时，却发生了SSH协议无法正常工作的问题。很久才找到了原因，不知道为什么，SSH协议的数据包不只经由一条实体线路，例如，有一条SSH协议的连接经由实体线路A，但奇怪的是，在传输了几个数据包之后，突然就将数据包往实体线路B上发送，因而导致SSH协议连接失败。

关于上述问题，下面再说明一下。以图10-23为例，假设负载均衡器上连接两条独立的ADSL，分别称为实体线路①和实体线路②，在正常情况下，当客户端主机通过负载均衡器连上因特网时，负载均衡器会帮其选择①或②的路径，如果负载均衡器选择路径①，那么送往因特网的数据包内的来源端IP一定就是实体线路①的IP，而这条访问因特网服务的会话(Session)上的所有数据包都会经由实体线路①来传输，如果是这样，客户端就可以正常访问因特网。

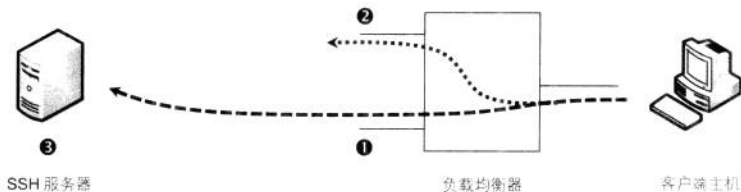


图10-23 SSH连接失败的原因

但我所遇到的奇怪现象是，如果客户端以Linux系统上SSH客户端来连接因特网，一开始数据包通过实体线路①来传输，但在某些特定情况下，原来应该经由实体线路①的数据包，却反常地往实体线路②上发送，而这些往实体线路②发送的数据包其内的IP仍然是实体线路①的IP，但这些数据包会因为ISP的封锁而无法传输到SSH服务器，最后导致连接失败。

再强调一点，如果ISP没有进行这样的限制，其实客户端是可以正常连接SSH服务器的，那么，ISP到底限制了什么？又为什么要这么做呢？我猜测，ISP的限制是这样的，如果ISP租给我们的ADSL所分配的IP是w.x.y.z，ISP就会限制“凡是从这条ADSL送入ISP机房的数

据包,其内容来源端的IP必须是w.x.y.z;如果不是,就将数据包丢弃”,至于ISP这样的限制是否合理,是见仁见智的。如果从正面的角度来看这件事,我认为ISP的目的是防止其客户对因特网实施DDOS攻击(攻击是会不断更改数据包内的来源端IP),如果从负面角度来看,ISP可能是在限制客户只能使用同一家ISP的产品,此话怎讲呢?如果图10-23是被ISP所接受的,那么,请你将客户端主机换成一台Web服务器试试,是否就可将Web服务器对外的流量非常平均地分摊在两条ADSL上呢?如此一来,就可以把低价的带宽合并成一条高速线路了,那ISP的高级产品又要卖给谁呢?

在此要披露一个小秘密, Linux中有一个名为Equalize的内核补丁,可以帮助我们对外流量平均分摊到每条实体线路上,如果你的ISP没有施加相应限制,这个方法就是可行的。

至于问题的原因,我怀疑是内核的Bug,为什么会这么说呢?因为已经在IP路由规则中作了规定,如下脚本的①与②,但内核似乎没有把特定的数据包转发给特定的路由表处理,不过也怪,为什么只有SSH协议会这样?还是我的测试不够周密呢?没关系,山不转,路转,路不转,人转,总会有解决办法的。

```
#!/bin/bash

ip rule add from 10.10.15.35 table 10           ❶
ip rule add from 111.82.149.58 table 20         ❷

ip route add 10.10.15.0/25 dev eth0 table 10
ip route add default via 10.10.15.1 dev eth0 table 10

ip route add default via 111.82.149.58 dev ppp0 table 20

ip route replace default \
                                nexthop via 10.10.15.1 dev eth0 weight 1 \
                                nexthop via 111.82.149.58 dev ppp0 weight 1

iptables -t nat -A POSTROUTING -o eth0 -s 192.168.122.0/24 \
-j SNAT --to 10.10.15.35
iptables -t nat -A POSTROUTING -o ppp0 -s 192.168.122.0/24 \
-j SNAT --to 111.82.149.58
```

以下就是将图10-20的示例修改后的内容,其中最主要的差异在于将路由规则改成依据fwmark来判断,即①和②的命令列,此外,③④⑤⑥行命令则是用于标记数据包的fwmark值,此处将从eth0送出的数据包标记为fwmark 1,从ppp0送出的数据包标记为fwmark 2。

```
#!/bin/bash

ip rule add fwmark 1 table 10           ❶
ip rule add fwmark 2 table 20         ❷
```



```

#ip route add 10.10.15.0/25 dev eth0 table F0
ip route add default via 10.10.15.1 dev eth0 table 10

ip route add default via 111.82.149.58 dev ppp0 table 20

ip route replace default: \
    nexthop via 10.10.15.1 dev eth0 weight 1 \
    nexthop via 111.82.149.58 dev ppp0 weight 1

iptables -t nat -A POSTROUTING -o eth0 -s 192.168.122.0/24 \
    -j SNAT --to 10.10.15.35
iptables -t nat -A POSTROUTING -o ppp0 -s 192.168.122.0/24 \
    -j SNAT --to 111.82.149.58

iptables -t mangle -A POSTROUTING -o eth0 -m state --state NEW \
    -j CONNMARK --set-mark 1 ③
iptables -t mangle -A POSTROUTING -o ppp0 -m state --state NEW \
    -j CONNMARK --set-mark 2 ④
iptables -t mangle -A PREROUTING -i eth1 -m connmark --mark 1 \
    -j MARK --set-mark 1
iptables -t mangle -A PREROUTING -i eth1 -m connmark --mark 2 \
    -j MARK --set-mark 2

```



提示

如果要回顾CONNMARK、MARK的处理方法及connmark的匹配方式，请返回阅读第3章的内容。

3 更加智能化的负载均衡器

虽然负载均衡器已经可以正常工作了，但还不够完美，因为负载均衡器还是以循环(Run Robin)方式来决定哪个会话(Session)该经由哪条实体线路。因此在这种模式中，依然很容易造成堵塞的假象。我们以图10-23为例进行说明，请假设两种情况：

不同来源端的堵塞

假设内部主机要针对因特网建立三条连接，而这三条连接的用途分别是：①使用FTP下载4GB的文件，②使用ping命令来测试网络的连接，③使用http协议下载3GB的文件。当第①条连接产生时，会被安排到ISP 1的线路上；当第②条连接产生时，则被安排到ISP 2的线路上；当第③条连接产生时，则被安排到ISP2的线路上。

从以上流程可以看到循环(Run Robin)模式的问题，因为第①及第②条连接都会被安排到同一条线路上，而使用ISP 1的实体线路堵塞，但ISP 2的线路却没什么流量。这个结果就是循环模式的致命伤，无法改变。

4. 相同来源端的拥塞

例如，内部有两台主机要从因特网下载同一个文件，我们假设第一条连接是经由ISP 1的线路，请问第二条连接会经由哪条线路呢？答案不用怀疑一定是ISP 1，为什么会是ISP 1呢？难道不会被安排在ISP 2吗？别忘了，稍早之前曾介绍过系统内有一个路由缓存机制，因为目的端都是同一个，因此，这两条连接都会经由ISP 1的线路，如此ISP 1的线路就会形成拥塞，但ISP 2却没什么流量。

有办法解决这个问题吗？答案是有，但不完美！我们可以使用`ip route flush cache`命令来清除路由缓存内的数据，但又不能无时无刻地清除缓存。而且缓存一旦被清除掉，系统又需要为新的连接来定义路由，反而会降低系统性能。

以上两个问题都是大多数负载均衡器的弊病，不过，其实有更聪明的解决办法，我们可以根据线路的负载情况来决定要使用哪条实体线路来传输数据，而这个强大功能是通过Netfilter的模块来达到的，这幕后功臣分别是RATEEST的处理方式，以及`rateest`的匹配方法。

此处以图10-24为例来说明如何配置这个“更加智能化的负载均衡器”，我们在设置Linux基于策略的路由时，最好把一个名为逆向路径过滤(Retrieve Path Filtering)的机制关闭，以免干扰Linux基于策略的路由的功能。可使用以下脚本将其关闭。

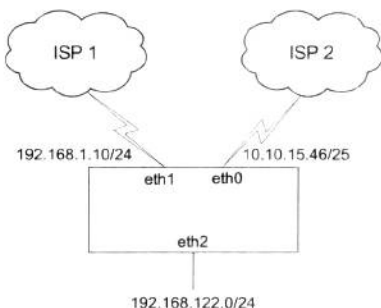


图10-24 负载均衡器连接分配的问题

```
for i in /proc/sys/net/ipv4/conf/*/rp_filter
do
    echo 1 > $i
done
```

我们要根据网络带宽的使用量来决定新连接要用哪条实体链路来传输数据，因此，需要一个能用来统计流量的机制，这个机制就是使用Netfilter的RATEEST模块，使用如下脚本，我们可在mangle Table中的PREROUTING进行流量统计。分别针对从eth0及eth1进入的流量，可以使用`--rateest-ewma 1s`(单位为秒)参数来设置“统计值的时间范围”。例如，每秒钟统计一次，另外，`--rateest-interval 500ms`(时间单位是s、ms及us)参数则设置在这1秒钟内的采样次数。当然，采样次数越多，最后得到的值就越精确。最后得到前一秒钟的带宽使用率，就使用`--rateest-name RATE_ETH0`参数来保存到RATE_ETH0数据库中。

```
iptables -t mangle -A PREROUTING -i eth0 -j RATEEST \
--rateest-name RATE_ETH0 \
--rateest-interval 500ms \
```

```
--rateest-ewma 1s

iptables -t mangle -A PREROUTING -i eth1 -j RATEEST \
--rateest-name RATE_ETH1 \
--rateest-interval 500ms \
--rateest-ewma 1s
```

有了带宽使用统计值后，就可以有比较的依据来决定新的连接该经由哪条线路，而比较操作就交给rateest模块来完成。接着看下面的脚本，如前所述：“经过ISP实体线路送到ISP的数据包，如果数据包内的来源端不属于这条实体线路的IP，那么这个数据包将被丢弃”，因此，我们在决定一条新的连接要经过哪条实体线路时，必须从一条连接的第一个数据包开始处理，否则可能造成连接中断问题。因此，这里使用-i eth2 -m state --state NEW来指定数据包是由内部网路送往因特网，而且是连接的第一个数据包才要进行处理。

接着使用-m rateest -rateest-delta指定由rateest模块来执行流量匹配的操作。要匹配什么呢？--rateest1 RATE_ETH0及--rateest2 RATE_ETH1是指定要被拿来匹配用的数据来自数据库RATE_ETH0及RATE_ETH1，但单纯匹配流量的大小是没有任何意义的。例如，有A、B两条实体线路，A的带宽是10M Bps，但B的带宽是2M Bps，此时若A的使用率是40%，但B的使用率只有10%，请问，新的连接产生时，要安排走实体线路A还是B呢？另外，由于这个参数的后面还要指定这条实体线路的带宽是多少(如有两条4M/1M的ADSL)，因此需要用参数--rateest-bps1 4mbit及--rateest-bps2 4mbit来设置两条实体线路的带宽都是4M Bps。

另外，--rateest-gt是大于的意思，由于在本例将RATE_ETH0放在RATE_ETH1之前，因此，这个语法是指当RATE_ETH0大于RATE_ETH1时，就把连接第一个数据包mark值设置为1；而第二条规则指出，如果RATE_ETH1大于RATE_ETH0，就把连接第一个数据包的mark值设置为2。

```
iptables -t mangle -A PREROUTING -i eth2 -m state --state NEW \
-m rateest --rateest-delta \
--rateest1 RATE_ETH0 --rateest-bps1 4mbit \
--rateest-gt \
--rateest2 RATE_ETH1 --rateest-bps2 4mbit \
-j MARK --set-mark 1

iptables -t mangle -A PREROUTING -i eth2 -m state --state NEW \
-m rateest --rateest-delta \
--rateest1 RATE_ETH1 --rateest-bps1 4mbit \
--rateest-gt \
--rateest2 RATE_ETH0 --rateest-bps2 4mbit \
-j MARK --set-mark 2
```

至此, 每一条要连到因特网的连接的第一个数据包, 都已经被标记上了 `nfmark`, 第3.1.2节的第20部分曾提过: “`nfmark`的有效范围仅限于单个数据包”, 因此, 当第一个数据包被标记 `mark` 值之后, 再使用以下命令将 `mark` 值复制下来, 而一条连接在第一个数据包之后的所有数据包, 都按照 `CONNMARK` 处理方式将之前复制的 `mark` 值, 逐一粘贴到该连接的每个数据包中。

```
iptables -t mangle -A PREROUTING -i eth2 -m state --state NEW \
    -j CONNMARK --save-mark
iptables -t mangle -A PREROUTING -i eth2 -j CONNMARK --restore-mark
```

最后还需要建立两条路由规则。如下, 我们使用 `fwmark(nfmark)` 作为判断条件, 把标记有 `mark` 值为1的数据包导入到 `table 1` 的路由表, 而 `mark` 值为2的数据包则导入到 `table 2` 的路由表, 但也别忘了将路由表1和路由表2的内容设置好。

```
ip rule add fwmark 1 table 1
ip rule add fwmark 2 table 2

ip route add default via 10.10.15.1 dev eth0 table 1
ip route add default via 192.168.1.254 dev eth1 table 2
```

另外, 在设置一对多NAT时, 请运行 `SNAT` 把从 `eth0` 接口送出去的数据包的来源端IP改成 `10.10.15.46`, 再把 `eth1` 接口送出去的数据包的来源端IP改为 `192.168.1.10`。至此止, “更加智能化的负载均衡器” 就完成设置了。

```
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.122.0/24 \
    -j SNAT --to 10.10.15.46
iptables -t nat -A POSTROUTING -o eth1 -s 192.168.122.0/24 \
    -j SNAT --to 192.168.1.10
```

最后, 将以上零散的脚本整理成下面完整的内容:

```
#!/bin/bash

for i in /proc/sys/net/ipv4/conf/*/rp_filter
do
    echo 1 > $i
done

iptables -t nat -F
iptables -t mangle -F

iptables -t mangle -A PREROUTING -i eth0 -j RATEEST --rateest-name RATE_ETH0 \
    --rateest-interval 500ms --rateest-ewma 1s
```

```

iptables -t mangle -A PREROUTING -i eth1 -j RATEEST --rateest-name RATE_ETH1 \
    --rateest-interval 500ms --rateest-ewma 1s

iptables -t mangle -A PREROUTING -i eth2 -m state --state NEW \
    -m rateest --rateest-delta \
    --rateest1 RATE_ETH0 --rateest-bps1 1mbit \
    --rateest-gt \
    --rateest2 RATE_ETH1 --rateest-bps2 1mbit \
    -j MARK --set-mark 1

iptables -t mangle -A PREROUTING -i eth2 -m state --state NEW \
    -m rateest --rateest-delta \
    --rateest1 RATE_ETH1 --rateest-bps1 4mbit \
    --rateest-gt \
    --rateest2 RATE_ETH0 --rateest-bps2 4mbit \
    -j MARK --set-mark 2

iptables -t mangle -A PREROUTING -i eth2 -m state --state NEW \
    -j CONNMARK --save-mark
iptables -t mangle -A PREROUTING -i eth2 -j CONNMARK --restore-mark

ip rule add fwmark 1 table 1
ip rule add fwmark 2 table 2

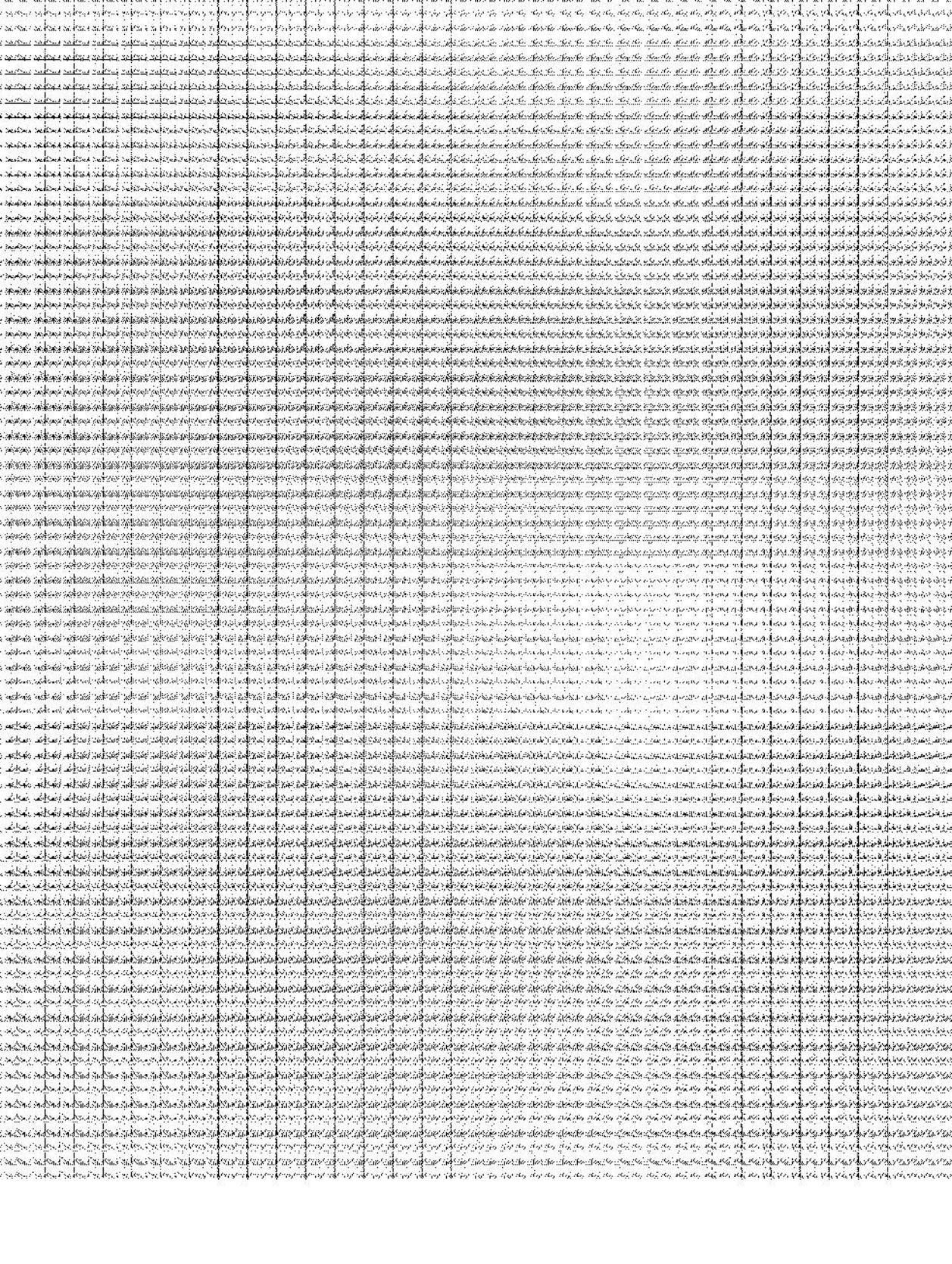
ip route add default via 10.10.15.1 dev eth0 table 1
ip route add default via 192.168.1.254 dev eth1 table 2

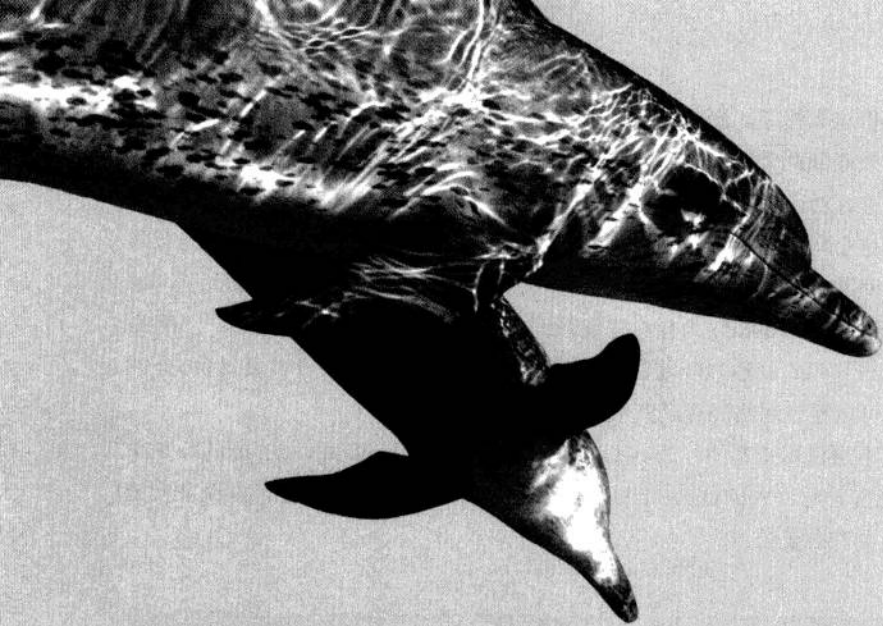
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.122.0/24 \
    -j SNAT --to 10.10.15.46
iptables -t nat -A POSTROUTING -o eth1 -s 192.168.122.0/24 \
    -j SNAT --to 192.168.10

```

10.5 小结

本章介绍了Linux系统的基于策略的路由机制，相信这个强大机制一定给你留下极为深刻的印象！其实市场上所谓的负载均衡设备都以Linux的基于策略的路由机制为核心，然后设计出一个美观的网页管理程序让我们来操作而已，如果你了解了本章的内容，相信你完全可以自己实现所需的功能，另外需要指出，Linux基于策略的路由机制在有些功能上可是很多商用路由器都所望尘莫及的！





Linux

|第11章| Linux的带宽管理

11

“为什么网络又变得这么慢了，到底是谁在下载文件？”我想，这句话是网络管理人员最常听到的抱怨声，如果遇到此类情况，该如何处理呢？难道真要使用网络流量分析软件来寻找“凶手”，然后再进行劝说吗？若真是这样，那网络工程师也未免太命苦了吧！

带宽管理正是解决这个问题的良方，我们可以使用带宽管理工具来有效划分企业的有限带宽，如此便可以确保网络带宽的合理使用。或许有人会说，现在网络带宽都这么大了，动不动就是10M、20M甚至50M、100M，为什么需要管理带宽呢？第10章中曾提到过：“网络带宽有限，人的欲望无穷”吗？记得17年前在BBS网站下载一个40KB的文件需要耗费5分钟时间，而现在我们已经可以通过网络观看高清电影了。

为了避免企业内部不当使用带宽的行为，如P2P下载软件(foxy、eMule、eDonkey)、P2P视频软件(PPStream、迅雷看看、YouTube、土豆网)等占用大量带宽的网络服务，在企业实施带宽管理机制其实是有很有必要的。

11.1 队列

如图11-1，计算机系统为了提升I/O性能，而设计了队列(Queue)，队列通常是一块系统的内存空间，这个空间内，暂存了准备往外发送的数据。例如，系统上有多个程序同时访问网络时，这些准备通过网卡发送到网络上的数据包，就会先暂存在队列中，但有这么多程序同时在访问网络，请问会优先传输哪一个程序的数据呢？



图11-1 队列

这时就不得不提到队列算法了。不同的队列算法提供不同的数据传输规则，例如在kernel 2.6.35版本中，网络接口默认的队列算法是pfifo_fast，pfifo_fast队列算法分为三个band，称为band0、band1和band2，我们可以将其想象成三个通道，而每个通道都是采取“先进先出”的策略，也就是先送进队列的数据先被发送，另外这三个队列还有优先级的差别，也就是band0通道内的数据传输完之后，才会轮到band1来传输数据。当然，band1通道内的数据传输完之后，才会轮到band2来传输数据，如此就可以把不同的数据送到不同的band，由此达到某种程度上带宽管理，只是这太简单了，不是我们要讨论的重点。

Linux系统内置了几种不同的队列算法，这些队列算法基本可分为“不可分类(ClassLess QDisc)”及“可分类(ClassFUL QDisc)”的队列算法，前者不能或不适于进行带宽管理，而

后者则是专为带宽管理而设计的。

11.1.1 不可分类的队列算法

Linux系统本身内置了非常多的队列算法，如果有兴趣的话，在内核源代码目录下执行 `make menuconfig` 命令，然后可以在 Networking support | Networking options | Qos and/or fair queuing 选择菜单中看到其所内置的所有网络队列算法。下面列出一些较常见的队列算法：

- **bfifo/pfifo**：这是Linux系统内置最简单的队列算法，是采用先进先出的方式来处理数据包发送的先后顺序。
- **pfifo_fast**：如前所述，在 **pfifo_fast** 队列算法中划分了三个band，称为band0、band1及band2，我们可以将其想象成三个通道，而每个通道都是采取“先进先出”的策略，也就是先送进队列的数据先被发送，另外这三个队列还有优先级的差别，也就是band0通道内的数据传输完之后，才会轮到band1来传输数据。当然，band1通道内的数据传输完之后，才会轮到band2来传输数据。
- **red**：red是Random Early Detection 的首字母缩写词，也就是在流量大的时候，以随机的方式丢弃掉一些数据包，通常是应用在骨干网络上。
- **sfq**：sfq是Stochastic Fairness Queueing的缩写，又被称为“随机公平队列”，其做法是将队列分为多个通道，并且将对外发送数据的会话(Session)，以随机的方式安排在不同通道中，每个通道都会有固定的数据发送时间，由此达到“公平”的目的。
- **tbf**：tbf是Token Bucket Filter的缩写，tbf算法用来限制网络流量，它限制的依据是“数据包/时间”，其做法有点类似hashlimit模块。

11.1.2 可分类的队列算法

与不可分类的队列算法一样，Linux系统中也内置了多种可分类的队列算法，一般最常用的有CBQ及HTB这两种，这两种队列算法都有人使用，且这两种队列算法的用法都差不多。不过，由于HTB算法比CBQ算法更晚诞生，所以HTB算法在各方面的表现都比CBQ算法要好，最重要的是，HTB算法更简单，学习起来比CBQ更容易上手，因此，本章内容将以HTB算法为例展开讨论。

图11-2是Linux系统默认的网络队列结构，其中所有的数据包都根据 **pfifo_fast** 队列算法来决定其传输的优先级别，只不过 **pfifo_fast** 队列的带宽管理能力很薄弱，以至于可以忽视其存在。

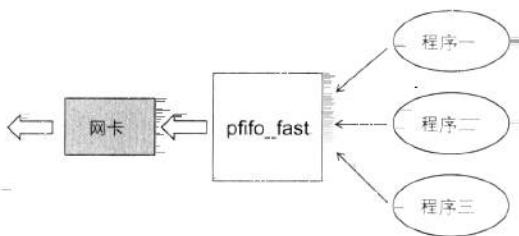


图11-2 默认的队列

为了拥有更完整的带宽管理机制，可以将队列算法更换为CBQ或HTB，在CBQ与HTB队列算法中，都以树状结构来划分带宽，带宽划分方式如图11-3所示。图中类别一的位置设置了对外的总带宽大小为10MB/s，接着，下面有两个分支分别是类别二和类别三，其中类别二设置为2MB/s，类别三设置为8MB/s；在类别三的位置又向下分成类别四和类别五，类别四设置为3MB/s，类别五则是5MB/s。

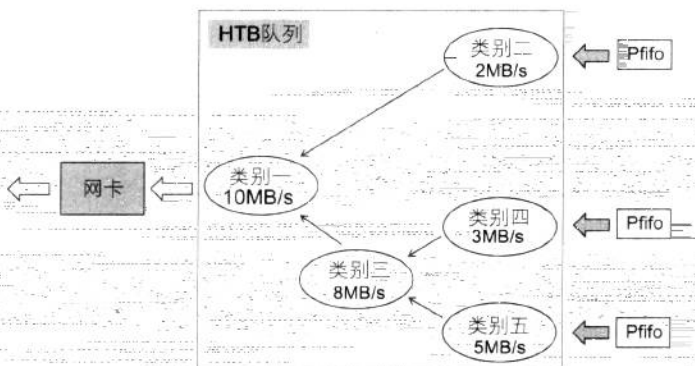


图11-3 HTB队列

我们可以想象，如果希望HTTP数据包最多只能占用5MB的带宽，那么，只要将HTTP数据包送入“类别五”，即可受到5MB上限的限制；如果是送到“类别四”，则会受到上限3MB的限制；送到“类别二”，则是受到上限2MB的限制。

11.2 Linux带宽管理

如图11-4所示，带宽管理结构主要分为“队列算法”及“过滤器(Filter)”两个部分，前者在于划分带宽，而后者功能在于将数据包分类并注入到特定的类别中。例如，可以在过

滤波器的位置设置http协议的数据包要送到类别二来处理，smtp协议的数据包要送到类别四处理，pop3协议的数据包要送到类别五来处理，至于这些协议可以使用多少带宽？就要看各个类别上的带宽划分了。

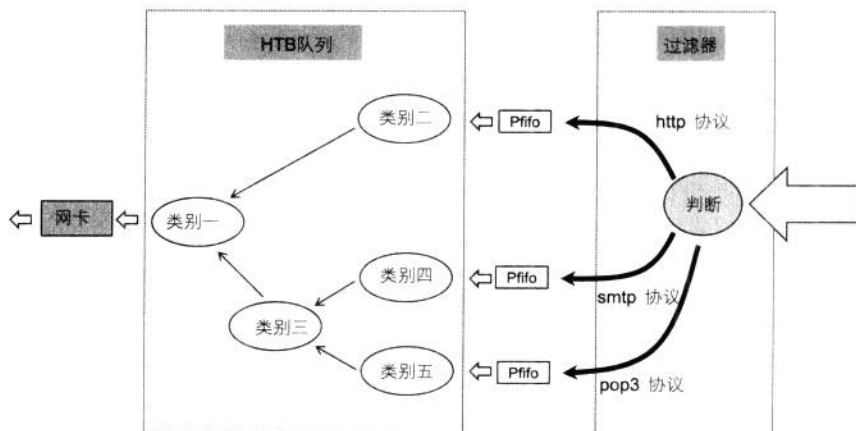


图11-4 QoS结构

另外有一点要特别注意，以上带宽管理机制只能管理到“数据包往外发送”所占用的带宽，并无法管理从网络上送入的数据包流量。道理很简单，因为CBQ及HTB队列是用来暂存“往外发送”的数据包，因此，只能限制送出去的数据流量，而无法管理送入的数据流量。

讲到这里或许你心里又有疑惑，如果我们无法管理从网络上送入的数据流量，又该如何限制使用者访问网络的带宽呢？其实这个问题并不难，如图11-5，如果我们要限制内部网络到因特网的带宽，便可以在带宽管理设备的eth0上设置限制；如果要限制内部使用者从因特网下载数据所能使用的带宽，可以在eth1上进行带宽的限制。因此，我们并不需要担心这个问题。

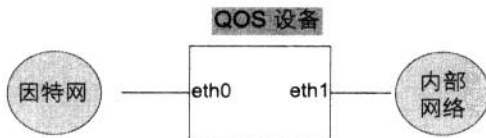


图11-5 QoS使用示例

11.3

Linux系统中内置了几种不同的过滤器(Filter)，较常见的有fw过滤器及U32过滤器。

11.3.1 FW过滤器

FW过滤器通过Netfilter的模块来分类数据包,例如,使用-p tcp --dport 80 -j MARK --set-mark 80语法来对所有目的端口为80的数据包标记MARK值。Netfilter的模块相当丰富,例如,我们可以使用state模块来识别NEW、ESTABLISHED、RELATED状态的数据包,甚至可以使用connmark及CONNMARK来识别相关的其他连接,还可以使用u32模块来定义数据包的特征并进行分类,因此我认为没有什么比Netfilter更好的过滤器可以选择了。

11.3.2 U32过滤器

U32过滤器可以由系统管理人员使用U32过滤器的语法来判断数据包包头中的任何数据(这个功能有点像Netfilter的u32模块),但管理员必须对数据包包头的结构非常熟悉才行,否则将无法正确地使用U32过滤器。此外,U32只能对单个的数据包进行分类,无法像FW过滤器能对一条连接进行分类,使用上有诸多限制,因此并不推荐使用U32过滤器。

11.4 带宽管理部署示例

设置带宽管理大概可以分为几个流程。首先要决定如何划分带宽,接着是选择所要使用的队列算法及设置队列规则,最后则是设置过滤器以及进行测试。由于带宽管理的设置步骤很多,这里将其分为四个步骤,并使用以下网络结构作为测试环境:

如图11-6所示,假设有三兄弟共同出资申请了一条1M/512K的ADSL,由于大哥的收入较高,所以支付了较多的费用,其次是二哥,由于小弟年纪较小,且没有任何收入,因此小弟不需要支付任何的费用。因为付出的费用不同,兄弟们也达成了一项协议,在这1M的带宽中,保证大哥及二哥分别拥有60KB及40KB的带宽,而小弟只能保证拥有28KB的带宽。有了以上条件之后,下面开始设置带宽管理的机制。

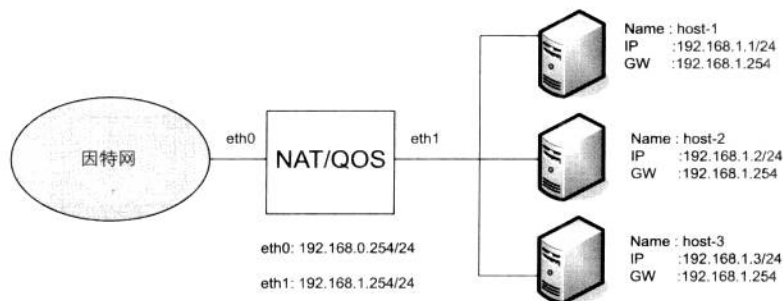


图11-6 QOS示例(一)

11.4.1 带宽划分

根据上述条件画出如图11-7的分类结构图。其中Class 1定义eth1对外的输出总带宽为128KB(1M bps)，接着，再把这128KB划分给Class 2、Class 3、Class 4三个分类，Class 2是大哥所能使用的带宽，Class 3是二哥所能使用的带宽，Class 4是小弟所能使用的带宽。

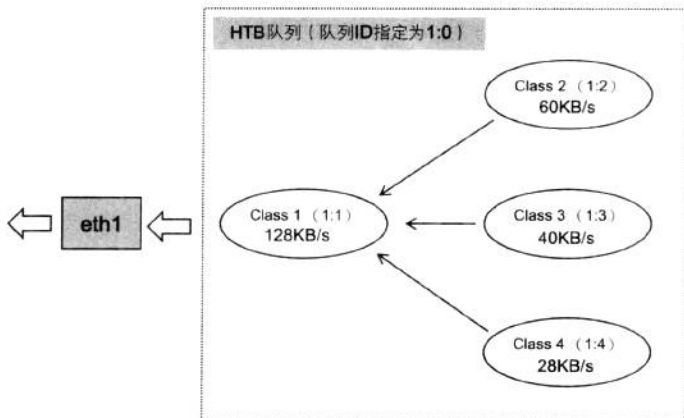


图11-7 分类结构图

此外，系统可能同时存在多个队列及类别，在Linux系统中，这些队列及类别都按照ID来命名，其格式是X:X(如2:3)，另外，队列与类别、或者类别与类别之间都是具有相关性的，而这个相关性就由ID的第一个字符来保证，例如在系统上有两个HTB队列，我们分别定义其ID是1:0及2:0，再假设1:0队列下有四个类别，这四个类别可以将之命名为1:1、1:2、1:3、1:4，其中第一个字符都是“1”，因此，这四个类别都是属于1:0队列下的类别，而:1、:2、:3、:4则代表四个类别的ID。

虽然类别与类别之间有层级的关系存在，但这个层级的关系与ID的名称完全无关，这点请特别注意！类别与类别之间的层级关系并不是由ID来决定的，而是在建立类别的时候特别设置的，关于这个问题稍后再加以讨论。

11.4.2 设置队列算法

如同Netfilter与iptables的关系，队列算法是内核中决定数据包传输顺序的机制，至于内核要使用什么队列算法，则需要一个User Space的管理工具，这个工具称为Traffic Control命令，简称tc。

```

[root@localhost ~]# tc qdisc-show ❶
qdisc pfifo_fast 0: dev eth0 root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: dev eth1 root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
[root@localhost ~]#
[root@localhost ~]# tc qdisc show dev eth1 ❷
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# tc qdisc add dev eth1 root handle 1:0 htb ❸
[root@localhost ~]#
[root@localhost ~]# tc qdisc show dev eth1 ❹
qdisc htb 1: root refcnt 2 r2q 10 default 0 direct_packets_stat 0
[root@localhost ~]#
[root@localhost ~]# tc qdisc del dev eth1 root ❺
[root@localhost ~]#
[root@localhost ~]# tc qdisc show dev eth1 ❻
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
[root@localhost ~]#

```

我们用上面的示例来说明tc命令的用法。可以使用tc qdisc show命令来查看目前系统上所有网卡所使用的队列算法❶，如果只是要查看特定网卡的队列，可以使用tc qdisc show dev eth1命令❷，其中pfifo_fast是系统默认的队列算法。此外可以使用tc qdisc add dev eth1 root handle 1:0 htb命令将eth1接口的队列算法改成htb，并设置这个队列的ID为1:0，其中root指出要改变的是根队列算法❸。

改变完毕后，再使用tc qdisc show eth1来查看，这时可以看到队列算法已经改为htb❹。当我们不想使用htb算法时，可以使用tc qdisc del dev eth1 root命令来删除htb队列算法❺，在删除htb队列算法后，就可以看到eth1的队列算法又恢复到默认的队列算法。

11.4.3 设置队列规则

在这个步骤中，我们根据图11-7设置类别，而这个操作依然需要通过tc命令来完成，在此使用如下的操作实例来说明如何通过tc命令来设置类别：

```

[root@localhost /]# tc class add dev eth1 parent 1:0 classid 1:1 htb-rate 128kbps ❶
[root@localhost /]#
[root@localhost /]# tc class add dev eth1 parent 1:1 classid 1:2 htb-rate 60kbps ❷
[root@localhost /]#
[root@localhost /]# tc class add dev eth1 parent 1:1 classid 1:3 htb-rate 40kbps ❸
[root@localhost /]#
[root@localhost /]# tc class add dev eth1 parent 1:1 classid 1:4 htb-rate 28kbps ❹
[root@localhost /]#
[root@localhost /]# tc class show dev eth1 ❺

```

```

class htb 1:1 root rate 1024Kbit ceil 1024Kbit burst 1599b cburst 1599b
class htb 1:2 parent 1:1 prio 0 rate 480000bit ceil 480000bit burst 1599b cburst 1599b
class htb 1:3 parent 1:1 prio 0 rate 320000bit ceil 320000bit burst 1600b cburst 1600b
class htb 1:4 parent 1:1 prio 0 rate 224000bit ceil 224000bit burst 1599b cburst 1599b
[root@localhost ~]#
[root@localhost ~]# tc class del classid 1:4 dev eth1      ⑥
[root@localhost ~]#
[root@localhost ~]# tc class show dev eth1                ⑦
class htb 1:1 root rate 1024Kbit ceil 1024Kbit burst 1599b cburst 1599b
class htb 1:2 parent 1:1 prio 0 rate 480000bit ceil 480000bit burst 1599b cburst 1599b
class htb 1:3 parent 1:1 prio 0 rate 320000bit ceil 320000bit burst 1600b cburst 1600b
[root@localhost ~]#

```

请参考图11-7，图中htb队列算法已经在上面一个步骤中设置完成了，且已经将其ID设置为1:0，接下来要在1:0这个队列中建立类别，这里准备将这个类别的ID设置为1:1，在此将使用tc class add dev eth1 parent 1:0 classid 1:1 htb rate 128kbps命令来完成①，其中parent 1:0指出这个类别是在1:0层级下面；classid 1:1则设置这个类别的ID，在此设置为1:1；rate 128kbps则是设置这个类别的带宽上限是1mbps。

在设置完Class1之后，我们分别使用②、③、④步骤将Class1:2、Class 1:3及Class 1:4设置完成，基本上，这些语法与步骤①大致相同，不过要特别注意，这三个Class的上层都是Class ID 1:1，请不要写错了。

在设置完类别之后，可以使用tc class show dev eth1命令将eth1接口下的所有类别列出⑤，如果不需要哪个类别，可以使用tc class del classid 1:4 dev eth1命令⑥，将Class ID 1:4的类别删除。有一点请注意，如果将队列1:0删除，原来挂在这个队列下的类别也会一并被删除。

11.4.4 设置过滤器

这是带宽管理设置的最后一个步骤，在这个步骤中，需要进行数据包分类，以及将分类后的数据包导入到特定的类别中。这两个操作的具体说明分别如下。

1. 数据包分类

在介绍下面示例之前，我们先来复习一下MARK的用法。一般的模块在匹配到数据包之后，该数据包就不再进行Netfilter的其他匹配，这就是我们之前说过的“优先匹配”，但MARK这个模块比较特殊，MARK模块在数据包设置完MARK值之后，数据包还会进行Netfilter的其他匹配，请特别注意这一点！

```
[root@localhost ~]#iptables -A FORWARD -i eth0 -d 192.168.1.1 \
-j MARK --set-mark 1 ❶
[root@localhost ~]#iptables -A FORWARD -i eth0 -d 192.168.1.2 \
-j MARK --set-mark 2 ❷
[root@localhost ~]#iptables -A FORWARD -i eth0 -d 192.168.1.3 \
-j MARK --set-mark 3 ❸
```

可以使用上面的Netfilter语法,对于从eth0进入且要送到192.168.1.1主机的数据包,设置其MARK值为1❶,对于送到192.168.1.2的数据包,设置其MARK值为2❷,对于送到192.168.1.3的数据包,设置其MARK值为3❸,这样就可以把三兄弟的数据包根据MARK值分为三类。

2. 将分类后的数据包导入到特定的类别

在数据包分类完成之后,接下来就是将不同MARK值的数据包导入到不同的队列规则中(确切地讲,应该说是不同类别class之中),在此可以使用tc filter add dev eth1 parent 1:0 protocol ip handle 1 fw classid 1:2来完成❶,其中filter add就是添加过滤器, parent 1:0表示这个过滤器是挂在1:0这个队列下面的过滤器, handle 1 fw 则指如果数据包的MARK值为1,就将数据包导入到Class ID为1:2的类别中,这样这个数据包就会受到队列规则的控制,从而达到管理带宽的目的。❷、❸与❶的原理相同。

```
[root@localhost ~]#tc filter add dev eth1 parent 1:0 protocol ip \
handle 1 fw classid 1:2 ❶
[root@localhost ~]#tc filter add dev eth1 parent 1:0 protocol ip \
handle 2 fw classid 1:3 ❷
[root@localhost ~]#tc filter add dev eth1 parent 1:0 protocol ip \
handle 3 fw classid 1:4 ❸
[root@localhost ~]#
```

11.4.5 测试

完成设置之后,我们就可以来测试目前所设置的带宽管理机制是否符合需要。测试方法是使用能记录当前网络传输速度的软件,我们分别在HOST-1、HOST-2及HOST-3主机上,使用wget命令通过HTTP协议下载因特网上的大文件,并观察它的下载速度,即可了解带宽管理机制是否已经正常工作。

- 在HOST-1主机上的测试结果:

```
[root@localhost ~]#
[root@localhost ~]# wget http://ftp.isu.edu.tw/pub/Linux/Fedora/linux/releases/14/Fedora/
x86_64/iso/Fedora-14-x86_64-DVD.iso
--2011-03-01 17:04:29-- http://ftp.isu.edu.tw/pub/Linux/Fedora/linux/releases/14/Fedora/
x86_64/iso/Fedora-14-x86_64-DVD.iso
正在查找主机 ftp.isu.edu.tw... 140.127.177.17
正在连接 ftp.isu.edu.tw|140.127.177.17|:80... 连上了。
已送出 HTTP 要求, 正在等候应答... 200 OK
长度: 3520802816 (3.3G) [application/octet-stream]
Saving to: `Fedora-14-x86_64-DVD.iso.1'

0% [ ] 4,376,881 55.7K/s eta 17h 35m
[root@localhost ~]#
```

● 在HOST-2主机上的测试结果:

```
[root@localhost ~]#
[root@localhost ~]# wget http://ftp.isu.edu.tw/pub/Linux/Fedora/linux/releases/14/Fedora/
x86_64/iso/Fedora-14-x86_64-DVD.iso
--2011-03-01 17:04:31-- http://ftp.isu.edu.tw/pub/Linux/Fedora/linux/Releases/14/Fedora/
x86_64/iso/Fedora-14-x86_64-DVD.iso
正在查找主机 ftp.isu.edu.tw... 140.127.177.17
正在连接 ftp.isu.edu.tw|140.127.177.17|:80... 连上了。
已送出 HTTP 要求, 正在等候应答... 200 OK
长度: 3520802816 (3.3G) [application/octet-stream]
Saving to: `Fedora-14-x86_64-DVD.iso.1'

0% [ ] 3,126,529 37.1K/s eta 25h 41m
[root@localhost ~]#
```

● 在HOST-3主机上的测试结果:

```
[root@localhost ~]#
[root@localhost ~]# wget http://ftp.isu.edu.tw/pub/Linux/Fedora/linux/releases/14/Fedora/
x86_64/iso/Fedora-14-x86_64-DVD.iso
--2011-03-01 17:04:33-- http://ftp.isu.edu.tw/pub/Linux/Fedora/linux/releases/14/Fedora/
x86_64/iso/Fedora-14-x86_64-DVD.iso
正在查找主机 ftp.isu.edu.tw... 140.127.177.17
正在连接 ftp.isu.edu.tw|140.127.177.17|:80... 连上了。
已送出 HTTP 要求, 正在等候应答... 200 OK
长度: 3520802816 (3.3G) [application/octet-stream]
Saving to: `Fedora-14-x86_64-DVD.iso'

0% [ ] 2,089,993 26.0K/s eta 39h 15m
[root@localhost ~]#
```

从上面的测试结果数据看, 带宽管理似乎已经发挥了作用; 但如果就测试数据来看, 好像与我们所限制的值有一定的差别, 而这个差值大约是4KB。如果你觉得这个误差值可以接受, 那就无需再做其他改变, 如果真的很在意这个问题, 那么就请你适当地修改限制带宽的上限即可。

11.5 带宽借用

带宽管理设置完毕后, 三兄弟使用起来还算很愉快, 毕竟小弟不用出钱就可以使用网络, 虽然慢了一点也没什么好抱怨的, 二哥由于出的钱少一些, 网络慢一点也是理所当然的。直到有一天小弟提出了一个要求: “两个哥哥在家的时间那么短, 难道你们不在家的时候, 网络带宽不能全部给我用吗?” 这一席话引起了工程师大哥的兴趣, 在一番钻研之后, 他将其带宽管理的规则改变为如图11-8的方式:

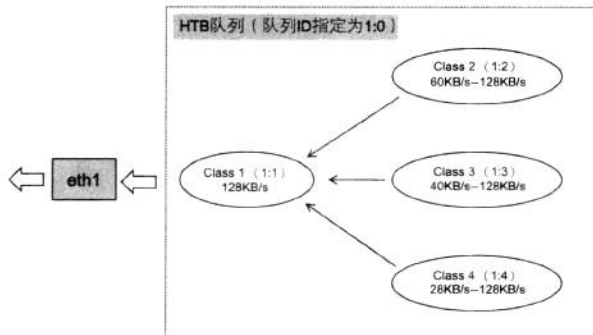


图11-8 类别结构图

图11-7的类别结构图中, 各个类别所拥有的带宽是固定的, 但在HTB队列算法中, 类别所拥有的带宽是可变的, 如图11-8中各类别的带宽就是定义成一个范围, 如Class 4的带宽被设置成28KB~128KB, 意即Class 4的带宽可以保证至少有28KB, 也就是不管网络多么拥塞, Class 4一定可以分到28KB的带宽, 但如果网络没其他人在用, Class 4就有权“借用”空闲的带宽, 而其所能借用到的带宽上限是128KB-28KB=100KB, 如果Class2及Class3都没人在使用网络, Class4能得到的最大带宽就是128KB。不过, 如果大哥回来开始使用网络, 这时小弟原来借用的带宽就会立即“归还”给大哥使用。

以上所描述的都是动态过程, 不太容易使用静态文字来表达, 我们换个方法来解释吧。晚上第一个回家的是小弟, 小弟回家后就打开计算机并连上因特网来下载一个10GB的文件, 这时因为两位大哥还没有回来, 因此下载软件显示目前的现在速度是128KB/s, 看到这

个数字小弟当然很开心；但过一会儿，二哥回来了，二哥也连上因特网下载一个10GB的文件，很快二哥下载文件的速度一下子就飙升到原来的40KB/s；当然这时小弟的下载速度就快速下降。由于大哥还没有回来，所以大哥所占用的60KB的带宽就借给了二哥和小弟来使用，不过，两个人借用的比率可是不相等的！HTB会根据借用者原有的带宽比率来分配，因此，二哥所借用到的带宽大约是 $60 \times 40 / (28 + 40) = 35\text{KB}$ ，小弟借用到的带宽大约是 $60 \times 28 / (28 + 40) = 25\text{KB}$ 。这个时候，大哥回来了，大哥也连上网络下载一个10GB的文件，二哥和小弟的下载速度马上就会被调回到原来的保证带宽。特别值得一提的是，HTB在带宽的借用调度上速度是非常快的，通常大约只需要5秒就可以完成。以下是这个示例的完整脚本：

```
#!/bin/bash
iptables -F

tc qdisc del dev eth1 root
tc qdisc add dev eth1 root handle 1:0 htb

tc class add dev eth1 parent 1:0 classid 1:1 htb rate 128kbps
tc class add dev eth1 parent 1:1 classid 1:2 htb rate 60kbps ceil 128kbps ❶
tc class add dev eth1 parent 1:1 classid 1:3 htb rate 40kbps ceil 128kbps ❷
tc class add dev eth1 parent 1:1 classid 1:4 htb rate 28kbps ceil 128kbps ❸

tc filter add dev eth1 parent 1:0 protocol ip handle 1 fw classid 1:2
tc filter add dev eth1 parent 1:0 protocol ip handle 2 fw classid 1:3
tc filter add dev eth1 parent 1:0 protocol ip handle 3 fw classid 1:4

iptables -A FORWARD -p tcp -i eth0 -d 192.168.1.1 -j MARK --set-mark 1
iptables -A FORWARD -p tcp -i eth0 -d 192.168.1.2 -j MARK --set-mark 2
iptables -A FORWARD -p tcp -i eth0 -d 192.168.1.3 -j MARK --set-mark 3
```

以上脚本与上一个示例的内容基本相同，唯一差别仅在❶❷❸这三行多了“ceil 128kbps”的参数。以rate 60kbps ceil 128kbps为例，其意思是指保证带宽为60kbps，但如有其他空闲的带宽，这个类别所占用的最大带宽可以为128kbps，其中128kbps-60kbps=68kbps是向其他类别借来的。

另外，类别还额外提供一个参数prio，是priority的缩写。我们可以使用tc show class dev eth1命令来查看系统上所有的类别，如果没有特别设置prio参数，系统默认的prio是0，也就是说，当队列有其他闲置带宽时，这些闲置带宽将根据比例分给所有要借用的类别。

```
[root@localhost QOS]# tc class show dev eth1
class htb 1:1 root rate 1024Kbit ceil 1024Kbit burst 1599b cburst 1599b
class htb 1:2 parent 1:1 prio 0 rate 480000bit ceil 1024Kbit burst 1599b cburst 1599b
class htb 1:3 parent 1:1 prio 0 rate 320000bit ceil 1024Kbit burst 1600b cburst 1599b
class htb 1:4 parent 1:1 prio 0 rate 224000bit ceil 1024Kbit burst 1599b cburst 1599b
[root@localhost QOS]#
```

我们将示例三脚本修改如下, 其中的差别在于第❶❷这两行的prio参数上, prio的值由0开始算起, 也是最高优先级。在以下示例中, 将Class 1:4的prio设置为1、Class 1:3的prio设置为2。当Class 1:4需要借用带宽时, Class 1:4将会借用到“所有”闲置带宽, 不管Class 1:3是否有带宽的借用需求。在Class 1:4的带宽借用停止之后, Class 1:3才能借到带宽。因此, 若在带宽的借用上有优先级方面的考虑, 可以使用prio参数。

```
#!/bin/bash

iptables -F

tc qdisc del dev eth1 root
tc qdisc add dev eth1 root handle 1:0 htb

tc class add dev eth1 parent 1:0 classid 1:1 htb-rate 128kbps
tc class add dev eth1 parent 1:1 classid 1:2 htb-rate 60kbps ceil 128kbps
tc class add dev eth1 parent 1:1 classid 1:3 htb-rate 40kbps ceil 128kbps
tc class add dev eth1 parent 1:1 classid 1:4 htb-rate 28kbps ceil 128kbps

tc filter add dev eth1 parent 1:0 protocol ip handle 1 fw classid 1:2
tc filter add dev eth1 parent 1:0 protocol ip handle 2 fw classid 1:3 prio ❶
tc filter add dev eth1 parent 1:0 protocol ip handle 3 fw classid 1:4 prio ❷

iptables -A FORWARD -p tcp -i eth0 -d 192.168.1.1 -j MARK --set-mark 1
iptables -A FORWARD -p tcp -i eth0 -d 192.168.1.2 -j MARK --set-mark 2
iptables -A FORWARD -p tcp -i eth0 -d 192.168.1.3 -j MARK --set-mark 3
```

11.6 类别中的队列

在数据包进入类别时, 如果我们有较为特殊的需求, 可以在类别中加入其他的队列算法。把前面的示例修改如下, 如❶❷❸就是在类别中加入其他队列算法的方法, 如tc qdisc add dev eth1 parent 1:4 handle 300:1 sfq指要将一个sfq队列算法添加到Class 1:4中, 而这个队列的ID是300:1。有了这个队列之后, 就可以通过它来控制进入Class 1:4的规则; 但一般而言, 我们较少有机会去进行这样的控制, 若你有特殊需求, 请参考其他队列的使用方法。

```
#!/bin/bash

iptables -F

tc qdisc del dev eth1 root
tc qdisc add dev eth1 root handle 1:0 htb
```

```

tc class add dev eth1 parent 1:0 classid 1:1 htb rate 128kbps
tc class add dev eth1 parent 1:1 classid 1:2 htb rate 60kbps ceil 128kbps
tc class add dev eth1 parent 1:1 classid 1:3 htb rate 40kbps ceil 128kbps
tc class add dev eth1 parent 1:1 classid 1:4 htb rate 28kbps ceil 128kbps

tc filter add dev eth1 parent 1:0 protocol ip handle 1 fw classid 1:2
tc filter add dev eth1 parent 1:0 protocol ip handle 2 fw classid 1:3
tc filter add dev eth1 parent 1:0 protocol ip handle 3 fw classid 1:4

tc qdisc add dev eth1 parent 1:2 handle 100:1 bfifo          ❶
tc qdisc add dev eth1 parent 1:3 handle 200:1 pfifo          ❷
tc qdisc add dev eth1 parent 1:4 handle 300:1 sfq             ❸

iptables -A FORWARD -p tcp -i eth0 -d 192.168.1.1 -j MARK --set-mark 1
iptables -A FORWARD -p tcp -i eth0 -d 192.168.1.2 -j MARK --set-mark 2
iptables -A FORWARD -p tcp -i eth0 -d 192.168.1.3 -j MARK --set-mark 3

```

11.7 Linux带宽管理的限制

Linux带宽管理虽然功能强大但也并非无所不能。例如，复杂通信协议就是带宽管理所无法做到的，FTP的被动模式便属于这一类。我们复习一下有关FTP协议的工作模式。图11-9显示FTP的主动模式的工作情况，首先，客户端使用端口XX连接到服务器的端口21，并且告诉客户端于端口YY的位置等待服务器来建立数据通道。最后，服务器就使用端口20连接到客户端的端口YY。以上除了端口21及端口20是固定的之外，XX及YY都是随机生成的。

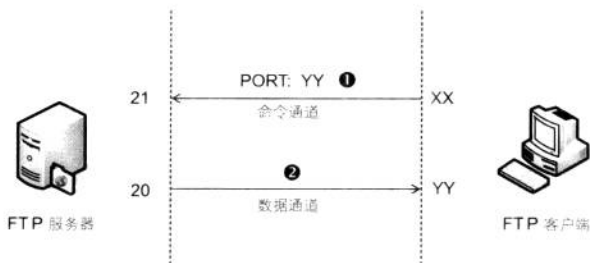


图11-9 FTP主动模式

接下来介绍FTP协议工作在被动模式的情况。首先客户端使用端口XX连接到服务器的端口21，并提出使用被动模式建立数据通道，接着，FTP服务器会告诉在端口ZZ等待客户端来建立数据通道。此时，客户端就会使用端口YY对FTP服务器的端口ZZ建立数据通道。以上除了端口21为固定的之外，端口XX、YY及ZZ都是随机生成的，如图11-10所示。



图11-10 FTP被动模式

了解以上流程后,大概可以推测FTP协议工作于主动模式时,我们仍然可以轻松的管理FTP协议所占用的带宽,这是因为用来传输数据的端口20是固定的,我们只要对端口20进行带宽管理即可。但在被动模式中,由于用来传输数据的端口不是固定的,因此,很难对其进行带宽管理。通过以上例子可以了解到,只要是复杂通信协议,基本上Linux的带宽管理都有困难。

难道FTP协议的被动模式真的无法进行带宽管理吗?其实也未必。以上例子只是给大家一个概念而已,某些复杂通信协议在Linux系统中还是可以进行带宽管理的。不过,在开始说明这个解决办法之前,先来回顾一下“第3.2.7一节中介绍的特殊用法。其中解释了CONNMARK可以为一条连接双向的所有数据包设置MARK值,然而CONNMARK可以做到的不仅仅是这些,在系统已经加载nf_conntrack_ftp模块的情况下,CONNMARK还可以为FTP协议的数据通道一起标记上MARK值。说明白一点,就是Netfilter可以支持复杂通讯协议。因此,CONNMARK可以为你的这些协议的所有数据包标记上MARK值。

以FTP协议为例来说明CONNMARK的特殊使用方式:

```
1. #!/bin/bash
2. modprobe nf_conntrack_ftp
3. iptables -F
4. iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
5. iptables -A INPUT -m connmark --mark 1 -j ACCEPT
6. iptables -A INPUT -p tcp --syn -m state --state NEW -j CONNMARK --set-mark 1
7. iptables -A INPUT -p tcp --syn -m state --state NEW --dport 21 -j ACCEPT
8. iptables -A INPUT -j DROP
```

以上就是一个放在FTP服务器上的单机防火脚的脚本。在脚本的第8行中,将所有送进系统的数据包都丢弃。为了让FTP服务器的端口21可以正常接收到数据包,在第7行的位置上设置了以下规则:只要是给本机端口21的数据包,就让这个数据包送进本机;同时也在第6行的位置设置与第7行相同条件的数据包。我们就在这些数据包上标记MARK值为1,因此客户端到本机端口21这条连接双向的所有数据包都会被标记MARK值为1。但因为在第

2行的位置已经加载了nf_conntrack_ftp的辅助模块，所以CONNMARK会自动判断出FTP协议的数据通道，并为数据通道这条连接双向的所有数据包标记MARK值为1。如此一来，数据通道就可以通过第5条规则返回FTP服务器，当然客户端与FTP服务器之间的FTP协议就可以畅通无阻了。

了解CONNMARK的特殊功能后，这里以FTP协议的被动模式为例来说明如何实现FTP复杂通信协议的带宽管理：

```
#!/bin/bash

modprobe nf_conntrack_ftp

iptables -F

tc qdisc del dev eth1 root
tc qdisc add dev eth1 root handle 1:0 htb

tc class add dev eth1 parent 1:0 classid 1:1 htb rate 128kbps
tc class add dev eth1 parent 1:1 classid 1:2 htb rate 40kbps

tc filter add dev eth1 parent 1:0 protocol ip handle 1 fw classid 1:2

iptables -A INPUT -i eth1 -p tcp --dport 21 -j CONNMARK --set-mark 1 ❶
iptables -A OUTPUT -m connmark --mark 1 -j MARK --set-mark 1 ❷
```

以上脚本定义了一个Class1:2的类别，并将这个类别的带宽设置为40KB/S，而这个脚本之所以能够成功，最重要原因在于nf_conntrack_ftp、connmark及CONNMARK这三个模块的组合使用。首先将所有送到本机tcp端口21的数据包标记MARK值为1(nf_mark)，如此将使得客户端到本机tcp端口21连接双向的所有数据包都会标记MARK值为1(nf_mark)❶。但因为我们已经加载了nf_conntrack_ftp模块，因此CONNMARK模块也会将FTP的数据通道标记MARK值为1(nf_mark)。但在Linux带宽管理中，用来识别的是fw_MARK而不是nf_MARK值。第3章曾经介绍过MARK的处理方法，其所标记的MARK值我们称为fw_mark，而CONNMARK处理方法所标记的MARK值则称为nf_mark。因此，我们在❷这一行上使用connmark的匹配方式来匹配数据包上有nf_mark值为1的，就在数据包上标记fw_mark值为1，如此FTP协议的数据通道就可以很正确地被标记上fw_mark值。

一旦要受管制的数据包正确地标记上fw_mark值，我们就可将此类数据包送入Class 1:2的类别，来进行带宽的管理。或许有读者会问：“如果Netfilter没有提供处理复杂通信协议的模块，带宽管理的操作又该如何来进行呢？”基本上这个问题很简单，就是和filter及NAT机制在处理复杂通讯协议一样，答案是做不到，唯一的办法就是等待Netfilter组织提供

新的模块，但这并不仅仅是我们的困扰而已，而是包括所有商业版的防火墙及带宽管理机制都存在的困扰。

11.8 网桥模式中的带宽管理

此前，我们都是网关位置执行带宽管理操作，但如果想要在一个现有的并且路由完整的网络上实施带宽管理，岂不是要对网络架构进行一番大改动？其实这与第9章所讨论的透明防火墙有异曲同工之妙，如果我們可以在网桥上实施带宽管理机制，那就无需再为路由问题而烦恼。

以图11-11为例来说明网桥模式中的带宽管理该如何进行。

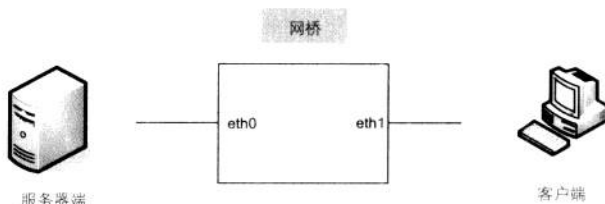


图11-11 网桥模式中的带宽管理

我们使用如下两个配置文件及两个脚本来完成这项任务：

- /etc/sysconfig/network-scripts/ifcfg-eth0:

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
```

- /etc/sysconfig/network-scripts/ifcfg-eth1:

```
DEVICE=eth1
ONBOOT=yes
BOOTPROTO=static
```

- /root/bridge.sh:

```
#!/bin/bash

## Delete Bridge br0
ifconfig br0 down
brctl del br br0
```



```
## Add Bridge br0
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 eth1
ifconfig br0 up
```

● /root/qos.sh:

```
#!/bin/bash

iptables -F

tc qdisc del dev eth1 root
tc qdisc add dev eth1 root handle 10:0 htb

tc class add dev eth1 parent 10:0 classid 10:1 htb rate 256kbps
tc class add dev eth1 parent 10:1 classid 10:10 htb rate 60kbps

tc filter add dev eth1 parent 10:0 protocol ip handle 80 fw classid 10:10

iptables -A FORWARD -p tcp --sport 80 -j MARK --set-mark 80
```

首先设置好eth0及eth1的网络配置，再使用/root/bridge.sh将系统的网络机制设置为网桥，最后再使用/root/qos.sh将带宽管理的机制执行起来。由于要对客户端执行带宽管理操作，因此在脚本中对eth1的输出流量进行管理。

11.9 多接口的带宽管理

在Linux系统中，队列会跟随着某一个设备。如我们之前使用的命令tc qdisc add dev eth1 root handle 10:0 htb，意思就是要将eth1设备的队列算法设置为htb，所以这个队列也就跟随着eth1这个设备。当然它的所有分类也会绑定到这个队列上，如果带宽管理主机上同时存在多个网卡，就会出现这个问题。

如图11-12所示，假设eth0连接因特网、eth1连接DMZ、eth2连接内部网络。在这个结构中，如果要限制内部及DMZ主机发送数据到因特网所能使用的带宽，基本上是很容易做到的，只需在eth0上设置htb队列算法即可。但如果要限制内部网络及DMZ网段从因特网下载数据所能使用的带宽，这就无法做到了。原因在于这个操作牵扯到两个网络接口eth1及eth2，而每个接口的队列又是各自独立管理，最多只能把带宽以手动方式分成两份，并分别设置到两个不同

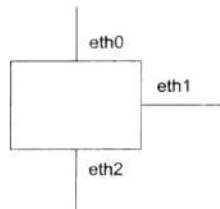


图11-12 多接口主机的带宽管理

的队列上。例如，因特网的下载带宽是10MB/s，可以先将这10MB/s分成7MB/s及3MB/s，接着分别在eth1及eth2的队列上设置其因特网下载数据的最大值，分别是7MB/s及3MB/s，但这样的设置将会丧失借用带宽的能力。

为了解决这个问题，<http://linuximq.net>组织设计了一个imq(Intermediate Queueing Device)机制。启动imq机制后，就会在系统上增加一个新的虚拟接口，而这个虚拟接口可以同时连接多个不同的实体接口。以图11-13为例，可以在要进行带宽管理的设备前增加一个虚拟接口，以便将imq0作为输出接口，并在imq0接口上执行带宽管理操作。另外值得一提的是，虽然多了一个虚拟接口，但并不改变原来数据包的路由路径，比如，系统原要将某一个数据包从eth1接口送出，虽然这个数据包被送入imq0接口，最后imq0处理完毕后，该数据包还是一样由eth1送出。

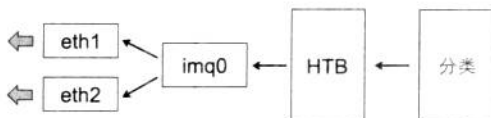


图11-13 imq接口结构

11.9.1 为内核及iptables安装补丁

可惜的是，imq直到kernel 2.6.35都还没有被收入到内核之中。因此，如果你想使用imq机制，就得自己给内核安装补丁。在撰写本章时，所使用的程序源代码版本分别如表11-1所示：

表11-1 程序源代码版本版本

程序源代码文件名	来源位置
linux-2.6.35.10	http://www.kernel.org/pub/linux/kernel/v2.6/longterm/v2.6.35/
linux-2.6.35-imq-test1.diff	http://linuximq.net/patches/linux-2.6.35-imq-test1.diff
iptables-1.4.10	http://www.netfilter.org/news.html#2010-08-06
iptables-1.4.6-imq.diff	http://linuximq.net/patches.html

关于内核及iptables的补丁安装及编译过程在此就不再详细说明了。如有不清楚之处，可以自己参阅第7章，这里仅就重要部分进行说明。

1. 内核编译

在开始真正编译内核之前，我们需要安装imq的补丁，可以使用patch -p1 <../linux-2.6.35-imq-test1.diff命令来完成。完成上述操作后，请在make menuconfig菜单中选择IMQ(intermediate queueing device)support(NEW)及IMQ target support(NEW)两个选项，其选择路径分别如下：

- IMQ(intermediate queueing device)support(NEW): Device Drivers | Network device support | IMQ(intermediate queueing device)support(NEW) ”。

- IMQ target support(NEW): Networking support | Networking options | Network packet filtering framework(Netfilter) | Core Netfilter Configuration | IMQ target support(NEW) ”。

2. iptables的编译

除了内核需要安装补丁外，iptables这个工具也需要安装imq的补丁，但imq patch文件在路径的安排上似乎不合常情，因此，当你在执行`patch -p1 <./iptables-1.4.6-imq.diff`命令之前，请将工作目录转换到`iptables-1.4.9.1/extensions`目录之下。否则，安装补丁进来的文件会放错位置。编译完毕后，我们应该可以在`iptables-1.4.9.1/extensions`目录下看到已经编译好的`libxt_IMQ.so`文件。

11.9.2 多接口带宽管理

完成内核及iptables的补丁安装及编译之后，接下来讨论如何使用IMQ机制。以图11-14为例，假设eth0及eth1分别是以1MB/s的带宽连接因特网，而eth2则是连接内部网络的接口，我们希望限制内部网络传输数据到因特网的带宽。由于对因特网有两个接口，因此，必须借助IMQ的功能才能达到这样的要求。

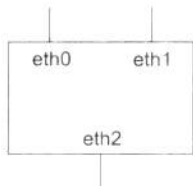


图11-14 imq接口使用示例

首先使用`modprobe imq`命令来加载imq模块，加载模块后就可以使用`ip link show`或`ifconfig -a`命令来查看配置好的imq接口。这时应该可以看到imq0到imq15一共16个接口。如果不需要用到这么多接口，可以使用`modprobe imq numdevs=1`命令来加载imq模块，其中`numdevs=1`参数用来设置所需的接口数量。如果需要把已经加载进来的imq模块卸载，可使用`modprobe -r imq`命令来执行卸载操作。但请注意，imq模块只有在没有被使用的情况下才可以被卸载。

在加载imq模块后，一个头疼的问题出现了。如图11-15所示，我们该如何去设置imq0接口是连接eth0及eth1接口呢？其实imq模块的用法确实有点特殊，imq模块必须与

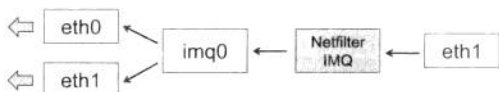


图11-15 imq机制的逻辑结构

Netfilter下的IMQ模块一起使用。有一点要特别注意，数据包默认不会进入imq0接口，而是直接从eth0或eth1接口送出。如果是这样，imq0接口根本就没有存在的价值。因此，我们可以通过Netfilter的IMQ模块，将特定数据包导向imq0接口。数据包被导入imq0接口之后，会从原来预定送出的接口送出。例如，有一个数据包原来的路由决定这个数据包要从eth1接口送出，但这个数据包被Netfilter的IMQ模块导入到imq0接口，那么这个数据包在离开imq0接口之后，依然会从eth1接口发送出去。

在下面的第一个示例中, 凡从eth2接口送入的数据包, 都用IMQ模块将其导入到imq0接口, 其中--todev 0用于设置要导入的接口名称。当然我们也未必要将所有数据包都导入到imq0接口。下面的第二个示例设置送往TCP端口80的数据包要导入到imq0接口。

```
iptables -t mangle -A PREROUTING -i eth2 -j IMQ --todev 0
```

```
iptables -t mangle -A PREROUTING -p tcp --dport 80 -i eth2 -j IMQ --todev 0
```

完成上面的操作后, 接下来就是带宽管理了。我们可以使用以下命令来完成带宽管理的操作。在第❶行命令中, 我们改变了imq0接口的队列算法, 因为有imq接口才能将原来的两个队列变成一个; 第❷行则设置imq0上的最大带宽是2M bps; 第❸行命令则建立四个类别, 并设置各类别的保证带宽为512KB/s, 最大使用带宽则是2MB/s; 第❹行命令则分别将主机192.168.0.1到192.168.0.4的数据包标记MARK值为1~4; 第❺行命令则将不同MARK值的数据包导入到不同类别中, 如此就可以将对外的2MB/s的带宽分给四台计算机来使用, 并设置保证带宽及最大带宽分别是512KB/s及2MB/s。

```
tc qdisc add dev imq0 root handle 10:0 htb ❶
tc class add dev imq0 parent 10:0 classid 10:1 htb rate 256kbps ❷
tc class add dev imq0 parent 10:1 classid 10:2 htb rate 64kbps2 ceil 256kbps
tc class add dev imq0 parent 10:1 classid 10:3 htb rate 64kbps2 ceil 256kbps
tc class add dev imq0 parent 10:1 classid 10:4 htb rate 64kbps2 ceil 256kbps
tc class add dev imq0 parent 10:1 classid 10:5 htb rate 64kbps2 ceil 256kbps ❸
tc filter add dev imq0 parent 10:0 protocol ip handle 1 fw classid 10:2
tc filter add dev imq0 parent 10:0 protocol ip handle 2 fw classid 10:3
tc filter add dev imq0 parent 10:0 protocol ip handle 3 fw classid 10:4
tc filter add dev imq0 parent 10:0 protocol ip handle 4 fw classid 10:5 ❹
iptables -A FORWARD -i eth2 -s 192.168.0.1 -j MARK --set-mark 1
iptables -A FORWARD -i eth2 -s 192.168.0.2 -j MARK --set-mark 2
iptables -A FORWARD -i eth2 -s 192.168.0.3 -j MARK --set-mark 3
iptables -A FORWARD -i eth2 -s 192.168.0.4 -j MARK --set-mark 4 ❺
```

最后, 将前面所有命令整理成以下的Shell脚本:

```
#!/bin/bash

## 清除所有filter及mangle的规则
iptables -t filter -F
iptables -t mangle -F

## 停用所有imq接口
for X in $(ip link show | grep 'imq.*UP' | awk -F : '{print $2}'); do
    ip link set $X down
done

## 加载imq模块
```

```

modprobe -r imq

## 设置imq0队列及分类并启动imq0接口
modprobe imq numdevs=1
ip link set imq0 up

## 设置imq0队列及分类
tc qdisc add dev imq0 root handle 10:0 htb

tc class add dev imq0 parent 10:0 classid 10:1 htb rate 256kbps

tc class add dev imq0 parent 10:1 classid 10:2 htb rate 64kbps ceil 256kbps
tc class add dev imq0 parent 10:1 classid 10:3 htb rate 64kbps ceil 256kbps
tc class add dev imq0 parent 10:1 classid 10:4 htb rate 64kbps ceil 256kbps
tc class add dev imq0 parent 10:1 classid 10:5 htb rate 64kbps ceil 256kbps

tc filter add dev imq0 parent 10:0 protocol ip handle 1 fw classid 10:2
tc filter add dev imq0 parent 10:0 protocol ip handle 2 fw classid 10:3
tc filter add dev imq0 parent 10:0 protocol ip handle 3 fw classid 10:4
tc filter add dev imq0 parent 10:0 protocol ip handle 4 fw classid 10:5

## 数据包以fw mark分类
iptables -A FORWARD -i eth2 -s 192.168.0.1 -j MARK --set-mark 1
iptables -A FORWARD -i eth2 -s 192.168.0.2 -j MARK --set-mark 2
iptables -A FORWARD -i eth2 -s 192.168.0.3 -j MARK --set-mark 3
iptables -A FORWARD -i eth2 -s 192.168.0.3 -j MARK --set-mark 4

## 把从eth2接口送入的数据包导入imq0接口
iptables -t mangle -A PREROUTING -i eth2 -j IMQ --to-dev 0

```

11.10 实际案例

假设有一家公司的网络架构如图11-16所示，这家公司使用的是10M/3M的光纤网络。因为经常出现网络速度变慢的问题，所以公司决定引入带宽管理机制。由于内部有对外提供服务的主机，且对外带宽在输出部分也只有3MB/s，因此，我们必须谨慎安排这3MB/s带宽的使用。

评估之后发现，公司对外最为重要的服务是Web服务、Mail服务、DNS服务，最后才是内部员工上网的需求。因此，我们把Web服务器的保证带宽设置为1600KB/s，Mail服务器的保证带宽设置为384KB/s，DNS服

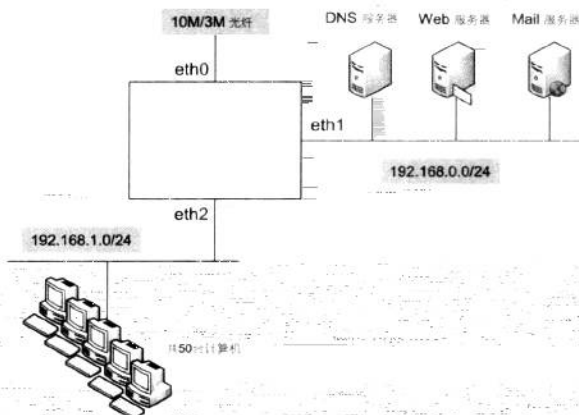


图11-16 带宽管理部部署示例

务器的保证带宽设置为64KB/s，内部网络访问因特网的保证带宽则设置为1024KB/s。另外为了避免某些服务的非高峰时间浪费带宽，为以上服务分别制定了带宽借用的使用量，如表11-2的“可使用的最大带宽”，但请注意，“可使用的最大带宽”实际是“保证带宽”+“可借用带宽”。

表11-2 企业对外带宽的分配计划表

来源名称	保证带宽	可使用的最大带宽
Web Server	1600KB/s	2048KB/s
Mail Server	384KB/s	1536KB/s
DNS Server	64KB/s	64KB/s
192.168.1.0/24	1024KB/s	1536KB/s
192.168.1.0/24每台计算机	20KB/s	64KB/s

接下来讨论eth1及eth2接口的输出流量，也就是定义从因特网传输到企业内部的数据包所占用的带宽。不过，这个方向的网络流量需要分成两个部分来讨论。

- 流向DMZ所占用的带宽：什么样的数据会从因特网送入到DMZ网段呢？这里分成两种数据来讨论，第一种是DMZ内的主机连上网络后下载文件所产生的数据流，如DMZ内的主机在进行系统更新；第二种则是从因特网上“主动”送到DMZ内的数据流，如客户端在进行DNS查询、浏览网页或有电子邮件要送到邮件服务器等。
- 流向内部网络所占用的带宽：会有数据流从因特网送入到内部网络，这通常都是内部网络使用者在访问因特网上的服务所产生的，如文件的下载、浏览网页和网络视频等。

综合以上分析，我们大概可以了解哪些流量较为重要，哪些不太重要，并分别为DNS服务器、Web服务器、Mail服务器设置了保证带宽。内部网络部分也有保证带宽的限制，另外更进一步地为内部网络的每一台主机都设置了保证带宽，以免使用者大量消耗带宽而导致其他人上网拥挤的问题。表11-3列出了企业下载带宽计划表。

表11-3 企业下载带宽计划表

目的名称	保证带宽	可使用的最大带宽
DNS Server	128KB/s	1024KB/s
Web Server	256KB/s	2048KB/s
Mail Server	3072KB/s	5120KB/s
192.168.1.0/24	6784KB/s	8192KB/s
192.168.1.0/24每台计算机	135KB/s	4096KB/s

定义了带宽的使用范围后，接下来就可以用类别结构图来表示，如图11-17及图11-18所示。

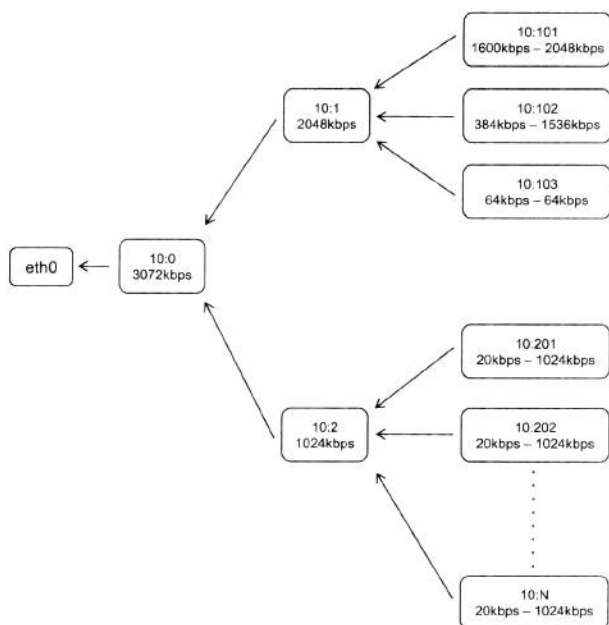


图11-17 eth0类别结构图

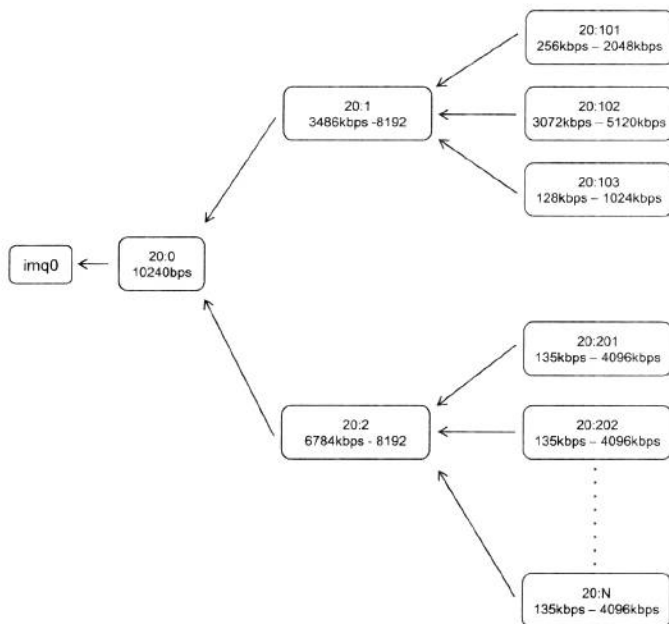


图11-18 imq0类别结构图

有了类别结构图之后,便可以使用脚本来实现这个结构。我们使用eth0.sh及imq0.sh两个脚本分别描述输出及输入的带宽限制:

● eth0.sh:

```
#!/bin/bash

DNS_SERVER=192.168.0.1          # FW Mark 53
MAIL_SERVER=192.168.0.2        # FW Mark 25
WEB_SERVER=192.168.0.3         # FW Mark 80
#=====<< 设置类别 >>=====
tc qdisc del dev eth0 root
tc qdisc add dev eth0 root handle 10:0 htb
tc class add dev eth0 parent 10:0 classid 10:1 htb rate 3072kbps
tc class add dev eth0 parent 10:1 classid 10:10 htb rate 2048kbps
tc class add dev eth0 parent 10:1 classid 10:20 htb rate 1024kbps

## 设置DMZ服务器的类别
tc class add dev eth0 parent 10:10 classid 10:101 htb \
    rate 1600kbps ceil 2048kbps      # Web Server
tc class add dev eth0 parent 10:10 classid 10:102 htb \
    rate 384kbps ceil 1536kbps      # Mail Server
tc class add dev eth0 parent 10:10 classid 10:103 htb \
    rate 64kbps ceil 64bps          # DNS Server

## 设置内部网络使用者的类别,其中bc命令为四则运算的工具
for (( I=1 ; I<=50 ; I++ ))
do
    tc class add dev eth0 parent 10:20 classid 10:${ echo 200 + $I | bc
} htb rate 20kbps ceil 1024kbps
done
#=====<< 设定选择器 >>=====
## 设置DMZ服务器的选择器
tc filter add dev eth0 parent 10:0 protocol ip \
    handle 53 fw classid 10:103      # DNS Server
tc filter add dev eth0 parent 10:0 protocol ip \
    handle 25 fw classid 10:102      # Mail Server
tc filter add dev eth0 parent 10:0 protocol ip \
    handle 80 fw classid 10:101      # Web Server

## 设置内部网络使用者的选择器
for (( IP=1 ; IP<=50 ; IP++ ))
do
    tc filter add dev eth0 parent 10:0 protocol ip handle $( \
        echo 200 + $IP | bc ) fw classid 10:${ echo 200 + $IP | bc }
done
#=====<< 使用Netfilter来划分数据包 >>=====
## 设置DMZ服务器数据包的Mark值
```

```
iptables -t mangle -A FORWARD -o eth0 -s $DNS_SERVER -j MARK --set-mark 53
iptables -t mangle -A FORWARD -o eth0 -s $MAIL_SERVER -j MARK --set-mark 25
iptables -t mangle -A FORWARD -o eth0 -s $WEB_SERVER -j MARK --set-mark 80
```

设置内部网络使用者数据包的Mark值

```
for (( IP=1 ; IP<=50 ; IP++ ))
do
    iptables -t mangle -A FORWARD -o eth0 -s 192.168.1.$IP -j MARK \
    --set-mark $( echo $IP + 200 | bc )
done
```

● imq0.sh:

```
#!/bin/bash
DNS_SERVER=192.168.0.1          # FW Mark 53
MAIL_SERVER=192.168.0.2        # FW Mark 25
WEB_SERVER=192.168.0.3         # FW Mark 80

#=====<< 设置IMQ接口 >>=====
modprobe imq numdevs=1
iptables -t mangle -A PREROUTING -i eth0 -j IMQ --todev 0
ip link set imq0 up
#=====<< 设置类别 >>=====
tc qdisc del dev imq0 root
tc qdisc add dev imq0 root handle 20:0 htb
tc class add dev imq0 parent 20:0 classid 20:1 htb rate 10240kbps
tc class add dev imq0 parent 20:1 classid 20:10 htb \
    rate 3486kbps ceil 8192kbps
tc class add dev imq0 parent 20:1 classid 20:20 htb \
    rate 6784kbps ceil 8192kbps

## 设置DMZ服务器的类别
tc class add dev imq0 parent 20:10 classid 20:101 htb \
    rate 256kbps ceil 2048kbps # Web Server
tc class add dev imq0 parent 20:10 classid 20:102 htb \
    rate 3072kbps ceil 5120kbps # Mail Server
tc class add dev imq0 parent 20:10 classid 20:103 htb \
    rate 128kbps ceil 1024bps # DNS Server

## 设置内部网络使用者的类别
for (( I=1 ; I<=50 ; I++ ))
do
    tc class add dev imq0 parent 20:20 classid 20:$I \
    echo 200 + $I | bc | htb \
    rate 135kbps ceil 4096kbps
done
#=====<< 设置选择器 >>=====
tc filter add dev imq0 parent 20:0 protocol ip \
    handle 53 fw classid 20:103 # DNS Server
tc filter add dev imq0 parent 20:0 protocol ip \
```

```

                                handle 25 fw classid 20:102          # Mail Server
tc filter add dev imq0 parent 20:0 protocol ip \
                                handle 80 fw classid 20:101        # Web Server

## 设置内部网络使用者的选择器
for (( IP=1 ; IP<=50 ; IP++ ))
do
    tc filter add dev imq0 parent 20:0 protocol ip handle $( \
        echo 200 + $IP | bc ) fw \
        classid 20:$( echo 200 + $IP | bc )
done
#=====<< 使用Netfilter来划分数据包 >>=====
## 设置DMZ服务器数据包的Mark值
iptables -A FORWARD -i eth0 -d $DNS_SERVER -j MARK --set-mark 53
iptables -A FORWARD -i eth0 -d $MAIL_SERVER -j MARK --set-mark 25
iptables -A FORWARD -i eth0 -d $WEB_SERVER -j MARK --set-mark 80

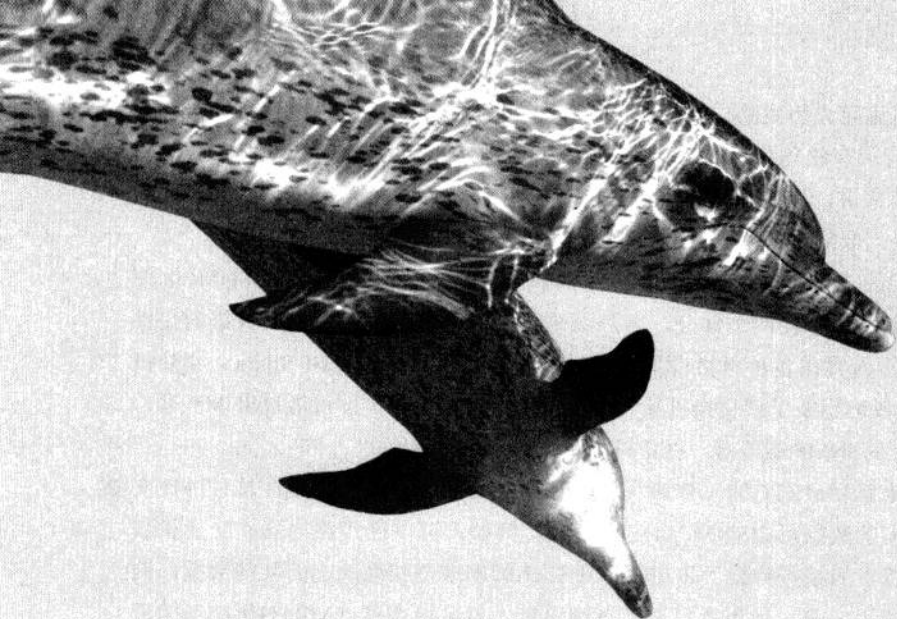
## 设置内部网络使用者数据包的Mark值
for (( IP=1 ; IP<=50 ; IP++ ))
do
    iptables -A FORWARD -i eth0 -d 192.168.1.$IP -j MARK \
        --set-mark $( echo $IP + 200 | bc )
done

```

以上仅是一个简单示例，并没有考虑Netfilter性能问题，也没有考虑复杂通信协议的带宽限制问题。在实际中实施带宽管理机制时，应该认真考虑这些细节。

11.11 小结

如果你所管理的网络存在使用者经常滥用网络带宽的情况，那么本章的内容将帮你快速解决这些令人头疼的问题。特别是文中所提到在网桥上实施带宽管理的技术，更是网络管理人员管理带宽的一大利器。另外告诉你市场上很多带宽管理器，其实内部都是Linux的。



Linux

|第12章| 流量统计

12

提到流量统计,大家首先想到的大概就是MRTG这个软件吧!MRTG是一套简单但功能强大的工具,非常适合用来将单纯的数据转换成图形,例如,网络流量、CPU使用率、硬盘使用率等数据。简单来讲,只要可以持续提供一个数值,MRTG就可以帮你将其转换成图形。不过,如果想轻松驾驭MRTG,那么Shell脚本是你的必修科目。

在开始介绍MRTG的工作原理之前,我们先来认识一下什么是SNMP协议。SNMP的使用过程可以分为客户端和服务端两个部分,客户端是数据的查询者,服务器是数据的提供者。SNMP服务器通常内置在各种网络设备中,如路由器、防火墙、Mail Spam、交换机(Switch)等设备上。当我们想要了解交换机某一个端口的网络流量时,即可通过SNMP客户端工具来连接交换机上的SNMP服务器,以获得所要查询的相关信息。

我们使用图12-1来解释MRTG的工作流程,图中MRTG这个程序会定期生成HTML文件,而MRTG绘图所需的数据可由SNMP协议来获取。例如:可以连接路由器、交换机或计算机主机来取得系统上的某些数据;也可以通过Shell脚本来捕获系统上的某些数据,例如CPU使用率、硬盘使用率等。使用者只要通过浏览器,就可以观察到MRTG所绘制的流量图。

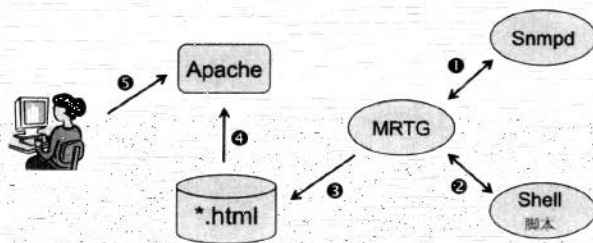


图12-1 MRTG结构

理解了上述概念后,接下来就使用MRTG来实际绘制Linux主机的网络流量图。

12.1 安装及测试SNMP服务器

12.1.1 安装SNMP服务器

首先要知道,一般操作系统自身并不会直接提供SNMP服务器,通常都是需要额外在主机上安装SNMP服务器,当然Linux系统也不例外。如果你使用CentOS或Fedora的系统(在撰写本书时,RedHat Enterprise Linux最新版本是6.0、CentOS最新版本是5.5、Fedora最新版本

是14)，在已经连接因特网的情况下，只需要运行`yum install net-snmp`及`yum install-snmp-utils`即可。如果你的系统是RedHat Enterprise Linux，那可能需要从光盘中找到这两个软件及其他所依赖的软件，并且将其安装起来；你也可以购买RedHat的RedHat Network service，即可用yum通过网络来进行安装。

12.1.2 测试SNMP服务器

安装完SNMP服务器及客户端后，接下来就是要设置SNMP服务器使其能正常工作。出于安全方面的考虑，在默认情况下，SNMP服务器不提供外部查询功能，MRTG自然也就无法从SNMP服务器取得任何数据。请你在`/etc/snmp/snmpd.conf`文件的最后一行添加“`rocommunity mrtg 127.0.0.0/8`”，其中的参数含义如下。

1. 参数rocommunity

在SNMP协议中，SNMP客户端除了可以用来读取SNMP服务器上的数据之外，SNMP客户端也可以通过SNMP服务器来设置路由器、交换机等设备。因此，我们可以通过`rocommunity`来指定哪些客户端可以来“读取”SNMP服务器上的数据，`rwcommunity`则用来指定哪些客户端可以通过SNMP服务器来设置设备自身，如交换机。

在MRTG的应用上，只需对SNMP服务器进行数据的读取即可，因此使用`rocommunity`参数。

2. 参数mrtg

`mrtg`这个参数用来设置SNMP客户端连接SNMP服务器时的帐户名，不过，在此并不需要设置密码。

3. 参数127.0.0.0/8

这个参数是用来设置可以访问服务器的IP范围。如果省略不设置，默认是127.0.0.1。此外，如果有多个网段需要访问SNMP服务器，则可以在配置文件中加入多行`rocommunity`参数，然后分别指向每个网段，如下：

```
rocommunity mrtg 127.0.0.0/8
rocommunity mrtg 192.168.0.0/24
```

在完成以上设置后，别忘了运行`service snmpd restart`，以重新启动SNMP服务器，接下来再测试SNMP服务器是否正常工作。在此提醒你，SNMP协议分为Version1、2、3三个版本，如果你的网络接口是Gigabit，那么请使用v2的协议与SNMP服务器通信。这是由于`snmpd`在v1协议下，其counter只有32位，因此，在Gigabit的网络接口中有可能造成counter溢

出, 而导致流量值不准确。在此可使用以下命令进行测试:

```
snmpwalk -v 2c 127.0.0.1 -c mrtg
```

如果这时显示屏幕上出现一长串信息(多达好几个屏幕)——就代表SNMP客户端可以正常地从SNMP服务器读取到数据。其中snmpwalk就是SNMP客户端的工具, -v 2c则是设置使用SNMP v2的版本与SNMP服务器进行通信, 127.0.0.1是SNMP服务器所在的IP, -c mrtg是登录SNMP服务器的帐户名。

12.2 安装及设置MRTG

12.2.1 安装MRTG

安装MRTG的方法与安装SNMP服务器的方法相同。可以使用yum install mrtg命令, 通过网络来安装MRTG软件。

12.2.2 设置MRTG

MRTG软件安装完毕后, 大概有三个地方需要特别注意:

- /etc/httpd/conf.d/mrtg.conf

这个文件其实是Apache Web服务器的配置文件。有了这个文件, 在设置完mrtg之后, 我们只需在浏览器上输入http://x.x.x.x/mrty/xxx.html即可看到网络流量图。不过, 如果直接使用默认值, 那么你就只能在安装MRTG的主机上以http://127.0.0.1/mrtg/xxx.html才能看到网页, 因为默认值只允许localhost来访问这个网页。

```
<Location /mrtg>
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1
    Allow from ::1
    # Allow from .example.com
</Location>
```

为了解除上面的这个限制, 请将# Allow from .example.com改为Allow from 192.168.0.0/24, 并重新启动Apache Web服务器, 以便让192.168.0.0/24网段的主机正常访问MRTG的网页。

- /etc/mrtg

MRTG可以同时绘制多个设备的流量图。一般情况下，建议你为每个设备编写专门的配置文件，而这些配置文件都可以放在这个目录下。

- /var/www/mrtg

这个目录中存放为MRTG制作好的网页。

12.2.3 使用cgmaker工具编写MRTG针对网卡的配置文件

了解了以上三部分后，接下来要编写MRTG针对网卡的配置文件。这个步骤可以使用MRTG自带的工具cgmaker来完成，只要事后再加以修改即可，使用方式如下：

```
snmpwalk -v 2c 127.0.0.1 -c mrtg > /etc/mrtg/net.cfg
```

生成的文件/etc/mrtg/net.cfg就是MRTG检测网卡的配置文件，也就是说，MRTG会根据这个文件的内容来生成HTML文件。但默认的内容一定要经过调整，否则是不符合我们的需求的。首先分析图12-2这个配置文件的结构，在这个配置文件中，以#字符开头的行表示这一行是被注释掉了的；此外，文件的开头是全局参数的定义区，其内容包含如生成好的网页放在哪个目录下，生成的网页语种等，反正就是与网卡本身的设置参数无关的内容。接着是网卡的设置值部分，这里，每张网卡的设置值都各自独立；当然，也可以将图12-2的内容拆成三个文件来保存。但请特别注意，这三个文件的开头都要有自己独立的全局参数设置区。

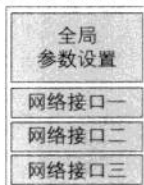


图12-2 MRTG配置文件结构

```
### Global Config Options
# for UNIX
# WorkDir: /home/http/mrtg
# or for NT
# WorkDir: c:\mrtgdata
### Global Defaults
# to get bits instead of bytes and graphs growing to the right
# Options[_]: growright, bits
EnableIPv6: no
```

默认的全局区域参数内容如上所示，下面整理了全局区域内的常用参数。

- WorkDir

设置MRTG所生成的网页要保存在哪个目录中。在本示例中，请将其设置为/var/www/mrtg。

● Language: big5

设置所生成网页是中文,但有一点请你特别注意,由于MRTG所生成的网页是big5编码,但Apache Web服务器的默认值却会告诉浏览器其网页是UTF-8编码,所以使用者看到这个网页将是乱码。为了解决这个问题,将/etc/httpd/conf/httpd.conf文件中的AddDefaultCharset UTF-8给注释掉,再重新启动Apache Web服务即可。

● Options[_]: growright, nopercent, bits

Options后面常用的参数有growright、nopercent、bits,这三个参数的含义如下:

- growright/growleft: 用来设置MRTG获取的最新数据是要绘制在图的最左边还是最右边。一般的习惯是设置为growright。也就是说,图的最左边是较旧的数据,最右边是较新的数据。在图12-3的流量图中,最新测得的流量数据是从图的最右边开始出现的。
- nopercent: 在流量的统计值部分不要显示%符号。如果省略这个参数,默认会显示%符号,如图12-3⑥所示的位置。
- bits: 设置MRTG是以位为显示单位。如果省略这个参数,默认会将字节作为显示单位。

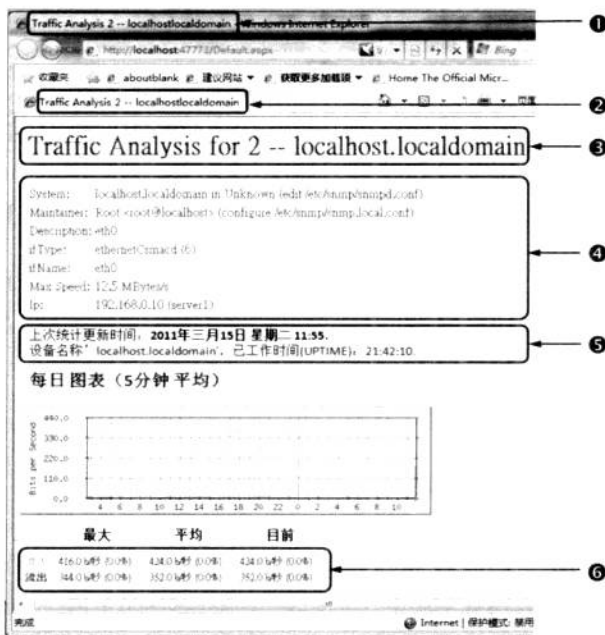


图12-3 MRTG网页说明

设置了全局参数之后,接下来设置网络接口的参数。下面整理了网络接口的参数。

● Target[localhost_2]: 2:mrtg@localhost:::2

Target用于设置“设备名称”，例如，设备可能是一台路由器或交换机。在我们的应用中，cfgmaker会固定设置为“计算机名称_第几个网卡”。由于在我的计算机上，这个网卡是在第二顺位被检测到的，故取名为localhost_2。不过，我通常都会手动更改这个名称，如更改为eth0之类的，因为这个名称关系到我们浏览MRTG的URL名称，如http://192.168.0.10/mrtg/localhost_2.html，其中的localhost_2.html文件就是由这个名称来决定的。

这个参数后面带的mrtg@localhost:::2参数则是指MRTG会使用mrtg这个帐户来登录localhost这台主机，2代表使用SNMP v2的协议。

● SetEnv[localhost_2]: MRTG_INT_IP="192.168.0.10" MRTG_INT_DESCR="eth0"

定义好localhost_2这个设备名称后，MRTG就可以通过SNMP通信协议来获取指定设备上的数据，但在这么多的数据中，哪个数据才会是我们所需要的呢？我们可以通过这个参数来告诉MRTG所要获取的是localhost_2这个设备上哪部分的数据。

● MaxBytes[localhost_2]:12500000

设置了以上两个参数后，MRTG已经可以获取到外部的数据值，而这个参数是设置所能接受这个值的上限。但请注意，如果是针对网络流量的话，这个值的单位是字节。

● Title[localhost_2]: Traffic Analysis for 2 -- localhost.localdomain

图12-3中第①及②的部分就是这个参数所指定的内容。建议将其改成你计算机主机的名称，如此会让人更容易理解。

● PageTop[localhost_2]: <h1>Traffic Analysis for 2 -- localhost.localdomain</h1>

图12-3的第③部分就是这个参数所设置的内容。也建议你将其改成你计算机主机的名称。

其他参数

图12-3的第④部分就是PageTop参数以下内容所生成的结果：

```
Target[localhost_2]: 2:mrtg@localhost:::2
SetEnv[localhost_2]: MRTG_INT_IP="192.168.0.10" MRTG_INT_DESCR="eth0"
MaxBytes[localhost_2]: 12500000
Title[localhost_2]: Traffic Analysis for 2 -- localhost.localdomain
PageTop[localhost_2]: <h1>Traffic Analysis for 2 -- localhost.localdomain</h1>
<div id="sysdetails">
  <table>
    <tr>
      <td>System:</td>
      <td>localhost.localdomain in Unknown (edit /etc/snmp/snmpd.
        conf)</td>
    </tr>
  </table>
</div>
```

```

        <td>Maintainer:</td>
        <td>Root &lt;root@localhost> (configure /etc/snmp/snmp.
            local.conf)</td>
    </tr>
    <tr>
        <td>Description:</td>
        <td>eth0 </td>
    </tr>
    <tr>
        <td>ifType:</td>
        <td>ethernetCsmacd (6)</td>
    </tr>
    <tr>
        <td>ifName:</td>
        <td>eth0</td>
    </tr>
    <tr>
        <td>Max Speed:</td>
        <td>12.5 MBytes/s</td>
    </tr>
    <tr>
        <td>Ip:</td>
        <td>192.168.0.10 (server1)</td>
    </tr>
    </table>
</div>

```

设置完/etc/mrtg/net.cfg文件之后，就可以使用MRTG并通过这个配置文件来生成初始化的HTML文件，在此可以使用env LANG=C /usr/bin/mrtg /etc/mrtg/net.cfg命令来完成。其中net.cfg就是告诉MRTG要使用的配置文件名称，env LANG=C则设置运行时的语言，而这个命令总共要运行三次才算完成初始化操作，如下所示：

```

[root@server1 mrtg]# env LANG=C /usr/bin/mrtg /etc/mrtg/net.cfg ❶
2011-03-15 14:33:36, Rateup WARNING: /usr/bin/rateup could not read the
primary log file for eth0
2011-03-15 14:33:36, Rateup WARNING: /usr/bin/rateup The backup log file for
eth0 was invalid as well
2011-03-15 14:33:36, Rateup WARNING: /usr/bin/rateup Can't remove eth0.old
updating log file
2011-03-15 14:33:36, Rateup WARNING: /usr/bin/rateup Can't rename eth0.log to
eth0.old updating log file
[root@server1 mrtg]#
[root@server1 mrtg]# env LANG=C /usr/bin/mrtg /etc/mrtg/net.cfg ❷
2011-03-15 14:33:38, Rateup WARNING: /usr/bin/rateup Can't remove eth0.old
updating log file
[root@server1 mrtg]#
[root@server1 mrtg]# env LANG=C /usr/bin/mrtg /etc/mrtg/net.cfg ❸
[root@server1 mrtg]#

```


如果在完成初始化后才发现没有正确编写配置文件，这时可以将/etc/mrtg/net.ok及/var/www/mrtg/eth0*的部分删除，然后再重新初始化即可。完成后，就可以在浏览器中输入http://192.168.0.10/mrtg/eth0.html，来浏览网络的使用流量图。

最后在/etc/crontab中设置MRTG每隔5分钟读取一次数据，并且重新绘制流量图，请将以下内容添加到/etc/crontab文件的最后一行，然后重新启动Crontd进程。

```
*/5 * * * * root /usr/bin/mrtg /etc/mrtg/net.cfg
```

12.3 另一种网络流量监测方式

经过以上示例解说之后，相信你应该可以顺利使用MRTG来获取网络的流量值，不过，我并不很喜欢搭配使用SNMP和MRTG，在防火墙应用上尤其如此。

12.3.1 结合使用Netfilter/Iptables和MRTG来监测网络流量

这里提供另一种更灵活的用法，那就是结合使用Netfilter/Iptables和MRTG。在前面的章节曾经介绍过iptables工具的一个参数-v。下面在❶的位置首先编写一条完全不会影响数据包传输路径的规则，并选用MARK方法。之所以会这么做，是因为有了这条规则之后，后面凡是从eth0接口送入的数据包，一定会匹配到这条规则。一段时间后，我们便可以使用第❷行iptables的语法将mangle Table中的PREROUTING列出，其中第一个字段是从eth0接口进入的数据包“数量”，而第二个字段就是从eth0接口送出的数据“总量”，但请注意其单位是“字节”。如有必要，这个值是可以清零的，我们可以使用❸的方法将mangle Table内的PREROUTING链第1行的Counter清零。

```
[root@server1]# iptables -t mangle -A PREROUTING -i eth0 -j MARK --set-mark 0
[root@server1]#
[root@server1]# iptables -t mangle -L PREROUTING -n -v ❶
Chain PREROUTING (policy ACCEPT 6262 packets, 6345K bytes)
 pkts bytes target prot opt in out source destination
 6262 6345K MARK all -- eth0 * 0.0.0.0/0 0.0.0.0/0 MARK and 0x0
[root@server1]#
[root@server1 mrtg]# iptables -t mangle -Z PREROUTING ❷
[root@server1 mrtg]#
[root@server1 mrtg]# iptables -t mangle -L PREROUTING -n -v ❸
Chain PREROUTING (policy ACCEPT 124 packets, 300K bytes) ❹
 pkts bytes target prot opt in out source destination
 0 0 MARK all -- eth0 * 0.0.0.0/0 0.0.0.0/0 MARK and 0x0
```

讲到这里,相信你已经了解要做什么了,我们可以编写一个Shell脚本并结合上面的机制来进行流量统计。首先在mangle表的PREROUTING及POSTROUTING链中加入以下两条规则,请确认是加在链的最前面,如下:

```
iptables -t mangle -I PREROUTING 1 -i eth0 -j MARK --set-mark 0
iptables -t mangle -I POSTROUTING 1 -o eth0 -j MARK --set-mark 0
```

之后,凡从eth0接口进入本机的数据包,一定会匹配到PREROUTING链的第一条规则;凡从eth0接口送出的数据包,一定会匹配到POSTROUTING链的第一条规则。因此,我们可以使用PREROUTING及POSTROUTING链的Counter来作为流量的统计依据。

前面曾提到过,MRTG除了可以通过SNMP协议来获取数据外,也可以通过“外部程序”来获取数据,这个外部程序你可以使用任何语言来编写。重点在于程序执行完之后要输出的四组数据,且这四组数据必须以独立的四行来呈现,如下:

```
12345
98765
AAAA
BBBBB
```

有了这四组数据之后,MRTG就可以将其以网页的方式来展现,使用下面的Shell脚本来实现这个程序。

我们先使用代码段❶来获取PREROUTING及POSTROUTING链的Counter值,并分别存入IN及OUT两个变量中。有一点请特别注意,当Netfilter的Counter数值大到一定程度之后,会自动以K或M来显示。因此,我们使用代码段❷❸将K转换成三个零,M转换成六个零。经过这样的处理之后所得到的值应该是字节值,另外,因为MRTG两次统计数据的最短时间是5分钟,而我们希望得到的结果是每秒多少字节(Bytes/S),因此,用代码段❹把IN及OUT的值除以300秒,最后得到Bytes/s。在此请你注意,如果是要检测网络流量,我们送给MRTG的值一定是以字节为单位,而不能是位。另外,❺❻分别是页面上的“设备名称”及“已工作时间”,如图12-3❺位置的信息。最后将整个Shell脚本保存为/root/mrtg/net.sh,但别忘了设置其权限为755。

```
#!/bin/bash

IN=$(iptables -t mangle -L PREROUTING -n -v | grep \
    ' MARK and 0x0' | awk '{print $2}') ❶
OUT=$(iptables -t mangle -L POSTROUTING -n -v | grep \
    ' MARK and 0x0' | awk '{print $2}')

IN=$(( echo $IN | sed 's/K/000/g' )) ❷
IN=$(( echo $IN | sed 's/M/000000/g' )) ❸
```

```

OUT=$( echo $OUT | sed 's/K/000/g' )
OUT=$( echo $OUT | sed 's/M/000000/g' )

echo "$OUT / 300 " | bc
echo " $IN / 300 " | bc

echo $(uptime | cut -d 'p' -f2 | cut -d , -f1,2)
echo "Linux Firewall"

```

12.3.2 手动编写MRTG的配置文件

接下来编写MRTG的配置文件。不过，这个配置文件就得自己动手编写了，因为cfmaker并不支持生成外部程序的MRTG配置文件，这里将其保存为/etc/mrtg/net.cfg，如下：

```

1. WorkDir: /var/www/mrtg
2. Language: big5
3. Target[eth0]: /root/mrtg/net-2.sh
4. MaxBytes[eth0]: 12500000
5. Options[eth0]: growright,nopercent,gauge
6. YLegend[eth0]: Bytes/s
7. ShortLegend[eth0]: Bytes/s
8. Legend0[eth0]: &nbsp;下行
9. Legend1[eth0]: &nbsp;上行
10. Title[eth0]: eth0 Traffic
11. PageTop[eth0]: <H1>eth0 Traffic</H1>
12. <TABLE>
13. <TR><TD>System:</TD><TD>Linux Firewall</TD></TR>
14. </TABLE>

```

文件说明如下：

- 第3行：这是我们编写的外部程序，但要特别注意外部程序前后的“这两个符号”，这两个符合并不是单引号。如图12-4所示，请看键盘左上角ESC键下面的那个按键，而这两个点就是由这个按键点出来的。



图12-4 键盘示意图

- 第5行：`gauge`参数，如果使用外部程序来获取数据，就需要加上这个参数；如果通过SNMP协议获取数据，则有可能不需要这个参数。不过，有时确实不太容易判断到底要不要加这个参数。最简单的办法就是先加上，若结果是无法绘制出图形，再把它去掉即可。

`nopercent`参数，如果没有加上这个参数，那么在MRTG所生成的网页上，在统计值的部分会出现%符号。如果加上这个参数，统计值的部分会自动带上第7行“ShortLegend[eth0]:”参数后面的字符串。

另外, 还可以带上另一个参数bits。如果没有加上这个参数, 默认流量是以Bytes/S来计算的, 否则是以Bits/S计算。

- 第6行: YLegend[eth0]:参数后面所带的参数会被内置到MRTG所生成的流量图的最左边。例如, 可以加上Bytes/S告诉浏览者, 流量图的单位是Bytes Per Second。
- 第8、9行: LegendO[eth0]及LegendI[eth0]分别获取外部程序所产生的第一个及第二个值, 而其后所带的字符串“上行”及“下行”则是统计值的提示字符串。

配置完所有配置文件后, 接下来就运行env LANG=C/usr/bin/mrtg/etc/mrtg/net.cfg命令来完成网页初始化(别忘了, 这个操作要执行三次), 最后在/etc/crontab中设置MRTG每隔5分钟获取一次数据。请将以下内容添加到/etc/crontab文件的最后一行, 然后重新启动Crontd进程。

```
* /5 * * * * root /usr/bin/mrtg /etc/mrtg/net.cfg
```

完成这个示例后, 请你务必记住两个重点。首先是外部程序编写的格式, 其次是如何在MRTG配置文件中获取外部程序所生成的数据。如果你真的了解, 就可以发挥想象力, 使用MRTG来解决问题。

12.4 外部程序及MRTG配置文件的示例

示例12.1 在Linux 防火墙上使用MRTG来绘制Web服务器所占用的带宽

● Netfilter设置:

```
iptables -t mangle -I FORWARD 1 -i eth1 -s $WEB_SRV -j ACCEPT
iptables -t mangle -I FORWARD 2 -i eth0 -d $WEB_SRV -j ACCEPT
```

● 外部程序:

```
#!/bin/bash

WEB_SRV=192.168.0.10

OUT=$(iptables -t mangle -L FORWARD -n -v | \
    grep ".* eth1 .* $WEB_SRV" | awk '{print $2}')
IN=$(iptables -t mangle -L FORWARD -n -v | \
    grep ".* eth0 .* $WEB_SRV" | awk '{print $2}')

echo " $OUT / 300 " | bc
echo " $IN / 300 " | bc

iptables -t mangle -Z FORWARD 1
iptables -t mangle -Z FORWARD 2

echo $(uptime | cut -d 'p' -f2 | cut -d , -f1,2)
echo "Linux Firewall"
```

● MRTG配置文件:

```
WorkDir: /var/www/mrtg
Language: big5

Target[web_traffic]: '/root/mrtg/web_traffic.sh'
MaxBytes[web_traffic]: 12500000
Options[web_traffic]: growright,nopercent,gauge
YLegend[web_traffic]: Bytes/s
ShortLegend[web_traffic]: Bytes/s
Legend0[web_traffic]: &nbsp;飞行
Legend1[web_traffic]: &nbsp;上行
Title[web_traffic]: Web Traffic
PageTop[web_traffic]: <H1> Web Server Traffic </H1>
<TABLE>
  <TR><TD>Web Server Traffic</TD></TR>
</TABLE>
```

示例12.2 使用MRTG绘制Web服务每秒的请求量

● Netfilter设置:

```
iptables -t mangle -I FORWARD 1 -i eth0 -p tcp --syn \
-d $WEB_SRV --dport 80 -j ACCEPT
iptables -t mangle -I FORWARD 2 -i eth0 -p tcp --syn \
-d $WEB_SRV --dport 443 -j ACCEPT
```

● 外部程序:

```
#!/bin/bash
HTTP=$(iptables -t mangle -L FORWARD -n -v | \
grep 'dpt:80' | awk '{print $1}')
HTTPS=$(iptables -t mangle -L FORWARD -n -v | \
grep 'dpt:443' | awk '{print $1}')

echo " SHTTP / 300 " | bc
echo " SHTTPS / 300 " | bc

iptables -t mangle -Z FORWARD 1
iptables -t mangle -Z FORWARD 2

echo $(uptime | cut -d 'p' -f2 | cut -d , -f1,2)
echo "Linux Firewall"
```

● MRTG配置文件:

```
WorkDir: /var/www/mrtg
Language: big5

Target[web_syn]: '/root/mrtg/web_syn.sh'
MaxBytes[web_syn]: 100000
Options[web_syn]: growright,nopercent,gauge
```

```
YLegend[web_syn]: Web Request
ShortLegend[web_syn]: 次/秒
LegendI[web_syn]: &nbsp;   HTTPS:
LegendO[web_syn]: &nbsp;   HTTP:
Title[web_syn]: HTTP / HTTPS Request
PageTop[web_syn]: <H1> HTTP / HTTPS Request </H1>
<TABLE>
  <TR><TD>HTTP / HTTPS Request </TD></TR>
</TABLE>
```

示例12.3 用MRTG绘制Linux防火墙主机的CPU使用率

● 外部程序:

```
#!/bin/bash

LANG=C

set $(sar -u 1 3 | grep 'Average' | awk '{print $3,$5}')
echo $1
echo $2
echo $(uptime | cut -d 'p' -f2 | cut -d , -f1,2)
echo "Linuf Firewall"
```

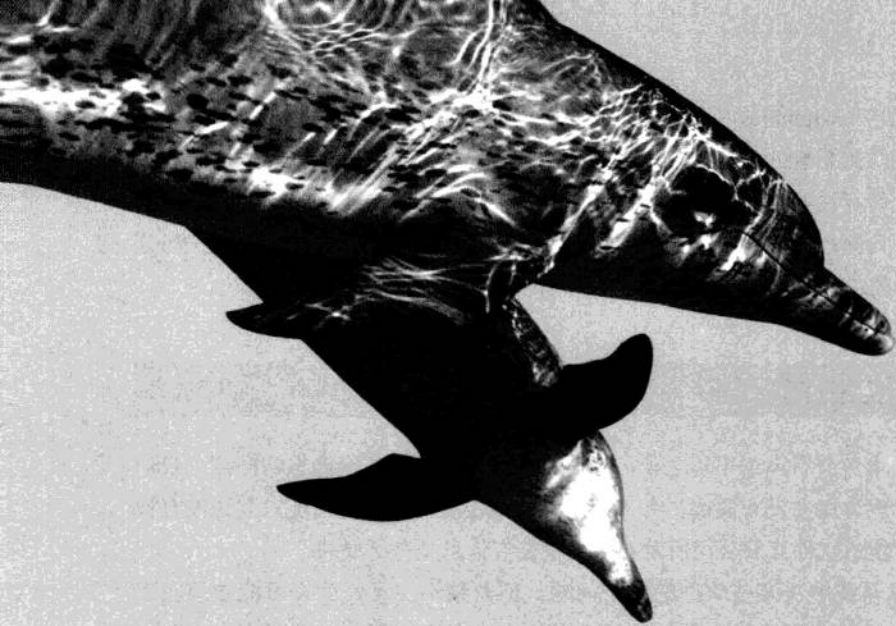
● MRTG配置文件:

```
WorkDir: /var/www/mrtg
Language: big5

Target[cpu]: `/root/mrtg/cpu.sh`
MaxBytes[cpu]: 100
Options[cpu]: nopercent, growright,gauge
YLegend[cpu]: CPU Usage (%)
ShortLegend[cpu]: %
LegendO[cpu]: &nbsp;   使用者负载;
LegendI[cpu]: &nbsp;   系统负载;
Title[cpu]: CPU Usage
PageTop[cpu]: <H1>CPU Usage</H1>
```

12.5 小结

网络数据的统计可以帮助我们了解很多事情。一提到数据统计,大多数人都会想到MRTG这个软件,只不过大多数使用者都会结合snmp来绘制网络流量图。在阅读完本章之后,相信你对于MRTG的应用一定会有更多想法,这是使用Linux的另一种乐趣。



Linux

|第13章| 弱点扫描、入侵检测及
主动防御系统

13



前面已经介绍了Netfilter/Iptables, 本章将介绍如何加强企业网络的安全性。将分为弱点扫描、入侵检测及主动防御系统三个部分来讨论这个主题。第一部分目的是及早发现已存在的安全漏洞, 并及时更新存在安全漏洞的服务程序; 第二部分则用于分析网络上传输中的数据包, 并根据数据包的传输行为及内容来判断网络上是否有非法行为正在进行。在第三部分中系统自动封锁入侵者的IP, 以便提供更安全的企业网络环境。

13.1 何谓弱点扫描

软件在开发过程中难免会有考虑不周之处, 这时可能会产生存在安全隐患的产品, 我们必须要有个概念: “软件现在没有漏洞, 并不代表它永远没有问题, 有可能只是还没有被发现而已”, 所以经常留意软件开发公司所发布的安全警告信息是有必要的。

虽然以上方法可以帮我解决某些软件的安全问题, 但若稍不注意, 就有可能导致应该更新的软件没有更新。由于我们平常会使用到的软件实在太多了, 因此, 想要完全获取你所使用的全部软件的安全信息, 也不是一件容易的事情。另一个很重要的概念就是不只是“计算机”才会有安全问题, 只要是需要设置IP的设备, 就有可能存在安全问题, 例如无线AP、IP分享器、打印机、交换机、路由器等, 可见安全漏洞可能随时出现在你的企业网络中。

为了能快捷高效地找出你企业内部已经存在的安全漏洞, 有些组织会“即时”收集网络系统中的安全漏洞, 快速制作出“漏洞检测软件”。我们就可以通过这些软件, 快速且方便地检测出企业网络中存在的已知安全问题, 这类的软件我们称为“弱点扫描”软件。

13.1.1 OpenVAS弱点扫描工具

这类弱点扫描工具网络上非常多, 但大部分都是商业版本, 而且价格非常昂贵, 并非一般中小企业的财力所能承担的。幸好在开源领域中也有不少弱点扫描软件可供选择, 其中以Nessus最为知名。Nessus在早期的版本是以GPL方式授权, 不过自从Nessus被TenableSecurity公司收购之后, Nessus宣布其Nessus 3.0(包括3.0)以后的版本不再以GPL方式授权, 目前Nessus分为商业授权版及个人授权版。至于其他更为详细的授权条约, 有兴趣的读者可以自己参考Nessus的官方网站。

不过还好有一些热心的开源程序设计师, 以Nessus 2.0为基础另外开发了一个名为OpenVAS的弱点扫描软件, 而且OpenVAS是完全免费的网络安全工具, 我们可以直接到OpenVAS的官方网站下载OpenVAS工具, 其官方网站是<http://www.openvas.org>。

13.1.2 OpenVAS弱点扫描工具的工作架构

如图13-1所示，OpenVAS系统分为服务器及客户端两部分，当我们要检测某台主机时，必须在OpenVAS客户端设置所要检测的项目等信息，接着通过OpenVAS客户端对OpenVAS服务器传达所要执行的操作。OpenVAS服务器接到命令之后，就会对目标主机进行检测，最后将检测结果交给OpenVAS客户端。

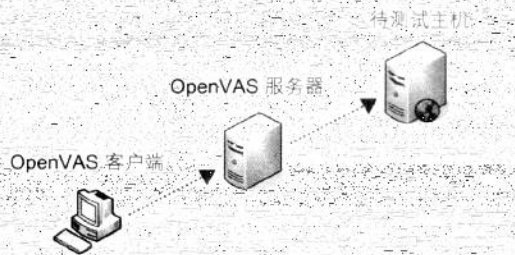


图13-1 OpenVAS工作架构图

13.1.3 下载及安装OpenVAS弱点扫描工具

1. 下载OpenVAS弱点扫描工具

可到官方网站直接下载OpenVAS软件，其网址是<http://www.openvas.org>，OpenVAS官方以人为本的态度令人肃然起敬。OpenVAS官方除了提供OpenVAS软件的下载之外，还提供了Virtual Machine的映像文件及Live CD的下载，因此使用者可以下载映像文件或Live CD来直接使用，甚至不必安装软件，如图13-2所示。

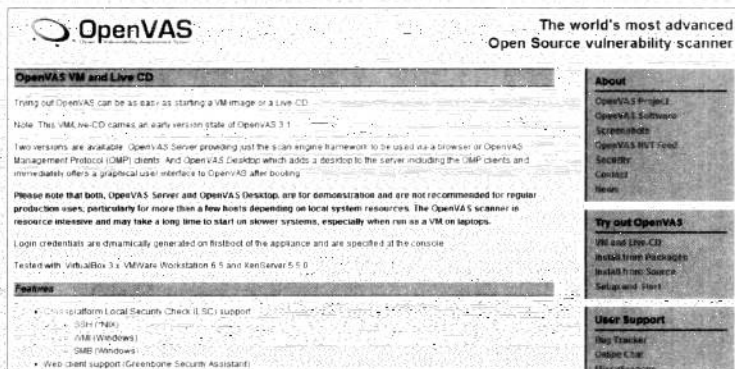


图13-2 OpenVAS下载网站

2. 安装OpenVAS软件

在撰写本章时，OpenVAS软件对于RHEL6/CentOS6的支持尚不完整。我根据官方的文件还无法正常地将OpenVAS安装在RHEL6系统上，因此以Fedora 14为平台来编写本章内容。值得一提的是，在FC 14中OpenVAS已经是一个内置的软件之一，我们只需在系统连接因特网

时, 执行以下命令即可将OpenVAS安装起来。

```
yum -y install openvas-client openvas-scanner
```

在安装完软件后, 最重要的事情就是安装OpenVAS的插件(Plugins)。那什么是插件呢? 这要从OpenVAS是如何检测目标主机是否存在弱点来谈起, 为了判断目标主机是否有弱点, OpenVAS首先写好用来入侵已知漏洞的程序, 如果目标主机可以被某个程序入侵, 那就代表目标主机有相对应的弱点, 只是这些用来检测弱点的程序不会进行破坏活动。不过, 由于安全漏洞会不断地被发现, 因此OpenVAS不可能写出一个万能的检测程序, 而其做法就是每个弱点写一个专用的程序, 而这些小程序就是插件, 最后再将这些程序放在官方网站上让大家来下载。请你务必定期更新这些插件, 手动安装方法如下:

```
openvas-nvt-sync
```

由于OpenVAS客户端与OpenVAS服务器之间的通信信息会使用SSL协议来进行加密保护, 因此接下来需要为OpenVAS服务器安装服务器证书, 那证书又是什么东西呢? 这个说起来就有点复杂了, 此处暂不讨论, 将会在第15章中完整介绍证书的相关技术内容。现阶段我们就使用OpenVAS服务器本身所提供的工具openvas-mkcert来生成证书即可, 方法如下:

Creation of the OpenVAS SSL Certificate

This script will now ask you the relevant information to create the SSL certificate of OpenVAS.

Note that this information will *NOT* be sent to anybody (everything stays local), but anyone with the ability to connect to your OpenVAS daemon will be able to retrieve this information.

```
CA certificate life time in days [1460]:
Server certificate life time in days [365]:
Your country (two letter code):CN
Your state or province name [none]: Taiwan
Your location (e.g. town) [Berlin]:Taipei
Your organization [OpenVAS Users United]:Bulls
```

在填完以上数据后, 最后可以看到如下的信息输出。这些信息记录了openvas-mkcert所生成的CA和服务证书所在位置及文件名。

Creation of the OpenVAS SSL Certificate

Congratulations. Your server certificate was properly created.

```
/etc/openvas/openvassd.conf updated
```

```
The following files were created:
```

```
. Certificate authority:
```

```
Certificate = /var/lib/openvas/CA/cacert.pem
```

```
Private key = /var/lib/openvas/private/CA/cakey.pem
```

```
. OpenVAS Server :
```

```
Certificate = /var/lib/openvas/CA/servercert.pem
```

```
Private key = /var/lib/openvas/private/CA/serverkey.pem
```

```
Press [ENTER] to exit
```

在完成以上三个操作后，我们就可以启动OpenVAS服务，方法如下：

```
service openvas-scanner restart
```

完成这个步骤后，OpenVAS就已经完成了所有安装操作。

3. 建立OpenVAS系统的使用帐户

接下来建立OpenVAS系统的专用帐户，不过请注意，OpenVAS的帐户与Linux系统的帐户没有任何关系，OpenVAS帐户的建立方法如下：

```
[root@localhost ~]# openvas-adduser
```

```
Using /var/tmp as a temporary file holder.
```

```
Add a new openvassd user
```

```
Login:: bulls
```

```
Authentication (pass/cert) [pass]:
```

```
Login password:
```

```
Login password (again):
```

```
User rules
```

```
openvassd has a rules system which allows you to restrict the hosts that jacky.chen has the right to test. For instance, you may want him to be able to scan his own host only. Please see the openvas-adduser(8) man page for the rules syntax.
```

```
Enter the rules for this user, and hit ctrl-D once you are done:
```

```
(the user can have an empty rules set)
```

```
192.168.0.0/24
```

```
10.10.15.0/24
```

```
Login : bulls
```

```
Password : *****
```



```
Rules      :
192.168.0.0/24
10.10.15.0/24

Is that ok? (y/n) [y]
user added.
[root@localhost ~]#
```

⑤

如上所示，OpenVAS建立帐户使用openvas-adduser命令，接着指定使用者的帐户名①。OpenVAS的验证方法有两种，一种是使用数字证书，另一种是使用密码来验证，如②所示OpenVAS的默认验证方法就是密码验证，还有就是帮助使用者bulls设置密码③，密码的部分会要求你连续输入两次以确认你密码的输入是否正确。

在OpenVAS系统上，为了更安全地使用帐户，每个帐户都可以设置其所能登录的IP地址④，在这里你可以设置一个IP地址或一个IP网段。如果要指定多个网段或IP，请你将其分成多行来输入即可。最后用Ctrl+D组合键来结束输入操作。这里要请你注意一点，如果没有设置IP，即代表这个帐户可以在任何一个IP登录。

13.1.4 进行弱点扫描

要进行弱点扫描的主机必须安装OpenVAS客户端软件，OpenVAS客户端可以安装在Linux及Windows平台，这里以Linux平台为例来说明如何使用OpenVAS客户端。图13-3显示的是在Fedora 14中打开OpenVAS客户端的方法，图13-4则显示了该界面。



图13-3 打开OpenVAS客户端

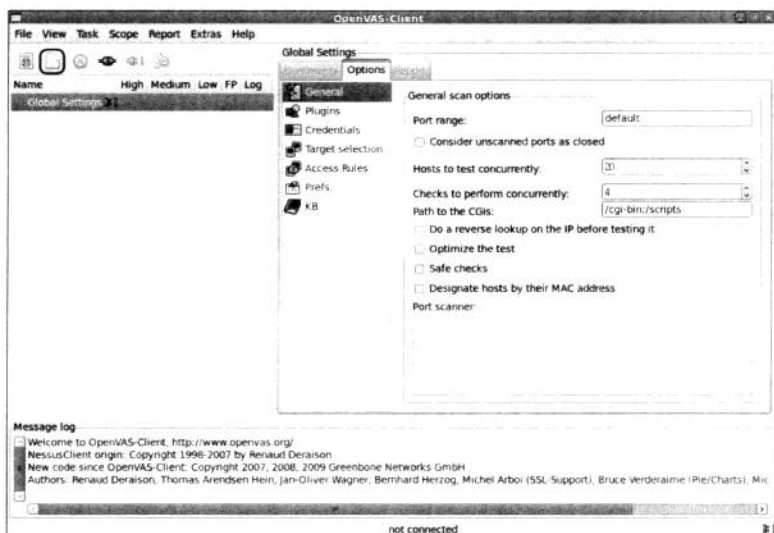


图13-4 OpenVAS客户端的界面

如图13-5所示，在OpenVAS客户端正常启动后我们可以使用鼠标选择在左上角中特别框起来的图标，以建立一个弱点扫描的任务(Task)。至于什么是任务呢？其实我们把它想象成“一个组”会比较容易理解，接着按如图13-6的方法来设置任务名称。

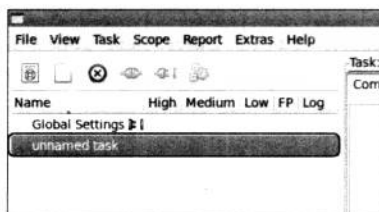


图13-5 建立新任务

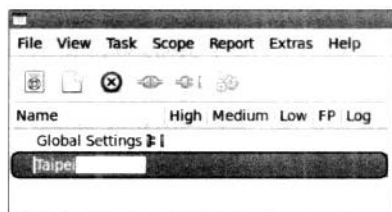


图13-6 设置任务的名称

设置好任务名称后，我们要在任务下建立新子任务，建立方法如图13-7所示。子任务建立好之后请为这个子任务设置一个容易记住的名称，如图13-8所示。

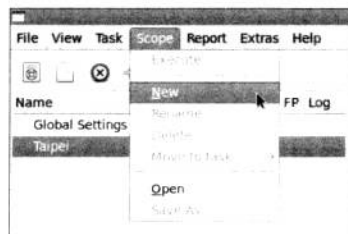


图13-7 建立新的子任务

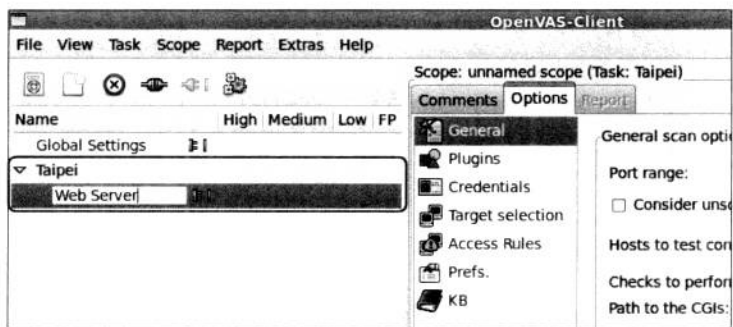


图13-8 设置子任务的名称

设置好任务名称后，接下来可以从图13-9①的位置看到这个子任务并没有连接到OpenVAS服务器，我们可以使用②所标示的图示来执行连接OpenVAS服务器的操作。另外在③的位置，设置所要连接的OpenVAS服务器及登录的帐户和密码。在正确连接OpenVAS服务器之后，OpenVAS客户端就会下载OpenVAS服务器上的所有插件，如图13-10所示。

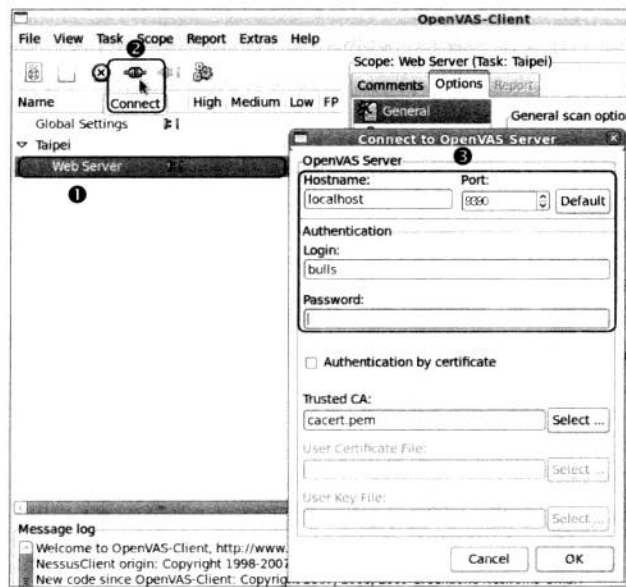


图13-9 连接OpenVAS服务器



图13-10 下载OpenVAS服务器上的插件

完成以上所有操作之后，最后设置这项子任务的内容，如图13-11所示，我们可以选择要使用哪些插件对待检测主机进行弱点扫描。从图中可以清晰地看到OpenVAS对于插件的分类，例如我们所要检测的是一台Linux的Web服务器，就不需要使用检测Cisco网络设备的插件，

因为Cisco网络设备的安全弱点应该不会出现Linux主机上。因此这里选择将用来检测Cisco设备的插件去掉,如果不去掉也没有什么关系,只是会消耗较长的时间去检测这台Linux主机而已。

另外有些弱点是在客户端登录之后才会遇到的弱点,因此我们需要告诉OpenVAS客户端要检测服务的帐户及密码,

如图13-12所示。接下来设置这个子任务所要测试的目标主机,我们只需输入其IP即可,如图13-13所示。

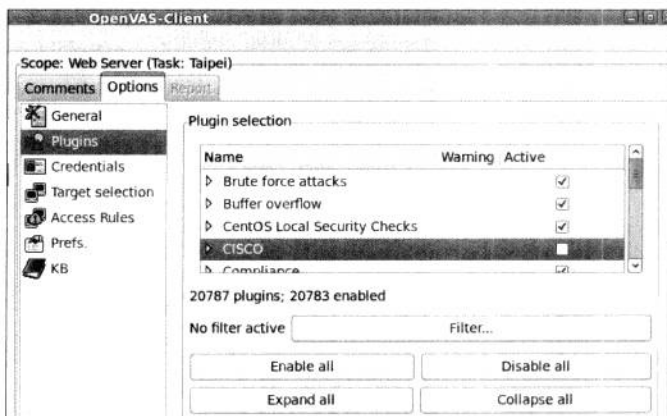


图13-11 选择插件

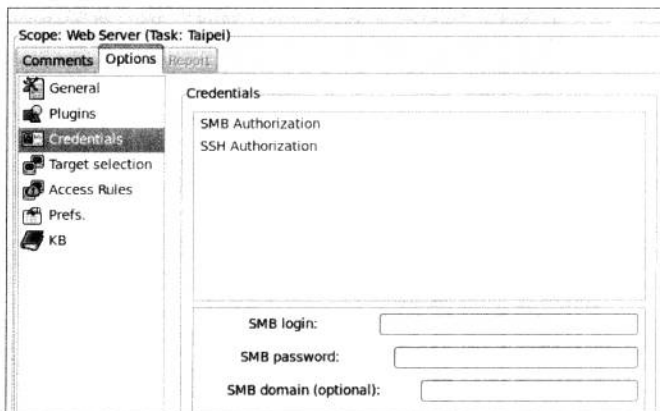


图13-12 设置服务的帐户及密码

终于到了大家翘首以待的时刻了,如图13-14所示,我们只需要按下图中上方已经框选的图标,OpenVAS就会开始对待检测主机进行弱点扫描,而整个扫描的操作需要耗费一段时间。如果待检测主机是在因特网的另一端,则需要更长的时间来执行检测操作。

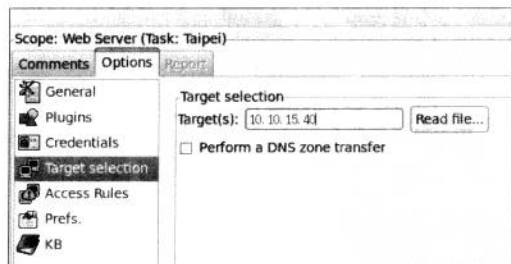


图13-13 设置所要扫描的主机



图13-14 执行扫描

经过漫长的等待之后，将可以看到如图13-15所示的内容。其中特别框选出来的位置就是OpenVAS检测到目标主机上的所有服务，如果其中有出现警告标志的服务，即代表这个服务有检测出弱点。在选取这个服务之后，可以在右边的窗口中看到这个弱点的相关信息。

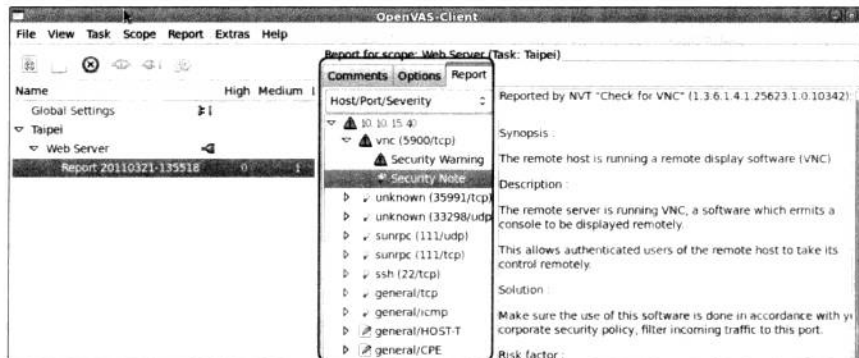


图13-15 查看扫描结果

如果你不满意上面的结果展现方式，OpenVAS客户端也提供了另外的结果输出功能，如图13-16和图13-17所示，我们可以使用Export功能将结果内容输出成文件，在这个过程中，还需要提供输出路径、文件名及格式。

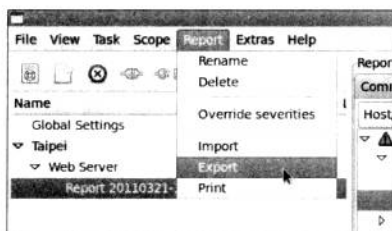


图13-16 输出扫描结果

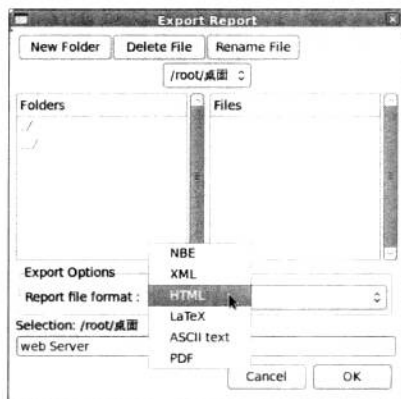


图13-17 设置输出路径及格式

图13-18就是输出文件，我们可以好好地阅读以下这个文件，这个文件会提供包含弱点的相关信息及补救措施等信息。

OpenVAS Scan Report - Mozilla Firefox

文件(F) 编辑(E) 查看(V) 历史(S) 书签(B) 工具(T) 帮助(H)

OpenVAS Scan Report

file:///home/liveuser/桌面/Web Server.html

访问最多 Fedora Documentation Fedora Project Red Hat Free Content

This report gives details on hosts that were tested and issues that were found. Please follow the recommended steps and proceed.

Scan Details

Hosts which were alive and responding during test	1
Number of security holes	0
Number of security warnings	1
Number of security notes	15
Number of false positives	0

Host List

Host(s)	Possible Issue
10.10.15.40	Security warning(s)

[return to top]

Analysis of Host

Address of Host	Port/Service	Issue regarding Port
10.10.15.40	ssh (22/tcp)	Security note(s)
10.10.15.40	sunrpc (111/tcp)	Security note(s)
10.10.15.40	vnc (5900/tcp)	Security warning(s)

图13-18 输出的报表内容

13.2 入侵检测系统

第12.2节中曾经介绍过WireShark这个工具,我们可以使用WireShark这类数据包的截取工具来抓取网络上正在传输的数据包,还可以通过WireShark工具来查看数据包中的数据信息。

其实WireShark就像“半个”入侵检测系统(Intrusion Detection System, IDS),因为所有网络攻击行为一定是通过数据包来进行,而且每种不同的攻击方法在网络上都会产生某些特定特征的数据包。例如,当发生Syn Flooding攻击时,网络上会瞬间涌入大量带有SYN标记的数据包,又如CodeRed病毒在感染Microsoft IIS Web服务时,在网络上一定会出现发送给网络服务器端口80的数据包,且数据包内一定包含“.idq?”字符串。

入侵检测系统其实就是一个数据包行为匹配系统,入侵检测系统需要先对各种已知的网络攻击行为进行分析,并将这些攻击行为的特征记录在行为数据库中。另外,入侵检测系统会不断地在网络上获取数据包,然后分析正在传输中的数据包是否符合数据库中的行为。如果有,那么入侵检测系统就会发出警报,这就是入侵检测系统的工作原理。

13.2.1 网络设备的限制

由于入侵检测系统是获取网络上正在传输的数据包来进行分析,因此关于数据包获取的知识我们就得好好研究一下了。首先我们要先了解交换机(Switch)及集线器(HUB)两种网络设备的差异。下面以图13-19为例来说明两者的差异。

● 集线器的特性:

从图13-19中我们可以看到,虽然这是一个6个端口的集线器,不过,在逻辑上这6个端口是连接在一起的。也就是说,当客户端A传输数据给客户端B时,因为这6个端口是连接在一起的,因此,客户端C与客户端D也是可以接收到这份数据的。所以在集线器所连接成的网络中,我们可以在集线器的任何一个端口上,获取到网络上任何一台计算机所发送的数据包。

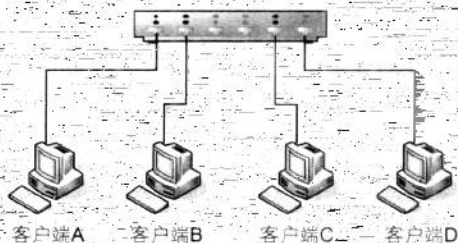


图13-19 集线器及交换机的差别

● 交换机的特性:

相对于集线器,交换机的特性就大不相同了。逻辑上,这6个端口都各自独立,但当要传输数据时,交换机就会帮我们吧某两个端口连接起来。例如,当客户端A要发送数据给客户端B时,交换机就会把第1及第2个端口连接起来。如此客户端A发送的数据就只有客户端

B可以收到，客户端C及客户端D就完全无法知道客户端A与客户端B正在传输数据。由此可见，在交换机的环境下要进行数据包的获取有一定难度的。

不过，如果真的要交换机的网络中获取数据包也不是不可能，如果你的交换机有提供生成端口(Spanning Port)的功能(Cisco称之为Switched Port Analyzer)那就没问题，这个功能的目的是把所有交换机端口上的数据包复制一份到特定的端口上，如此我们就可以在这个特定的交换机端口检测到整个交换机内所传输的数据包。但并不是所有的交换机都支持“生成端口”功能，如果没有，这个方法当然就不可行了。在此提供一个简单判断交换机是否支持“生成端口”的方法，只要交换机有支持VLAN(802.1Q)，那就应该有支持“生成端口”功能。

不过你要记住，如果我们启用Switched Port Analyzer功能之后，会对交换机设备带来很大负担，因此，交换机自身的负载能力就必须加以考虑。

13.2.2 入侵检测系统的分类

入侵检测系统因为“检测范围”的不同而有所差异，我们通常将其分为“主机型入侵检测系统”及“网络型入侵检测系统”，其差别如下。

● 主机型入侵检测系统：

例如在ISP机房内放置一台Linux主机(这种业务ISP称之为Colocation，中文是主机托管的意思)，为了维护这台Linux主机的系统安全，我们可以在这台主机上安装入侵检测系统，以便提高本机的安全性。由于我们所要监控的对象只有“本机”，因此称其为“主机型入侵检测系统”。

● 网络型入侵检测系统：

例如为了提升整个企业网络的安全性，决定在企业内部架设一台入侵检测系统，以提升整个“网段”的安全性。由于我们所要监控的对象是整个网段，因而称其为“网络型入侵检测系统”。

13.2.3 入侵检测系统的部署

在了解了网络设备的限制以及入侵检测系统的差异后，接下来说明部署入侵检测系统时应该注意的问题。以图13-20为例，首先注意图中所使用的网络设备是交换机，因此在DMZ及内部网络上部署“网络型入侵检测系统”是不太可行的，除非你愿意牺牲交换机的性能。我通常采用的处理方法如下。

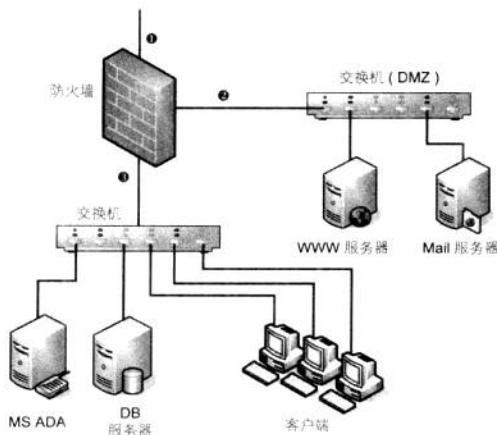


图13-20 入侵检测系统部署示例

● 如果防火墙主机无法改变：

在无法改变防火墙主机的情况下，我们可以选择在①②③的位置上串接一个集线器，然后将入侵检测系统主机连接到集线器上，如此就可以检测到内部网络段或DMZ网段对外的数据包传输情况。至于要串接在①②③的哪个位置，那就得依据你想监控的范围来确定。

一般来说，以上这种做法就足够了，但如果你担心在交换机下主机与主机之间的安全问题，不妨考虑在重要的主机上单独安装主机型入侵检测系统。

● 如果防火墙主机是可改变的：

如果防火墙主机将Linux或Windows与其他防火墙软件结合使用，或许就可以考虑在防火墙主机上直接安装入侵检测系统，如此就可以监测到整个企业对外的数据包传输情况，而无需额外增加任何网络设备或主机。

13.2.4 Snort入侵检测系统介绍

Snort早在1998年就已经开发出来了，并以GPL方式授权使用，其作者是Marty Roesch，是使用开源模式所开发出来的入侵检测系统，其官方网站是<http://www.snort.org>。

Marty Roesch在2001年创立了信息安全公司Sourcefire，虽然Snort软件本身依然可以免费下载，但Snort的规则数据库(Rules)则不再提供最新的免费版本。也就是说Snort的规则数据库是需要花钱购买的，并且价格也不便宜。如图13-21是Snort组织在2011年3月21日的报价，不过Snort仍可以免费提供不是最新的(晚30天)Rules下载，如果只是用于个人测试，你可以直接到Snort官方网站下载；但如果要用于企业，笔者建议你还是应该花钱购买最新的规则数据库。

VRT Rule Subscriptions

Subscribe Now for VRT Rules Updates

Registered users of Snort.org have the option to purchase a subscription to Sourcefire VRT Rule Updates. Subscribers have access to Rules updates 30 days faster than registered users. You must have a user account on Snort.org in order to purchase. If you do not have an account, please complete the [registration](#) process before proceeding.

Personal One Year Subscription

This subscription type is for use in a home network environment or for educational purposes only.

1 or more units - **\$29.99 each**

Quantity

Business One Year Subscription

This subscription type is for use in businesses, non-profit organizations, colleges and universities, government agencies, consultancies, etc. where Snort sensors are in use in a production or lab environment.

1 to 5 units - **\$499.00 each**

6 or more units - **\$399.00 each**

Quantity

图13-21 Snort规则数据库报价

13.2.5 下载及安装Snort入侵检测系统

可以在Snort网站<http://www.snort.org/snort-downloads/rhel5/>看到如下的信息内容，在这里我们选择使用RPM软件来安装Snort，因此选择如图13-22的第二个选项。接着按照网页的提示信息，在<http://vscojot.free.fr/dist/snort/snort-2.9.0.4/RHEL6/i386>网页下载Snort机相关的软件。在这里我们总共需要下载snort-2.9.0.4-11.e16.i686.rpm、daq-0.5-8.e16.i686.rpm、libdnet-1.12-7.e16.i686.rpm及libpcap1-1.1.1-9.e16.i686.rpm四个文件，此处将这四个文件保存到/root/snort目录下。

RHEL 5/CentOS 5 Users

As of Snort 2.9.0, and DAQ, Snort now requires the use of a libpcap version greater than 1.0. Unfortunately for people using RHEL 5 (and below), CentOS 5.5 (and below), and Fedora Core 13 (and below), there is not an official RPM for libpcap 1.0.

Sourcefire will not repack libpcap and distribute libpcap with Snort as part of an RPM, as it may cause other problems and will not be officially supported by Redhat.

This leaves you with one of two options:

1. Compile libpcap on your own. Get the Source code [here](#)
2. Use Vincent Cojot's RPMs. They are found [here](#)

Sourcefire does not make any endorsement to Vincent's RPMs or their contents. Feel free to use these at your own risk. Sourcefire will always recommend that you download and compile Snort from its source (and in this case, downloading and compiling libpcap from source as well). Snort's source code is available from our website [here](#)

Please see our blog post [here](#) and if there are any questions, please feel free to write me at joel@snort.org or use the [Mailing lists](#).

图13-22 Snort软件下载信息

接下来可以通过yum软件管理程序来安装Snort，命令运行方式如下所示：

```
yum -y localinstall --nogpgcheck snort-2.9.0.4-11.el6.i686.rpm \
daq-0.5-8.el6.i686.rpm libdnet-1.12-7.el6.i686.rpm libpcap1-1.1.1-9.el6.i686.rpm
```

13.2.6 下载及安装Snort的规则数据库

安装完Snort软件后，我们还无法使用Snort系统。你可以尝试查看/etc/snort/rules目录，这个目录里面应该还是空的，这是因为Snort软件并不包含Rules。因此接下来需要先连接到<https://www.snort.org/signup>注册为会员，才能下载Rule。这里注册的方法很简单，如图13-23所示，我们只需输入帐户名称、E-Mail、密码，并阅读Snort VRT的授权合约。如果同意这个合约则选中Accept VRT License Agreement，最后按Sign Up即可。



图13-23 Snort会员注册界面

在提交会员注册之后几分钟，Snort会发送一封E-Mail给你来激活帐户，如图13-24所示。请你单击E-Mail中的http链接，并输入你的帐户及密码即可激活帐户。

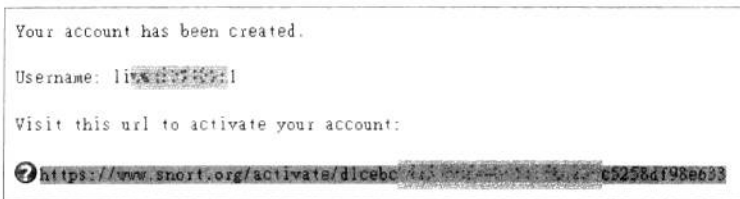


图13-24 Snort帐户激活

等到一切都就绪后，就可以到<http://www.snort.org/snort-rules/#rules>下载Rule，如图13-25所示。图的上半部是付费用户专区，图的下半部分是免费用户专区，由于我们之前下载的是Snort 2.9.X版，因此在这里就必须下载Snort V2.9版专用的Rule。

Subscriber Release		
The Subscription Release provides registered users of Snort.org with immediate access to the most up to date Sourcefire VRT Certified Rules available. Real-time access requires a paid annual subscription. For more information on a subscription click here or to purchase a VRT Rules subscription online visit the VRT Store .		
Documentation		
VRT advisory Ruleset change log		
Rule Documentation (opensource.gx)	MD5	16 Feb. 2011
Snort v2.9		
snortrules-snapshot-2902.tar.gz	MD5	22 Mar. 2011
snortrules-snapshot-2903.tar.gz	MD5	22 Mar. 2011
snortrules-snapshot-2904.tar.gz	MD5	22 Mar. 2011
Snort v2.8.6.*		
snortrules-snapshot-2861.tar.gz	MD5	22 Mar. 2011
Registered User Release		
The Registered User Release makes Sourcefire VRT Certified Rules updates available to registered users of Snort.org free of charge 30-days after the initial release to subscribers.		
Documentation		
VRT advisory Ruleset change log		
Rule Documentation (opensource.gx)	MD5	16 Feb. 2011
Snort v2.9		
snortrules-snapshot-2902.tar.gz	MD5	17 Feb. 2011
snortrules-snapshot-2903.tar.gz	MD5	17 Feb. 2011
snortrules-snapshot-2904.tar.gz	MD5	17 Feb. 2011
snortrules-snapshot-2901.tar.gz	MD5	17 Feb. 2011
Snort v2.8.6.*		
snortrules-snapshot-2861.tar.gz	MD5	17 Feb. 2011

图13-25 下载Snort的规则数据库

接着将snortrules-snapshot-2902.tar.gz解压缩到/root/rule目录下，如下就是解压缩之后的内容结构，接下来请将preproc_rules、rules及so_rules三个目录复制到/etc/snort目录下，另外再把etc目录下的内容复制到/etc/snort目录下，如此即可完成所有文件的复制操作。

```
[root@localhost rule]# ls -l
总计 16
drwxr-xr-x 2 1210 1210 4096 2011-02-18 03:36 etc
drwxr-xr-x 2 1210 1210 4096 2011-02-18 03:36 preproc_rules
drwxr-xr-x 2 1210 1210 4096 2011-02-18 03:36 rules
drwxr-xr-x 4 1210 1210 4096 2011-02-18 03:16 so_rules
[root@localhost rule]#
[root@localhost rule]# ls -l etc/
总计 2336
-rw-r--r-- 1 1210 1210 3621 2011-02-18 03:36 classification.config
-rw-r--r-- 1 1210 1210 25271 2011-02-18 03:36 gen-msg.map
-rw-r--r-- 1 1210 1210 1112 2010-01-20 02:34 open-test.conf
-rw-r--r-- 1 1210 1210 666 2011-02-18 03:36 reference.config
-rw-r--r-- 1 1210 1210 2266572 2011-02-18 03:42 sid-msg.map
-rw-r--r-- 1 1210 1210 19820 2011-02-18 03:36 snort.conf
-rw-r--r-- 1 1210 1210 2335 2011-02-18 03:36 threshold.conf
-rw-r--r-- 1 1210 1210 53841 2011-02-18 03:36 unicode.map
[root@localhost rule]#
```

不过默认的Snort配置文件/etc/snort/snort.conf并不适用于当前的环境。我们只有适当地调整内容所记录的某些路径,才能够让Snort正常工作。首先要调整第77行的内容,下面是修改前及修改后的内容。

● 修改前:

```
# Path to your rules files (this can be a relative path)
# Note for Windows users: You are advised to make this an absolute path,
# such as: c:\snort\rules
var RULE_PATH ../rules
var SO_RULE_PATH ../so_rules
var PREPROC_RULE_PATH ../preproc_rules
```

● 修改后:

```
# Path to your rules files (this can be a relative path)
# Note for Windows users: You are advised to make this an absolute path,
# such as: c:\snort\rules
var LIB_PATH /usr/lib
var CONF_PATH /etc/snort
var RULE_PATH $CONF_PATH/rules
var SO_RULE_PATH $CONF_PATH/so_rules
var PREPROC_RULE_PATH $CONF_PATH/preproc_rules
```

第二个需要调整的内容是第165行,下面是修改前及修改后的内容。

● 修改前:

```
#####
# Step #4: Configure dynamic loaded libraries.
# For more information, see Snort Manual, Configuring Snort - Dynamic Modules
#####
# path to dynamic preprocessor libraries
dynamicpreprocessor directory /usr/local/lib/snort_dynamicpreprocessor/
# path to base preprocessor engine
dynamicengine /usr/local/lib/snort_dynamicengine/libsf_engine.so
# path to dynamic rules libraries
dynamicdetection directory /usr/local/lib/snort_dynamicrules
```

● 修改后:

```
#####
# Step #4: Configure dynamic loaded libraries.
# For more information, see Snort Manual, Configuring Snort - Dynamic Modules
#####
# path to dynamic preprocessor libraries
dynamicpreprocessor directory $LIB_PATH/snort_dynamicpreprocessor
```



```
# path to base preprocessor engine
dynamicengine $LIB_PATH/snort_dynamicengine/libsfe_engine.so
# path to dynamic rules libraries
dynamicdetection directory $LIB_PATH/snort_dynamicrules
```

13.2.7 配置Snort

完成上面所有的操作之后，Snort就已经可以正常启动了。但如果希望Snort的工作能更符合我们的环境，那么适当调整snort.conf是必要的。接下来分析需要调整哪些参数。

- 设置Snort的监控范围：

监控范围是由ipvar HOME_NET参数来设置的，其默认值是any，意即只要是Snort在网络上可以获取到的数据包，都会列入监控范围。如果我们需要的是“网络型入侵检测系统”，那么这个值是可以接受的；但如果需要的是“主机型入侵检测系统”，就需要调整这个参数了。例如，主机的IP是192.168.1.10，我们就可以将var HOME_NET参数设置为192.168.1.10，如此Snort就只会匹配网络上与192.168.1.10相关的数据包。

正确设置var HOME_NET参数将有助于降低Snort的系统负载，又可以减少误判，因为Snort不必关心与本机无关的数据包。var HOME_NET参数的设置方式有下面几种。

- 主机型入侵检测系统：

```
ipvar HOME_NET 192.168.1.10
```

- 网络型入侵检测系统：

```
ipvar HOME_NET 192.168.1.0/24 ; 监测单一网段
```

```
ipvar HOME_NET [192.168.1.0/24,192.168.2.0/24] ; 监测多网段
```

- 告诉Snort所要匹配的数据库：

Snort的Rule放在/etc/snort/rules目录下，并且使用不同的文件名来分类不同的监测对象。例如，virus.rules是检测病毒的规则文件，pop3.rules是检测攻击pop3服务的相关规则。我们可以在snort.conf配置文件中决定要使用的匹配规则有哪些，因为并不是所有规则都会用得到。例如，如果只是单纯作为Web服务器的入侵检测系统，就不需要使用pop3.rules规则数据库，加上pop3.rules并没有任何意义，而且规则越多，Snort在进行匹配时所用的时间就越长，Snort的效率就越低。我们可以在snort.conf文件中看到很多如下的设置参数，这就是用来告诉Snort需要加载pop3.rules数据库。如果我们确定不使用这个数据库，只需在这行的第一个字符前加上#字符，然后重新启动Snort即可，其中#符号表示将这一行注释掉。

```
include $RULE_PATH/pop3.rules
```

● /etc/sysconfig/snort配置文件

Snort除了/etc/snort/snort.conf这个配置文件之外，/etc/sysconfig/snort也是一个重要的配置文件。不过在大多数情况下并不需要修改。如果安装Snort的主机上有多个网卡，就可能需要修改这个文件，需要修改的参数如下：

```
INTERFACE=eth0
```

INTERFACE参数是用来设置Snort的工作接口，其默认值是eth0，意思是说Snort只会通过eth0接口来获取网络上的数据包，但如果攻击行为是发生在eth1的接口上，Snort将无法发现任何异常行为。如果我们要让Snort同时监控多个实体网段，就需要调整这个参数，设置方法如下所示：

```
INTERFACE="eth0 eth1 eth2"
```

13.2.8 Snort的启停

Snort是以SysV Init Runlevels的方式进行管理的，因此，我们可以通过service命令来操作Snort，使用方法如下：

```
service snortd [ start | stop | restart ]
```

其中参数start代表启动，stop代表停止，而restart则代表重新启动(先停止，再启动)的意思；另外如果我们希望系统重新开机之后自动启动snort，可以使用chkconfig来达到这个目的，使用方法如下。

```
chkconfig snortd on
```

13.2.9 Snort的警告

在启动Snort之后，接下来就是等着攻击行为的发生，如果Snort检测到攻击行为，Snort会将这个攻击行为记录在/var/log/snort/alert文件中，但如果我们选择检测多个网络接口，记录文件就会变成/var/log/snort/INTERFACE_Name/alert。

下面特意使用OpenVAS对装有Snort系统的主机进行扫描，如此Snort就会检测到很多入侵行为，并将这些入侵行为以日志形式记录在/var/log/snort/alert文件中。

```

03/23-16:32:25.244660 [**] [1:17276:5] WEB-MISC Multiple vendor Antivirus magic byte
detection
evasion attempt [**] [Classification: Attempted User Privilege Gain] [Priority: 1] (TCP)
10.10.15.53:80 ->
10.10.15.47:55023
03/23-16:32:25.532019 [**] [1:3813:7] WEB-CGI awstats.pl configdir command execution
attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1] (TCP) 10.10.15.47:55025 ->
10.10.15.53:80
03/23-16:32:25.534349 [**] [1:3813:7] WEB-CGI awstats.pl configdir command execution
attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1] (TCP) 10.10.15.47:55026 ->
10.10.15.53:80
03/23-16:32:25.541070 [**] [1:3813:7] WEB-CGI awstats.pl configdir command execution
attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1] (TCP) 10.10.15.47:55027 ->
10.10.15.53:80
03/23-16:32:25.546921 [**] [1:3813:7] WEB-CGI awstats.pl configdir command execution
attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1] (TCP) 10.10.15.47:55028 ->
10.10.15.53:80
03/23-16:32:28.255716 [**] [1:402:8] ICMP Destination Unreachable Port Unreachable [**]
[Classification: Misc activity] [Priority: 3] (ICMP) 10.10.15.53 -> 10.10.15.47
03/23-16:32:28.681278 [**] [1:402:8] ICMP Destination Unreachable Port Unreachable [**]
[Classification: Misc activity] [Priority: 3] (ICMP) 10.10.15.53 -> 10.10.15.47
03/23-16:32:28.767942 [**] [1:6403:7] WEB-PHP horde help module arbitrary command
execution attempt
[**] [Classification: Web Application Attack] [Priority: 1] (TCP) 10.10.15.47:55066 ->
10.10.15.53:80
03/23-16:32:28.794946 [**] [1:6403:7] WEB-PHP horde help module arbitrary command
execution attempt
[**] [Classification: Web Application Attack] [Priority: 1] (TCP) 10.10.15.47:55073 ->
10.10.15.53:80

```

13.3 主动防御系统

在了解入侵检测系统的原理之后，你可能会想：“入侵检测系统对企业网络安全真的会有帮助吗？”有这样的想法其实一点都不奇怪，因为入侵检测系统检测到入侵行为时，最多是留下“警告信息”，其余什么都不会做。若这个警告信息是发生在半夜三更，应该也没有人可以立即处理，等到第二天早上上班时，该发生的入侵行为可能早就已经发生了。

为了解决入侵检测系统的缺陷，主动防御系统(Intrusion Prevention System, IPS)应运而生。所谓主动防御系统是指当有入侵行为发生时，系统可以产生自动的反应操作，例如，将入侵者的IP封锁等。说到这里你或许会想，那Linux系统中有没有主动防御系统呢？其实

是有的, Guardian是<http://www.chaotic.org/guardian/>组织为Snort所设计的插件, 我们只需将Guardian结合Snort来安装即可。

Guardian的原理其实很简单, Guardian是使用Perl语言编写的守护进程(Daemon), 这个守护进程会一直监视Snort的警告文件, 只要警告文件中有新的警告信息产生, 守护进程就会调用其他外部程序来封锁入侵者的IP, 例如iptables等。一段时间后, 守护进程会再次调用外部程序来清除掉被封锁的IP, 而这段时间的长短则可以由管理员自行设置。或许你会觉得很奇怪, 为什么要清除入侵者的IP呢? 道理很简单, 因为网络上的入侵行为很多都是以乱枪打鸟的方式进行的, 当入侵行为发生在你的主机上时, 我们就以Snort加上Guardian将其封锁, 当然这次入侵行为也就无效了。但别忘了这个入侵者依然还是在以乱枪打鸟的方式进行着, 只是下次要再打到我们的概率就低很多了, 因此, 我们可以暂时将其IP清除掉。另外Netfilter的规则越多, 防火墙的效率也就会越低, 这个清除操作将有助于避免防火墙性能的降低。

13.3.1 下载Guardian

Guardian的官方网站是<http://www.chaotic.org/guardian/>。我们可以从官方网站下载到Guardian软件, 并存放在/tmp目录下。在撰写本书时, 其最新版本是1.7版。

13.3.2 安装Guardian

下载到Guardian的软件后, 接着就可以开始安装Guardian, 首先将Guardian解压缩到/tmp目录下:

```
[root@localhost ~]# cd /tmp
[root@localhost tmp]# tar -zxvf guardian-1.7.tar.gz
```

解压完毕后, 接着将必要的文件复制到正确路径下, 并建立几个Guardian所需的配置文件, 如下:

```
cp guardian.pl /usr/local/bin/ ❶
cp scripts/iptables_block.sh /usr/local/bin/guardian_block.sh ❷
cp scripts/iptables_unblock.sh /usr/local/bin/guardian_unblock.sh ❸
cp guardian.conf /etc/snort/ ❹
touch /etc/snort/guardian.ignore ❺
touch /etc/snort/guardian.target ❻
touch /var/log/snort/guardian.log ❼
```

其中❶所指的就是Guardian的执行文件, ❷就是Guardian要封锁某一个IP时所调用的外部程序, ❸就是Guardian要解除对某一个IP的封锁时, 所需要调用的外部程序, ❹是

Guardian的配置文件，⑤是要求Guardian忽略某些IP的行为列表。例如，可以将Nessus服务器主机的IP写入到guardian.ignore，以免Nessus在进行弱点扫描时被Guardian封锁，⑥是告诉Guardian当入侵者对哪些IP进行入侵时Guardian才需要有所反应。例如，在eth0接口设置了A、B、C三个IP，在默认情况下，Guardian只会对入侵A(eth0接口上的IP)这个IP有反应，至于B(eth0:0)、C(eth0:1)两个IP是不会有反应的。如果希望当有入侵者试图入侵C和C两个IP时，Guardian也会做出反应，那就把B和C两个IP填写到guardian.target这个文件中，⑦是Guardian运行时所生成的日志。

13.3.3 设置Guardian

Guardian的配置文件位于/etc/snort/guardian.conf。不过默认的内容无法让Guardian正常工作。因此，接下来需要调整guardian.conf的文件内容，如下所示。

```
Interface eth0 ①
LogFile /var/log/snort/guardian.log ②
AlertFile /var/log/snort/alert ③
IgnoreFile /etc/snort/guardian.ignore ④
TargetFile /etc/snort/guardian.target ⑤
TimeLimit 86400 ⑥
```

在讨论配置文件内容之前，我们先来看看Guardian调用的外部程序内容，如下所示。

● /usr/local/bin/guardian_block.sh:

```
source=$1
interface=$2
/sbin/iptables -I INPUT -s $source -i $interface -j DROP
```

● /usr/local/bin/guardian_unblock.sh:

```
source=$1
interface=$2
/sbin/iptables -D INPUT -s $source -i $interface -j DROP
```

从这两个Shell脚本中我们可以清晰地看到，Guardian调用iptables工具做了哪些事情，而其中的source及interface两个变量就是Guardian在调用外部程序时所传入的参数。其中source就是Guardian从Snort的警告文件中取得的入侵者IP地址，interface则是配置文件中①所设置的。另外，你可能需要注意一下，从文件内容看，这两个外部程序只适用于“单机”IPS，因为规则是被加入的INPUT链，因此如果我们需要的是“网关式”IPS，就得手动去修改这两个Shell脚本文件的内容，将INPUT链改为FORWARD，当然，如果你选择再加一行iptables的语法，也是可以的。

②是设置Guardian工作时所产生日志的存放位置, ③是设置Snort警告文件的路径及文件名。前面曾经提到过, 如果Snort监控的接口不仅只有eth0, 其产生的警告文件就会有所改变。如果你忘记的话, 不妨回顾第13.2.7一节中的内容。④是设置ignore文件的路径及文件名, ⑤是设置target文件的路径及文件名, ⑥则是设置当一个IP被封锁之后, 在多长时间后将其清除, 其时间单位是秒。

13.3.4 Guardian的启停

Guardian并不像CentOS或RedHat Enterprise Linux上的其他服务一样, 可以通过服务命令来进行管理。Guardian的所有事情都需要自己手动完成, 因此我编写了如下Shell脚本来操作Guardian。

```
guardian.sh [ start | stop | restart | status ]
```

其中start代表启动, stop代表停止, restart代表重新启动, status代表Guardian的工作状态。

如果我們希望在系統重新開機之後自動運行Guardian, 可以將guardian.sh复制到/usr/local/bin目录下, 接着在/etc/rc.d/rc.local文件最后加上下面的一行命令。

```
/usr/local/bin/guardian.sh start
```

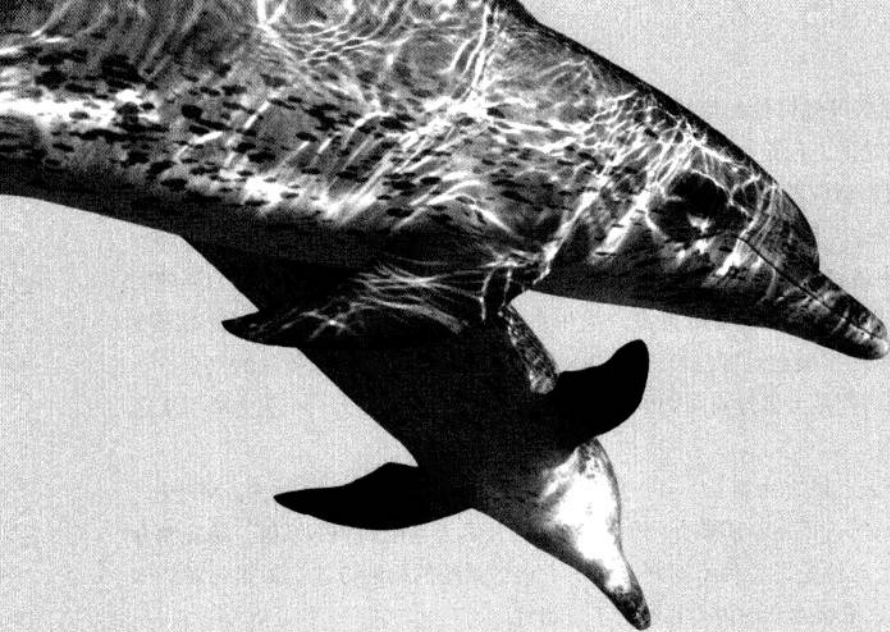
```
#!/bin/bash
#
# File Name : guardian.sh
#
start()
{
export PATH=$PATH:/usr/local/bin
/usr/local/bin/guardian.pl -c/etc/snort/guardian.conf
}
stop()
{
ps aux | grep 'guardian.pl *-c' 2>&1 > /dev/null
if [ $? -eq 0 ];
then
kill `ps aux | grep 'guardian.pl *-c' | awk '{print $2}`
else
echo "Guardian is not running .... "
fi
}
status()
{
```



```
ps aux | grep 'guardian.pl *-c' 2>&1 > /dev/null
if [ $? -eq 0 ];
then
echo "Guardian is Running ....."
else
echo "Guardian is not Running ...."
fi
}
case "$1" in
start)
start
;;
stop)
stop
;;
restart)
stop
start
;;
status)
status;;
*)
echo $"Usage: $0 {start|stop|restart|status}"
esac
```

13.4 小结

信息安全并非局限于防火墙之上，在学习完前面几章后，相信你已经意识到有很多事情是防火墙无法办到的，就以Web服务器自身的安全弱点为例，防火墙就不一定派得上用场。为了弥补防火墙功能上的不足，入侵检测就显得尤其重要了，当然大家常说的“预防胜于治疗”这句话也是适用于信息安全领域的，定期进行弱点扫描以找出潜在的安全漏洞更是一项重要的工作。



Linux

| 第14章 | VPN基础篇

14

前面的章节用了很多篇幅讨论如何加强企业网络安全。本章还将继续讨论这方面的话题,只不过这里所要讨论的方向是如何让企业网络的通信变得更安全。其实,加强通信安全的方法很多,如使用调制解调器来拨号或是帧中继(Frame Relay)的使用等,都可以提高网络上数据传输的安全,不过从方便性、经济性及安全性角度来综合考虑,我认为VPN(Virtual Private Network)是其中最好的选择。但是好一点的商业版VPN设备动辄几十万,如果使用VPN的人数很多,这些VPN的授权费用更是一般中小企业所无法负担的。然而近年来在Linux系统上,VPN技术的发展已足够与商业版VPN系统平起平坐,其所能提供的安全性、稳定性及高效性等优点,都是大多数商用VPN系统所望尘莫及的,当然另一个更为吸引之处在于它是完全免费的。

VPN技术发展到今天已经有几年了,且衍生出多种不同的VPN技术,比如说,MPPE、SSL VPN、CIPE、IPSec等。而本章所要介绍的是以IPSec技术为基础的VPN系统。除完整介绍IPSec的相关技术外,还讨论了各种不同网络架构下VPN系统的构建方式。如果你对VPN技术有兴趣或有所需求,那么本章的内容你绝对不能错过。

14.1 何谓VPN

VPN(Virtual Private Network)中文翻译是“虚拟专用网络”,而VPN的功能是将因特网虚拟成企业网络来使用,这样的描述似乎不是很容易理解,笔者以图14-1为例来说明VPN的功能及使用VPN的必要性。

如图14-1所示,笔者假设某家企业有两个办公地点,分别是台北总公司和上海分公司,而这两个办公地点的企业网络都设置了防火墙,且内部网络都是使用私有IP,另外由于公司的老板经常到世界各地出差。在这个网络的架构中,请你思考一个问题,即上海分公司及经常出差的老板,是否能够正常访问台北总公司内的文件服务器?答案当然是“不可能”,因为台北总公司内的文件服务器是放在NAT后面的私有IP网段上,因此上海分公司及出差的老板是不可能访问到文件服务器,但这样的需求在现实的网络应用中确实是需要的,难道没有很好的解决方案吗?

VPN正是为了解决这个问题而发展而来的技术。以前我们为了连接两个远程的私有IP网段,必须向电信局申请帧中继的服务,而帧中继感觉就像是电信局帮我们拉的一条很长的网线,然后通过这条网线来连接两个私有IP网段,虽然帧中继可以帮助我们解决以上问题,但是帧中继并不是万能的,下面就来看看帧中继的优缺点:

- 帧中继的优点:

帧中继的最大优点就在于“带宽保证”,也就是说,如果你申请的是1MB/s的帧中继,

就一定会有1MB/s的带宽可供使用，不会因为因特网的拥塞而打任何的折扣。

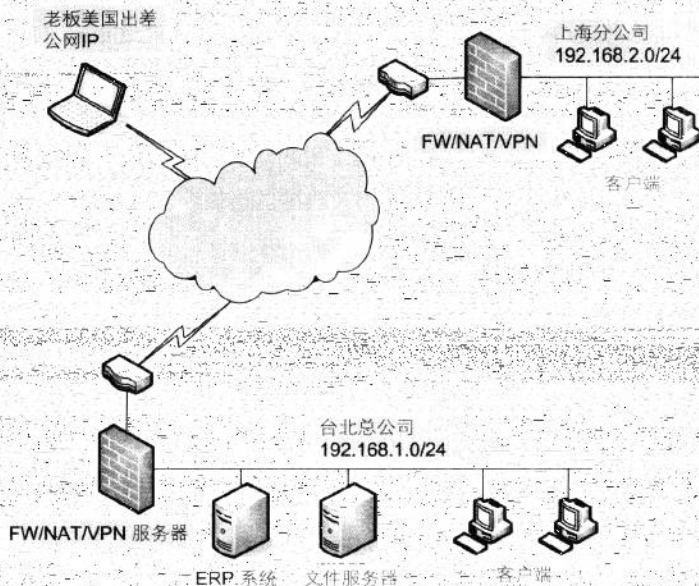


图14-1 VPN介绍(一)

● 帧中继的缺点：

帧中继服务的费用并不便宜，而且在帧中继网络中所传输的数据未经加密，因此其安全性比较差；此外，帧中继服务仅能应用于固定位置(地理环境)，如图14-2所示，老板是带着笔记本电脑到处出差，并没有固定位置，因此无法使用帧中继服务。

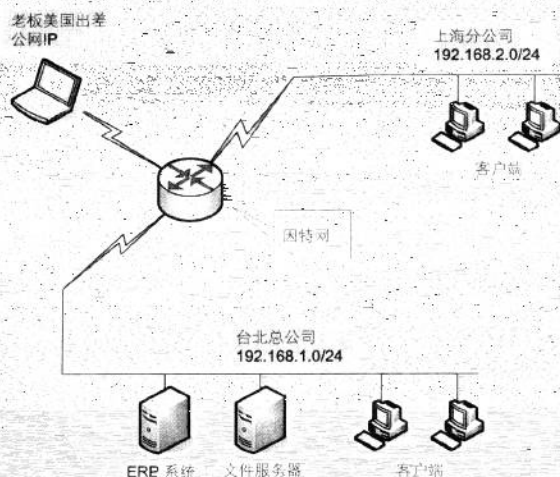


图14-2 VPN介绍(二)

相对于帧中继来说,VPN就不一样了,在早期由于局限于因特网的带宽限制,如果企业想高速跨越因特网来连接两个网段,那么帧中继绝对是首选,但近年来,因为ISP不断加大其所拥有的带宽,使得上网的速度不像以前那样慢、不稳定,再加上数据加密技术已经成熟且传输速度足够快,因此VPN技术已逐渐被企业所接收。那VPN究竟是什么样的技术呢?如果以图14-1为例来解释VPN技术,其逻辑上的架构就会如图14-2所示,也就是说,VPN将整个因特网虚拟成一个路由器,因此上海分公司及出差的老板,就可以通过这个路由器来访问企业内部的文件服务器。

14.1.1 VPN的原理

说到这里,你或许会问:“来源端及目的端的IP都是私有IP,VPN是如何让这些私有IP在因特网上传输的呢?”,这就是VPN一个最重要的特点。下面以图14-3为例来说明VPN技术如何让两台私有IP的主机能够跨越因特网来连接。

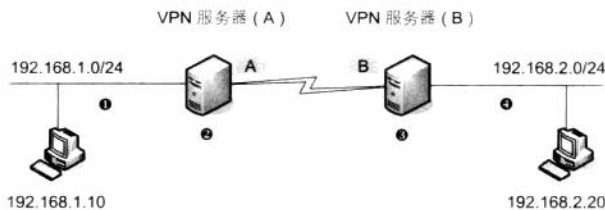


图14-3 VPN的原理

从图14-3中可以看到两台VPN服务器,各自都有两张网卡,其中一张是连接在因特网,我们假设其IP分别是A及B两个公网IP,另外的一张网卡则分别连接到192.168.1.0/24及192.168.2.0/24两个私有IP的网段。接着来分析192.168.1.10主机如何通过VPN技术,跨越因特网来连接192.168.2.20主机。

首先192.168.1.10主机发送数据包给192.168.2.20主机,当这个数据包发送到①的位置时,其数据包的来源端IP为192.168.1.10,目的端IP为192.168.2.20,如图14-4所示。



图14-4 VPN技术中的数据包处理（一）

但当这个数据包被发送到VPN服务器(A)之后②,VPN服务器会采用特殊方式处理这个数据包。如图14-5所示,VPN服务器会把原来的整个数据包当成其所要传输的数据内容,并且重新生成一个新的IP包头,而这个新的IP包头中的来源端IP就是VPN服务器(A)上的公网IP A,目的端IP则是VPN服务器(B)的公网IP B,如图14-5所示。

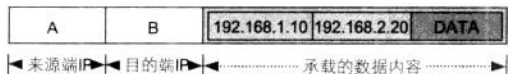


图14-5 VPN技术中的数据包处理（二）

IPB。如此这个数据包当然就可以从VPN服务器(A)，跨过因特网传输给VPN服务器(B)。

等到VPN服务器(B)收到这个数据包后③，其内容应该还是如图14-5所示，接着VPN服务器(B)就会去除掉整个新的IP包头，之后的内容就如图14-4所示，最后VPN服务器(B)将这个数据包送到192.168.2.0/24的网段上④，这样一来，两个私有IP的主机就可以跨过因特网来连接。

可以使用A和B两个IP来搭建出一条隧道，然后让192.168.1.10及192.168.2.20两台主机所传输的数据包得以穿梭于隧道之中，这种逻辑上的技术我们称之为隧道技术(Tunnel)。而VPN的隧道技术有很多种，一般较常见的有IP in IP Tunnel、L2TP Tunnel及PPTP Tunnel，但目前我们所讨论的是IP in IP Tunnel。

14.1.2 常见的VPN架构

VPN的架构大概可分为Site-to-Site和Client-to-Site两种。以图14-1为例，我们使用VPN将台北公司与上海分公司的两个网段串接起来，称之为Site-to-Site或是Subnet-to-Subnet的VPN架构；此外，老板用笔记本电脑通过VPN来连接台北总公司的网段，则称为Client-to-Site或Client-to-Subnet的VPN架构。

14.1.3 VPN的安全问题

在分析完VPN的原理之后，你或许无法接受VPN的技术。如图14-5所示，如果192.168.1.10与192.168.2.20传输的是极其机密的数据，那么在图14-3中，我们在VPN服务器A与服务器B之间使用WireShark之类的数据包获取软件，不就可以看到这些机密的数据内容吗？如果你有这样的想法，就表明你对网络数据传输的安全性已经有正确的认识。不过这个问题并不需要太关心，因为VPN机制允许我们在VPN服务器A与B之间进行数据加密，因此你可以暂时把加密过的数据包，想象成如图14-6的形式，这样就不必担心企业内的机密数据被窃取了。

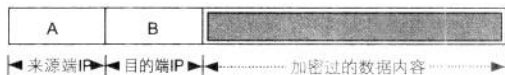


图14-6 VPN技术中数据包处理（三）

14.1.4 VPN机制的优缺点

VPN机制除了提供Site-to-Site及Client-to-Site功能之外，还提供了安全的通信环境。例如，当你到客户的公司洽谈生意时，如果通过客户公司的网络连接回公司查询产品成本信息时，难道你就不担心公司的成本信息会被窃取吗？因为你的客户只需要在机房里，就可以截取到你传输的所有数据，此时我们就可以使用Client-to-Site的VPN架构，从你的笔记本电脑中建立VPN Tunnel连回公司，并使用VPN加密的特性来保护传输中的所有数据。

还有谈到VPN的通信加密,就不免将其拿来与一般应用层的加密机制比较一下。这里以图14-7的HTTP通信协议为例来说明应用层加密机制与VPN加密机制的差异,首先,我们可以查看在HTTP的应用中,可选择两条路径将数据传输出去,其一是将HTTP数据包直接送到传输层,接着转发到网络层,最后由实体网络接口送离本机,如果是这条路径,那么HTTP数据包将无法得到SSL的加密保护;其二是将HTTP数据包先送到SSL加密机制处理,再将数据包转发到传输层、网络层,最后由实体网络接口送离本机。因此,如果是选择这条路径,那么HTTP数据包将可以受到SSL的加密保护。

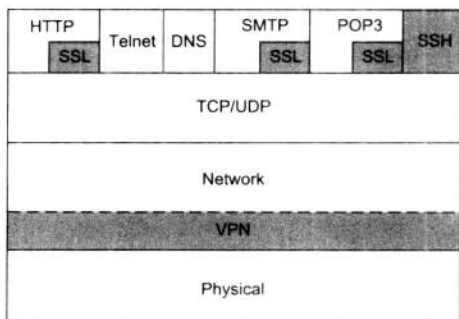


图14-7 应用层的加密机制

从以上的流程不难发现,应用层加密机制的缺点,就是这些加密机制只能加密特定的应用层协议。例如,ICMP数据包就无法使用这些加密协议来保护,因为ICMP协议是工作在网络层上的应用,因此ICMP数据包根本不可能用SSL的加密保护。而VPN就不是这样的,VPN是工作在网络层偏下一点的应用,因此,IP上的协议通通都可以使用VPN加密保护。

虽然VPN机制有这么多的优点,但如果在你的企业中有居心不良的员工,将企业的机密数据通过VPN连接传输给竞争对手,仍会束手无策,因为VPN将所有的数据包都加密了,所以无法使用任何方式来监控这类行为。因此建议你“限制从企业内部网络对因特网建立VPN连接”,以免发生这种事情。

14.2 数据加解密

由于因特网技术的不断发展,使得新的应用不断推陈出新。在过去,企业网络的安全问题几乎都落在防火墙之上,但由于网络远程访问需求与日俱增,使得通信加密的议题也逐渐受到重视,其中又以VPN的应用最受企业的重视。因此我们必须先对“加解密”技术的原理有所了解,如此才能对VPN的原理及配置有充分的了解及掌握。

14.2.1 何谓“明文”

“明文”泛指任何未经处理的原始数据。例如,“I Love YOU”就是明文,明文在网络上传输是非常不安全的。如图14-8是POP3协议应用中所截取到的数据包,而POP3协议就是以明文的方式来传输数据,因此我们可以直接阅读这些数据包中的数据内容。

80	15.842895	192.168.5.123	192.168.5.137	POP	Res
81	16.022280	192.168.5.137	192.168.5.123	TCP	122
82	16.903239	192.168.5.137	192.168.5.123	POP	Req
83	16.940824	192.168.5.123	192.168.5.137	TCP	pop
* Frame 80 (696 bytes on wire, 696 bytes captured)					
* Ethernet II, Src: Vmware_b6:2f:71 (00:0c:29:b6:2f:71), Dst: Gigabyte_99:c					
* Internet Protocol, Src: 192.168.5.123 (192.168.5.123), Dst: 192.168.5.137					
* Transmission Control Protocol, Src Port: pop3 (110), Dst Port: 1229 (1229					
* Post Office Protocol					
0240	38 35 39 2e 6d 31 4e 49	78 6f 39 6d 30 30 32 34	859.m1n1 xo9m0024		
0250	35 39 40 6c 6f 63 61 6c	68 6f 73 74 2e 6c 6f 63	590local host.loc		
0260	61 6c 64 6f 6d 61 69 6e	3e 0d 0a 54 6f 3a 20 6a	alldomain s to: 1		
0270	61 63 6b 79 40 6c 6f 63	61 6c 68 6f 73 74 2e 6c	ackyoaloc alhost.		
0280	6f 63 61 6c 6d 6f 6d 61	69 6e 0d 0a 33 73 62 6a	ocaldoma in. Subj		
0290	65 63 74 3a 20 49 20 4c	6f 76 65 20 59 6f 75 0d	ect: 1 L ove you.		
02a0	0a 0d 0a 0d 0a 49 20 4c	6f 76 65 20 59 6f 75 20 L ove you		
02b0	2e 0d 0a 0d 0a 2e 0d 0a				

图14-8 传输明文数据

14.2.2 何谓“密文”

由于明文在网络上传输极不安全，因此就有人发明出数据加密及解密的技术。所谓的加密和解密指我们可以使用数学算法将一份明文数据转换成密文。同样，也必须可以将密文转换回明文，如此就是数据的加密和解密。为此笔者随意设计出一个数据加密和解密的过程。相信你在看完之后，对此会有更进一步的认识。不过，在开始讨论这个示例之前，我们必须先了解XOR及移位这两种逻辑运算，如此对稍后的示例才不会感到困惑。

● XOR逻辑运算：

如图14-9所示，A XOR B得到C，我们将A和B可能出现的情况做了排列组合，可以得到四种情况，根据XOR运算后得到的结果。最后我们可以得到一个结论，只要A等于B，所得到的结果就是0，如果A不等于B，所得到的结果就是1。

● 移位运算：

例如，我们有一个字节的数据是0000-1111，如果要让它右移一位，就是将其最右边的一位拿到最左边，最后会变成1000-0111，如果要让它左移两位，就是将其最左边的两位拿到最右边，最后会变成0011-1100。

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

图14-9 XOR逻辑运算

在了解了以上两个逻辑运算后，接下来我们使用笔者所设计的加解密算法，来尝试将“I Love You”字符串加密并解密。首先将待加密的字符串填写到图14-10最左边的位置，并且将这些字符转换成ASCII码，接着让它右移4位。另外，在运算过程中，我们需要额外提供一个常数，例如，在使用WinZIP对一个文件加密时，WinZIP会要求我们输入一组密码。本例全部以0000-1111为常数，而所谓的常数在密码学中则称之为“密钥”，接着我们把右移4位之后的值与0000-1111做XOR的逻辑运算。当运算完毕后所得到的值就是密文，最后我们

用16进制来表示密文，即9B、0D、CB、F9、68、59、0D、9A、F9、58。

明文	明文的ASCII	右移4位	加密常数	密文	16进制
I	0100 1001	1001 0100	0000 1111	1001 1011	9B
空白	0010 0000	0000 0010	0000 1111	0000 1101	0D
L	0100 1100	1100 0100	0000 1111	1100 1011	CB
o	0110 1111	1111 0110	0000 1111	1111 1001	F9
v	0111 0110	0110 0111	0000 1111	0110 1000	68
e	0110 0101	0101 0110	0000 1111	0101 1001	59
空白	0010 0000	0000 0010	0000 1111	0000 1101	0D
Y	0101 1001	1001 0101	0000 1111	1001 1010	9A
o	0110 1111	1111 0110	0000 1111	1111 1001	F9
u	0111 0101	0101 0111	0000 1111	0101 1000	58

图14-10 自制的加解密算法

所以9B、0D、CB、F9、68、59、0D、9A、F9、58原来的明文就是I Love You。我们实在很难从中找到任何关联性，当然这就是数据加密所带来的好处：保护机密的数据不被他人随意阅读。但对数据的接收者来说又该如何看懂这些被加密的数据呢？接着来看如何将以上数据通过解密算法还原成I Love You字符串，首先将9B、0D、CB、F9、68、59、0D、9A、F9、58数据放到图14-10表格中最右边的列里面，并且将其转换成二进制的密文。接着解密者必须知道加密时所使用的密钥，就像我们使用WinZIP对文件进行解密时，会要求输入密码的意思。当然如果给错，就无法将被加密的数据还原回来。接着，将密文部分与常数进行XOR的逻辑运算，然后再将数据左移4位，最后将运算后的值转换成ASCII码对应字符，就可还原成I Love You字符串了。

14.3 数据加密类型

所有加解密算法大概可以分成对称式加密及非对称式加密两种，下面列出其差异及优缺点。

14.3.1 对称加密

图14-11就是加密及解密的过程，图中我们试着将“I Love You.”字符串进行加密及解密。首先必须将待加密的数据，以及加密时所使用的密钥提供给加密算法，如此即可将“I

Love You.”加密，而加密之后的内容就变成一堆无法阅读的数据；当我们需要阅读这些数据时，需要将被加密过的数据及解密时所需要的密钥提供给解密算法，如此即可将被加密的数据还原成我们可以直接阅读的数据。



图14-11 对称加密

何谓对称加密算法呢？如果加密和解密所使用的密钥是相同的，就称其为对称加密。与非对称加密相比，对称加密的优缺点如下。

● 优点：

对称加密最大的优点在于加密速度很快，也就是说，对称式加密占用的CPU运算资源较少。

● 缺点：

对称加密的最大缺点是密钥传输不便。如图14-11所示，如果数据加密者位于上海，而数据的接收者位于地球的另一端时，试问数据的接收者该如何获得加密者所使用的密钥呢？是用E-Mail、电话还是传真呢？以上这些方式我们都无法保证密钥在传输过程中不会被第三者所截获，如果不幸被他人所截获，我们的机密数据也就等于被他人窃取了。

14.3.2 非对称加密

在开始讨论非对称加密时，必须先了解什么是公钥系统。在公钥系统中，文件的接收者必须拥有一对密钥，分别是公钥(Public Key)和私钥(Private Key)。这两种密钥有一个很重要的特性，就是都可用来加密数据，而且加密后的数据就只能使用另一半来解密。例如，使用公钥加密后的数据，就只有私钥能解密；而使用私钥加密后的数据，就只有公钥才能解密。所以从公钥及私钥的字面意义上看，不难想象公钥应该是可以让他人轻易获得的数据，但私钥就该严密保管不能被他人取得。

在了解公钥及私钥后，下面以图14-12为例来说明非对称加密的工作流程。首先，加密者可以轻易取得文件接收者的公钥，以便对数据进行加密，而这份被加密后的数据就只有

“拥有这个公钥所对应的私钥”才可以解得开。但由于这个私钥就只有一个人拥有，因此，当接收者收到这份文件后，就可以使用私钥来将数据解密。

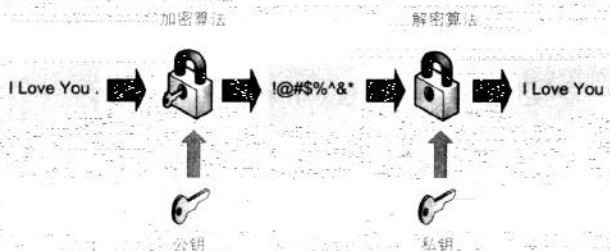


图14-12 非对称加密

下面列出非对称加密的优缺点。

- 优点：

由于非对称加密系统使用公钥及私钥来进行加解密运算，因此非对称加密系统不存在密钥传输问题，比对称式加密系统更安全。

- 缺点：

非对称式加密系统的最大缺点在于其进行加解密运算时，所消耗的CPU资源比较多。

14.4 哈希算法

哈希(hash)算法在密码学中占据了非常重要的地位。在哈希算法中，我们可以任意将数据拿来运算。如下所示：使用openssl命令对file.txt进行md5的哈希计算，而计算完毕所得到的“92ab70b47c2285708f83ae097053df8e”则称为数字指纹(fingerprint)。

```
[root@localhost tmp]# openssl md5 file.txt
MD5(file.txt)= 92ab70b47c2285708f83ae097053df8e
```

指纹有什么特点呢？就如同我们的指纹一样，你不可能找到两个不同的人有相同的指纹，在哈希算法的应用中也是这样的，不相同的两个文件(文件的内容不同)不可能计算出相同的指纹，这种“唯一”性可以让我们应用在很多不同的地方。例如，通过因特网下载文件回来，该如何去确认这些文件在传输的过程中完全没有错误？因此，我们在下载某些文件时，这些网站的管理人员会提供如图14-13的信息。也就是说，网站的管理人员会告诉我们，这些文件在正确的情况下文件的指纹应该是什么，如此我们就可以使用与网站管理者相同的哈希算法，来对下载的文件进行哈希运算，并将运算后所得到的指纹与网站所提供的指

纹进行匹配。如果匹配结果相同，就足以证明下载回来的文件内容是正确的；反之，你就得再下载一次了！

```
2007-Dec-22: iptables-1.4.0
iptables 1.4.0 has been released

ChangeLog
iptables-1.4.0.tar.bz2: GPG signature (key): md5sum 90cfa8a554a29b0b859a625e701af2a7
Patch against 1.4.0rc1: md5sum 0a45db471aa0ccb52438dda004a07bb
```

图14-13 应用哈希算法

14.4.1 常见的哈希算法

哈希算法只是一个笼统的名称，实际上，哈希算法可以分为好几种，例如：md2、md4、md5、rmd160、sha、sha1等，一般比较常用的是md5、sha及sha1。以下示例则使用openssl工具来执行各种不同的哈希运算。

```
[root@localhost tmp]# openssl md2 file.txt
MD2(file.txt)= 913af0b1a393c31f5d41a3e835945d3f

[root@localhost tmp]# openssl md4 file.txt
MD4(file.txt)= e4a7fbca4857a2943104a3453311a4c0

[root@localhost tmp]# openssl md5 file.txt
MD5(file.txt)= 92ab70b47c2285708f83ae097053df8e

[root@localhost tmp]# openssl rmd160 file.txt
RIPEMD160(file.txt)= 369d3ba14770918e40c2f71e0edd5210fbfbf26f

[root@localhost tmp]# openssl sha file.txt
SHA(file.txt)= 759b33290657faadd48df6f78d61c4a87ff6d1e2

[root@localhost tmp]# openssl sha1 file.txt
SHA1(file.txt)= 8a1af798345bd3b1b8769973786bb96fcfa38631
```

14.4.2 哈希算法的特性

哈希算法的使用相当简单，如果我们可以对其详细了解，就可以更好地应用它。其特性如下。

- 计算的数据来源无大小限制。
- 相同的哈希算法会有固定长度的指纹输出。
- 不同的数据内容计算后所得到的指纹不同。

- 无法由指纹还原出原始文件的内容。

14.5 基于IPSec的VPN

现在介绍IPSec技术的书籍很多，且大部分都写得太过于详细并且重于理论，对于IPSec技术的初学者来说，并不太容易理解其内容。这里将着重讨论与IPSec部署有关的主题，至于一些与部署实施无关的理论部分，受篇幅的限制就不加以论述。如果你有兴趣研究IPSec协议，相信在研读完本书之后，你应该可以轻松阅读其他IPSec专著。

要了解IPSec的工作及部署并没有想象中困难，只要根据此处编排的内容顺序来阅读，应该很容易就可以了解IPSec技术，这样对于整个IPSec设置及部署就不会有太大的问题。

14.5.1 IPSec的工作模式

IPSec在不同的应用需求下会有不同的工作方式，分别是传输模式(Transport Mode)及隧道模式(Tunnel Mode)，接下来介绍这两种工作模式的使用环境。

- 传输模式：

如图14-14所示，所谓传输模式的IPSec VPN是指可以将两台主机之间所传输的数据机密。例如，我们使用笔记本电脑通过POP3协议，连接回公司的邮件服务器收取邮件时，就可以在邮件服务器与笔记本电脑之间建立传输模式的IPSec VPN，以免邮件内容被他人窃取。



图14-14 传输模式

- 隧道模式：

如图14-15所示，如果我们要使用IPSec VPN来加密两个不同的网段所传输的数据内容，或者需要将两个私有IP的网段通过IPSec VPN来跨越因特网连接，就需要用到隧道模式的IPSec VPN。

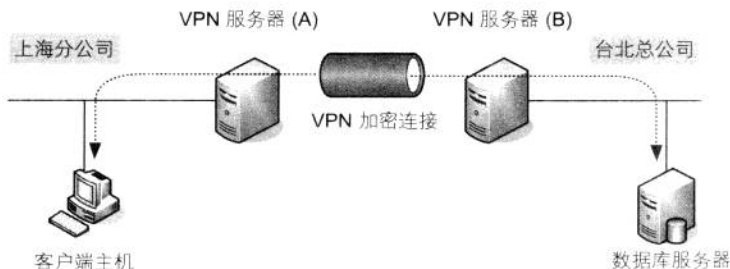


图14-15 隧道模式

14.5.2 IPSec的组成要素

IPSec通信协议其实是由两个不同的协议组成，分别是AH(Authentication Header)和ESP(Encapsulated Security Payload)，而这两个协议所负责的任务功能是不同的，其中AH协议的功能是“完整性验证”，ESP协议的功能是“完整性验证与加密”。接下来分析AH与ESP协议的原理及工作方式。

● AH:

AH(Authentication Header)协议是以数字签名的方法来确保数据的完整性。例如，我们可以使用AH协议来确保传输过程中的数据内容没有遭到篡改或因传输质量不佳造成错误，另外AH协议可以工作在传输模式及隧道模式两种模式中，但要注意AH协议不包含加密功能。下面说明AH协议在传输模式及隧道模式中的工作原理。

● 传输模式中的AH协议:

如图14-16所示，如果要在两台主机之间，使用AH协议来保护所传输的数据包，那么必须先为两台主机设置一个“加密密钥”。如图中，我们假设主机A的密钥是Key(A)，主机B的密钥是Key(B)，但因为AH协议是使用“对称加密”，因此，我们必须把主机A的密钥送给主机B，主机B的密钥送给主机A。不过目前你先不要关心密钥的生成方式、设置方式及交换方式，稍后将会进行完整的说明。



图14-16 传输模式中的AH协议

当准备好AH协议运行时所需要的加密密钥后，接下来介绍AH协议的工作原理。以图14-17的主机A发送数据给主机B为例。首先，AH协议会在原有的IP包头及TCP包头之间插入

一段AH包头,接着对整个数据包进行哈希运算。事实上,某些在IP包头及AH包头中的数据并不包含在内,因为这些数据可能会是变动的。例如,IP包头中的TTL值就不包含在内,因此在计算之前,这些数据字段会被设置为0,而计算后所得到的指纹会以Key(A)为加密密钥来进行对称加密。加密完毕得到结果称为验证数据,最后将验证数据填入AH包头之中。

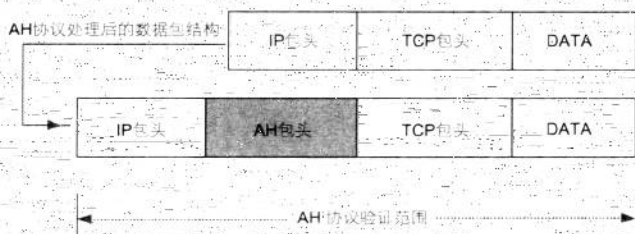


图14-17 传输模式中AH协议的数据包结构

等到主机B收到数据包之后,主机B就可以将Key(A)作为解密密钥,将AH包头中的验证数据解密,如此即可得到主机A所计算出来的指纹,我们令其为F(A)。接着主机B也对收到的整个数据包进行哈希计算,这样当然也会得到一个指纹,而这个指纹是立即计算出来的,我们令其为F(B)。最后如果F(A)等于F(B),那么根据哈希算法的特性,我们就可以确认数据包在传输过程中,其数据内容没有任何改变。

在了解AH协议在传输模式中的工作方式后,或许你会觉得很奇怪:“为什么A、B主机各需要一个加密密钥?而不是只需要一个加密密钥就够了?”这是因为IPSec当初在设计时,特别把AH协议设计成“单向的”。也就是说,我们可以只选择主机A发送数据给主机B时要用AH协议,而主机B发送数据给主机A时则不使用AH协议。当然,你也可以反过来设置,不过通常我们的选择会是“双向”都使用AH协议。

AH协议之所以被设计成单向,最大考虑因素在于“安全”。如果A、B主机只有一个公用的加密密钥,可能因为加密密钥容易被他人猜中,而导致A、B主机双向传输数据的不安全。但如果A、B主机的加密密钥是各自独立的,以上风险至少可以降低一半,因此AH协议才被设计成单向。

图14-18是传输模式中AH协议的实际数据包结构,从结构中我们可以验证图14-17的内容,也就是AH协议在IP包头及TCP包头之间插入了AH包头,另外AH包头中的IV(Integrity Value)字段,就是前面我们所提到的“验证数据”(被加密后的指纹)。最后我们要了解,AH协议并不提供任何对数据包进行“加密”的机制,因此,从图14-18右下角,可以直接看到数据包内所承载的数据内容。

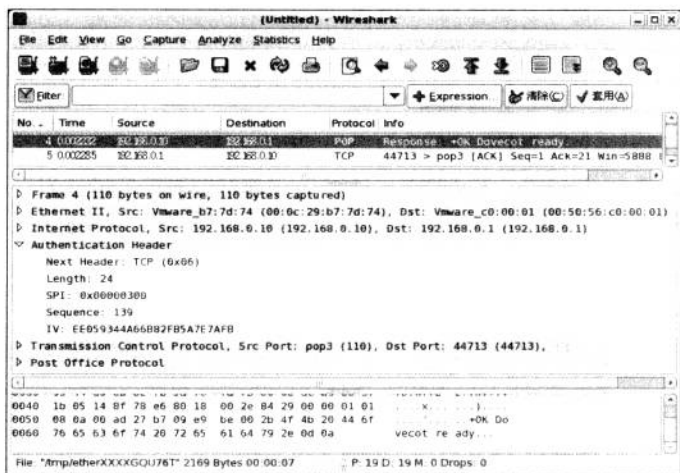


图14-18 传输模式中AH协议的数据包结构

• 隧道模式中的AH协议:

如图14-19所示, 如果我们希望通过两台VPN主机来保护192.168.0.0/24及192.168.2.0/24两个网段所传输的数据包。在传输模式中, 我们必须先将加密密钥准备好, 再互相交换两台主机的加密密钥, 这部分的原理与传输模式并无太大差别。

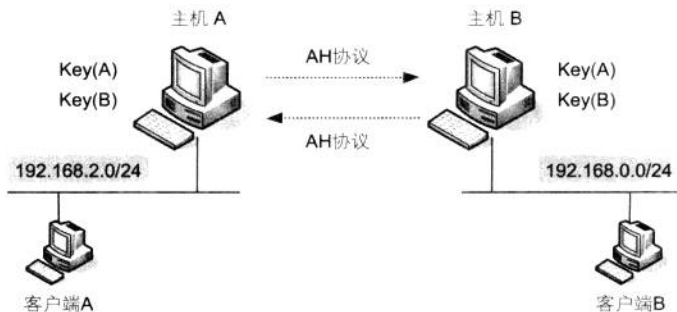


图14-19 隧道模式中的AH协议

接下来分析在隧道模式中AH协议处理数据包的方式。如图14-20所示, 当客户端A发送数据包给客户端B时, 其数据包就如图中上半部分的结构, 当这个数据包送到VPN主机A时, 主机A会将整个数据包当成数据来看, 并在这个原来的数据包之前加入一个AH包头, 然后再生成一个新的IP包头, 接着使用哈希算法计算出整个数据包的指纹, 再使用Key(A)将指纹加密, 然后放到AH包头之中。整个工作过程与传输模式中的工作过程是一样的。



图14-20 隧道模式中AH协议的数据包结构

VPN主机B收到这个数据包后，VPN主机B就会使用AH包头中的验证码来执行完整性验证，整个工作过程与AH的传输模式相同，如果验证没有成功就去丢弃这个数据包；反之，VPN主机B会将新的IP包头及AH包头去除，此时这个数据包就可以通过VPN主机B的路由表，将数据包发送到正确的位置。

图14-21就是隧道模式中AH协议的实际数据包结构，从结构中可以看到图14-20的内容。可以看到新的IP包头内有SRC：10.0.1.200、DST：10.0.1.210，也可以看到AH包头及其内的验证码，还有原始IP包头，从其中可以看到SRC：192.168.2.10、DST：192.168.0.10，接着是一个TCP包头，最后可以看到数据包所承载的是POP3的应用。

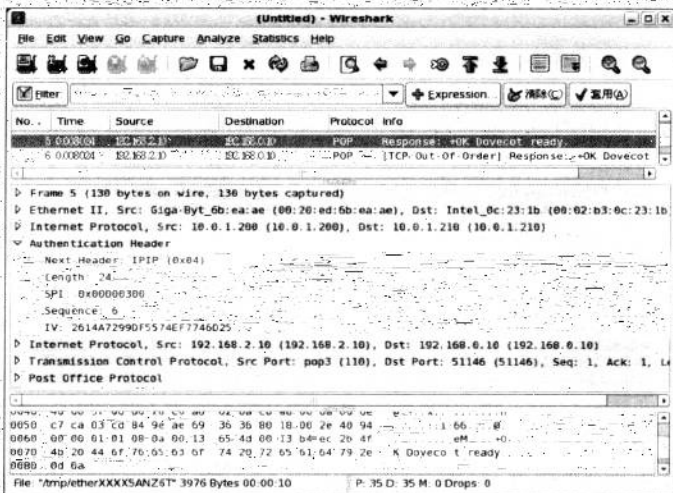


图14-21 隧道模式中AH协议的数据包结构

● ESP:

在ESP(Encapsulated Security Payload)协议中，这里要讨论的是“加密”及“完整性验证”两项功能，其中加密的目的在于避免数据包传输时遭到截取而泄露。而完整性验证的目的

的在于确认数据包在传输过程中是否遭到篡改，或者是否因为网络传输质量不好而造成的数据错误。这两项功能在ESP协议中可以分开使用，也可以结合使用，这完全根据使用者的需求来确定，只不过一般在使用ESP协议的情况下，通常都会选择加密功能，至于完整性验证的部分就不一定会用得上，当然如果可以一起使用，那么对于数据包的安全性就能得到更高的保证，接下来看看ESP协议中的加密及完整性验证的工作方式。

● 传输模式中的ESP协议

如图14-22所示，相信你会觉得怎么会有这么多密钥，如果你对于前面所提到的数据加解密理解比较深刻的话，那么这一堆的密钥一定难不倒你。首先来认识这几个加密密钥的用途。因为在ESP协议中有加密及完整性验证两项功能，如果我们使用了加密功能，就必须提供一个加密密钥，即图中的Host-A_key-E；另外，如果也使用了完整性验证，则必须提供另一个完整性验证时所需的加密密钥，即图中的Host-A_key-A。最后，为了让主机B可以解开主机A所送来的数据包，就必须将主机A的两个密钥传输给主机B，所以主机B也会有Host-A_key-E及Host-A_key-A两个密钥。

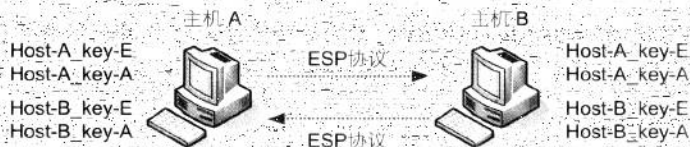


图14-22 传输模式中的ESP协议

如同AH协议一样，ESP协议也是单向的，因此如果我们要使用双向的ESP协议，就必须给主机B也准备两个密钥：分别是为数据加密用的Host-B_key-E，以及数据包完整性验证使用的Host-B_key-A。当然，这两个密钥也必须送给主机A，如此一来，主机A及主机B上分别会有4个密钥。

了解ESP协议工作时所需的密钥之后，接下来讨论ESP协议加密工作的流程。我们假设主机A发送数据包给主机B，如果不使用ESP协议，其数据包的结构就如图14-23上半部分所表示的样子；但如果使用ESP协议，首先会在IP包头及TCP包头之间插入一个ESP包头，并在数据包的末端加入ESP Trailer字段，而这个字段内主要是保存ESP协议在工作时的一些参数，但这部分并不在本书的讨论范围内，是否了解都不会影响到我们对于IPSec VPN的设置。完成以上操作后，接着将TCP包头、DATA及ESP Trailer三个部分以Host-A_key-E加密。通常使用Host-A_key-E(TCP包头、DATA、ESP Trailer)来表示加密操作及加密范围，如此就完成了ESP协议的加密操作。

但如果要使用ESP协议的完整性验证功能，接下来系统就会对ESP包头及Host-A_key-

E(TCP包头、DATA、ESP Trailer)进行哈希计算,接着再使用Host-A key-A对指纹进行加密,加密完成后的数据再附加在数据包的最末端,如此即完成了ESP协议的完整性验证处理。

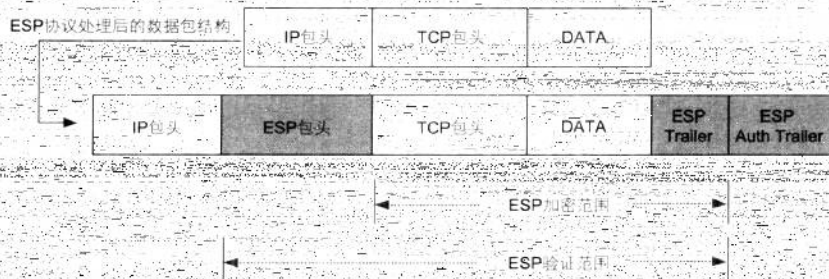


图14-23 传输模式中ESP协议的数据包结构

在主机B接收到这个数据包后,主机B就会先对ESP包头及Host-A key-E(TCP包头、DATA、ESP Trailer)进行哈希计算,我们假设计算完成后的指纹是F(A),接着再使用主机B上的Host-A key-A把ESP Auth Trailer部分解开,如此即可得到当初主机A所计算出来的指纹,我们令其为F(B)。如果F(A)等于F(B),就可以判断数据包从主机A发往主机B的过程中,数据包的内容没有任何改变,如此即完成完整性验证的检查;最后主机B使用上面的Host-A key-E将Host-A key-E(TCP包头、DATA、ESP Trailer)解密,此时,主机B就可以完全获取到主机A所传输过来的数据包。

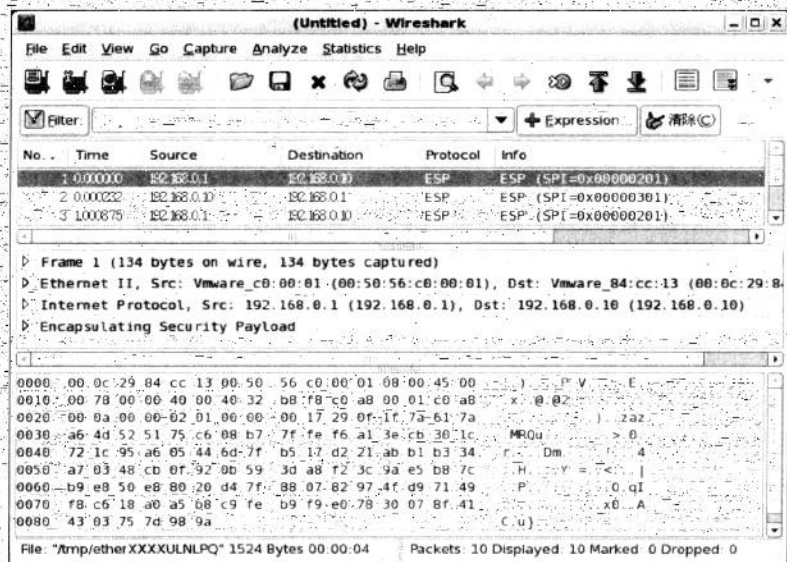


图14-24 传输模式中ESP协议的数据包结构

我们可以从图14-24看到ESP协议在传输模式中的数据包结构，而图中的内容也验证了图14-23的内容。在IP包头中，除ESP包头之外什么也看不到，因为所有数据内容都被加密了。

- 隧道模式中的ESP协议：

以图14-25为例，对于隧道模式与传输模式中的ESP协议，其所要准备的加密密钥基本上是一致的。因此对于密钥这部分，这里就不再多做说明了。

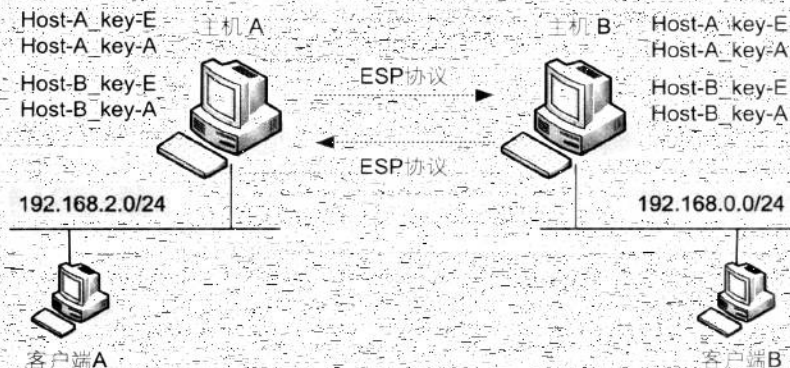


图14-25：隧道模式中的ESP协议

接下来分析客户端A与客户端B是如何通过ESP协议来跨过因特网传输数据。首先，当客户端A发送数据包给客户端B时，其数据包结构就如图14-26上半部分所示，但当这个数据包送到VPN主机A时，主机A随即在原数据包前面加上一个ESP包头，并在数据包末端加上ESP Trailer，然后使用Host-A_key-E密钥将原来的IP包头、TCP包头、DATA及ESP Trailer进行加密，接着生成一个新的IP包头。ESP协议的加密操作至此就完成了。

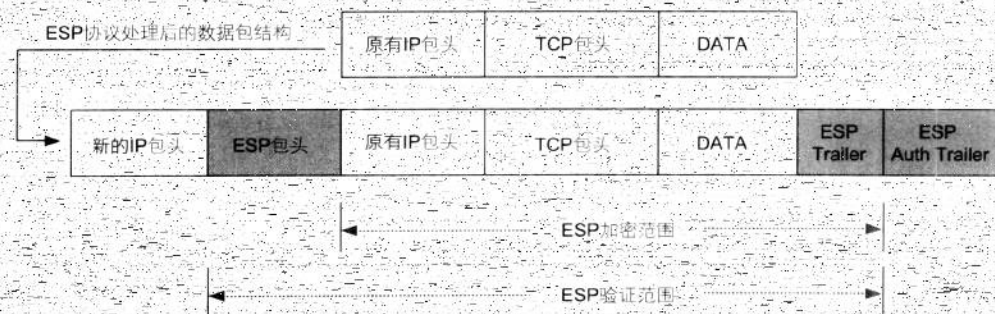


图14-26：隧道模式中ESP协议的数据包结构

如果我们要使用ESP协议中的完整性验证，系统会接着使用哈希算法对ESP包头、Host-A_key-E(原有IP包头、TCP包头、DATA及ESP Trailer)进行计算，计算完所得到的指纹

再使用Host-A_key-A进行加密,最后再把加密后的指纹附加到整个数据包的最末端,即ESP Auth Trailer字段中。

主机B收到数据包后,就可以通过ESP Auth Trailer的内容来验证数据包的完整性。如果验证结果没有问题,就把数据包内被加密的部分解开,并将解密后的数据包交给系统的路由表来决定数据包该如何传输,如此客户端B即可接到客户端A发送过来的数据包。

图14-27是ESP协议在隧道模式中的数据包结构,其实这个数据包与传输模式中的ESP数据包结构相同。但从图中可以看到,这个数据包的来源端是10.0.1.200,而目的端IP是10.0.1.210,并不是客户端A和客户端B的IP,同时也验证了图14-26的内容。

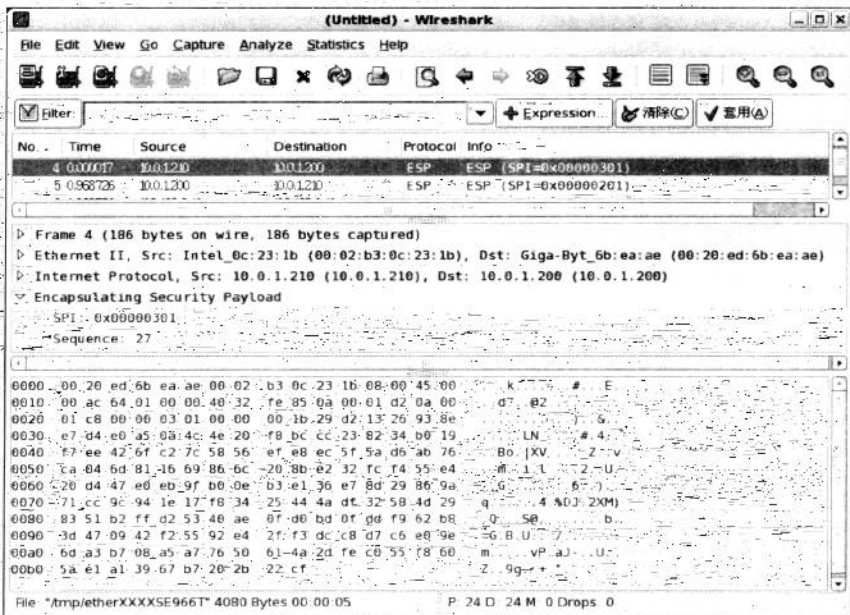


图14-27 隧道模式中ESP协议的数据包结构

在IPSec协议中,我们除了可以单独使用AH或ESP协议外,也可以同时使用它们,甚至还可以决定在IPSec的环境中,先使用AH还是先使用ESP协议。不过,在大多数情况下,都只会使用ESP协议,而不是用AH协议,因为在使用VPN机制时,通常都会进行加密处理。此外因为AH协议较少用,所以在RFC中已经有人提议废除IPSec协议中的AH协议。至于是否使用AH协议就有你自己来决定了。如果你的VPN服务器硬件够好,使用AH协议应该也不是什么大问题。根据我的经验,通常只使用ESP协议。

14.5.3 AH及ESP协议运行时需要设置的参数

在完整了解AH及ESP协议的工作流程及原理之后，接下来就可以试着把传输模式及隧道模式的IPSec VPN搭建起来。不过在开始搭建VPN之前，需要首先整理一下AH协议及ESP协议所需要的参数有哪些，以及用途是什么，如此才能将VPN顺利地搭建起来。

● AH协议所需参数：

- 工作模式：设置AH的工作模式是传输模式还是隧道模式。
- 目的端IP：设置VPN另一个端点的IP地址。
- 目的端端口：在IPSec中，我们可以设置哪些流量需要经过IPSec处理，哪些流量不需要经过IPSec处理，而流量则依据协议(TCP、UDP或ICMP)及端口来划分。
- 设置数据包传输方向：在IPSec环境中，我们需要设置数据包的方向，例如数据包是送入还是送出。
- 验证算法：AH协议的完整性验证是使用哈希算法来实现的，因此我们必须设置所要使用的验证算法是哪种。在AH协议中，验证算法的选择有hmac-md5及hmac-sha1两种。
- 加密密钥：验证算法计算出来的指纹需要加密，我们必须准备加密用的密钥，而加密密钥的长度会因验证算法而异，如表14-1所示。

表14-1 算法与所需密钥长度对照表

算法	密钥长度(位)	密钥长度(字节)
hmac-md5	128	16
hmac-sha1	160	20

● ESP协议所需参数：

- 工作模式：设置ESP的工作模式是传输模式还是隧道模式。
- 目的端IP：设置VPN另一个端点的IP位置。
- 目的端端口：在IPSec的使用中，我们可以设置哪些数据需要经过IPSec处理，哪些数据不需要经过IPSec处理，而流量则依据协议(TCP、UDP或ICMP)及端口来划分。
- 设置数据包传输的方向：在IPSec环境中，我们需要设置数据包的方向，例如数据包是送入还是送出。

- 验证算法: ESP协议的完整性验证是使用哈希算法来实现的, 因此我们必须设置所要使用的验证算法是哪种, 在ESP协议中验证算法的选择有hmac-md5及hmac-sha1两种。
- 完整性验证所需的加密密钥: 验证算法计算出来的指纹需要加密, 我们必须准备加密用的密钥, 而加密密钥的长度会因验证算法而异, 如表14-2所示。

表14-2 算法与所需密钥长度对照表

算法	密钥长度(位)	密钥长度(字节)
hmac-md5	128	16
hmac-sha1	160	20

- 加密算法: 在ESP协议的加密机制中, 我们可以选择DES及3DES两种。实际上, 在Linux平台所能选择的加密协议有很多, 若你感兴趣, 可以在Linux环境中, 使用man setkey命令来查询。但由于Windows系统及大多数的商用VPN系统只支持DES及3DES两种, 如果需要更安全的加密算法如aes-ctr, 则VPN系统的两个端点必须都是Linux才行。
- 加密算法所需的加密密钥: 加密算法的密钥长度与我们所选择的加密算法有关, 如表14-3所示。

表14-3 算法与所需密钥长度对照表

算法	密钥长度(位)	密钥长度(字节)
Null	-	-
des-cbc	64	8
3des-cbc	192	24
blowfish-cbc	400-448	-
cast128-cbc	40-128	-
rijndael-cbc	128/192/256	-
des-deriv	64	8
twofish-cbc	0-256	-
aes-ctr	160/224/288	-
camellia-cbc	128/192/256	-

14.5.4 安装IPSec参数的管理工具

如果要想IPSec可以正常工作，那么需要适当的管理工具来管理AH及ESP运行所需的参数，而这类管理工具在开源领域中从不缺乏，就看你习惯使用什么样的软件。我个人喜欢使用ipsec-tools这个软件，主要是因为ipsec-tools软件的配置文件比较直接而且不容易弄错，我们前面所提到的setkey就是包含在ipsec-tools软件中的工具。ipsec-tools的官方网站是<http://ipsec-tools.sourceforge.net>，不过官方网站只提供源代码，因此安装上比较不方便。另外RedHat Enterprise Linux 6.0及CentOS 6.0中并没有提供这个软件，不过我们可以轻松地在<http://rpm.pbone.net>网站上找到ipsec-tools软件，此处选用的是ipsec-tools-0.7.3-5.fc14这个版本，请你在下载后，使用以下命令将其安装起来即可。

```
yum localinstall ipsec-tools-0.7.3-5.fc14.i686.rpm
```

14.5.5 配置传输模式IPSec VPN

现在以图14-28为例来实际搭建传输模式的VPN。由于这是我们首次搭建IPSec VPN，因此，首先将示例尽量简化，从简单开始然后慢慢让它变得更完整，如此才能清晰地了解到IPSec VPN的配置，而这个示例的最终目的是完成AH协议、ESP协议(包含完整性验证及加密)。



图14-28 传输模式VPN

示例14.1 启用数据库服务器及客户端之间IPSec的AH协议

```
1. flush;
2. spdflush;
3. #=====<< SAD >>=====
4. add 192.168.0.1-192.168.0.10-ah 0x200 -m transport ...
5.   -A hmac-sha1 0x73f6e61bf4c8307020c230b367296e26a5262fb5;
6. add 192.168.0.10 192.168.0.1 ah 0x300 -m transport
7.   -A hmac-sha1 0x3f3f0cd71d0e300d5788127cc78db64ea3f21107;
```

```

8. #=====<< SPD >>=====
9. spdadd 192.168.0.1 192.168.0.10 any -P out ipsec
10.  ah/transport//require;
11.
12.spdadd 192.168.0.10 192.168.0.1 any -P in ipsec
13.  ah/transport//require;

```

以上就是一个只启用AH协议的IPSec VPN的配置文件, 请将这些内容保存到/root/192-168-0-1.conf文件, 代表这是数据库服务器上的配置文件。对于配置文件的内容, 在阅读时请特别注意两点。第一, 如果看到“#”符号, 就代表其后的所有字符串都是注释; 第二, 如果没有看到“;”符号, 则代表目前这一行与下一行对于系统而言是同一行。下面使用代码段来讨论配置文件的内容:

```

1. flush;
2. spdflush;

```

上面的设置是指在加载文件内所定义的IPSec参数之前, 先清除IPSec原有的参数内容, 而IPSec的参数内容分别保存在SAD(Security Association Database)及SPD(Security Policy Database)两个数据库中。其中, flush命令是指清除SAD数据库内容, spdflush则是指清除SPD数据库的内容, 至于什么是SAD, 什么又是SPD? 稍后将详细解释, 现阶段请你先暂时忽略。

```

3. #=====<< SAD >>=====
4. add 192.168.0.1 192.168.0.10 ah 0x200 -m transport
5.      -A hmac-sha1 0x73f6e61bf4c8307020c230b367296e26a5262fb5;
6. add 192.168.0.10 192.168.0.1 ah 0x300 -m transport
7.      -A hmac-sha1 0x3f3f0cd71d0e300d5788127cc78db64ea3f21107;

```

以上内容就是加入到SAD数据库的参数, 此外别忘了我们在第14.5.2一节中曾提过AH及ESP协议是单向的, 因此如果我们要进行双向的AH协议, 参数就得准备两份。因为第4行及第5行之间并没有“;”符号, 因此, 第4、5行是同一行, 第6、7行也是同一行, 下面是这些参数的含义:

- 第4行的含义:

- add是将其后的参数加入到SAD数据库的命令。
- 192.168.0.1 192.168.1.10则代表192.168.0.1传输数据给192.168.0.10。
- ah指这些参数是定义给AH协议来使用的。
- 0x200暂时不介绍, 后面的内容将会有详细说明。不过请先记住, 这个数值不能重复。

- `-m transport`表示使用传输模式，如果不设置默认就是传输模式，另外，如果要使用隧道模式，其参数就是`-m tunnel`。
- `-A hmac-sha1`代表我们选择的验证算法是hmac-sha1。
- `0x……`是加密密钥的设置，而密钥的设置方法有两种。以hmac-sha1算法为例，因为hmac-sha1算法所需结合使用的加密密钥长度是20字节，因此我们可以使用`-A hmac-sha1"12345678901234567890"`的方法来设置，只要"`"`"中的字符是20个字节(byte)就可以，至于内容是什么并不重要。另外也可以使用Linux系统中的命令来生成加密用的密钥，如下所示。

```
[root@localhost ~]# dd if=/dev/random count=20 bs=1 | xxd -ps
0bffdce8e51d9b39b1d200818882f75225d95952
20+0 records in
20+0 records out
20 bytes (20 B) copied, 0.000304166 秒, 65.8 kB/s
```

其中“0bffdce8e51d9b39b1d200818882f75225d95952”就是以随机方式生成的十六进制数，而命令中的`count=20`表示要生成160位($20 \times 8 = 160$ ，因为1字节=8位)的长度，如此我们就可以把这些数据作为加密密钥来使用，不过因为这是十六进制的数据，因此我们需要在密钥的最前面加上`0x`，以表示这是十六进制的数据。

● 第6行的含义：

第6行与第4行的参数是相同的含义，只不过“方向”不同。而且这些参数可以与第4行完全不同。例如在第4行中，我们选择hmac-sha1算法，但在第6行选用hmac-md5算法。不过，通常建议设置成较为安全的hmac-sha1算法。此外，关于加密密钥的部分最好不要相同，这样对整个加密的应用会更安全。

```
8. #=====<< SPD >>=====
9. spdadd 192.168.0.1 192.168.0.10 any -P out ipsec
10.      ah/transport//require;

11. spdadd 192.168.0.10 192.168.0.1 any -P in ipsec
12.      ah/transport//require;
```

以上就是加入SPD数据库的内容。这些参数用于控制进出VPN主机的数据包中哪些是需要使用IPSec协议，以及使用IPSec协议中的哪些协议，下面就来看看这些参数所代表的含义。

● 第9行的含义：

- `spdadd`是将其后参数加入SPD数据库的命令。

- “192.168.0.1 192.168.0.10 any -P out ipsec” 参数是设置需要使用IPSec的数据包有哪些，只不过这些参数较为复杂，下面是这些规则的定义方法：

```
192.168.0.1 [any] 192.168.0.10[110] tcp -P out ipsec
```

我们先假设上面的参数是在192.168.0.1这台主机上设置的，而这些参数的意思是说“如果192.168.0.1使用任何一个端口，发送数据包给192.168.0.10的110端口，且通信协议是TCP，那么这些数据包必须使用IPSec协议”。另外如果端口的部分没有定义，其默认值就是any，也就是指任何一个端口的意思。

```
192.168.0.1 [any] 192.168.0.10[53] udp -P out ipsec
```

在本例中，我们还是假设参数是在192.168.0.1这台主机上设置的，这些参数的意思是说，192.168.0.1主机如果通过192.168.0.10主机进行名称解析，那么这些流量都要经过IPSec处理。

```
192.168.0.1 192.168.0.10 icmp -P out ipsec
```

本例中的意思你应该可以很轻松了解，不过要特别说明一点，因为端口概念不适用于ICMP协议，因此在处理ICMP协议时，就不能设置端口，否则将造成命令的语法错误。

- 第10行的含义：

“ah/transport/require” 参数是指如果数据包流量符合第9行的定义，那么这些数据包是必须使用“传输模式”的AH协议来处理。

- 第11行及第12行的含义：

第11行还有12行其实只是第9行及第10行的反方向定义而已。

在定义了所有参数后，我们可以使用SCP工具将/root/192-168-0-1.conf文件传输到192.168.0.10主机的/root目录下，并且保存为192-168-0-10.conf文件，操作方法如下：

```
scp /root/192-168-0-1.conf 192.168.0.10:/root/192-168-0-10.conf
```

目前192-168-0-10.conf与192-168-0-1.conf内容是相同的，请思考一个问题，这两台主机上的IPSec参数定义完全相同，你觉得合理吗？还有哪些需要修改之处呢？其实这两个文件的内容除了SPD部分out及in的方向要互换之外，其他内容应该完全相同。下面列出这两个文件的内容。

```
1. flush;
2. spdflush;
3. #=====<< SAD >>=====
```

```

4. add 192.168.0.1 192.168.0.10 ah 0x200 -m transport
5.     -A hmac-shal 0x73f6e61bf4c8307020c230b367296e26a5262fb5;

6. add 192.168.0.10 192.168.0.1 ah 0x300 -m transport
7.     -A hmac-shal 0x3f3f0cd71d0e300d5788127cc78db64ea3f21107;
8. #=====<< SPD >>=====
9. spdadd 192.168.0.1 192.168.0.10 any -P out ipsec
10.     ah/transport//require;

11.spdadd 192.168.0.10 192.168.0.1 any -P in ipsec
12.     ah/transport//require;

```

192-168-0-1.conf文件内容

```

1. flush;
2. spdflush;
3. #=====<< SAD >>=====
4. add 192.168.0.1 192.168.0.10 ah 0x200 -m transport
5.     -A hmac-shal 0x73f6e61bf4c8307020c230b367296e26a5262fb5;

6. add 192.168.0.10 192.168.0.1 ah 0x300 -m transport
7.     -A hmac-shal 0x3f3f0cd71d0e300d5788127cc78db64ea3f21107;
8. #=====<< SPD >>=====
9. spdadd 192.168.0.1 192.168.0.10 any -P in ipsec
10.     ah/transport//require;
11.
12.spdadd 192.168.0.10 192.168.0.1 any -P out ipsec
13.     ah/transport//require;

```

192-168-0-10.conf文件内容

在设置完这两个文件后，接下来可以使用setkey命令来启动IPSec，操作方法如下：

```
setkey -f 192-168-0-1.conf
```

等到这两台主机上的IPSec都启动后，我们就可以在192.168.0.1主机上，使用任意协议与192.168.0.10主机通信。如ping 192.168.0.10，因为在本例中我们是对any(任意协议)流量进行IPSec处理，所以如果使用WireShark来截取网络数据包，数据包的内容应该如图14-29所示，从图中可以看到192.168.0.1与192.168.0.10两台主机正在使用ICMP协议通信，并从数据包的结构可以看到，在IP包头与ICMP包头之间多了一个AH包头。这样即可证明我们的实验是成功的，最后可以使用以下命令来取消IPSec的运行。

```
setkey -D -F
setkey -P -F
```

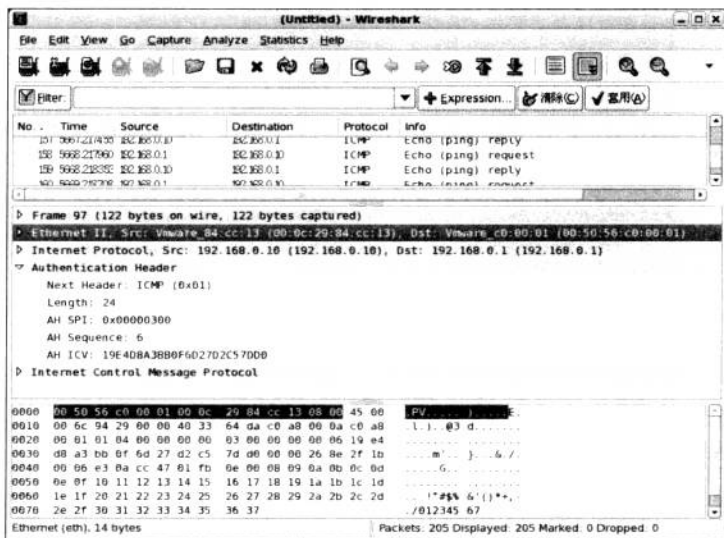


图14-29 实际截取AH协议中的数据

示例14.2 启用数据库服务器及客户端之间IPSec的ESP协议

```

1. flush;
2. spdflush;

3. #=====<< SAD >>=====
4. add 192.168.0.1 192.168.0.10 esp 0x201 -m transport
5.     -E 3des-cbc 0xb99d4139d9976665eede3f74d4e897617c5a7452e5789f9f
6.     -A hmac-shal 0x3f3f0cd71d0e300d5788127cc78db64ea3f21107;

7. add 192.168.0.10 192.168.0.1 esp 0x301 -m transport
8.     -E 3des-cbc 0xbb025b18e28ea7fb15479ba25346e24fd9ac1e7cd502a25
9.     -A hmac-shal 0x73f6e61bf4c8307020c230b367296e26a5262fb5;
10. #=====<< SPD >>=====
11. spdadd 192.168.0.1 192.168.0.10 icmp -P out ipsec
12.     esp/transport//require;

13. spdadd 192.168.0.10 192.168.0.1 icmp -P in ipsec
14.     esp/transport//require;

```

在了解AH协议的设置之后，以上的内容应该就很容易理解了，其实这些参数与AH协议参数的设置上大致相同，下面来解释这个配置文件的内容：

- 第5行指定ESP协议中所要使用的加密算法及加密密钥，密钥的生成方法可以与AH协议中产生密钥的方法相同，但要注意密钥的长度。

- 第6行指定ESP协议中完整性验证要使用的验证算法及加密密钥，如果省略这一行则表示不启用ESP完整性验证。
- 第8行及第9行指定反方向ESP协议的参数。
- 第12行及第14行指定启用ESP协议中的传输模式。

在两台主机都启用IPSec机制后，可以截取到结构如图14-30的数据包，从图中已经无法看到数据包内实际传输的数据内容，因为所有数据都被ESP协议加密了，所以可以从数据包的结构部分看到，在IP包头之后就只剩下ESP包头及被加密的数据，如此即可证明我们的实验是成功的。

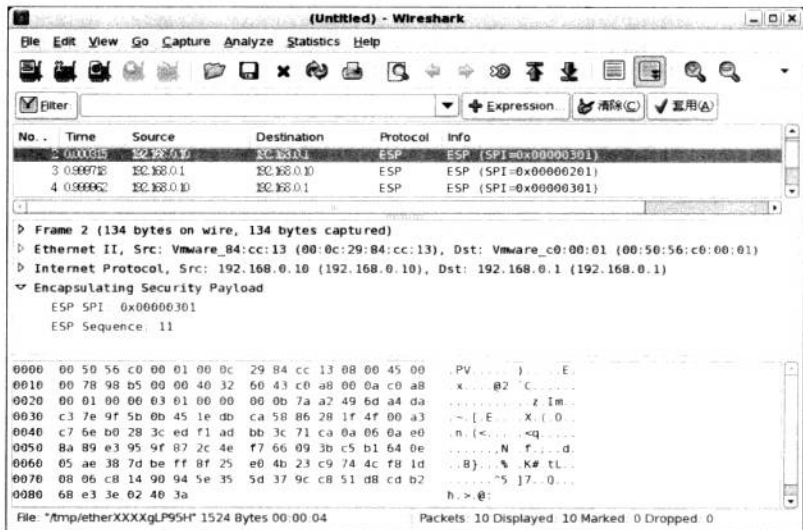


图14-30 实际获取ESP协议中的数据

示例14.3 启用据库服务器与客户端之间IPSec的AH及ESP协议

```
1. flush;
2. spdflush;

3. #=====<< SAD >>=====
4. add 192.168.0.1 192.168.0.10 ah 0x200 -m transport
5.   -A hmac-sha1 0x73f6e61bf4c8307020c230b367296e26a5262fb5;

6. add 192.168.0.10 192.168.0.1 ah 0x300 -m transport
7.   -A hmac-md5 0x570e55a9560bf191e38398979aa6e967;

8. add 192.168.0.1 192.168.0.10 esp 0x201 -m transport
```

```
9.      -E 3des-cbc 0xb99d4139d9976665eede3f74d4e897617c5a7452e5789f9f;  
  
10.add 192.168.0.10 192.168.0.1 esp 0x301 -m transport  
11.      -E des-cbc 0xcce630577afbaa83;  
12.#=====<< SPD >>=====
```

```
13.spdadd 192.168.0.1 192.168.0.10 icmp -P out ipsec  
14.      esp/transport//require  
15.      ah/transport//require;  
  
16.spdadd 192.168.0.10 192.168.0.1 icmp -P in ipsec  
17.      esp/transport//require  
18.      ah/transport//require;
```

经过示例14.1及示例14.2的解释之后,相信示例14.3的内容你一看就可以理解,其实这只是把前两个示例的内容加在一起而已。不过为了加深你对AH及ESP协议的单向性了解,本示例特意把双向验证算法及加密算法设为不一样,当然加密密钥也需要随之改变。

此外,差别较大的部分大概是在第14、15、17及18行,不过这样的写法相信你都应该能接受,其实也就是表明要同时使用AH及ESP协议。不过在AH及ESP协议都需要的情况下,是AH先执行,还是ESP先执行呢?其实都可以,只要两端的VPN主机设置成一样就可以被接受。在默认情况下,Windows系统先执行ESP,再执行AH。而且目前示例中的设置是先执行ESP,然后执行AH协议,因此我们实际截取到的数据包如图14-31所示,可以清楚看到ESP包头在AH包头之后。

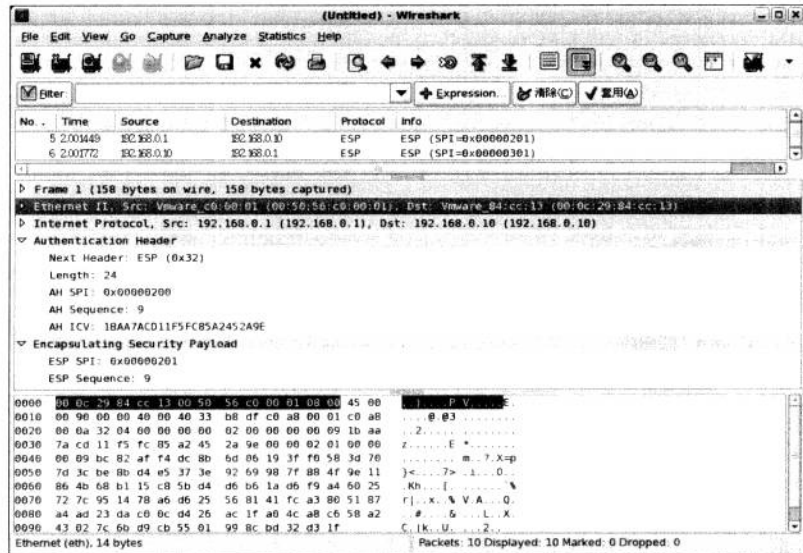


图14-31 先ESP后AH的数据包结构

但如果按以下方式修改示例14-3的SPD部分,则首先执行AH协议,然后才执行ESP协议,如此我们在网络上实际获取到的数据包结构就如图14-32所示。因为ESP协议是在后面才执行的,且开始执行时生成的AH包头也会被加密,所以我们只能在图中看到ESP包头。

```
12. #=====<< SPD >>=====
13.spdadd 192.168.0.1 192.168.0.10 icmp -P out ipsec
14.     ah/transport//require
15.     esp/transport//require;

16.spdadd 192.168.0.10 192.168.0.1 icmp -P in ipsec
17.     ah/transport//require
18.     esp/transport//require;
```

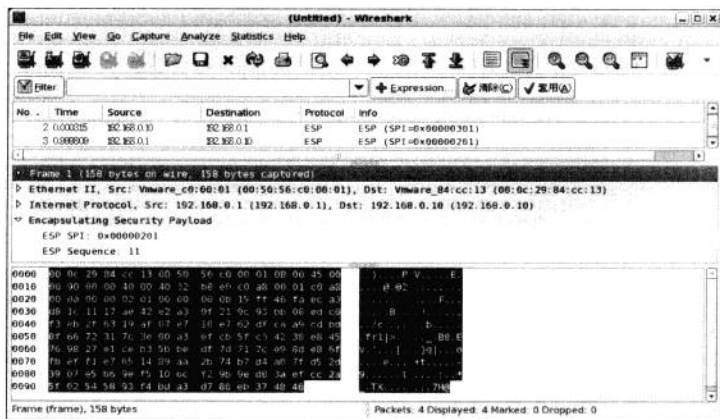


图14-32 先AH后ESP的数据包结构

以上的示例都使用传输模式。接下来以图14-33为例来搭建隧道模式的IPSec VPN,并同时启动AH及ESP协议。

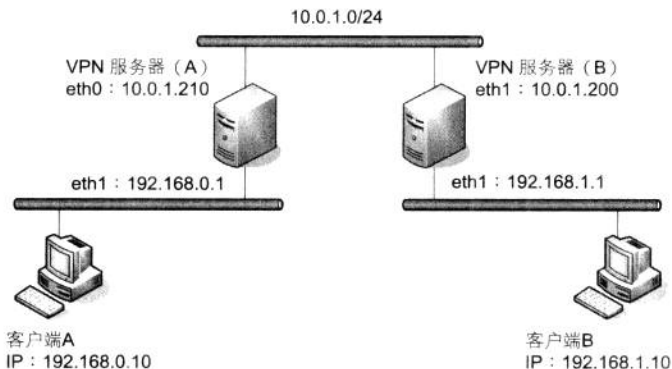


图14-33 隧道模式VPN

```

1. flush;
2. spdflush;

3. #=====<< SAD >>=====
4. add 10.0.1.200 10.0.1.210 ah 0x300 -m tunnel
5.     -A hmac-shal 0x3f3f0cd71d0e300d5788127cc78db64ea3f21107;
6. add 10.0.1.210 10.0.1.200 ah 0x200 -m tunnel
7.     -A hmac-shal 0x73f6e61bf4c8307020c230b367296e26a5262fb5;
8. add 10.0.1.200 10.0.1.210 esp 0x201 -m tunnel
9.     -E 3des-cbc 0xb99d4139d9976665eede3f74d4e897617c5a7452e5789f9f;

10.add 10.0.1.210 10.0.1.200 esp 0x301 -m tunnel
11.     -E 3des-cbc 0xbb025b18e28ea7fb15479ba25346e24fd9acbe7cd502a25;

12.#=====<< SPD >>=====
13.spdadd 192.168.0.0/24 192.168.1.0/24 any -P out ipsec
14.     esp/tunnel/10.0.1.210-10.0.1.200/require
15.     ah/tunnel/10.0.1.210-10.0.1.200/require;

16.spdadd 192.168.1.0/24 192.168.0.0/24 any -P in ipsec
17.     esp/tunnel/10.0.1.200-10.0.1.210/require
18.     ah/tunnel/10.0.1.200-10.0.1.210/require;

```

以上是启用隧道模式VPN的设置参数，其参数的定义与传输模式的参数几乎相同，而其中差别较大的是SPD的部分。这里只介绍SPD的部分，下面先以第13、14及15行为例来解释：

- 第13行定义凡从192.168.0.0/24网段送给192.168.1.0/24网段的任意数据包，都需要通过IPSec来处理。
- 第14行定义执行ESP协议的隧道模式，而隧道起点是10.0.1.210，隧道终点是10.0.1.200。
- 第15行定义执行AH协议的隧道模式，而隧道起点是10.0.1.210，隧道终点是10.0.1.200。

对于隧道模式的IPSec VPN而言，其AH及ESP仍然是单向的，如果我们希望双向流量都使用IPSec协议，那就必须定义另一个反方向的规则，其中第16、17及18行就是反方向的IPSec VPN定义。

14.6 Linux中的IPSec架构

与Netfilter一样，Linux也通过加载外部模块来支持IPSec机制，而这些模块的位置如图14-34所示。其中ah4.ko及esp4.ko就是Linux在IPV4环境下支持IPSec的模块，不过ah4.ko及esp4.ko其实只是执行AH及ESP协议的一个机制而已，如果我们不提供验证算法、加密算法及

加密密钥等, ah4.ko、esp4.ko就只能是一个框架而已, 除了占用一点内存之外, 什么事情都做不了。那么如何指定AH及ESP的工作参数呢? 这就得使用setkey管理工具将IPSec运行时所需要的参数写入到SPD(Security Policy Database, 安全策略数据库)及SAD(Security Association Database, 安全参数数据库)。

```
[root@desktopl ~]# ls /lib/modules/2.6.32-71.el6.x86_64/kernel/net/ipv4/
ah4.ko      ipip.ko      tcp_htcp.ko  tcp_vegas.ko  xfrm4_mode_beet.ko
esp4.ko     netfilter    tcp_hybla.ko  tcp_veno.ko   xfrm4_mode_transport.ko
inet_diag.ko tcp_bic.ko    tcp_illinois.ko  tcp_westwood.ko  xfrm4_mode_tunnel.ko
ipcomp.ko   tcp_diag.ko  tcp_lp.ko     tcp_yeah.ko    xfrm4_tunnel.ko
ip_gre.ko   tcp_highspeed.ko  tcp_scalable.ko  tunnel4.ko
```

图14-34 Linux中的AH及ESP扩充模块

回想一下, 我们在启动IPSec VPN时, 是不是运行了setkey -f 192-168-0-1.conf的操作, 而192-168-0-10.conf文件内只有====<SPD>====及====<SAD>====两个段落, 这两个段落的参数就是分别要写入到SPD及SAD的内容, 待参数写入到SPD及SAD之后, IPSec机制就可以按照SPD及SAD中的参数来执行任务; 相反, 如果我们使用setkey管理工具将SPD及SAD中的参数删除, IPSec就会停止工作。

14.6.1 IPSec机制的SPD

SPD的内容是用来存放IPSec的规则, 而这些规则用来定义哪些流量需要使用IPSec。这个数据库的内容相当多, 这里仅介绍我们需要了解的部分, 这些数据如: 目的端IP、来源端IP、只执行AH或ESP、同时执行AH及ESP、目的端端口、来源端端口、使用传输模式或隧道模式。图14-35就是SPD的结构, 从结构内容我们可以看到, 第一条及第二条数据刚好构成一条双向VPN连接, 另外, 由于一台主机可能同时与多台主机进行IPSec连接, 因此数据库的内容可能同时存在多条数据。

来源端IP	目的端IP	执行协议	来源端口	目的端口	工作模式
192.168.0.1	192.68.0.10	AH/ESP	Any	110	Transport
192.168.0.10	192.168.0.1	AH/ESP	110	Any	Transport
192.168.0.1	192.168.0.20	ESP	Any	1433	Transport
192.68.0.20	192.168.0.1	ESP	1433	Any	Transport

图14-35 SPD数据库结构

有了以上信息, 当VPN主机有数据包要发送出去时, 这个数据包由SPD来进行匹配; 如果数据包的来源端及目的端IP不等于SPD所记录的内容, 那么这个数据包就不会交给AH或ESP协议来处理; 反之数据包就会被送给AH或ESP协议来处理, 至于会送给谁来处理, 就由

SPD的“执行协议”这个字段的内容来定。

我们可以通过setkey工具来管理SPD的内容,例如可以使用setkey -D -P来检查SPD的内容,也可以使用setkey -P -F来清除SPD的内容。

14.6.2 IPSec机制的SAD

保存在SAD数据库中的参数有SPI值、目的端IP、AH或ESP、AH验证算法、AH验证的加密密钥、ESP验证算法、ESP验证算法的加密密钥、ESP的加密算法、ESP加密算法的加密密钥、使用传输模式还是隧道模式;其中SPI(Security Parameter Index,索引值)是两台VPN主机之间使用随机数设置的唯一值或者手动设置的唯一值,其目的是要作为数据库的索引值,在整个IPSec工作中没有其他用途。此外,需要注意AH与ESP协议的参数是分开的,因此可以将SAD分为两个部分,并用执行协议来区分。图14-36显示了SAD数据库结构的一部分。

SPI	目的端IP	执行协议	AH验证算法	AH加密密钥	工作模式
0x200	192.168.0.10	AH	hmac-sha1	xxxxxxxxxx	Transport
0x300	192.168.0.1	AH	hmac-sha1	xxxxxxxxxx	Transport

图14-36 SAD数据库结构(一)

在SPD中决定数据包必须执行AH、ESP或AH及ESP协议后,就会从SAD数据库中找到处理这个数据包的参数。例如,我们是192.168.0.1这台主机,当SPD决定192.168.0.1送到192.168.0.10的TCP端口110的数据包需要执行AH及ESP协议,在数据包交给ESP机制之后,ESP机制会根据目的端IP来找到处理这个数据包的参数,如图14-37中的第一条数据。此外,ESP机制在处理过程中会将SPI值加入ESP包头中,就如图14-31中ESP包头内中SPI值,稍后将全面介绍这个SPI值的用途。

SPI	目的IP	执行协议	ESP验证算法	ESP验证算法的加密密钥	ESP的加密算法	ESP的加密密钥	工作模式
0x201	192.168.0.10	ESP	na	na	3des-cbc	kkkkkkk	Transport
0x301	192.168.0.1	ESP	na	na	3des-cbc	jjjjjjjjj	Transport
0x401	192.168.0.20	ESP	hmac-sha1	rrrrrrrr	3des-cbc	iiiiiiiiii	Transport
0x501	192.168.0.1	ESP	hmac-md5	tttttttttt	des-cbc	ggggggggg	Transport

图14-37 SAD数据库结构(二)

ESP协议处理完毕后，接着将数据包交给AH机制处理，AH机制从图14-36中找到目的端IP是192.68.0.10的哪一条数据，并从数据中找到处理这个数据包的参数。此外，在AH处理过程中，也会将SPI值一起放到AH包头中，也就是如图14-31中AH包头内所看到的SPI值，稍后将完整介绍SPI值的用途。

我们同样可以使用setkey工具来管理SAD的内容，例如，可以使用setkey -D来检查SAD的内容，或是使用setkey -D -F来清除。

接下来以图14-38为例来说明IPSec的工作流程。假设主机A要发送数据给主机B，当数据送到网络层的下端时①，IPSec的过滤器会将数据包的特征与SPD数据库的内容匹配②，如果数据包特征与SPD数据库的内容都不符合，这个数据包就会直接通过网络传输给主机B③，在这个数据包进入到主机B之后，主机B的IPSec过滤器会将这个数据包的特征与其SPD数据库的内容匹配④，如果匹配的结果不符合，那么数据包就直接送往上层⑤。

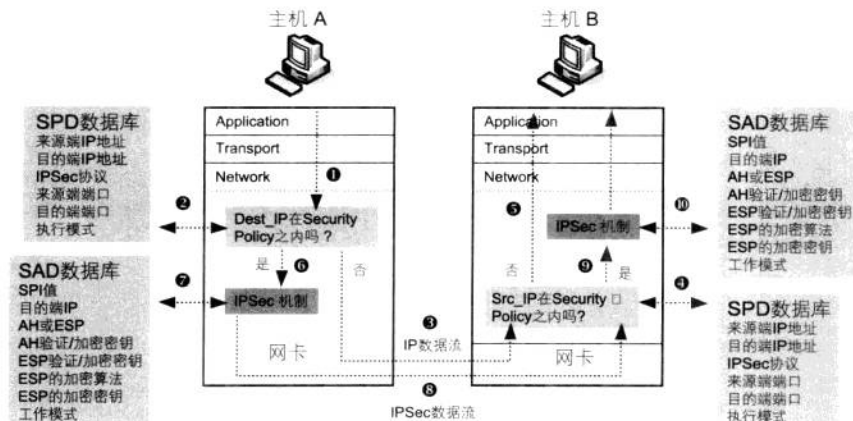


图14-38 IPsec的工作流程

但如果主机A送给主机B的数据包特征符合主机A上SPD数据库内容，数据包就会被送到IPsec的AH及ESP机制中⑥，接着AH、ESP就会到SAD数据库中找到处理这个数据包的参数⑦。处理完毕后的数据包随即被送往主机B⑧，在主机B找到这个数据包之后，就把这个数据包的特征与其SPD数据库的内容进行匹配④。如果匹配的结果符合，这个数据包就会送入AH及ESP机制处理⑨，接着AH、ESP就会从SAD数据库中找到处理这个数据包的参数⑩，最后将处理完毕的数据送往上层。看完以上流程后，相信你应该进一步了解了IPsec的工作流程。

最后来看看AH及ESP包头中的SPI值的用途。试想对于192.168.0.10这台VPN主机，当从网络上收到一个IPsec处理后的数据包，请问要如何去验证这个数据包的完整性，及对这个数据包进行解密呢？首先是找到AH及ESP的参数，因为有这些参数我们才能知道数据包是以

哪种验证算法来计算的,并了解AH的加密密钥、ESP的加密算法及ESP的加密密钥都是什么。

由于在启动IPSec VPN时,会将AH及ESP的参数分别写入到两台VPN主机的SPD及SAD数据库中,因此处理这个数据包的AH及ESP参数在本机的SAD数据库中一定可以找到,但问题是如何查找?从图14-39可以看到,这个数据包的来源端IP是192.168.0.1,目的端IP是192.168.0.10,而且在数据包中AH及ESP包头的部分可以分别看到一个SPI参数,这个例子是AH: 0x200、ESP: 0x201。这样当我们收到这个数据包后,就可以根据目前使用的协议AH、数据包的目的端IP及数据包内AH包头内的SPI值,来找到SAD中的一条数据,而这条数据一定会是唯一的,如此一来,就可以取得处理这个数据包的AH及ESP参数。但这个SPI值从何而来呢?生成SPI值的方法有两种,其一是由系统自动生成;其二是我们自己手动设置的,回顾一下IPSec的配置文件内容,如下第4、6、8、10等四行中的0x200字段就是手动设置SPI的位置。

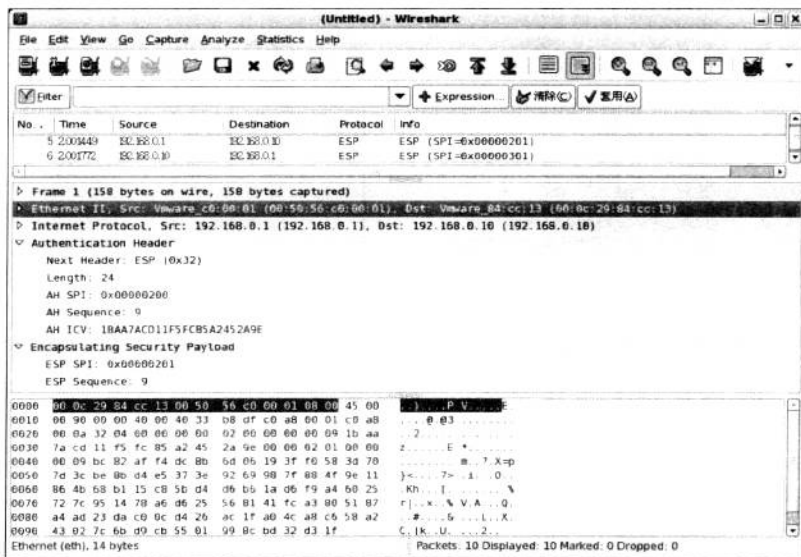


图14-39 IPSec的数据包结构

```
1. flush;
2. spdflush;

3. #=====<< SAD>>=====
4. add 10.0.1.200 10.0.1.210 ah 0x300 -m tunnel
5.   -A hmac-sha1 0xf3f0cd71d0e300d5788127cc78db64ea3f2107f
6. add 10.0.1.210 10.0.1.200 ah 0x200 -m tunnel
7.   -A hmac-sha1 0x73f6e61bf4c8307020c230b367296e26a5262fb5;
```

```

8. add 10.0.1.200 10.0.1.210 esp 0x201 -m tunnel
9.      -E 3des-cbc 0xb99d4139d9976665eede3f74d4e897617c5a7452e5789f9f;
10.add 10.0.1.210 10.0.1.200 esp 0x301 -m tunnel
11.      -E 3des-cbc 0xbb025b18e28ea7fb15479ba25346e24fd9acble7cd502a25;

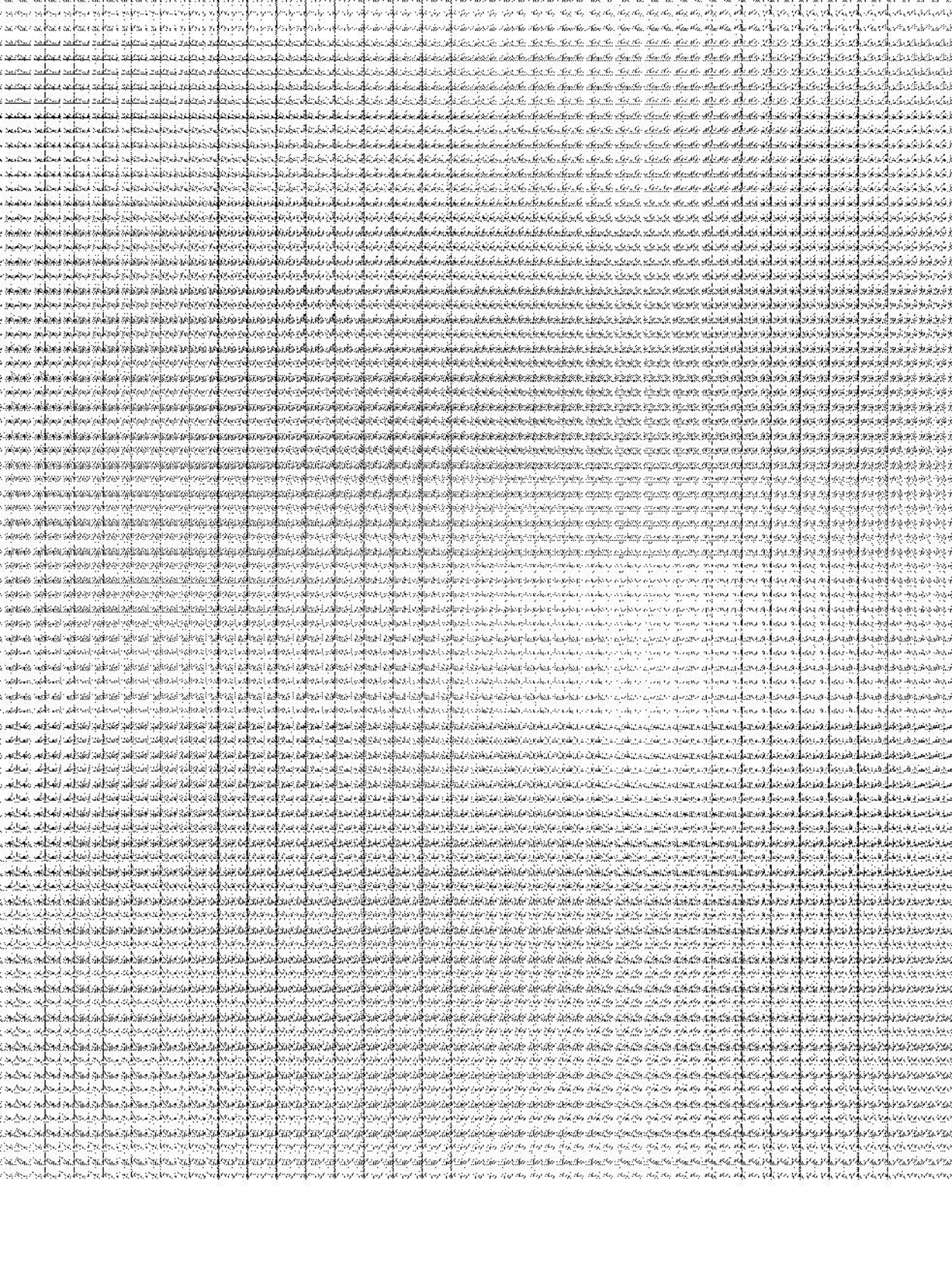
12.#=====<< SPD >>=====
13.spdadd 192.168.0.0/24 192.168.1.0/24 any -P out ipsec
14.      esp/tunnel/10.0.1.210-10.0.1.200/require
15.      ah/tunnel/10.0.1.210-10.0.1.200/require;

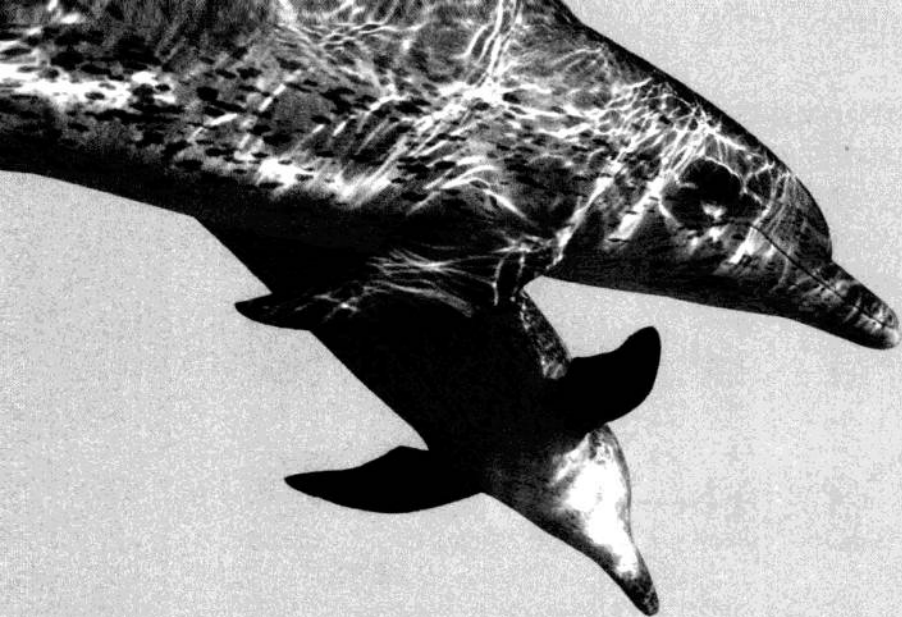
16.spdadd 192.168.1.0/24 192.168.0.0/24 any -P in ipsec
17.      esp/tunnel/10.0.1.200-10.0.1.210/require
18.      ah/tunnel/10.0.1.200-10.0.1.210/require;

```

14.7 小结

阅读完本章之后，相信你对于网络安全有了更深入的了解，不过由于VPN这个主题实在太宽泛了，无法用一章篇幅来说明，因此本章仅介绍VPN的入门知识而已，接下来的两章将继续深入讲解VPN知识。





Linux

|第15章| VPN实战篇

15

第14章中已经介绍过如何配置IPSec VPN，但你一定会觉得很麻烦：“怎么会有这么多加密密钥！”，其实在第8章的示例中，这些密钥还算少的，在实际应用中，一家企业拥有四五台VPN服务器是司空见惯的，以我所属的公司为例就有多达五台的VPN服务器，仅是生成及交换这么密钥就让人头疼不已。更重要的是，交换完成后绝对不能出错，否则整个企业的VPN系统可能就无法建立起来。这里以图15-1为例来说明密钥数量有多么惊人。

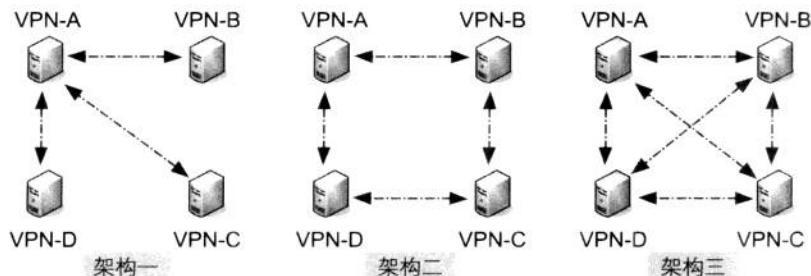


图15-1 VPN结构规划

在图15-1中共有三个VPN结构，如果可以让我们来选择，我想你可能会选择架构三，因为在架构三的环境中，每两个站点之间的路由距离都是最短的，并且可以进行多路径的选择，不会因为其中一台VPN服务器宕机，而导致其他站点的VPN连接中断。不过在这个架构中，因为每台VPN服务器都与其他三台VPN服务器相连，所以每台VPN服务器都要对外建立三条VPN连接。如果我们只启用ESP协议，那么VPN服务器上的密钥除了自身以外，还需要有另外三台VPN服务器的密钥，如此一台VPN服务器上就要有4个密钥，如果再把AH协议也启动起来，那么每台VPN服务器将多达8个密钥，试问你还能记清楚哪个密钥是给哪一台VPN服务器使用的吗？是给AH还是ESP协议使用？

另外我们还需要了解，对称式加密所使用的加密密钥如果长时间使用而不更换，那么当攻击者收集足够多由这个密钥加密的数据包之后，就可以通过这些数据包计算出加密密钥。如果攻击者真的计算出加密密钥，那加密系统就等于土崩瓦解了。第14章所介绍的VPN搭建方式就属于固定加密密钥的VPN系统。为了使VPN系统的加密机制更安全，就必须不断更改加密密钥，因此我们必须找到一种方便、安全而且可以不断自动更改密钥的机制，如此才能让VPN系统更加安全。

15.1 IKE

为了解决固定密钥模式中，VPN系统不安全的问题，因特网工程任务小组(Internet

Engineer Task Force: IETF)设计了Internet Key Exchange(因特网密钥交换协议;IKE)。在IKE架构中,系统管理人员只需设置加解密算法以及验证算法等信息,IKE就会定时地自动生成加密密钥,而以上两部分加起来就是SA,也就是要搭建一条单向VPN所需的参数。但如果要建立双向VPN连接就至少需要两个SA,而保存SA的地方就是在上一章中所提到的SAD,等到两个端点的VPN主机需要建立VPN连接时,IKE机制就会自动生成一组SA,并与另一个端点的VPN主机交换SA。如此一来,我们就不必再关心密钥的生成及交换等问题了。

以图15-2为例来解释IKE在IPSec环境中扮演的角色及其工作流程。首先,我们必须先定义好IKE及安全策略(Security Policy)等参数。此时SPD中应该有数据存在,而SAD的内容应该是空的。另外我们假设主机A上应用程序要与主机B上的应用程序通信时①,这个数据包会被送入到IPSec安全策略来进行匹配,以决定这个数据包是否需要经过IPSec协议来处理。如果不需要,则数据包被送入到IP层②,接着通过网络实体层将数据包发送到主机B④。但如果IPSec安全策略判断数据包需要经过IPSec机制来处理,数据包就会被送到IPSec机制中加以处理③,但此时因为SAD的内容是空的,所以IPSec机制会因为没有SA参数而不知道该如何处理这个数据包,因此IPSec机制将会触发IKE机制⑤,使得主机A的IKE机制与主机B的IKE机制进行数据交换⑦,进而产生两台主机SAD的内容⑥⑧,因此在这个过程中,应用程序将暂时出现无法通信的情形。

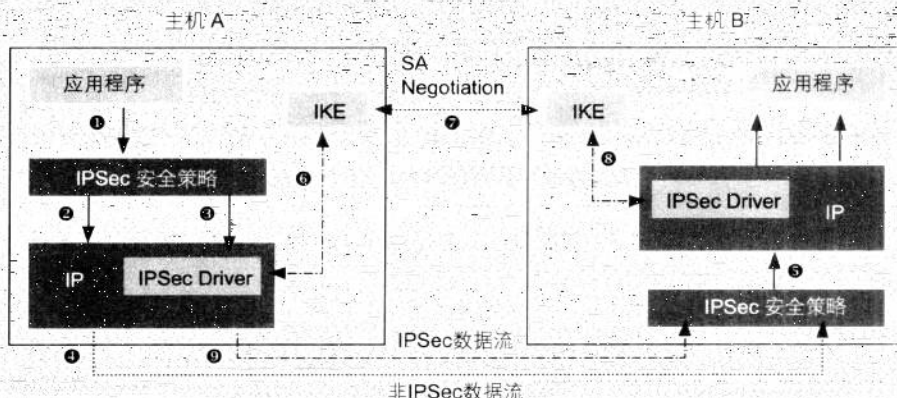


图15-2 IKE工作流程

如下面示例在第一次运行Ping命令时,其应答信息是“connect: Resource temporarily unavailable”,但在第二次运行Ping命令时,网络即可正常通信。另外我们以图15-3来佐证上述的情况,从图中可以清楚看到第12个及第13个数据包是ESP数据包,不过在开始通信之

前, 10.0.1.200及10.0.1.210这两台主机必须先经过第1~11个数据包的过程, 而这个过程就是IKE的交互过程。

```
[root@localhost ~]# ping 10.0.1.210
connect: Resource temporarily unavailable.
[root@localhost ~]# racoonl# ping 10.0.1.210
PING 192.168.0.10 (10.0.1.210): 56(84) bytes of data:
64 bytes from 10.0.1.210: icmp_seq=1 ttl=64 time=0.401 ms
64 bytes from 10.0.1.210: icmp_seq=2 ttl=64 time=0.296 ms
64 bytes from 10.0.1.210: icmp_seq=3 ttl=64 time=0.298 ms
```

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.1.200	10.0.1.210	ISAKMP	Identity Protection (Main Mode)
2	0.014470	10.0.1.210	10.0.1.200	ISAKMP	Identity Protection (Main Mode)
3	0.029353	10.0.1.200	10.0.1.210	ISAKMP	Identity Protection (Main Mode)
4	0.043128	10.0.1.210	10.0.1.200	ISAKMP	Identity Protection (Main Mode)
5	0.050192	10.0.1.200	10.0.1.210	ISAKMP	Identity Protection (Main Mode)
6	0.051126	10.0.1.210	10.0.1.200	ISAKMP	Identity Protection (Main Mode)
7	0.051494	10.0.1.210	10.0.1.200	ISAKMP	Informational
8	0.052134	10.0.1.200	10.0.1.210	ISAKMP	Informational
9	1.057292	10.0.1.200	10.0.1.210	ISAKMP	Quick Mode
10	1.062903	10.0.1.210	10.0.1.200	ISAKMP	Quick Mode
11	1.067335	10.0.1.200	10.0.1.210	ISAKMP	Quick Mode
12	1.247051	10.0.1.200	10.0.1.210	ESP	ESP (SPI=0x00586995)
13	1.247940	10.0.1.210	10.0.1.200	ESP	ESP (SPI=0x01c331f4)

Frame 15 (134 bytes on wire (134 bytes captured))

Ethernet II, Src: Vmware_2d:ea:9c (00:0c:29:2d:ea:9c), Dst: Vmware_99:2c:2f (08:00:06:99:2c:2f)

Internet Protocol, Src: 10.0.1.210 (10.0.1.210), Dst: 10.0.1.200 (10.0.1.200)

Encapsulating Security Payload

图15-3 IKE交互过程的数据包情况

IKE是一个协议, 而提供IKE机制的软件非常多。我个人习惯使用ipsec-tools。ipsec-tools软件中包含两个很重要的工具, 分别是setkey和racoon。setkey是SAD和SPD数据库的管理工具, racoon则是IKE机制。在了解这些关系后, 接下来将讨论IKE的交互过程, 这个过程共分为两个阶段: Main Mode和Quick Mode, 下面来分析这两个阶段到底都做了哪些事情。

图15-4是一个典型的racoon配置文件, 首先来看第14~19行, 这部分指定IPSec中AH及ESP协议的运行参数, 下面介绍这些参数的含义。

```
1. path include "/etc/racoon";
2. path pre_shared_key "/etc/racoon/psk.txt";
3. path certificate "/etc/racoon/certs";

4. remote 192.168.0.10 {
5.     exchange_mode main;

6.     proposal {
7.         authentication_method pre_shared_key;
8.         dh_group modp1024;
```

```

9:             hash_algorithm          sha1;
10:             encryption_algorithm     3des;
11:             lifetime_time             1 hour;
12:
13:
14: sainfo:anonymous {
15:             encryption_algorithm     3des;
16:             authentication_algorithm  hmac_sha1;
17:             lifetime_time             30 minute;
18:             compression_algorithm    deflate;
19: }

```

图15-4 一个典型的racoon配置文件

● encryption_algorithm:

指定ESP协议中所要使用的加密算法，这里可供选择的有des、3des、blowfish、cast128、rijndael等。

● authentication_algorithm:

指定AH及ESP协议中所要使用的验证算法，这里可供选择的有md5和sha1两种算法。

● lifetime_time:

指定AH及ESP协议中所有加解密密钥的淘汰时间。这个值设置得越短，VPN连接就越安全，但外一方面，VPN主机的负载就越重，笔者建议这个值是30~60分钟即可。

● compression_algorithm:

指定IPCOMP(IP Payload Compression)的压缩算法，这项功能可以缩短数据传输时间。

● AH及ESP的加解密密钥:

关于加解密密钥，我们并不需要特别去设置，因为IKE将会根据IPSec Driver所要运行的任务来生成对应的密钥。例如，我们在SPD中定义只需要运行ESP协议时，IKE会自动生成两个加密密钥，分别是ESP协议中的数据加密密钥，以及ESP协议中的验证机制所需要的加密密钥；但如果我们在SPD中定义AH及ESP协议都需要使用时，IKE将自动生成三个密钥。

在两台VPN主机上设置完以上参数之后，我们必须让两台主机知道对方的IPSec参数，这样才可以正常建立VPN连接，而这些参数就是在Quick Mode阶段中所要传输交换的数据内容，但这些重要的参数如果没有在一个很好的安全环境下进行交换，势必产生无法预料的后果。因此IKE在执行Quick Mode之前，会先执行Main Mode，其最主要目的有两个，其一是验证VPN主机的另一个端点是否可信，其二则是搭建一个安全的通信环境。图15-4的第6~13

行配置Main Mode的参数, 下面是这些参数的含义。

- authentication_method pre_shared_key

前面提到过, Main Mode的第一个任务是验证VPN的另一个端点是否可信。验证的方法有以下两种。

- pre_shared_key(Preshared Keys):

使用Preshared Keys验证方法来搭建VPN系统时, 速度较快而且简单。方法是以一个“字符串”作为验证依据, 也就是说如果对方的字符串与我方的字符串相等, 就认为对方是可信的。但由于字符串是固定的, 这种验证方法不太安全。

- rsasig(X.509 Certificates):

将证书作为验证的依据, 其优点是比较安全, 因为证书是一种无法被伪造的数字信息。但这个方法的问题在于不够方便, 因为我们还必须建立企业的证书中心。虽然如此, 还是比较建议使用这种方法, 毕竟搭建VPN的目的不就是要建立一个安全的通信环境吗?

- dh_group

在Main Mode阶段会使用Diffie-Hellman算法, 安全地在两台VPN主机间交换一个共用的加解密密钥。而在这个过程中, 我们可以决定其交换密钥的安全等级, 分别是modp768、modp1024及modp1536, 另外也可以使用1、2及3来分别代表前三项参数, 其中modp768是较低的安全等级, 而modp1536是最高的等级, 不过通常使用modp1024就足够了。

- hash_algorithm

配置Quick Mode阶段中数据传输的验证算法, 可以选择如md5、sha1两种算法。

- encryption_algorithm

配置Quick Mode阶段中数据传输的加密算法, 这里可以选择如des、3des、blowfish及cast128等。

- lifetime time

由于在图15-4第17行中定义的AH及ESP协议的加解密密钥每30分钟要重新生成及交换一次, 因此每30分钟就会执行一次Quick Mode。但Quick Mode的安全性是由Main Mode来支持的, 因此如果Main Mode中的加密密钥长时间不改变的话, 那么在某种程度上就会降低安全性。在图15-4的第6行到第13行Main Mode的参数配置中, 我们同样可以指定在Main Mode中加密密钥多久需要更换一次, 如图15-4的第11行就指定了加密密钥的生存时间。

在了解了Main Mode及Quick Mode的任务及配置方式后，接下来说明图15-4的其他相关参数。

- remote

指定VPN另一个端点的IP位置，如果对方的IP地址是未知的，我们可以将其指定为anonymous。

- exchange mode

设置Main Mode中的数据交换方法，有main、aggressive及base三种模式可供选择。至于这三种模式之间的差异性，在此就不加说明了。若你有兴趣，不妨自己参阅RFC 2408。这里仅略提一下：如果注重效率，请选择aggressive；如果注重安全，则请选择main。

- path include "/etc/racoon"

指定racoon工具配置文件所在的位置。

- path pre-shared-key "/etc/racoon/psk.txt"

指定在Preshared Keys验证模式中，Preshared Keys字符串所保存的文件名及路径。

- path certificate "/etc/racoon/certs"

指定X.509 Certificates验证模式中证书的保存位置。

最后需要提醒的是，Main Mode、Quick Mode在有些资料中会被称为Phase1及Phase2，当你看到这样的术语时，请不要觉得奇怪。

15.2 Preshared Keys验证模式下的传输模式VPN

这里以图15-5为例来演示如何使用Preshared Keys验证模式搭建传输模式的VPN系统。

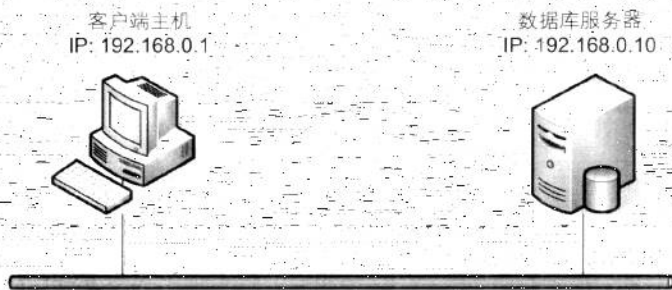


图15-5 传输模式VPN系统

15.2.1 数据库服务器的设置

● Racoon的设置:

请将/etc/racoon/racoon.conf的内容设置如下, 并将文件的读写权限设为(-rw----- root root)。

```
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote 192.168.0.1 {
    exchange_mode    main;
    proposal {
        authentication_method    pre_shared_key;
        dh_group                  modp1024;
        hash_algorithm            sha1;
        encryption_algorithm      3des;
        lifetime time              1 hour;
    }
}

sainfo anonymous {
    encryption_algorithm      3des;
    authentication_algorithm  hmac_sha1;
    lifetime time              30 minutes;
    compression_algorithm     deflate;
}
```

● 设置Preshared Keys:

本例将/etc/racoon/psk.txt作为密钥的保存位置, 其含义是指“如果要与192.168.0.1这台VPN主机建立连接, 要使用1234567作为验证密钥”。密钥部分是可以任意设置的, 并没有长度限制, 不过请把握一条原则: “密钥越长越复杂就越好”, 最后将文件读写权限改为“-rw----- root root”。

192.168.0.1	1234567
-------------	---------

● 设置安全策略:

安全策略与第8章所介绍的设置方法完全相同, 这里就不再重复介绍这些内容。在这个示例中, 请把文件保存为/etc/racoon/setkey.conf, 并将文件读写权限改成“-rw----- root root”。


```
flush;
spdf flush;

spdadd 192.168.0.1 192.168.0.10 any -P in ipsec
    esp/transport//require
    ah/transport//require;

spdadd 192.168.0.10 192.168.0.1 any -P out ipsec
    esp/transport//require
    ah/transport//require;
```

15.2.2 客户端主机的设置

● Racoon的设置:

请将/etc/racoon/racoon.conf的内容设置如下, 并将文件的读写权限设置为“-rw----- root root”。

```
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote 192.168.0.10 {
    exchange_mode    main;
    proposal {
        authentication_method    pre_shared_key;
        dh_group                  modp1024;
        hash_algorithm            sha1;
        encryption_algorithm      3des;
        lifetime time              1 hour;
    }
}

sainfo anonymous {
    encryption_algorithm      3des;
    authentication_algorithm  hmac_sha1;
    lifetime time              30 minutes;
    compression_algorithm     deflate;
}
```

● 设置Preshared Keys:

在这个示例中, 我们将/etc/racoon/psk.txt作为密钥的保存位置, 此外请将文件的读写权限设置为“-rw----- root root”。

```
192.168.0.10      1234567
```

- 设置安全策略:

在这个示例中, 请将内容保存为/etc/racoon/setkey.conf, 并将文件的读写权限设置为“-rw----- root root”。

```
flush;
spdf flush;

spdadd 192.168.0.1 192.168.0.10 any -P out ipsec
    esp/transport//require
    ah/transport//require;

spdadd 192.168.0.10 192.168.0.1 any -P in ipsec
    esp/transport//require
    ah/transport//require;
```

设置完数据库服务器及客户端主机上VPN的参数后, 你会发现两边的VPN配置文件内容几乎相同, 差别仅在于racoon配置文件中的remote VPN的位置、Preshared Keys验证的对象以及安全策略的方向, 其他内容都是相同的。

15.2.3 启动VPN

接下来启动VPN。这里要分为两部分来讨论, 其一是IKE机制的启动, 其二是安全策略(Security Policy)的设置, 方法分别如下。

- 启动IKE机制:

RedHat Enterprise Linux、CentOS及Fedora等Linux版本内置的ipsec-tools软件并没有为racoon准备SysV初始脚本, 因此不管racoon的启动或停止必须手动来执行, 如下所示。

- Racoon的启停

```
[root@localhost ~]# service racoon { start | stop | restart }
```

- 设置、查看、清除SPD及SAD的内容

- 设置安全策略

```
[root@localhost ~]# setkey -f /etc/racoon/setkey.conf
```

- 查看SAD内容

```
[root@localhost ~]# setkey -D
```

- 查看SPD内容

```
[root@localhost /]# setkey -D -P
```

- 清除SAD内容

```
[root@localhost /]# setkey -D -F
```

- 清除SPD内容

```
[root@localhost /]# setkey -P -F
```

设置完安全策略后，我们就可以在客户端主机上运行ping 192.168.0.10。ping开始时应该没有应答的，不过请多执行两三次，就会产生正常的应答了。图15-6记录了这个过程。

```
[root@localhost racoon]# ping 192.168.0.10
connect: Resource temporarily unavailable
[root@localhost racoon]# ping 192.168.0.10
connect: Resource temporarily unavailable
[root@localhost racoon]# ping 192.168.0.10
PING 192.168.0.10 (192.168.0.10) 56(84) bytes of data.
64 bytes from 192.168.0.10: icmp_seq=1 ttl=64 time=0.473 ms
64 bytes from 192.168.0.10: icmp_seq=2 ttl=64 time=0.305 ms
```

图15-6 VPN的启动过程

15.3 Preshared Keys验证模式下的隧道模式VPN

下面以图15-7为例来演示如何将Preshared Keys用作验证模式来搭建隧道模式的VPN系统。

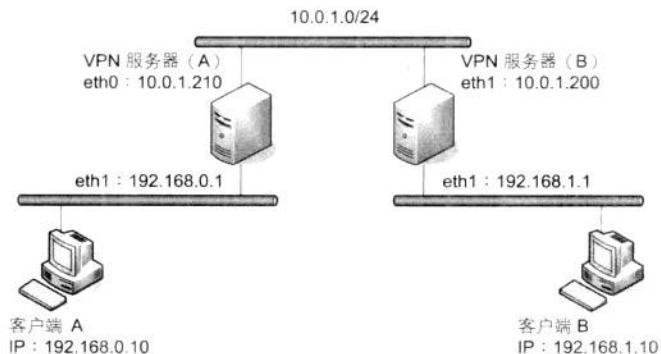


图15-7 隧道模式VPN示例

15.3.1 VPN 服务器(A)主机上的设置

● Racoon的设置

请将/etc/racoon/racoon.conf的内容设置如下，并将文件的读写权限设置为“-rw----- root root”。

```
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote 10.0.1.200 {
    exchange_mode    main;
    proposal {
        authentication_method    pre_shared_key;
        dh_group                  modp1024;
        hash_algorithm            sha1;
        encryption_algorithm      3des;
        lifetime time              1 hour;
    }
}

sainfo anonymous {
    encryption_algorithm      3des;
    authentication_algorithm  hmac_sha1;
    lifetime time              30 minutes;
    compression_algorithm     deflate;
}
```

● 设置Preshared Keys

本示例将密钥的存放位置设置为/etc/racoon/psk.txt，另外请将文件的读写权限设置为“-rw----- root root”。

```
10.0.1.200      1234567
```

● 设置安全策略

在这个示例中，请将以下内容保存为/etc/racoon/setkey.conf，并将文件的读写权限设置为“-rw----- root root”。

```
flush;
spdflush;

spdadd 192.168.0.0/24 192.168.1.0/24 any -P out ipsec
        esp/tunnel/10.0.1.210-10.0.1.200/require;

spdadd 192.168.1.0/24 192.168.0.0/24 any -P in ipsec
        esp/tunnel/10.0.1.200-10.0.1.210/require;
```

15.3.2 VPN 服务器(B)主机上的设置

● Racoon的设置

请将/etc/racoon/racoon.conf的内容设置如下，并将文件的读写权限设置为“-rw----- root root”。

```
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote 10.0.1.210 {
exchange_mode main;
proposal {
    authentication_method      pre_shared_key;
    dh_group                    modp1024;
    hash_algorithm              sha1;
    encryption_algorithm        3des;
    lifetime time                1 hour;
}
}

sainfo anonymous {
    encryption_algorithm        3des;
    authentication_algorithm     hmac_sha1;
    lifetime time                30 minutes;
    compression_algorithm        deflate;
}
```

● 设置Preshared Keys

在这个示例中，我们将密钥的存放位置设置为/etc/racoon/psk.txt，另外请将文件的读写权限设置为“-rw----- root root”。

```
10.0.1.210      1234567
```

● 设置安全策略

在这个示例中，请把下面的内容保存为/etc/racoon/setkey.conf，并将文件的读写权限设置为“-rw----- root root”。

```
flush;
spdflush;

spdadd 192.168.1.0/24 192.168.0.0/24 any -P out ipsec
    esp/tunnel/10.0.1.200-10.0.1.210/require;

spdadd 192.168.0.0/24 192.168.1.0/24 any -P in ipsec
    esp/tunnel/10.0.1.210-10.0.1.200/require;
```

在以上两台VPN主机都设置完毕后，我们就可以在192.168.0.10主机上运行ping 192.168.1.10，如果顺利应该可以得到192.168.1.10主机的应答。

15.4 何谓数字证书

在现实生活中，如果我们需要确认某人的身份时，可以通过对方的身份证来确认，但在网络世界，当我们访问某些重要服务时，可能也需要确认服务器的身份，这时就可以通过服务器的数字证书来验证。数字证书是一种无法被篡改的数字信息，所以在我们的系统中，它其实就是一种以文件形式存在的数字信息。

15.4.1 数字证书的必要性

一般在访问某些重要网站时，通常网站都会要求我们使用https的安全链路来连接网页服务器，下面以图15-8为例来说明https协议的工作流程。



图15-8 https的工作流程

首先在Web服务器上需要先准备一组密钥，分别是公钥(Public Key)和私钥(Private Key)，至于公钥和私钥笔者在第14.3节中已经介绍过，在此就不再多加说明，https协议的工作流程如下：

步骤01：客户端对Web服务器提出https连接请求①。

步骤02：Web服务器将其上的公钥传输给客户端②。

步骤03：客户端的应用程序随机生成一个加密密钥，我们称其为会话密钥(Session Key)③。

步骤04：客户端的应用程序使用公钥将会话密钥加密，并且发送给Web服务器④。

步骤05: Web 服务器使用私钥将被加密后的会话密钥解开, 如此这个会话密钥就只有客户端及Web 服务器拥有⑤。

步骤06: 然后这两台主机间所传输的数据都使用会话密钥来进行对称加密⑥。

从以上流程来看, 客户端及Web 服务器之间所传输的数据似乎已得到安全的保证, 不过事实并非如此, 其实以上结构相对脆弱且不安全, 因为我们无法确认所连接的Web 服务器就是要访问的对象。例如, 某网上银行的网址是`https://www.XXXXX.com`, 而其FQDN所对应的IP应该是192.168.0.10, 在此, 我们可以搭建一台Web 服务器, 并特意把网页内容做得与`https://www.XXXXX.com`相同, 这样一般使用者就很难分辨出网站的真伪, 接着就可以通过某些方式来篡改客户端主机的hosts文件内容。

例如, 以计算机病毒的方式来达到这个目的, 并且在其hosts的文件内把对`https://www.XXXXX.com`的访问操作转向到我们自己构建的网页, 这样当客户端主机在浏览`https://www.XXXXX.com`时, 其所连接到的网站将是我们的网站, 而非该银行的网站。接着使用者如果输入其帐户及密码, 我们就可以成功地窃取使用者的帐户及密码。因此, 如图15-8的结构只能保证客户端及服务器之间所传输的数据都是经过加密的, 但无法保证所连接到网站就是我们所要访问的对象。

15.4.2 证书管理中心

为了解决网络上身份识别的问题, 我们可以借助证书管理中心(Certificate Authority, CA)的架构。而CA其原意是指受信任的第三方公正机构, 其目的是扮演一个公证角色。例如, 我们都有身份证, 因此当你出示身份证时, 这就代表是你。

下面以构建一个拥有https连接能力的网站为例, 来介绍CA在其中所扮演的角色, 并且从中了解CA发放证书的方式及原理, 并讨论如何验证证书的真伪。

1. 生成证书申请单

为了部署一个具备https连接能力的网站, 并证明自己的身份, 那就必须为网站申请一个证书, 如此才能达到以上两个要求。要获取证书必须先为网站填写一份证书申请单, 然后提交给CA来处理, 下面示范如何为网站生成证书申请单。

```
[root@localhost /]# cd /etc/pki/tls/misc ① 将工作目录改为 /etc/pki/tls/misc
[root@localhost misc]#
[root@localhost misc]# ls
CA  c_hash  c_info  c_issuer  c_name
[root@localhost misc]#
[root@localhost misc]# ./CA -newreq ② 执行CA工具并使用newreq表明要填写证书申请单
```

```

Generating a 1024 bit RSA private key
+++++
.....
writing new private key to 'newkey.pem'
newkey.pem
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
1 请输入你网站的信息,而这些信息最后会存放在你申请下来的证书之中
Country Name (2 letter code) [GB]:CN
State or Province Name (full name) [Berkshire]:Taiwan
Locality Name (eg, city) [Newbury]:Taipei
Organization Name (eg, company) [My Company Ltd]:SYSTEX
Organizational Unit Name (eg, section) []:EDU
Common Name (eg, your name or your server's hostname) []:edu.yuu.com.tw
Email Address []:XXXXXn@system.com.tw

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Request is in newreq.pem, private key is in newkey.pem
10
[root@localhost misc]#

```

③ 开始以随机方式生成私钥

④ 将生成的私钥存储成文件

newkey.pem

⑤ 请输入一组密码来加密私钥

⑥ 请再输入一次密码以确认密码的正确性

⑧ 请直接按Enter即可

⑨ 请直接按Enter即可

⑩

下面详细讨论上面的每个步骤:

(1) 将当前工作目录转换到/etc/pki/tls/misc①。

(2) 在当前工作目录下运行一个可执行文件必须在命令前加上./,而-newreq参数表示要生成一份证书申请单②。

(3) 系统生成网站的私钥,由于是以文件形式保存,为了确保私钥的安全性,习惯上会使用3DES将私钥加密保存③④⑤⑥。

(4) 输入网站的信息,这些信息将保存在申请下来的证书之中⑦。

(5) 直接按Enter键跳过即可⑧⑨。

(6) 生成证书申请单之后,申请单及私钥会被保存在当前路径下,文件名是newreq.pem及newkey.pem。

2. 证书申请单及私钥的结构

● 证书申请单的结构:

如果我们直接打开newreq.pem文件, 其内容就如图15-9所示, 不过这些内容其实包含有两项重要的信息。如图15-10所示, 证书申请单的内容包含网站信息(即我们在生成证书申请单的第⑦个步骤中所输入的内容)以及网站的公钥。

```
-----BEGIN CERTIFICATE REQUEST-----
MIIB0TCCAToCAQAwgZAxChAqBgNVBAYTA1RlR3MScwJQYJKoZIhvcNAQkBFhhqYWNreV9jaGVu
BgNVBAMTDmVkdS51dXUuY29tLnR3MScwJQYJKoZIhvcNAQkBFhhqYWNreV9jaGVu
QHN5c3RleC5jb20udHcwZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANznBiPl
38qQ5uqVEKNQfQEc7fHiIzft065sIeNHws99WjOmgoRuet7Bp9PhIyCzpD3WFSK/
VSYusMf0ywwpWuwq29ZrDPmn8vSMjK/9czuh/qeizrzhHgbtrYstg9kqKpU1Qrmd
BT8J/Z0cfUamlw5zmtAtYKN5ny0fdKYPw4gNagMBAAGgADANBgkqhkiG9w0BAQUF
AA0BgQB7TaTTV1lhBkN/3+HKgCw7rPzi9IXS316r2Lk1JEMbdOHGhbWA9ThFSiIF
ftHX00fsyFI9fpljWvX0u176jmIfR2bndcrwBKQdyXyD5MQb3SCy+RPsL9Q3aPxD
TmnGhwk0Tk/uaYzWtJkZJfDFX4shw1IOsp9xtfyJZRbB7sjYHA==
-----END CERTIFICATE REQUEST-----
```

图15-9 newreq.pem文件内容

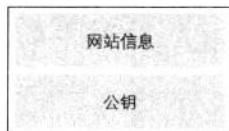


图15-10 证书申请单结构

● 私钥文件的结构:

图15-11是一个有设置密码保护的私钥文件, 从文件最上端部分我们可以清楚地看到加密信息。私钥文件不见得非要加密, 只不过一般若从安全角度考虑会设置密码来保护。图15-12是一个没有加密保护的私钥文件, 从图中可以清楚地看到, 并不存在加密包头的的数据。

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-EDE3-CBC, 094C4750B56836EE

WGfhcQlfiO/ADLWoCEW94RprCvB6byVuttrTGyo3v77gmdLpYCoMwlvR4IiH04
7bvhd5fIXheLeWercJbB7juLEc203lZoUp9IX7zKMTKZVZ6G6RIO6ABSNI4o1S
5xWwXiQp+c88WdWuexPteXjjtmxgjiYptZF0ksRjrYAwfBk8miE7tNyXqiy0jER
Py/LNg/EKwDktY33C9Cirs3JtxeebcBK2i3jMg9hOKT07qUT0NV7RLEGo0jzr5zP
3ZNTREqXhmkLxm644/sCQRiaN6YuVxuB615L4/pe8U/lQXrGl6C6STUlrSWcTi8
```

```
yJHAKrszZTxdM1fhVYfe4wbRktYqEVi7i72ba2uv+B36BAEOYK8Ops4/0bqJ5GqL
v/1kxBfw6oRxZ4hzpRNN86F5PidDDZRPmr6Ppcy+u/+V0tIaelFBXLeasQBkcpel
iFuZaxOWQLsbn+ni24Ds7wJiy8FNQhh3AYiRRXCB8bhPiTD/qjVTosMPm2MpBvqH
rpXn2ye7x+n83yUW709v6LT5Zn8sReXz4re8PUTteS74EDJQu2BK9q0FrG6EoXuG
IHC1rumBAZleTqJ/n69K3LeTUZHGP8Yblq59HHzx9HEaHDjHTg0aSn/xl4nrYj24
MtXg68hcuRzHXbIE9Z+87dEOEp+6yDi0t/aj1hZ1072m1JRNK6+J4UohcXYCwK9f
LBQkc/AQ9KFd5KdzN9Ac6LE4NEDonAlpiJ4EWZsNqAYPFSftWeYGqPPXotxQeLG/
rBaFBuq9k+KNxekdM0Cf+Cgoi+SdjiDgCweyXipuuUc=
-----END RSA PRIVATE KEY-----
```

图15-11 证书申请单结构

```
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQDZWzYj5d/Kk0bq1RCjUH0BH03x4iM37dOubCHjR8LPfVozpoKE
VHrewafT4SMgs6Q91hUivlUmLrDH9MsMKVrsKtvWawz5p/L0jlyv/XM7of6nos68
4R4G7a2LLYP2KiQVNUK5nQU/Cf2TnH1Gptc0c5rQLWCjeZ8tH3SmD8OIDQIDAQAB
AoGAU9nhBeMeXhvDo9T+t9B/wxKsYKA/g90on5u8uSXUo89zmqWxf82GbF4yo+P2
GBEJ2AFFLNzFkXRLQT1UvMHjw6hPS1WvJItXpq52qr10ePNyZYwLGdN+A7YaztR6
LcF6++ei8Tcx7PI5bkYvFFTD27u5whRLxPvf0ZLNXCAlcECQQDtH3ORKazAY8w1
6JLIwIu2HQLF2hm2NxnqZbRrRLGmFNikk22ENon+Y2jge5zcsj43R4I/ZYWhGaw
sU/AyGEhAkeA53iIJ/qCkJAGul3dtXR37hXUs2ZC85sXTus4wdtprbsLSyzDLNQH
uQL4OUcl/CovJYVvznbtKqMw7w895ONbQJAW0ZvCuK637fKpLmsyZNSdCaIqb2B
gkvECoGpyFzB8VacogS7KykjgJ28GsNqEUCPADTufw7KOr9UGiaZ2n8YQJALS06
LctXk0l8wPU/Nbhg+dg3y7LTG5tqyp/p15jJtPdZwTMIJi6iDiLJ+q6ADBfiUwv
vOb2+Zj75kv7/jofNQJANWAZZfaJtZBu8saP6HXVPCfAAdloUqcZDSy6xaYQqj9z
z5KuB0TDaTC7Jtvhrkg7yDkPvcVVR91PQWQ4WS/6jw==
-----END RSA PRIVATE KEY-----
```

图15-12 证书申请单结构

3. 私钥的加解密

私钥加密与否可以自行决定，因此一个被加密过的私钥随时都可以将加密的部分取消，也可以将一个未加密的私钥加上密码来进行保护。以下说明如何为私钥加上保护及去除保护。

- 为私钥设置密码保护

```
openssl rsa -des3 -in newkey.txt -out newkey.des3
```

- 为私钥去除密码保护

```
openssl rsa -in newkey.pem -out newkey.txt
```

4. 将证书申请单提交CA处理

在图15-8中，https工作流程中最大的问题是，用户端无法确认所连接的主机是否就是我们所要连接的对象，不过这个问题到目前为止已经得到初步解决。从图15-10可以看到，我

们在公钥上附加了公钥的拥有者，因此用户端主机拿到公钥之后，还可以通过其上所附加的信息来识别公钥的拥有者是谁，不过这样真的就能证明网站的身份吗？其实是不够的。在图15-10的结构中，并没有任何防伪机制，因此任何人只要在Linux平台上就可以伪造出其他网站的信息。

为了使以上两项信息更具公信力，我们可以把证书提交给CA来进行处理，那CA在什么地方呢？其实CA在网络上到处都有，只要按照网页上的说明，将证书申请单提交给CA即可。CA会要求你将公司的营业执照等书面材料传真给CA，接着，CA会审核证书申请单所填写的网站信息与书面材料是否吻合，如不吻合则拒绝处理，并要求申请者重新上传正确的证书申请单；如吻合，CA会进一步处理证书申请单。

首先CA会将它的信息加入到证书申请单之中，以表明证书申请单是由哪家CA所处理的③，接着加入有效的使用期限④，最后CA会对处理后的证书申请单进行签名的操作。而所谓的签名是将①②③④这四项信息经过哈希计算，再把计算后所得到的指纹使用CA的私钥进行加密，所以签名完成后的证书申请单就是证书，如图15-13所示。



图15-13 证书结构

5. 如何验证证书的有效性

当我们在网络上拿到一个证书时，该如何去验证证书的有效性呢？以图15-14为例来说明证书的验证过程，开始时先将证书中的①②③④四个部分使用哈希算法进行计算①，而计算所得到的值我们定义为Fingerprint(A)，接着如果我们可以取得原来由CA所计算出来的指纹，那么就有了验证证书的依据，但是原来CA所计算出来的指纹已经使用CA的私钥加密过，因此，我们不可能直接取出内容。不过，如果可以取得CA的公钥，就可以将原来CA计算得到的指纹取出，但从何处可以取得CA的公钥呢？关于这个问题倒是很简单，因为在因特网上各大CA的证书早就内置在各大操作系统中。以Windows 7为例，我们可以选择“开始”|“控制面板”|“Internet选项”|“内容”|“证书”|“受信任的根证书颁发机构”，在如图15-15所示的界面中可以看到因特网上各大CA的证书，而证书中会包含公钥。

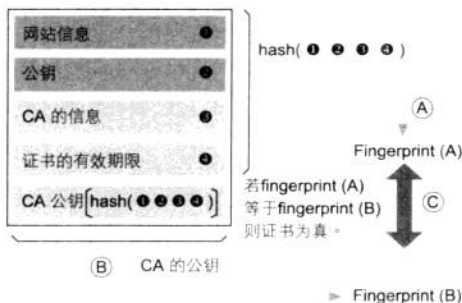


图15-14 证书的验证过程



图15-15 Windows 7的证书数据库

在得到CA的公钥之后，我们就可以轻松解开原来被加密过的指纹，定义为Fingerprint(B)，最后如果Fingerprint(A)等于Fingerprint(B)，就可以确定证书内容真实，没有被篡改过；反之，证书就是无效的。

6. 回顾https的工作流程

了解CA的工作原理之后，接着再来看一次https的工作流程。首先在Web 服务器我们必须先将网站的证书及私钥准备好。当客户端对Web 服务器进行https的服务请求时①，Web 服

服务器会将其证书传送给客户端主机②，在客户端主机收到Web服务器的证书之后，接着进行证书的验证，如果验证的结果有误，客户端的应用程序将会提出警告信息③，否则客户端就使用随机数生成会话密钥④，并且使用证书中的公钥来加密会话密钥，然后发送给Web服务器⑤。等到Web服务器收到被加密过的会话密钥之后，Web服务器就会用它的私钥将会话密钥解开⑥。接下来客户端与Web服务器之间所传输的所有数据，都会用会话密钥来进行对称加密。图15-16详细说明了这个过程。

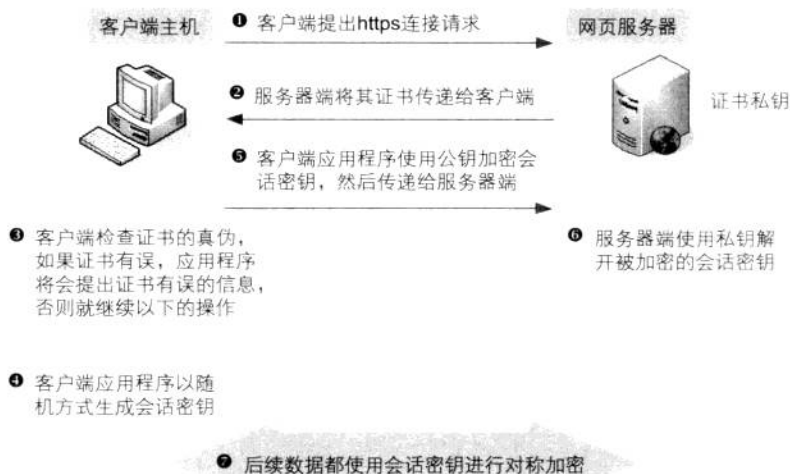


图15-16 回顾https的工作流程

15.4.3 将Linux系统作为企业的CA

由于证书的处理是需要付费的，且通常费用都不便宜，但是SSL的加密连接机制在目前的企业网络中使用的范围及机会都相当高，如Web Mail、POP3S、IMAPS及SMTPS等都使用SSL加密连接机制。只要使用到SSL的加密连接机制，在服务器端就一定需要提供证书，如果没有证书，则SSL机制就无法工作。因此在企业网络中，证书的需要量很大，如果我们真为每个服务申请一个证书，那么企业每年所需负担的成本一定会相当惊人。

幸运的是在开源软件领域，OpenSSL组织开发了一个名为OpenSSL的软件，我们只需要通过这个软件，就可以轻松地在企业中搭建属于自己的证书中心，不过这样的证书中心在使用上是有限制的。至于限制是什么，笔者稍后会做详细介绍，我们先来了解如何使用OpenSSL建立企业的证书中心。

1. 建立企业证书中心

```
[root@localhost /]# cd /etc/pki/tls/misc
[root@localhost misc]#
[root@localhost misc]# ls
CA c_hash c_info c_issuer c_name
[root@localhost misc]#
[root@localhost misc]# ./CA -newca
mkdir: cannot create directory '../..CA': File exists
mkdir: cannot create directory '../..CA/private': File exists
CA certificate filename (or enter to create)
Making CA certificate ...
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to '../..CA/private/..cakey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:CN
State or Province Name (full name) [Berkshire]:Taiwan
Locality Name (eg, city) [Newbury]:Taipei
Organization Name (eg, company) [My Company Ltd]:SYSTEX
Organizational Unit Name (eg, section) []:EDU
Common Name (eg, your name or your server's hostname) []:ca.systex.com.tw
Email Address []:jacky_chen@systex.com.tw

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /etc/pki/tls/openssl.cnf
Enter pass phrase for ../..CA/private/..cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number: 0 (0x0)
    Validity
        Not Before: Mar 22 21:43:34 2008 GMT
        Not After : Mar 22 21:43:34 2011 GMT
```

① 变换路径至 /etc/pki/tls/misc

② 使用-newca参数表明要建立企业CA

③ 建立好企业CA后, 确定存储其证书及私钥的文件名
这里直接按Enter使用默认值即可

④ 输入保护企业CA私钥的密码

⑤ 输入CA的信息

⑥ 请按Enter跳过即可

⑦ 请按Enter跳过即可

⑧ 请输入保护CA私钥的密码

⑨ 企业CA正式建立完成, 如下信息即为企业CA的证书

```
Subject:
countryName = CN
stateOrProvinceName = Taiwan
organizationName = SYSTEX
organizationalUnitName = EDU
commonName = ca.systemex.com.tw
emailAddress = XXXXX@systemex.com.tw
```

以上就是企业证书中心的建立过程，下面将解释流程中的每一个步骤：

- (1) 把当前工作路径设置到/etc/pki/tls/misc路径下，以便建立CA①。
 - (2) 执行CA命令，并输入-newca参数说明是要建立新的企业CA②。
 - (3) 建立好企业CA之后，设置保存其证书及私钥使用的文件名，直接按Enter键使用默认值即可继续③。
 - (4) 设置保护企业CA私钥的密码④。
 - (5) 输入CA的信息⑤。
 - (6) 按Enter键跳过即可继续⑥⑦。
 - (7) 输入保护CA私钥的密码，也就是第4个步骤中所设置的密码⑧。
 - (8) 企业CA部署成功，接着在下面输出CA证书的内容⑨。
- 建立了企业CA后，其证书及私钥分别保存为以下文件。

- 证书的保存路径及文件名

```
/etc/pki/CA/cacert.pem
```

- 私钥的保存路径及文件名

```
/etc/pki/CA/private/cakey.pem
```

2. 证书的审核及签发

当使用者提交其证书申请单之后，我们必须将证书申请单放到/etc/pki/tls/misc路径下，且文件名一定要为newreq.pem，这两个条件只要其中一项不成立，就无法完成证书的签发工作。当条件满足后，就可进行下面的证书签发流程。

```
[root@localhost /]# cd /etc/pki/tls/misc          ① 变更路径至/etc/pki/tls/misc
[root@localhost misc]# ls
CA c_hash c_info c_issuer c_name newreq.pem
[root@localhost misc]#
[root@localhost misc]# ./CA -sign                ② 执行CA命令，并使用-sign表明要执行证书签发作业
```

Using configuration from /etc/pki/tls/openssl.cnf
 Enter pass phrase for ../../CA/private/akey.pem: ③ 输入CA私钥的密码
 Check that the request matches the signature
 Signature ok
 Certificate Details: ④ 显示出使用者的凭证申请书内容, 以便管理者进行核对作业

Serial Number: 1 (0x1)

Validity

Not Before: Mar 22 22:08:06 2008 GMT

Not After : Mar 22 22:08:06 2009 GMT

Subject:

countryName = CN

stateOrProvinceName = TW

localityName = Taipei

organizationName = SYSTEX

organizationalUnitName = EDU

commonName = edu.uuu.com.tw

emailAddress = XXXXX@system.com.tw

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

Netscape Comment:

OpenSSL Generated Certificate

X509v3 Subject Key Identifier:

26:71:21:5B:D8:88:1D:11:68:07:A5:8C:57:31:51:0E:1C:89:29:E3

X509v3 Authority Key Identifier:

keyid:08:64:48:C1:58:A3:69:0E:6E:9C:E6:CC:08:43:D1:81:8C:0C:73:C6

Certificate is to be certified until Mar 22 22:08:06 2009 GMT (365 days)

Sign the certificate? [y/n]:y ⑤ 是否执行凭证签发作业

1 out of 1 certificate requests certified, commit? [y/n]:y ⑥ 请再确认执行凭证签发作业

Write out database with 1 new entries

Data Base Updated

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 1 (0x1)

Signature Algorithm: sha1WithRSAEncryption

Issuer: C=CN, ST=Taiwan, O=SYSTEX, OU=EDU, CN=ca.system.com.tw/

emailAddress=XXXXX@system.com.tw

Validity

Not Before: Mar 22 22:08:06 2008 GMT

Not After : Mar 22 22:08:06 2009 GMT

Subject: C=CN, ST=TW, L=Taipei, O=SYSTEX, OU=EDU, CN=edu.uuu.com.tw/

emailAddress=XXXXX@system.com.tw

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

```

Modulus (1024 bit):
00:b8:7f:ac:f1:df:d1:d2:c0:0d:8d:14:46:a8:b2:
86:cd:2c:80:70:9c:ba:99:19:fd:ab:3a:6f:72:fe:
fb:41:72:4c:56:ba:37:13:c4:5c:6a:30:e6:2f:a7:
32:21:34:ae:80:b8:dc:d5:13:c6:4f:48:f4:3f:dd:
05:ee:89:f0:72:14:59:1a:86:32:53:a8:76:77:19:
1f:95:92:78:a5:9e:49:f1:f8:4f:24:d0:59:96:d0:
05:57:79:12:f3:12:e2:19:f7:2b:a0:c8:ac:cb:d9:
c2:52:c0:ec:bd:96:66:ad:2c:02:31:f8:bb:22:76:
9f:7f:5d:24:22:96:e4:00:d9
Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
26:71:21:5B:D8:88:1D:11:68:07:A5:8C:57:31:51:0E:1C:89:29:E3
X509v3 Authority Key Identifier:
keyid:08:64:48:C1:58:A3:69:0E:6E:9C:E6:CC:08:43:D1:81:8C:0C:73:C6
Signature Algorithm: sha1WithRSAEncryption
61:ea:70:a3:6e:53:92:e5:ba:a6:94:cb:69:7f:b4:f5:25:a3:
9a:ab:85:94:c7:7c:3a:30:c9:87:f5:61:8e:ef:83:9d:4e:f4:
b3:11:5b:da:2f:c3:75:b6:24:d8:d0:dc:09:a1:e0:3e:21:1e:
24:74:59:ba:92:4a:51:a6:40:44:7c:66:fc:23:b1:c6:bd:f9:
c3:7d:96:51:ee:04:08:3d:70:a9:10:6f:c1:9b:e5:b8:5e:07:
34:41:8d:f7:51:87:5f:d2:0e:f3:67:28:b5:f3:1e:f3:c6:ea:
b4:97:49:33:70:7c:c6:eb:75:34:87:2f:ca:ee:df:27:14:5b:
21:72
-----BEGIN CERTIFICATE-----
MIIDADCCAmgAwIBAgIBATANBgkqhkiG9w0BAQUFAADCBgTElMAkGA1UEBhMCVFcx
DzANBgNVBAgTB1RhaXdhbjEPMA0GA1UEChMGU11TVEVYMjQwCgYDVQQLZWwNFRFUX
GTAXBgNVBAMTEGhhLnN5c3R1bS5jb20udHcxJzA1B2kqhkiG9w0BCQEWGGPhY2t5
X2NoZW5Ac3lzdGV4LmNvbS50d2AeFw0wODAzMjIyMjA4MDZaFw0wOTAzMjIyMjA4
MDZaMIGMMQswCQYDVQQGEwJUVzELMAkGA1UECBMCVFcxDzANBgNVBAcTB1RhaXB1
aTEPMA0GA1UEChMGU11TVEVYMjQwCgYDVQQLZWwNFRFUXFzAVBgNVBAMTDmVkdS51
dXUuY29tLnR3MScwJQYJKoZIhvcNAQkBFhhqYWNreV9jaGVuQHN5c3R1eC5jb20u
dHcwZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBALH/rPHf0dLADY0URqiyhs0s
gHCcupkZ/as6b3L++0FyTFa6NxPEXGow5i+nMiE0roC43NUTxk9I9D/dBe6J8HIU
WRqGMlOodncZH5WSeKWeSfH4TyTQWZbQBvd5EvMS4hn3K6DlrMv2w1LA7L2W2q0s
AjH4uyJ2n39dJCKW5ADZAgMBAAGjczB5MAkGA1UdEwQCMAAwLAYJYIZIAyB4QgEN
BB8WHU9wZW5TU0wgr2VuzXJhdGVkIEN1cnRpZmljYXR1MB0GA1UdDgQWBWbQmcSfb
2IgdEWgHpYxXWVEOHlKp4zAfBgNVHSMEGDAwGBQIzEjBWNpDm6c5swIQ9GBjAxz
xjANBgkqhkiG9w0BAQUFAAOBQBH6n6Cj51OS5bqmlMtpf7T1JaOaq4WUx3w6MMmH
9WG074OdTvSzEVvaL8N1tiTY0NwJoeA+IR4kdFm6kkrRpkBEfGb8I7HGvfnDfZZR
7gQIPXcPg/Bm+W4XgcQY33UYdf0g7zZyil8x7zxuq010kzcH2G63U0hy/K7t8n
FFshcg==
-----END CERTIFICATE-----

```

```
Signed certificate is in newcert.pem
[root@localhost misc]#
```

以上就是证书的审核及签发流程，下面将说明流程中的每一个步骤：

- (1) 将当前工作路径设置到/etc/pki/tls/misc，以便证书签发操作的进行①。
- (2) 执行CA命令，输入参数-sign表明是要执行证书签发操作②。
- (3) 证书的签发操作需要使用CA的私钥，而CA的私钥是有设置密码保护的，如果没有CA私钥的密码，我们将无法进行证书的签发操作，所以要输入CA私钥的密码③。
- (4) 列出使用者证书申请单的内容，以便管理者进行核对操作④。
- (5) 确认是否进行证书签发操作⑤。
- (6) 再次确认是否进行证书的签发操作⑥。

签发完毕后，新生成的证书保存在/etc/pki/tls/misc目录中，且文件名必须是newcert.pem。接着把证书交还给申请者，而这个交换的方法通常会使用E-Mail来寄送，最后就可以把CA上保留下来的证书申请单以及新生成的证书给删除掉，因为在CA上并不需要保存这些文件。

3. 如何注销证书

证书的目的在于证明证书内公钥拥有者的身份，公钥的用途是作为某人要把机密数据加密传输给我们时用来加密数据的密钥。这些被加密过的数据就只有证书内公钥所对应的私钥才可以解开，如果我们不小心遗失了私钥，那么得到这个私钥的人就可以使用私钥来解开所有的机密数据。万一真的遗失了，为将损失降至最低，请迅速注销你的证书，以免扩大损失，下面来分析如何注销证书。

要注销证书，首先需要生成一个证书注销清单，清单上将列出所有被注销的证书列表，而所有客户端会定时到CA获取证书注销清单，这样客户端才能知道哪些证书是已经注销了的。

● 生成空白的证书注销清单：

```
[root@localhost /]# echo "01" > /etc/pki/CA/crlnumber ①
[root@localhost /]# cd /etc/pki/CA/crl ②
[root@localhost crl]# openssl ca -gencrl -out crl.pem ③
```

其中①的目的在于生成证书注销的编号，这个编号从01开始算起，每注销一个证书，则数值累加1，而②的目的在于改变工作路径，以便生成证书注销清单，最后③的操作就是实际生成证书注销清单。

● 注销证书：

在将某个证书注销之前，我们必须取得这个证书，否则无法进行证书注销操作。不过这并不需要使用者提供证书，因为当我们在CA上签发证书的同时，系统会自动保留一份签发

出去的证书，并保存在/etc/pki/CA/newcerts目录下。因此在注销一份证书之前，请先找到要注销的证书文件，接着就可以进行证书注销操作，如下所示。

```
cd /etc/pki/CA/crl
openssl ca -revoke ../newcerts/01.pem ❶
openssl ca -gencrl -out crl.pem ❷
```

其中❶是指要注销01.pem这份证书，❷则是重新生成一份最新的证书注销清单。

15.5 数字证书验证模式下的传输模式VPN

下面以图15-17为例来搭建传输模式的VPN，并改用数字证书的方法来进行双向身份确认。

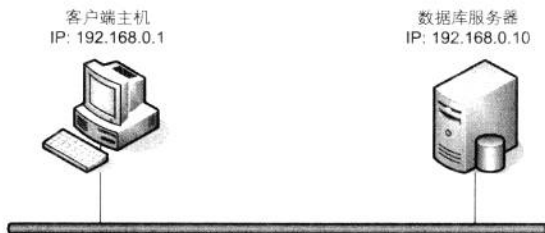


图15-17 传输模式VPN示例

15.5.1 证书的生成及保存

首先，必须为两台VPN主机生成各自的证书及私钥，此外，还要准备好CA的证书及证书注销清单。我们还需要注意一点，因为证书具有有效的起始时间和结束时间，因此CA及VPN主机的系统时间必须要正确，否则可能会因为时间不对，而造成证书验证错误，最后导致VPN建立失败，表15-1列出了这些所需的信息。

表15-1 所需的证书文件列表

主机名称	客户端主机	数据库服务器
VPN主机的证书	192.168.0.1.pem	192.168.0.10.pem
VPN主机的私钥	192.168.0.1.key	192.168.0.10.pem
CA的证书	cacert.pem	cacert.pem
证书作废清单	crl.pem	crl.pem

接着请将这些文件分别复制到两台VPN主机的/etc/racoon/certs目录下,我们必须针对cacert.pem及crl.pem两个文件生成hash link,因为racoon会根据hash link来查找CA的证书及证书注销清单。hash link的生成方法如下。

```
[root@localhost ~]# cd /etc/racoon/certs/
[root@localhost certs]# ln -s cacert.pem $(openssl x509 -noout -hash < cacert.pem).0 ❶
[root@localhost certs]# ln -s crl.pem $(openssl x509 -noout -hash < cacert.pem).r0 ❷
[root@localhost certs]# ls -l
total 16
lrwxrwxrwx 1 root root 10 Mar 26 01:48 5eb82dea.0 -> cacert.pem
lrwxrwxrwx 1 root root 7 Mar 26 01:48 5eb82dea.r0 -> crl.pem
-rw-r--r-- 1 root root 887 Mar 24 01:14 192.168.0.1.key
-rw-r--r-- 1 root root 3237 Mar 24 01:14 192.168.0.1.pem
-rw-r--r-- 1 root root 3210 Mar 24 01:14 cacert.pem
-rw-r--r-- 1 root root 544 Mar 24 01:14 crl.pem
[root@localhost certs]#
```

其中5eb82dea.0就是cacert.pem文件的hash link,而5eb82dea.r0则是crl.pem的hash link,另外,要提醒你的是❶❷生成hash link的命令中,“x509”及行尾的“.0”,还有“.r0”中0是数字的“零”,不要弄错了。

15.5.2 客户端VPN主机的设置

● /etc/racoon/racoon.conf配置文件

```
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote 192.168.0.10 {
    exchange_mode                main;
    certificate_type              x509 "192.168.0.1.pem" "192.168.0.1.key"; ❶
    verify_cert                  on; ❷
    my_identifier                asnldn; ❸
    peers_identifier             asnldn; ❹
    proposal {
        authentication_method    rsasig; ❺
        dh_group                 modp1024;
        hash_algorithm            sha1;
        encryption_algorithm      3des;
        lifetime time             1 hour;
    }
}

sainfo anonymous {
```

```

encryption_algorithm    3des;
authentication_algorithm  hmac_shal;
lifetime time           30 minutes;
compression_algorithm    deflate;
}

```

以上就是客户端主机上的`racoon.conf`配置文件，这个配置文件的内容与Preshared Keys验证方法的配置文件内容基本相同。其中的不同之处这里特别标记出来了，接着说明这些参数所代表的意思。

- `Certificate_type`

指定证书的类型、证书的文件名以及私钥的文件名，其中x509是证书的标准，192.168.0.1.pem是证书的文件名，192.168.0.1.key是私钥的文件名①。

- `verify_cert`

启用验证另一端VPN主机上证书的有效性②。

- `my_identifier`

本地端VPN主机的识别码，而识别码的类型有很多种，例如，`address`、`fqdn`、`user_fqdn`、`keyid`及`asn1dn`(其中1是数字)等，此处选用`asn1dn`，`asn1dn`识别码以证书中的数据为依据，如“Subject: C=CN, ST=TW, L=Taipei, O=SYSTEX, OU=EDU, CN=edu.uuu.com.tw/emailAddress=XXXXXX@systex.com.tw”③。

- `peers_identifier`

另一个端点主机的识别码，这个识别码的表示方法与`my_identifier`相同，而它的值一定要与另一个端点VPN主机的`my_identifier`相同④。

- `authentication_method`

定义两个端点VPN的验证方法。我们使用证书来验证双方，因此这里请填写`rsasig`⑤。

- 安全策略配置文件：

```

flush;
spdf flush;

spdadd 192.168.0.1 192.168.0.10 any -P out ipsec
    esp/transport//require;

spdadd 192.168.0.10 192.168.0.1 any -P in ipsec
    esp/transport//require;

```

安全策略的内容与之前所介绍的内容完全一样，因此这里就不再多加描述。

- 数据库服务器的IKE配置文件:
- /etc/racoon/racoon.conf配置文件:

```
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote 192.168.0.1 {
    exchange_mode                main;
    certificate_type              x509 "192.168.0.10.pem" "192.168.0.10.key";
    verify_cert                   on;
    my_identifierasnlidn;
    peers_identifier              asnlidn;
    proposal {
        authentication_method    rsasig;
        dh_group                  modp1024;
        hash_algorithm            sha1;
        encryption_algorithm      3des;
        lifetime time             1 hour;
    }
}

sainfo anonymous {
    encryption_algorithm          3des;
    authentication_algorithm      hmac_sha1;
    lifetime time                 30 minutes;
    compression_algorithm         deflate;
}
```

● 安全策略配置文件

```
flush;
spdflush;

spdadd 192.168.0.1 192.168.0.10 any -P in ipsec
    esp/transport//require;

spdadd 192.168.0.10 192.168.0.1 any -P out ipsec
    esp/transport//require;
```

● 启动IPSec

当两台VPN主机上的配置文件都设置完毕后,接下来就可以启动IPSec的机制,而启动方法与Preshared Keys验证模式是相同的。

15.6 数字证书验证模式下的隧道模式VPN

这里以图15-18为例来构建隧道模式的VPN，并且改用证书方法来进行双向身份确认。

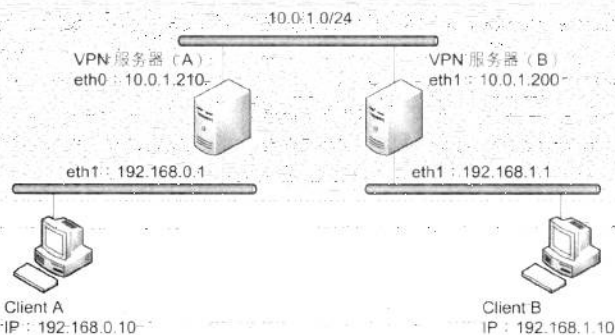


图15-18 隧道模式VPN示例

15.6.1 证书的生成及保存

证书的处理方法与传输模式VPN中的处理方法完全相同，这里不再重复说明。本例分别将10.0.1.200.pem、10.0.1.200.key，以及10.0.1.210.pem、10.0.1.210.key用作两台VPN主机上证书及私钥的文件名。

15.6.2 设置VPN 服务器(A)

- /etc/racoon/racoon.conf配置文件

```
path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote 10.0.1.200 {
    exchange_mode          main;
    certificate_type        x509 "10.0.1.210.pem" "10.0.1.210.key";
    verify_cert             on;
    my_identifier asn1dn;
    peers_identifier        asn1dn;
    proposal {
        authentication_method    rsasig;
        dh_group                  modp1024;
        hash_algorithm            sha1;
        encryption_algorithm      3des;
        lifetime time             1 hour;
    }
}
```

```

    }

sainfo anonymous {
    encryption_algorithm    3des;
    authentication_algorithm hmac_shal;
    lifetime time           30 minutes;
    compression_algorithm   deflate;
}

```

• 安全策略配置文件

```

flush;
spdflush;

spdadd 192.168.0.0/24 192.168.1.0/24 any -P out ipsec
    esp/tunnel/10.0.1.210-10.0.1.200/require;

spdadd 192.168.1.0/24 192.168.0.0/24 any -P in ipsec
    esp/tunnel/10.0.1.200-10.0.1.210/require;

```

15.6.3 设置VPN 服务器(B)

• /etc/racoon/racoon.conf 配置文件

```

path include "/etc/racoon";
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote 10.0.1.210 {
    exchange_mode            main;
    certificate_type          x509 "10.0.1.200.pem" "10.0.1.200.key";
    verify_cert               on;
    my_identifier             asnldn;
    peers_identifier          asnldn;
    proposal {
        authentication_method rsasig;
        dh_group               modp1024;
        hash_algorithm          sha1;
        encryption_algorithm    3des;
        lifetime time           1 hour;
    }
}

sainfo anonymous {
    encryption_algorithm      3des;
    authentication_algorithm   hmac_shal;
}

```



```
lifetime time          30 minutes;  
compression_algorithm  deflate;
```

- 安全策略配置文件

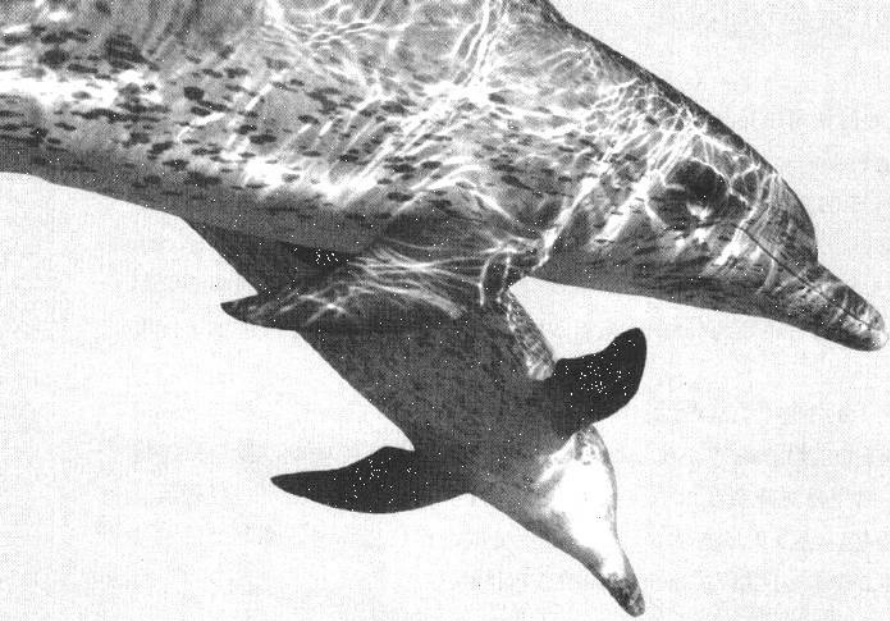
```
flush;  
spdf flush;  
  
spdadd 192.168.0.0/24 192.168.1.0/24 any -P in ipsec  
        esp/tunnel/10.0.1.210-10.0.1.200/require;  
  
spdadd 192.168.1.0/24 192.168.0.0/24 any -P ou ipsec  
        esp/tunnel/10.0.1.200-10.0.1.210/require;
```

15.6.4 启动IPSec

当两台VPN主机上配置文件都设置完毕后，接下来就可以启动IPSec机制，启动方法与Preshared Keys验证模式中是相同的。

15.7 小结

本章深入探讨了VPN技术，同时讨论了数字证书的相关技术，也介绍了证书是如何在VPN中应用的，希望能通过这个章节的内容来帮助你真正领会Linux中以IPSec为基础的VPN技术。



LINUX

|第16章| VPN: L2TP Over IPSec

16

前面的章节讲述了如何使用IPSec来构建VPN,但构建VPN除了使用IPSec之外,在业界L2TP(Layer 2 Tunneling Protocol, L2TP)也是一种常用来构建VPN的通信协议。根据目前操作系统发展的情况来看,L2TP最大的优势就在于其简便性,例如当你带着笔记本电脑外出时,该如何通过IPSec来建立VPN连接,来访问公司内部的数据呢?答案是MMC(Microsoft Management Console, MMC),但MMC并非所有人都会使用,并且笔记本电脑的网络配置会因为环境而异。因此我们只有不断通过MMC来修改IPSec的参数,才能在不同环境中使用VPN来连接企业网络。

相对于IPSec在使用上的不便,L2TP就好用多了。在目前各大操作系统中,如Microsoft Windows、WinCE、Mac OS及Linux等系统,都已经内置了L2TP的连接功能。我们只需将拨号网络设置好,并在你需要访问企业内部数据时启用L2TP的拨号功能,就可以轻松通过L2TP对企业建立VPN连接;更令人欣慰的是,不管你所处的网络环境如何,都不会影响使用者的操作行为,因此L2TP大多应用在Client to Site的VPN结构中。

16.1 何谓PPP

早期计算机与计算机之间的连接是通过调制解调器来连接,图16-1是一个很典型的PPP(Point-to-Point Protocol)连接。而PPP协议的工作原理相当简单,在图16-1中,两台计算机是使用调制解调器通过电话网络来连接,在这个结构中,主机A发送的数据就一定是给主机B的,而主机B所发送的数据也一定是给主机A的,绝对不会发生其他情况。简单来说,PPP在TCP 四层结构中是属于链路层的范畴。

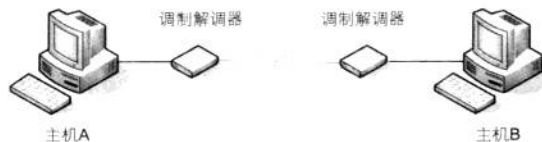


图16-1 PPP通信协议

16.2 何谓L2TP协议

在VPN隧道技术中,除了IPSec所提供的IP in IP Tunnel之外,我们还可以通过L2TP协议来建立隧道,而L2TP是属于TCP/IP网络结构中链路层,属于OSI七层结构中的第二层,其目

的是用来作为建立隧道用的协议，因此称之为Layer 2 Tunnel Protocol。

16.2.1 L2TP协议的原理

以图16-2为例，在L2TP服务器上有一组公网IP，在这个示例中，笔者假设其为10.0.1.210，我们假设客户端主机的IP是DHCP服务器所分配的，当L2TP连接建立之后，在L2TP服务器及客户端主机上会分别多出一个虚拟接口，这个虚拟接口的名称通常是PPP0，而且这两个PPP接口上的IP都属于同一个网段，因此这个结构看起来有点像把整个因特网当成一条虚拟连接来建立PPP连接，所以从192.168.0.1发送出去的数据就一定是给192.168.0.10，而从192.168.0.10发送出去的数据也一定是给192.168.0.1。

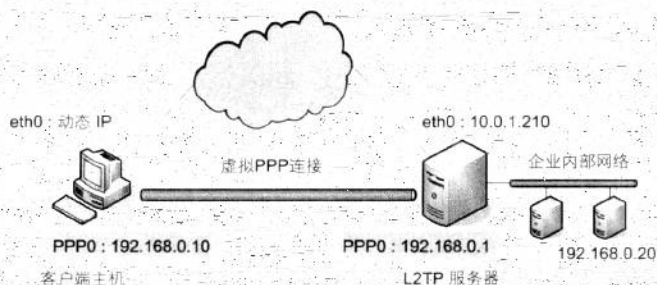


图16-2 L2TP示例图

在了解PPP及L2TP的工作模式之后，接下来以图16-2及图16-3为例来说明L2TP协议的工作原理。首先我们假设客户端主机要通过L2TP来建立VPN连接，并使用TELNET协议来连接192.168.0.20这台主机。一开始使用者在客户端主机上运行telnet 192.168.0.20命令，通过TELNET协议来连接到192.168.0.20这台主机，然后这个数据包被送到传输层，并且在数据包前面封装上TCP包头①，接着数据包也被送入到网络层的位置，并且在数据包前面加上一个IP包头②。不过此时IP包头内的Source IP应该是192.168.0.10，而Destination IP应该是192.168.0.20，但接下来数据包并不是直接送到以太网来处理，而是送到PPP机制来处理③，并且在数据包的前端加上PPP包头。按照一般的做法，接下来数据包应该直接从网络实体层送离本机，但这里最特殊之处在于PPP会将数据包转发给L2TP机制④，并且在数据包的前面封装一个L2TP包头。

L2TP则又是一个更特殊的机制，L2TP会把整个数据包送回传输层⑤，接着在整个数据包的最前面再封装一个UDP包头，然后数据包会继续往下层发送到网络层⑥，这时数据包的最前端会再被封装一个IP包头，不过这个时候IP包头内的Source IP是客户端主机网络接口上

的IP。在这个示例中是10.0.1.200，而Destination IP将会是10.0.1.210，如此这个数据包将可以通过这两个公网IP，将数据包跨过因特网送到L2TP服务器上。但在数据包送达L2TP服务器之后，数据包又是如何送到192.168.0.20呢？当然这又是另一种技术，不过这里暂不讨论，稍后将会进行完整描述。

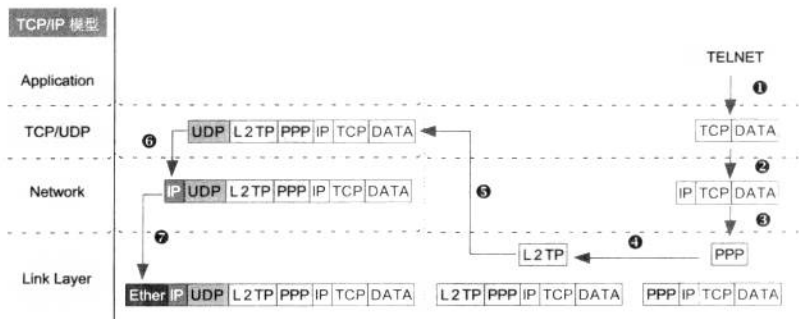


图16-3 L2TP的工作原理

在了解了整个L2TP协议的工作原理之后，再来看实际截取的L2TP协议下的数据包。从图16-4中的数据包结构可以看到，其最下方的位置应该就是应用层协议的部分，接下来被封装上一个TCP包头，再封装一个IP包头，从这里我们可以清楚地看到IP包头内的IP是一组私有IP，接着则是被封装上一个PPP包头，再被加上L2TP包头；然后我们可以看到数据包被送回传输层，并且封装上UDP包头，接着再封装上IP包头，这个IP包头内的IP也是一组公网IP，最后数据包被送到以太网来处理，最后送离本机。

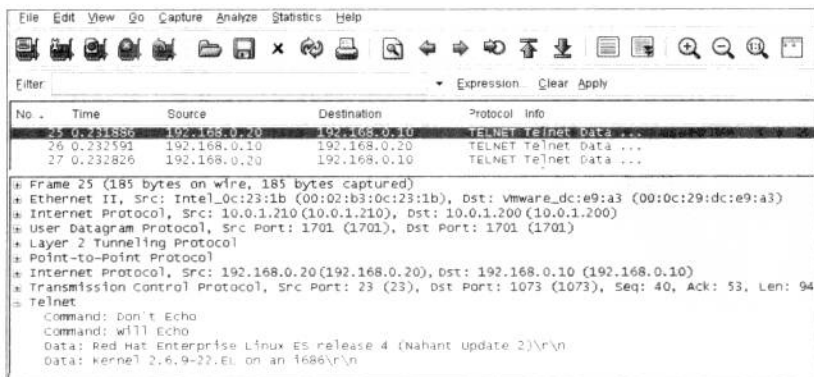


图16-4 L2TP协议的数据包结构

16.2.2 L2TP协议的安全问题

从图16-4可以清楚看到L2TP数据包内所承载的数据内容, 另外我们还要了解, 在L2TP拨号过程中, 需要进行帐户及密码的验证操作, 不过这个帐户密码的验证操作是由PPP协议来完成的, 图16-5就是PPP协议在进行验证操作时获取的数据包, 从数据包可以清楚看到传输中的帐户是jacky_chen, 或许你会觉得很奇怪, 如果真是这样, 那L2TP协议所建立的VPN也未免太不安全了吧! 不过, L2TP协议的确就是如此。

No.	Time	Source	Destination	Protocol	Info
27	5.026506	192.168.2.101	192.168.2.102	PPP LC	Code Reject
28	5.027231	192.168.2.102	192.168.2.101	PPP CH	Response
29	5.052501	192.168.2.101	192.168.2.102	PPP CH	Success (MESS)
30	5.052577	192.168.2.101	192.168.2.102	PPP CC	Configuration

* Frame 28 (83 bytes on wire, 83 bytes captured)
* Ethernet II, Src: VMware_dce9:a3 (00:0c:29:dc:e9:a3), Dst: Intel_0c:23:1
* Internet Protocol, Src: 192.168.2.102 (192.168.2.102), Dst: 192.168.2.101
* User Datagram Protocol, Src Port: 12tp (1701), Dst Port: 12tp (1701)
* Layer 2 Tunneling Protocol
* Point-to-Point Protocol
* PPP Challenge Handshake Authentication Protocol
Code: Response (2)
Identifier: 21
Length: 31
* Data (27 bytes)
value: size: 16
value: 1A9B0AE252874E54F08402E309155198
Name: jacky_chen

图16-5 PPP协议的验证

16.2.3 L2TP协议安全问题的解决方案

因为L2TP协议本身并没有考虑安全的问题, 如果仅仅用L2TP来建立企业的VPN, 那么笔者只能说“你真是勇气可嘉!”。在一般使用L2TP协议时, 绝对不会单独只使用L2TP协议, 通常都会在客户端主机与L2TP服务器之间建立传输模式的IPSec, 这样就可以在网络上安全地使用L2TP协议。

图16-6是L2TP协议与IPSec协议一起使用时的 workflows, 其实这个流程与单独使用L2TP协议的工作流程差不多, 而其中的差异就在于第⑥、⑦步骤之间, 在L2TP加上IPSec之后, 其数据包结构就如同图16-7所示。因为在第⑥步之前的所有内容都被加密了, 所以看起来与一般的IPSec数据包没什么不同, 这个方法我们称为L2TP Over IPSec。

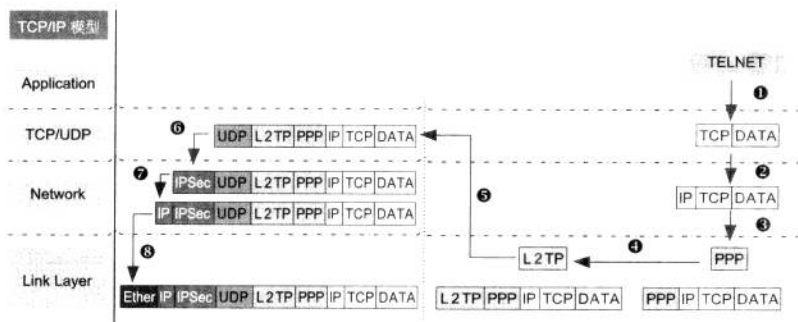


图16-6 L2TP Over IPSec

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.1.200	10.0.1.210	ESP	ESP (SPI=0x06492058)
2	0.000425	10.0.1.210	10.0.1.200	ESP	ESP (SPI=0x32d3a122)
3	0.000912	10.0.1.200	10.0.1.210	ESP	ESP (SPI=0x06492058)
4	0.001366	10.0.1.210	10.0.1.200	ESP	ESP (SPI=0x32d3a122)
5	2.001016	10.0.1.200	10.0.1.210	ESP	ESP (SPI=0x06492058)
6	2.001473	10.0.1.210	10.0.1.200	ESP	ESP (SPI=0x32d3a122)
7	3.000955	10.0.1.200	10.0.1.210	ESP	ESP (SPI=0x06492058)
8	3.001409	10.0.1.210	10.0.1.200	ESP	ESP (SPI=0x32d3a122)
# Frame 3 (142 bytes on wire, 142 bytes captured)					
# Ethernet II, Src: CompalE1_93:b2:7d (00:0f:b0:93:b2:7d), Dst: Intel_0c:23:1b (00:02:b3:0c:23:1b)					
# Internet Protocol, Src: 10.0.1.200 (10.0.1.200), Dst: 10.0.1.210 (10.0.1.210)					
# Encapsulating Security Payload					
SPI: 0x06492058					
Sequence: 61					
Data (100 bytes)					
0000	00 02 b3 0c 23 1b 00 0f	b0 93 b2 7d 08 00 45 00E.		
0010	00 80 e8 a5 00 00 80 32	cb 8c c0 a8 02 64 c0 a82....		
0020	02 65 06 49 20 58 00 00	00 3d 64 60 e6 b3 d3 38	e.I.....8		
0030	1a 00 18 2e 5d 61 0f 0f	7c 3a 18 e8 b6 31 97 cbf.....		
0040	44 4f 4c 3e 80 19 6e 94	76 fa d8 5f 5b ef 83 40	ZOLs..n.&[...8		
0050	be b8 5c 34 7f 9f ae 41	16 08 53 7b ab 3a b0 42	.4...A.S[...8		
0060	28 3f 0f 62 69 ac a3 dc	f2 1b 48 8d 1e 1d c9 2a	.b1...H...*		
0070	0c 47 bc d0 3d 42 90 43	5f b4 4b 62 0d d9 f7 ec	...JB...Kb...*		
0080	e2 ae c8 94 80 94 2d 1d	47 f3 7f 45 6d 73G..Enc		

图16-7 L2TP Over IPSec的数据包结构

16.2.4 Client to Site的L2TP VPN结构探讨

以图16-8为例来说明L2TP的完整架构及工作流程。首先，因为L2TP客户端要连接L2TP服务器③，会触发IKE机制来进行SA信息的交换①，在SA信息交换完之后，传输模式的IPSec也就建立完成②。接着L2TP客户端所传输的数据包就可以通过IPSec的保护，安全地送达L2TP服务器③。而L2TP服务器在收到L2TP客户端的请求之后，随即启动PPP服务器并生成一个PPP的接口④，而PPP接口上的IP则是由L2TP服务器所分配的，接着L2TP服务器应答给L2TP客户端，而这个数据包也是通过IPSec所加密保护的⑤。在L2TP客户端收到应答之后，L2TP客户端随即启动PPP客户端，在PPP客户端启动之后随即会生成一个PPP接口⑥，而PPP接口上的网络配置则是由L2TP服务器所分配的。在完成以上流程之后，L2TP VPN连接就建立起来了，如此我们就可以在客户端主机上，直接访问到192.168.0.10这个IP以及整个企业网络了。

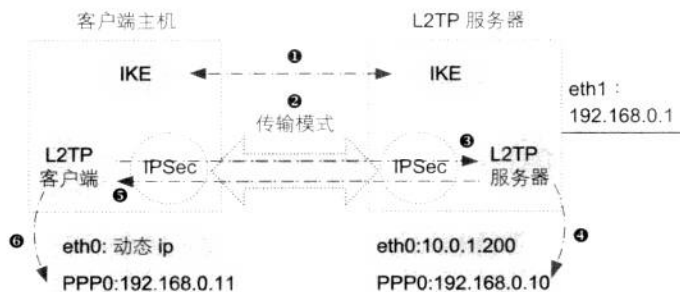


图16-8 Client to Site的L2TP结构

16.2.5 L2TP 客户端及服务器之间网段的选择

建立了L2TP连接后，其网络的逻辑结构就如图16-9所示。也就是说，L2TP服务器从某个角度来看，扮演了路由器的角色，但如果路由器左右两侧的网段相同，这个路由器就无法发挥它的作用。简单来说，就是路由器左右两侧的主机是无法互通的。但如果是这样，为什么这里会将L2TP客户端及L2TP服务器之间所使用的网段设置成与企业内部所使用的网段相同呢？



图16-9 L2TP连接的逻辑结构

关于以上这个问题，我们可能会有两种不同的考虑，如下说明：

- 将L2TP客户端与L2TP服务器之间设置成与企业内“不同”的网段：

这种情况下，我们不必担心路由器左右两侧网段相同的问题，但问题是在客户端主机拨号网络中必须设置“使用远程网络的默认网关”选项，否则在L2TP连接建立完成后，客户端主机无法与企业内部主机沟通，至于“使用远程网络的默认网关”是什么？稍后将进行详细说明。

- 将L2TP客户端与L2TP服务器之间设置成与企业内“相同”的网段：

某些情况下，我们可能不想启用“使用远程网络的默认网关”选项，但又想要正常访问企业内主机，这时就可以将L2TP客户端与L2TP服务器之间的网段设置成与企业内部网络相同的网段，但这样做会使得路由器左右两侧的网段是相同的。也就是说，建立了L2TP连接后，客户端主机还是无法访问到企业内部的主机，不过，在此可以使用Proxy ARP的机制来轻松解决这个问题。

16.2.6 Proxy ARP的工作原理

在建立了L2TP客户端与L2TP服务器之间的连接之后，其网络的逻辑结构就如图16-9所示。接下来分析一下，如果L2TP服务器左右两侧的网段相同，将会产生什么样的问题，以及Proxy ARP机制如何解决这个问题。

- 产生问题的原因：

首先我们假设L2TP客户端对企业内的文件服务器进行数据访问时，这个服务器请求数

据包内的Source IP是192.168.0.11, 而Destination IP则是192.168.0.60。因此, 这个数据包将由L2TP客户端的PPP0接口送出, 因为这是PPP连接, 所以由L2TP客户端PPP0接口送出去的数据包, 一定就是由连接的另一个端点来接收, 也就是由L2TP服务器的PPP0接口接收, 接着L2TP服务器会根据其上的路由表来决定如何处理数据包。但别忘了, 这个数据包的Source IP及Destination IP分别是192.168.0.11及192.168.0.60。如图16-10是L2TP服务器上的路由表, 根据路由表的判断, 这个数据包将由eth1接口送出, 并由192.168.0.60主机来接收, 如此L2TP客户端就可以正常地将数据包发送给企业内的文件服务器。

```
[root@localhost ~]# route -n
Kernel IP routing table
Destination      Gateway         Genmask         Flags    Metric  Ref    Use    Iface
192.168.0.11     0.0.0.0        255.255.255.255 UH        0        0        0      ppp0
192.168.0.0      0.0.0.0        255.255.255.0   U         0        0        0      eth1
10.0.1.200       0.0.0.0        255.255.255.0   U         0        0        0      eth0
0.0.0.0          10.0.1.254     0.0.0.0         UG        0        0        0      eth0
[root@localhost ~]#
```

图16-10 Proxy ARP的原理

不过, 当文件服务器收到这个数据包之后, 随即会对数据包内所记录的Source IP主机做出应答, 这个应答数据包内的Source IP及Destination IP将会是192.168.0.60以及192.168.0.11。但此时问题产生了, 当文件服务器要将数据包发送出去之前会发现, 目的端主机与本机在同一个网段中, 因此文件服务器会直接在网络上发出ARP的广播数据包, 试图查找到192.168.0.11主机网卡上的MAC地址。但因为广播数据包无法跨越路由器, 况且ARP的工作范围仅局限于单一的实体网段, 文件服务器也不可能取得192.168.0.11所对应的MAC地址, 当然文件服务器就无法将应答数据包送给L2TP客户端, 所以L2TP客户端是无法与文件服务器进行通信的。如果你忘了ARP协议的原理, 那么请回顾第1.2.1一节中的内容。

● 解决问题的办法:

想要解决以上的问题, 其实一点也不难。如图16-11所示, 我们只需在L2TP服务器上启动Proxy ARP机制, 就可以轻松解决。L2TP客户端与文件服务器无法通信的原因在于, 文件服务器无法把应答数据包正确地回送给发送端, 导致网络无法通信。不过当Proxy ARP机制启动后, 状况就有所改变了, Proxy ARP机制会代替L2TP客户端将数据包收下, 并将数据包交给L2TP服务器的路由表来决定数据包该如何处理。图16-10路由表第一行记录着如果数据包要送往192.168.0.11这台主机的话, 数据包就由PPP0接口送出, 既然数据包是由PPP0接口发送出去的, 那么这个数据包就一定会被192.168.0.11主机所接收, 如此一来网络自然就通了。



图16-11 Proxy ARP的原理

图16-12是使用Wireshark软件在L2TP服务器的eth1接口上所截取下来的数据包，从这个图可以证明Proxy ARP的存在。首先，当L2TP客户端发送第一个ICMP数据包给文件服务器时，这个数据包会通过PPP连接将数据包先发送到L2TP服务器，接着，L2TP服务器将数据包由eth1接口发送给文件服务器，此时L2TP服务器就会在eth1所临近的网段上发送ARP的广播，以找到IP 192.168.0.60所对应的MAC地址。图中第一个数据包的info字段就证明了这个过程，当然我们也可以从第一个数据包中得知L2TP服务器eth1接口上的MAC地址是00:50:56:C0:00:01，接着192.168.0.60主机随即通过ARP协议来应答其所对应的MAC地址，如图中的第二个数据包，在L2TP服务器知道192.168.0.60主机网卡的MAC地址之后，随即即将L2TP客户端所要传输的ICMP数据包发送给192.168.0.60这台主机。

No.	Time	Source	Destination	Protocol	Info
1	0.000000	00:50:56:c0:00:01	Broadcast	ARP	who has 192.168.0.60? Tell 192.168.0.1
2	0.000059	00:0c:29:b9:30:c7	00:50:56:c0:00:01	ARP	192.168.0.60 is at 00:0c:29:b9:30:c7
3	0.000075	192.168.0.11	192.168.0.60	ICMP	Echo (ping) request
4	0.000658	00:0c:29:b9:30:c7	Broadcast	ARP	who has 192.168.0.11? Tell 192.168.0.60
5	0.347501	00:50:56:c0:00:01	00:0c:29:b9:30:c7	ARP	192.168.0.11 is at 00:50:56:c0:00:01
6	0.347731	192.168.0.60	192.168.0.11	ICMP	Echo (ping) reply
7	1.040210	192.168.0.11	192.168.0.60	ICMP	Echo (ping) request
8	1.040528	192.168.0.60	192.168.0.11	ICMP	Echo (ping) reply
9	2.059801	192.168.0.11	192.168.0.60	ICMP	Echo (ping) request
10	2.059987	192.168.0.60	192.168.0.11	ICMP	Echo (ping) reply

图16-12 Proxy ARP的工作原理

在192.168.0.60主机收到这个数据包后，随即应答数据包给192.168.0.11，但因为192.168.0.60主机发现目的端主机与自己在同一个网段，因此192.168.0.60主机随即在网络发送ARP的广播数据包，以查找192.168.0.11这个IP所对应的MAC地址，如图中第四个数据包。但接下来神奇的事情发生了，192.168.0.11并不属于这个实体网段的IP。但这时确实有主机来应答这个ARP广播，如图中的第五个数据包。不过请你仔细看一下，第五个数据包的应答信息却是：“我是192.168.0.11，而且我的MAC地址是00:50:56:C0:00:01”，不过00:50:56:C0:00:01不就是L2TP服务器eth1接口的MAC地址吗？接下来，因为192.168.0.60这台主机确定192.168.0.11这个IP所对应的MAC地址是00:50:56:C0:00:01，因此192.168.0.60主机随即即将应答数据包送给L2TP服务器，接着L2TP服务器根据路由表将数据包转发给L2TP客户端，如此通过Proxy ARP的帮助，L2TP客户端及L2TP服务器就可以正常通信了。

16.3 构建L2TP VPN

这里以图16-13为例来构建L2TP VPN环境。在图中我们假设L2TP服务器 eth0接口的IP是10.0.1.200，eth1接口上的IP是192.168.0.1，另外在企业内部有一台文件服务器，其IP是192.168.0.20；至于客户端的部分，则以目前市场占有率最高的Windows XP/Windows 7为例，客户端的网络配置不做限制，只要能与10.0.1.200正常通信即可。

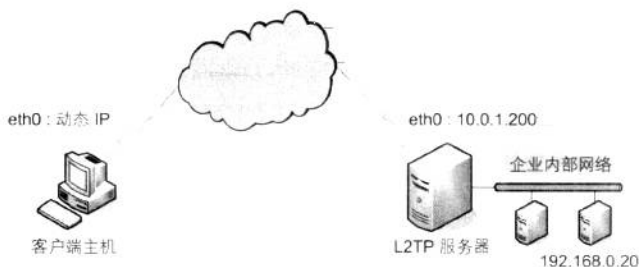


图16-13 L2TP VPN部署实例

16.3.1 配置L2TP服务器

首先介绍L2TP服务器的配置，L2TP服务器在开源领域有很多不同的软件可供选择。这里选用的是XL2TPD这个软件，你可以从XL2TPD的官方网站下载，其网址是[ftp://ftp.xelerance.com/xl2tpd/binaries/](http://ftp.xelerance.com/xl2tpd/binaries/)。请选择较新的版本，下载之后，直接使用rpm命令安装即可，安装方法如下。

```
[root@localhost tmp]# rpm -ivh xl2tpd-1.1.09-1.i386.rpm
warning: xl2tpd-1.1.09-1.i386.rpm: Header V3 RSA/SHA1 signature: NOKEY, key ID b5cc27e1
Preparing...                                [100%]
 1:xl2tpd                                    [100%]
[root@localhost tmp]#
```

安装完毕后，接下来设置XL2TPD服务器的配置文件。XL2TPD服务器的配置文件是/etc/xl2tpd/xl2tpd.conf，请将其内容调整如下。

```
[global]
listen_addr = 10.0.1.200
port = 1701

[ins_default]
ip_range = 192.168.0.241-192.168.0.254
```



```
local ip = 192.168.0.240
name = xl2tp-server
pppoptfile = /etc/ppp/options.xl2tpd
```

其中各个参数的含义如下。

- listen-addr

指定XL2TPD启动后，要在哪个IP上进行工作。

- port

指定XL2TPD启动后，要在哪个端口上进行工作；不过请注意，L2TP是属于UDP协议的服务，而且L2TP服务器的标准端口是1701。

- ip range

在L2TP VPN架构中，客户端的IP是由L2TP服务器分配的。本示例将VPN网段的IP设置为与企业内部相同，且分配的IP范围是192.168.0.241到192.168.0.254之间，也就是说最多可以有14个L2TP客户端同时上线，不过既然我们已经将这个网段的IP分配给VPN客户端来使用，所以在企业内部就不能将这些IP分配给其他主机。

- local ip

ip range参数定义了VPN客户端的IP，也就是说目前只定义了PPP连接其中一个端点的IP，至于另一个端点的IP就由local ip这个参数来定义。本示例假设为192.168.0.240，因此当VPN客户端上线时，我们就可以在L2TP服务器上看到一个或多个的PPP接口，而且这些接口的IP都会是192.168.0.240。

- name

name参数只是给这个服务设置一个名称，而这个名称会在帐户数据库中使用。稍后将完整介绍这方面的内容。

- pppoptfile

在L2TP架构中必须与PPP协议协同工作，而PPP是由XL2TPD来调用的，因此我们必须告诉XL2TPD要用哪个PPP的配置文件来执行PPPD。本示例将PPPD的运行参数定义在/etc/ppp/options.xl2tpd文件中。

在配置好所有这些参数后，我们就可以使用以下方法来启动或停止XL2TPD。

```
[root@localhost ~]# service xl2tpd-restart
Stopping xl2tpd: [ 确定 ]
Starting xl2tpd: [ 确定 ]
[root@localhost ~]#
```

16.3.2 配置PPP服务器

PPP服务器并没有固定的配置文件，其配置文件的名称是由XL2TPD来设置的。本示例将其设置为/etc/ppp/options.xl2tpd，请将这个文件的内容设置如下：

```
auth
proxyarp
ms-wins 192.168.0.4
ms-dns 192.168.0.3
```

下面解释各个参数的含义。

- auth

在传输数据时，先进行验证操作。

- proxyarp

启用ProxyARP的机制。

- ms-wins

将企业内部的Windowss服务器的IP配置到VPN客户端的网络参数中。

- ms-dns

将企业内部的DNS服务器的IP配置到VPN客户端的网络参数中。

16.3.3 建立VPN的拨号帐户

在L2TP VPN的架构中，使用者的身份验证由PPP协议来完成。PPP协议的验证方法有很多种，例如：PAP、CHAP、MSCHAP及MSCHAP-V2。在这几种验证方法中，我们大概可以将其分为两类，其一是以明文方式来进行验证，例如PAP就是属于这一类；其二是验证过程中，帐户及密码是受到保护的，例如：CHAP、MSCHAP及MSCHAP-V2。另外由于系统帐户数据库的密码是经过加密的，使得PPP服务器无法使用/etc/passwd或/etc/shadow作为其帐户数据库。

如果我们选择PAP验证方法，那么帐户数据库将是/etc/ppp/pap-secrets，不过目前大概没有人会选择PAP，因为PAP在网络上直接传输密码，又无任何的保护措施；那如果我们选择PAP以外的验证方法，帐户数据库将会是/etc/ppp/chap-secrets这个文件，如果要新增VPN的拨号帐户，只要直接编辑这个文件就可以了，其格式如下。

```
# Secrets for authentication using CHAP
# client      server  secret          IP addresses
jacky_chen    xl2tp-server  12345           192.168.0.0/24
david_chang   xl2tp-server  54321           *
```

这个文件共有四个字段，其含义分别如下：

- 字段一：

指定使用者的帐户名称。

- 字段二：

指定这组帐户是要用在哪个服务中。例如，在L2TP服务器上同时提供了L2TP VPN及调制解调器的拨号服务，就可以用这个字段来设置这组帐户是要给L2TP VPN还是调制解调器拨号服务来使用。而这个“字符串”是由启动PPP服务器的应用程序来设置。例如在L2TP VPN架构中，PPP服务器是由XL2TPD来启动，所以在XL2TPD的配置文件/etc/xl2tpd/xl2tpd.conf中必须去设置这个“字符串”，在XL2TPD这个应用程序中则使用name这个参数来设置。此外，我们还可以在这个字段上填写“*”，表示这个帐户可以用于本机上所有的PPP服务。

- 字段三：

指定帐户对应的密码，密码是直接以明文形式来保存的。

- 字段四：

限制帐户可以从哪些IP地址连接上来，这里可以设置的格式有：10.0.1.50、10.0.1.0/24，或者用*(表示任意一个IP)。

16.3.4 证书的生成及保存

关于L2TP服务器上证书的生成及保存方法，与第15.5一节中介绍的完全相同，这里就不再重复说明，但这里要先提醒你一件事，在Linux系统上是使用OpenSSL这个软件来进行CA的工作。不过根据我实际测试的结果发现，如果是使用RedHat Enterprise Linux 5.X及CentOS 5.X光盘中自带的OpenSSL软件，那么OpenSSL所生成的证书在Linux平台上使用并不会有问题，但如果是在Windows平台上使用就会有问题。因此建议你你可以选择RedHat Enterprise Linux 4.X/6.X或CentOS 4.X/6.X系统来作为CA的操作系统，因为在这几个版本的系统上并不会出现证书无法在Windows系统中使用的问题。

16.3.5 配置安全策略

以下是L2TP服务器上安全策略的配置文件/etc/racoon/setkey.conf，这个配置文件的内容比较特殊，一般在设置安全策略时，总是要考虑哪些“送入”及“送出”的数据需要使用IPSec，因此安全策略通常会有两条，但在L2TP VPN的结构中，L2TP客户端的IP通常不会是固定的，我们也无法预先设置双向安全策略，最多只能设置单向安全策略。

```

1. flush;
2. spdflush;
3.
4. spdadd 10.0.1.200[1701] 0.0.0.0/0[0] any -P out ipsec
5.      esp/transport//require;

```

在说明以上示例之前请先思考两个问题，由于L2TP协议使用UDP 端口1701，因此对于L2TP服务器而言，我们只需要让与端口1701有关的数据使用IPSec即可；另外因为L2TP客户端的IP不是固定的，因此无法预先设置好L2TP客户端对L2TP服务器这个方向的安全策略，不过可以肯定的是，如果客户端访问L2TP服务器的UDP端口1701，L2TP服务器一定是使用UDP端口1701来应答，所以我们可以定义凡是从L2TP服务器的端口1701发送出去的数据包，不管对象是谁、目的端口是什么，这些数据全部都经由IPSec，因此我们定义了第4行的安全策略。

安全策略一定要定义双向的，才能让L2TP Over IPSec正常工作，不过使用现有的信息，我们只能建立单向安全策略，那另一个方向的安全策略呢？其实这个问题很容易解决，因为IKE配置文件中提供了一个名为generate_policy的参数，我们只需将这个参数设置为on，IKE就会自动生成另一个方向的安全策略。

16.3.6 IKE配置文件

在这个步骤中，我们将设置IKE的配置文件/etc/racoon/racoon.conf，这个配置文件的内容与第15.5一节中所介绍的内容几乎完全相同，唯一的差别是第6行的“generate_policy”参数，这个参数在上一节有特别的说明，这里就不再赘述。

```

1. path include "/etc/racoon";
2. path pre_shared_key "/etc/racoon/psk.txt";
3. path certificate "/etc/racoon/certs";

4. remote anonymous {
5.     exchange_mode          main;
6.     generate_policy         on;
7.     certificate_type        x509 "server.pem" "server.key";
8.     verify_cert             on;
9.     my_identifier           asnldn;
10.    peers_identifier         asnldn;
11.    proposal {
12.        authentication_method    rsasig;
13.        encryption_algorithm     3des;
14.        hash_algorithm           sha1;

```

```

15.             dh group                      modp1024;
16.         )
17.     }

18.     sainfo_anonymous {
19.         lifetime time                      1 hour;
20.         encryption_algorithm              3des;
21.         authentication_algorithm          hmac_sha1;
22.         compression_algorithm             deflate;
23.     }

```

16.3.7 启动L2TP服务器

L2TP服务器是由多个功能模块所组合而成，因此，我们必须将其逐次启动，启动过程没有先后之分，分别如下。

● 启动XL2TP守护程序

```
[root@localhost ~]#service xl2tpd { start | stop | restart }
```

● 启动racoon守护程序

```
[root@localhost ~]#service racoon { start | stop | restart }
```

● 配置安全策略

```
[root@localhost ~]#setkey -fw/etc/racoon/setkey.conf
```

16.4 配置L2TP客户端

目前市场上的绝大多数操作系统几乎都能支持L2TP协议，如Windows 9x/NT/Win2K/WinXP/Win2003/Win Vista/Win7、MAC、Windows CE、Linux等，在此仅以使用者较多的Windows XP及Windows 7为例进行介绍，其余部分请你自己参考其他资料。

16.4.1 生成L2TP客户端证书

由于我们使用较为安全的证书验证方式，因此必须在L2TP客户端主机上安装证书，首先请在CA上为L2TP客户端生成证书，然后再将证书安装到L2TP客户端主机上，至于生成服务器证书的过程，就不再重复说明了，你可以参阅第15.4.2一节的内容。另外证书生成过程中所需填写的资料，只要你觉得没问题就可以了，因为这些资料与VPN能否成功建立无关。

只要切记证书的有效时间及VPN主机的时间必须合理。

这里假设将生成的证书及私钥分别保存为client.pem、client.key两个文件，不过Windows系统并不支持这种文件格式，因此我们必须先将证书及私钥转换成Windows系统所支持的PKCS12的文件格式。这种文件中包含“证书”及“私钥”，其转换方法如下。

```
[root@localhost misc]# openssl pkcs12 -export -in client.pem -out
client.p12 -inkey client.key ❶
Enter pass phrase for private.key: ❷
Enter Export Password: ❸
Verifying - Enter Export Password ❹
```

其中标记❶的-out client.p12是转换后的文件名，另外文件client.pem是私钥，因为私钥默认情况下设置密码保护，如果我们没有把私钥的密码告诉openssl工具，那么openssl将无法读取我们的私钥，当然也就转换不成功。因此，必须告诉openssl私钥的密码❷，在client.p12文件生成之后，openssl工具会要求我们提供一个密码来保护client.p12的文件内容❸❹，等我们将client.p12文件导入Windows系统，Windows系统就会要求提供这个密码，否则导入操作将无法完成。

16.4.2 将证书导入Windows XP/7系统前的准备工作

在将证书导入Windows XP/7系统之前，我们必须先准备好CA的证书及L2TP客户端的证书文件，在此假设这两个证书的文件分别是cacert.pem及client.p12，由于Windows系统不识别.pem文件，因此将cacert.pem的后缀改为.crt，即cacert.crt，这样Windows系统才能识别这个文件。

16.4.3 设置Windows XP系统上的L2TP客户端

下面将以Windows XP系统为例来说明L2TP客户端的设置方法。

1. 启动及配置MMC

在Windows XP上必须借助于MMC工具导入证书，而MMC是一个多功能工具。在使用MMC之前，必须先配置好MMC工具，配置方法如下：

步骤01：选择“开始”|“运行”，并输入mmc命令，单击“确定”，如图16-14所示。

步骤02：MMC管理工具的初始界面如图16-15所示。

步骤03：配置MMC，选择“文件”|“添加/删除管理单元”，如图16-16所示。

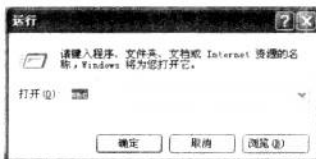


图16-14 运行mmc命令



图16-15 MMC的初始界面

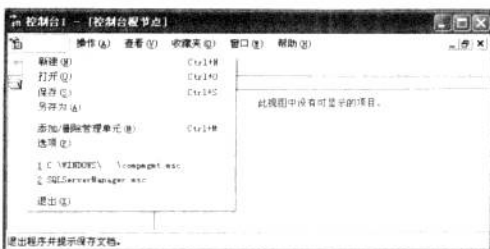


图16-16 配置MMC

步骤04: 单击“添加”按钮, 增加MMC的管理单元, 如图16-17所示。

步骤05: 请选择“证书”|“添加”, 以增加证书管理单元, 如图16-18所示。

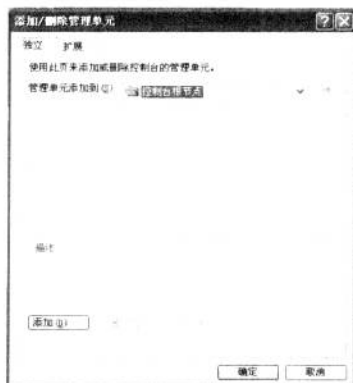


图16-17 添加MMC管理单元



图16-18 添加证书管理单元

步骤06: 选择“计算机帐户”的管理单元, 选择“下一步”, 如图16-19所示。

步骤07: 请选择“本地计算机”的管理单元, 单击“完成”按钮, 如图16-20所示。



图16-19 选择“计算机帐户”的管理单元

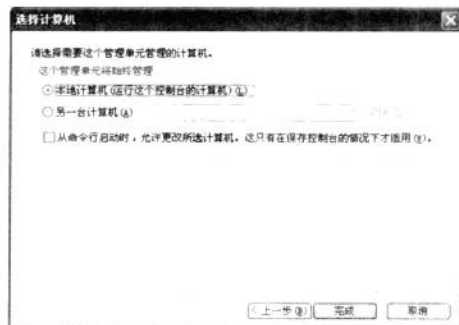


图16-20 选择“本地计算机”的管理单元

步骤08: 单击“关闭”, 结束添加管理单元, 如图16-21所示。

步骤09: 单击“确定”按钮, 结束MMC的配置, 如图16-22所示。



图16-21 结束添加管理单元

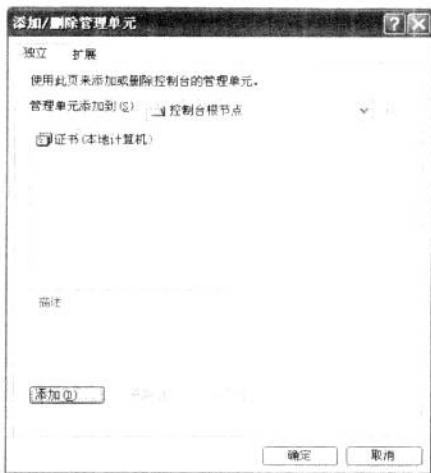


图16-22 结束MMC的配置

2. 导入“计算机”及“CA”的证书

配置完MMC工具之后, 接下来导入计算机证书及CA的证书, 步骤如下:

步骤01: 选择“证书(本地计算机)”|“个人”, 右击选择“所有任务”|“导入”, 如图16-23所示。

步骤02: 单击“下一步”按钮, 开始导入计算机证书, 如图16-24所示。



图16-23 进入“导入”界面

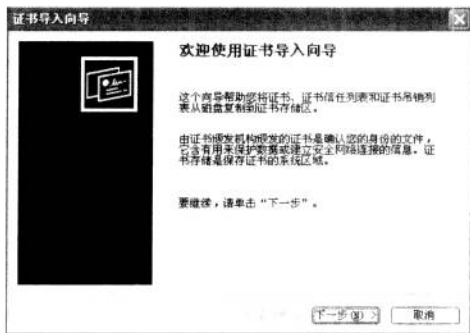


图16-24 进行计算机证书的导入操作

步骤03: 单击“浏览”按钮, 指定证书的路径和文件名, 如图16-25所示。

步骤04: 输入client.p12文件的密码, 如图16-26所示。



图16-25 设置计算机证书的路径及文件名



图16-26 输入证书的保护密码

步骤05: 设置证书所要导入的位置, 这里请选择“个人”, 如图16-27所示。

步骤06: 请单击“完成”按钮, 结束计算机证书的导入操作, 如图16-28所示。

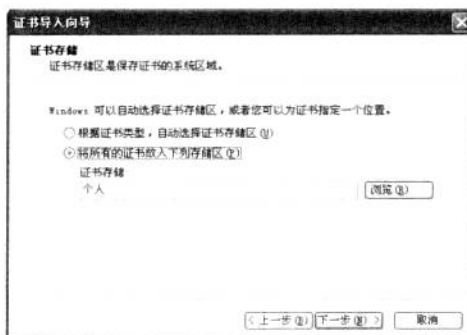


图16-27 设置证书的保存位置



图16-28 完成导入操作

步骤07: 在计算机证书导入完毕之后, 接下来就是导入CA的证书, 请选择“证书(本地计算机)”|“受信任的根证书颁发机构”|“证书”, 右击“证书”, 选择“所有任务”|“导入”, 如图16-29所示。

步骤08: 单击“下一步”, 进行CA证书的导入操作, 如图16-30所示。



图16-29 导入CA证书

步骤09: 在“文件名”中输入CA证书的路径及文件名, 单击“下一步”按钮, 继续证书的导入操作, 如图16-31所示。



图16-30 进行CA证书的导入操作



图16-31 设置CA证书文件的路径及文件名

步骤10: 选择证书的存储区, 这里请选择“受信任的根证书颁发机构”, 单击“下一步”, 如图16-32所示。

步骤11: 单击“完成”, 结束证书的导入操作, 如图16-33所示。

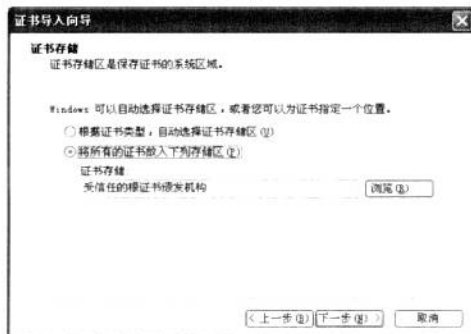


图16-32 选择证书的存储区



图16-33 结束证书的导入操作

3. 在Windows XP系统中配置拨号网络

步骤01: 选择“开始”|“控制面板”|“网络连接”|“新建连接向导”, 如图16-34所示。

步骤02: 单击“下一步”, 进行拨号网络的设置, 如图16-35所示。

步骤03: 选择“连接到我的工作场所的网络”, 单击“下一步”按钮, 如图16-36所示。



图16-34 建立一个新的网络连接



图16-35 进行拨号网络的配置

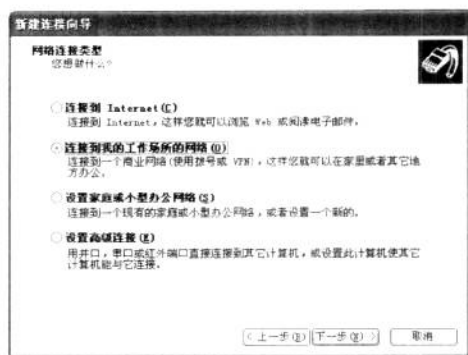


图16-36 设置网络连接类型

步骤04: 请选择“虚拟专用网络连接”, 单击“下一步”, 如图16-37所示。

步骤05: 设置一个连接名称, 以便以后使用, 单击“下一步”, 如图16-38所示。

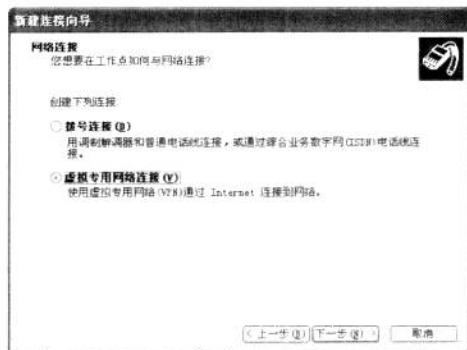


图16-37 设置网络连接方式



图16-38 设置网络连接的名称

步骤06: 设置L2TP服务器的IP或主机名称都可以, 单击“下一步”, 如图16-39所示。

步骤07: 如果希望将拨号网络的图标显示在桌面上, 就请选中“在我的桌面上添加一个到此连接的快捷方式”, 接着单击“完成”按钮, 即可完成拨号网络的设置, 如图16-40所示。

步骤08: 完成L2TP的拨号网络设置之后, 在“网络连接”面板里面可以看到多了一个“虚拟专用网络”的拨号选项, 如图16-41所示。

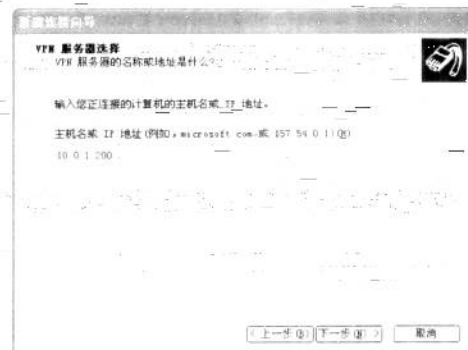


图16-39 设置L2TP服务器的主机地址



图16-40 完成新建连接的操作



图16-41 “网络连接”面板中的内容

步骤09：然后需要连接L2TP VPN时，请双击刚生成的网络连接图标，在出现的界面上输入拨号的帐户及密码即可，如果希望保存密码，请选中“为下面用户保存用户名及密码”，如图16-42所示。

4. 拨号网络中默认网关的问题

如图16-43所示，在拨号网络的高级设置中有一个“在远程网络上使用默认网关”选项，这个选项选择与否的差异，如图16-44及图16-45所示。



图16-42 输入拨号者的帐户及密码



图16-43 拨号网络的默认网关



图16-44 没有选择“在远程网络上使用默认网关”

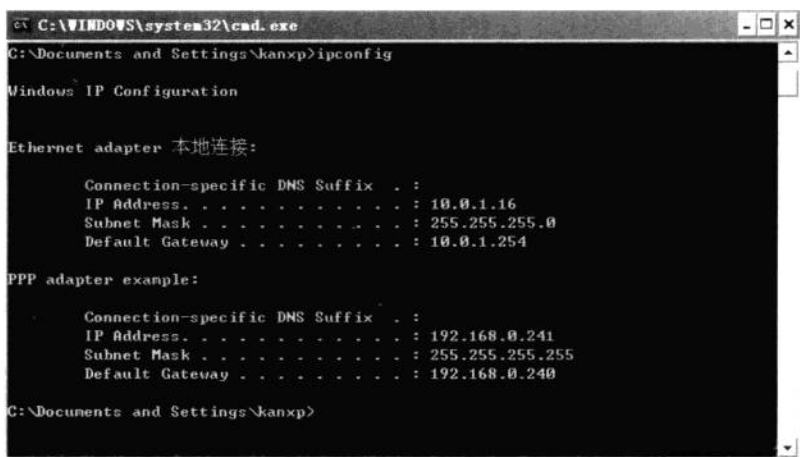


图16-45 选择了“在远程网络上使用默认网关”

首先来看不选择“在远程网络上使用默认网关”的情况，在这种情况下，我们必须在L2TP服务器上，把客户端主机到L2TP服务器这个网段设置为与企业内部相同，再启动Proxy ARP来解决两个实体网段间的问题。从图16-44可以发现，如果数据包不是发送到PPP接口所在的网段，那么所有数据包将全部往10.0.1.254这个网关发送，如果假设客户端主机到L2TP服务器这一个网段的IP是192.168.2.0/24，那么客户端主机将不会知道192.168.0.0/24的网段在L2TP服务器后面，而所有送到192.168.0.0/24网段的数据包也会全部从10.0.1.254这个网关发送，因此虽然VPN拨通了，但客户端主机也一样无法与企业内部网段通信。

如果选择了“在远程网络上使用默认网关”，那么客户端主机到L2TP服务器这个网段的IP就不需要与企业内部相同，从图16-45可以发现，在PPP接口上的网络配置中多了一个Default Gateway值，而这个值将会暂时取代网卡上所记录的Default Gateway。这样不管客户端主机要发送数据包到哪里，所有数据包都会通过PPP连接，送到192.168.0.240这个网关，而这个网关就是L2TP服务器PPP接口的IP。因此如果是送往企业内部网络的数据包，都会通过L2TP服务器上的路由表转发到企业内部；如果不是送往企业内部的数据包，就会通过L2TP服务器上的路由表，将数据包转发到因特网。

综合以上两种情况我们不难发现，选择或者不选择“在远程网络上使用默认网关”都有一些优点，如下说明。

- 在远程网络上不使用默认网关：

这种设置的优点是，如果数据包不是送往企业内部网络的，那么数据包就直接往因特网发送，而不需要先通过L2TP Tunnel送到L2TP服务器，然后再转发到因特网，因此效率较高。但如果这个客户端主机位于一个开放的网络环境中，那么客户端主机往企业内部以外的地方传输数据，便无法用IPSec的加密保护，它所传输的数据内容就可能被他人窃取。

- 在远程网络上使用默认网关：

在这种情况下，无论数据包是要送往哪里，都必须先经过L2TP Over IPSec Tunnel的通道绕回企业，因此网络传输性能比较差，但如果客户端主机是位于一个开放的网络环境中，也不用担心其所传输的数据会被他人所窃取。因此在这种情况下，客户端主机传输的任何数据都能受到L2TP Over IPSec Tunnel的保护。

16.4.4 设置Windows 7系统中的L2TP客户端

下面将以Windows 7系统为例来说明L2TP客户端的设置方法。

1. 启动及配置MMC

如同Windows XP中使用MMC一样，在Windows 7中导入证书也通过MMC工具来完成。我们必须先配置好MMC工具，配置方法如下。

步骤01：请单击Windows 7界面左下角的Windows图标，在“搜索程序和文件”中输入“mmc”，回车执行mmc命令，如图16-46所示。



图16-46 启动MMC工具

步骤02：MMC管理工具的初始界面，如图16-47所示。



图16-47 MMC的初始界面

步骤03: 配置MMC，选择“文件”|“添加/删除管理单元”，如图16-48所示。



图16-48 配置MMC

步骤04: 在“可用的管理单元”中选择“证书”，单击“添加”，单击“确定”，如图16-49所示。



图16-49 添加MMC的管理单元

步骤05：选择“计算机帐户”，单击“下一步”，如图16-50所示。



图16-50 设置“计算机帐户”的管理单元

步骤06：选择“本地计算机”，单击“完成”，如图16-51所示。

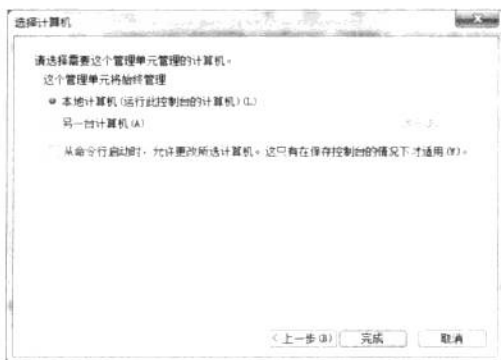


图16-51 设置“本地计算机”的管理单元

步骤07：完成MMC工具的证书管理配置，单击“确定”，如图16-52所示。



图16-52 完成MMC工具的证书管理配置

步骤08: 配置完成后的MMC管理界面, 如图16-53所示。



图16-53 配置完毕后的MMC管理界面

2. 导入“计算机”及“CA”的证书

在配置完MMC工具后, 接下来导入计算机证书及CA证书, 步骤如下。

步骤01: 选择“控制台根节点”|“证书”|“个人”, 右击“个人”|“所有任务”|“导入”, 如图16-54所示。

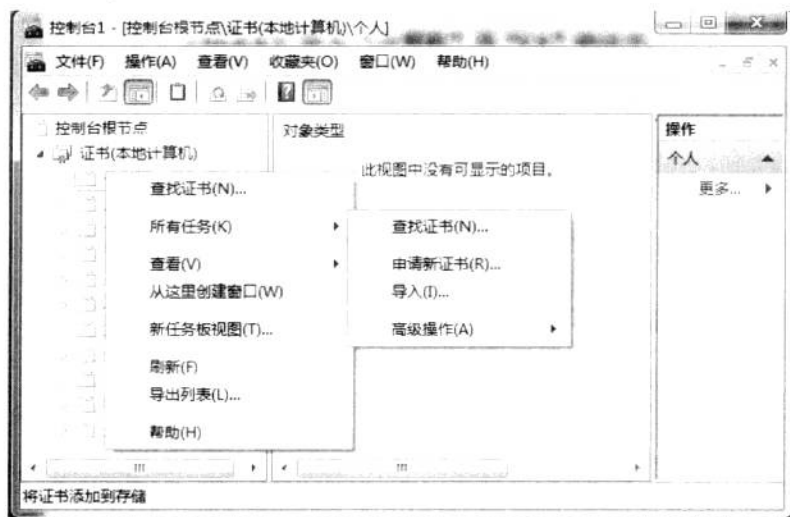


图16-54 导入计算机证书界面

步骤02: 单击“下一步”导入计算机证书, 如图16-55所示。

步骤03: 单击“浏览”按钮, 指定证书的路径和文件名, 如图16-56所示。

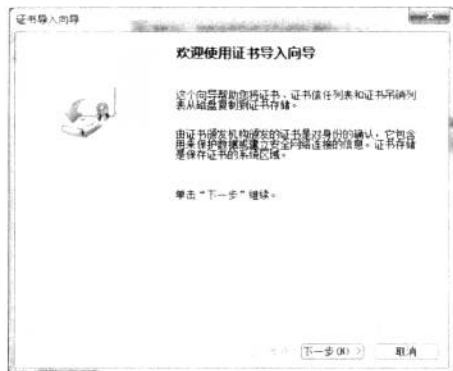


图16-55 进行计算机证书的导入操作



图16-56 设置证书所在的路径及文件名

步骤04: 输入保护client.p12文件的密码, 单击“下一步”, 如图16-57所示。

步骤05: 选择证书的存储位置, 这里请选择“个人”, 单击“下一步”, 如图16-58所示。



图16-57 输入证书的导入密码



图16-58 设置证书所要导入的位置

步骤06: 单击“完成”, 完成证书的导入, 如图16-59所示。

步骤07: 计算机的证书导入完成之后, 接下来就是导入CA的证书, 请选择“控制台根节点”|“证书(本地计算机)”|“受信任的根证书颁发机构”|“证书”, 右击选择“证书”|“所有任务”|“导入”, 如图16-60所示。



图16-59 完成“导入”操作

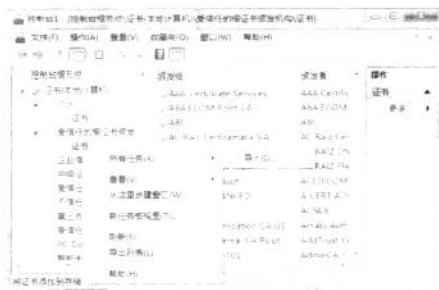


图16-60 导入CA证书

步骤08: 单击“下一步”，进行CA证书的导入操作，如图16-61所示。

步骤09: 单击“浏览”按钮，指定CA证书的路径及文件名，单击“下一步”，如图16-62所示。



图16-61 进行CA证书的导入操作

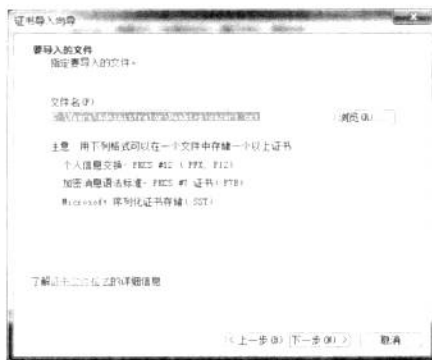


图16-62 设置CA证书的路径及文件名

步骤10: 选择证书的存储位置，请选择“受信任的根证书颁发机构”，接着单击“下一步”，如图16-63所示。

步骤11: 单击“完成”，完成证书的导入操作，如图16-64所示。



图16-63 配置CA证书的导入位置



图16-64 完成证书的导入操作

3. 在Windows 7系统中配置拨号网络

步骤01: 选择“开始”|“控制面板”|“网络与共享中心”|“设置新的连接或网络”，如图16-65所示。

步骤02: 选择“连接到工作区”，单击“下一步”，如图16-66所示。

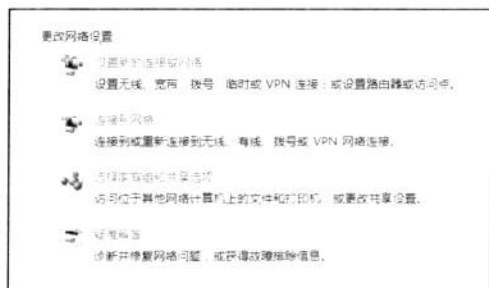


图16-65 建立一个新的L2TP连接



图16-66 选择连接选项

步骤03: 选择新建的L2TP连接是否要建立在其他的连接上，例如在拨号L2TP连接之前可能需要先进行PPPOE的网络连接，这里要根据你的实际情况来决定，如图16-67所示。

步骤04: 在这个示例中，我们选择使用已有的因特网连接来进行L2TP连接拨号，如图16-68所示。



图16-67 设置网络连接类型



图16-68 设置网络连接方式

步骤05: 设置L2TP服务器的网络地址及新建立的L2TP连接名称，单击“下一步”，如图16-69所示。

步骤06: 设置L2TP拨号的帐户及密码，单击“连接”，如图16-70所示。

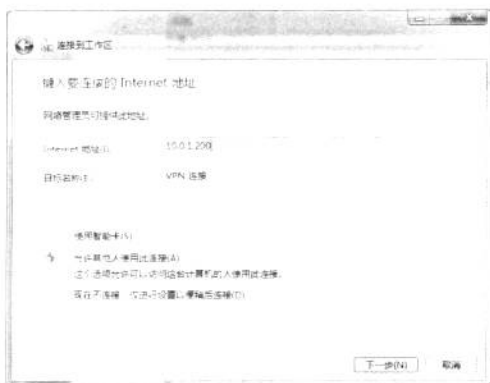


图16-69 设置L2TP服务器的IP及连接名称



图16-70 设置L2TP拨号的帐户及密码

步骤07: 完成L2TP拨号网络的设置之后,我们就可以在网络连接功能表中看到新建立的VPN连接,如图16-71所示。

步骤08: 双击“VPN连接”图标,可以看到L2TP拨号界面,如图16-72所示。

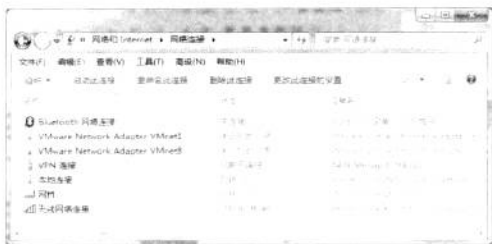


图16-71 完成新建连接的操作



图16-72 L2TP的拨号界面

4. Windows 7拨号网络中的网关设置

如同在Windows XP一样,Windows 7拨号网络中同样存在默认网关选项,如图16-73所示,在拨号网络的高级设置中有一个“在远程网络上使用默认网关”的选项,这个选项选择与否的差别,如图16-74及图16-75所示。



图16-73 拨号网络的默认网关

```

C:\Users\Administrator>ipconfig

Windows IP 配置

PPP 接口 VPN连接:

    连接特定的 DNS 后缀 . . . . . : 
    IPv4 地址 . . . . . : 192.168.0.241
    子网掩码 . . . . . : 255.255.255.255
    默认网关. . . . . : 

以太网适配器 网桥:

    连接特定的 DNS 后缀 . . . . . : corp
    IPv4 地址 . . . . . : 10.0.1.200
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 10.0.1.254
  
```

图16-74 没有选中“在远程网络上使用默认网关”选项

```

C:\Users\Administrator>ipconfig

Windows IP 配置

PPP 接口 VPN连接:

    连接特定的 DNS 后缀 . . . . . : 
    IPv4 地址 . . . . . : 192.168.0.241
    子网掩码 . . . . . : 255.255.255.255
    默认网关. . . . . : 0.0.0.0

以太网适配器 网桥:

    连接特定的 DNS 后缀 . . . . . : corp
    IPv4 地址 . . . . . : 10.0.1.200
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 10.0.1.254
  
```

图16-75 选中“在远程网络上使用默认网关”选项

16.5 IPSec连接穿透NAT的问题

IPSec数据包在穿越NAT时会遇到一个很严重的问题，因为NAT机制会修改数据包的IP及TCP或UDP包头内容，而IPSec为了确保数据包的安全性，也会逐一检查每个IPSec的数据包，一旦数据包内容(包含包头)有任何变动，这些数据包就会被IPSec机制丢弃，所以IPSec数据包是无法穿越NAT的。但如果真是这样，那IPSec在使用上将受到严重限制，幸运的是在大家的努力下，IPSec协议终于可以穿越NAT了，这项技术规范是在rfc3947中定义的，称为“NAT-Traversal”，简称为NAT-T。那NAT-T到底是如何让IPSec数据包可以穿越NAT的呢？这里以图16-76及图16-77为例来说明。

22 13.906161	10.10.15.51	10.10.15.40	ESP	ESP (SPI=0x0bc487d8)
23 13.906331	10.10.15.40	10.10.15.51	ESP	ESP (SPI=0xeb1b68f4)

▶ Frame 20: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
 ▶ Ethernet II, Src: AsustekC a7:ad:6b (48:5b:39:a7:ad:6b), Dst: RealtekU af:03:c7 (52:54:00:af:03:c7)
 ▶ Internet Protocol, Src: 10.10.15.40 (10.10.15.40), Dst: 10.10.15.51 (10.10.15.51)
 ▶ Encapsulating Security Payload

图16-76 非NAT-T的IPSec数据包结构

5 3.946543	192.168.122.40	10.10.15.40	ESP	ESP (SPI=0x0309d3ec)
6 5.436880	192.168.122.40	10.10.15.40	ESP	ESP (SPI=0x0309d3ec)
7 6.021954	192.168.122.40	10.10.15.40	ESP	ESP (SPI=0x0309d3ec)

▶ Frame 4: 158 bytes on wire (1264 bits), 158 bytes captured (1264 bits)
 ▶ Ethernet II, Src: RealtekU 8b:97:0e (52:54:00:8b:97:0e), Dst: fe:54:00:8b:97:0e (fe:54:00:8b:97:0e)
 ▶ Internet Protocol, Src: 192.168.122.40 (192.168.122.40), Dst: 10.10.15.40 (10.10.15.40)
 ▶ User Datagram Protocol, Src Port: ipsec-nat-t (4500), Dst Port: ipsec-nat-t (4500)
 ▶ UDP Encapsulation of IPsec Packets
 ▶ Encapsulating Security Payload

图16-77 NAT-T的IPSec数据包结构

从以上两个IPSec数据包的结构图，我们可以很清楚看到图16-77中在ESP及IP包头之间多了一个UDP包头，这是IPSec数据包可以穿越NAT的关键之处。当IPSec数据包有了这个UDP包头之后，IPSec数据包再穿越NAT主机时，NAT主机便可以修改IP包头及UDP包头的内容，但IPSec数据包的接收端并不会检查IP包头及UDP包头的内容，因而使得IPSec数据包可以正常穿越NAT主机。

那我们要如何在racoon下启用NAT-T的功能呢？以图16-78及配置文件为例，可以看到racoon的配置文件与以前的内容并无太大差异。主要差别在第5行到第16行及第20行，其中第9行及第10行的IP是L2TP服务器在因特网的公网IP，除此之外，其他设置步骤与之前的示例完全相同。

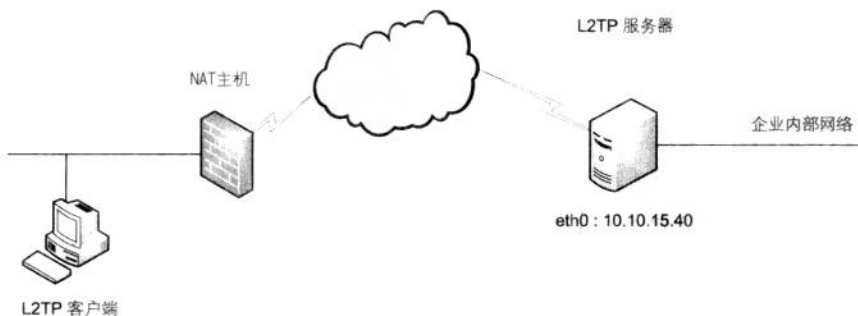


图16-78 NAT-T示例

```

1 path include "/etc/racoon";
2 path pre_shared_key "/etc/racoon/psk.txt";
3 path certificate "/etc/racoon/certs";

```

```

4 log debug;
5 timer {
6     natt_keepalive 10sec;
7 }
8 listen {
9     isakmp 10.10.15.40[500];
10    isakmp_natt 10.10.15.40[4500];
11 }
12 padding {
13     max_payload_length 20;
14     randomize off;
15     strict_check off;
16     exclusive_tail off;
17 }
18 remote_anonymous {
19     exchange_mode main,aggressive;
20     nat_traversal on;
21     generate_policy on;
22     certificate_type x509,"cert.pem","key.pem";
23     verify_cert on;
24     verify_identifier on;
25     my_identifier asnldn;
26     peers_identifier asnldn;
27     proposal {
28         encryption_algorithm 3des;
29         hash_algorithm sha1;
30         authentication_method rsaig;
31         dh_group modp1024;
32     }
33 }
34 sainfo_anonymous {
35     lifetime_time 1 hour;
36     encryption_algorithm 3des;
37     authentication_algorithm hmac_sha1;
38     compression_algorithm deflate;
39 }

```

16.6 小结

除了以IPSec为基础的VPN技术之外，L2TP也是业界常用来构建VPN的技术。本章完整介绍了L2TP技术，同时也说明了该如何通过IPSec来保护不安全的L2TP协议。本章旨在帮助你了解如何让Windows系统与Linux的VPN服务器建立起安全的通信环境，以防止非法的网络监听行为，保障企业的通信安全。



Linux 网络安全技术与实现 (第2版)

《Linux网络安全技术与实现(第2版)》首先讨论网络基础架构,然后循序渐进地讲解安全、基于策略的路由、流量控制和虚拟专用网络等知识,带您在网络安全世界中尽情畅游。如果您准备投身Linux网络安全领域,那么这本将理论与实践完美融为一体的书籍将是您的良师益友,将全面系统地指导您构建固若金汤的企业网络安全屏障。

- 深入浅出地介绍TCP/IP通信知识。
- 透彻讲解防火墙技术。
- 讨论如何灵活使用防火墙模块。
- 介绍常见的网络攻击手段以及相应的防御之道。
- 分析如何使用Squid来节省带宽并保护脆弱的Web服务器。
- 披露重要的内核补丁和编译技术。
- 介绍灵活多变的透明防火墙,以及如何将Linux防火墙升级为应用层防火墙。
- 讨论优于商用路由器的基于策略路由和多路带宽合并技术。
- 讲解Linux带宽管理技术。
- 讨论如何灵活使用MRTG来统计防火墙管理信息。
- 介绍弱点扫描、入侵检测及主动防御系统。
- 全面介绍数字证书及其应用。
- 讨论如何用Linux系统构建基于IPSec和L2TP的VPN系统。

清华大学出版社数字出版网站

WQBook 书文局泉

www.wqbook.com

悦知文化
Delight Press

www.delightpress.com.tw

上架建议: 操作系统/Linux网络安全

ISBN 978-7-302-27886-3



9 787302 278863 >

定价: 68.00元