**IPFW中文手册**

　ipfw是FreeBSD下的一个用户界面，它用来控制ipfw防火墙和流量整形。

　　ipfw的配置，也叫做规则集，它是由一系列的规则组成的，这些规则的编号是从1到65535。通过ipfw的数据包可以来自协议栈的许多各个的地方（这主要依赖于数据包的来源和目的，某些数据包很可多次经过防火墙）。数据包在通过防火墙的时候，要和防火墙规则集逐条对比。如果发现匹配的规则，则执行相应规则的动作。

　　在特定的规则动作和系统设置下，数据包在匹配完一个规则后，可能在某些规则之后重新进入防火墙。

　　一个ipfw规则集总是包含一个默认的规则（编号为65535），这个默认规则匹配所有的数据包，不能修改或删除。根据内核配置的不同，这个默认规则或者是拒绝所以数据包通过，或者是允许所有数据包通过。

　　如果规则集中存在一个或多个包含有keep-state（状态保持）或limit（限制）的规则，那么ipfw就表现为状态保持。例如：当这种规则匹配了一个数据包时，ipfw将创建一个动态规则，让后来的，同这个数据包的地址和端口参数严格一致的数据包，与这个动态规则相匹配。

　　这些动态规则，都有一个有限的生存期。生存期在第一次遇到check-state、keep-state或者是limit规则时被检查。这些动态规则的典型应用，是用来保证合法的数据包在通过时开启防火墙。查看下面的“状态防火墙”和“示例”章节，获取更多关于状态保持的信息。

　　包括动态规则在内的所有规则，都有几个相关的计数器：一个包计数器，一个字节计数器，一个记录计数器和一个用来提示最后匹配时间的时间戳。这些计数器都可以被ipfw命令显示和清零。

　　可以用add命令来添加规则；可以用delete命令来删除单个或一组规则；可以用flush命令来删除所有的规则（set 31除外）；可以用show和list命令来显示包括计数器在内的信息。最后，计数器可以被zero和resetlog命令清零。

　　并且，每个规则隶属于32个不同的规则集中的一个。有一些ipfw的命令可以独立地操作这些规则集，例如：启动、禁止、交换规则集；把一个规则集中的所有规则移动到另一个规则集；删除一个规则集中的所有规则。这对于安装临时配置，或者是测试配置是非常有用的。查看“规则的规则集”一节，获取更多关于set的信息。

　　以下是ipfw命令可以利用的选项：

-a　　使用 list 时，列出数据包的计数器，show命令包含这个选项。

-b　　只显示动作和注释，不显示规则体。包含-c选项。

-c　　进入或显示规则时，用紧凑格式输出。例如：在没有任何附加意义的情况下，不显示“ip from any to any”字符串。

-d　　在list时，在显示静态规则的同时，显示动态规则。

-e　　在list时，如果指定了-d选项，则显示过期的动态规则。

-f　　对于有些会导致问题的命令，不要提出确认的询问。例如：flush命令。哪果一个进程没有相关的tty，这个是默认被包含的选项。(译注：没看懂这句话的意思，好像是如果不是在tty中输入的命令，比如在scripts中，默认是有-f的。）

-n　　只检查命令的语法，不真正执行命令。

-N　　尝试在输出中，解析地址和服务名。

-q 　 当新增 (add)、归零(zero)、重置日志(resetlog)或清除(flush) 时，不要显示任何回应信息（就像使用-f选项一样）。这对于在脚本里利用ipfw命令来调整防火墙规则（例如执行“sh /etc/rc.firewall”脚本）、或者是处理与一个远程登录相关的有多条ipfw命令的，是非常有用的。同时，它防止无意间，在规则中加入或删除一个列表而导致的错误。（译注：没看懂，好像下面是一个例子。）在正常的模式，默认内核配置下，一个flush命令会输出一个信息。因为所有的规则都被清除掉，这个消息又不能传送到登录的进程中，脚本后面的规则将无法执行，所以将导致远程连接中断。这时，只好回到计算机前去工作了。

-S 当列出一个规则的时候，显示每条规则的规则集号。如果没有指定这个标识，则被禁止的规则不再显示。

-s [field]

当显示pipe时，按照四个计数器之一排序显示。（对于所有的数据包、当前数据包或者是字节数）

-t 当列出时，显示最后匹配的时间戳。（显示用ctime()转化后的时间）

-T 当列出时，显示最后匹配的时间戳。（从新纪元到现在的秒数）。这个格式更容易被脚本处理。

　　为了使配置更容易，规则可以写到一个文件里供ipfw处理，语法规则参照大纲栏最后一行（译注：指本文最前面的语法部分最后一行，即：“ipfw [-cfnNqS] [-p preproc [preproc-flags]] pathname”）。注意必须是绝对路径。这个文件会被逐行读取并且将作为ipfw工具的参数处理。

　　作为选择，你还可以使用-p preproc参数来使用预处理程序，其中路径是用管道来传入的。可以使用的预处理程序包括cpp和m4。如果preproc不是以"/"开头，那么将搜索PATH里的路径。在使用它的时候，要谨慎，因为在很多时候，ipfw启动时，系统文件可能没有挂接（比如：文件系统是通过NFS挂接的）。一旦使用了 -p参数，所有其它的参数，都被传送到预处理程序去等待处理。

　　ipfw的pipe和queue命令用来配置流量整形，你可以下面的“TRAFFIC SHAPER (DUMMYNET) CONFIGURATION”节看到看到相关内容。

　　如果系统的world和内核不能保持一致，ipfw的ABI将失效，以防你加入任何规则。这样能保证你的顺利地完成启动过程。你可以利用ipfw临时禁止的机会登录你的网络，来修复你的问题。

PACKET FLOW（数据包流程）

　　一个数据包在几个内核变量的控制下，在协议栈的多个地方被防火墙检查。这些地方和变量如下图所示，要在头脑中保持下图，它对于规划一下正确的规则集是非常重要的。

^ 到上层 V

| |

+----------->-----------+

^ V

[ip(6)\_input] [ip(6)\_output] net.inet(6).ip(6).fw.enable=1

| |

^ V

[ether\_demux] [ether\_output\_frame] net.link.ether.ipfw=1

| |

+-->--[bdg\_forward]-->--+ net.link.bridge.ipfw=1

^ V

| 到设备 |

　　如图所示，根据数据包的源和目的，以及系统配置的不同，同一个数据包穿过防火墙的次数在0和4之间变化。

　　需要注意的是，在包通过不同栈的时候，头会被加上或剥离，可能会导致这些包被检查不到。例如：当ipfw被ether\_demux()调用的时候，进入的包将包括MAC头；而当ipfw被ip\_input()或者ip6\_input()调用的时候，同一个数据包的MAC头将被剥离。

　　还需要注意的是：不论检查发生在什么地方，也不论数据包源自哪里，每个数据包也整个规则进行对照。（译注：是整个规则集吗？有时候只对照一部分的时候就中止检查）。如果一个规则包含的一些匹配模型或动作不符合某处的要求（比如：在ip\_input或ip6\_input处匹配MAC头），那么匹配模型将不会被匹配。然而，带有not前缀操作符的匹配模型，总是可以和这些数据包相匹配。所以，如果有必要，在有很多地方可以选择的时候，有能力的程序员可以为写出的规则集选择一个合适的位置。skipto规则在这儿是非常有用的。例如：

# packets from ether\_demux or bdg\_forward

ipfw add 10 skipto 1000 all from any to any layer2 in

# packets from ip\_input

ipfw add 10 skipto 2000 all from any to any not layer2 in

# packets from ip\_output

ipfw add 10 skipto 3000 all from any to any not layer2 out

# packets from ether\_output\_frame

ipfw add 10 skipto 4000 all from any to any layer2 out

（对，在这儿，没有办法区别ether\_demux和bdg\_forward）

SYNTAX（语法规则）

　　通常，每个关键字或者参数都是作为独立的命令行参数来提供的，前后都不能有空格。关键字是区分大小写的，而参数按照它们本身的含义，可以是，也可以不是区分大小写的。（例如，udi要区分大小写，而hostname没有必要区分大小写）

　　在ipfw2中，为了提高可读性，可以在逗号后面加上一个空格。当然，你也可以把整个命令（包括参数）当作一个参数来处理（译注：就是加上双引号，作为一个字符串放到某个命令中）。例如，以下几种格式效果相同：

ipfw -q add deny src-ip 10.0.0.0/24,127.0.0.1/8

ipfw -q add deny src-ip 10.0.0.0/24, 127.0.0.1/8 （译注：127前面有个空格）

RULE FORMAT（规则格式）

　　ipfw的规则格式如下：

[rule\_number] [set set\_number] [prob match\_probability] action

[log [logamount number]] [altq queue] [{tag | untag} number] body

　　规则的 body 部分用来设定包过滤的信息，具体细节如下：

Layer-2 header fields 第二层头段，什么时候可用

IPv4 and IPv6 Protocol TCP, UDP, ICMP, 协议等

Source and dest. addresses and ports　 源和目的。地址和端口

Direction 数据包的流向，见“流量整形”一节

Transmit and receive interface 传输和接收界面，用名字或地址表示

Misc. IP header fields 组合IP头字段，包括版本、服务类型、报文长度、唯一标识、分段标志、生存期

IP options 　　　　　　　 IP选项字段

IPv6 Extension headers IPv6扩展头，包括分片报头、中继选项报头、路由报头、路路由选择报头、IP层安全服务选项、Source routing rthdr0, Mobile IPv6 rthdr2（译注：最后两项不知道怎么翻译）

IPv6 Flow-ID　　　　　　　　　　　　　IPv6的流量标识

Misc. TCP header fields 组合tcp头字段，有TCP标志（syn,fin,ack,rst等）、序列号、回复号，窗口等。

TCP options　　　　　　　　　　　　　　TCP的选项

ICMP types ICMP包

ICMP6 types ICMP6包

User/group ID 本地栈所归属的用户和组

Divert status 是不是源自一个divert接口（例如：natd(8))

　　需要注意，上面的很多信息中，比如源MAC和IP地址、TCP/UDP端口等，是比较容易伪造的，所以只对这些信息进行过滤不能保证达到预期的效果。

rule\_number

　　每个规则都有规则号，规则号从1－65535。65535是保留的默认规则。ipfw将按规则号的顺序依次检查这些规则。可以有多个规则用同一个规则号，在这种情况下，将按照它们的添加顺序依次检查或列出。如果一个规则没有指定规则号，内核将自动赋于一个规则号，使它成为默认规则前的最后一个规则（即小于65535）。自动添加的规则号有个步进值，在不大于65535的情况下，每次添加规则时，都自动加上这一步进值。它默认是100，对应于net.inet.ip.fw.autoinc\_step这个内核变量。

set set\_number

　　每个规则都对应于一个规则集编号，规则集编号取值为：0－31。规则集可以单独地被禁止或启用，所以，它对于规则集的原子操作是

至关重要的。它还可以轻松地删除掉一组规则。如果一个规则没有指定规则集编号，它默认的规则集编号就是0。规则集编号31是个特殊的规则集，它不能被禁止，也不能用ipfw的flush命令删除，它是默认规则的规则集。规则集编号31可以用ipfw delete set 31命令删除。（译注：这在远程调试排斥式防火墙时特别有用，防止把自己锁在外面。）

　prob match\_probability

　　这个语句只用来指定可能性（从0到1之间浮点值）。这在很多随机丢包的应用中，或者（在同dummynet联用的情况下）模拟多路径条件下无序包传递的效果中，是非常有用的。

　　注意：在检查时，这个条件比其它所有条件都先被检查，包括会有副作用的keep-state或者check-state。

　log [logamount number]

　　当一个包匹配到一个有log关键字的规则时，一条消息会被syslogd记录到一个LOG\_SECURITY设备中。记录动作只会在同时满足下面两个条件时才发生：1、内核选项net.inet.ip.fw.verbose等于1（当编译内核时带着IPFIREWALL\_VERBOSE参数时，这是默认值）；2、所记录的包数量还达不到logamount number所指定的数量。如果没有指定logamuont，则记录的包的上限受内核变量net.inet.ip.fw.verbose\_limit的限制。在这两种情况中，logamount取值为零时，则取消记录数量的限制。

　　当记录数量达到上限时，可以通过重置记录计数器或包计数器的方式再重新开始记录。详细情况请查看resetlog命令。

注意：记录的时间是在所有的包检测条件都完成之后，并且在采取相应的动作（接受、拒绝等）之前。

tag number

　　当一个包和一个带有tag关键字的规则相匹配时，一个指定的数字标签就被附加到这个包上，数字标签的范围是1-65534。这个标签作为一个内部的记号（它不会从线路上发送出去），以便后来可以区分这些数据包。这有时候非常有用，比如：提供接口之间相互信任和开启一个基于安全的过滤。一个包可以同时有多个标签。标签是“顽固的”，意思就是只要标签被应用到某个数据包上，那么它就一直存在，直到被明确地解除为止。这个标签在内核中一直存在，但是一旦数据包离开内核，标签就会消失。例如：在当把数据包发送到网络或者发送到divert栈时，标签就会消失。

　　要检查已经存在的标签，可以用tagged规则选项，要删除存在的标签，用untag关键字。

　　注意：既然标签是在内核空间内一直存在的，所以它们就可以被内核的网络子系统（使用mbuf\_tags设备）设置或者是取消，而不仅仅依靠ipfw的tag和untag关键字。例如：可以存在一个netgraph(4)节点做流量分析，并且给数据包打上标签，以供后面的防火墙进行检查。

untag number

　　当一个数据包匹配一个带有untag关键字的规则时，将在这个数据包所附加的标签中查找untag number所指定的标签号，如果找到，则将其删除。对于数据包上的其它标签号，则原封不动。

altq queue（译注：交错队列， Alternate Queuing，参见PF文档）

　　当一个数据包匹配了一个带有altq关键字的规则时，将被附加上一个指定队列（见altq(4)）的ALTQ标识符。注意，ALTQ标签只适用于从ipfw“向外出”的数据包，而不是被防火墙拒绝或者准备到divert界面的数据包。还要注意的是，当处理数据包的时候，如果没有足够内存，那么数据包将不会被打上标签，所以，一个明智的做法，是使你的“默认”ALTQ队列规则记录下这种情况。如果有多个altq规则匹配于同一个数据包，那么只加上第一条规则的ALTQ类别标签。那样做的效果是：在规则集中靠前的altq队列规则能起到流量整形的作用，而靠后的规则只达到过滤的效果。例如：把还有ALTQ标签的、check-state和keep-state规则放到规则集的后部，有提供真正的包过滤的作用。

　　在IPFW使用altq的名字来检查队列之前，你必须用pfctl(8)工具来设置队列。一旦重置ALTQ命令，那么包含队列标识符的规则将过期，需要被重新读取才能生效。过期的队列标识符将会产生错误的分类。

　　所有的ALTQ系统进程都可以被ipfw的enable altq和disable altq命令启用或关闭。ALTQ流量整形有真实规则动作，永远附加在ALTQ标签之后，和net.inet.ip.fw.one\_pass毫无关系。

RULE ACTIONS（规则动作）

　　一个规则可以和下面动作中的一个或多个相关联，如果数据包匹配了规则体，那么这些动作将被执行。

allow | accept | pass | permit 允许匹配规则的数据包通过，规则搜索中止。

check-state

　　将数据包和动态规则集里的规则进行对照。如果找到，则执行产生这条动态规则的那个规则相关联的动作，如果没找到，则进行下一条规则。check- state没有规则体。如果在规则集里没有找到check-state，则在第一次出现keep-state或者limit规则时，检查动态规则集。

count 　　　 更新和本条规则相匹配的所有数据包的计数器。继续进行下一条搜索。

deny | drop 抛弃和本条规则相匹配的数据包。搜索中止。

divert port 把和本条规则相匹配的数据包转向到divert(4)栈所绑定的端口。搜索中止。

　　fwd | forward ipaddr | tablearg[,port]

　　修改匹配数据包的下一跳地址，有可能是一个IP地址，也有可能是一个主机名。当使用tablearg关键字时，下一跳地址可以不是一个明确的地址，它可以从最后的数据包检查表中获取。如果匹配本规则，搜索中止。

（译注：tablearg是FreeBSD7.0的ipfw新增功能，它可以使查表的结果作为规则的一部分来使用，可以用来优化一些规则集，或实现基于策略的路由，在使用时，要配合table命令使用。）

　　如果指定的ipaddr是一个本地地址，所匹配数据包将被传递到本地机器的相应端口（如果在规则中没有指定端口，则是数据包中的端口号）。如果ipaddr不是一个本地地址，那么端口号（如果指定的话）将被忽略，数据包将会根据本机的路由表被转发到远程地址。

　　一个fwd规则不会和一个二层的数据包（用ether\_input,ether\_output或者bridged接受）相匹配。fwd不会修改数据包中的任何数据，特别是目的地址。所以，除非在目的系统中有相应的规则接受这些数据包，否则，被转发到目的系统的数据包将被拒绝。对于被转发到本地的数据包，将把本地地址作为它的原始目的地址。这将使netstat(1)看起来有些古怪，但是，对于透明代理服务器来说，却是有用的。

　　要使用fwd，必须在内核中使用IPFIREWALL\_FORWARD选项，并重新编译内核。

nat nat\_nr

Pass packet to a nat instance (for network address translation,

address redirect, etc.): see the NETWORK ADDRESS TRANSLATION

(NAT) Section for further information.

　　将数据包传送到一个nat界面（用来做地址转换或地址重定向等）：详细信息，请查看NETWORK ADDRESS TRANSLATION (NAT)一节。

pipe pipe\_nr

　　将数据包传送到一个dummynet(4)的“管道”（用来实现流量整形，延迟等）。详细信息请查看TRAFFIC SHAPER (DUMMYNET) CONFIGURATION一节。如果匹配，则搜索中止。但是，在离开管道的时候，并且内核变量net.inet.ip.fw.one\_pass没有设置，数据包将被送回到防火墙的下一条规则。

queue queue\_nr

　　将数据包传送到一个dummynet(4)的“队列”（用WF2Q+实现流量整形）

reject (Deprecated).

　　不可到达的同义词。

reset

　　丢掉和本规则相匹配的数据包，如果是一个TCP数据包，则发送一个重置(RST)消息。搜索中止。

reset6

　　丢掉和本规则相匹配的数据包，如果是一个TCP数据包，则发送一个重置(RST)消息。搜索中止。

skipto number

　　跳到指定的规则，忽略掉比指定规则号小的规则。搜索从指定的规则号继续。

tee port

　　将相匹配的数据包复制一份到divert(4)所绑定的端口。搜索继续。

unreach code

　　丢弃和本规则相匹配的包，并且尝试发送一条编号为指定编号的不可到达的ICMP消息。编号范围是0－255，或者使用下列别名之一：net, host, protocol, port, needfrag, srcfail, net-unknown,host-unknown, isolated,net-prohib, host-prohib, tosnet, toshost, filter-prohib,host-precedence or　precedence-cutoff。搜索中止。

unreach6 code

　　丢弃和本规则相匹配的包，并且尝试发送一条编号为指定编号的不可到达的ICMPv6消息。编号为0,1,3,4或者是下列别名之一：no-route, admin-prohib, address or port。搜索中止。

netgraph cookie

　　使用指定的cookie把数据包转向到netgraph。搜索中止。如果稍后数据包众netgraph返回，则受内核变量net.inet.ip.fw.one\_pass的影响，或者被接受，或者从下一条规则继续。

ngtee cookie

　　将数据包复制一份，转发到netgraph，原始的数据包则受net.inet.ip.fw.one\_pass的影响，或者被接受，或者从下一条规则继续。查看ng\_ipfw(4)可以获取更多关于netgraph和ngtee动作的信息。

RULE BODY（规则体）

　　规则体包括零个或多个匹配模式（比如指定源和目的的地址或端口，协议选项，进入和流出的界面等。），这些匹配模式必须按顺序对数据包进行识别。通常，匹配模式之间用一个隐含的“和”操作连接，比如：所有规则中的模型都必须去和数据包进行匹配。单个的模型可以使用“not”操作符作为前缀，用来表示相反的匹配结果，就象在下例中：

　　ipfw add 100 allow ip from not 1.2.3.4 to any

　　特别的，当用来表示二选一的“或”模型时，可以使用一对小括号“（）”或一对大括号“｛｝”括起来的“or”操作符，比如：

　　ipfw add 100 allow ip from { x or not y or z } to any

　　只允许有一级小括号。特别提示：大多数shell中，小括号和大括号都有特殊的意义，所以，为了防止被shell误解，强烈建议在小括号和大括号前加上反斜杠“\”。

　　规则体中，至少要包含一个源地址和目的地址的语句。"any"关键字可以用在多种地方，用来表明源或目标可以是任意的。

　　规则体的格式如下：

　　[proto from src to dst] [options]

　　第一个部分（proto from src to dst）用来和早期版本的FreeBSD相兼容。在新的FreeBSD版本中，所有的匹配模型（包括MAC头，IP协议，地址和端口）都可以在options里指定。

　　各规则字段的含义如下：

　　proto: protocol | { protocol or ... }

　　protocol：[not] protocol-name | protocol-number

　　　　一个IP协议可以用序号或名称（在/etc/protocols里可以查看完整的名称列表），或者是下面的关键字之一：

　　　　ip4 | ipv4　　　　和IPv4数据包相匹配

　　　　ip6 | ipv6 和IPv6数据包相匹配

　　　　ip | all 和所有数据包相匹配

　　　　ipv6在proto选项中，将被作为内部协议。并且，ipv4在prto选项中是不适用的。

　　　　{ protocol or ... } 格式的“或”的功能只是方便使用，不提倡这种做法。

　　src and dst: {addr | {addr or ... }} [[not] ports]

　　　　一个地址（或者一个列表，见后），可以使用ports修饰符。

　　　　第二种格式（用or连接起来的多个地址）只是方便使用，却不提倡这种做法。

　　addr: [not] {any | me | me6 | table(number[,value]) | addr-list | addr-set}

any 匹配任意地址

me 匹配系统中所有设置的IP地址。

me6 匹配系统中所有设置的IPv6地址。地址列表是在分析数据包的时候才被列出的。

table(number[,value])

如果一个数据包在表序号为number的表中存在相应的条目，则匹配。如果指定了一个可选的32位无符号数value，则条目序号为value的数据包才被匹配。查看LOOKUP TABLES一节，可以获取更多信息。

addr-list: ip-addr[,addr-list]

ip-addr:　一个主机或者一段地址，可以采用下列格式：

numeric-ip | hostname

　匹配单个IPv4地址，可以采用点分四组格式或主机名。当采用主机的时候，在添加规则的时候将被解析为相应的IP。

addr/masklen

　匹配于指定的基地址addr（可以是单个IP地址、网络地址或者是主机名）和掩码位masklen。例如：1.2.3.4/25或1.2.3.0/25 匹配于从1.2.3.0到1.2.3.127的IP地址。

addr:mask

　匹配于指定的基地址addr（可以是单个地址、网络地址或主机名）和指定的子网掩码mask，用点分四组表示法。例如：1.2.3.4: 255.0.255.0或者1.0.3.0:255.0.255.0匹配于1.\*.3.\*。只建议在非连续的子网掩码中使用这种格式，对于连续的子网掩码，请使用addr/masklen格式，它的格式更紧凑，也不容易出错。

　（译注：这个和反掩码有此类似）

addr-set: addr[/masklen]{list}

　list:{num | num-num}[,list]

　　匹配于基地址是addr（可以是IP地址、网络地址或主机名），并且最后一个数字在｛　｝列表中的地址。请注意在大括号和数字之间不能有空格（在逗号之后可以有空格）。大括号中的每个元素可以是单个的条目，也可以是一个范围。masklen字段通常用来限制某一段地址的长度，取值范围为24到32，默认取24。

　　这个对处理在某一个地址段内多个分散的IP地址最有效，因为是用掩码位来匹配，所以节省了时间，也显著地降低了规则的复杂程序。

　　一个例子：1.2.3.4/24{128,35-55,89}或者1.2.3.0/24{128,35-55,89}表示下列的IP地址：

　　　　1.2.3.128, 1.2.3.35到1.2.3.55，1.2.3.89

　　　（译注：由于很多shell中大括号有特殊意义，所以通常情况下，要在大括号前加上反斜杠“\”，表示转义）

addr6-list: ip6-addr[,addr6-list]

ip6-addr: 一个主机或者一段地址，可以采用下列格式：

　　numeric-ip | hostname

　　同单一的inet\_pton(3)所允许的IP地址或主机名相匹配。主机名在添加规则时被解析为相应的IP。

　　addr/masklen

　　匹配所有基地址是addr（可以是inet\_pton允许的地址或主机名），并且掩码位数是masklen的地址。

　　没有提供批量的IPv6地址，因为IPv6地址在前缀之后，是随机的。（译注：不是很懂）

ports: {port | port-port}[,ports]

　　对于支持端口的协议，可以指定一个可选的单个端口、多个端口、或者是一段端口，这些端口用逗号分开，并且不能有空格。还可以使用可选的not操作符。“-”用来表示包括边界在内的一段范围。

　　可以使用服务名（见/etc/services）来代替端口号。端口号或端口范围列表最大数目是30，如果想指定更多的端口，可以在options中使用or符号。

　　如果服务名中存在减号“-”，可以在前面加上反斜杠来转义。（在shell中，如果要使用反斜杠，则要用两次，否则，shell将会把它作为转义字符处理。），例如：

　　　　　ipfw add count tcp from any ftp\\-date-ftp to any

　　非零偏移量的分段的数据包（比如不是第一个段），不会匹配于任何带端口号的规则。参见frag一节。

RULE OPTIONS (MATCH PATTERNS)：（规则选项（匹配模式））

　　可以在规则中使用附加的可选匹配模式。在规则中，可以没有，也可以有多个选项，它们可以在前面加上not运算符，也可以用or来连接。

　　可以使用的选项如下所示（按照字母顺序）：

　　//这是一个注释

　　在一个规则中插入一段文字作为注释。在//后面所有的东西都被作为注释来处理。你可以使用只有注释的规则，它将只有一个count动作。

bridged layer2的别名

diverted 　　　　　　只和一个转发界面产生的数据包相匹配。

diverted-loopback 　只匹配于从divert堆栈重新回到IP堆栈的进入的数据包。

diverted-output 　只匹配于从divert堆栈重新回到IP堆栈的向外发送的数据包。

dst-ip ip-address 和目的地址是参数中指定的IPv4地址之一的数据包相匹配。

{dst-ip6 | dst-ipv6} ip6-address

和目的地址是参数中指定的IPv6地址之一的数据包相匹配。

dst-port ports 和目的端口是参数中指定端口之一的数据包相匹配。

established 　和设置了RST和ACK位的TCP数据包相匹配。

ext6hdr header

　匹配于IPv6包的扩展头中包括参数指定的报头的数据包。可用的报头包括：

Fragment, (frag), Hop-to-hop options (hopopt), any type of Rout-

ing Header (route), Source routing Routing Header Type 0

(rthdr0), Mobile IPv6 Routing Header Type 2 (rthdr2), Destination

options (dstopt), IPSec authentication headers (ah), and IPSec

encapsulated security payload headers (esp).

flow-id labels

匹配于包含指定流量标签的IPv6数据包。labels是一个用逗号隔开的流量标签号列表。

frag

　　　　　　匹配于IP报文中，不是第一个段的分段。需要注意的是这些数据包不存在下一个协议头（例如：TCP，UDP），所以任何于协议头相关的选项，都不会和它们相匹配。

gid group

如果发接收或传输的TCP或UDP数据包隶属于该指定组group，则匹配。一个group可以用名称，也可以用代号。（译注：group为系统中的组，下面还有uid，指的是user id。）

jail prisonID

匹配于所有发送自或发往被囚禁的TCP或UDP包，囚禁标识用prisonID给出。

icmptypes types

匹配于指定列表中的icmp类型的包。类型列表可以是单种类型的任意组合，用逗号隔开。不支持范围。支持的单个类型如下：

echo reply (0), destination unreachable (3), source quench (4),

redirect (5), echo request (8), router advertisement (9), router

solicitation (10), time-to-live exceeded (11), IP header bad

(12), timestamp request (13), timestamp reply (14), information

request (15), information reply (16), address mask request (17)

and address mask reply (18).

icmp6types types

匹配于指定列表中的icmp6类型的包。类型列表可以是单种类型的任意组合，用逗号隔开。不支持范围。支持的单个类型如下：

in | out

　分别匹配于进入或流出的数据包。in和out是互斥的（实际上，out就是指不是in）。

ipid id-list

仅对IPv4数据包有效。匹配于ip\_id字段为指定id-list列表中数值的包。id-list的规则和port一样，可以是单个的数字，也可以是一个列表，还可以是一个范围。

iplen len-list

　　　　仅对IPv4数据包有效。匹配于所有长度为指定len-list列表中数值的包，长度值括头和数据。len-list的规则和port一样，可以是单个的数字，也可以是一个列表，还可以是一个范围。

ipoptions spec

　　　仅对IPv4数据包有效，如果在头中的IP选项中，包含spec参数列表中指定的选项，则匹配。spec参数列表可以用逗号隔开。可以使用的IP选项如下：

　　　ssrr（严格源路由）、lsrr（宽松源路由）、rr（记录包路由）、和ts（时间戳）。可以用叹号'!'表示没有这个选项。

ipprecedence precedence

仅对IPv4数据包有效，如果优先级字段的值和指定的precedence相同，则匹配。

ipsec

匹配于曾经和IPSEC相关的包。（例如：内核存在选项IPSEC和IPSEC\_FILTERTUNNEL选项时，在IPSEC中，被加密的包和正确解密的包。）

提示：这地方的ipsec和后面将提到的ipsec不同，ipsec协议将只检查IP的协议字段，而不管内核是不是支持，也不管IPSEC数据是否有效。

　　　　　　还需要注意：如果内核没有IPSEC支持，这个标识将被忽略而不给出任何消息。在处理规则的时候，就像ipsec不存在一样，不会影响对其它关键字的处理。

iptos spec

对于IPv4包，如果tos字段包含spec列表中指定的值，则匹配。支持的IP服务格式为：

lowdelay (IPTOS\_LOWDELAY), throughput (IPTOS\_THROUGHPUT),

reliability (IPTOS\_RELIABILITY), mincost (IPTOS\_MINCOST),

congestion (IPTOS\_CE). The absence of a particular type may be

denoted with a `!'.

最小延迟（IPTOS\_LOWDELAY)，最大呑吐量 (IPTOS\_THROUGHPUT),

最高可靠性 (IPTOS\_RELIABILITY), 最低费用 (IPTOS\_MINCOST),

拥挤？？ (IPTOS\_CE)。可以用叹号“!”来表示没有这个选项。（译注：congestion(IPTOS\_CE)：没找到资料）

ipttl ttl-list

仅对IPv4数据包有效，如果它的生存时间包含在ttl-list中，则匹配。ttl-list同ports列表一样，既可以是一个单独的值，也可以是一个列表或一个范围，还可以是服务名称。

ipversion ver

　　　匹配于IP版本字段等于指定的ver的数据包。

keep-state

　　　如果匹配，防火墙将创建一个动态规则，它的默认动作是匹配在源和目的的IP/port之间，使用同一协议的双向流量。动态规则有一定的生存期（受一系列内核变量的控制），生存时间在每遇到匹配的包时，都进行刷新。

layer2

　　　仅匹配第二层的包。例如：从ether\_demux()和ether\_output\_frame()来，通过ipfw的包。

limit {src-addr | src-port | dst-addr | dst-port} N

　　 在规则指定的相同参数下，防火墙只允许N个连接。可以指定至少一个源地址和目的地址。目前，仅支持IPv4。

{ MAC | mac } dst-mac src-mac

　　　匹配于指定的dst-mac和src-mac地址。可以用any来表示所有的MAC地址，或者用冒号分开的6组16进制的数字，还可以在后面指定有效位数。掩码位数的指定方法有如下两种：

　　　　1、用斜杠“/”后跟随着有效位数的方法。例如，要指定33位的有效地址，可以用：

MAC 10:20:30:40:50:60/33 any

　　　　2、 用逻辑和符号“&”跟随一个6组用冒号隔开的位掩码，例如：一个最后16位为有效位的地址：

MAC 10:20:30:40:50:60&00:00:00:00:ff:ff any

需要注意“&”符号在许多shell中有特殊意义，所以在用的时候，通常需要进行转义处理。

注：源MAC地址和目的MAC地址可以互换，但是IP地址是不可以的。（译注：没读懂）

mac-type

　　　如果数据包中，以太网类型字段中的值等于参数mac-type指定值中的一个，则匹配。mac-type的指定方式同ports（例如：单一的值，范围等）。对于vlan，ipv4，ipv6之类的已知类型，可以用符号名称来表示。

　　　mac-type的值可以采用十进制或十六进制（以0x开头），但是在打印的时候，只输出16进制（除非使用-N选项来尝试解析符号名称）。

proto protocol

　　　匹配于指定protocol协议的包。

recv | xmit | via {ifX | if\* | ipno | any}

　　　匹配于接收、传输或者通过某个界面的数据包。界面可以是一个网卡名称(ifx)，或者设备名(if\*），或者是IP地址，或者是其它界面。

　　　via关键字使防火墙检查通过某个界面的所有数据包。如果是recv或xmit代替了via，则只在界面上检查相应的接收或传输方向的数据包。如果同时指定了两个，则也同时检查接收至界面和传输出界面的数据包。例如：

ipfw add deny ip from any to any out recv ed0 xmit ed1

　　　　recv也可检查某界面上流出或注入的数据包，但是xmit只能检查某界面上流出的数据包。所以，在使用xmit的时候，必须同时指定out。

　　　一个数据包可以没有接收或传输界面：源自本地主机的数据包没有接收界面，目的为本地主机的数据包没有传输界面。（译注：为了好理解，recieve意思为接收自，xmit意思为传输至）

setup

　　　匹配于设置了SYN位，而没有设置ACK位的TCP包。这就是“'tcpflags syn,!ack'”的缩写。（译注：注意引号当中有个单引号）

src-ip ip-address

　　　仅对IPv4数据包有效，匹配于源IP是参数ip-address中指定地址之一的包。

src-ip6 ip6-address

　　　仅对IPv6数据包有效，匹配于源IP是参数ip-address中指定地址之一的包。

src-port ports

　　　匹配于源端口是参数ports中指定端口之一的包。

tagged tag-list

　　　如果数据包的tags包含在参数tag-list中，则匹配。tag-list格式和ports一样，可以是一个单一的值，也可以是一个列表或范围。tags是用来给规则打标签的一个参数。（具体情况可以参见tags参数）

tcpack ack

仅对TCP包有效。如果TCP包的头部设置了ACK字段，则匹配。

tcpdatalen tcpdatalen-list

　　　对于TCP数据包，如果TCP的长度是参数tcpdatalen-list中的一个，则匹配。tcpdatalen-list的规则同ports一样，可以是一个单一的值，也可以是一个列表或范围。

tcpflags spec

仅对TCP包有效。如果TCP的头部包含参数spec中指定的值，则匹配。spec是一个用逗号隔开的一个标志列表。支持的TCP标志有：

　　　fin，syn，rst，psh，ack和urg。可以在前面加上叹号：“!”取反。包含tcpflags的规则可能永远都不会匹配于一个非零偏移的分段包。详情可以参看frag选项。

tcpseq seq

　　　仅对TCP包有效。如果TCP头部的序列号（sequece）字段为参数seq指定的值，则匹配。

tcpwin win

　　　仅对TCP包有效。如果TCP头部的窗口（window）字段为参数win指定的值，则匹配。

tcpoptions spec

　　　仅对TCP包有效。如果TCP头部的选项字段（option）为参数指定的spec，则匹配。支持的TCP选项（option）有：

　mss (最大分段长度）, window (tcp窗口大小建议), sack (选择性的ack), ts (rfc1323 时间戳) and cc (rfc1644 t/tcp 连接计数)。可以用前置的叹号“!”来表示取反。（译注：sack表示选择性的ack，常用来乱序传输TCP数据，它可以用来查找特定的遗失的数据报，所以可以快速恢复状态）

uid user

　　　如果发出或接收的tcp或UDP数据包隶属于参数user指定用户，则匹配。user可以是用户名，也可以是用户的序号。

verrevpath

　　　对于进入的数据包，将对包的源地址路由表进行检查。如果包进入系统的界面和包路由流出的界面相同（译注：也就是数据包的路由界面正确），则匹配。否则是不匹配。所有流出的包和没有流入界面的包，都匹配。

　　　　这个选项的名称和功能都和思科的IOS命令类似：

　　　　ip verify unicast reverse-path

　　　　这个选项可以用制作anti-spoofing（反欺骗）规则，来防止源地址来自不正确的界面。还可以阅读antispoof选项。

versrcreach

　　　　对于进入的数据包，将对包的源地址路由表进行检查。如果对于源地址的路由存在，但是不是默认的路由或黑洞/拒绝路由，则匹配。否则不匹配。所有流出的包都匹配。

　　　　这个选项的名称和功能都和思科的IOS命令类似：

ip verify unicast source reachable-via any

这个选项可以用来制作anti-spoofing（反欺骗）规则，来阻止所有源地址不可到达的包。（注：这段不是很懂）

　　　　对于进入的数据包，则检查源地址是不是属于一个直接连接的网络。如果来自直接连接的网络，则数据包的流入界面和连接的网络进行对比。如果源地址的界面和直接连接的网络界面不一致，则不匹配。否则匹配。所有流出的包都匹配。

　　　这个选项常用来制作anti-spoofing（反欺骗）规则，用来阻止试图进入直接连接的网络，但是不是通过那个界面。这个选项和verrevpath相似，但是比它范围窄一些，因为它只检查来自直接连接网络的数据包，而verrevpath检查所有流入的包。

LOOKUP TABLES（查询表）

　　Lookup 表常用来处理大量的分散的地址集，特别是成千上万个地址。可以使用高达128个不同的lookup项目，编号从0－127。

　　每个项目格式为：addr[/masklen]，将匹配于所有基地址相同（表示为一个IP地址或一个主机名）和掩码位数相同的地址（表示为一个IP地址或一个主机名）。如果掩码位数没指定，则默认为32。

当在一个表中查寻一个IP地址的时候，将匹配一个最精确的项目。每个项目都有一个与之相关的32位无符号值，规则匹配代码可以使用它地进行检查。如果在添加规则时没有指定这个值，则默认为0。

　　一个项目可以被添加进一个表(add），或者从一个表中删除（delete），或者列出一个表的项目（list），还可以被清除（flush）。

每个表在内部存储时，使用和路由表相同的基数树方式。可以参考route(4)

Lookup表现在只支持IPv4。

tablerg提供了一个功能，它可以用在规则规则的动作、规则的参数或规则的选项中，使防火墙在表中寻找一个指定的值。

当然后fwd的时候，可以使用IP地址或主机名的格式来指定一个表的项目。可以参考示例章节的“使用表和tablearg关键字”。

SETS OF RULES（规则集合）

　　每个规则都属于32个不同规则集中的一个，规则集编号从0到31。规则集编号31是为默认规则保留的。

　　默认情况下，在添加一个新的规则时，除非指定set N属性，否则规则将被放入规则集0中。规则集可以被独立地和原子性地被启用或禁止，所以，这个方法提供了一个配置防火墙和切换规则的快速的简易的方法。启用或禁止规则集的方法是：

ipfw set [disable number ...] [enable number ...]

　　在上面的书写格式中，可以指定多个enable或disable。在命令中指定的每一个规则集，都会被单独处理。默认所有的规则集都是启用的。

　　　当你禁止一个规则集时，除了一个特殊的情况之外，它的所有规则就像是在防火墙配置里不存在一样：

　　在规则被禁止之前创建的动态规则，会一直保持住，直到过期为止。如果你想删除掉一个动态规则，你必须明确地删除创建它的父规则。

　　规则集号可以用move来改变。

ipfw set move {rule rule-number | old-set} to new-set

　　并且，还可以交换两个规则集：

ipfw set swap first-set second-set

　　查看“示例”章节可以了解更多关于规则集的规则。

STATEFUL FIREWALL（状态保持）

状态保持使防火墙可以在遇到匹配的指定数据包时，动态地创建一条相关的规则。支持动态规则的操作符有：check-state，keep-state和limit。

当一个数据包匹配于keep-state或limit规则时，将自动创建一个动态规则。可以匹配于所有的指定协议下的，基于同一个源IP/源端口、目的IP/目的端口的地址对的数据包。（源和目的在这儿仅表示第一次匹配的地址，但是他们对于后面的数据包来说，是完全等价的）。动态规则在第一次遇到check-state、keep-state或limit时被检查，动作也它父规则中的动作相同。

　　提示：在动态规则中，只检查协议、IP地址、端口，不再检查其它的附加属性。（译注：比如TOS等）

　　动态规则的一个典型应用，是在一个关闭的防火墙中，内部网络的第一个包TCP SYN通过时动态创建一个规则，就可以使以后的双向流量中，该次会话所有的数据包都通过，而不会被防火墙拒绝。

ipfw add check-state

ipfw add allow tcp from my-subnet to any setup keep-state

ipfw add deny tcp from any to any

　　一个近似的目的是有知UDP上，从内部网络的第一个UDP包可以创建一个动态规则，以便让后面的数据包通过防火墙。

ipfw add check-state

ipfw add allow udp from my-subnet to any keep-state

ipfw add deny udp from any to any

　　动态规则在一定的时间内将过期，这依赖于流量的状态和某些内核变量的控制。查看SYSCTL VARIABLES，以获取更多的细节。对于TCP会话来说，动态规则在临近失效时，可以发送一个keeplive包，来刷新这个状态。

在“示例”章节，有更多的如何使用动态规则的示例。

TRAFFIC SHAPER (DUMMYNET) CONFIGURATION（流量整形配置（dummynet））

　　ipfw同时也是流量整形工具（dummynet）的用户界面。

　　dummynet先用ipfw防火墙选出所需的数据包，然后再将这些包分成不同的流量。在数据包选择的过程中，可以利用所有的ipfw参数。根据本地规则的不同，一个流量含有的数据包可以属于一个TCP连接，也可以属于指定的源/目的主机、也可以属于整个子网络、还可以属于某一种协议等。

　　同一个流量下的数据包将通过下面两种对象之一，这两种对象是实现流量管理的基本要素。

　　pipe　　一个管道用来模拟一个给定带宽、延迟、队列大小和丢包率的连接。当数据包从防火墙中分类以后，就进入管道的头部，然后根据管道的配置再通过管道。

　　queue　　一个队列就是一个WF2Q+（Worst-case Fair Weighted Fair Queueing，译注：没找到中文翻译)策略的实现，WF2Q+是一个对WFQ的改进策略。

　　　　　　每个队列都对应于一个weight和一个相关的pipe流量。所有和一个pipe相关的存积的流量（例如，在队列中的数据包），将根据每个队列weight的比例，来共同分享pipe所指定的带宽。需要注意：weight并不是优先级。虽然在有较高weight流量的数据包积压的时候，一个有较低weight的流量依然能保证分享到一定的带宽。（译注：好象这就是WFQ）

　　实际上，pipe用来实施带宽的限制，而queues则用来决定不同的流量怎么样来分享这些带宽。（译注：这是流量整形中最难理解的部分，本人费了近两天的时间才搞清楚这些关系。）

　　pipe和queue配置可用的命令格式如下：

pipe number config pipe-configuration

queue number config pipe-configuration

pipe专用的参数如下：

bw bandwidth | device

Bandwidth, 单位为： [K|M]{bit/s|Byte/s}.

　　　　如果设置为0，则是没有带宽限制。这些单位必须紧跟在数字后面。（译注：也就是不能有空格之类的东西。）就如下例一样：

ipfw pipe 1 config bw 300Kbit/s

　　　　如果用一个设备名来代替上面的数值，如：

ipfw pipe 1 config bw tun0

　　　　那么，这种设备将提供传输时钟，到目前为止，只有在ppp上跑的tun(4)设备才有这种能力。

delay ms-delay

　　　　传播延迟是以毫秒为单位的。实际上，这个值近似等于一个时钟嘀嗒(典型的是10ms，但是，建议在内核配置中，使选项HZ=1000，这样可以将其减少到1ms或更少)的倍数，默认为0时，表示没有延迟。（译注：clock tick是计算机中，时间的最小单位。本人理解不一定正确：括号的内容说的就是clock tick，而不是指延迟的单位。）

queue专用参数如下：

pipe pipe\_nr

　　　　将一个队列queue连接到一个指定的管道pipe。多个具有不同weight值的队列可以连接到同一个管道，这些队列将共用一个pipe指定的带宽。

weight weight

　指定一个weight值，用在匹配这个队列的流量上。weight的范围是1-100，默认为1。

　　最后，下面的参数，可以用在pipe和queue中：

buckets hash-table-size

　　指定用来存储队列的哈希表的大小。默认是64，范围是16-65536。可以用内核变量net.inet.ip.dummynet.hash\_size来调整这个值。

mask mask-specifier

从ipfw规则送到pipe或queue的数据包可以再进一步划分成多个流量，每个流量都有一个动态的pipe或者queue。这些流量的分类标识，是在pipe或queue的配置中，用IP地址、端口号和协议的掩码组合来区分（译注：是“和”，不是“或”）。对于不同的流量标识，将自动创建一个新的pipe或queue，参数和原始的对象一样，用来匹配后来发到pipe或queue的数据包。

　　　这样，当用动态管道pipe的时候，每个流量将使用和原始的管道相同的流量，然而，对于动态的队列queue来说，每个流量去共同分享原始的queue所相关的那个pipe流量。可以用下列掩码格式的组合：

dst-ip mask, dst-ip6 mask, src-ip mask, src-ip6 mask, dst-port

mask, src-port mask, flow-id mask, proto mask ， all,

　　all的意思指的是所有的位。

noerror

当一个包被dummynet的队列或管道丢弃的时候，将在内核中产生一个错误，来通知调用例程。当一个队列设备满了的时候，也会产生一个这样的错误。如果设置了这个参数，则在成功的传输数据包之后，将报告调用例程，这对于一些专业的，模拟远程路由上的丢失或堵塞的实验，是很有用的。

plr packet-loss-rate

丢包率。参数packet-loss-rate是一个界于0和1之间的浮点数，为0时，表示不丢包，为1时表示全丢。丢包率在内部用第31位表示。（译注：不是很明白，也许是表示默认值为31）

queue {slots | sizeKbytes}

　　　队列的长度，用槽（slot）或size（Kbytes）表示。默认是50槽，这个数值正是以太网典型的队列大小。注意，对于慢速的连接，必须使这个值小，否则将会导致严重的队列延迟。比如：在一个管道为30Kbit/s的队列中，50个以太网最大长度的包，即总长度为600Kbit，将会现20秒的延迟。（译注：翻译这句查了很多资料，结果网上大多数的资料都把人引入一个岐途，让人越看越不明白。实际上这句是警告我们：在限速的时候，如果限制的速度比较小，那么就要注意队列长度别太大。）

　　　如果在某个界面上的数据包长度大于MTU，比如说在一个本地回路上出现16KB的包，那将是更糟糕的。（译注：还没有测试到底能糟糕到什么程度，老外说话，有时候比较幽默。）

red | gred w\_q/min\_th/max\_th/max\_p

　　　启用RED(Random Early Detection)队列算法机制。w\_q和max\_p是界于0和1（不包含1）之间的浮点数，min\_th和max\_th是一个表示队列管理的阈值的整数（如果队列用字节定义，则阈值用字节表示，否则用槽表示）。dummynet(4)也支持gentle RED（gred）。有三个内核变量可以用来控制RED：

net.inet.ip.dummynet.red\_lookup\_depth

　　在连接空闲时，用来指定计算队列的平均命中率。（默认为256，必须大于0）（译注：对RED不懂，只能从字面翻译）

net.inet.ip.dummynet.red\_avg\_pkt\_size

　　　　指定期望有数据包的平均大小。（默认为512，必须大于0）

net.inet.ip.dummynet.red\_max\_pkt\_size

定义期望的数据包的最大值，仅当queue阈值用字节表示的时候才能使用(默认为1500，必须 大于0)

对于IPv6的数据包，dummynet还有许多限制。必须注意的一个，送到某个界面的link-local数据包经过dummynet后，就不可用了，因为在外部的路径中就被丢弃掉了。所在要记住，link-local数据包不会通过dummynet。（译注：link-local数据包是IPv6中定义的一个本地地址，用在自动配置、没有路由的链接中等。）

CHECKLIST（注释）

Here are some important points to consider when designing your rules:

在规划你的规则的时候，要注意几点：

　　1、记住要过滤in和out两个方向上的数据包，大多数的数据包都需要在两个方向上过滤。

　　2、作测试时要小心，最好在控制台前做测试。如果不能在控制台前，一定要做好回溯的脚本，可以参考：/usr/share/examples/ipfw/change\_rules.sh.

　　3、不要忘记了本地回路。（译注：127.0.0.1）

FINE POINTS（注意的几点）

　　1、在多种情况下，分段的报文会被无条件地丢弃。头部小于20字节的TCP包，头部小于8字节的UDP包、头部小于4字节的ICMP包，都将被丢弃。这些包将会被记录为“pullup failed”，因为这些包中是不完整的。

　　2、分段偏移量为1的TCP包将被无条件丢弃。这是一个有效的包，但是只有一个用途，就是去绕过防火墙。如果开启了日志功能的话，将报告被负1（-1）规则丢弃。

　　3、如果你通过一个网络接入到系统，并且要执行kld来挂接ipfw模拟时，可能会出现问题。（译注：因为默认的ipfw规则是deny all。）建议命令如下：

kldload ipfw && \

ipfw add 32000 allow ip from any to any

　　 同样，执行下面的操作也是非常愚蠢的。（译注：因为flush命令要求本地确认，远程无法确认。所以最好用ipfw -q flush，这样就不要求确认了。）

ipfw flush

　　4、如果security level大于3的时候，防火墙规则将不能更改。（关于security level，参见init(8)）

PACKET DIVERSION（数据包的转发）

一个divert(4)栈都有一个绑定的指定端口，任何发送到该端口上数据包都将被转发。如果没有绑定端口，或者是divert模块没有被挂接，或者内核不支持divert栈，那么数据包将被丢弃。

NETWORK ADDRESS TRANSLATION (NAT)（地址转换）

　　　地址转换配置可以用以下命令：（译注：nat命令是一个内核级的地址转换工具，是从ipfw6.3开始新加入到FreeBSD中的，由于是用aliase库写的，所以翻译中，就直接译为：别名。）

nat nat\_number config nat-configuration

　　　在配置中可以用以下参数：

ip ip\_address

定义一个用来做别名的IP地址。

if nic

　　　用指定接口nic上的IP地址作为别名，如果接口上的地址改变，则自动改变相应的IP地址。

log 在该NAT上启用日志记录功能。

deny\_in

　　　禁止所有的外部连接进入。

same\_ports

　　　做地址转换时，尽量维持转换后的端口和本地的端口号一致。（译注：其实做地址转换的时候，同时也在进行端口的转换。）

unreg\_only

　　　如果从本地来的数据包，不是源自私有IP地址空间，则被忽略。（译注：也就是必须来自192.168.0.0-192.168.255.255,172.16.0.0-172.31.255.255,10.0.0.0-10.255.255.255这三段私有IP地址。）

reset

复位地址转换列表。

reverse

　　　使libalias以相反的方式进行地址转换。（译注：在libalias中有介绍，但是没看懂。）

proxy\_only

　　　仅仅允许透明代理规则，不再进行包的地址转换。

　　　如果希望数据包在进行了地址转换后继续进入防火墙，需要设置内核变量net.inet.ip.fw.one\_pass=0。如果想了解更多的关于alias模块的信息，请对照libalias的man页面。在Examples章节里，可以看到更多的关于nat的用法。

REDIRECT AND LSNAT SUPPORT IN IPFW（IPFW支持的重定向和反向地址转换）

　　　重定向和反向地址转换（LSNAT）的语法和后面的natd的语法相近。查看“示例”可以看到如何进行重定向和反向地址转换。

SYSCTL VARIABLES（内核变量）

　　　防火墙和相关的模块（dummynet，bridge）都受一系列内核变量的控制。这儿把它们集合在一起，解释一下默认的值（请用sysctl(8)来查看它实际的值）和它们的意义。

net.inet.ip.dummynet.expire: 1

　　　将没有流量通过的一些过期的动态管道/队列删除。你可以把它设置为0，那样就只能在管道/队列数达到阙值时被删除掉。

net.inet.ip.dummynet.hash\_size: 64

　　　动态管道/队列的默认哈希表的大小。在配置管道/队列时，如果指定了no buckets，则使用这个值。

net.inet.ip.dummynet.max\_chain\_len: 16

　　　在哈希链中，管道/队列的最大数目。在net.inet.ip.dummynet.expire=0时，管道/队列的数目超过阙值时，空的管道/队列将过期。这个阙值就是用max\_chain\_len\*hash\_size来决定。

net.inet.ip.dummynet.red\_lookup\_depth: 256

net.inet.ip.dummynet.red\_avg\_pkt\_size: 512

net.inet.ip.dummynet.red\_max\_pkt\_size: 1500

上面的三个，是在RED算法中，用来计算丢弃率的参数。

net.inet.ip.fw.autoinc\_step: 100

当自动产生规则编号时的步进值。范围是1－1000。（译注：就是ipfw后不指定规则号时的自动编号）

net.inet.ip.fw.curr\_dyn\_buckets: net.inet.ip.fw.dyn\_buckets

当前哈希表中的动态规则条目的数目。（只读）

net.inet.ip.fw.debug: 1

控制ipfw调试信息的输出。

net.inet.ip.fw.dyn\_buckets: 256

哈希表中动态规则的条目数。必须是2的整数次幂，直到65536。它只有当所有的动态规则都失效时才起作用，所以在修改了它的值以后，建议用flush命令来确保改变了哈希表的大小。

net.inet.ip.fw.dyn\_count: 3

Current number of dynamic rules (read-only).

当前动态规则的数目。（只读）

net.inet.ip.fw.dyn\_keepalive: 1

对于状态保持规则的TCP连接，产生一个keepalive的包。对于生存期还剩下20秒的连接，将每隔5秒钟，就对双方都发送一个keepalive包。（译注：可能理解的不对，last的意思不是很明确）

net.inet.ip.fw.dyn\_max: 8192

动态规则的最大数目。当动态规则的数目达到这个值是，就必须等到老的动态规则过期。

net.inet.ip.fw.dyn\_ack\_lifetime: 300

net.inet.ip.fw.dyn\_syn\_lifetime: 20

net.inet.ip.fw.dyn\_fin\_lifetime: 1

net.inet.ip.fw.dyn\_rst\_lifetime: 1

net.inet.ip.fw.dyn\_udp\_lifetime: 5

net.inet.ip.fw.dyn\_short\_lifetime: 30

　　　这些变量控制着动态规则的生存期。以秒为单位。对于初始化的SYN交换，要短一些；到了双方都看见SYN的时候，要加长生存期；到交换FIN和接收RST时，又要减小。dyn\_fin\_lifetime和dyn\_rst\_lifetime这两个等待期必须都小于5秒。这些由防火墙强制执行。（译注：SYN和FIN，RST的意义，请参考TCP的三次握手。）

net.inet.ip.fw.enable: 1

　　　启用防火墙。把它的值变为0，将禁用防火墙，既使把它编译到内核里也是一样。

net.inet6.ip6.fw.enable: 1

对IPv6也同样提供上面的功能。

net.inet.ip.fw.one\_pass: 1

当设置为1的时候，离开dummynet的管道或着ng\_ipfw节点时，不再回到防火墙。否则，数据包将在上面的动作之后，又重新返回到防火墙，继续与下面的规则相匹配。

net.inet.ip.fw.verbose: 1

　　　　启用日志记录功能。

net.inet.ip.fw.verbose\_limit: 0

记录日志的次数。

net.inet6.ip6.fw.deny\_unknown\_exthdrs: 1

拒绝带有未知的IPv6扩展头的数据包。

net.link.ether.ipfw: 0

控制第二层（数据链路层）的数据包能不能通过防火墙。默认不通过。

net.link.bridge.ipfw: 0

控制从桥上通过的数据包能不能通过IPFW。默认不能通过。

EXAMPLES（示例）

IPFW有着广泛的应用范围，这儿只能举很少的一些例子。

BASIC PACKET FILTERING（基本的包过滤）

　　下面这个命令将阻止所有的来自wolf.tambov.su主机，到cracker.evil.org的telnet端口的连接。

ipfw add deny tcp from cracker.evil.org to wolf.tambov.su telnet

下面这句将阻止123.45.67.0/24这个网络的IP对my.host.org主机的访问。

ipfw add deny ip from 123.45.67.0/24 to my.host.org

一个基本的、高效的限制访问的方法（不使用动态规则）如下：

ipfw add allow tcp from any to any established

ipfw add allow tcp from net1 portlist1 to net2 portlist2 setup

ipfw add allow tcp from net3 portlist3 to net3 portlist3 setup

...

ipfw add deny tcp from any to any

　　第一条规则将快速地匹配正常的TCP数据包，但是不能和初始化用的SYN包相匹配。初始化的SYN包只和setup规则相匹配，并且只匹配于选定的源/目的地址对。其它所有的初始化SYN包都被最后一句拒绝。

　　如果你掌管一个或多个子网，那么你就可以利用地址集和"or"运算符写出短小精湛的规则，用来有选择地启动服务或禁掉某个机器。如下所示：

goodguys="{ 10.1.2.0/24{20,35,66,18} or 10.2.3.0/28{6,3,11} }"

badguys="10.1.2.0/24{8,38,60}"

ipfw add allow ip from ${goodguys} to any

ipfw add deny ip from ${badguys} to any

... normal policies ...

verrevpath选项可以用来进行自动的反欺骗，一般的用法，是将带有verrevpath选项的规则放到最前面。

ipfw add deny ip from any to any not verrevpath in

　　这个规则丢弃掉所有的在错误的界面上出现的数据包。例如，一个源自被保护的内部网络地址的数据包，如果尝试通过外部的界面，则被丢弃处理。

　　antispoof选项可以用在相似的情况，但是更为严格。可以把它用在规则的最前头。

ipfw add deny ip from any to any not antispoof in

　　如果数据包看起来源自于一个直连的网络，但是却出现在错误的界面，这样的包则被该规则丢弃。例如，一个包的源IP是192.168.0.0/24，配置为fxp0网卡，来自fxp1的源是192.168.0.0/24的包，将被丢弃掉。

DYNAMIC RULES（动态规则）

为了保护一个站点免受flood的攻击，包括虚假的TCP包攻击，可以使用动态规则：（译注：以前还真不知道有这个功能。）

ipfw add check-state

ipfw add deny tcp from any to any established

ipfw add allow tcp from my-net to any setup keep-state

这个创建的动态规则仅匹配于来自于内部网络的正规的SYN包。动态规则在第一次遇到check-state或keep-state时被检查。一个check-state规则通常放在规则集靠前的地方，来减少扫描规则集的负担。当然，你会有你自己的想法。

为了限制用户的并发访问数量，可以用以下规则：

ipfw add allow tcp from my-net/24 to any setup limit src-addr 10

ipfw add allow tcp from any to me setup limit src-addr 4

　　第一条规则（假设运行在一个网关上）只允许每个在/24网络上的机器同时打开10个TCP连接。第二个规则一般应用于服务器，可以保证一个客户端的并发连接数不超过4个。

　　小心：动态规则很容易导致DOS攻击，因为SYN-flood（syn洪水）可以打开数目众多的动态规则。可以用一系列内核变量的调整来操作防火墙，从而限制这种攻击的效果。（译注：可惜没说到底应该设置哪些内核参数。）

　　用list命令查看记录和时间戳信息是一个很好的主意。

ipfw -at list

　　或者可以用一种没有时间戳的短格式：

ipfw -a list

　　这等同于：

ipfw show

　　下面这个规则，将来自192.168.2.0/24这个网络的数据包都转发到5000端口。（译注：怎么突然来了这么一段？本来讲动态规则的，思维的弯拐得太急了吧，有好几次都以为复制时出错了。）

ipfw divert 5000 ip from 192.168.2.0/24 to any in

TRAFFIC SHAPING（流量整形）

　　下面的规则将演示ipfw和dummynet的一些应用，或者模拟应用。

这条规则以丢包率为5%的机率，随机丢掉进入的包：

ipfw add prob 0.05 deny ip from any to any in

也一个可以利用dummynet模拟这个效果。

ipfw add pipe 10 ip from any to any

ipfw pipe 10 config plr 0.05

　　我们可以利用pipe来限制带宽，例如，在一个作为路由器的机器上，如果我们想用来限制来自本地192.168.2.0/24的客户端的流量：

ipfw add pipe 1 ip from 192.168.2.0/24 to any out

ipfw pipe 1 config bw 300Kbit/s queue 50KBytes

　　注意我们使用out修改符，所以这个规则不会使用两次。要记住，事实上，ipfw规则在流出和进入两个方向上都进行检查。

　　我们也可以用来模拟双向的带宽限制，正确的做法如下：

ipfw add pipe 1 ip from any to any out

ipfw add pipe 2 ip from any to any in

ipfw pipe 1 config bw 64Kbit/s queue 10Kbytes

ipfw pipe 2 config bw 64Kbit/s queue 10Kbytes

　　上面的这个规则是非常有用的。例如，如果你想看到一个在住宅区里，用慢速设备上网的用户在上你心爱的网站，那么你就不要用同一条pipe规则来限制两个方向的流量，除非你想模拟一个半工的连接（例如：AppleTalk，Ethernet，IRDA）。在某些配置里，没有必要使两个管道pipe的配置相同，所以你可以模拟一个非对称的连接。

　　如果想用RED队列管理算法来验证网络的性能：

ipfw add pipe 1 ip from any to any

ipfw pipe 1 config bw 500Kbit/s queue 100 red 0.002/30/80/0.1

　　流量整形的另一个典型应用是：在通信中加入一些延迟。这对于一些大量应用远程进程调用的程序来说，会有比较大的影响，在这些应用中，连接的重返时间限制经常比限制带宽更重要。

ipfw add pipe 1 ip from any to any out

ipfw add pipe 2 ip from any to any in

ipfw pipe 1 config delay 250ms bw 1Mbit/s

ipfw pipe 2 config delay 250ms bw 1Mbit/s

　　对“每个流量”进行限制的队列有很多用途，一个简单的应用是用来计算数据包的个数和：

ipfw add pipe 1 tcp from any to any

ipfw add pipe 1 udp from any to any

ipfw add pipe 1 ip from any to any

ipfw pipe 1 config mask all

　　上面的一组规则将对所有的连接创建一个队列（和收集统计信息）。因为管道没有限制，所以唯一的效果就是对数据做出统计。需要注意，要使用3个规则，而不是仅仅最后一个。这是因为当IPFW匹配IP包的时候，不考虑端口，所以，在第三句里，我们看不到连接中端口的信息。

　　另外广泛的应用是在一个网络中，对于“每个主机”出口的流量进行限制，要比对于“每个网络”的限制好得多。

ipfw add pipe 1 ip from 192.168.2.0/24 to any out

ipfw add pipe 2 ip from any to 192.168.2.0/24 in

ipfw pipe 1 config mask src-ip 0x000000ff bw 200Kbit/s queue 20Kbytes

ipfw pipe 2 config mask dst-ip 0x000000ff bw 200Kbit/s queue 20Kbytes

LOOKUP TABLES（表查询）

　　在下面的示例中，我们需要把流量带宽分成好几类，我们需要把不同的主机/网络对应于不同的类中。我们为每一类创建一个管道pipe，并且做好相应的配置。然后我们创建一个单一的表，并且填上IP子网和地址。对每一个子网/主机，我们设置了和准备使用管道相同的号码。然后，我们就可以用一条规则来分开这些流量：

ipfw pipe 1 config bw 1000Kbyte/s

ipfw pipe 4 config bw 4000Kbyte/s

...

ipfw table 1 add 192.168.2.0/24 1

ipfw table 1 add 192.168.0.0/27 4

ipfw table 1 add 192.168.0.2 1

...

ipfw add pipe tablearg ip from table(1) to any

　　使用fwd运用，表项将包含主机名和IP地址：

ipfw table 1 add 192.168.2.0/24 10.23.2.1

ipfw table 1 add 192.168.0.0/27 router1.dmz

...

ipfw add 100 fwd tablearg ip from any to table(1)

SETS OF RULES（规则集）

　　自动创建一个规则集，比如：set 18

ipfw set disable 18

ipfw add NN set 18 ... # 可以重复

ipfw set enable 18

　　要自动删除一个规则集，可以简单在用以下命令：

ipfw delete set 18

　　要测试一个规则集，先禁用它，然后再重新控制，测试一下有没有错误：

ipfw set disable 18

ipfw add NN set 18 ... # repeat as needed

ipfw set enable 18; echo done; sleep 30 && ipfw set disable 18

　　如果事情进展顺利，可以在“sleep”前按ctrl+C将其中断，则规则集就是处在激活的状态。否则，如果你不能操作你的机器，规则集将在睡眠一段时间后自动禁止，又可以回复到以前的状态。

　　要查看规则集中的规则：

ipfw set 18 show

　　要查看被禁用的规则集中的规则：

ipfw -S set 18 show

　　要将某个集合的计数器清除：

ipfw set 18 zero NN

　　要删除某个规则集中的某个规则：

ipfw set 18 delete NN

NAT, REDIRECT AND LSNAT（地址转换、重定向和反向地址转换）

　　首先，把所有流量重定向到nat 123例程。

ipfw add nat 123 all from any to any

　　然后，配置例程nat 123，以ip地址192.168.0.123作地址转换，还要阻止所有进入的连接、尝试在两端保持相同的端口号、在IP地址改变时清除地址转换列表、并且还要记录通信/链接统计。

ipfw nat 123 config ip 192.168.0.123 log deny\_in reset same\_ports

　　或者改变例程nat 123的地址，地址转换表将被清除（参阅reset选项）

ipfw nat 123 config ip 10.0.0.1

　　要查看nat 123的配置：

ipfw nat 123 show config

　　要查看例程111－999的记录：

ipfw nat 111-999 show

　　要查看所有例程的配置：

ipfw nat show config

　　一个混杂模式的重定向规则，看起来应该如下：

ipfw nat 123 config redirect\_addr 10.0.0.1 10.0.0.66

redirect\_port tcp 192.168.0.1:80 500

redirect\_proto udp 192.168.1.43 192.168.1.1

redirect\_addr 192.168.0.10,192.168.0.11

10.0.0.100 # LSNAT

redirect\_port tcp 192.168.0.1:80,192.168.0.10:22

500 # LSNAT

　　它也可以被分成以下几句规则：

ipfw nat 1 config redirect\_addr 10.0.0.1 10.0.0.66

ipfw nat 2 config redirect\_port tcp 192.168.0.1:80 500

ipfw nat 3 config redirect\_proto udp 192.168.1.43 192.168.1.1

ipfw nat 4 config redirect\_addr

192.168.0.10,192.168.0.11,192.168.0.12

10.0.0.100

ipfw nat 5 config redirect\_port tcp

192.168.0.1:80,192.168.0.10:22,192.168.0.20:25 500

SEE ALSO

cpp(1), m4(1), altq(4), divert(4), dummynet(4), if\_bridge(4), ip(4),

ipfirewall(4), ng\_ipfw(4), protocols(5), services(5), init(8),

kldload(8), reboot(8), sysctl(8), syslogd(8)

HISTORY

ipfw工具第一次出现在FreeBSD2.0中。dummynet(4)从FreeBSD2.2.8开始被引入。状态保持在FreeBSD4.0时被引入。ipfw2出现在2002年夏天。

AUTHORS

Ugen J. S. Antsilevich,

Poul-Henning Kamp,

Alex Nash,

Archie Cobbs,

Luigi Rizzo.

API based upon code written by Daniel Boulet for BSDI.

In-kernel NAT support written by Paolo Pisati < piso@FreeBSD.org该E-mail地址已受到防止垃圾邮件机器人的保护，您必须启用浏览器的Java Script才能看到。 > as part

of a Summer of Code 2005 project.

Work on dummynet(4) traffic shaper supported by Akamba Corp.

BUGS

　　语法经过多年的发展，有时候比较容易混淆。不幸的是，向后兼容性阻碍了在定义语法时就清除错误。

!!! 警告 !!!

　　不正确的配置，会导致你的计算机停止了所有的服务，进入不可使用的状态，使你不得不到机器前面去操作。

　　被转发的分段的包，是在转发界面中，在转发之前重新封装的。用在这些包上的动作，就是匹配了第一个分段包的那个规则的动作。

　　一个包被转发到用户空间后，将被插入到一个用户空间的进程中，这将会失去一些包的属性。如果包的长度小于8字节，并且用户进程保存和重用sockaddr\_in（就像natd(8))，包的源界面名将被保留；否则，将会消失。如果一个包以这种方式被重新插进来，后来的规则会有错误的应用，所以，在制定规则的顺序时，按照转发的顺序进行，是很重要的。

　　Dummynet在遇到IPv6的link-local地址时，会丢掉所有的包。

　　使用uid或gid的规则可能会达不到预期的效果。典型地，进入的SYN包就没有一个相关的uid或gid，因为它们不属于任何的TCP连接。并且，如果相关的进程使用setuid(2)或其它类似的系统调用，数据包相关的uid/gid也会不是预期的值。

　　规则语法受命令行的环境影响，其它的一些匹配模式必须用反斜杠或引号来进行转义。

　　受libalias(3)架构的限制，ipfw 地址转换不兼容tcp的TSO（tcp segmentation offloading，分段卸载）。所以，当在网络中使用nat的时候，请使用ifconfig工具将网卡的TSO禁止。