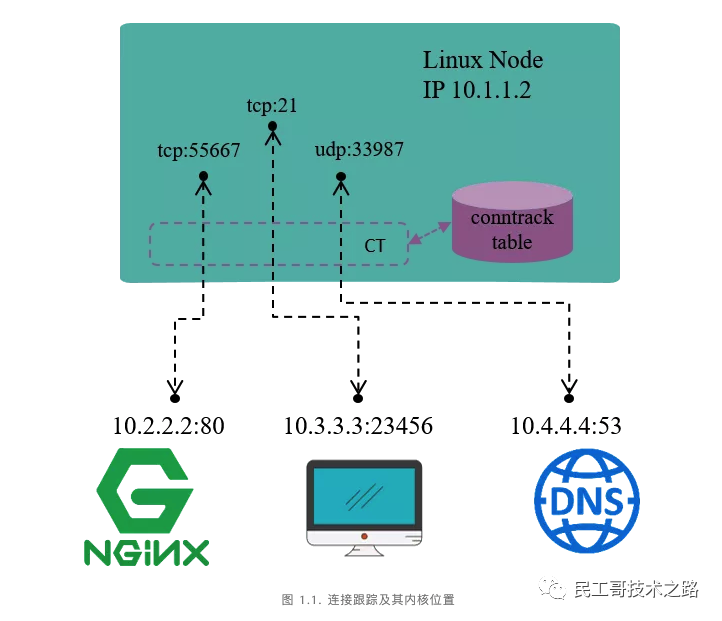
# Linux 防火墙软件 IPtables！

## 连接跟踪（conntrack）

连接跟踪是许多网络应用的基础。例如，Kubernetes Service、ServiceMesh sidecar、 软件四层负载均衡器 LVS/IPVS、Docker network、OVS、iptables 主机防火墙等等，都依赖连接跟踪功能。



连接跟踪，顾名思义，就是跟踪（并记录）连接的状态。例如，图 1.1 是一台 IP 地址为 10.1.1.2 的 Linux 机器，我们能看到这台机器上有三条 连接：

* 机器访问外部 HTTP 服务的连接（目的端口 80）

* 外部访问机器内 FTP 服务的连接（目的端口 21）

* 机器访问外部 DNS 服务的连接（目的端口 53）

连接跟踪所做的事情就是发现并跟踪这些连接的状态，具体包括：

* 从数据包中提取元组（tuple）信息，辨别数据流（flow）和对应的连接（connection）。

* 为所有连接维护一个状态数据库（conntrack table），例如连接的创建时间、发送 包数、发送字节数等等。

* 回收过期的连接（GC）。

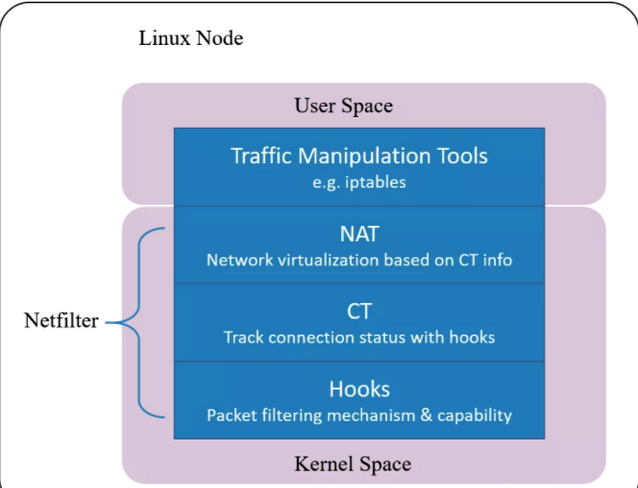
* 为更上层的功能（例如 NAT）提供服务。

需要注意的是，连接跟踪中所说的“连接”，概念和 TCP/IP 协议中“面向连接”（connection oriented）的“连接”并不完全相同，简单来说：

* TCP/IP 协议中，连接是一个四层（Layer 4）的概念。TCP 是有连接的，或称面向连接的（connection oriented），发送出去的包都要求对端应答（ACK），并且有重传机制。UDP 是无连接的，发送的包无需对端应答，也没有重传机制。

* conntrack(CT) 中，一个元组（tuple）定义的一条数据流（flow ）就表示一条连接（connection）。后面会看到 UDP 甚至是 ICMP 这种三层协议在 CT 中也都是有连接记录的，但不是所有协议都会被连接跟踪。

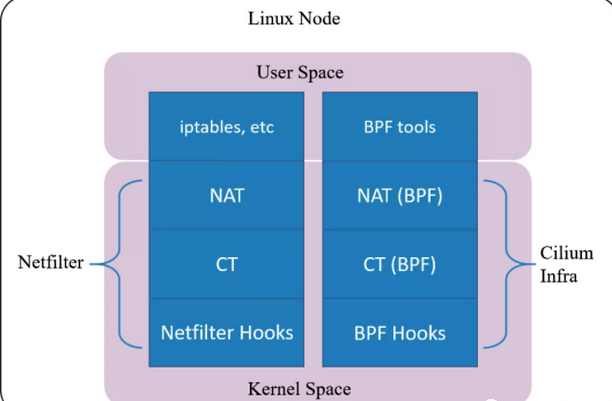
## Netfilter



Linux 的连接跟踪是在 Netfilter 中实现的。

Netfilter 是 Linux 内核中一个对数据 包进行控制、修改和过滤（manipulation and filtering）的框架。它在内核协议栈中设置了若干 hook 点，以此对数据包进行拦截、过滤或其他处理。

现在提到连接跟踪（conntrack），可能首先都会想到 Netfilter，Netfilter 只是 Linux 内核中的一种连接跟踪实现。换句话说，只要具备了 hook 能力，能拦截到进出主机的每个包，完全可以在此基础上自己实现一套连接跟踪。



云原生网络方案 Cilium 在 1.7.4+ 版本就实现了这样一套独立的连接跟踪和 NAT 机制 （完备功能需要 Kernel 4.19+）。其基本原理是：

* 基于 BPF hook 实现数据包的拦截功能（等价于 netfilter 里面的 hook 机制）

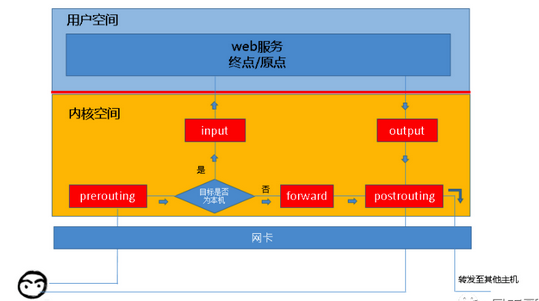
* 在 BPF hook 的基础上，实现一套全新的 conntrack 和 NAT 因此，即便卸载掉 Netfilter ，也不会影响 Cilium 对 Kubernetes ClusterIP、NodePort、ExternalIPs 和 LoadBalancer 等功能的支持。由于这套连接跟踪机制是独立于 Netfilter 的，因此它的 conntrack 和 NAT 信息也没有 存储在内核的（也就是 Netfilter 的）conntrack table 和 NAT table。所以常规的 conntrack/netstats/ss/lsof 等工具是看不到的，要使用 Cilium 的命令，例如：

$ cilium bpf nat list  
$ cilium bpf ct list global

## Iptables

Iptables 是配置 Netfilter 过滤功能的用户空间工具。netfilter 才是防火墙真正的安全框架（framework），netfilter 位于内核空间。iptables 其实是一个命令行工具，位于用户空间，我们用这个工具操作真正的框架。Iptable 根据规则所定义的方法来处理数据包，如放行（accept）、拒绝（reject）和丢弃（drop）等。

例如当客户端访问服务器的web服务时，客户端发送报文到网卡，而 tcp/ip 协议栈是属于内核的一部分，所以，客户端的信息会通过内核的 TCP 协议传输到用户空间中的 web 服务中，而此时，客户端报文的目标终点为 web 服务所监听的套接字（IP:Port）上，当web服务需要响应客户端请求时，web 服务发出的响应报文的目标终点则为客户端，这个时候，web 服务所监听的 IP 与端口反而变成了原点，我们说过，netfilter 才是真正的防火墙，它是内核的一部分，所以，如果我们想要防火墙能够达到”防火”的目的，则需要在内核中设置关卡，所有进出的报文都要通过这些关卡，经过检查后，符合放行条件的才能放行，符合阻拦条件的则需要被阻止。



iptables 包含 4个表，5个链。其中表是按照对数据包的操作区分（过滤， NAT等）的，链是按照不同的 Hook 点来区分的，表和链实际上是netfilter的两个维度。

iptables 的四个表分别是 filter，mangle，nat，raw，默认表是filter。

* filter 表：用来对数据包进行过滤，具体的规则要求决定如何处理一个数据包。

* nat 表：主要用来修改数据包的 IP 地址、端口号信息。

* mangle 表：主要用来修改数据包的服务类型，生存周期，为数据包设置标记，实现流量整形、策略路由等。

* raw 表：主要用来决定是否对数据包进行状态跟踪。

iptables 的五个链分别是 PREROUTING，INPUT，FORWARD，OUTPUT，POSTROUTING。

* input 链：当收到访问本机地址的数据包时，将应用此链中的规则。

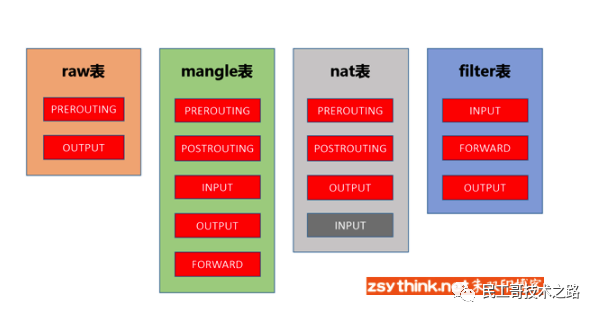
* output 链：当本机向外发送数据包时，将应用此链中的规则。

* forward 链：当收到需要转发给其他地址的数据包时，将应用此链中的规则，注意如果需要实现forward转发需要开启Linux内核中的ip\_forward功能。

* prerouting 链：在对数据包做路由选择之前，将应用此链中的规则。

* postrouting 链：在对数据包做路由选择之后，将应用此链中的规则。

表和链的对应关系如下图所示：



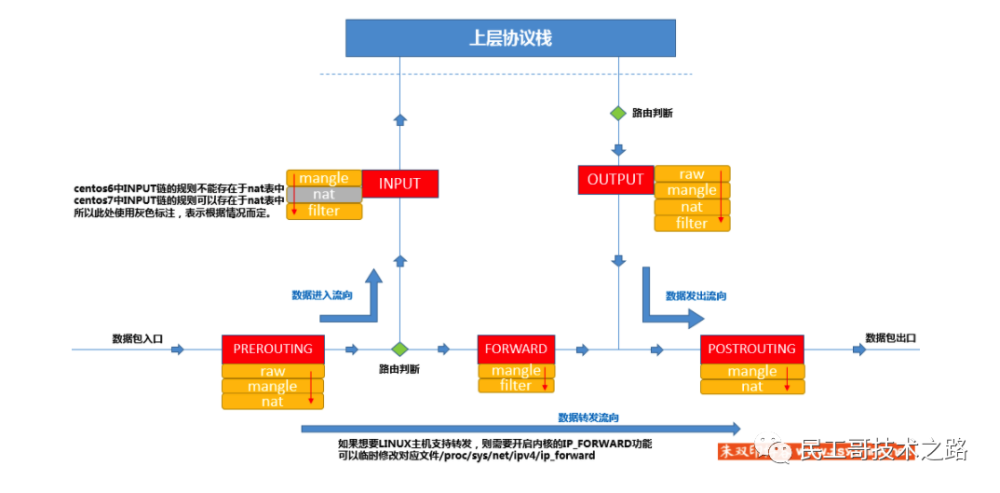
我们能够想象出某些常用场景中，报文的流向：

* 到本机某进程的报文：PREROUTING –> INPUT。

* 由本机转发的报文：PREROUTING –> FORWARD –> POSTROUTING。

* 由本机的某进程发出报文（通常为响应报文）：OUTPUT –> POSTROUTING。

我们可以将数据包通过防火墙的流程总结为下图：



## 查询规则

-t：表名  
-n：不解析IP地址  
-v：会显示出计数器的信息，数据包的数量和大小  
-x：选项表示显示计数器的精确值  
--line-numbers：显示规则的序号（简写为--line）  
-L：链名

#iptables -t filter -nvxL DOCKER --line  
Chain DOCKER (1 references)  
num pkts bytes target prot opt in out source destination  
1 5076 321478 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.2 tcp dpt:8443  
2 37233 54082508 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.2 tcp dpt:22  
3 1712 255195 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.3 tcp dpt:9000  
4 0 0 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.3 tcp dpt:8000  
5 40224 6343104 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.4 tcp dpt:3443  
6 21034 2227009 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.5 tcp dpt:3306  
7 58 5459 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.6 tcp dpt:80  
8 826 70081 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.6 tcp dpt:443  
9 10306905 1063612492 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.9 tcp dpt:3306  
10 159775 12297727 ACCEPT tcp -- !docker0 docker0 0.0.0.0/0 172.17.0.7 tcp dpt:11111

## 增加规则

在指定表的指定链的尾部添加一条规则，-A 选项表示在对应链的末尾添加规则，省略 -t 选项时，表示默认操作 filter 表中的规则：

命令语法：iptables -t 表名 -A 链名 匹配条件 -j 动作  
示例：iptables -t filter -A INPUT -s 192.168.1.146 -j DROP

在指定表的指定链的首部添加一条规则，-I 选型表示在对应链的开头添加规则：

命令语法：iptables -t 表名 -I 链名 匹配条件 -j 动作  
示例：iptables -t filter -I INPUT -s 192.168.1.146 -j ACCEPT

在指定表的指定链的指定位置添加一条规则：

命令语法：iptables -t 表名 -I 链名 规则序号 匹配条件 -j 动作  
示例：iptables -t filter -I INPUT 5 -s 192.168.1.146 -j REJECT

## 删除规则

按照规则序号删除规则，删除指定表的指定链的指定规则，-D 选项表示删除对应链中的规则。示例表示删除filter表中INPUT链中序号为3的规则。：

命令语法：iptables -t 表名 -D 链名 规则序号  
示例：iptables -t filter -D INPUT 3

按照具体的匹配条件与动作删除规则，删除指定表的指定链的指定规则。示例表示删除filter表中INPUT链中源地址为192.168.1.146并且动作为DROP的规则。：

命令语法：iptables -t 表名 -D 链名 匹配条件 -j 动作  
示例：iptables -t filter -D INPUT -s 192.168.1.146 -j DROP

删除指定表的指定链中的所有规则，-F选项表示清空对应链中的规则：

命令语法：iptables -t 表名 -F 链名  
示例：iptables -t filter -F INPUT

## 修改规则

修改指定表中指定链的指定规则，-R 选项表示修改对应链中的规则，使用 -R 选项时要同时指定对应的链以及规则对应的序号，并且规则中原本的匹配条件不可省略。示例表示修改filter表中INPUT链的第3条规则，将这条规则的动作修改为ACCEPT， -s 192.168.1.146为这条规则中原本的匹配条件，如果省略此匹配条件，修改后的规则中的源地址可能会变为0.0.0.0/0：

命令语法：iptables -t 表名 -R 链名 规则序号 规则原本的匹配条件 -j 动作  
示例：iptables -t filter -R INPUT 3 -s 192.168.1.146 -j ACCEPT

设置指定表的指定链的默认策略（默认动作）：

命令语法：iptables -t 表名 -P 链名 动作  
示例：iptables -t filter -P FORWARD ACCEPT

## 保存规则

**方式一**

当我们对规则进行了修改以后，如果想要修改永久生效，必须使用下面命令保存规则：

service iptables save

当然，如果你误操作了规则，但是并没有保存，那么使用 service iptables restart 命令重启 iptables 以后，规则会再次回到上次保存 /etc/sysconfig/iptables 文件时的模样。

**centos7**中，已经不再使用 init 风格的脚本启动服务，而是使用 unit 文件，所以，在 centos7 中已经不能再使用类似 service iptables start 这样的命令了，所以 service iptables save 也无法执行，同时，在 centos7中，使用 firewall 替代了原来的 iptables service，不过不用担心，我们只要通过 yum 源安装 iptables与iptables-services 即可（iptables 一般会被默认安装，但是iptables-services 在 centos7 中一般不会被默认安装），在centos7 中安装完 iptables-services 后，即可像 centos6 中一样，通过 service iptables save 命令保存规则了，规则同样保存在 /etc/sysconfig/iptables 文件中。此处给出 centos7 中配置 iptables-service 的步骤：

#配置好yum源以后安装iptables-service  
yum install -y iptables-services  
#停止firewalld  
systemctl stop firewalld  
#禁止firewalld自动启动  
systemctl disable firewalld  
#启动iptables  
systemctl start iptables  
#将iptables设置为开机自动启动，以后即可通过iptables-service控制iptables服务  
systemctl enable iptables

上述配置过程只需一次，以后即可在 centos7 中使用 service iptables save 命令保存 iptables 规则了。

**方式二**

还可以使用另一种方法保存 iptables 规则，就是使用 iptables-save 命令。使用 iptables-save 并不能保存当前的 iptables 规则，但是可以将当前的 iptables 规则以”保存后的格式”输出到屏幕上。

所以，我们可以使用 iptables-save 命令，再配合重定向，将规则重定向到 /etc/sysconfig/iptables 文件中即可。

iptables-save > /etc/sysconfig/iptables

## 加载规则

我们也可以将 /etc/sysconfig/iptables 中的规则重新载入为当前的iptables 规则，但是注意，未保存入 /etc/sysconfig/iptables 文件中的修改将会丢失或者被覆盖。

使用 iptables-restore 命令可以从指定文件中重载规则，示例如下

iptables-restore < /etc/sysconfig/iptables

## 匹配条件

当规则中同时存在多个匹配条件时，多个条件之间默认存在”与”的关系，即报文必须同时满足所有条件，才能被规则匹配。

-s 用于匹配报文的源地址,可以同时指定多个源地址，每个IP之间用逗号隔开，也可以指定为一个网段。

#示例如下  
iptables -t filter -I INPUT -s 192.168.1.111,192.168.1.118 -j DROP  
iptables -t filter -I INPUT -s 192.168.1.0/24 -j ACCEPT  
iptables -t filter -I INPUT ! -s 192.168.1.0/24 -j ACCEPT

-d 用于匹配报文的目标地址,可以同时指定多个目标地址，每个 IP 之间用逗号隔开，也可以指定为一个网段。

#示例如下  
iptables -t filter -I OUTPUT -d 192.168.1.111,192.168.1.118 -j DROP  
iptables -t filter -I INPUT -d 192.168.1.0/24 -j ACCEPT  
iptables -t filter -I INPUT ! -d 192.168.1.0/24 -j ACCEPT

-p 用于匹配报文的协议类型,可以匹配的协议类型 tcp、udp、udplite、icmp、esp、ah、sctp 等（centos7 中还支持 icmpv6、mh）。

#示例如下  
iptables -t filter -I INPUT -p tcp -s 192.168.1.146 -j ACCEPT  
iptables -t filter -I INPUT ! -p udp -s 192.168.1.146 -j ACCEPT

-i 用于匹配报文是从哪个网卡接口流入本机的，由于匹配条件只是用于匹配报文流入的网卡，所以在 OUTPUT 链与 POSTROUTING 链中不能使用此选项。

#示例如下  
iptables -t filter -I INPUT -p icmp -i eth4 -j DROP  
iptables -t filter -I INPUT -p icmp ! -i eth4 -j DROP

-o 用于匹配报文将要从哪个网卡接口流出本机，于匹配条件只是用于匹配报文流出的网卡，所以在 INPUT 链与 PREROUTING 链中不能使用此选项。

#示例如下  
iptables -t filter -I OUTPUT -p icmp -o eth4 -j DROP  
iptables -t filter -I OUTPUT -p icmp ! -o eth4 -j DROP

## 扩展匹配条件

**tcp扩展模块**

常用的扩展匹配条件如下：

–sport：用于匹配 tcp 协议报文的源端口，可以使用冒号指定一个连续的端口范围。  
–dport：用于匹配 tcp 协议报文的目标端口，可以使用冒号指定一个连续的端口范围。  
–tcp-flags：用于匹配报文的tcp头的标志位。  
–syn：用于匹配 tcp 新建连接的请求报文，相当于使用 –tcp-flags SYN,RST,ACK,FIN SYN。

注意，-p tcp与 -m tcp 并不冲突，-p 用于匹配报文的协议，-m 用于指定扩展模块的名称，正好，这个扩展模块也叫 tcp。

#示例如下  
iptables -t filter -I OUTPUT -d 192.168.1.146 -p tcp -m tcp --sport 22 -j REJECT  
iptables -t filter -I INPUT -s 192.168.1.146 -p tcp -m tcp --dport 22:25 -j REJECT  
iptables -t filter -I INPUT -s 192.168.1.146 -p tcp -m tcp --dport :22 -j REJECT  
iptables -t filter -I INPUT -s 192.168.1.146 -p tcp -m tcp --dport 80: -j REJECT  
iptables -t filter -I OUTPUT -d 192.168.1.146 -p tcp -m tcp ! --sport 22 -j ACCEPT  
iptables -t filter -I INPUT -p tcp -m tcp --dport 22 --tcp-flags SYN,ACK,FIN,RST,URG,PSH SYN -j REJECT  
iptables -t filter -I OUTPUT -p tcp -m tcp --sport 22 --tcp-flags SYN,ACK,FIN,RST,URG,PSH SYN,ACK -j REJECT  
iptables -t filter -I INPUT -p tcp -m tcp --dport 22 --tcp-flags ALL SYN -j REJECT  
iptables -t filter -I OUTPUT -p tcp -m tcp --sport 22 --tcp-flags ALL SYN,ACK -j REJECT  
iptables -t filter -I INPUT -p tcp -m tcp --dport 22 --syn -j REJECT

**udp 扩展模块**

常用的扩展匹配条件：

* –sport：匹配udp报文的源地址。

* –dport：匹配udp报文的目标地址。

#示例  
iptables -t filter -I INPUT -p udp -m udp --dport 137 -j ACCEPT  
iptables -t filter -I INPUT -p udp -m udp --dport 137:157 -j ACCEPT

**icmp 扩展模块**

常用的扩展匹配条件：

* –icmp-type：匹配icmp报文的具体类型。

#示例  
iptables -t filter -I INPUT -p icmp -m icmp --icmp-type 8/0 -j REJECT  
iptables -t filter -I INPUT -p icmp --icmp-type 8 -j REJECT  
iptables -t filter -I OUTPUT -p icmp -m icmp --icmp-type 0/0 -j REJECT  
iptables -t filter -I OUTPUT -p icmp --icmp-type 0 -j REJECT  
iptables -t filter -I INPUT -p icmp --icmp-type "echo-request" -j REJECT

**multiport 扩展模块**

常用的扩展匹配条件如下：

* -p tcp -m multiport –sports 用于匹配报文的源端口，可以指定离散的多个端口号,端口之间用”逗号”隔开。

* -p udp -m multiport –dports 用于匹配报文的目标端口，可以指定离散的多个端口号，端口之间用”逗号”隔开。

#示例如下  
iptables -t filter -I OUTPUT -d 192.168.1.146 -p udp -m multiport --sports 137,138 -j REJECT  
iptables -t filter -I INPUT -s 192.168.1.146 -p tcp -m multiport --dports 22,80 -j REJECT  
iptables -t filter -I INPUT -s 192.168.1.146 -p tcp -m multiport ! --dports 22,80 -j REJECT  
iptables -t filter -I INPUT -s 192.168.1.146 -p tcp -m multiport --dports 80:88 -j REJECT  
iptables -t filter -I INPUT -s 192.168.1.146 -p tcp -m multiport --dports 22,80:88 -j REJECT

**iprange 模块**

包含的扩展匹配条件如下：

* –src-range：指定连续的源地址范围。

* –dst-range：指定连续的目标地址范围。

#示例  
iptables -t filter -I INPUT -m iprange --src-range 192.168.1.127-192.168.1.146 -j DROP  
iptables -t filter -I OUTPUT -m iprange --dst-range 192.168.1.127-192.168.1.146 -j DROP  
iptables -t filter -I INPUT -m iprange ! --src-range 192.168.1.127-192.168.1.146 -j DROP

**string 模块**

常用扩展匹配条件如下：

* –algo：指定对应的匹配算法，可用算法为bm、kmp，此选项为必需选项。

* –string：指定需要匹配的字符串

我们想要达到的目的是，如果报文中包含”OOXX”字符，我们就拒绝报文进入本机：

#示例  
iptables -t filter -I INPUT -m string --algo bm --string "OOXX" -j REJECT

**time 模块**

常用扩展匹配条件如下：

* –timestart：用于指定时间范围的开始时间，不可取反。

* –timestop：用于指定时间范围的结束时间，不可取反。

* –weekdays：用于指定”星期几”，可取反。

* –monthdays：用于指定”几号”，可取反。

* –datestart：用于指定日期范围的开始日期，不可取反。

* –datestop：用于指定日期范围的结束时间，不可取反。

#示例  
iptables -t filter -I OUTPUT -p tcp --dport 80 -m time --timestart 09:00:00 --timestop 19:00:00 -j REJECT  
iptables -t filter -I OUTPUT -p tcp --dport 443 -m time --timestart 09:00:00 --timestop 19:00:00 -j REJECT  
iptables -t filter -I OUTPUT -p tcp --dport 80 -m time --weekdays 6,7 -j REJECT  
iptables -t filter -I OUTPUT -p tcp --dport 80 -m time --monthdays 22,23 -j REJECT  
iptables -t filter -I OUTPUT -p tcp --dport 80 -m time ! --monthdays 22,23 -j REJECT  
iptables -t filter -I OUTPUT -p tcp --dport 80 -m time --timestart 09:00:00 --timestop 18:00:00 --weekdays 6,7 -j REJECT  
iptables -t filter -I OUTPUT -p tcp --dport 80 -m time --weekdays 5 --monthdays 22,23,24,25,26,27,28 -j REJECT  
iptables -t filter -I OUTPUT -p tcp --dport 80 -m time --datestart 2017-12-24 --datestop 2017-12-27 -j REJECT

**connlimit 模块**

常用的扩展匹配条件如下：

* –connlimit-above：单独使用此选项时，表示限制每个IP的链接数量。

* –connlimit-mask：此选项不能单独使用，在使用–connlimit-above选项时，配合此选项，则可以针对”某类IP段内的一定数量的IP”进行连接数量的限制，如果不明白可以参考上文的详细解释。

#示例  
iptables -I INPUT -p tcp --dport 22 -m connlimit --connlimit-above 2 -j REJECT  
iptables -I INPUT -p tcp --dport 22 -m connlimit --connlimit-above 20 --connlimit-mask 24 -j REJECT  
iptables -I INPUT -p tcp --dport 22 -m connlimit --connlimit-above 10 --connlimit-mask 27 -j REJECT

**limit 模块**

connlimit 模块是对连接数量进行限制的，limit 模块是对”报文到达速率”进行限制的。用大白话说就是，如果我想要限制单位时间内流入的包的数量，就能用 limit 模块。我们可以以秒为单位进行限制，也可以以分钟、小时、天作为单位进行限制。常用的扩展匹配条件如下：

* –limit-burst：类比”令牌桶”算法，此选项用于指定令牌桶中令牌的最大数量。

* –limit：类比”令牌桶”算法，此选项用于指定令牌桶中生成新令牌的频率，可用时间单位有second、minute 、hour、day。

示例表示限制外部主机对本机进行ping操作时，本机最多每6秒中放行一个ping包

#示例，注意，如下两条规则需配合使用  
#令牌桶中最多能存放3个令牌，每分钟生成10个令牌（即6秒钟生成一个令牌）。  
iptables -t filter -I INPUT -p icmp -m limit --limit-burst 3 --limit 10/minute -j ACCEPT  
#默认将icmp包丢弃  
iptables -t filter -A INPUT -p icmp -j REJECT

**state 扩展模块**

当我们通过 http 的 url 访问某个网站的网页时，客户端向服务端的 80 端口发起请求，服务端再通过 80 端口响应我们的请求，于是，作为客户端，我们似乎应该理所应当的放行 80 端口，以便服务端回应我们的报文可以进入客户端主机，于是，我们在客户端放行了 80 端口，同理，当我们通过 ssh 工具远程连接到某台服务器时，客户端向服务端的 22 号端口发起请求，服务端再通过 22 号端口响应我们的请求，于是我们理所应当的放行了所有 22 号端口，以便远程主机的响应请求能够通过防火墙，但是，作为客户端，如果我们并没有主动向 80 端口发起请求，也没有主动向 22 号端口发起请求，那么其他主机通过 80 端口或者 22 号端口向我们发送数据时，我们可以接收到吗？应该是可以的，因为我们为了收到 http 与 ssh 的响应报文，已经放行了 80 端口与 22 号端口，所以，不管是”响应”我们的报文，还是”主动发送”给我们的报文，应该都是可以通过这两个端口的，那么仔细想想，这样是不是不太安全呢？此时 state 扩展模块就派上用场了。

对于 state 模块的连接而言，”连接”其中的报文可以分为**5种状态**，分别为：

* NEW：连接中的第一个包，状态就是 NEW，我们可以理解为新连接的第一个包的状态为 NEW。

* ESTABLISHED：我们可以把 NEW 状态包后面的包的状态理解为 ESTABLISHED，表示连接已建立。

* RELATED：从字面上理解 RELATED 译为关系，但是这样仍然不容易理解，我们举个例子。比如 FTP 服务，FTP 服务端会建立两个进程，一个命令进程，一个数据进程。命令进程负责服务端与客户端之间的命令传输（我们可以把这个传输过程理解成 state 中所谓的一个”连接”，暂称为”命令连接”）。数据进程负责服务端与客户端之间的数据传输 (我们把这个过程暂称为”数据连接”)。但是具体传输哪些数据，是由命令去控制的，所以，”数据连接”中的报文与”命令连接”是有”关系”的。那么，”数据连接”中的报文可能就是 RELATED 状态，因为这些报文与”命令连接”中的报文有关系。(注：如果想要对ftp进行连接追踪，需要单独加载对应的内核模块 nf\_conntrack\_ftp，如果想要自动加载，可以配置 /etc/sysconfig/iptables-config 文件)

* INVALID：如果一个包没有办法被识别，或者这个包没有任何状态，那么这个包的状态就是 INVALID，我们可以主动屏蔽状态为 INVALID 的报文。

* UNTRACKED：报文的状态为 untracked 时，表示报文未被追踪，当报文的状态为 Untracked 时通常表示无法找到相关的连接。

刚才举例中的问题即可使用 state 扩展模块解决，我们只要放行状态为 ESTABLISHED 的报文即可，因为如果报文的状态为 ESTABLISHED，那么报文肯定是之前发出的报文的回应，这样，就表示只有回应我们的报文能够通过防火墙，如果是别人主动发送过来的新的报文，则无法通过防火墙：

iptables -t filter -I INPUT -m state --state ESTABLISHED -j ACCEPT

## mangle 表

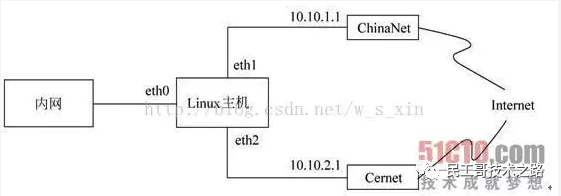
mangle 表的主要功能是根据规则修改数据包的一些标志位，以便其他规则或程序可以利用这种标志对数据包进行过滤或策略路由。mangle 表主要有以下 3 种操作：

* TOS：用来设置或改变数据包的服务类型域。这常用来设置网络上的数据包如何被路由等策略。注意这个操作并不完善，有时得不所愿。它在Internet 上还不能使用，而且很多路由器不会注意到这个域值。换句话说，不要设置发往 Internet 的包，除非你打算依靠 TOS 来路由，比如用 iproute2。

* TTL：用来改变数据包的生存时间域，我们可以让所有数据包只有一个特殊的 TTL。它的存在有 一个很好的理由，那就是我们可以欺骗一些ISP。为什么要欺骗他们呢？因为他们不愿意让我们共享一个连接。那些 ISP 会查找一台单独的计算机是否使用不同的 TTL，并且以此作为判断连接是否被共享的标志。

* MARK 用来给包设置特殊的标记。iproute 2能识别这些标记，并根据不同的标记（或没有标记） 决定不同的路由。用这些标记我们可以做带宽限制和基于请求的分类。

例如内网的客户端通过 Linux 主机连入 Internet，而 Linux 主机与Internet 连接时有两条线路，它们的网关如图所示。现要求对内网进行策略路由，所有通过 TCP 协议访问 80 端口的数据包都从 ChinaNet 线路出去，而所有访问 UDP 协议 53 号端口的数据包都从 Cernet 线路出去。



这是一个策略路由的问题，为了达到目的，在对数据包进行路由前，要先根据数据包的协议和目的端口给数据包做上一种标志，然后再指定相应规则，根据数据包的标志进行策略路由。为了给特定的数据包做上标志，需要使用mangle 表，mangle 表共有 5 条链，由于需要在路由选择前做标志，因此应该使用 PREROUTING 链，下面是具体的命令：

iptables -t mangle -A PREROUTING -i eth0 -p tcp --dport 80 -j MARK --set-mark 1;  
iptables -t mangle -A PREROUTING -i eth0 -p udp --dprot 53 -j MARK --set-mark 2;

数据包经过 PREROUTING 链后，将要进入路由选择模块，为了对其进行策略路由，执行以下两条命令，添加相应的规则：

ip rule add from all fwmark 1 table 10  
ip rule add from all fwmark 2 table 20

以上两条命令表示所有标志是1的数据包使用路由表 10 进行路由，而所有标志是 2 的数据包使用路由表 20 进行路由。路由表 10 和 20 分别使用了 ChinaNet 和 Cernet 线路上的网关作为默认网关，具体设置命令如下所示：

ip route add default via 10.10.1.1 dev eth1 table 10  
ip route add default via 10.10.2.1 dev eth2 table 20

以上两条命令在路由表 10 和 20 上分别指定了 10.10.1.1 和 10.10.2.1 作为默认网关，它们分别位于 ChinaNet 和 Cernet 线路上。于是，使用路由表 10 的数据包将通过 ChinaNet 线路出去，而使用路由表20的数据包将通过 Cernet 线路出去。

## 自定义链

当默认链中的规则非常多时，不方便我们管理。想象一下，如果 INPUT 链中存放了 200 条规则，这 200 条规则有针对 httpd 服务的，有针对 sshd 服务的，有针对私网 IP 的，有针对公网 IP 的，假如，我们突然想要修改针对 httpd 服务的相关规则，难道我们还要从头看一遍这 200 条规则，找出哪些规则是针对 httpd 的吗？这显然不合理。

所以，iptables 中，可以自定义链，通过自定义链即可解决上述问题。假设，我们自定义一条链，链名叫 IN\_WEB，我们可以将所有针对 80 端口的入站规则都写入到这条自定义链中，当以后想要修改针对 web 服务的入站规则时，就直接修改 IN\_WEB 链中的规则就好了，即使默认链中有再多的规则，我们也不会害怕了，因为我们知道，所有针对 80 端口的入站规则都存放在IN\_WEB链中。

**创建自定义链**

#在filter表中创建IN\_WEB自定义链  
iptables -t filter -N IN\_WEB

**引用自定义链**

#在INPUT链中引用刚才创建的自定义链  
iptables -t filter -I INPUT -p tcp --dport 80 -j IN\_WEB

**重命名自定义链**

#将IN\_WEB自定义链重命名为WEB  
iptables -E IN\_WEB WEB

**删除自定义链**

删除自定义链需要满足两个条件：

* 1、自定义链没有被引用。

* 2、自定义链中没有任何规则。

#第一步：清除自定义链中的规则  
iptables -t filter -F WEB  
#第二步：删除自定义链  
iptables -t filter -X WEB

## LOG 动作

LOG 动作默认会将报文的相关信息记录在/var/log/message文件中，当然，我们也可以将相关信息记录在指定的文件中，以防止 iptables 的相关信息与其他日志信息相混淆，修改 /etc/rsyslog.conf 文件（或者 /etc/syslog.conf），在 rsyslog 配置文件中添加如下配置即可：

kern.warning /var/log/iptables.log

完成上述配置后，重启rsyslog服务（或者syslogd）：

service rsyslog restart

LOG 动作也有自己的选项，常用选项如下：

* –log-level 选项可以指定记录日志的日志级别，可用级别有 emerg，alert，crit，error，warning，notice，info，debug。

* –log-prefix 选项可以给记录到的相关信息添加”标签”之类的信息，以便区分各种记录到的报文信息，方便在分析时进行过滤。–log-prefix 对应的值不能超过 29 个字符。

比如，我想要将主动连接22号端口的报文的相关信息都记录到日志中，并且把这类记录命名为”want-in-from-port-22″,则可以使用如下命令：

iptables -I INPUT -p tcp --dport 22 -m state --state NEW -j LOG --log-prefix "want-in-from-port-22"

完成上述配置后，我在IP地址为 192.168.1.98 的客户端机上，尝试使用 ssh 工具连接上例中的主机，然后查看对应的日志文件（已经将日志文件设置为 /var/log/iptables.log）：

如上图所示，ssh 连接操作的报文的相关信息已经被记录到了 iptables.log 日志文件中，而且这条日志中包含”标签”：want-in-from-port-22，如果有很多日志记录，我们就能通过这个”标签”进行筛选了，这样方便我们查看日志，同时，从上述记录中还能够得知报文的源IP与目标IP，源端口与目标端口等信息，从上述日志我们能够看出，192.168.1.98 这个 IP 想要在 14点11分 连接到 192.168.1.139（当前主机的 IP）的22号端口，报文由eth4网卡进入，eth4 网卡的 MAC 地址为 00:0c:29:b7:f4:d1，客户端网卡的 MAC 地址为 f4:8e:38:82:b1:29。

## 参考文档

*转自：Se7en的架构笔记*

*原文：https://url.hi-linux.com/8A1AB*

<https://mp.weixin.qq.com/s/H-CN1WfksjXxmLcDoqgGiw>