

Netfilter Tutorial

Introduction

One of the best things, I thought, about the 2.4.x kernel was the netfilter suite. Sadly, there isn't a lot of documentation out there about how to use it. What is there is excellent, but a lot of it is written to be backward compatible with ipchains. I've thought many times that I'd like to see a netfilter only tutorial. So here goes.

I don't vouch for grand quality, but the scripts presented here will work, and hopefully they will be readable to some one some where. This is still a work in progress, and I gladly accept suggestions on it.

First off, the iptables man page is terribly helpful. If you have time, read it. If you don't have time, read it anyway.

Tables, Chains, and Rules

The tables are filter, nat, and mangle. The names of the filter and nat tables pretty much explain what they do. The mangle table is used for 'specialized packet alteration', and is really beyond the scope of this tutorial. I've never had a reason to use it, and for the rest of this document will ignore it.

The purpose of a table is to contain chains. Inside the filter table are the INPUT, FORWARD, and OUTPUT chains. Inside the nat table are the PREROUTING, POSTROUTING, and OUTPUT chains. The user can also define more chains, and place them in whatever table they want.

Chains contain rules. Rules can be terminating, or non-terminating. A packet enters evaluation based on its type (incoming, outgoing, or forwarded), then traverses tables and rules until it encounters a terminating rule. The order of traversal is as follows:

type	processing
incoming	nat/PREROUTING -> filter/INPUT
forwarded	nat/PREROUTING -> filter/FORWARD -> nat/OUTPUT -> nat/POSTROUTING
outgoing	filter/OUTPUT -> nat/PREROUTING -> nat/OUTPUT -> nat/POSTROUTING

Adding Rules to a Chain

The command for adding a rule to a chain follows this form:

```
iptables -t table -A chain specifiers -j target
```

This will add the new rule at the end of the specified chain

Specifiers

Specifiers include (but are not limited to)

Specifier	Meaning
-s ADDR	Packet came from source ADDR
-d ADDR	Packet is going to ADDR
-i IFACE	Packet came from interface IFACE

Netfilter Tutorial

Specifier	Meaning
-o IFACE	Packet will leave through IFACE
-m COND	Packet matches the condition COND.
-p PROTO	Packet uses the protocol PROTO

Addresses can be fully qualified domain names ("www.crrhalpin.org"), IP addresses ("10.0.0.1"), CIDR style network specifications ("10.0.0.0/24"), or the logical not of any of those things (ie "!www.crrhalpin.org").

Interfaces are eth0, ppp0, tun0, and so on. Protocols are tcp, udp, and icmp.

Conditions can be a complicated beast, and I won't talk much about them. For more information, read the 'man iptables'. I only use two conditions '-m state --state STATES' and '-m limit'.

Possible states for '-m state' are NEW, RELATED, ESTABLISHED, and INVALID. The names describe the meaning pretty well.

The '-m limit --limit TIME' lets you limit how frequently a given rule is matched. This is most useful when used with the LOG target, to keep your system logs from filling up. This pops up in the example script at the end.

Targets

Targets include (but are not limited to)

Target	Meaning
ACCEPT	Accept this packet. Terminating Valid in the filter table
DROP	Ignore this packet. The sender gets no notification. Terminating Valid in the filter table
REJECT	Reject this packet, and send an icmp message back to the sender to indicate that this packet died. Terminating Valid in the filter table
SNAT --to-source ADDR	Change the source address of this packet to ADDR. Non-terminating Valid in the nat table
DNAT --to-destination ADDR	Change the destination address of this packet to ADDR. Non-terminating Valid in the nat table.
LOG	Log this packet to syslog. Non-terminating Valid in all tables

Netfilter Tutorial

Target	Meaning
<i>CHAIN</i>	Punt processing of this packet to <i>CHAIN</i> Terminates if <i>CHAIN</i> contains a terminating rule that matches this packet Valid in all tables

Chain Manipulations

command	meaning
<code>iptables -t table -N chain</code>	Create a new chain called <i>chain</i> inside <i>table</i>
<code>iptables -t table -F chain</code>	Remove all rules from <i>chain</i> in <i>table</i>
<code>iptables -t table -X chain</code>	Delete <i>chain</i> from <i>table</i>

Built-in chains (INPUT, FORWARD, OUTPUT) can also have a 'policy', which will be invoked if no other rule on the chain matches. These policies can be ACCEPT, REJECT, or DROP. Policies are set with 'iptables -t filter -P CHAIN policy'.

Tips and Style Issues

Things to keep in mind when using netfilter (mostly style issues):

- **Always make the table you use explicit**
Though this isn't required, specifying it will make it such that your scripts are readable to everyone. You won't ever have to worry about getting a rule or a chain in the wrong table.
- **The state matchers are your friend**
It's easy to forget that a lot of outgoing connections will have related incoming connections, or that outgoing forwards will have corresponding incoming forwards. While it's possible to write individual allow rules to catch these, matching based on state "-m state --state RELATED,ESTABLISHED" is much more concise.
- **Write using addresses, not input/output specs**
The old ipchains stuff specified everything in terms of what interface it came into, and what interface it went out of. iptables does still support this, but it's easy (at least for me) to get confused in this way. iptables supports specifying everything in terms of source and destination addresses, which is somewhat more portable, and much easier to read.

If you have suggestions, please contact me.

This example is for a firewall and NAT box

```
#!/bin/bash
IFACE=#the public interface of your box.
PUBLIC=ifconfig $IFACE | grep "inet addr" | cut -d ":" -f 2 | cut -d " " -f 1`
#That shell-insanity will extract the public IP of $IFACE for you.
#This can be useful if you're on a PPP connection, and dont always know
#What the IP of $IFACE will be.
LOCALNET= #The IP ranges of your local net. Example: 10.0.0.0/24
/sbin/modprobe -k ip_conntrack_ftp
#Clear all chains, and set default policies.
# I consider it good practice to set your policies to DROP
```

Netfilter Tutorial

```
# whenever you are changing your firewall configuration
iptables -t filter -P INPUT DROP; iptables -t filter -F INPUT
iptables -t filter -P FORWARD DROP; iptables -t filter -F FORWARD
iptables -t nat -F POSTROUTING

#Set up a chain for bad packets.
# Packets sent here will be logged, no more than five per hour from the
# same source, and then dropped.
iptables -F log; iptables -X log; iptables -N log
iptables -A log -m limit --limit 5/hour -j LOG
iptables -A log -j DROP

# NAT
iptables -t nat -A POSTROUTING -s $LOCALNET -d \! $LOCALNET -j SNAT --to-source
$PUBLIC
#Packets to accept on the public interface
#Catch and log 'INVALID' packets
iptables -t filter -A INPUT -d $PUBLIC -m state --state INVALID -j log
#Accept packets that are from and to the loopback interface.
iptables -t filter -A INPUT -s localhost -d localhost -j ACCEPT
#Accept packets relating to already-established connections
iptables -t filter -A INPUT -d $PUBLIC -m state --state RELATED,ESTABLISHED -j
ACCEPT
#Accept ssh connections
iptables -t filter -A INPUT -p tcp -d $PUBLIC --dport 22 -j ACCEPT
#Add entries for other services that you want to accept here.

#Anything else gets dropped and logged.
iptables -t filter -A INPUT -d $PUBLIC -j log

#Packets to forward from private outward.
# Catch and log any INVALID data.
iptables -t filter -A FORWARD -m state --state INVALID -j log
#Accept packets related to pre-existing connections
iptables -t filter -A FORWARD -d $LOCALNET -m state --state RELATED,ESTABLISHED -j
ACCEPT
#Allow the private network to connect to anything
iptables -t filter -A FORWARD -s $LOCALNET -j ACCEPT
#Drop and log anything else
iptables -t filter -A FORWARD -j log
```

Common Mistakes (or Don't Follow in my Footsteps)

More will be added to this as I remember all of the various idiotic things I've done in the past.

Overly General Rules

Consider the following situation: You have a firewall (say it's 10.0.0.1). Behind it is both your private internal network, and your webserver. You want to redirect traffic on port 80 to the webserver (say it's 10.0.0.10). You write the following:

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination
10.0.0.10:80
```

This will redirect traffic from outside to your webserver. However, it will also redirect traffic from inside to your webserver as well. God help you if your firewall is your webserver, and you want to go to a website. A more appropriate rule would be:

Netfilter Tutorial

```
iptables -t nat -A PREROUTING -p tcp -s ! 10.0.0.0/24 --dport 80 -j DNAT --to-destination 10.0.0.10:80
```

Similar kinds of things can happen if you have a web proxy that you want to redirect traffic from inside thorough. Remember to let traffic from the proxy out.