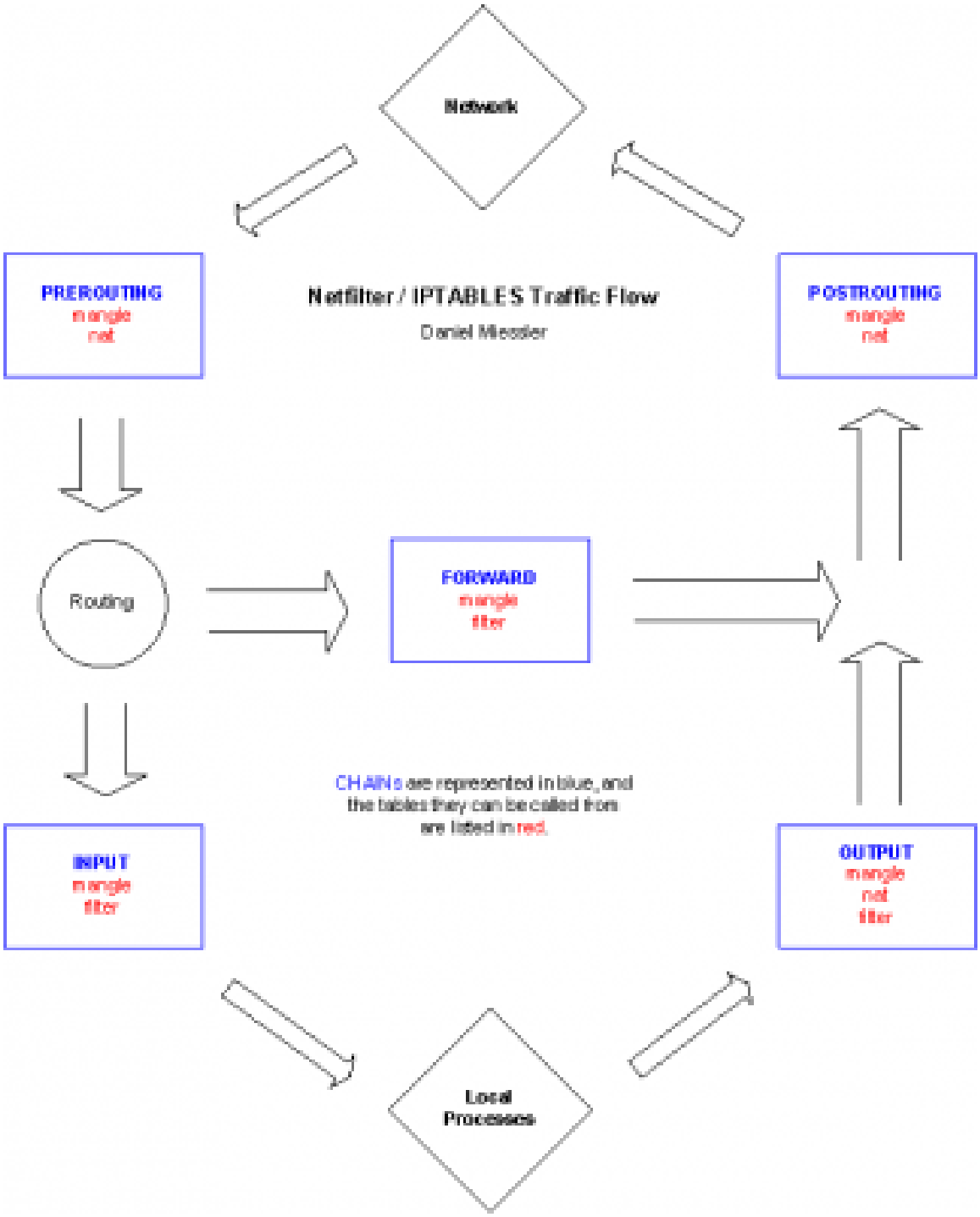
An IPTABLES Primer



`iptables` is the packet filtering technology that's built into the 2.4 Linux kernel. It's what allows one to do firewalling, nating, and other cool stuff to packets from within Linux. Actually, that's not quite right — `iptables` is just the command used to control *netfilter*, which is the real underlying technology. We'll just call it iptables though, since that's how the whole system is usually referred to.

STATEFUL PACKET INSPECTION

First off, many have heard a number of different definitions of "stateful" firewalls and/or "SPI" protection, and I think it's worth the time to take a stab at clearing up the ambiguity. "Stateful Inspection" actually gives its true definition away in its name; *it's nothing more and nothing less than attempting to ensure that traffic moving through the firewall is legitimate by determining whether or not it's part of (or related to) an existing, accepted connection.*

When you hear that this firewall or that firewall does "SPI", it could really mean anything; it's a big buzzword right now, so every company out there wants to add it to their sales pitch. Remember, the definition is broad, so there can be (and is) a big difference between so-called SPI protection on a $50 SOHO router as compared to what's offered on something like Check Point FW-1. The former could do a couple TCP-flag checks and call it SPI, while the latter does a full battery of tests. Just keep that in mind; *not all SPI is created equal*.

NETFILTER BASICS

`iptables` is made up of some basic structures, as seen below:

- TABLES

- CHAINS
- TARGETS

TABLES

TABLES are the major pieces of the packet processing system, and they consist of FILTER, NAT, and MANGLE. FILTER is used for the standard processing of packets, and it's the default table if none other is specified. NAT is used to rewrite the source and/or destination of packets and/or track connections. MANGLE is used to otherwise modify packets, i.e. modifying various portions of a TCP header, etc.

CHAINS

CHAINS are then associated with each table. Chains are lists of rules within a table, and they are associated with "hook points" on the system, i.e. places where you can intercept traffic and take action. Here are the default table/chain combinations:

- FILTER: Input, Output, Forward
- NAT: Prerouting, Postrouting, Output
- MANGLE: Prerouting, Postrouting, Input, Output, Forward

And here's when the different chains do their thing:

- **PREROUTING:** Immediately after being received by an interface.
- **POSTROUTING:** Right before *leaving* an interface.
- **INPUT:** Right before being handed to a local process.

- **OUTPUT:** Right after being *created* by a local process.
- **FORWARD:** For any packets coming in one interface and leaving out another.

In other words, if you want to process packets as they leave your system, but without doing any NAT or MANGLE(ing), you'll look to the OUTPUT chain within the FILTER table. If you want to process packets coming from the outside destined for your local machine, you'll want to use the same FILTER table, but the INPUT chain. See the image below for a visual representation of this.

TARGETS

TARGETS determine what will happen to a packet within a chain if a match is found with one of its rules. A two most common ones are DROP and ACCEPT. So if you want to drop a packet on the floor, you write a rule that matches the particular traffic and then jump to the DROP target. Conversely, if you want to allow something, you jump to the ACCEPT target — simple enough.

HOW PACKETS MOVE

Packets move through netfilter by traversing chains. Each non-empty chain has a list of rules in it, which packets are checked against one after another. If a match is found, the packet is processed. If no match is found, the default action is taken. The default action for a chain is also called its *policy*. By default, chain policies are to jump to the ACCEPT target, but this can be set to DROP if so desired (I suggest it).

DIGGING IN

So, with that ~~inadequate~~ short intro out of the way, let's dig into it with some diagrams and a couple of cookbook-style examples :

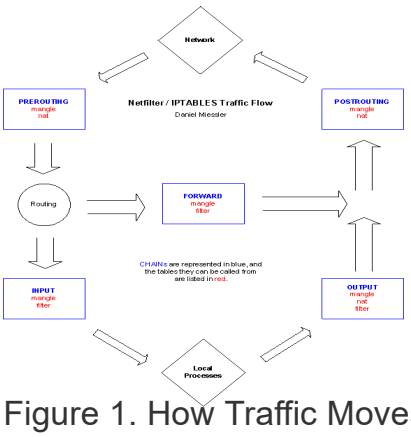


Figure 1. How Traffic Moves Through Netfilter

Allow Outgoing (Stateful) Web Browsing

iptables -A OUTPUT -o eth0 -p TCP –dport 80 -j ACCEPT iptables -A INPUT -i eth0 -p TCP -m state –state ESTABLISHED,RELATED –sport 80 -j ACCEPT

In the first rule, we're simply adding (appending) a rule to the OUTPUT chain for protocol TCP and destination port 80 to be allowed. We are also specifying that the incoming packets will need to exit the machine over interface eth0 (-o is for "output") in order to trigger this rule; this interface designation is important when you start dealing with machines with multiple interfaces. You can also add additional checks beyond those seen here, such as what source ports are allowed, etc., but we'll keep it simple for the examples here. The second rule allows the web traffic to come back (an important part of browsing).

Notice the "state" stuff; that's what makes netfilter a "stateful" firewalling technology. Packets are not able to move through this rule and get back to the client unless they were created via the rule above it, i.e. they have to be part of an established or related connection, *and* be coming from a source port of 80 — which is usually a web server. Again, you can add more checks here, but you get the point.

Allowing Outgoing Pings

iptables -A OUTPUT -o eth0 -p icmp –icmp-type echo-request -j ACCEPT iptables -A INPUT -i eth0 -p icmp –icmp-type echo-reply -j ACCEPT

Here, we're *appending* (-A) to the *output* (OUTPUT) chain, using the *icmp* (-p) protocol, of type *echo-request* (–icmp-type echo request), and *jumping* (-j) to the ACCEPT target (which means ACCEPT it, strangely enough). That's for the outgoing piece. For the return packet, we append to the INPUT chain instead of OUTPUT, and allow *echo-reply* instead of echo-request. This, of course, means that incoming echo-requests to your box will be dropped, as will outgoing replies.

"PASSING PORTS" INTO A NATD NETWORK

One of the most commonly-used functions of firewall devices is "passing ports" inside to other private, hidden machines on your network running services such as web and mail. In corporate environments this is out of necessity, and at home it's often for gaming or in a hobbyist context. Either way, here's how you do it with Netfilter/IPTABLES:

iptables -t nat -A PREROUTING -i eth0 -p tcp -d 1.2.3.4 –dport 25 -j DNAT –to 192.168.0.2:25

If we break this down, we see that we're actually using the **nat** table here rather than not specifying one. Remember, if nothing is mentioned as far as tables are concerned, you're using the filter table by default. So in this case we're using the nat table and appending a rule to the PREROUTING chain. If you recall from the diagram above, the PREROUTING chain takes effect right after being received by an interface from the outside.

This is where DNAT occurs. This means that destination translation happens *before* routing, which is good to know. So, we then see that the rule will apply to the TCP protocol for all packets destined for port 25 on the **public IP**. From there, we jump to the DNAT target (Destination NAT), and "jump to" (–to) our internal IP on port 25. Notice that the syntax of the internal destination is IP:PORT.

Ah, but this is only half of the work. If you have any experience with corporate-class firewalls such as Check Point or Astaro, you know there are two parts to enabling connectivity like this — the *NAT* portion, and the **rules** portion. Below is what we need to get the traffic through the firewall:

```
iptables -A FORWARD -i eth0 -o eth1 -p tcp –dport 25 -d 192.168.0.2 -j ACCEPT
```

In other words, if you just NAT the traffic, it's not ever going to make it through your firewall; you have to pass it through the rulebase as well. Notice that we're accepting the packets *in* from the first interface, and allowing them *out* the second. Finally, we're specifying that only traffic to destination port (–dport) 25 (TCP) is allowed — which matches our NAT rule.

The key here is that you need two things in order to pass traffic inside to your hidden servers — NAT and rules.

CONCLUSION

Ok, so that about does it for now. I have obviously only scratched the surface here, but hopefully I've covered the very basics in a way that can help someone. I intend to keep adding to this as time goes on, both for the sake of being thorough and to clarify things as needed. If, however, I have missed something major, or made some sort of error, do feel free to contact me and set me straight. Also, be sure to check out the manpage as well as the links below.

REFERENCES

The Netfilter/Iptables Project
http://www.netfilter.org

Linux Firewalls, Second Edition
http://www.amazon.com/exec/obidos/ASIN/0735710996/103-1806976-4827810